

Does Beautiful Code Matter? We think, So...

by

Nicholas Vaidyanathan

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved September 2018 by the
Graduate Supervisory Committee:

Dr James Collofello, Chair
Dr. Robert Atkinson
Dr. Hessam Sarjoughan
Dr. Hasan Davulcu

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

In this dissertation, I explored the state of the literature of software engineering and found that while multiple schools of thought exist, most practitioners argue there is no consensus on a theory to guide software engineering [1, 2, 3]. Agile practitioners have developed a set of guiding principles and tools commonly advocated in industry such as the SOLID design principles for writing code, but these remain experiential rather than experimentally verified.

Cognitive Science developed a theoretical framework for guiding the development of curricular material called Cognitive Load Theory (CLT)[4, 5]. CLT derives from the earliest discoveries of neuroscience [6], its central precepts involve the design of instructional content and arrangement in a way that is cognitively available. CLT seeks to understand the human memory model, specifically the limits of short-term “working memory” [7], and optimize content for memorability. CLT has been effectively applied in a variety of instructional materials and verified via lab experiments since the in 1970s.

Software is essentially a concretion of a programmer’s understanding of the world to create emergent behavior. The core activity of programming is the organization and arrangement of information as realized in data structures and algorithms. Software has explored the complexity of this practice through a variety of metrics, some of which include McCabe’s Complexity Metrics [8], Halstead’s Software Science [9], Albrecht’s Function Points [10], and Wang’s Cognitive Complexity Metrics [11, 12]. I found no examples of known metrics that leveraged CLT in their development.

My work explores a conceptual link between CLT and software engineering best practices. I provide a partial mapping of refactoring techniques and SOLID principles to CLT principles. This link moves software engineering towards a theoretical framework based on human cognition. This dualistic tie can help both fields. Instruc-

tional Designers can organize webs of content according to distributed systems design principles, while software engineers can leverage principles backed by a theoretical framework, experimental approach, and known principles of human cognition.

I designed an experiment that explored the effects of applying these principles on an established software library. I measured the perceived cognitive load, time to debug/mean-time to resolution, and defects introduced via broken tests using a 2x2 Factorial Design with experienced and novice software engineers. With a sample size of $n=188$, I measured that the average mean time to resolution and number of bugs reported by both experienced and novice programmers. I found that the mean time to resolution and introduced defect rate is less for both experienced and novice developers when debugging the refactored code, aligning with concepts from Cognitive Load Theory. I also find that those programmers reported less perceived cognitive load.

Agile design principles combined with the precepts of Cognitive Load Theory can produce software that is measurably easier to debug and understand. Augmenting known refactoring patterns with additional heuristics—such as managing the size of classes and methods to around Miller’s Magic Number and naming concepts according to their usage— produces novel software architectural principles as a consequence of this work. This has implications on Cognitive Load as a measurement for and conceptual backbone of technical debt. Future work should probe advanced ways of measuring cognitive load and programmer experience and the effect of programming language and domain.

This research provides a significant contribution by applying concepts and experimental design from CLT to software engineering. The study measurably shows that code refactored according to specific principles designed to manage cognitive load results in software that is better understood and easier to debug. Looking at

code comprehensibility for programmers through the lens of CLT may lead to new techniques of quantifying readability and analyzing technical debt.

*To my Poobah, who has never stopped believing in me and always inspired me. I
love you.*

TABLE OF CONTENTS

	Page
LIST OF TABLES	ix
LIST OF FIGURES.....	x
CHAPTER	
1 INTRODUCTION	1
2 LITERATURE REVIEW	4
2.1 Establishing the link between Cognitive Load Theory and Software Engineering through Software Craftsmanship	9
2.1.1 Open-Closed Principle.....	10
2.1.2 Newspaper Metaphor.....	10
2.1.3 Design Patterns.....	11
2.1.4 Intrinsic, Germane, and Extraneous Cognitive Load	11
2.1.5 Split Attention Effect	12
2.1.6 Expertise Reversal Effect	12
2.1.7 Applied Example of the types of Cognitive Load in software.	13
2.2 Theory & Predictions	19
2.3 Research Question: Can Cognitive Load Theory provide empirical evidence and a conceptual framework for the efficacy of Refactoring?	21
2.3.1 Why will this be meaningful?	21
2.3.2 What could go wrong?	22
2.3.3 How can one measure the Cognitive Load of code?	22
3 OVERVIEW OF PRESENT EXPERIMENT	24
3.1 Participants	24
3.2 Materials.....	25
3.2.1 Software Chosen - Joda Time	25

CHAPTER	Page
3.2.2	Why Joda Time? 25
3.2.3	What experimental intervention did I make to Joda-time? .. 26
3.2.4	What change did the participants have to make? 27
3.2.5	Bug: ISO8601 Years..... 27
3.2.6	Analysis of Accepted Solution..... 30
3.3	Experimental Environment 31
3.3.1	Aside: The IDE effect 31
3.3.2	Cognitive Load measurement - Likert Scale 33
3.3.3	Minimal Cognitive Load of expressions axioms 34
3.3.4	Likert Scale: line-by-line 35
3.3.5	Minimal Cognitive Load of lines axioms 35
3.3.6	Likert Scale: blocks/scopes 36
3.3.7	Minimal cognitive load of blocks/scopes axioms 36
3.3.8	Open Question: How do we measure code flow cognitive complexity? 36
4	METHOD 40
4.1	Materials Design..... 40
4.2	Intervention, Measures, and Procedure 41
5	RESULTS 42
5.1	Mean Response Time for participants who fixed the bug 42
5.2	Mean Regressions for those who did not fix the bug 44
5.3	Perceived Cognitive Load..... 45
5.4	Discusion 46
6	LIMITATIONS 48

CHAPTER	Page
7 IMPLICATIONS	49
8 FUTURE DIRECTIONS.....	50
REFERENCES	52
APPENDIX	
A LIKERT SCALE FOR CONTROL STUDY	59
B LIKERT SCALE FOR EXPERIMENTAL STUDY.....	61
C MEASURING THE COGNITIVE LOAD OF CODE:	64
C.1 Review.....	64
C.2 Bug: ISO8601 Years	64
C.3 Analysis of Accepted Solution	65
C.4 Experiment: contrast debugging time and performance of accepted solution versus CLT optimized	65
C.5 Development of Experimental Version	66
C.5.1 Starting small: sequencing, chunking, and intrinsic com- plexity at the variable and method level	66
C.5.2 Building up: moving on to the class level	68
C.6 Reducing Control Flow Complexity	75
C.6.1 Checking my biases: adapting to peer feedback.....	89
C.6.2 Architectural adaptation: Simplifying DateTimeFormatter ..	89
C.6.3 Signaling architecture by organization – using packages as chunks	146
C.6.4 Making Derived Components Expressive, the value of names	152
C.6.5 The Final Form – calculate in the NumberFormatter, Nu- mericSequence	155

CHAPTER	Page
C.7 The Experimental Solution	159
C.7.1 Side-by-Side comparison	159

LIST OF TABLES

Table	Page
5.1 Mean Response Time of fixing bug	42
5.2 Test of Between Subjects Effects	43
5.3 Mean Response Time of fixing bug	44
5.4 Test of Between Subjects Effects	44

LIST OF FIGURES

Figure		Page
2.1	insert caption	13
2.2	insert caption	14
2.3	insert caption	15
2.4	insert caption	16
2.5	insert caption	17
2.6	insert caption	17
2.7	insert caption	17
3.1	insert caption	28
3.2	Shankar's Fix	29
3.3	insert caption	35
3.4	Date Time Formatter	38
5.1	Estimated Marginal Means of Time	43
5.2	Estimated Marginal Means of Regressions	45
C.1	insert caption	66
C.2	insert caption	68
C.3	insert caption	69
C.4	insert caption	70
C.5	insert caption	71
C.6	insert caption	71
C.7	insert caption	72
C.8	insert caption	72
C.9	insert caption	73
C.10	insert caption	74
C.11	insert caption	74

Figure		Page
C.12	insert caption	75
C.13	insert caption	76
C.14	insert caption	76
C.15	insert caption	77
C.16	insert caption	78
C.17	insert caption	79
C.18	insert caption	79
C.19	insert caption	80
C.20	insert caption	80
C.21	insert caption	81
C.22	insert caption	82
C.23	insert caption	82
C.24	insert caption	83
C.25	insert caption	83
C.26	insert caption	84
C.27	insert caption	84
C.28	insert caption	85
C.29	insert caption	85
C.30	insert caption	86
C.31	insert caption	86
C.32	insert caption	87
C.33	insert caption	87
C.34	insert caption	88
C.35	insert caption	88

Figure		Page
C.36	insert caption	89
C.37	insert caption	90
C.38	insert caption	91
C.39	insert caption	91
C.40	insert caption	92
C.41	insert caption	92
C.42	insert caption	93
C.43	insert caption	94
C.44	insert caption	95
C.45	insert caption	96
C.46	insert caption	96
C.47	insert caption	97
C.48	insert caption	97
C.49	insert caption	98
C.50	insert caption	98
C.51	insert caption	99
C.52	insert caption	99
C.53	insert caption	100
C.54	insert caption	101
C.55	insert caption	102
C.56	insert caption	103
C.57	insert caption	104
C.58	insert caption	105
C.59	insert caption	106

Figure		Page
C.60	insert caption	107
C.61	insert caption	108
C.62	insert caption	109
C.63	insert caption	110
C.64	insert caption	111
C.65	insert caption	112
C.66	insert caption	113
C.67	insert caption	114
C.68	insert caption	115
C.69	insert caption	116
C.70	insert caption	117
C.71	insert caption	118
C.72	insert caption	119
C.73	insert caption	120
C.74	insert caption	121
C.75	insert caption	121
C.76	insert caption	122
C.77	insert caption	123
C.78	insert caption	123
C.79	insert caption	124
C.80	insert caption	124
C.81	insert caption	125
C.82	insert caption	125
C.83	insert caption	126

Figure		Page
C.84	insert caption	126
C.85	insert caption	127
C.86	insert caption	128
C.87	insert caption	129
C.88	insert caption	129
C.89	insert caption	130
C.90	insert caption	130
C.91	insert caption	131
C.92	insert caption	131
C.93	insert caption	132
C.94	insert caption	132
C.95	insert caption	133
C.96	insert caption	134
C.97	insert caption	134
C.98	insert caption	135
C.99	insert caption	135
C.100	insert caption	136
C.101	insert caption	136
C.102	insert caption	137
C.103	insert caption	138
C.104	insert caption	139
C.105	insert caption	140
C.106	insert caption	140
C.107	insert caption	141

Figure	Page
C.108 insert caption	142
C.109 insert caption	143
C.110 insert caption	144
C.111 insert caption	145
C.112 insert caption	145
C.113 insert caption	146
C.114 insert caption	147
C.115 insert caption	147
C.116 insert caption	148
C.117 insert caption	148
C.118 insert caption	149
C.119 insert caption	149
C.120 insert caption	150
C.121 insert caption	150
C.122 insert caption	151
C.123 insert caption	151
C.124 insert caption	152
C.125 insert caption	153
C.126 insert caption	153
C.127 insert caption	154
C.128 insert caption	154
C.129 insert caption	155
C.130 insert caption	155
C.131 insert caption	156

Figure	Page
C.132 insert caption	156
C.133 insert caption	157
C.134 insert caption	157
C.135 insert caption	158
C.136 insert caption	158
C.137 insert caption	159
C.138 insert caption	159
C.139 insert caption	160
C.140 insert caption	160
C.141 insert caption	161
C.142 insert caption	161
C.143 insert caption	162
C.144 insert caption	162

Chapter 1

INTRODUCTION

Software engineering is one of the youngest and most dynamic fields in engineering. Naur and Randell helped formally define and introduce Software engineering as an area of study in 1968 with their landmark NATO software engineering report. [13] The most striking finding in the report was the startling conclusion that 68% of all software projects fail. Since then, researchers have investigated many different approaches to extract software from the “tar pit” [14]. Explorations have spanned a variety of both technical and non-technical solutions, spawning research in areas as disparate as software cost and estimation, software architecture and design, and software project and process management. Nevertheless, despite the dedicated effort of researchers and practitioners, in over 40 years since the landmark initial paper, research has made minimal ingress against the problem. A follow-up report by the Standish group in the 1990s suggested that overruns are still close to 189% [15], though the numbers are in dispute [16, 17]. The 2015 CHAOS report shows that 52% of projects are challenged, 19% failed [15].

One possible root cause of this churn arises from the inherent difficulty of the base activity of software engineering: computer programming. Leading software engineering practitioners argue that there are better ways to engineer software [18, 19], the Agile software philosophy. Tools such as Test-Driven Development [20, 21], Design Patterns [22, 23], and Object-Oriented Programming [24, 25, 26, 27]. Unfortunately, there is a dearth of empirical evidence supporting these practices [28] [1]. Moreover, software engineering has yet to develop a strong, unifying theory that can serve as a conceptual framework for software engineering study [2] [3].

In parallel, cognitive science developed from the frontiers of psychology into a broad, impactful field that enhanced understanding of the human brain. Developing quickly from George Armitage Miller’s insight about the limits of human working memory [6], cognitive science blossomed into a myriad of disciplines exploring different aspects of human cognition. One of these fields emerged in the 1980s when John Sweller studied problem solving and developed Cognitive Load Theory (CLT) . Based on the work of Miller, Baddeley’s working memory model [7], and collected evidence about short-term “working” memory, Cognitive Load Theory studies effects the arrangement and presentation of learning content has on learner memory and performance. From years of development across a variety of domains, CLT has shown evidence for particular phenomena such as the Split-Attention Effect [29] and the Redundancy Effect [5]. Instructional Designers adopted the findings of CLT into their materials and began to achieve measurable, repeatable results that validated these findings and produced more efficient learning material.

These two fields are evolving independently, in parallel. But should they? Fred Brooks describes software as complex to build because “The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures...” [30] Additionally, Kent Beck describes the purpose of programming as ”What is it that we want to get out of code? The most important thing is learning. The way I learn is to have a thought, then test it out to see if it is a good thought. Code is the best way I know of to do this.” [31] Elements of memory, organization of information, and learning appear prodigiously through software literature, but ties to CLT are scant to nonexistent.

The question of whether CLT could inform the theory and practice of software

engineering and its fundamental activity of computer programming drove this research. In this dissertation, I showed that it can. I suggested that a sampling of the best practices of object oriented analysis and design corresponded to conceptual best practices in Cognitive Load Theory. Using this premise, I designed an experiment similar to traditional CLT experiments that measured the efficiency of programmers to debug and fix software as a function of time to resolve an issue, defects introduced, and perceived cognitive load. I found that amongst both novice and experienced programmers, more people were capable of fixing a bug in refactored code, and that it took less time and introduced less average defects. I found that the subject's internal measurement of Cognitive Load in the refactored code was less than the original code. This suggests that CLT can make a promising contribution into the scientific understanding of the complexity of software systems, providing a basis to evaluate software designs and understand technical debt.

Chapter 2

LITERATURE REVIEW

In the introduction, I gave a high level overview of software engineering and cognitive load theory and the origins of the fields. In this section, I start by reviewing the literature of software engineering and computer science as it has related to specific attempts to influence the practice of programming. Software engineering research covers a space as disparate as processes surrounding the elicitation of software requirements, visualization techniques for software designs, estimation practices for predicting delivery cycles, probabilistic testing, and more. I viewed CLT specifically from the frame of how it can influence the practice and understanding of programming. I progress chronologically, looking at specific programming recommendations of the 1970s and 1980s and moving towards the 2000s and 2010. I also review the way computer science has attempted to quantify the concept of complexity in computer programs through the development of metrics. I find that although CLT is neither directly invoked nor indirectly referenced, much of the modern practitioner literature of Agile Software Development and Software Craftsmanship [32] extensively focus on the comprehensibility of computer programs. The practices and principles proposed in these works have conceptual similarity to cognitive load theory. By the end of this section, the reader should have a sense of established research that is similar to mine in software engineering and a high-level overview of cognitive load theory.

Others have attempted to improve programming practice in the past. Edsger Dijkstra published the provocative *Go To Statement Considered Harmful* [33] in a letter to the editor of Communications of the ACM in 1968. This paper, premised with the assertion that the “the quality of programmers is a decreasing function of

the density of go to statements in the programs they produce” essentially helped launch the conversation that “the way you write code matters” by arguing that one particular tool in high level programming languages, the goto statement, made programs harder to understand. Dijkstra continued refining these ideas and published a set of notes which led to a book on Structured Programming [34]. Structured Programming prescribed the use of control structures— such as if/else statements, cases, and looping statements—instead of only variable assignment statements and jumps to labeled sections. Structured Programming influenced the development of new programming languages like Kernighan and Ritchie’s C Programming Language [35], which incorporated control structures as the fundamental primitives of writing code. Structured programming triggered the elaboration of programming paradigms as diverse as imperative programming, procedural programming, and object-oriented programming. Imperative programming simply arranges program statements in order of execution, very similar to writing a task list. Procedural programming organizes imperative code into subroutines, named groups of imperative statements that can form a basis for abstraction and apply Parnas principle of information hiding [36]. Object-oriented programming defines systems in terms of message passing between independent units that encapsulate data and the operations on that data called objects [37]. In a parallel track, Donald Knuth developed Literate Programming [38, 39]. Programmers who write in a “literate” style combine prose and narrative to elaborate the function of the application. The author of a literate program writes with readability and comprehensibility in mind as a direct consequence of the formulations in accompanying text. Literate Programming has yet to penetrate the majority of the programming zeitgeist, remaining a fairly niche research interest. Nevertheless, many authors have experimented with it and reported enhanced clarity [40]. There are many other paradigms of programming, as have been extensively collected and

classified by Van Roy [67]. All of these paradigms seek to apply a specific set of constraints and models of interaction to computer programs to make it easier for a programmer to reason about and implement their design. Some researchers have looked at the potential cognitive elements of programming paradigms [41, ?], others have looked at cognitive elements independently of programming paradigm [42], but the discipline lacks a de facto standard for the evaluation of the cognitive impact of programming paradigm.

Rather than measure complexity through the impact on human cognition directly, Software metrics comprise effort by the software engineering research community to quantify the results of the software process. The original, most cited software metric is the Line of Code (LOC) and the analogous KLOC (thousands of lines of code). An oft-maligned metric— that does not account for the internal complexity or structure of a program— LOC has nonetheless remained common due to simplicity of measurement. Attempts to provide more information about the inner working of a program have led to the creation of metrics such as Halstead’s Software Science [9], McCabe’s Cyclomatic Complexity [8], and Albrecht’s Function Points [10]. These approaches have found limited success. Empirical studies have produced very little conclusive evidence as to the efficacy of these metrics [43], with many equivocal examples [44]. Modern metrics research, however, is attempting to incorporate elements of cognitive science into the art of understanding software. Wang’s study of the cognitive complexity of software serves as a bridge for the metrics work into this world [11] [12]. Wang develops a set of meta-cognitive models for the description and understanding of computer programs, then constructs a set of experiments that compare and contrast the cognitive complexity of different constructs. The argument is that the metrics generated through such analysis are more accurate descriptors of software complexity than symbolic, structural, or functional complexity metrics.

Cognitive metrics for software scratches the surface of what cognitive science can offer computer science and software engineering. Software maintainers spend more time reading code than they do writing new code. Many developers who write new code often find themselves working with unfamiliar technologies or in an unfamiliar domain. People commonly believe there are similarities between programming and education, but no unifying link in theory has tied successful programming outcomes to established educational theory. One such relevant theory is Cognitive Load Theory [45], which informs the way educational designers seek to develop curricular materials. Human Computer Interface (HCI) designers leverage it to provide better user experiences [46, 47]. Nevertheless, programmers have not applied this work towards actually changing the way they write code for other programmers. Software engineering research has been focused on other areas.

Software Engineering literature of the early 1990s continued the work of the 1980s in software processes and modeling, leading to the development of the Unified Modeling Language (UML) [48] and the associated Rational Unified Process (RUP) [49]. In 1994, the Gang of Four published their seminal Design Patterns [22] catalog, a hybrid design/architecture and implementation publication that inspired a rush of patterns oriented research activity in the community, including implementation patterns [50], testing patterns [20], enterprise application architecture patterns [51], language implementation patterns [52], and others. In February 2001, a group of experienced and well-respected software engineers published the Agile Manifesto [19]. A counterpoint to RUP, agile software development emphasized less documentation, tools, and repeatable processes. Instead, it championed greater emphasis on relationship building and collaboration. This approach quickly became popular with industry, especially with the rise of web application development. Its emphasis on results effectively captured the spirit of the rise of the Internet and fast-paced nature of web development.

Consequently, an explosion of interest for effective ways of organizing and managing agile software teams led to the development of models such as Extreme Programming [31, 53], and Kanban [54].

Many of the forerunners of Agile were leading evolution in multiple areas of software engineering. Andrew Hunt and Dave Thomas wrote *The Pragmatic Programmer* [55], fleshing out the baseline principles of the Agile Manifesto into a programming philosophy built on pragmatism. Kent Beck helped elaborate Extreme Programming [31], codified best practice Implementation Patterns [50], and wrote the most popular Java variant of the xUnit testing framework, JUnit, as part of his work in proselytizing Test-Driven Development [20]. Martin Fowler developed a widely regarded Refactoring patterns catalog [56] extending from Bill Opdike’s PhD thesis [57] to provide a structured, behavior-preserving methodology for modifying existing code to promote clarity and reuse. Robert C. Martin helped expound some basic principles that have become the bedrock of SOLID object-oriented software development [58], including the Single Responsibility Principle, the Interface Segregation Principle, and Dependency Inversion alongside Barbara Liskov’s Substitution Principle [59] and Bertrand Meyer’s Open/Closed Principle [60].

These industrial contributions have ironically transformed much of the software engineering landscape—particularly in web and mobile software development, the preeminent paradigms of the 21st century—at arguably a broader scale than academic software engineering research. Recognition of this dichotomy between the research emphasis and industrial literature has fueled the rise of the Software Craftsmanship movement [32]. Software Craftsmanship espouses treating programming as a craft more than a science or an engineering discipline. In such epistemology, dedicated artisans construct the best software. Such artisans hone their craft through discipline, practice, and apprenticeship/journeyman style interactions with masters.

One could see Software Craftsmanship as a reaction against and rejection of Software Engineering. It eschews the emphasis on process, models, and up-front work to build software systems. Alternately, one could argue that it represents a competing school of thought. Much like Cubism and Expressionism were competing schools of thought in 20th century painting, software craftsmanship and SWEBOK [61] [56] based approaches may be seen as competing schools of thought within 21st century software engineering. One can relate this to the difference shown in DeRemer and Kron's "Programming-in-the large versus programming-in-the-small" [62]. Many software craftsmen seek to apply cognitive science to improve their craft. Andy Hunt wrote a book to help programmers "refactor their wetware" by applying concepts from cognitive science [63]. They have yet to link cognitive science to writing code. CLT is the link.

2.1 Establishing the link between Cognitive Load Theory and Software Engineering through Software Craftsmanship

Before I dive into the conceptual overlap between software craftsmanship best practices and CLT, it will be helpful to define the concepts individually. I begin by outlining concepts from software craftsmanship, then defining concepts from CLT, then describing their connection.

The Single Responsibility Principle states that "a class should have only one reason to change." [58] For example, the design of a vending machine class may include methods to add money, vend a product, set the prices of individual products, restock individual products, and refund money. If the currency of the machine has to change, the implementation may need to be modified. If the product stocking needs to support batches, the implementation may need to be modified. This design conflates inventory management with price calculation. A Single Responsibility Principle-based

design would break apart the functionality such that the vending machine has an inventory and a cost calculator. Application of the SRP tends to break larger objects into smaller, more cohesive objects. This increased cohesion is reminiscent of the cognitive concepts of chunking, essentially the SRP calls for smaller, independent chunks. It also aligns with CLT guidelines of paring content down to essentials and writing high coherent text for low-knowledge learners.

2.1.1 Open-Closed Principle

The Open-Closed Principle states that software entities should be open for extension but closed for modification [58]. This seemingly unachievable goal drives designs towards simple abstractions that change rarely but can add behaviors to a system through different implementations. For example, an online retailer may select the right hello message to display on a landing page based on the country the customer is in based on a Greeter class. If that class uses a hardcoded array, whenever that retailer expands to a new country, the Greeter will have to change. If instead the design employs a STRATEGY for Greeting, new implementations can be added without modifying old code. This also optimizes chunking to differentiate between abstraction and implementation, which is reminiscent of the CLT guidelines of teaching system components before teaching the whole process and teaching supporting knowledge separate from procedure steps.

2.1.2 Newspaper Metaphor

The Newspaper Metaphor is described as a way of arranging code like the headlines and sections of a newspaper, with headlines and headings that describe the content to follow [18]. In this style of arrangement, a programmer tries to co-locate functions with their usage such that the narrative flows from abstract to concrete. For example,

“In order to parse HTML, one reads from a string and tokenizes it. In order to read from a string, one reads character by character into a buffer until reaching the null terminator. In order to tokenize, one splits the string by a delimiter...” This most directly relates to CLT in terms of sequencing. It is reminiscent of the CLT guideline of giving learners control over pacing and managing cognitive load when pacing must be instructionally controlled.

2.1.3 Design Patterns

Design Patterns emerged from Software Engineering adopting the approach of Christopher Alexander [64] to describe different architectural patterns and applying it to software. There are patterns for object-oriented analysis and design, implementation, software architectures, and many other areas of software engineering. Design Patterns evince the CLT guideline of imposing Germane Cognitive Load to enhance efficiency by helping learners automate new knowledge and skills, forming a specialized lexicon that can enable experienced and proficient practitioners to communicate complex concepts quickly.

2.1.4 Intrinsic, Germane, and Extraneous Cognitive Load

Cognitive Load Theory often classifies load into three types: Intrinsic, germane, and extraneous [5]. Intrinsic Cognitive Load is the basis of CLT, the irreducible complexity of concepts. Intrinsic Cognitive Load derives as a function of the number of distinct elements working memory needs to process and the interconnections between them [7]. CLT does not seek to eliminate intrinsic cognitive load as it cannot be eliminated without obliterating the underlying concept. Instead, CLT seeks to manage intrinsic cognitive load. Two strategies often employed to limit the intrinsic cognitive load are sequencing and chunking [4]. Chunking takes similar elements and

groups them together, allowing the working memory to treat the numerous units as a single unit for storage and retrieval. Sequencing sorts the elements in an ordering that enables progression with minimal backtracking or cross-referencing. Germane Cognitive Load is the cognitive load involved in activities that is actually in service to the desired outcome. For example, doing different types of problems when learning equations to see how they apply in various contexts can increase the germane cognitive load. Learning specific vocabularies such as Design Patterns to succinctly communicate concepts also increases Germane Cognitive Load. Extraneous cognitive load is caused by a sub-optimal arrangement or presentation of content demands more working memory attention from things that are irrelevant. Redundant verbiage or distracting pictures can introduce extraneous cognitive load by shifting attention away from the desired goal. A lot of CLT work seeks to manage intrinsic load, optimize germane load, and reduce extraneous load as much as possible.

2.1.5 Split Attention Effect

Colloquially, split attention occurs when one is reading a passage and feels annoyance when it instructs them to flip back or forward for more information [65]. It is very similar to the idea of the overhead involved in context switching when it comes to concurrent thread management in software engineering. The Split-Attention Effect is mitigated by integrating content to be self-descriptive and cohesive without having to jump back and forth.

2.1.6 Expertise Reversal Effect

An important finding for instructional design is that many of the techniques effective to help novices acquire schemas have no effect or actually impede the understanding of experts [66]. As someone gains experience and builds the necessary

mental schemas, it's often helpful to reduce the cognitive load management methods used because they serve as substitutes for pre-built schemas for novices. In any study examining CLT with respect to software, it's helpful to consider if evidence of the expertise reversal effect can be observed. For instance, LIterate Programming when writing a compiler may be extremely helpful for someone unfamiliar with compiler theory and design, but may actually impede the understanding of an expert who is just trying to find the language's grammar and doesn't need a lesson in Context Free Grammars in the comments.

2.1.7 Applied Example of the types of Cognitive Load in software

As an applied but simplistic example, consider the case of an electronic retailer. Perhaps in an initial rapid prototype the retailer's website offers some static Hyper-Text Markup Language to greet a customer.

```
<html>
  <head>
    <title>An E-commerce retailer</title>
  </head>
  <body>
    <h1> Hello Dear Customer!</h1>
    <h2> Please <a href="buy/12345">buy this product</h2>
  </body>
</html>
```

Figure 2.1: insert caption

The Intrinsic Cognitive Load of this markup includes an understanding of the syntax and semantics of HTML, the ability to read the English language, and the knowledge that there is another component referenced by a hyperlink that completes the purchasing process. The Germane Cognitive Load includes some semantic understanding of the intent of the application—such as knowing that the 12345 corresponds to a product identifier that is used by another page to trigger purchasing. The Extra-

neous Cognitive Load includes the lack of an ending close tag for the anchor- `</a>`. Someone reading this document later may have difficulty understanding whether the intent was to only make the word “buy” a hyperlink, or whether the intention was to “buy this product.”

Suppose market research discovers that more purchases occur if the greeting is warm and friendly. The product team asks the development team to change the greeting to a formulation that has been shown to lead to more purchases, addressing the customer as “Dear, dear customer.” The development team has design options. They may implement the change by simply changing the text, as shown below:

```
<html>
  <head>
    <title>An E-commerce retailer</title>
  </head>
  <body>
    <!-- Feature1: Market research revealed
    that "Dear dear Customer" leads to more sales -->
    <h1> Hello Dear dear Customer!</h1>
    <h2> Please <a href="buy/12345">buy this product</h2>
  </body>
</html>
```

Figure 2.2: insert caption

This simple change adds another word to the document, but has minimal effect on the intrinsic, germane, or extraneous cognitive load. A comment has been added to clarify why this changed.

Suppose further market research reveals that “Dear dear Customer” works well for people who shop with their browser set to the Spanish language, but “Valued Shopper” works better for others. The Intrinsic Cognitive Load of this application will increase. Either different pages will need to be loaded by the HTTP server based upon the customer’s location, or the markup will need to be updated in a way that

enables the greeting to be displayed dynamically. The development team may choose the second option and generate the code below:

```
<html>
  <head>
    <title>An E-commerce retailer</title>
  </head>
  <body>
    <!-- Feature1: Market research revealed
    that "Dear dear Customer" leads to more sales -->
    <h1> Hello <span id="greeting">Placeholder</span>!</h1>
    <h2> Please <a href="buy/12345">buy</a> this product</h2>
  </body>
  <script type="text/javascript">
    var loadGreeting = function() {
      var span = document.querySelector("#greeting");
      if ('textContent' in span) {
        span.textContent = greeting();
      } else {
        span.innerHTML = greeting();
      }
    }

    /*
    used for language detection from
    https://stackoverflow.com/questions/1043339/javascript-for-detecting-browser-language-preference
    */
    var greeting = function() {
      var lang = window.navigator.languages[0] ||
        [window.navigator.language || window.navigator.userLanguage]
      if (lang === "es") {
        return "Dear dear Customer";
      }
      return "Valued Shopper";
    }

    document.onready = loadGreeting();
  </script>
</html>
```

Figure 2.3: insert caption

Not only has the Intrinsic Cognitive Load of this application increased conceptually, but the Cognitive Load imposed by the code has increased dramatically as well. A dynamic element has been introduced to the document through the use of Javascript, which will replace the HTML `<span>` with an id of “greeting” rendered text based on the detected value of the language when the document is ready. The Germane Cognitive Load now includes an understanding of Javascript language syntax, semantics,

and eventing model, localization issues of language, and the platform-specific nuances of different browsers. The comment that was helpful in the static “simple” update is now Extraneous Cognitive Load, because what is rendered is no longer just “Dear dear Customer”, it may also be “Valued Shopper.” Future developers may question whether the code is correct because the comment tells them it should do one thing, but the code itself does another.

The Intrinsic Cognitive Load of the application cannot be reduced without changing its functionality. This Cognitive Load can be managed by sequencing and chunking. An enterprising developer may refactor the application as show below:

```
<html>
  <head>
    <title>An E-commerce retailer</title>
  </head>
  <body>
    <h1> Hello <span id="greeting">Placeholder</span>!</h1>
    <h2> Please <a href="buy/12345">buy</a> this product</h2>
  </body>
  <script src="greeting.js" type="module"></script>
  <script type="module">
    import greetCustomer from './greeting.js'
    greetCustomer(document.querySelector("#greeting"));
  </script>
</html>
```

Figure 2.4: insert caption

```

import getUserPreferredLanguage from './language.js'
import crossPlatformSpanTextUpdateCommand from './cross_platform_span_update.js'

export default function greetCustomer(location) {
  loadGreeting(location, getUserPreferredLanguage());
}

function loadGreeting(span, lang) {
  var text = greeting(lang);
  crossPlatformSpanTextUpdate(span, text);
}

function greeting(lang) {
  if (lang === "es") {
    return "Dear dear Customer";
  }
  return "Valued Shopper";
}

```

Figure 2.5: insert caption

```

/*
used for language detection from
https://stackoverflow.com/questions/1043339/javascript-for-detecting-browser-language-preference
*/
export default function getUserPreferredLanguage() {
  return window.navigator.languages[0] ||
    [window.navigator.language || window.navigator.userLanguage];
}

```

Figure 2.6: insert caption

```

export default function crossPlatformSpanTextUpdateCommand(span, text) {
  if ('textContent' in span) {
    span.textContent = text;
  } else {
    span.innerText = text;
  }
}

```

Figure 2.7: insert caption

This refactoring introduces the concepts of sequencing and chunking. It adds Germane Cognitive Load by using JavaScript's modules functionality, but the complexity of the original HTML file is dramatically reduced as its bits are chunked across multiple files, meant to be read and understood sequentially. I propose the following hypotheses as links between software craftsmanship practices and CLT:

- The Single Responsibility Principle is a technique that reduces Extraneous Cognitive Load and manages the Intrinsic Cognitive Load of classes by promoting smaller classes. In the above example, the single responsibility of the HTML page is to invoke the greetCustomer function.
- The Open/Closed Principle helps beginners manage Germane Cognitive Load by enabling consumers of classes to leverage existing components while avoiding cognitive overload by abstracting away inner details. In the above example, getUserPreferredLanguage is a default function whose implementation is closed to modification, but communicates intention through its name.
- The Newspaper Metaphor is a technique for minimizing the Split Attention Effect, reducing Extraneous Cognitive Load. In the above example, methods are arranged according to their usage.
- Design Patterns form a higher-level vocabulary that increases Germane Cognitive Load for intermediate/advanced developers, allowing richer conversations with less distracting detail. In the above example, the COMMAND pattern is used to convey that a function has a void return type and performs a side-effect.
- Smaller functions/methods with descriptive names align closely with Miller's Magic Number Seven Plus or Minus Two [58], allowing programmers to hold more of a method's functionality in their heads at once. This reduces context switching and makes the overall component easier to understand. This pattern scales. Small methods, small classes, small packages, small libraries, small frameworks are more cognitively available for novices, enabling quicker, less error-prone development. In the above example, none of the functions have more than 7 lines.

2.2 Theory & Predictions

Cognitive Load Theory has shown measurable and repeatable results that have influenced Instructional Design. Refactoring patterns are widely accepted for reducing technical debt and making code easier to understand and modify, but it is hard to quantify this effect. In some software engineering companies, the process of peer code review is used as a mechanism for quality control and to manage technical debt. In code review discussions, the application of certain Refactoring techniques or the order in which they are applied can give rise to disagreements that they are simply about style versus substance. Someone without previous background in patterns literature may argue that the introduction of a pattern makes code more complicated. Sometimes one with such a background may argue that the wrong pattern is being applied. Decision points about whether a block of code should become a separate method or class, the names of identifiers, the control structures leveraged, and other points of the software design and implementation can become a horse trading affair where software engineers try to persuade each other based on appeals to authority, tribal knowledge, and individual preference. Cognitive Load Theory offers a structured approach for measuring and quantifying the technical debt of a software project, and a conceptual framework to guide these discussions. Software teams can apply a practice of periodically reviewing certain classes in their code-base and subjectively reporting the estimated cognitive load. Such a practice would allow, over time, to use the measured change in reported cognitive load as a metric to include in the calculation of the technical debt of a project.

Additionally, CLT guidelines can change the way we think about documenting and understanding code. For instance, an application of the Modality Effect suggests that embedding UML diagrams directly in source code that involves spatial reason-

ing, and providing audio narration could improve the comprehensibility of the code. Integrated Developer Environments such as Eclipse Juno, IntelliJ IDEA 2014 and Visual Studio 2010 don't support this type of experience natively. The guidelines of worked examples and self-explanations can provide a cognitive basis for the efficacy of Test/Behavior Driven Development. Guidelines related to separating system components from processes and supporting knowledge from procedure steps can inform arguments behind the viability of Interface Oriented Design [67] and the choices behind particular abstractions.

The complexity of software most often becomes an issue when the code has to change. The primary drivers of change in software are finding a defect or desiring a new feature. When the code must be changed, there is a risk that previous correct behavior may become incorrect, or that the performance profile—in terms of CPU utilization, memory usage, IO, or other factors—may change in a dramatic way that affects the availability of the system. This is when the technical debt associated with the system may cause the necessary change to take hours, weeks, or more. If Cognitive Load Theory can quantify the technical debt a system experiences effectively, one would expect to find that systems with high technical debt impose high cognitive load on their maintainers and users. These high technical debt systems would contain more design flaws, more implementation bugs, and changes to these systems would take longer to implement and validate, hence being more likely to introduce regressions from desired behavior. Conversely, systems with lower technical debt should impose lower cognitive load. The number of design flaws and implementation defects should be lower, the average time to resolve issues should be less, and changes should introduce fewer regressions. Consequently, code that is refactored to manage its cognitive load more effectively should be code where one can more easily fix an outstanding bug, the attempt at fixing should introduce less regressions on average,

and the engineers who attempt the fix should report less perceived cognitive load from the code they read and modify. A useful experiment would look at an existing piece of software with well-defined functionality and a known bug, create a refactored version that maintains the presence of the bug but more effectively manages the cognitive load, then verify that it is easier to fix the refactored version and that engineers feel less cognitive load while trying.

2.3 Research Question: Can Cognitive Load Theory provide empirical evidence and a conceptual framework for the efficacy of Refactoring?

The non-directional null-hypotheses to test, in mathematical terms, are:

1. Time of completion (control) = Time of completion (refactored)
2. Perceived Cognitive Load (control) = Perceived Cognitive Load (refactored)
3. Average defects introduced (control) = average defects introduced (refactored)

These hypotheses should be testable with a 95% ($p < .05$) confidence level and desired power of 80% ($\pi = 0.80$).

2.3.1 *Why will this be meaningful?*

One would expect to find the code refactored to manage cognitive load requires less average time to fix defects and introduces less regressions while trying. This will enable the research community to explore further interventions and evaluate questions such as “do some principles have a higher impact when applied than others?”, “is the effect of the principles in any way tied to programming language?”, and “could results be affected familiarity with program domain?” Additionally, showing the efficacy of Cognitive Load Theory’s application to software engineering creates a wealth of opportunities to explore its recommendations for visualizations, organization of project

documentation, and personalized tooling support for software engineer of various skill levels.

2.3.2 What could go wrong?

It is possible that authors wildly misrepresent the effects of the programming heuristics. Consequently, divergent results upon measurement may be unlike previous Cognitive Load experiments. Perhaps even with classical techniques of measuring Cognitive Load one will be unable to find a relationship between programming performance and Cognitive Load optimization techniques. This would suggest CLT is not as strong a fit for software as the original hypotheses suppose. Additionally, Cognitive Load Theory states the principle of the Expertise Reversal Effect, which says that some practices that highly benefit novices will flummox experts and vice-versa. A robust experiment should identify instances of this by targeting an equal mix of novice and expert participants. One would expect that content increases Germane Cognitive Load, such as the lexicon of Design Patterns, would improve the performance of intermediate and proficient practitioners but may induce cognitive overload—and hence have a negative impact—on novice performance. Alternately, partitioning methods and classes to minimize the risk of cognitive overload may be unnecessary for an expert. Such efforts may improve novice performance, but inhibit the analysis of an expert who can “hold more of the code in their head.”

2.3.3 How can one measure the Cognitive Load of code?

The Cognitive Load Theory literature describes multiple different options for measuring cognitive load. Paas et al provide a useful literature review [68] that surveys analytical and empirical methods such as subjective rating, physiological response, and concurrent-task assessment. In separate work, Paas et al compared subjective

ratings with psychophysiological response readings and found that subjective rating is reliable and sensitive enough to be useful without the significant cost and implementation barriers that come with more technological measurement [69]. Brunken included direct forms of cognitive load measurement such as EEG [70] and found that a dual-task method may work well for multimedia. Given that the introduction and measurement of CLT are new when it comes to software, I chose a subjective rating scale where participants numerically rank items based on perceived difficulty—commonly known as a 7-point Likert Scale [93]—to measure the induced cognitive load of code snippets.

With these conditions in mind, the next section presents the experiment that tested these hypotheses.

Chapter 3

OVERVIEW OF PRESENT EXPERIMENT

In this section I detail why certain participants are selected, which piece of software is used for the trial and why it is picked, the experimental procedure and how it relates to the research questions driving this work. At the end of this section, the reader should have the necessary background to understand the experiment.

3.1 Participants

In this study, I sought to achieve meaningful results with a 95% confidence interval and 80% power looking at the differences between real production code with refactoring supported by cognitive load theory principles applied and code without. I wanted to explore the difference in mean time to fix a bug, regressions introduced, and perceived cognitive load between experienced programmers and novices. Using the G*Power application [73], I computed a minimal sample size of 185 participants. For ease of calculation, I got 188.

The development practice of those who have multiple years of experience is different from those without, as is commonly seen in expert/novice studies in areas such as chess [71], circuit analysis[72], and nursing [73]. Studies in programming have explored expert schema generation and ways of reading code [74, 44, 75]. Many of these studies have very basic modeling of programming expertise, classifying participants as experts or novices. Models such as the five stage Dreyfus Model [76] have not been widely applied in the research literature. Nevertheless, companies often hire programmers with criterion based on years of experience, despite the lack of a strong research body of evidence that shows correlation between expertise and years pro-

gramming. For the purposes of this study, I chose to segment years of programming experience based on common industry qualifications: <5 years for novices, 5+ years for experienced.

3.2 Materials

3.2.1 *Software Chosen - Joda Time*

For this study, I used the Joda Time date/time library. Joda is a popular open-source library used in thousands of projects for date/time manipulation for Java. Java already has a Date/Calendar library built into the language designed for these types of operations. However, many developers within the Java community were dissatisfied with the complexity of the interface Date/Calendar provides for achieving operations such as Date/Time arithmetic, time zone conversion, and formatting dates to standard formats such as ISO8601. Because of this complexity—which this research can show is a measurable difference in the cognitive load imposed by these two libraries—Joda Time has become so popular that the author of the library was asked to completely re-write date/time handling for Java 8.

3.2.2 *Why Joda Time?*

1. Joda Time seeks to reduce the complexity of an existing interface, meaning that it has made an effort to manage the intrinsic cognitive load of date/time manipulation for users.
2. Joda Time has a very large user base and is a high impact project within the Java community. Joda Time sits in the dependency tree of many popular software packages such as Spring, the Lift web application framework, Hibernate annotations, and many, many more. As such, this is not a contrived exercise

dreamed up by a graduate student in a lab to show an effect. This is real code, battle-tested and used by millions.

3. Joda Time is written in Java. Java and C++ are the most well-known programming languages in the community. Java is taught in many college curriculums; finding familiar programmers is less difficult than other languages.
4. Joda Time solves a very general problem—date & time manipulation. Most languages have date libraries for common operations like arithmetic, formatting, or timezone conversion. Good and usable libraries provide broad, usable abstractions for complicated functionality. Date & Time manipulation is very complicated, often causing bugs even in major professional software engineering environments. Programmers across different software engineering domains—whether they build web applications, defense contracting products, or video games—are likely to be able to “grok” it. While esoteric details of handling locale differences and nanoseconds may themselves be complex, the generality of the problem makes it more suitable for development by engineers with various backgrounds than a task such as modifying a Machine Learning library like Weka.

3.2.3 *What experimental intervention did I make to Joda-time?*

I used two versions of Joda Time. The control version is unmodified source from GitHub, forked from the master branch on July 24th, 2015. The experimental version applies Refactoring to the control version aligning with precepts of Cognitive Load Theory such that:

- Variable identifiers are renamed for clarity

- Clean Code: Avoid mental mappings → CLT: Integrate Explanatory Text Close to Related Visuals on Pages and Screens to Avoid Split Attention
- Each method has no more than 7 ± 2 lines
 - Clean Code : functions do no more than one thing → CLT: Write High Coherent Texts for Low Knowledge Readers
- Methods are arranged according to their usage
 - Clean Code : Newspaper Metaphor → CLT : Display Worked Examples and Completion Problems in Ways That Minimize Extraneous Cognitive Load
- Each class has no more than 7 ± 2 methods
 - Clean Code : Classes do no more than one thing → CLT: Write High

3.2.4 *What change did the participants have to make?*

Joda Time at this commit contained a bug in its parsing logic for ISO8601 dates. The ISO8601 standard is common enough to be implemented in many different programming languages and be a standard convention for transmitting dates in distributed systems. This makes the bug broadly comprehensible, have mass applicability, and a good candidate for investigation.

3.2.5 *Bug: ISO8601 Years*

The “devil in the detail” is all of the “edge” cases where bugs happen. One of the devilish details is the existence of a large, amorphous, ever-changing set of formats describing a date. The International Standards Organization attempted to ameliorate this with ISO8601[82], which defines a canonical format for the interchange of dates

and times across locales. One of the quirks of the standard is that the published format is YYYY-MM-DD, however, the standard allows for more than 4-digit years in cases where sender and receiver have agreed upon extra digits by prepending with a + or - [$\pm$ YYYYYY]. Joda Time did not originally account for this. This led to the filing of the bug report:

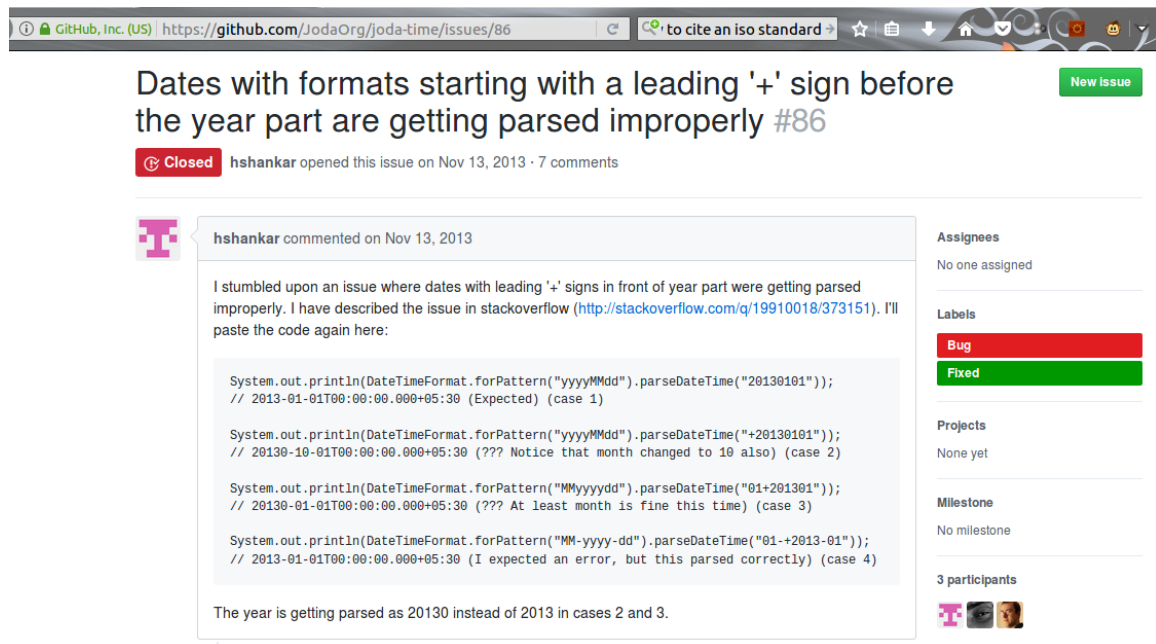


Figure 3.1: insert caption

Hari Shankar explored the depths of the code, identified an issue, and made a fix. He did not report how long it took to find the bug, nor how difficult it was to come up with a solution. When I first found this issue, it was still open. In October of 2015, Steven Colebourne (primary author/maintainer of the library) merged in Shankar's fix.

Fix basic parsing where additional plus sign

Based on code by Hari Shankar - <https://github.com/hshankar>

Fixes #86

[Browse files](#)

jodastephen committed on Oct 24, 2015
 1 parent 4289f5e commit 0a201881f01bce85efece778345ebd60cf58ba35

Showing 3 changed files with 36 additions and 10 deletions.

Unified
 Split

3

 RELEASE-NOTES.txt

View

@@ -20,6 +20,9 @@ Changes in 2.8.3

20 20
 21 21 - Faster parsing of time-zone identifiers [#282]
 22 22
 23 23 +- Fix parsing of basic form ISO style where year has unnecessary plus sign [#86]
 24 24 + For example, +20151030 will now be correctly parsed as year 2015.
 25 25 +
 26 26 - Added Interval.parseWithOffset(String) [#299, #296]
 27 27 Provides a way to parse the fixed offset in an interval string
 28 28

21

 src/main/java/org/joda/time/format/DateTimeFormatterBuilder.java

View

@@ -1304,25 +1304,21 @@ public int parseInt(DateTimeParserBucket bucket, CharSequence text, int positio

1304 1304 int limit = Math.min(Math.maxParsableDigits, text.length() - position);
 1305 1305 boolean negative = false;
 1306 1306 boolean positive = false;
 1307 1307 int length = 0;
 1308 1308 while (length < limit) {
 1309 1309 char c = text.charAt(position + length);
 1310 1310 if (length == 0 && (c == '-' || c == '+') && !signed) {
 1311 1311 negative = c == '-';
 1312 1312 positive = c == '+';
 1313 1313 // Next character must be a digit.
 1314 1314 if (length + 1 >= limit ||
 1315 1315 (c = text.charAt(position + length + 1)) < '0' || c > '9')
 1316 1316 {
 1317 1317 (c = text.charAt(position + length + 1)) < '0' || c > '9' {
 1318 1318 break;
 1319 1319 }
 1320 1320 length++;
 1321 1321
 1322 1322 if (negative) {
 1323 1323 length++;
 1324 1324 } else {
 1325 1325 // Skip the '+' for parseInt to succeed.
 1326 1326 position++;
 1327 1327 }
 1328 1328 // Expand the limit to disregard the sign character.
 1329 1329 limit = Math.min(limit + 1, text.length() - position);
 1330 1330 continue;
 1331 1331
 1332 1332 if (length >= 9) {
 1333 1333 // Since value may exceed integer limits, use stock parser
 1334 1334 // which checks for this.
 1335 1335 value = Integer.parseInt(text.subSequence(position, position += length).toString());
 1336 1336 if (positive) {
 1337 1337 value = Integer.parseInt(text.subSequence(position + 1, position += length).toString());
 1338 1338 } else {
 1339 1339 value = Integer.parseInt(text.subSequence(position, position += length).toString());
 1340 1340 }
 1341 1341 value = Integer.parseInt(text.subSequence(position, position += length).toString());
 1342 1342 } else {
 1343 1343 int i = position;
 1344 1344 if (negative) {
 1345 1345 if (negative || positive) {
 1346 1346 i++;
 1347 1347 }
 1348 1348 }
 1349 1349 try {

Figure 3.2: Shankar's Fix

3.2.6 *Analysis of Accepted Solution*

The accepted solution follows a common maintenance programmer practice: change as little code as possible and duplicate where necessary and with slight tweaks to achieve desired behavior. It is not exactly but similar in vein to “Programming By Difference.” It is a conservative approach that tends to be favored when code is in obsolescence, has a large number of consumers, or is otherwise hard to change.

“Other reasons” that code can be hard to change include a lack of confidence in correct behavior, or difficulty in understanding algorithms and data structures written by someone else. The commit that introduced the fix included unit test cases that manifest the bug (though notably only try 5 digit years, perhaps not fully exercising failure modes), suggesting the maintainer understood the code enough to augment the existing test suite, a software best practice. The original code was not written solely by Stephen Colebourne. Attribution information in the comments suggests it was a collaborative effort between Colebourne, Brian S. O’Neill, and Fredrik Borgh. It’s possible this had an effect on the aggressiveness of the fix.

Using SonarQube 5.4 for analysis, I found that the `DateTimeFormatterBuilder` alone had 1600 LoC. 2625 total lines (including comments) with 162 issues found via static source analysis, with 2.3% duplications, estimated to take 2.5 days to fix. It had a Cyclomatic Complexity score of 550, the highest in the library.

When gathering statistics of the library with the “Run Tests with Coverage” option of IDEA IntelliJ, Joda Time exhibits an impressive 98% class/91% line/90% method coverage. Such a high level of coverage potentially provides developers the option to experiment with more drastic structural Refactoring and verify existing behavior, but time and other issues may not always make such re-architecture possible.

For the purposes of this experiment, I hold the state of JodaTime as reflected in

Git commit on August 2nd 2015 as the “control” group, not including the given fix. I use the provided fix as the “gold standard”, as it was accepted by the core library contributor. I then ask participants to attempt to fix the bug following a short tutorial on the ISO8601 standard and the Joda Time library. I follow a 2x2 factorial design where I account for novice/expert and control/experimental measuring debugging time, number of defects introduced as detected by failing unit tests, and perceived cognitive load as reported on the Likert Scale. Non-software related factors that potentially introduce error in measurement include time of day, environmental factors in the deviation from “normal programming place” (office, coffee shop, home, et cetera) versus “clean room”, and participant mood/focus/mental factors. I treat non-software related factors as nuisance variables.

3.3 Experimental Environment

To run the study, I developed a lab machine with multiple Integrated Development Environments to suit the needs of Java programmers: Eclipse, IDEA IntelliJ, Oracle NetBeans, Sublime Text Editor, Vim, and Emacs. The lab machine included all necessary software, an Ubuntu 11.10 base OS running Java Development Kit 8 update 21, Git, installed versions of Eclipse Neon, IntelliJ IDEA 14, NetBeans 8.2, Sublime Text 3, VIM 7.1, and emacs 24.2.

This was a conscious concession to permit the introduction of another nuisance variable—IDE familiarity—into the project.

3.3.1 *Aside: The IDE effect*

Java developers have a lot of flexibility in the toolchain they use, limiting to one particular environment could artificially narrow the participant pool or influence the results. For example, an experienced Eclipse developer may have a significantly

impaired code navigability and debugging experience in Vim, or an Emacs guru could spend valuable code investigation time instead learning the graphical user interface of IntelliJ. Making one tool mandatory could significantly deviate the sample from the programmer population, discouraging some from involvement and hamstringing the rest. I wanted to simulate the field experience of programming as closely as possible, giving participants leverage to integrate prior knowledge and work with code in their naturally preferred workflows. The trade-off is that I did observe effects where toolchain differences had an effect on the study data, including load/processing times and perspective switches of Eclipse and a stability issue with NetBeans.

The in-person approach worked effectively to pilot the study and normalize the workflow and data collection. The only problem was the time-intensiveness of the process. Since the target number of participants was 185, I needed a way of scaling out the study such that participants could independently and in isolation do the exercise in parallel. Consequently, I replicated the lab machine setup in a virtual machine image using Oracle VirtualBox 4.0. I uploaded this virtual machine image onto my website, then wrote a script with instructions sent via email to participants who registered using the online questionnaire. The virtual machine was configured to automatically record the screen and capture audio, creating a very similar experience to the in-person session. Participants would securely send me the captured video file and the filled out Likert Scale for the code listing, along with any notes that corresponded to their impressions after a debrief prompt. This development enabled my research to reach for more participants than it may otherwise have; it seems an ideal way to make a study like this capable of reaching a large number of programmers from different backgrounds all over the world.

3.3.2 Cognitive Load measurement - Likert Scale

I give participants an annotated listing (included in appendices) that applies a 7-point Likert Scale to the code listing in order to analyze perceived Cognitive Load according to common techniques.

Aside: Likert Scale construction

Likert Scales are not commonly applied to code listings. Figuring out how to do so is a point of experimental design. It is an expansive design space, abundant with options. Naively, one may consider applying a scale at points such as every statement/expression, every line of code, every function/method, every class definition, at package level, or even at the bundled executable/library level. I'll discuss some of the strengths and shortcomings of this approach.

Likert Scale: every expression/statement

A Likert Scale on every single expression or statement offers perhaps the most granular sensor for how difficult code is to understand. It's also the most "cost prohibitive" in measurement. Any code whose full functionality that does more computation/evaluation than simple algorithms such as sum/max/min can become incredibly difficult both to instrument this way and for participants to respond. For my study, I chose "real-life production code" precisely to avoid generalized inference of programmer thought process from "toy examples", a common critique of research into the cognitive aspects of software. As measured by SonarQube, the `DateTimeFormatterBuilder` alone has 2625 total lines, and multiple expressions or statements can be in a single line. Giving participants 5000 elements to rank assuredly biases them towards Respondent Fatigue. As this is cutting-edge research intended

to establish the viability of this approach, it is absolutely imperative to minimize corruption of the data based on Respondent Fatigue. I can ameliorate this concern by setting some axioms of things that are “easy to understand”—where one would expect the average participant to respond 1 on a 1-7 least difficult/most difficult scale.

3.3.3 *Minimal Cognitive Load of expressions axioms*

- Declaration of a variable (e.g. `int x;`)
- Assignment of a value to a variable (e.g. `int x = 2;`)
- Unary and Binary mathematical operations (e.g. `x++`, `x + y`, `x mod y`)

These axioms are challengeable. For instance, the usage of a type in a variable declaration is programming language dependent. There may be measurable differences in simplicity of understanding declarations based on the inherent type systems of the language. The way that such affects comprehensibility may be unexpected. Perhaps in a typeless language such as Assembly, or a language that allow anonymous types/”Duck Typing” such as Javascript, `var x = 4` causes respondents less Cognitive Load than `int x = 4`. Conversely, perhaps the Germane Cognitive Load induced by the assumptions of types such as “an Integer in Java must be a whole number between -231 to 231-1” makes the second statement easier to understand. Both of these conditions may actually be determined by the target programmer’s prior experience with programming and languages.

For another example, bit shifting of whole numeric values can be succinctly described as unary/binary mathematics operations, but that may underestimate the cognitive complexity a programmer has to compute to understand what is happening. This comes up in my study in a very concrete way, as the computation of year values during the parsing step contains expressions that befuddled many participants.

```
value = ((value << 3) + (value << 1)) + text.charAt(i++) - '0';
```

Figure 3.3: insert caption

This computation relies on a traditional programmer idiom from the assembly days, that $i*10 = (i<<3) + (i<<1)$. This shortcut is not documented in the code itself and is not one high-level application developers and other software engineers encounter on a daily basis. Every participant that encountered the line during in-person trials reported feeling uncomfortable with it. Some took the time to prove to themselves that the formula worked, many took it as given. Surely this line is not equivalent in cognitive complexity to a simple variable assignment. In these cases, I can compute a metric using Halstead’s software metrics or Mira’s Modified Cognitive Complexity Measure [77], but none of these metrics could quite capture the Cognitive Load of a prior knowledge lookup from a well-known programmer vocabulary pattern-matching schema. In principle, capturing the subjective Cognitive Load experienced by programmers encountering this line in production code could inform researchers more about the average conceptual vocabulary of programming than other mechanisms.

3.3.4 Likert Scale: line-by-line

A Likert Scale on every single line of code is slightly less expansive than expression/statement, but not by much. Respondent Fatigue is still an overriding concern. The axioms can be expanded further to include line specific concepts.

3.3.5 Minimal Cognitive Load of lines axioms

- Return statements (e.g. `return x + y;`)

- Comments (e.g. `// Next character must be a digit`)

3.3.6 *Likert Scale: blocks/scopes*

This was ultimately the approach I chose to have participants respond to. Functions/Methods, classes, packages—all of these artifacts are essentially programming language conveniences to define scope at different levels of abstraction. This scope management maps very cleanly to Cognitive Load Theory’s concepts of Chunking and Sequencing. Code blocks form lexical and conceptual closures where programmers can partition their understanding of the global state of the program to management pieces. As there fewer than 100 blocks to be analyzed, I avoid the problem of Responder Fatigue but can still capture valuable granular data.

3.3.7 *Minimal cognitive load of blocks/scopes axioms*

- Getters (e.g. `public int getCurrentPosition() { return currentPosition};`
`)`
- Setters (e.g. `public void setPosition(final int position) { this.position`
`= position}; )`

3.3.8 *Open Question: How do we measure code flow cognitive complexity?*

Likert Scales at the block level get me closer, but still don’t quite capture all of the complexity programmers encounter when understanding code. For an applied example, I can examine the code flow of the `DateTimeFormatter`. The `DateTimeFormatter` uses an `InternalParser`, which is implemented by a static abstract class `NumberFormatter` inside of `DateTimeFormatterBuilder`. This `NumberFormatter` is a nested class that serves as the root of an inheritance hierarchy, for classes that include `UnpaddedNumber`, `PaddedNumber`, and `FixedNumber`. Many participants who provided feedback about

the design in the study expressed confusion and incredulity, some expressed outright contempt as they struggled to navigate between classes and trace control flow. This type of activity is simple to observe but difficult to capture quantitatively, and the self-reported Cognitive Load of block of code may not be able to tell us about the cognitive complexity added by interactions between the blocks. This remains an active area of potential innovation in study design and measurement for subsequent research. As a general guideline, software designers should avoid deep inheritance hierarchies or long method chains where the number of jumps in control flow begins to approach human short-term memory limits.

The full Likert scale instrument for measuring cognitive load can be found in the appendices, here I'll show an example question for maximum clarity:

The instrument presents some code—a method of a class— with a 7 point scale where respondents can rank its complexity. For consistency and simplicity of data analysis, in my study this same scale is repeated for every question, only the code changes.

Figure 1: DateTimeFormatter#parseDateTime

```
/**
 * Parses a date-time from the given text, returning a new DateTime.
 * <p>
 * The parse will use the zone and chronology specified on this formatter.
 * <p>
 * If the text contains a time zone string then that will be taken into
 * account in adjusting the time of day as follows.
 * If the {@link #withOffsetParsed()} has been called, then the resulting
 * DateTime will have a fixed offset based on the parsed time zone.
 * Otherwise the resulting DateTime will have the zone of this formatter,
 * but the parsed zone may have caused the time to be adjusted.
 *
 * @param text the text to parse, not null
 * @return the parsed date-time, never null
 * @throws UnsupportedOperationException if parsing is not supported
 * @throws IllegalArgumentException if the text to parse is invalid
 */
public DateTime parseDateTime(String text) {
    InternalParser parser = requireParser();

    Chronology chrono = selectChronology(null);
    DateTimeParserBucket bucket = new DateTimeParserBucket(InstantLocal: 0, chrono, iLocale, iPivotYear, iDefaultYear);
    int newPos = parser.parseInto(bucket, text, position: 0);
    if (newPos >= 0) {
        if (newPos >= text.length()) {
            long millis = bucket.computeMillis(resetFields: true, text);
            if (iOffsetParsed && bucket.getOffsetInteger() != null) {
                int parsedOffset = bucket.getOffsetInteger();
                DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
                chrono = chrono.withZone(parsedZone);
            } else if (bucket.getZone() != null) {
                chrono = chrono.withZone(bucket.getZone());
            }
            DateTime dt = new DateTime(millis, chrono);
            if (iZone != null) {
                dt = dt.withZone(iZone);
            }
            return dt;
        } else {
            newPos = ~newPos;
        }
        throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
    }
}
```

How hard is this method to understand?

	1	2	3	4	5	6	7	
Very easy	☐	☐	☐	☐	☐	☐	☐	Very hard

Figure 3.4: Date Time Formatter

In this section I have detailed the rationales for criterion I used to select participants, the software library I used and how I modified it for the experimental purpose, the environment developed to conduct the experiment, and the construction of the measurement instrument for cognitive load. In the next section, I'll detail the execu-

tion of the study.

Chapter 4

METHOD

In the previous section I established that I provided an overview of the experiment I would run to explore a link between software refactoring and its corresponding concepts in Cognitive Load Theory. I proposed this through a classical CLT experimental design: run a 2x2 Factorial experiment split between novices and experienced software engineers for a control and refactored version of the JodaTime Java library trying to fix a bug and measuring time to complete, regressions introduced, and perceived cognitive load. In this section, I detail how I did so. Participants

I solicited participants through an online questionnaire. This questionnaire (included in the appendices) was simply for intake screening and validated eligibility per compliance with the IRB requirements of the study and prerequisite knowledge. This questionnaire was distributed through college professor classrooms, company mailing lists, developer bulletin-boards, and online advertising. I contacted qualifying participants after assigning them to 1 of 4 blocks based on their responses (Novice/Control, Novice/Experimental, Expert/Control, Expert/Experimental), assigning them a randomized identifier, and discarding non-essential personal information to obtain their informed consent. I then scheduled a 1 hour session for conducting the study in person for local participants and sent a link to a virtual machine lab environment for remote participants.

4.1 Materials Design

I developed the experimental refactored version of Joda Time relying heavily on Refactoring patterns while applying the CLT principles of sequencing, chunking, re-

moving redundancy, and introducing germane cognitive load through the direct usage of Design Patterns. The exact same behavior was maintained, while new classes were added using `SPROUT CLASS`, new methods were added using `EXTRACT METHOD`, and variables were renamed and re-ordered for clarity. Afterwards, I ran SonarQube 5.4's code analysis for metrics to compare their complexity using traditional software engineering tools. From a pure complexity metrics perspective, `DateTimeFormatterBuilder` went from a control complexity score of 550 to 531 in the refactored. The complexity per function dropped from 3.5 to 3.4. The number of lines of code dropped by 58. The number of duplications dropped from 2.3% to 0.7%. `DateTimeFormatter` showed a slightly more pronounced effect. The complexity score dropped from 98 to 64. The complexity per function went from 2.5 to 1.7. The number of lines of code dropped by 110. The 2.9% duplications dropped to 0. I've included the full elaboration of the set of transformations in the appendices.

4.2 Intervention, Measures, and Procedure

Participants reviewed a short tutorial on ISO8601 and debugging in their preferred development environment before commencing the timed study. Participants had the opportunity to ask clarifying questions and take notes before they started exploring the code. When ready, they began a 1-hour timed session where they could explore and modify code while trying to fix the bug. When they ran the tests to verify a fix, I took note of failing tests as a measure of regressions introduced. When the participants either fixed the bug or an hour passed, the session ended and I recorded the total elapsed time to the nearest minute. After the session was complete, participants received a survey via e-mail with 7-point Likert Scales with questions where they answered "how hard is this code to understand? (1 = very easy, 7 = very hard)" for the code they investigated.

Chapter 5

RESULTS

5.1 Mean Response Time for participants who fixed the bug

I ran a 2x2 factorial ANOVA using SPSS for the response time amongst successful participants and recorded the following results:

➔ **ANOVA for Mean Response Time of fixing bug**

Between-Subjects Factors			
		Value Label	N
Trial	0	Refactor	27
	1	Control	13
Expertise	0	Expert	27
	1	Novice	13

Descriptive Statistics

Dependent Variable: Time

Trial	Expertise	Mean	Std. Deviation	N
Refactor	Expert	46.42	8.739	19
	Novice	50.50	5.732	8
	Total	47.63	8.082	27
Control	Expert	53.87	6.034	8
	Novice	54.20	6.458	5
	Total	54.00	5.930	13
Total	Expert	48.63	8.643	27
	Novice	51.92	6.048	13
	Total	49.70	7.969	40

Table 5.1: Mean Response Time of fixing bug

Tests of Between-Subjects Effects								
Dependent Variable: Time								
Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared	Noncent. Parameter	Observed Power ^b
Corrected Model	450.093 ^a	3	150.031	2.666	.062	.182	7.997	.602
Intercept	83606.728	1	83606.728	1485.383	.000	.976	1485.383	1.000
Trial	247.518	1	247.518	4.397	.043	.109	4.397	.532
Expertise	38.586	1	38.586	.686	.413	.019	.686	.127
Trial * Expertise	28.037	1	28.037	.498	.485	.014	.498	.106
Error	2026.307	36	56.286					
Total	101280.000	40						
Corrected Total	2476.400	39						

a. R Squared = .182 (Adjusted R Squared = .114)

b. Computed using alpha =

Table 5.2: Test of Between Subjects Effects

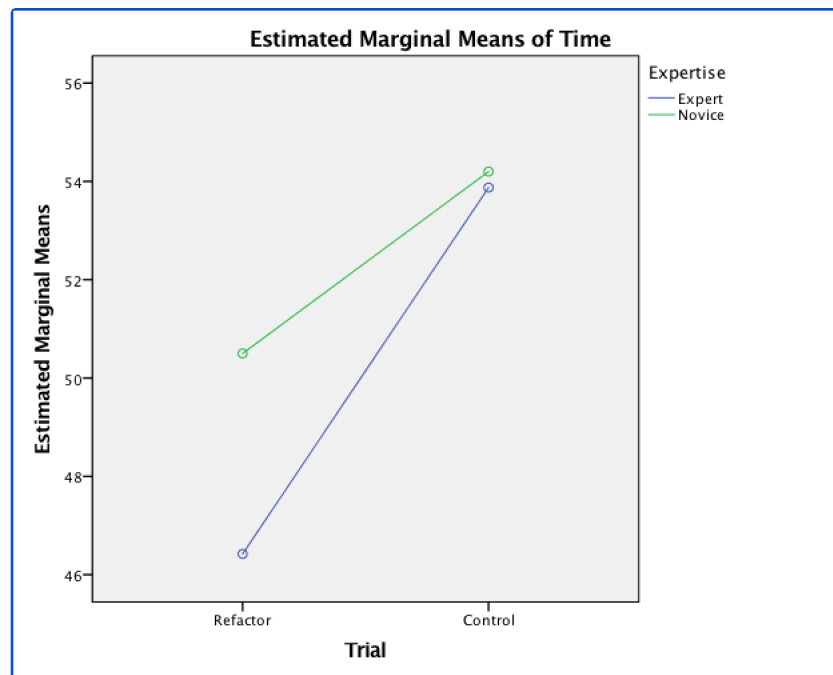


Figure 5.1: Estimated Marginal Means of Time

The data shows strongly that the main effect of Refactoring according to CLT principles causes a statistically significant difference in mean time to resolution for fixing a bug at $\alpha=.05$. Hence we reject the null hypothesis that the mean time to resolution is equal between the cases.

5.2 Mean Regressions for those who did not fix the bug

I ran a 2x2 factorial ANOVA using SPSS for the regression rate amongst participants who did not fix the bug and recorded the following results:

→ Regression rate ANOVA for those who did not fix bug

[DataSet1]

Between-Subjects Factors

	Value Label	N
Expertise	0 Expert	64
	1 Novice	81
Trial	0 Refactor	65
	1 Control	80

Descriptive Statistics

Dependent Variable: Regressions

Expertise	Trial	Mean	Std. Deviation	N
Expert	Refactor	42.62	103.499	26
	Control	492.39	972.431	38
	Total	309.67	780.505	64
Novice	Refactor	407.21	1224.063	39
	Control	1002.05	1579.214	42
	Total	715.64	1441.972	81
Total	Refactor	261.37	962.403	65
	Control	759.96	1342.681	80
	Total	536.46	1209.378	145

Table 5.3: Mean Response Time of fixing bug

Tests of Between-Subjects Effects

Dependent Variable: Regressions

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared	Noncent. Parameter	Observed Power ^b
Corrected Model	16170714 ^a	3	5390238.15	3.909	.010	.077	11.726	.819
Intercept	33093130.9	1	33093130.9	23.997	.000	.145	23.997	.998
Expertise	6691012.62	1	6691012.62	4.852	.029	.033	4.852	.590
Trial	9553139.04	1	9553139.04	6.927	.009	.047	6.927	.743
Expertise * Trial	184222.229	1	184222.229	.134	.715	.001	.134	.065
Error	194442959	141	1379028.08					
Total	252342376	145						
Corrected Total	210613674	144						

a. R Squared = .077 (Adjusted R Squared = .057)

b. Computed using alpha =

Table 5.4: Test of Between Subjects Effects

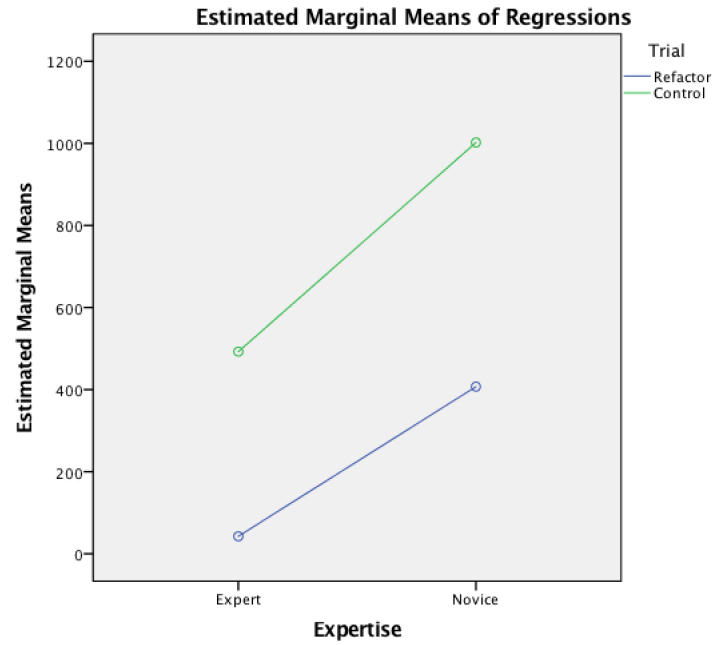


Figure 5.2: Estimated Marginal Means of Regressions

From this data we can observe that there are statistically significant main effects for both expertise and the effect of refactoring, with no interaction effect. Thus, we can reject the null hypothesis that the regression rate is equal between the control code and the refactored code.

5.3 Perceived Cognitive Load

I analyzed the perceived cognitive load by comparing the mean, median, and mode across the control and refactored conditions and examining the differences between overlapping questions. For the refactored condition, the average perceived cognitive load was 3.38, the median was 3, and the mode was 2. Amongst novices, the mean was 3.68, the median was 4, and the mode was 2. Amongst experienced, the mean was 3.08, the median was 3, and the mode was 2. For the control condition, the average perceived cognitive load was 3.73, the median was 3, and the mode was 3. Amongst

novices, the mean was 4.06, the median was 3.5, and the mode was 3. Amongst experienced, the mean was 3.39, the median was 3, and the mode was 1.

The number of questions asked in the survey differed between the treatments, which I'll return to in limitations. There was encountered code that remained unchanged between the two treatments, so exploring the reported perceived cognitive load between them may shed light on the total load imposed by the code base. If the average perceived load is similar, that suggests that programmers partition off the effects of the whole architecture and analyze methods individually. If it is not, that raises the question of whether the perceived cognitive load is influenced by the code around it. For the questions with overlap, the median and mode perceived cognitive load were the same in both the experimental and the control conditions. This suggests that the perceived cognitive load of methods can be reliably measured independent of surrounding context.

5.4 Discussion

- i. Time of completion (control) = Time of completion (experiment)
- ii. Perceived Cognitive Load (control) = Perceived Cognitive Load (experiment)
- iii. Average defects introduced (control) = average defects introduced (experiment)

Based on the preceding results, I can summarize that experience matters in solving a complex problem, as more experts fixed the bug than novices. It required them less time and fewer mistakes on average. But the effect of managing Cognitive Load in code via Refactoring enhances the performance of both. The time of completion and average defects introduced are not equal @ $p < .05$. The average and median of the perceived cognitive load across the 16 experimental questions and the 10 control questions are similar, but the experienced cognitive load in the experimental condition

is often smaller. The control case had 11 questions, the experimental case had 20 (the refactored code was broken out into smaller chunks, so took more blocks to encompass). Some of the code in the questions between the two was the same, much was different. The average cognitive load reported in the code that was the same was approximately equal. This coincides with the theory that Refactoring manages Germane Cognitive Load and removes Extraneous Cognitive Load but does not reduce Intrinsic Cognitive Load. That is usually done by re-engineering. I did not find evidence of the Expertise Reversal Effect when reducing method and class size to more granularly partition out functionality amongst methods and classes. I did not find evidence that the use of Design Patterns terminology applied Germane Cognitive Load to the problem solving process—some experienced engineers self-reported not knowing what a STRATEGY is for the DateTimeParsingStrategy, others felt it was misapplied in this context. More work is needed to identify programmer familiarity with Design Patterns and identify their effect on cognitive load as a function of familiarity.

Chapter 6

LIMITATIONS

For the purposes of selecting programmer populations and sorting by experience, I used a conventional industry classification of years of experience [78] as barometer of expertise. This is widely panned in popular literature [79] and not very well established in the research literature. Many studies are completed with self-selected experts [80] or use distinctions between number of courses taken and skill level [81]. After this study began, new tools that gamify programming such as HackerRank and CodeFights emerged that provide more a more granular analysis of expertise than years of experience. Such tools, if they measure skill and expertise more distinctly, may be better to use in subsequent studies to target experts and novices. Without using such tools, it would be helpful to calibrate participants with 1-2 practice problems before conducting the study and having a panel of experts review their responses to more accurately assign them.

The survey used to measure the perceived cognitive load contained the same language between the experimental and control conditions, but not the same number of questions. It is impossible to rule out that the longer refactored survey had results impacted by respondent fatigue. Although its mean cognitive load was still computed to be lower than the control, in future studies the number of questions should be held constant across both.

Chapter 7

IMPLICATIONS

Finding that the main effect was statistically significant for the Refactoring according to CLT principles but expertise was not for mean time to resolution of the bug was provocative. It suggests that fears that the Expertise Reversal Effect may make the code easier to read only for novices are as yet unsubstantiated. Finding that the perceived cognitive load of overlapping code snippets was similar for both the control and refactored case suggests that CLT may be reliable measure for the technical debt of a software project. Reliability and validity are two important attributes of useful conceptual frameworks and robust metrics, cognitive load as it relates to code may have both to offer the software engineering community. Showing that there is a statistically significant effect for refactored code that adheres to the principles of Cognitive Load Theory introduces new opportunities for the application of CLT to software engineering tooling. Taking into account the Modality Effect when developing documentation to describe a system or constructing tooling to create systems may dramatically affect the way we think about software visualization and implementation. Cognitive Load Theory itself may be informed by software engineering concepts for designing modules/interconnected systems. The principles of Information Hiding, Encapsulation, and resource independence inherent in Service Oriented Architectures may impact the way Instructional Designers organize blocks of content.

FUTURE DIRECTIONS

The measurement of Cognitive Load in the code was collected via a follow-up survey with Likert Scale on the blocks of code that were actively debugged and engaged. There is a possibility that the perceived ex post facto differs from the load experienced at the time. It would be useful to devise an IDE plugin or some kind of instrument that could get participant feedback in real-time. There may also be interesting results derived through the use of some biometric technologies such as eye tracking, EEG, or fMRI.

This study focused as much as possible on testing the validity of its hypotheses with “real world code.” The JodaTime library is at times arcane, complex, and has years of accrued complexity that many participants remarked on throughout the study. There are multiple ways it would be helpful to reproduce these results with different libraries. For one, there may be so much accumulated cognitive load in a large library that looking for smaller libraries with defined problem spaces like JavaPoet may be helpful. Attempting on general purpose code that works in different problem domains such as the Apache Tomcat web application server or the OpenJDK may serve as validation for generality or find specific domains where managing cognitive load produces novel results.

Finally, it’s often argued that programming languages themselves have differences in “expressivity” [82] in a way that suggests the intrinsic cognitive load may vary between languages such as Java, LISP, and Python. It would be worthwhile to investigate whether the same techniques yield similar results across languages, or whether the effects are different and language-dependent. Moreover, many of the techniques

we applied were harvested from Refactoring catalogs that had discovered analogues to Cognitive Load Theory based from the Object-Oriented Programming paradigm. It's often argued by practitioners that Functional Programming is more cognitively available, though the pattern literature for FP was not as voluminous when the study began. It would be helpful to have a deeper investigation of the links between functional programming and Cognitive Load Theory.

The measurement of Cognitive Load in code remains an open area of exploration. In this study, I chose to measure based on blocks of cohesive code at the method/function level. It is an open question how to quantify and explore the cognitive load imposed by an entire class as “something more than the sum of the cognitive load of its methods.” The relationship between classes, their usage, and how that affects the aggregated Cognitive Load of a component is also worth exploring.

Finally, as systems move from a co-located in the same address space model towards distributed systems/service-oriented architectures, additional complexity is introduced with respect to consistency, availability, and partition tolerance [83]. This complexity is often addressed through tactics that include retrying in case of network failures, caching, and consensus-based algorithms, all of which impose additional cognitive load. As software becomes increasingly complex and responsible for more of the important work of humanity—such as autonomous vehicles and intelligent personal assistants—understanding development practices to make software easy to understand and fix defects in becomes ever more vitally important. Cognitive Load Theory offers a useful set of principles that have already been applied in instructional design, this work suggests it can be useful in software engineering and may help us “escape the tar pit.”

References

- [1] L. Briand, “Embracing the Engineering Side of Software Engineering,” *IEEE Software*, vol. 29, no. 4, p. 96, Jul. 2012. [Online]. Available: <http://dx.doi.org/10.1109/ms.2012.86>
- [2] P. Johnson, M. Ekstedt, and I. Jacobson, “Where’s the Theory for Software Engineering?” *IEEE Software*, vol. 29, no. 5, p. 96, Sep. 2012. [Online]. Available: <http://dx.doi.org/10.1109/ms.2012.127>
- [3] I. Jacobson and I. Spence, “Why We Need a Theory for Software Engineering,” Oct. 2009. [Online]. Available: <http://www.drdoobs.com/architecture-and-design/why-we-need-a-theory-for-software-engine/220300840>
- [4] J. Sweller, “Cognitive Load During Problem Solving: Effects on Learning,” *Cognitive Science*, vol. 12, no. 2, pp. 257–285, Apr. 1988. [Online]. Available: http://dx.doi.org/10.1207/s15516709cog1202_4
- [5] J. Sweller and P. Chandler, “Evidence for cognitive load theory,” *Cognition and instruction*, vol. 8, no. 4, pp. 351–362, 1991.
- [6] G. A. MILLER, “The magical number seven plus or minus two: some limits on our capacity for processing information,” *Psychological review*, vol. 63, no. 2, pp. 81–97, Mar. 1956. [Online]. Available: <http://view.ncbi.nlm.nih.gov/pubmed/13310704>
- [7] A. Baddeley, “Working memory: looking back and looking forward,” *Nature Reviews. Neuroscience*, vol. 4, no. 10, p. 829, 2003. [Online]. Available: https://labs.wsu.edu/attention-perception-performance/documents/2016/05/baddeley_review_2003.pdf
- [8] T. J. McCabe, “A complexity measure,” *Software Engineering, IEEE Transactions on*, no. 4, pp. 308–320, 1976. [Online]. Available: <http://juacompe.mrchoke.com/natty/thesis/FrameworkComparison/A%20complexity%20measure.pdf>
- [9] M. H. Halstead, *Elements of software science*, ser. Elsevier computer science library : operational programming systems series. New York, NY: North-Holland, 1977.
- [10] A. J. Albrecht and J. E. Gaffney, “Software function, source lines of code, and development effort prediction: a software science validation,” *Software Engineering, IEEE Transactions on*, no. 6, pp. 639–648, 1983.

- [11] Y. Wang, “On the cognitive complexity of software and its quantification and formal measurement,” *International Journal of Software Science and Computational Intelligence (IJSSCI)*, vol. 1, no. 2, pp. 31–53, 2009.
- [12] J. Shao and Y. Wang, “A new measure of software complexity based on cognitive weights,” *Electrical and Computer Engineering, Canadian Journal of*, vol. 28, no. 2, pp. 69–74, 2003.
- [13] P. Naur and B. Randell, Eds., *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*, 1969. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1102020>
- [14] B. Moseley and P. Marks, “Out of the Tar Pit,” *Complexity*. [Online]. Available: <http://shaffner.us/cs/papers/tarpit.pdf>
- [15] S. Hastie and S. Wojewoda. (2015, Oct.) Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch. [Online]. Available: <https://www.infoq.com/articles/standish-chaos-2015>
- [16] M. Jørgensen and K. Moløkken-Østvold, “How large are software cost overruns? A review of the 1994 CHAOS report,” *Information and Software Technology*, vol. 48, no. 4, pp. 297–301, Apr. 2006. [Online]. Available: <http://dx.doi.org/10.1016/j.infsof.2005.07.002>
- [17] R. L. Glass, “The Standish report: does it really describe a software crisis?” *Communications of the ACM*, vol. 49, no. 8, pp. 15–16, Aug. 2006. [Online]. Available: <http://dx.doi.org/10.1145/1145287.1145301>
- [18] R. C. Martin, J. O. Coplien, K. Wampler, J. W. Grenning, B. L. Schuchert, J. Langr, T. R. Ottinger, and M. C. Feathers, *Clean code : a handbook of agile software craftsmanship*, 1st ed. Prentice Hall, Aug. 2016. [Online]. Available: <http://www.worldcat.org/isbn/0132350882>
- [19] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mallor, K. Shwaber, and J. Sutherland, “The Agile Manifesto,” The Agile Alliance, Tech. Rep., 2001.
- [20] K. Beck, *Test Driven Development: By Example*, 1st ed. Addison-Wesley Professional, Nov. 2002. [Online]. Available: <http://www.worldcat.org/isbn/0321146530>
- [21] E. Hendrickson, “Driving Development with Tests: ATDD and TDD,” in *STAR-WEST*. Software Quality Engineering, Oct. 2008.
- [22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, Nov. 1994. [Online]. Available: <http://www.worldcat.org/isbn/0201633612>

- [23] E. Freeman, B. Bates, K. Sierra, and E. Robson, *Head First Design Patterns: A Brain-Friendly Guide*, 1st ed. O'Reilly Media, Nov. 2004. [Online]. Available: <http://www.worldcat.org/isbn/0596007124>
- [24] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Conallen, and K. A. Houston, *Object-Oriented Analysis and Design with Applications (3rd Edition)*, 3rd ed. Addison-Wesley Professional, Apr. 2007. [Online]. Available: <http://www.worldcat.org/isbn/020189551X>
- [25] A. Shalloway and J. R. Trott, *Design Patterns Explained: A New Perspective on Object-Oriented Design (2nd Edition)*, 2nd ed. Addison-Wesley Professional, Oct. 2004. [Online]. Available: <http://www.worldcat.org/isbn/0321247140>
- [26] K. Beck, W. Cunningham, and W. S. Services. (1989) A laboratory for teaching object-oriented thinking. <http://c2.com/doc/oopsla89/paper.html>. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.129.4074>
- [27] R. Wirfs-Brock, B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software*, 1st ed. Prentice Hall, Jun. 1990. [Online]. Available: <http://www.worldcat.org/isbn/0136298257>
- [28] G. Wilson, "What We Actually Know About Software Development, and Why We Believe It's True," in *CUSEC*, 2010.
- [29] R. A. Tarmizi and J. Sweller, "Guidance during mathematical problem solving." *Journal of educational psychology*, vol. 80, no. 4, p. 424, 1988.
- [30] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)*, anniversary ed. Addison-Wesley Professional, Aug. 1995. [Online]. Available: <http://www.worldcat.org/isbn/0201835959>
- [31] K. Beck, *Extreme Programming Explained: Embrace Change*, us ed ed. Addison-Wesley Professional, Oct. 1999. [Online]. Available: <http://www.worldcat.org/isbn/0201616416>
- [32] P. McBreen, *Software Craftsmanship: The New Imperative*, 1st ed. Addison-Wesley Professional, Sep. 2001. [Online]. Available: <http://www.worldcat.org/isbn/0201733862>
- [33] E. W. Dijkstra, "Letters to the Editor: Go to Statement Considered Harmful," *Commun. ACM*, vol. 11, no. 3, pp. 147–148, Mar. 1968. [Online]. Available: <http://dx.doi.org/10.1145/362929.362947>
- [34] —, "Notes on structured programming. 1969, appeared in OJ Dahl, EW Dijkstra, and CAR Hoare (eds): *Structured Programming*," 1972.
- [35] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Prentice Hall, Apr. 1988. [Online]. Available: <http://www.worldcat.org/isbn/0131103628>

- [36] D. L. Parnas, “On the Criteria to Be Used in Decomposing Systems into Modules,” *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972. [Online]. Available: <http://dx.doi.org/10.1145/361598.361623>
- [37] A. C. Kay, “The Early History of Smalltalk,” *SIGPLAN Not.*, vol. 28, no. 3, pp. 69–95, Mar. 1993. [Online]. Available: <http://doi.acm.org/10.1145/155360.155364>
- [38] D. E. Knuth, *Literate Programming (Center for the Study of Language and Information - Lecture Notes)*, 1st ed. Center for the Study of Language and Inf, Jun. 1992. [Online]. Available: <http://www.worldcat.org/isbn/0937073806>
- [39] —, “Literate Programming,” *The Computer Journal*, vol. 27, no. 2, pp. 97–111, Jan. 1984. [Online]. Available: <http://comjnl.oxfordjournals.org/content/27/2/97.full.pdf+html>
- [40] N. Ramsey, “Literate Programming Simplified,” *IEEE Softw.*, vol. 11, no. 5, pp. 97–105, Sep. 1994. [Online]. Available: <http://dx.doi.org/10.1109/52.311070>
- [41] G. White and M. Sivitanides, “Cognitive Differences Between Procedural Programming and Object Oriented Programming,” *Information Technology and Management*, vol. 6, no. 4, pp. 333–350, Oct. 2005.
- [42] C. Douce, “The stores model of code cognition,” in *In Programmer Psychology Interest Group*, 2008. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.222.9075>
- [43] B. Curtis, “Measurement and experimentation in software engineering,” *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1144–1157, 1980.
- [44] B. Curtis, I. Forman, R. Brooks, E. Soloway, and K. Ehrlich, “Psychological perspectives for software science,” *Information Processing & Management*, vol. 20, no. 1-2, pp. 81–96, Jan. 1984. [Online]. Available: [http://dx.doi.org/10.1016/0306-4573\(84\)90041-4](http://dx.doi.org/10.1016/0306-4573(84)90041-4)
- [45] R. C. Clark, F. Nguyen, and J. Sweller, *Efficiency in Learning: Evidence-Based Guidelines to Manage Cognitive Load*. Pfeiffer, Dec. 2005. [Online]. Available: <http://www.worldcat.org/isbn/0787977284>
- [46] N. Hollender, C. Hofmann, M. Deneke, and B. Schmitz, “Integrating cognitive load theory and concepts of human–computer interaction,” *Computers in Human Behavior*, vol. 26, no. 6, pp. 1278–1288, Nov. 2010. [Online]. Available: <http://dx.doi.org/10.1016/j.chb.2010.05.031>
- [47] S. Oviatt, “Human-centered Design Meets Cognitive Load Theory: Designing Interfaces That Help People Think,” in *Proceedings of the 14th Annual ACM International Conference on Multimedia*, ser. MULTIMEDIA ’06. New York, NY, USA: ACM, 2006, pp. 871–880. [Online]. Available: <http://dx.doi.org/10.1145/1180639.1180831>

- [48] J. Rumbaugh, R. Jacobson, and G. Booch, *The Unified Modelling Language Reference Manual*, 1st ed. Addison-Wesley, Jan. 1999. [Online]. Available: www.amazon.co.uk/exec/obidos/ASIN/020130998X/026-2174472-9898019
- [49] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, 1st ed. Addison-Wesley Professional, Feb. 1999. [Online]. Available: <http://www.worldcat.org/isbn/0201571692>
- [50] K. Beck, *Implementation Patterns*, 1st ed. Addison-Wesley Professional, Nov. 2007. [Online]. Available: <http://www.worldcat.org/isbn/0321413091>
- [51] M. Fowler, *Patterns of Enterprise Application Architecture*, 1st ed. Addison-Wesley Professional, Nov. 2002. [Online]. Available: <http://www.worldcat.org/isbn/0321127420>
- [52] T. Parr, *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages (Pragmatic Programmers)*, 1st ed. Pragmatic Bookshelf, Jan. 2010. [Online]. Available: <http://www.worldcat.org/isbn/193435645X>
- [53] L. Rising and N. S. Janoff, “The Scrum software development process for small teams,” *Software, IEEE*, vol. 17, no. 4, pp. 26–32, Jul. 2000. [Online]. Available: <http://dx.doi.org/10.1109/52.854065>
- [54] D. J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, Apr. 2010. [Online]. Available: <http://www.worldcat.org/isbn/0984521402>
- [55] A. Hunt and D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*, 1st ed. Addison-Wesley Professional, Oct. 1999. [Online]. Available: <http://www.worldcat.org/isbn/020161622X>
- [56] M. Fowler and K. Beck, *Refactoring improving the design of existing code*, 1st ed. Addison-Wesley, Jul. 2013. [Online]. Available: <http://www.worldcat.org/isbn/0201485672>
- [57] W. F. Opdyke, “Refactoring object-oriented frameworks,” Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1992. [Online]. Available: <http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7302/ReadingMaterial/Opdyke92.pdf>
- [58] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, 1st ed. Prentice Hall, Oct. 2002. [Online]. Available: <http://www.worldcat.org/isbn/0135974445>
- [59] B. Liskov, “Keynote Address - Data Abstraction and Hierarchy,” in *Addendum to the Proceedings on Object-oriented Programming Systems, Languages and Applications (Addendum)*, ser. OOPSLA ’87. New York, NY, USA: ACM, 1987, pp. 17–34. [Online]. Available: <http://dx.doi.org/10.1145/62138.62141>

- [60] B. Meyer, *Object-Oriented Software Construction (Book/CD-ROM) (2nd Edition)*, 2nd ed. Prentice Hall, Apr. 1997. [Online]. Available: <http://www.worldcat.org/isbn/0136291554>
- [61] A. Abran, P. Bourque, R. Dupuis, and J. W. Moore, Eds., *Guide to the Software Engineering Body of Knowledge - SWEBOK*. Piscataway, NJ, USA: IEEE Press, 2001.
- [62] F. DeRemer and H. Kron, "Programming-in-the Large Versus Programming-in-the-small," *SIGPLAN Not.*, vol. 10, no. 6, pp. 114–121, Apr. 1975. [Online]. Available: <http://dx.doi.org/10.1145/390016.808431>
- [63] A. Hunt, *Pragmatic Thinking and Learning: Refactor Your Wetware (Pragmatic Programmers)*, 1st ed. Pragmatic Bookshelf, Nov. 2008. [Online]. Available: <http://www.worldcat.org/isbn/1934356050>
- [64] C. Alexander, *The timeless way of building*. New York: Oxford University Press, 1979, vol. 1.
- [65] P. Chandler and J. Sweller, "The split-attention effect as a factor in the design of instruction," *British Journal of Educational Psychology*, vol. 62, no. 2, pp. 233–246, 1992.
- [66] S. Kalyuga, P. Ayres, P. Chandler, and J. Sweller, "The Expertise Reversal Effect," *Educational Psychologist*, vol. 38, no. 1, pp. 23–31, Mar. 2003. [Online]. Available: http://dx.doi.org/10.1207/s15326985ep3801_4
- [67] K. Pugh, *Interface-Oriented Design (Pragmatic Programmers)*. Pragmatic Bookshelf, 2006.
- [68] F. Paas, J. E. Tuovinen, H. Tabbers, and P. W. M. Van Gerven, "Cognitive load measurement as a means to advance cognitive load theory," *Educational psychologist*, vol. 38, no. 1, pp. 63–71, 2003.
- [69] F. G. W. C. Paas, J. J. G. Van Merriënboer, and J. J. Adam, "Measurement of cognitive load in instructional research," *Perceptual and motor skills*, vol. 79, no. 1, pp. 419–430, 1994. [Online]. Available: https://www.researchgate.net/profile/Fred_Paas/publication/15390085_Measurement_of_cognitive_load_in_instructional_research/links/55ad2fdb08ae98e661a41759/Measurement-of-cognitive-load-in-instructional-research.pdf
- [70] R. Brunken, J. L. Plass, and D. Leutner, "Direct measurement of cognitive load in multimedia learning," *Educational psychologist*, vol. 38, no. 1, pp. 53–61, 2003. [Online]. Available: http://steinhardtapps.es.its.nyu.edu/create/courses/2174/reading/Bruenen_Plass_Leutner_EP.pdf
- [71] A. Degroot, *Thought and Choice in Chess (Psychological Studies)*, 2nd ed. Mouton De Gruyter, 1968. [Online]. Available: <http://www.worldcat.org/isbn/9027979146>

- [72] D. Egan and B. Schwartz, “Chunking in recall of symbolic drawings,” vol. 7, no. 2, pp. 149–158, 1979. [Online]. Available: <http://dx.doi.org/10.3758/bf03197595>
- [73] P. Benner, “Using the Dreyfus Model of Skill Acquisition to Describe and Interpret Skill Acquisition and Clinical Judgment in Nursing Practice and Education,” *Bulletin of Science, Technology & Society*, vol. 24, no. 3, pp. 188–199, Jun. 2004. [Online]. Available: <http://dx.doi.org/10.1177/0270467604265061>
- [74] K. Ehrlich and E. Soloway, “Human Factors in Computer Systems,” J. C. Thomas and M. L. Schneider, Eds. Norwood, NJ, USA: Ablex Publishing Corp., 1984, ch. An Empirical Investigation of the Tacit Plan Knowledge in Programming, pp. 113–133. [Online]. Available: <http://portal.acm.org/citation.cfm?id=823>
- [75] R. S. Rist, “Plans in Programming: Definition, Demonstration, and Development,” in *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*. Norwood, NJ, USA: Ablex Publishing Corp., 1986, pp. 28–47. [Online]. Available: <http://portal.acm.org/citation.cfm?id=28884>
- [76] S. E. Dreyfus and H. L. Dreyfus, “A five-stage model of the mental activities involved in directed skill acquisition,” DTIC Document, Tech. Rep., 1980.
- [77] Y.-H. Choe, C.-Y. Jong, and S. Han, “Software cognitive information measure based on relation between structures,” *arXiv preprint arXiv:1304.0374*, 2013. [Online]. Available: <https://arxiv.org/pdf/1304.0374.pdf>
- [78] L. Carlson. (2014) Career Path of a Programmer. <https://www.ctl.io/developers/blog/post/career-path-of-a-programmer/>. [Online]. Available: <https://www.ctl.io/developers/blog/post/career-path-of-a-programmer/>
- [79] J. Atwood. (2008) The Years of Experience Myth. <https://blog.codinghorror.com/the-years-of-experience-myth/>. [Online]. Available: <https://blog.codinghorror.com/the-years-of-experience-myth/>
- [80] T. Mastaglio and J. Rieman, “How experts infer novice programmer expertise: a protocol analysis of LISP code evaluation,” in *Proc. Empirical Studies of Programmers, Fourth Workshop*, 1991, pp. 177–195.
- [81] E. Soloway and K. Ehrlich, “Empirical studies of programming knowledge,” *IEEE Transactions on software engineering*, no. 5, pp. 595–609, 1984.
- [82] P. Graham, “Beating the Averages,” 2001. [Online]. Available: <http://www.paulgraham.com/avg.html>
- [83] S. Gilbert and N. Lynch, “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services,” *SIGACT News*, vol. 33, no. 2, pp. 51–59, Jun. 2002. [Online]. Available: <http://doi.acm.org/10.1145/564585.564601>

APPENDIX A

LIKERT SCALE FOR CONTROL STUDY

Cognitive Code Study Code Paths Likert Scale A Likert Scale is a psychometric scale that is commonly used in Cognitive Load Theory driven trials to gauge the subjective level of effort someone feels they need to apply to understand a given stimulus.

1. Email address *

2. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

3. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

4. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

5. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

6. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

7. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

8. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

9. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

10. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

11. How hard is this method to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

☐ Send me a copy of my responses.

APPENDIX B

LIKERT SCALE FOR EXPERIMENTAL STUDY

Cognitive Code Study Code Paths Likert Scale A Likert Scale is a psychometric scale that is commonly used in Cognitive Load Theory driven trials to gauge the subjective level of effort someone feels they need to apply to understand a given stimulus.

1. Email address *

2. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

3. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

4. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

5. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

6. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ○ ○ ○ ○ ○ ○ ○ Very hard

7. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

8. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

9. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

10. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

11. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

12. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

13. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

14. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

15. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

16. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

17. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

18. How hard is this code to understand? Mark only one oval.

1 2 3 4 5 6 7

Very easy ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very hard

APPENDIX C

MEASURING THE COGNITIVE LOAD OF CODE:

Investigating the use of CLT techniques to minimize debugging time and errors

C.1 Review

As I established at the beginning of this dissertation, Cognitive Load Theory is a fruitful field of principles from instructional and user interface design that lies fallow for software engineering. Professional software engineering involves many non-programming related activities, but the core unifying glue is always code. As software engineering practitioner literature has argued, code that is poorly written and difficult to understand can cost companies time, money, and even—such as in the case of the HEARTBLEED OpenSSL bug—expose massive security vulnerabilities just due to a missing set of “ ” for an if-statement.

Code style is arguably important, but attempts to analytically quantify the impact of different techniques remains more art than science. Additionally, researchers such as Lionel Briad and Ivar Jacobson have decried the lack of strong theoretical principles for software engineering. This work explores the intersection of Cognitive Load Theory and software engineering best practices to answer the question: “Is there a relationship between evidence driven best practices for organizing educational content and published best practices for writing code?” Experiment: Fix a bug in date-time manipulation in Joda Time I chose to apply software engineering best practice techniques identified in Clean Code and Refactoring to a popular open-source Java library called Joda-time. Joda-time is one of the most highly-used libraries in the Java ecosystem. Jodatime sits in the dependency tree of many popular software packages such as Spring, the Lift web application framework, Hibernate annotations, and many, many more. As such, this is not a contrived exercise dreamed up by a graduate student in a lab to show an effect. This is real, battle-tested, used by millions code.

I also chose Joda-time because it is written in Java. Java and C++ are the most well-known programming languages in the community. Java is taught in many college curriculums; finding familiar programmers is less difficult than other languages. Joda-time solves a very general problem—date & time manipulation. Most languages have date libraries for common operations like arithmetic, formatting, or timezone conversion. Good and usable libraries provide broad, usable abstractions for complicated functionality. Date & Time manipulation is very complicated, often causing bugs even in major professional software engineering environments.

C.2 Bug: ISO8601 Years

The “devil in the detail” is all of the edge cases. One of the devilish details is the existence of a large, amorphous, ever-changing set of formats describing a date. The International Standards Organization attempted to ameliorate this with ISO8601, which defines a canonical format for the interchange of dates and times across locales. One of the quirks of the standard is that the published format is YYYY-MM-DD,

however, the standard allows for more than 4-digit years in cases where sender and receiver have agreed upon extra digits by prepending with a + or - [$\pm$ YYYYYY].

Jodatime did not originally account for this. This led to <https://github.com/JodaOrg/joda-time/issues/86>. Hari Shankar explored the depths of the code and identified an issue and a fix. He did not report how long it took to find the bug, or how difficult it was to come up with a solution. When I first found this issue, it was still open. In October of 2015, Steven Colebourne (primary author/maintainer of the library) merged in Shankar's fix.

C.3 Analysis of Accepted Solution

The accepted solution follows a common maintenance programmer practice: change as little code as possible and duplicate where necessary and with slight tweaks to achieve desired behavior. It is not exactly but similar in vein to "Programming By Difference." It is a conservative approach that tends to be favored when code is in obsolescence, has a large number of consumers, or is otherwise hard to change.

"Other reasons" that code can be hard to change include a lack of confidence in correct behavior, or difficulty in understanding algorithms and data structures written by someone else. The commit that introduced the fix included unit test cases that manifest the bug (though notably only try 5 digit years, perhaps not fully exercising failure modes), suggesting the maintainer understood the code enough to augment the existing test suite, a software best practice. The original code was not written solely by Stephen Colebourne. Attribution information in the comments suggests it was a collaborative effort between Colebourne, Brian S. O'Neill, and Fredrik Borgh. It's possible this had an effect on the aggressiveness of the fix.

Using SonarQube 5.4 for analysis, we find that the `DateTimeFormatterBuilder` alone has 1600 LoC. 2625 total lines (including comments) with 162 issues found via static source analysis, with 2.3% duplications, estimated to take 2.5 days to fix. It has a Cyclomatic Complexity score of 550, the highest in the library.

When gathering statistics of the library with the "Run Tests with Coverage" option of IDEA IntelliJ, Joda Time exhibits an impressive 98% class/91% line/90% method coverage. Such a high level of coverage potentially provides developers the option to experiment with more drastic structural Refactoring and verify existing behavior, but time and other issues may not always make such re-architecture possible.

C.4 Experiment: contrast debugging time and performance of accepted solution versus CLT optimized

For the purposes of this experiment, we hold the state of JodaTime as reflected in Git commit on August 2nd 2015 as the "control" group, not including the given fix. We use the provided fix as the "gold standard", as it was accepted by the core library contributor. We follow a 2x2 factorial design where we account for novice/expert and control/experimental for debugging time, number of defects introduced as measured by failing tests, and perceived cognitive load as measured on our Likert Scale.

With such setup in mind, I will proceed to show how I applied software engineering best practices influenced by Cognitive Load Theory to develop the experimental treatment.

C.5 Development of Experimental Version

C.5.1 Starting small: sequencing, chunking, and intrinsic complexity at the variable and method level

I began with trying to simplify a complicated if-statement that included assignment, array indexing, and bounding checking in one compound predicate. I was guided by the presence of a comment. Robert Martin’s Clean Code suggests that “Comments are Lies.” They often become out of date as code changes with time.

Comments also signal a failure to communicate, a missed opportunity for the code to be self-descriptive. A programmer who reads a comment incurs the additional cognitive load of reading the comment, having to verify whether it accurately summarizes the situation, and maintaining a mental mapping between the descriptive comment and the cryptic code. Cognitive Load Theory shows that redundant information adds extraneous cognitive load—learning outcomes for material that has a picture and paragraph repetitively describing the same phenomena are generally improved if the information is concisely expressed in one form. Thus, we use the technique of INTRODUCE EXPLAINING VARIABLE to make the conditional more descriptive. Realizing that these temporary variables had added lines to answer a limited scope question within the body of a method that was already 64 lines long, increasing Germane Cognitive Load for understanding the full execution of this method, I used the CLT technique of managing Germane Cognitive Load by sequencing and chunking, applying REPLACE TEMP WITH QUERY to push knowledge of this question into a cohesive chunk, a method.

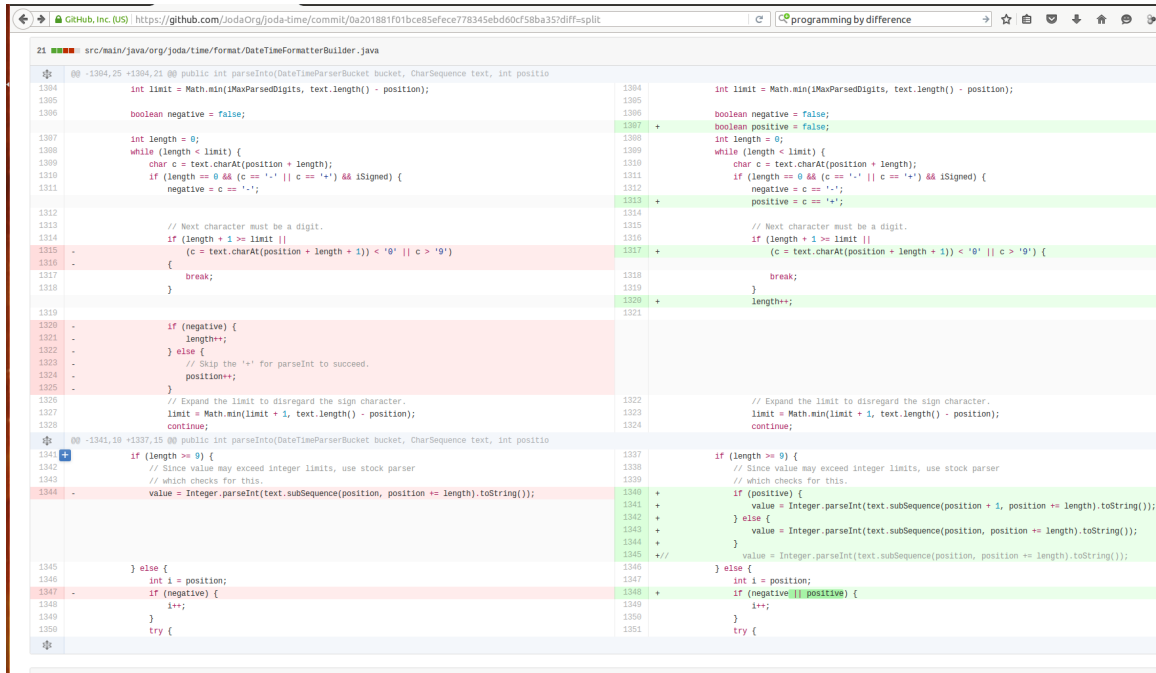


Figure C.1: insert caption

When I ran the tests, I discovered something troubling: a test was broken. The behavior had changed.

```
Tests run: 4161, Failures: 3, Errors: 1, Skipped: 0, Time elapsed:
6.298 sec <<<FAILURE! - in org.joda.time.TestAllPackages
```

```
testFormat_year(org.joda.time.format.TestDateTimeFormat) Time elapsed:
0.015 sec <<<ERROR!
```

```
java.lang.StringIndexOutOfBoundsException: String index out of range:
```

```
1
```

```
    at java.lang.String.charAt(String.java:646)
```

```
    at org.joda.time.format.DateTimeFormatterBuilder$NumberFormatter.parseInto(Date
```

```
    at org.joda.time.format.DateTimeFormatter.parseDateTime(DateTimeFormatter.java:
```

```
    at org.joda.time.format.TestDateTimeFormat.testFormat_year(TestDateTimeFormat.j
```

I stared at the code for a good 20 minutes, not understanding what went wrong. Opdyke defines Refactoring as “structural code changes that exhibit no external behavioral difference.” Clearly what I’d done had ended up not being Refactoring, despite the fact that I’d leveraged the baked in functionality of IDEA IntelliJ Ultimate Edition 14, which includes a robust static analysis checker that usually detects such changes. Pouring over the expression hardly helped, the change “looked safe.” Attaching a debugger and stepping through the broken test revealed the problem.

The predicate of the if statement had caught my eye for having a lot of complexity and a comment to describe it. A hidden complexity that wasn’t immediately obvious to me was that it relied on the short-circuit evaluation behavior of the `——` operator in Java as an implicit GUARD CLAUSE

```
length + 1 >= limit || (c = text.charAt(position + length + 1)) < '0'
|| c > '9')
```

In the previous expression, the `length + 1 >= limit` prevents the `text.charAt(position + length + 1)` from being evaluated when `length + 1` would result trying to index a character outside of the string, causing an error. Thus, there is an additional semantic that provides safety based on the ordering of expressions.

This is exactly the type of hidden complexity that creates anxiety in maintenance engineers in modifying existing code. It is typified by reliance on a subtle semantic provided by the language runtime. This can be considered germane cognitive load—language semantics are part of a programmer’s toolkit. Expert programmers often use these kinds of techniques, as experts are shown to better handle an increase in germane cognitive load. Novices, however, may experience information overload.

I can simplify this by rewriting the code to be a little longer, but more explicit.

applied <https://sourcemaking.com/refactoring/introduce-explaining-var...> [Browse files](#)

applied <https://sourcemaking.com/refactoring/introduce-explaining-variable> some more to simplify expression.

applied <https://sourcemaking.com/refactoring/replace-temp-with-query>

fix-bug-86-experimental-narrative

Vaidyanathan committed with Nick Vaidyanathan on Aug 2, 2015 1 parent 6719673 commit bd2ef554e938312b939b2c0d4385c1e072f6d24b

Showing 1 changed file with 11 additions and 3 deletions. [Unified](#) [Split](#)

14 src/main/java/org/joda/time/format/DateTimeFormatterBuilder.java [View](#)

```

@@ -1310,9 +1310,8 @@ public int parseInto(DateTimeParserBucket bucket, CharSequence text, int positio
1310 1310         if (length == 0 && (c == '-' || c == '+') && iSigned) {
1311 1311             negative = c == '-';
1312 1312
1313 -            // Next character must be a digit.
1314 -            if (length + 1 >= limit ||
1315 -                (c = text.charAt(position + length + 1)) < '0' || c > '9')
1313 +
1314 +            if (isPastBoundaryOrNotDigit(text, position, limit, length))
1316 1315             {
1317 1316                 break;
1318 1317             }
@@ -1364,6 +1363,15 @@ public int parseInto(DateTimeParserBucket bucket, CharSequence text, int positio
1364 1363         bucket.saveField(iFieldType, value);
1365 1364         return position;
1366 1365     }
1366 +
1367 +    private boolean isPastBoundaryOrNotDigit(final CharSequence text, final int position, final int limit, final int length) {
1368 +        final char c; // Next character must be a digit.
1369 +        final char nextCharacter = (c = text.charAt(position + length + 1));
1370 +        final boolean isNotDigit = nextCharacter < '0' || c > '9';
1371 +
1372 +        final boolean isPastBoundary = length + 1 >= limit;
1373 +        return isPastBoundary || isNotDigit;
1374 +    }
1367 1375     }

```

Figure C.2: insert caption

In this example, I make the GUARD CLAUSE more explicit and match the standard form, and use an explanatory named variables to describe the why of the calculation's what.

C.5.2 Building up: moving on to the class level

Returning my attention to the original expression, I apply the same INTRODUCE EXPLANATORY VARIABLE and EXTRACT METHOD technique.:

short-circuit evaluation for `!!` in Java. This is an interesting edge ...

...case.

fix-bug-86-experimental-narrative

Vaidyanathan committed with V on Aug 2, 20151 parent bd2ef55commit b48c762f7131919b8abde2846d78cb762d064347

Showing 1 changed file with 4 additions and 5 deletions.

UnifiedView

9 src/main/java/org/joda/time/format/DateTimeFormatterBuilder.javaView

✱

@@ -1365,12 +1365,11 @@ public int parseInto(DateTimeParserBucket bucket, CharSequence text, int positio

13651365

}

13661366

13671367

private boolean isPastBoundaryOrNotDigit(final CharSequence text, final int position, final int limit, final in

1368

- final char c; // Next character must be a digit.

1369

- final char nextCharacter = (c = text.charAt(position + length + 1));

1370

- final boolean isNotDigit = nextCharacter < '0' || c > '9';

1371

-

13721368

final boolean isPastBoundary = length + 1 >= limit;

1373

- return isPastBoundary || isNotDigit;

+ if (isPastBoundary) { return true; }

+ final char nextCharacter = text.charAt(position + length + 1);

+ final boolean isNotDigit = nextCharacter < '0' || nextCharacter > '9';

+ return isNotDigit;

13741373

}

13751374

}

13761375

✱

Figure C.3: insert caption

I also made the chunked methods static to signify that they were “pure functions.” They had no reliance on the `DateTimeFormatterBuilder`’s internal state and solely exist as calculations. This structural change helped me realize that much of the body of the `parse` method was isolated from the rest of the class. This is often an opportune situation to apply REPLACE METHOD WITH METHOD OBJECT, which can decrease the length and intrinsic cognitive load of the original class via chunking its content with the created collaborating class.

applied <https://sourcemaking.com/refactoring/introduce-explaining-var...>
Browse files

fix-bug-86-experimental + fix-bug-86-experimental-solution

Vaidyanathan committed on Aug 2, 2015
1 parent 2333757
commit 16add8df19b242935857e3f043e483abd93da76d

Showing 1 changed file with 9 additions and 5 deletions.
Unified Split

14 src/main/java/org/joda/time/format/DateTimeFormatterBuilder.java View

```

@@ -1307,10 +1307,10 @@ public int parseInt(DateTimeParserBucket bucket, CharSequence text, int positio
    int length = 0;
    while (length < limit) {
        char c = text.charAt(position + length);
-       if (length == 0 && (c == '-' || c == '+') && isSigned) {
+       final boolean isFirstCharacterOperator = length == 0 && (c == '-' || c == '+');
+       if (isFirstCharacterOperator && isSigned) {
            negative = c == '-';
        }
        if (isPastBoundaryOrNotDigit(text, position, limit, length)) {
            break;
        }
@@ -1326,7 +1326,7 @@ public int parseInt(DateTimeParserBucket bucket, CharSequence text, int positio
        limit = Math.min(limit + 1, text.length() - position);
        continue;
    }
-   if (c < '0' || c > '9') {
+   if (isNotADigit(c)) {
        break;
    }
    length++;
@@ -1364,13 +1364,17 @@ public int parseInt(DateTimeParserBucket bucket, CharSequence text, int positio
    return position;
}

- private boolean isPastBoundaryOrNotDigit(final CharSequence text, final int position, final int limit, final int length) {
+ private static boolean isPastBoundaryOrNotDigit(final CharSequence text, final int position, final int limit, final int length) {
    final boolean isPastBoundary = length + 1 >= limit;
    if (isPastBoundary) { return true; }
    final char nextCharacter = text.charAt(position + length + 1);
-   final boolean isNotDigit = nextCharacter < '0' || nextCharacter > '9';
+   final boolean isNotDigit = isNotADigit(nextCharacter);
    return isNotDigit;
}

+ private static boolean isNotADigit(final char c) {
+   return c < '0' || c > '9';
+ }
}

//-----

```

Figure C.4: insert caption

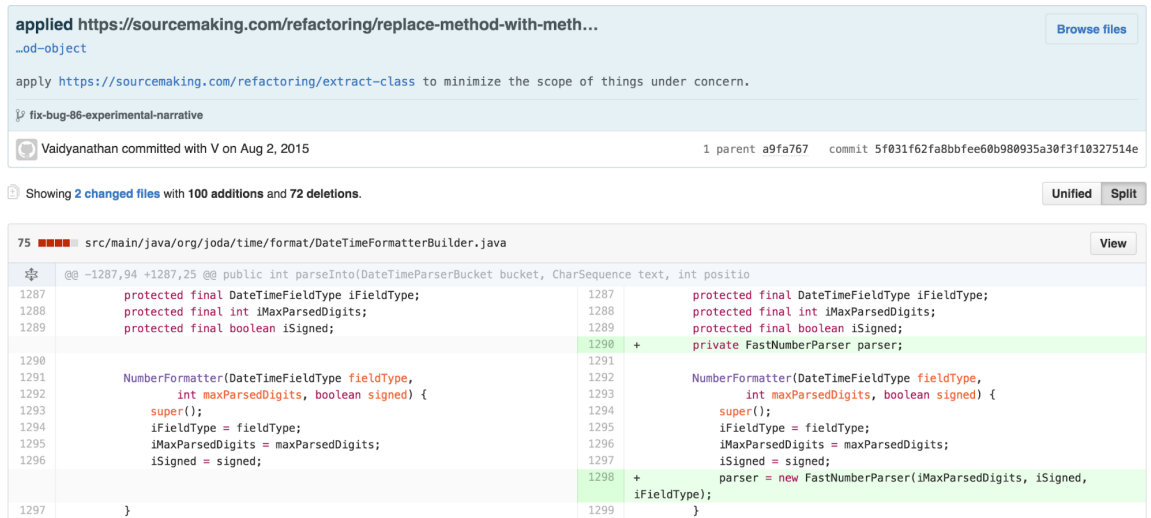


Figure C.5: insert caption



Figure C.6: insert caption

1329	-	if (isNotADigit(c)) {		
1330	-	break;		
1331	-	}		
1332	-	length++;		
1333	-	}		
1334	-			
1335	-	if (length == 0) {		
1336	-	return ~position;		
1337	-	}		
1338	-			
1339	-	int value;		
1340	-	if (length >= 9) {		
1341	-	// Since value may exceed integer limits, use stock parser		
1342	-	// which checks for this.		
1343	-	value = Integer.parseInt(text.subSequence(position,		
	-	position += length).toString());		
1344	-	} else {		
1345	-	int i = position;		
1346	-	if (negative) {		
1347	-	i++;		
1348	-	}		
1349	-	try {		
1350	-	value = text.charAt(i++) - '0';		
1351	-	} catch (StringIndexOutOfBoundsException e) {		
1352	-	return ~position;		
1353	-	}		
1354	-	position += length;		
1355	-	while (i < position) {		
1356	-	value = ((value << 3) + (value << 1)) +		
	-	text.charAt(i++) - '0';		
1357	-	}		
1358	-	if (negative) {		
1359	-	value = -value;		
1360	-	}		
1361	-	}		
1362	-			

Figure C.7: insert caption

1363	-	bucket.saveField(ifFieldType, value);		
1364	-	return position;		
1365	-	}		
1366	-			
1367	-	private static boolean isPastBoundaryOrNotDigit(final CharSequence		
	-	text, final int position, final int limit, final int length) {		
1368	-	final boolean isPastBoundary = length + 1 >= limit;		
1369	-	if (isPastBoundary) { return true; }		
1370	-	final char nextCharacter = text.charAt(position + length + 1);		
1371	-	final boolean isNotDigit = isNotADigit(nextCharacter);		
1372	-	return isNotDigit;		
1373	-	}	1307	}
1374	-		1308	
1375	-	private static boolean isNotADigit(final char c) {		
1376	-	return c < '0' c > '9';		
1377	-	}		
1378	-	}	1309	}
1379	-		1310	
1380	-	//-----	1311	//-----
	-	----		----
	-			

Figure C.8: insert caption

```
97 ■■■■ src/main/java/org/joda/time/format/FastNumberParser.java View
... .. @@ -0,0 +1,97 @@
1 +package org.joda.time.format;
2 +
3 +import org.joda.time.DateTimeFieldType;
4 +import org.joda.time.format.DateTimeParserBucket;
5 +
6 +final class FastNumberParser {
7 +    private final DateTimeFieldType iFieldType;
8 +    private final int iMaxParsedDigits;
9 +    private final boolean iSigned;
10 +
11 +    public FastNumberParser(final int maxParsedDigits,
12 +                           final boolean signed,
13 +                           final DateTimeFieldType fieldType) {
14 +        iFieldType = fieldType;
15 +        iMaxParsedDigits = maxParsedDigits;
16 +        iSigned = signed;
17 +    }
18 +
19 +    public int parse(final DateTimeParserBucket bucket,
20 +                    final CharSequence text,
21 +                    final int position) {
22 +        int limit = Math.min(iMaxParsedDigits, text.length() - position);
23 +
24 +        boolean negative = false;
25 +        int length = 0;
26 +        while (length < limit) {
27 +            char c = text.charAt(position + length);
28 +            final boolean isFirstCharacterOperator = length == 0 && (c == '-' || c == '+');
29 +            if (isFirstCharacterOperator && iSigned) {
30 +                negative = c == '-';
31 +
32 +                if (isPastBoundaryOrNotDigit(text, position, limit, length)) {
33 +                    break;
34 +                }
35 +            }
```

Figure C.9: insert caption


```

36 +         if (negative) {
37 +             length++;
38 +         } else {
39 +             // Skip the '+' for parseInt to succeed.
40 +             position++;
41 +         }
42 +         // Expand the limit to disregard the sign character.
43 +         limit = Math.min(limit + 1, text.length() - position);
44 +         continue;
45 +     }
46 +     if (isNotADigit(c)) {
47 +         break;
48 +     }
49 +     length++;
50 + }
51 +
52 + if (length == 0) {
53 +     return ~position;
54 + }
55 +
56 + int value;
57 + if (length >= 9) {
58 +     // Since value may exceed integer limits, use stock parser
59 +     // which checks for this.
60 +     value = Integer.parseInt(text.subSequence(position, position += length).toString());
61 + } else {
62 +     int i = position;
63 +     if (negative) {
64 +         i++;
65 +     }
66 +     try {
67 +         value = text.charAt(i++) - '0';
68 +     } catch (StringIndexOutOfBoundsException e) {
69 +         return ~position;
70 +     }
71 +     position += length;
72 +     while (i < position) {
73 +         value = ((value << 3) + (value << 1)) + text.charAt(i++) - '0';
74 +     }
75 +     if (negative) {
76 +         value = -value;

```

Figure C.10: insert caption

```

77 +     }
78 + }
79 +
80 + bucket.saveField(iFieldType, value);
81 + return position;
82 + }
83 +
84 + private static boolean isPastBoundaryOrNotDigit(final CharSequence text, final int position, final int limit, final int length) {
85 +     final boolean isPastBoundary = length + 1 >= limit;
86 +     if (isPastBoundary) {
87 +         return true;
88 +     }
89 +     final char nextCharacter = text.charAt(position + length + 1);
90 +     final boolean isNotDigit = isNotADigit(nextCharacter);
91 +     return isNotDigit;
92 + }
93 +
94 + private static boolean isNotADigit(final char c) {
95 +     return c < '0' || c > '9';
96 + }
97 +}

```

Figure C.11: insert caption

The new class is 97 lines long, with the majority of the work happening in the 63

line long `parse` method. The result of `parse` can be tested and understood independently of details from the `NumberFormatter` inside of `DateTimeFormatterBuilder`.

C.6 Reducing Control Flow Complexity

Some might argue that a 63 line method is “simple enough.” Working heuristics of working memory, however, suggest that the human brain can fit between 4-9 “chunks” in memory at a particular time. This aligns well with the guidance in Clean Code that “Functions should do One Thing” and “Functions should be small”. Consequently, it’s worth turning attention to simplifying the `parse` method.

The cognitive complexity of the function is compounded by a variety of control structures. There are 2 while loops, 9 if statements, and 3 return statements. Extracting methods out of such a function is difficult because the multiple control flow breaks and updates to local variables require reasoning that is beyond many static analysis code tools, such as the Refactoring tools in IDEA IntelliJ. Often it helps to start “from the bottom up”, simplifying small blocks of complicated control structures until the tangled web of control flow clarifies.

First, a little cleanup. I can remove an unused import, as imports add to the cognitive load of a class by signaling interconnection with other components of a system.

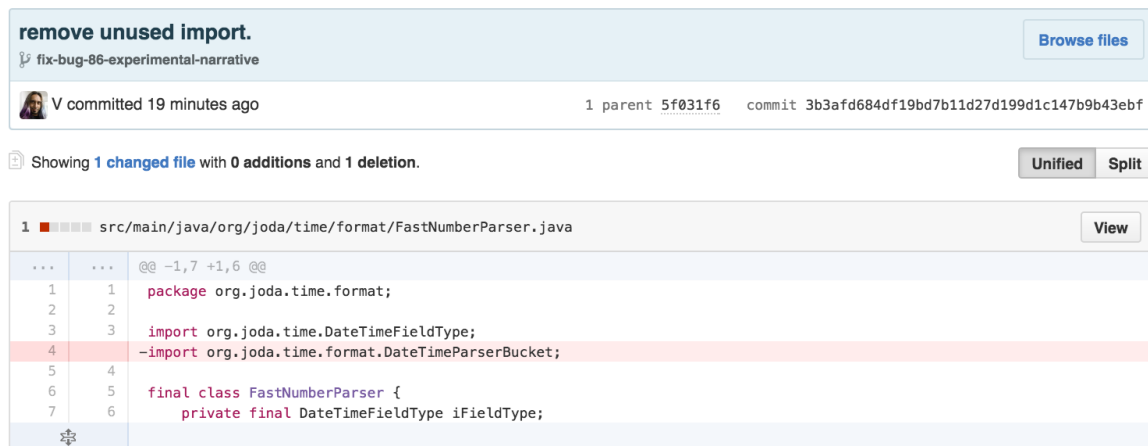


Figure C.12: insert caption

Next, I simplify according to the Principle of Least Astonishment by replacing the use of a `StringIndexOutOfBoundsException` for flow control (commonly argued in the literature) with a `GUARD CLAUSE`.

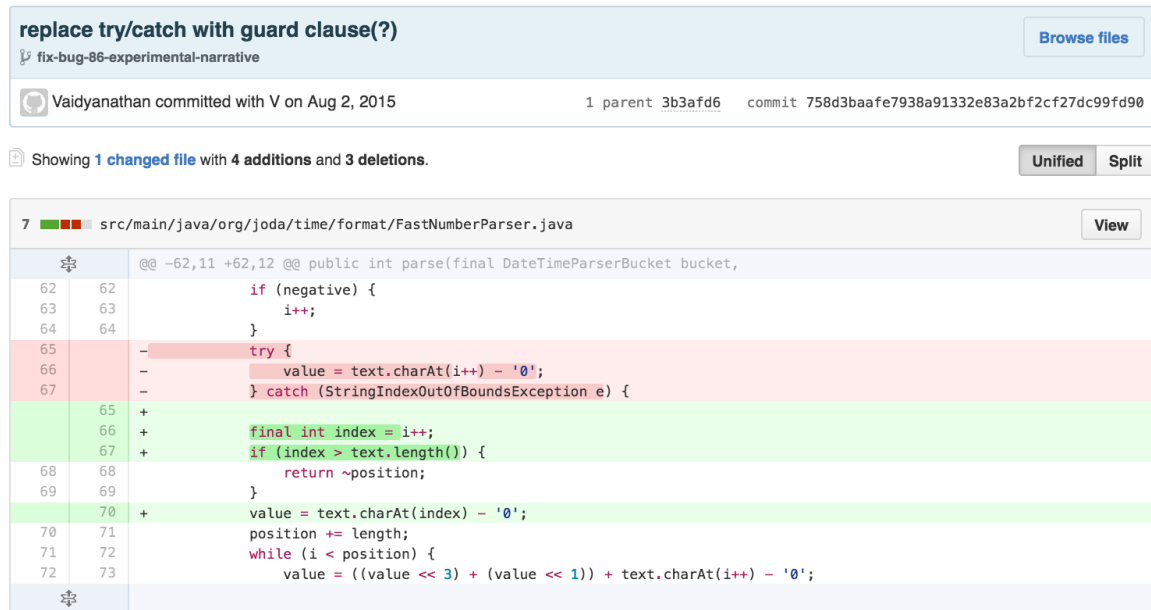


Figure C.13: insert caption

The body of `parse` has a strong visual indicator of separation between the first while loop, which updates the `position` and `length` variables, and a second “section” that calculates the value. In order to be able to break out methods effectively, I want to simplify the control flow of each.

Examining the body of the while loop, I see that it basically increments `length` every iteration. It breaks out of the loop if the current character is not a digit, and applies some special case updates if the first character is an operator and the parser knows the input contains one. I can simplify the special casing using explanatory variables and use REPLACE NESTED CONDITIONALS WITH GUARD CLAUSES to reduce the number of nested scopes. I also changed the previously extracted method `isPastBoundaryOrNotDigit` - -which by its very name does more than 1 thing- into 2 separate methods that are invoked by the caller.

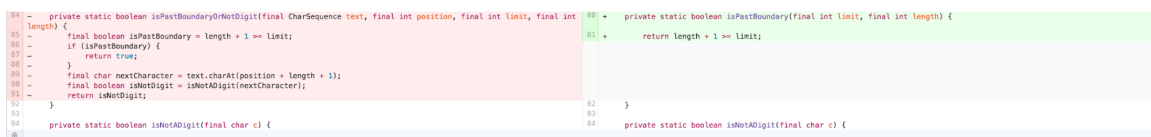


Figure C.14: insert caption

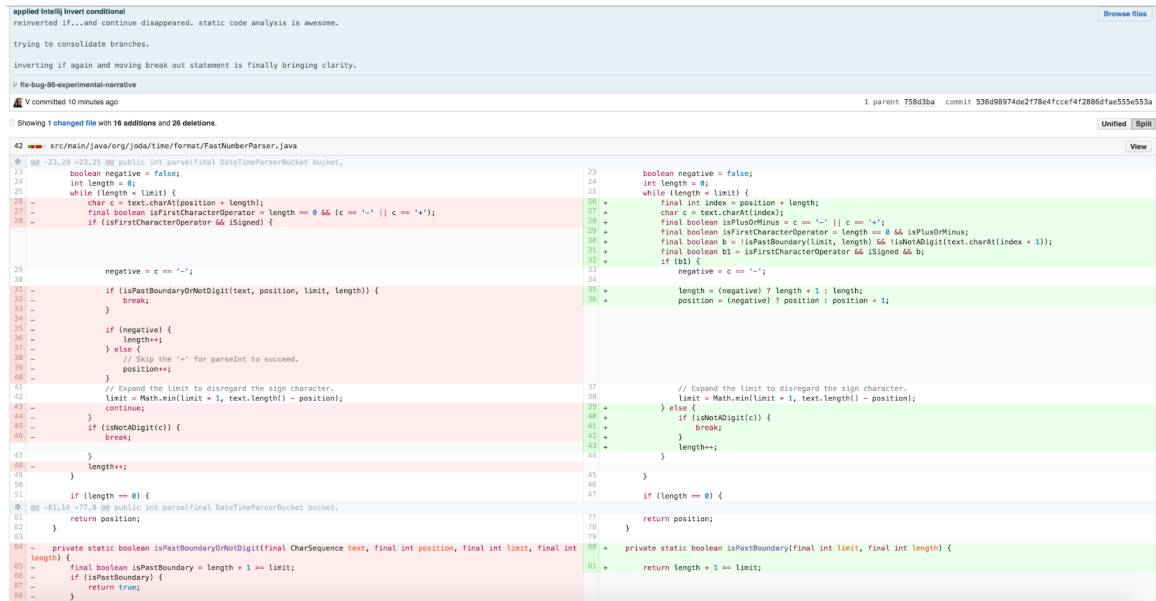


Figure C.15: insert caption

Seeking to further simplify the remaining **if/else** block inside the body of the **while**, I looked carefully at the special case logic for an operator and the **else** body and determined that both code paths incremented **length**. This type of conditional check to do the same thing in both cases is clearly extraneous cognitive load, so I applied a variant of CONSOLIDATE CONDITIONAL EXPRESSION.

inverting if again and moving break out statement is finally bringing...
... clarity.
[Browse files](#)

fix-bug-86-experimental-narrative

Vaidyanathan committed 1 parent 536d989 commit 936fd8c45e141775cfd5c47a2298d68ad83aa24f with V on Aug 2, 2015

Showing 1 changed file with 4 additions and 5 deletions.
[Unified](#) [Split](#)

9 src/main/java/org/joda/time/format/FastNumberParser.java [View](#)

29	29	final boolean isFirstCharacterOperator = length == 0 && isPlusOrMin
30	30	final boolean b = !isPastBoundary(limit, length) && !isNotADigit(te
31	31	final boolean b1 = isFirstCharacterOperator && isSigned && b;
	32 +	if (!b1 && isNotADigit(c)) {
	33 +	break;
	34 +	}
32	35	if (b1) {
33	36	negative = c == '-';
34	37	
35	38	length = (negative) ? length + 1 : length;
36	39	position = (negative) ? position : position + 1;
37	40	// Expand the limit to disregard the sign character.
38	41	limit = Math.min(limit + 1, text.length() - position);
39	-	} else {
40	-	if (isNotADigit(c)) {
41	-	break;
42	-	}
43	-	length++;
44	42	}
	43 +	length = length + 1;
45	44	}
46	45	
47	46	if (length == 0) {

Figure C.16: insert caption

I then had an opportunity to simplify the while loop further by applying a similar procedure to a variant of REMOVE CONTROL FLAG to remove the break statement and clarify the loop predicate.

Applied variant of REMOVE CONTROL FLAG using a method and INLINE TEMP...
... VARIABLE to condense iteration logic

fix-bug-86-experimental-narrative
Vaidyanathan committed with V on Aug 2, 2015
1 parent 936f8bc commit b0bc2ce3dc05d35ae818ba841907528bb7e45438

Showing 1 changed file with 14 additions and 13 deletions.

Unified Split

View

```
27 src/main/java/org/joda/time/format/FastNumberParser.java
28 @@ -22,19 +22,9 @@ public int parse(final DateTimeParserBucket bucket,
29     boolean negative = false;
30     int length = 0;
31     while (length < limit) {
32         final int index = position + length;
33         char c = text.charAt(index);
34         final boolean isPlusOrMinus = c == '-' || c == '+';
35         final boolean isFirstCharacterOperator = length == 0 && isPlusOrMinus;
36         final boolean b = isPastBoundary(limit, length) && !isNotADigit(text.charAt(index + 1));
37         final boolean bl = isFirstCharacterOperator && !signed && b;
38         if (!bl && !isNotADigit(c)) {
39             break;
40         }
41         if (bl) {
42             negative = c == '-';
43         }
44         length = (negative) ? length + 1 : length;
45         position = (negative) ? position : position + 1;
46         // Expand the limit to disregard the sign character.
47     }
48     @@ -76,18 +66,21 @@ public int parse(final DateTimeParserBucket bucket,
49     return position;
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
```

Figure C.17: insert caption

With this logic greatly simplified, I'm now able to push the calculation into a smaller, more cohesive chunk of logic using REPLACE METHOD WITH METHOD OBJECT.

applied <https://sourcemaking.com/refactoring/replace-method-with-meth...>
...od-object to simplify shared state into OffsetCalculator.

fix-bug-86-experimental-narrative
Vaidyanathan committed with V on Aug 2, 2015
1 parent b0bc2ce commit 5c88115b6774fb1d735136b71c544585dc381a

Showing 1 changed file with 69 additions and 29 deletions.

Unified Split

View

```
98 src/main/java/org/joda/time/format/FastNumberParser.java
99 @@ -18,28 +18,11 @@ public FastNumberParser(final int maxParsedDigits,
100     public int parse(final DateTimeParserBucket bucket,
101         final CharSequence text,
102         int position) {
103     int limit = Math.min(maxParsedDigits, text.length() - position);
104     boolean negative = false;
105     int length = 0;
106     while (length < limit && !isADigit(text.charAt(position + length)) || prefixedWithPlusOrMinus(text, position,
107     limit, length)) {
108         if (prefixedWithPlusOrMinus(text, position, limit, length)) {
109             negative = text.charAt(position + length) == '-';
110             length = (negative) ? length + 1 : length;
111             position = (negative) ? position : position + 1;
112             // Expand the limit to disregard the sign character.
113             limit = Math.min(limit + 1, text.length() - position);
114         }
115         length = length + 1;
116     }
117     if (length == 0) {
118         return -position;
119     }
120     @@ -44,7 +35,7 @@ public int parse(final DateTimeParserBucket bucket,
121     value = Integer.parseInt(text.subSequence(position, position + length).toString());
122     } else {
123         int i = position;
124         if (negative) {
125             i++;
126         }
127     }
128     while (i < position) {
129         value = (value << 3) + (value << 1) + text.charAt(i++) - '0';
130     }
131     if (negative) {
132         value = -value;
133     }
134 }
```

Figure C.18: insert caption

66	return position;	57	return position;
67	}	58	}
68		59	
69	- private boolean prefixedWithPlusOrMinus(final CharSequence text, final int position, final int limit, final int	60	+ private static class OffsetCalculator {
70	length) {	61	+ private final CharSequence text;
71	- final int index = position + length;	62	+ private int position;
72	- final boolean isFirstCharacterOperator = length == 0 && (text.charAt(index) == '-' text.charAt(index) ==	63	+ private int length;
73	='+');	64	+ private boolean negative;
74	- final boolean isNextCharacterADigit = index < text.length() - 1 && isADigit(text.charAt(index + 1));	65	+ private int limit;
75	- return isFirstCharacterOperator && isSigned && isPastBoundary(limit, length) && isNextCharacterADigit;	66	
76	- private static boolean isPastBoundary(final int limit, final int length) {	67	+ public OffsetCalculator(final CharSequence text,
77	- return length + 1 >= limit;	68	+ final int position) {
78	- }	69	+ this.text = text;
79		70	+ this.position = position;
80		71	+ length = 0;
81		72	+ negative = false;
82		73	+ }
83		74	
84	- private static boolean isADigit(final char c) {	75	+ int getPosition() {
85	- return !isNotADigit(c);	76	+ return position;
86		77	+ }
87		78	
88	- private static boolean isNotADigit(final char c) {	79	+ int getLength() {
89	- return c < '0' c > '9';	80	+ return length;
90		81	+ }
91		82	
92		83	+ boolean isNegative() {
93		84	+ return negative;
94		85	+ }
95		86	
96		87	+ public OffsetCalculator invokeOf(final int @MaxParamDigits, final boolean isSigned) {
97		88	+ limit = Math.min(@MaxParamDigits, text.length() - position);
98		89	+ while (length < limit && isADigit(text.charAt(position + length)))
99		90	+ isPrefixedWithPlusOrMinus() && isSigned) {
100		91	+ updateBasedOnSign();
101		92	+ length = length + 1;
102		93	+ }
103		94	+ return this;
104		95	+ }
105		96	
106		97	+ private void updateBasedOnSign() {
107		98	+ if (isPrefixedWithPlusOrMinus()) {
108		99	+ negative = text.charAt(position + length) == '-';
109		100	+ length = (negative ? length + 1 : length;
110		101	+ position = (negative ? position : position + 1;
111		102	+ // Expand the limit to disregard the sign character.
112		103	+ limit = Math.min(limit + 1, text.length() - position);
113		104	+ }
114		105	+ }

Figure C.19: insert caption

106		105	+ }
107		106	+ }
108		107	+ private boolean isPrefixedWithPlusOrMinus() {
109		108	+ final int index = position + length;
110		109	+ final char currentCharacter = text.charAt(index);
111		110	+ final boolean isFirstCharacterOperator = length == 0 && (currentCharacter == '-' currentCharacter ==
112		111	='+');
113		112	+ final boolean hasNextDigitCharacter = index < text.length() - 1 && isADigit(text.charAt(index + 1));
114		113	+ return isFirstCharacterOperator && isBeforeBoundary() && hasNextDigitCharacter;
115		114	+ }
116		115	
117		116	+ private boolean isBeforeBoundary() {
118		117	+ return length + 1 <= limit;
119		118	+ }
120		119	
121		120	+ private static boolean isADigit(final char c) {
122		121	+ return !isNotADigit(c);
123		122	+ }
124		123	
125		124	+ private static boolean isNotADigit(final char c) {
126		125	+ return c < '0' c > '9';
127		126	+ }
128		127	+ }

Figure C.20: insert caption

Next I wanted to ensure that the new class was inline with CLT’s guidance on sequencing content effectively for comprehension, in line with the recommendations of Clean Code’s STEPDOWN RULE/The Newspaper Metaphor.

re-arrange methods according to Newspaper Metaphor.
Browse files

fix-bug-86-experimental-narrative

Vaidyanathan committed with V on Nov 28, 2015
1 parent 5c08115
commit 34a168f7983bae53b5dcc5783147e5249d8237f6

Showing 1 changed file with 14 additions and 14 deletions.
Unified Split

28 src/main/java/org/joda/time/format/FastNumberParser.java View

```

@@ -94,14 +94,12 @@ public OffsetCalculator invoke(final int iMaxParsedDigits, final boolean iSigned
    94      94      return this;
    95      95      }
    96      96
    97  - private void updateBasedOnSign() {
    98  -     if (isPrefixedWithPlusOrMinus()) {
    99  -         negative = text.charAt(position + length) == '-';
   100  -         length = (negative) ? length + 1 : length;
   101  -         position = (negative) ? position : position + 1;
   102  -         // Expand the limit to disregard the sign character.
   103  -         limit = Math.min(limit + 1, text.length() - position);
   104  -     }
   105  + private static boolean isADigit(final char c) {
   106  +     return !isNotADigit(c);
   107  +     }
   108  + private static boolean isNotADigit(final char c) {
   109  +     return c < '0' || c > '9';
   110  +     }
   111  + private boolean isPrefixedWithPlusOrMinus() {
   112  +     return length + 1 <= limit;
   113  +     }
   114  + private boolean isBeforeBoundary() {
   115  +     return length + 1 <= limit;
   116  +     }
   117  - private static boolean isADigit(final char c) {
   118  -     return !isNotADigit(c);
   119  -     }
   120  - private static boolean isNotADigit(final char c) {
   121  -     return c < '0' || c > '9';
   122  -     }
   123  + private void updateBasedOnSign() {
   124  +     if (isPrefixedWithPlusOrMinus()) {
   125  +         negative = text.charAt(position + length) == '-';
   126  +         length = (negative) ? length + 1 : length;
   127  +         position = (negative) ? position : position + 1;
   128  +         // Expand the limit to disregard the sign character.
   129  +         limit = Math.min(limit + 1, text.length() - position);
   130  +     }
   131  +     }
   132  +     }
   133  +     }
   134  +     }
   135  +     }
   136  +     }
   137  +     }
   138  +     }
   139  +     }
   140  +     }
   141  +     }
   142  +     }
   143  +     }
   144  +     }
   145  +     }
   146  +     }
   147  +     }
   148  +     }
   149  +     }
   150  +     }
   151  +     }
   152  +     }
   153  +     }
   154  +     }
   155  +     }
   156  +     }
   157  +     }
   158  +     }
   159  +     }
   160  +     }
   161  +     }
   162  +     }
   163  +     }
   164  +     }
   165  +     }
   166  +     }
   167  +     }
   168  +     }
   169  +     }
   170  +     }
   171  +     }
   172  +     }
   173  +     }
   174  +     }
   175  +     }
   176  +     }
   177  +     }
   178  +     }
   179  +     }
   180  +     }
   181  +     }
   182  +     }
   183  +     }
   184  +     }
   185  +     }
   186  +     }
   187  +     }
   188  +     }
   189  +     }
   190  +     }
   191  +     }
   192  +     }
   193  +     }
   194  +     }
   195  +     }
   196  +     }
   197  +     }
   198  +     }
   199  +     }
   200  +     }
   201  +     }
   202  +     }
   203  +     }
   204  +     }
   205  +     }
   206  +     }
   207  +     }
   208  +     }
   209  +     }
   210  +     }
   211  +     }
   212  +     }
   213  +     }
   214  +     }
   215  +     }
   216  +     }
   217  +     }
   218  +     }
   219  +     }
   220  +     }
   221  +     }
   222  +     }
   223  +     }
   224  +     }
   225  +     }
   226  +     }
   227  +     }
   228  +     }
   229  +     }
   230  +     }
   231  +     }
   232  +     }
   233  +     }
   234  +     }
   235  +     }
   236  +     }
   237  +     }
   238  +     }
   239  +     }
   240  +     }
   241  +     }
   242  +     }
   243  +     }
   244  +     }
   245  +     }
   246  +     }
   247  +     }
   248  +     }
   249  +     }
   250  +     }
   251  +     }
   252  +     }
   253  +     }
   254  +     }
   255  +     }
   256  +     }
   257  +     }
   258  +     }
   259  +     }
   260  +     }
   261  +     }
   262  +     }
   263  +     }
   264  +     }
   265  +     }
   266  +     }
   267  +     }
   268  +     }
   269  +     }
   270  +     }
   271  +     }
   272  +     }
   273  +     }
   274  +     }
   275  +     }
   276  +     }
   277  +     }
   278  +     }
   279  +     }
   280  +     }
   281  +     }
   282  +     }
   283  +     }
   284  +     }
   285  +     }
   286  +     }
   287  +     }
   288  +     }
   289  +     }
   290  +     }
   291  +     }
   292  +     }
   293  +     }
   294  +     }
   295  +     }
   296  +     }
   297  +     }
   298  +     }
   299  +     }
   300  +     }
   301  +     }
   302  +     }
   303  +     }
   304  +     }
   305  +     }
   306  +     }
   307  +     }
   308  +     }
   309  +     }
   310  +     }
   311  +     }
   312  +     }
   313  +     }
   314  +     }
   315  +     }
   316  +     }
   317  +     }
   318  +     }
   319  +     }
   320  +     }
   321  +     }
   322  +     }
   323  +     }
   324  +     }
   325  +     }
   326  +     }
   327  +     }
   328  +     }
   329  +     }
   330  +     }
   331  +     }
   332  +     }
   333  +     }
   334  +     }
   335  +     }
   336  +     }
   337  +     }
   338  +     }
   339  +     }
   340  +     }
   341  +     }
   342  +     }
   343  +     }
   344  +     }
   345  +     }
   346  +     }
   347  +     }
   348  +     }
   349  +     }
   350  +     }
   351  +     }
   352  +     }
   353  +     }
   354  +     }
   355  +     }
   356  +     }
   357  +     }
   358  +     }
   359  +     }
   360  +     }
   361  +     }
   362  +     }
   363  +     }
   364  +     }
   365  +     }
   366  +     }
   367  +     }
   368  +     }
   369  +     }
   370  +     }
   371  +     }
   372  +     }
   373  +     }
   374  +     }
   375  +     }
   376  +     }
   377  +     }
   378  +     }
   379  +     }
   380  +     }
   381  +     }
   382  +     }
   383  +     }
   384  +     }
   385  +     }
   386  +     }
   387  +     }
   388  +     }
   389  +     }
   390  +     }
   391  +     }
   392  +     }
   393  +     }
   394  +     }
   395  +     }
   396  +     }
   397  +     }
   398  +     }
   399  +     }
   400  +     }
   401  +     }
   402  +     }
   403  +     }
   404  +     }
   405  +     }
   406  +     }
   407  +     }
   408  +     }
   409  +     }
   410  +     }
   411  +     }
   412  +     }
   413  +     }
   414  +     }
   415  +     }
   416  +     }
   417  +     }
   418  +     }
   419  +     }
   420  +     }
   421  +     }
   422  +     }
   423  +     }
   424  +     }
   425  +     }
   426  +     }
   427  +     }
   428  +     }
   429  +     }
   430  +     }
   431  +     }
   432  +     }
   433  +     }
   434  +     }
   435  +     }
   436  +     }
   437  +     }
   438  +     }
   439  +     }
   440  +     }
   441  +     }
   442  +     }
   443  +     }
   444  +     }
   445  +     }
   446  +     }
   447  +     }
   448  +     }
   449  +     }
   450  +     }
   451  +     }
   452  +     }
   453  +     }
   454  +     }
   455  +     }
   456  +     }
   457  +     }
   458  +     }
   459  +     }
   460  +     }
   461  +     }
   462  +     }
   463  +     }
   464  +     }
   465  +     }
   466  +     }
   467  +     }
   468  +     }
   469  +     }
   470  +     }
   471  +     }
   472  +     }
   473  +     }
   474  +     }
   475  +     }
   476  +     }
   477  +     }
   478  +     }
   479  +     }
   480  +     }
   481  +     }
   482  +     }
   483  +     }
   484  +     }
   485  +     }
   486  +     }
   487  +     }
   488  +     }
   489  +     }
   490  +     }
   491  +     }
   492  +     }
   493  +     }
   494  +     }
   495  +     }
   496  +     }
   497  +     }
   498  +     }
   499  +     }
   500  +     }
   501  +     }
   502  +     }
   503  +     }
   504  +     }
   505  +     }
   506  +     }
   507  +     }
   508  +     }
   509  +     }
   510  +     }
   511  +     }
   512  +     }
   513  +     }
   514  +     }
   515  +     }
   516  +     }
   517  +     }
   518  +     }
   519  +     }
   520  +     }
   521  +     }
   522  +     }
   523  +     }
   524  +     }
   525  +     }
   526  +     }
   527  +     }
   528  +     }
   529  +     }
   530  +     }
   531  +     }
   532  +     }
   533  +     }
   534  +     }
   535  +     }
   536  +     }
   537  +     }
   538  +     }
   539  +     }
   540  +     }
   541  +     }
   542  +     }
   543  +     }
   544  +     }
   545  +     }
   546  +     }
   547  +     }
   548  +     }
   549  +     }
   550  +     }
   551  +     }
   552  +     }
   553  +     }
   554  +     }
   555  +     }
   556  +     }
   557  +     }
   558  +     }
   559  +     }
   560  +     }
   561  +     }
   562  +     }
   563  +     }
   564  +     }
   565  +     }
   566  +     }
   567  +     }
   568  +     }
   569  +     }
   570  +     }
   571  +     }
   572  +     }
   573  +     }
   574  +     }
   575  +     }
   576  +     }
   577  +     }
   578  +     }
   579  +     }
   580  +     }
   581  +     }
   582  +     }
   583  +     }
   584  +     }
   585  +     }
   586  +     }
   587  +     }
   588  +     }
   589  +     }
   590  +     }
   591  +     }
   592  +     }
   593  +     }
   594  +     }
   595  +     }
   596  +     }
   597  +     }
   598  +     }
   599  +     }
   600  +     }
   601  +     }
   602  +     }
   603  +     }
   604  +     }
   605  +     }
   606  +     }
   607  +     }
   608  +     }
   609  +     }
   610  +     }
   611  +     }
   612  +     }
   613  +     }
   614  +     }
   615  +     }
   616  +     }
   617  +     }
   618  +     }
   619  +     }
   620  +     }
   621  +     }
   622  +     }
   623  +     }
   624  +     }
   625  +     }
   626  +     }
   627  +     }
   628  +     }
   629  +     }
   630  +     }
   631  +     }
   632  +     }
   633  +     }
   634  +     }
   635  +     }
   636  +     }
   637  +     }
   638  +     }
   639  +     }
   640  +     }
   641  +     }
   642  +     }
   643  +     }
   644  +     }
   645  +     }
   646  +     }
   647  +     }
   648  +     }
   649  +     }
   650  +     }
   651  +     }
   652  +     }
   653  +     }
   654  +     }
   655  +     }
   656  +     }
   657  +     }
   658  +     }
   659  +     }
   660  +     }
   661  +     }
   662  +     }
   663  +     }
   664  +     }
   665  +     }
   666  +     }
   667  +     }
   668  +     }
   669  +     }
   670  +     }
   671  +     }
   672  +     }
   673  +     }
   674  +     }
   675  +     }
   676  +     }
   677  +     }
   678  +     }
   679  +     }
   680  +     }
   681  +     }
   682  +     }
   683  +     }
   684  +     }
   685  +     }
   686  +     }
   687  +     }
   688  +     }
   689  +     }
   690  +     }
   691  +     }
   692  +     }
   693  +     }
   694  +     }
   695  +     }
   696  +     }
   697  +     }
   698  +     }
   699  +     }
   700  +     }
   701  +     }
   702  +     }
   703  +     }
   704  +     }
   705  +     }
   706  +     }
   707  +     }
   708  +     }
   709  +     }
   710  +     }
   711  +     }
   712  +     }
   713  +     }
   714  +     }
   715  +     }
   716  +     }
   717  +     }
   718  +     }
   719  +     }
   720  +     }
   721  +     }
   722  +     }
   723  +     }
   724  +     }
   725  +     }
   726  +     }
   727  +     }
   728  +     }
   729  +     }
   730  +     }
   731  +     }
   732  +     }
   733  +     }
   734  +     }
   735  +     }
   736  +     }
   737  +     }
   738  +     }
   739  +     }
   740  +     }
   741  +     }
   742  +     }
   743  +     }
   744  +     }
   745  +     }
   746  +     }
   747  +     }
   748  +     }
   749  +     }
   750  +     }
   751  +     }
   752  +     }
   753  +     }
   754  +     }
   755  +     }
   756  +     }
   757  +     }
   758  +     }
   759  +     }
   760  +     }
   761  +     }
   762  +     }
   763  +     }
   764  +     }
   765  +     }
   766  +     }
   767  +     }
   768  +     }
   769  +     }
   770  +     }
   771  +     }
   772  +     }
   773  +     }
   774  +     }
   775  +     }
   776  +     }
   777  +     }
   778  +     }
   779  +     }
   780  +     }
   781  +     }
   782  +     }
   783  +     }
   784  +     }
   785  +     }
   786  +     }
   787  +     }
   788  +     }
   789  +     }
   790  +     }
   791  +     }
   792  +     }
   793  +     }
   794  +     }
   795  +     }
   796  +     }
   797  +     }
   798  +     }
   799  +     }
   800  +     }
   801  +     }
   802  +     }
   803  +     }
   804  +     }
   805  +     }
   806  +     }
   807  +     }
   808  +     }
   809  +     }
   810  +     }
   811  +     }
   812  +     }
   813  +     }
   814  +     }
   815  +     }
   816  +     }
   817  +     }
   818  +     }
   819  +     }
   820  +     }
   821  +     }
   822  +     }
   823  +     }
   824  +     }
   825  +     }
   826  +     }
   827  +     }
   828  +     }
   829  +     }
   830  +     }
   831  +     }
   832  +     }
   833  +     }
   834  +     }
   835  +     }
   836  +     }
   837  +     }
   838  +     }
   839  +     }
   840  +     }
   841  +     }
   842  +     }
   843  +     }
   844  +     }
   845  +     }
   846  +     }
   847  +     }
   848  +     }
   849  +     }
   850  +     }
   851  +     }
   852  +     }
   853  +     }
   854  +     }
   855  +     }
   856  +     }
   857  +     }
   858  +     }
   859  +     }
   860  +     }
   861  +     }
   862  +     }
   863  +     }
   864  +     }
   865  +     }
   866  +     }
   867  +     }
   868  +     }
   869  +     }
   870  +     }
   871  +     }
   872  +     }
   873  +     }
   874  +     }
   875  +     }
   876  +     }
   877  +     }
   878  +     }
   879  +     }
   880  +     }
   881  +     }
   882  +     }
   883  +     }
   884  +     }
   885  +     }
   886  +     }
   887  +     }
   888  +     }
   889  +     }
   890  +     }
   891  +     }
   892  +     }
   893  +     }
   894  +     }
   895  +     }
   896  +     }
   897  +     }
   898  +     }
   899  +     }
   900  +     }
   901  +     }
   902  +     }
   903  +     }
   904  +     }
   905  +     }
   906  +     }
   907  +     }
   908  +     }
   909  +     }
   910  +     }
   911  +     }
   912  +     }
   913  +     }
   914  +     }
   915  +     }
   916  +     }
   917  +     }
   918  +     }
   919  +     }
   920  +     }
   921  +     }
   922  +     }
   923  +     }
   924  +     }
   925  +     }
   926  +     }
   927  +     }
   928  +     }
   929  +     }
   930  +     }
   931  +     }
   932  +     }
   933  +     }
   934  +     }
   935  +     }
   936  +     }
   937  +     }
   938  +     }
   939  +     }
   940  +     }
   941  +     }
   942  +     }
   943  +     }
   944  +     }
   945  +     }
   946  +     }
   947  +     }
   948  +     }
   949  +     }
   950  +     }
   951  +     }
   952  +     }
   953  +     }
   954  +     }
   955  +     }
   956  +     }
   957  +     }
   958  +     }
   959  +     }
   960  +     }
   961  +     }
   962  +     }
   963  +     }
   964  +     }
   965  +     }
   966  +     }
   967  +     }
   968  +     }
   969  +     }
   970  +     }
   971  +     }
   972  +     }
   973  +     }
   974  +     }
   975  +     }
   976  +     }
   977  +     }
   978  +     }
   979  +     }
   980  +     }
   981  +     }
   982  +     }
   983  +     }
   984  +     }
   985  +     }
   986  +     }
   987  +     }
   988  +     }
   989  +     }
   990  +     }
   991  +     }
   992  +     }
   993  +     }
   994  +     }
   995  +     }
   996  +     }
   997  +     }
   998  +     }
   999  +     }
   1000  +     }
   1001  +     }
   1002  +     }
   1003  +     }
   1004  +     }
   1005  +     }
   1006  +     }
   1007  +     }
   1008  +     }
   1009  +     }
   1010  +     }
   1011  +     }
   1012  +     }
   1013  +     }
   1014  +     }
   1015  +     }
   1016  +     }
   1017  +     }
   1018  +     }
   1019  +     }
   1020  +     }
   1021  +     }
   1022  +     }
   1023  +     }
   1024  +     }
   1025  +     }
   1026  +     }
   1027  +     }
   1028  +     }
   1029  +     }
   1030  +     }
   1031  +     }
   1032  +     }
   1033  +     }
   1034  +     }
   1035  +     }
   1036  +     }
   1037  +     }
   1038  +     }
   1039  +     }
   1040  +     }
   1041  +     }
   1042  +     }
   1043  +     }
   1044  +     }
   1045  +     }
   1046  +     }
   1047  +     }
   1048  +     }
   1049  +     }
   1050  +     }
   1051  +     }
   1052  +     }
   1053  +     }
   1054  +     }
   1055  +     }
   1056  +     }
   1057  +     }
   1058  +     }
   1059  +     }
   1060  +     }
   1061  +     }
   1062  +     }
   1063  +     }
   1064  +     }
   1065  +     }
   1066  +     }
   1067  +     }
   1068  +     }
   1069  +     }
   1070  +     }
   1071  +     }
   1072  +     }
   1073  +     }
   1074  +     }
   1075  +     }
   1076  +     }
   1077  +     }
   1078  +     }
   1079  +     }
   1080  +     }
   1081  +     }
   1082  +     }
   1083  +     }
   1084  +     }
   1085  +     }
   1086  +     }
   1087  +     }
   1088  +     }
   1089  +     }
   1090  +     }
   1091  +     }
   1092  +     }
   1093  +     }
   1094  +     }
   1095  +     }
   1096  +     }
   1097  +     }
   1098  +     }
   1099  +     }
   1100  +     }
   1101  +     }
   1102  +     }
   1103  +     }
   1104  +     }
   1105  +     }
   1106  +     }
   1107  +     }
   1108  +     }
   1109  +     }
   1110  +     }
   1111  +     }
   1112  +     }
   1113  +     }
   1114  +     }
   1115  +     }
   1116  +     }
   1117  +     }
   1118  +     }
   1119  +     }
   1120  +     }
   1121  +     }
   1122  +     }
   1123  +     }
   1124  +     }
   1125  +     }
   1126  +     }
   1127  +     }
   1128  +     }
   1129  +     }
   1130  +     }
   1131  +     }
   1132  +     }
   1133  +     }
   1134  +     }
   1135  +     }
   1136  +     }
   1137  +     }
   1138  +     }
   1139  +     }
   1140  +     }
   1141  +     }
   1142  +     }
   1143  +     }
   1144  +     }
   1145  +     }
   1146  +     }
   1147  +     }
   1148  +     }
   1149  +     }
   1150  +     }
   1151  +     }
   1152  +     }
   1153  +     }
   1154  +     }
   1155  +     }
   1156  +     }
   1157  +     }
   1158  +     }
   1159  +     }
   1160  +     }
   1161  +     }
   1162  +     }
   1163  +     }
   1164  +     }
   1165  +     }
   1166  +     }
   1167  +     }
   1168  +     }
   1169  +     }
   1170  +     }
   1171  +     }
   1172  +     }
   1173  +     }
   1174  +     }
   1175  +     }
   1176  +     }
   1177  +     }
   1178  +     }
   1179  +     }
   1180  +     }
   1181  +     }
   1182  +     }
   1183  +     }
   1184  +     }
   1185  +     }
   1186  +     }
   1187  +     }
   1188  +     }
   1189  +     }
   1190  +     }
   1191  +     }
   1192  +     }
   1193  +     }
   1194  +     }
   1195  +     }
   1196  +     }
   1197  +     }
   1198  +     }
   1199  +     }
   1200  +     }
   1201  +     }
   1202  +     }
   1203  +     }
   1204  +     }
   1205  +     }
   1206  +     }
   1207  +     }
   1208  +     }
   1209  +     }
   1210  +     }
   1211  +     }
   1212  +     }
   1213  +     }
   1214  +     }
   1215  +     }
   1216  +     }
   1217  +     }
   1218  +     }
   1219  +     }
   1220  +     }
   1221  +     }
   1222  +     }
   1223  +     }
   1224  +     }
   1225  +     }
   1226  +     }
   1227  +     }
   1228  +     }
   1229  +     }
   1230  +     }
   1231  +     }
   1232  +     }
   1233  +     }
   1234  +     }
   1235  +     }
   1236  +     }
   1237  +     }
   1238  +     }
   1239  +     }
   1240  +     }
   1241  +     }
   1242  +     }
   1243  +     }
   1244  +     }
   1245  +     }
   1246  +     }
   1247  +     }
   1248  +     }
   1249  +     }
   1250  +     }
   1251  +     }
   1252  +     }
   1253  +     }
   1254  +     }
   1255  +     }
   1256  +     }

```

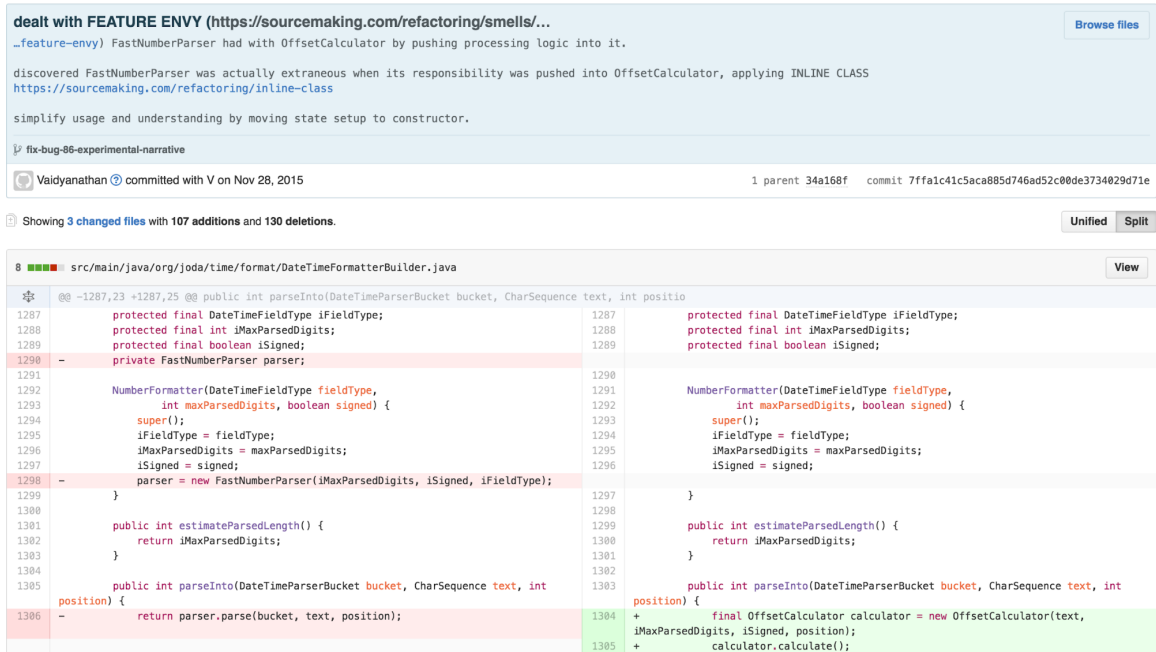



Figure C.22: insert caption

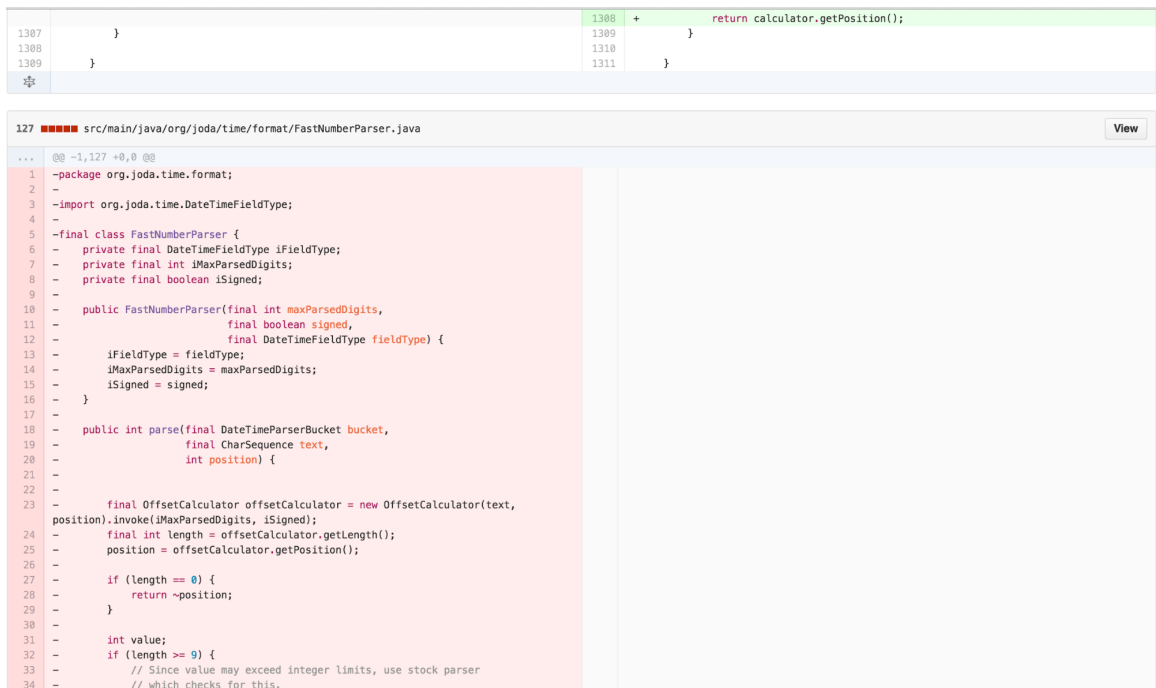


Figure C.23: insert caption

```

35 -         value = Integer.parseInt(text.subSequence(position,
36 - position += length).toString());
37 -     } else {
38 -         int i = position;
39 -         if (offsetCalculator.isNegative()) {
40 -             i++;
41 -         }
42 +         final int index = i++;
43 -         if (index > text.length()) {
44 -             return ~position;
45 -         }
46 -         value = text.charAt(index) - '0';
47 -         position += length;
48 -         while (i < position) {
49 -             value = ((value << 3) + (value << 1)) +
text.charAt(i++) - '0';
50 -         }
51 -         if (offsetCalculator.isNegative()) {
52 -             value = -value;
53 -         }
54 -     }
55 -
56 -     bucket.saveField(iFieldType, value);
57 -     return position;
58 - }
59 -
60 - private static class OffsetCalculator {
61 -     private final CharSequence text;
62 -     private int position;

```

Figure C.24: insert caption

```

63 -     private int length;
64 -     private boolean negative;
65 -     private int limit;
66 -
67 -     public OffsetCalculator(final CharSequence text,
68 -                             final int position) {
69 -         this.text = text;
70 -         this.position = position;
71 -         length = 0;
72 -         negative = false;
73 -     }
74 -
75 -     int getPosition() {
76 -         return position;
77 -     }
78 -
79 -     int getLength() {
80 -         return length;
81 -     }
82 -
83 -     boolean isNegative() {
84 -         return negative;
85 -     }
86 -
87 -     public OffsetCalculator invoke(final int
iMaxParsedDigits, final boolean iSigned) {
88 -         limit = Math.min(iMaxParsedDigits, text.length() -
position);
89 -         while (length < limit &&
(isADigit(text.charAt(position + length))

```

Figure C.25: insert caption

```

90 -         || isPrefixedWithPlusOrMinus() && isSigned)) {
91 -             updateBasedOnSign();
92 -             length = length + 1;
93 -         }
94 -         return this;
95 -     }
96 -
97 -     private static boolean isADigit(final char c) {
98 -         return !isNotADigit(c);
99 -     }
100 -
101 -     private static boolean isNotADigit(final char c) {
102 -         return c < '0' || c > '9';
103 -     }
104 -
105 -     private boolean isPrefixedWithPlusOrMinus() {
106 -         final int index = position + length;
107 -         final char currentCharacter = text.charAt(index);
108 -         final boolean isFirstCharacterOperator = length == 0 && (currentCharacter
109 - == '-' || currentCharacter == '+');
110 -         final boolean hasNextDigitCharacter = index < text.length() - 1 &&
111 - isADigit(text.charAt(index + 1));
112 -         return isFirstCharacterOperator && isBeforeBoundary() &&
113 - hasNextDigitCharacter;
114 -     }
115 -
116 -     private boolean isBeforeBoundary() {
117 -         return length + 1 <= limit;
118 -     }
119 -
120 -     private void updateBasedOnSign() {
121 -         if (isPrefixedWithPlusOrMinus()) {
122 -             negative = text.charAt(position + length) == '-';
123 -             length = (negative) ? length + 1 : length;
124 -             position = (negative) ? position : position + 1;
125 -             // Expand the limit to disregard the sign character.
126 -             limit = Math.min(limit + 1, text.length() - position);
127 -         }
128 -     }
129 - }
130 - ~)

```

Figure C.26: insert caption

```

102 src/main/java/org/joda/time/format/OffsetCalculator.java
... @@ -0,0 +1,102 @@
1
+package org.joda.time.format;
2
+final class OffsetCalculator {
3
+    private final CharSequence text;
4
+    private final int maxParsedDigits;
5
+    private final boolean isSigned;
6
+    private int position;
7
+    private int length;
8
+    private boolean negative;
9
+    private int limit;
10
+    private int value;
11
+
12
+    OffsetCalculator(final CharSequence text,
13
+                    final int iMaxParsedDigits,
14
+                    final boolean isSigned,
15
+                    final int position) {
16
+        this.text = text;
17
+        this.position = position;
18
+        length = 0;
19
+        negative = false;
20
+        maxParsedDigits = iMaxParsedDigits;
21
+        isSigned = isSigned;
22
+    }
23
+
24
+    int getPosition() {
25
+        return position;
26
+    }
27
+
28
+    int getValue() {
29
+        return value;
30
+    }
31
+
32
+    void calculate() {
33
+        limit = Math.min(maxParsedDigits, text.length() - position);
34
+        while (length < limit && (isADigit(text.charAt(position + length))
35
+ || isPrefixedWithPlusOrMinus() && isSigned)) {
36
+            updateBasedOnSign();
37
+            length = length + 1;
38
+        }
39
+    }
40
+
41
+    if (length == 0) {
42
+        position = ~position;

```

Figure C.27: insert caption

```

43 +         return;
44 +     }
45 +
46 +     if (length >= 9) {
47 +         // Since value may exceed integer limits, use stock parser
48 +         // which checks for this.
49 +         value = Integer.parseInt(text.subSequence(position, position +=
length).toString());
50 +     } else {
51 +         int i = position;
52 +         if (negative) {
53 +             i++;
54 +         }
55 +
56 +         final int index = i++;
57 +         if (index > text.length()) {
58 +             position = ~position;
59 +             return;
60 +         }
61 +         value = text.charAt(index) - '0';
62 +         position += length;
63 +         while (i < position) {
64 +             value = (value << 3) + (value << 1) + text.charAt(i++) - '0';
65 +         }
66 +         if (negative) {
67 +             value = -value;
68 +         }
69 +     }
70 +
71 + }
72 +
73 + private static boolean isADigit(final char c) {
74 +     return !isNotADigit(c);
75 + }
76 +
77 + private static boolean isNotADigit(final char c) {
78 +     return c < '0' || c > '9';
79 + }
80 +
81 + private boolean isPrefixedWithPlusOrMinus() {
82 +     final int index = position + length;
83 +     final char currentCharacter = text.charAt(index);
84 +     final boolean isFirstCharacterOperator = length == 0 && (currentCharacter ==
'-') || currentCharacter == '+';
85 +     final boolean hasNextDigitCharacter = index < text.length() - 1 &&
isADigit(text.charAt(index + 1));

```

Figure C.28: insert caption

```

86 +     return isFirstCharacterOperator && isBeforeBoundary() &&
hasNextDigitCharacter;
87 + }
88 +
89 + private boolean isBeforeBoundary() {
90 +     return length + 1 <= limit;
91 + }
92 +
93 + private void updateBasedOnSign() {
94 +     if (isPrefixedWithPlusOrMinus()) {
95 +         negative = text.charAt(position + length) == '-';
96 +         length = (negative) ? length + 1 : length;
97 +         position = (negative) ? position : position + 1;
98 +         // Expand the limit to disregard the sign character.
99 +         limit = Math.min(limit + 1, text.length() -
position);
100 +     }
101 + }
102 +}

```

Figure C.29: insert caption

We can simplify the body of calculate further by applying EXTRACT METHOD, INTRODUCE EXPLAINING VARIABLE, and applying One Return Per Function, an arguably easier-to-understand practice stemming from the principles of Structured Programming, to simplify control.

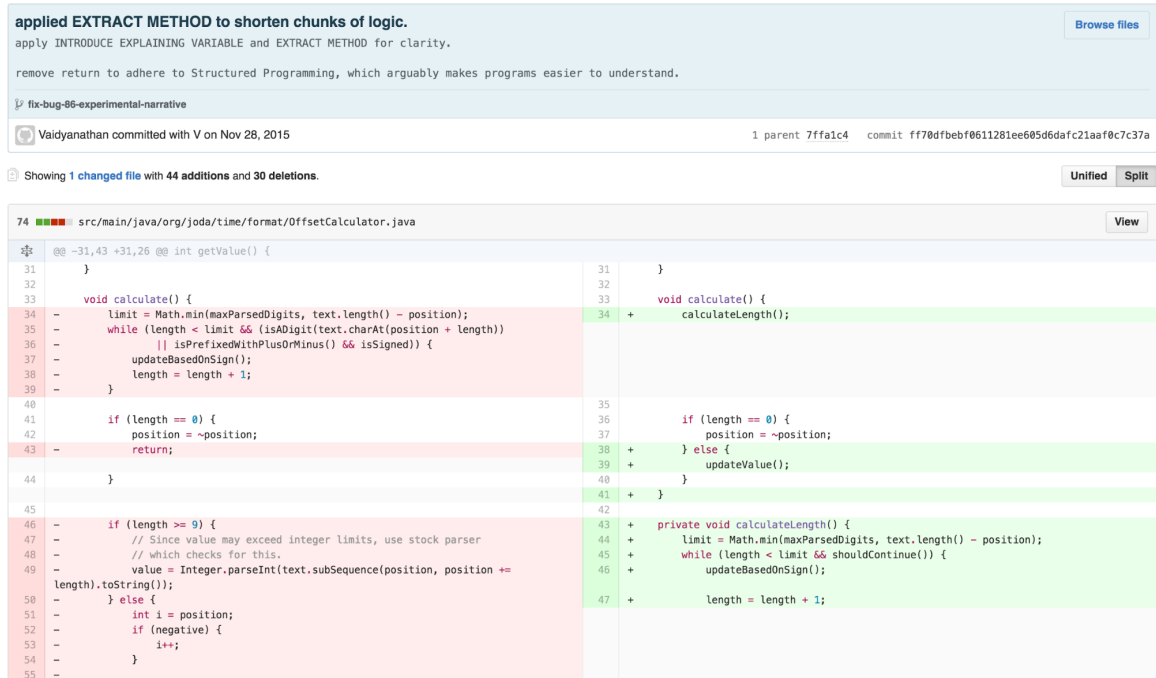


Figure C.30: insert caption



Figure C.31: insert caption

```

102 +     final int index = i++;
103 +     if (index > text.length()) {
104 +         position = ~position;
105 +         return;
106 +     }
107 +     value = text.charAt(index) - '0';
108 +     position += length;
109 +     while (i < position) {
110 +         value = ((value << 3) + (value << 1)) + text.charAt(i++) - '0';
111 +     }
112 +     if (negative) {
113 +         value = -value;
114 +     }
115 + }
116 }

```

Figure C.32: insert caption

I then simplified the logic of `calculateValueForLengthBetween1And8` by applying EXTRACT METHOD and inverting the logic of the conditional, as Clean Code suggests negatives are harder to understand and the inversion allowed me to remove a return statement that broke control flow.

inverted logic because negatives are harder to understand (Clean Code...
 ...) and applied EXTRACT METHOD for clarity.

turn into pure function to try to isolate side effects.

fix-bug-86-experimental-narrative

Vaidyanathan committed with V on Nov 28, 2015

1 parent ff70dfb commit 0ffdb81fe9d406189bf1d078346a3aee697df732f

Showing 1 changed file with 13 additions and 7 deletions.

Unified Split

20 src/main/java/org/joda/time/format/OffsetCalculator.java

```

@@ -100,17 +100,23 @@ private void calculateValueForLengthBetween1And8() {
100 }
101 }
102     final int index = i++;
103 -    if (index > text.length()) {
104         position = ~position;
105         return;
106     }
107     value = text.charAt(index) - '0';
108     position += length;
109     while (i < position) {
110         value = ((value << 3) + (value << 1)) + text.charAt(i++) - '0';
111     }
112     if (negative) {
113         value = -value;
114     }
115 }
116 }

```

```

100 }
101     final int index = i++;
102     if (index < text.length()) {
103         position += length;
104         value = processRemainingCharacters(i, index);
105     } else {
106         position = ~position;
107     }
108 }
109 +
110 + private int processRemainingCharacters(int startingIndex, final int currentIndex)
111 + {
112 +     int calculated = text.charAt(currentIndex) - '0';
113 +     while (startingIndex < position) {
114 +         calculated = ((calculated << 3) + (calculated << 1)) +
115 +             text.charAt(startingIndex++) - '0';
116 +     }
117 +     if (negative) {
118 +         calculated = -calculated;
119 +     }
120 +     return calculated;
121 + }
122 }

```

Figure C.33: insert caption

I then started seeking code to remove, as I detected that some of the methods were redundant from features already offered by the built-in Java libraries. I also applied further EXTRACT METHOD to describe the algorithm declaratively using names and separate the levels of abstraction according to the Newspaper Metaphor.

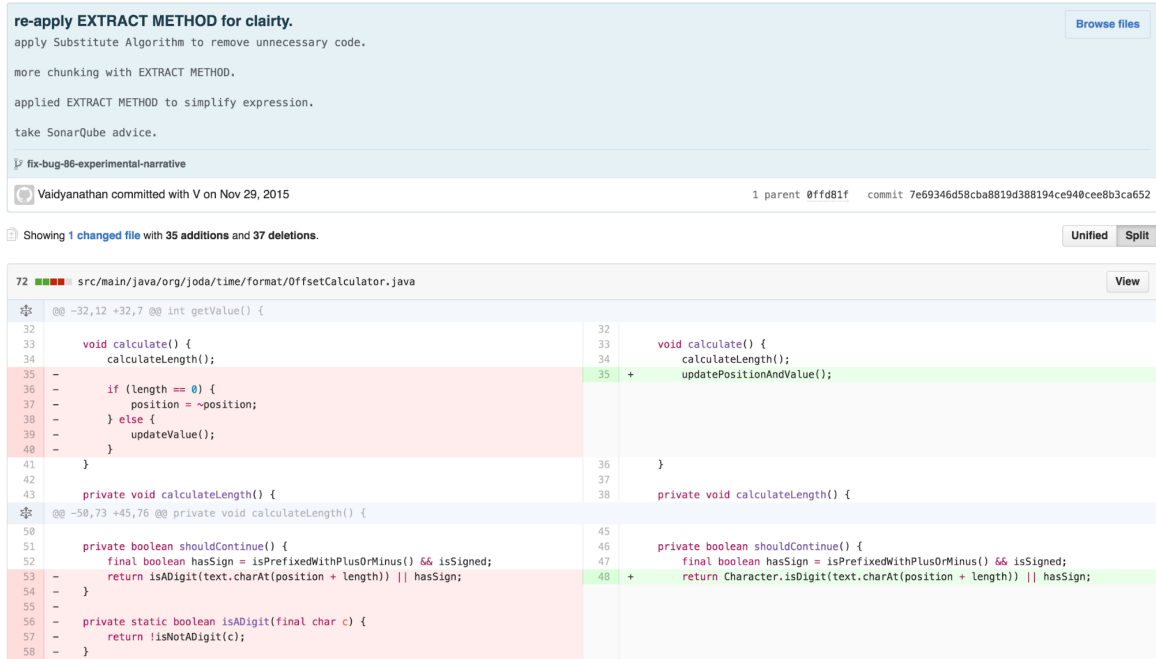


Figure C.34: insert caption



Figure C.35: insert caption

96 - private void calculateValueForLengthBetween1And8() {	87 + private void useDefaultParser() {
97 - int i = position;	88 + // Since value may exceed integer limits, use stock parser
98 - if (negative) {	89 + // which checks for this.
99 - i++;	90 + final String toParse = text.subSequence(position, position +
100 - }	length).toString();
	91 + value = Integer.parseInt(toParse);
	92 + position += length;
	93 + }
	94 + }
	95 + private void useFastParser() {
	96 + int i = negative ? position + 1 : position;
101	
102 final int index = i++;	97
103 if (index < text.length()) {	98 final int index = i++;
104 position += length;	99 if (index < text.length()) {
105 - value = processRemainingCharacters(i, index);	100 position += length;
106 } else {	101 + value = negative ? -calculateValue(i, index) : calculateValue(i, index);
107 position = ~position;	102 } else {
108 }	103 position = ~position;
109 }	104 }
110 }	105 }
111 - private int processRemainingCharacters(int startingIndex, final int currentIndex) {	106 }
112 - int calculated = text.charAt(currentIndex) - '0';	107 + private int calculateValue(final int i, final int index) {
	108 + int startingIndex = i;
	109 + int calculated = getAsciiCharacterFor(index);
113	110
114 while (startingIndex < position) {	111 while (startingIndex < position) {
115 - calculated = ((calculated << 3) + (calculated << 1)) +	112 + calculated = ((calculated << 3) + (calculated << 1)) +
text.charAt(startingIndex++) - '0';	getAsciiCharacterFor(startingIndex++);
116 - }	
117 - if (negative) {	
118 - calculated = -calculated;	
119 }	113 }
120 return calculated;	114 return calculated;
121 }	115 }
122 }	116 +
	117 + private int getAsciiCharacterFor(final int index) {
	118 + return text.charAt(index) - '0';
	119 + }
	120 }

Figure C.36: insert caption

At this point, from a pure “code metrics” perspective, `OffsetCalculator` was significantly simpler. Each method was no more than 9 lines long, the class had 3 getter methods, 3 “utility” style 1-line descriptive methods, and 8 “logical” methods. Being pretty experienced with the code at this point, it was fairly easy for me to see where to “apply the fix” to resolve the bug in this reduced scope.

C.6.1 Checking my biases: adapting to peer feedback

I was pretty confident the new code would be significantly easier to understand. In order to validate this hypothesis, I sought feedback from some expert and novice software engineers. In these dry-run, non-formal talk throughs, I discovered something surprising. Although the `OffsetCalculator` was chunked well for comprehensibility, its overall role within the parsing process was difficult to understand. The obviously tricky `useFastParser()` that used bit shifting was cognitively dense, but even the calculation of length and position that immediately preceded it were difficult to grok in terms of the intent of the calculation. Initial passes at debugging showed that the architecture itself, from the relationship of the `DateTimeFormatter` to the `DateTimeFormatterBuilder` to the `NumberFormatter` to the `OffsetCalculator`, was complex. I realized I need to spend more time with the `DateTimeFormatter` to make it easier to bypass its functionality and get to the `OffsetCalculator` quicker.

C.6.2 Architectural adaptation: Simplifying `DateTimeFormatter`

I started with applying EXTRACT METHOD on spots of obvious duplication the calculation of Chronology, into a method called `getChronology`. I then saw repeated duplication with only detail variation in `parseDateTime`, `parseLocalDateTime`, and `parseMutableDateTime`, so I extracted what was different in those methods out. In

order to create a seam for myself to be able to move the body of those methods into an independently simplified piece without changing the externally facing interface, I extracted the body of the public methods into separate internal private methods.

applied EXTRACT METHOD to eliminate DRY violation.

applied EXTRACT METHOD to push similar lines into similar methods.

refactored public parsing methods to prepare for move.

fix-bug-86-experimental-narrative

V committed on Mar 30

1 parent 7e69346 commit f4868b8be94eff5221085bb38ce9f50777269b1b

Showing 1 changed file with 64 additions and 43 deletions.

Unified Split

107 src/main/java/org/joda/time/format/DateTimeFormatter.java

@@ -747,18 +747,23 @@ public int parseInto(ReadableInstant instant, String text, int position) {

747 instantLocal, chrono, iLocale, iPivotYear, defaultYear);

748 int newPos = parser.parseInto(bucket, text, position);

749 instant.setMillis(bucket.computeMillis(false, text));

750 + chrono = getChronology(chrono, bucket);

751 + instant.setChronology(chrono);

752 + if (iZone != null) {

753 + instant.setZone(iZone);

754 + }

755 + return newPos;

756 + }

757 + }

758 + private Chronology getChronology(Chronology chrono, DateTimeParserBucket bucket)

759 + {

760 + if (iOffsetParsed && bucket.getOffsetInteger() != null) {

761 + int parsedOffset = bucket.getOffsetInteger();

762 + DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);

763 + chrono = chrono.withZone(parsedZone);

764 + } else if (bucket.getZone() != null) {

765 + chrono = chrono.withZone(bucket.getZone());

766 + }

767 + return chrono;

768 + }

747 instantLocal, chrono, iLocale, iPivotYear, defaultYear);

748 int newPos = parser.parseInto(bucket, text, position);

749 instant.setMillis(bucket.computeMillis(false, text));

750 + chrono = getChronology(chrono, bucket);

751 + instant.setChronology(chrono);

752 + if (iZone != null) {

753 + instant.setZone(iZone);

754 + }

755 + return newPos;

756 + }

757 + }

758 + private Chronology getChronology(Chronology chrono, DateTimeParserBucket bucket)

759 + {

760 + if (iOffsetParsed && bucket.getOffsetInteger() != null) {

761 + int parsedOffset = bucket.getOffsetInteger();

762 + DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);

763 + chrono = chrono.withZone(parsedZone);

764 + } else if (bucket.getZone() != null) {

765 + chrono = chrono.withZone(bucket.getZone());

766 + }

767 + return chrono;

768 + }

Figure C.37: insert caption

90

764	/**	769	/**
774	* @throws IllegalArgumentException if the text to parse is invalid	779	* @throws IllegalArgumentException if the text to parse is invalid
775	*/	780	*/
776	public long parseMillis(String text) {	781	public long parseMillis(String text) {
		782	+ return getMillis(text);
		783	+ }
		784	+ }
		785	+ private long getMillis(String text) {
777	InternalParser parser = requireParser();	786	InternalParser parser = requireParser();
778	Chronology chrono = selectChronology(iChrono);	787	Chronology chrono = selectChronology(iChrono);
779	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);	788	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
831	* @since 2.0	840	* @since 2.0
832	*/	841	*/
833	public LocalDateTime parseLocalDateTime(String text) {	842	public LocalDateTime parseLocalDateTime(String text) {
		843	+ return getLocalDateTime(text);
		844	+ }
		845	+ }
		846	+ private LocalDateTime getLocalDateTime(String text) {
834	InternalParser parser = requireParser();	847	InternalParser parser = requireParser();
835	-	848	+ Chronology chrono = selectChronology(null).withUTC(); // always use UTC,
836	Chronology chrono = selectChronology(null).withUTC(); // always use UTC,	849	avoiding DST gaps
837	avoiding DST gaps	850	Chronology chrono = selectChronology(null).withUTC(); // always use UTC,
838	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);	851	avoiding DST gaps
839	int newPos = parser.parseInto(bucket, text, 0);	852	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
840	if (newPos >= 0) {	853	int newPos = parser.parseInto(bucket, text, 0);
841	if (newPos >= text.length()) {	854	if (newPos >= 0) {
842	long millis = bucket.computeMillis(true, text);	855	if (newPos >= text.length()) {
843	if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed()	856	+ return getLocalDateTime(text, chrono, bucket);
844	as being true		
845	int parsedOffset = bucket.getOffsetInteger();		
846	DateTimeZone parsedZone =		
847	DateTimeZone.forOffsetMillis(parsedOffset);		
848	chrono = chrono.withZone(parsedZone);		
849	} else if (bucket.getZone() != null) {		
850	chrono = chrono.withZone(bucket.getZone());		
851	} else {		
852	newPos = ~newPos;		
853	}		

Figure C.38: insert caption

854	throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));	859	throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
855	}	860	}
856		861	
		862	+ private LocalDateTime getLocalDateTime(String text, Chronology chrono,
		863	DateTimeParserBucket bucket) {
		864	+ long millis = bucket.computeMillis(true, text);
		865	+ if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() as
		866	being true
		867	int parsedOffset = bucket.getOffsetInteger();
		868	DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
		869	chrono = chrono.withZone(parsedZone);
		870	} else if (bucket.getZone() != null) {
		871	chrono = chrono.withZone(bucket.getZone());
		872	+ }
		873	+ return new LocalDateTime(millis, chrono);
		874	+ }
875	/**	875	/**
876	* Parses a date-time from the given text, returning a new DateTime.	876	* Parses a date-time from the given text, returning a new DateTime.
877	* <p>		* <p>
878	* @throws IllegalArgumentException if the text to parse is invalid	879	* @throws IllegalArgumentException if the text to parse is invalid
879	*/	880	*/
880	public DateTime parseDateTime(String text) {	881	public DateTime parseDateTime(String text) {
		882	+ return getDateTime(text);
		883	+ }
		884	+ }
		885	+ private DateTime getDateTime(String text) {
886	InternalParser parser = requireParser();	886	InternalParser parser = requireParser();
887	-	887	+ Chronology chrono = selectChronology(null);
888	Chronology chrono = selectChronology(null);	888	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale,
889	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);	889	avoiding DST gaps
890	int newPos = parser.parseInto(bucket, text, 0);	890	Chronology chrono = selectChronology(null);
891	if (newPos >= 0) {	891	avoiding DST gaps
892	if (newPos >= text.length()) {	892	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
893	long millis = bucket.computeMillis(true, text);	893	int newPos = parser.parseInto(bucket, text, 0);
894	if (iOffsetParsed && bucket.getOffsetInteger() != null) {	894	if (newPos >= 0) {
895	int parsedOffset = bucket.getOffsetInteger();	895	if (newPos >= text.length()) {
896	DateTimeZone parsedZone =	896	+ return getDateTime(text, chrono, bucket);
897	DateTimeZone.forOffsetMillis(parsedOffset);		
898	chrono = chrono.withZone(parsedZone);		
899	} else if (bucket.getZone() != null) {		
900	chrono = chrono.withZone(bucket.getZone());		
901	} else {		
902	newPos = ~newPos;		
903	}		

Figure C.39: insert caption

889	- }		
890	- DateTime dt = new DateTime(millis, chrono);		
891	- if (iZone != null) {		
892	- dt = dt.withZone(iZone);		
893	- }		
894	- return dt;		
895	- }	904	}
896	- } else {	905	- } else {
897	- newPos = ~newPos;	906	- newPos = ~newPos;
898	- }	907	- }
899	- throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,	908	- throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,
900	- newPos));	909	- newPos));
901	- }	910	- }
		911	+ private DateTime getDateTime(String text, Chronology chrono, DateTimeParserBucket bucket) {
		912	+ long millis = bucket.computeMillis(true, text);
		913	+ chrono = getChronology(chrono, bucket);
		914	+ DateTime dt = new DateTime(millis, chrono);
		915	+ if (iZone != null) {
		916	+ dt = dt.withZone(iZone);
		917	+ }
		918	+ return dt;
		919	+ }
		920	+ }
902	/**	921	/**
903	* Parses a date-time from the given text, returning a new MutableDateTime.	922	* Parses a date-time from the given text, returning a new MutableDateTime.
904	* <p>	923	* <p>
✱	@ -917,33 +936,35 @ public DateTime parseDateTime(String text) {		
917	- * @throws IllegalArgumentException if the text to parse is invalid	936	- * @throws IllegalArgumentException if the text to parse is invalid
918	- */	937	- */
919	- public MutableDateTime parseMutableDateTime(String text) {	938	+ public MutableDateTime parseMutableDateTime(String text) {
		939	+ return getMutableDateTime(text);
		940	+ }
		941	+ }
		942	+ private MutableDateTime getMutableDateTime(String text) {
920	- InternalParser parser = requireParser();	943	- InternalParser parser = requireParser();
921	- Chronology chrono = selectChronology(null);	944	+ Chronology chrono = selectChronology(null);
922	- DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, ilocale,	945	- DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, ilocale,
923	- iPivotYear, iDefaultYear);	946	- iPivotYear, iDefaultYear);
924	- int newPos = parser.parseInto(bucket, text, 0);	947	- int newPos = parser.parseInto(bucket, text, 0);
925	- if (newPos == 0) {	948	- if (newPos == 0) {
926	- if (newPos >= text.length()) {	949	- if (newPos >= text.length()) {
927	- long millis = bucket.computeMillis(true, text);	950	+ return getMutableDateTime(text, chrono, bucket);

<pre> 928 - if (iOffsetParsed && bucket.getOffsetInteger() != null) { 929 - int parsedOffset = bucket.getOffsetInteger(); 930 - DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset); - chrono = chrono.withZone(parsedZone); 931 - } else if (bucket.getZone() != null) { 932 - chrono = chrono.withZone(bucket.getZone()); 933 - } 934 - MutableDateTime dt = new MutableDateTime(millis, chrono); 935 - if (iZone != null) { 936 - dt.setZone(iZone); 937 - } 938 - return dt; 939 - } 940 - } else { 941 - newPos = ~newPos; 942 - } 943 - throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos)); 944 - } 945 - 946 - </pre>	<pre> 951 - } 952 - } else { 953 - newPos = ~newPos; 954 - } 955 - throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos)); 956 - } 957 - 958 + private MutableDateTime getMutableDateTime(String text, Chronology chrono, DateTimeParserBucket bucket) { 959 + long millis = bucket.computeMillis(true, text); 960 + chrono = getChronology(chrono, bucket); 961 + MutableDateTime dt = new MutableDateTime(millis, chrono); 962 + if (iZone != null) { 963 + dt.setZone(iZone); 964 + } 965 + return dt; 966 + } 967 + </pre>
<pre> 947 - /** 948 - * Checks whether parsing is supported. 949 - */ </pre>	<pre> 968 - /** 969 - * Checks whether parsing is supported. 970 - */ </pre>

I then pushed the responsibility for `Chronology` calculation into a `ChronologyFactory`.

EXTRACT METHOD for Chronology functionality.

push Chronology selection functionality to ChronologyFactory.

fix-bug-86-experimental-narrative

V committed on Mar 30

1 parent f4868b0

commit 4a6c2077d2fa249fd49e07d8b3945c278b26a04b

Showing 2 changed files with 42 additions and 22 deletions.

UnifiedSplit

36 src/main/java/org/joda/time/format/ChronologyFactory.java

View

...

...

@@ -0,0 +1,36 @@

1 +package org.joda.time.format;

2 +

3 +import org.joda.time.Chronology;

4 +import org.joda.time.DateTimeUtils;

5 +import org.joda.time.DateTimeZone;

6 +

7 +final class ChronologyFactory {

8 + private ChronologyFactory() {}

9 + /**

10 + * Determines the correct chronology to use.

11 + *

12 + * @param defaultChronology

13 + * @param defaultTZ @param chrono the proposed chronology

14 + * @return the actual chronology

15 + */

16 + static Chronology selectChronology(Chronology defaultChronology, DateTimeZone defaultTZ, Chronology chrono) {

17 + chrono = getChronologyWithDefaultValue(defaultChronology, chrono);

18 + chrono = getChronologyWithTimeZone(chrono, defaultTZ);

19 + return chrono;

20 + }

21 +

22 + static Chronology getChronologyWithDefaultValue(Chronology defaultChronology, Chronology chrono) {

23 + chrono = DateTimeUtils.getChronology(chrono);

24 + if (defaultChronology != null) {

25 + chrono = defaultChronology;

26 + }

27 + return chrono;

28 + }

29 +

Figure C.42: insert caption

93

```

30 + static Chronology getChronologyWithTimeZone(Chronology chrono, DateTimeZone defaultTimeZone) {
31 +     if (defaultTimeZone != null) {
32 +         chrono = chrono.withZone(defaultTimeZone);
33 +     }
34 +     return chrono;
35 + }
36 +}

```

28 src/main/java/org/joda/time/format/DateTimeFormatter.java View

```

@@ -669,7 +669,7 @@ public String print(ReadablePartial partial) {
669 669
670 670     private void printTo(Appendable appendable, long instant, Chronology chrono) throws IOException {
671 671         InternalPrinter printer = requirePrinter();
672 - chrono = selectChronology(chrono);
672 + chrono = ChronologyFactory.selectChronology(iChrono, iZone, chrono);
673 673         // Shift instant into local time (UTC) to avoid excessive offset
674 674         // calculations when printing multiple fields in a composite printer.
675 675         DateTimeZone zone = chrono.getZone();
@@ -741,7 +741,7 @@ public int parseInto(ReadWritableInstant instant, String text, int position) {
741 741         Chronology chrono = instant.getChronology();
742 742         int defaultYear = DateTimeUtils.getChronology(chrono).year().get(instantMillis);
743 743         long instantLocal = instantMillis + chrono.getZone().getOffset(instantMillis);
744 - chrono = selectChronology(chrono);
744 + chrono = ChronologyFactory.selectChronology(iChrono, iZone, chrono);
745 745
746 746         DateTimeParserBucket bucket = new DateTimeParserBucket(
747 747             instantLocal, chrono, iLocale, iPivotYear, defaultYear);
@@ -784,7 +784,7 @@ public long parseMillis(String text) {
784 784
785 785     private long getMillis(String text) {
786 786         InternalParser parser = requireParser();
787 - Chronology chrono = selectChronology(iChrono);
787 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, iChrono);
788 788         DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
789 789         return bucket.doParseMillis(parser, text);
790 790     }
@@ -846,7 +846,7 @@ public LocalDateTime parseLocalDateTime(String text) {
846 846     private LocalDateTime getLocalDateTime(String text) {
847 847         InternalParser parser = requireParser();
848 848
849 - Chronology chrono = selectChronology(null).withUTC(); // always use UTC, avoiding DST gaps
849 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null).withUTC(); // always use UTC, avo

```

Figure C.43: insert caption

850	850	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
851	851	int newPos = parser.parseInto(bucket, text, 0);
852	852	if (newPos >= 0) {
✳		@@ -895,7 +895,7 @@ public DateTime parseDateTime(String text) {
895	895	private DateTime getDateDateTime(String text) {
896	896	InternalParser parser = requireParser();
897	897	
898	-	Chronology chrono = selectChronology(null);
898	+	Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
899	899	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
900	900	int newPos = parser.parseInto(bucket, text, 0);
901	901	if (newPos >= 0) {
✳		@@ -942,7 +942,7 @@ public MutableDateTime parseMutableDateTime(String text) {
942	942	private MutableDateTime getMutableDateTime(String text) {
943	943	InternalParser parser = requireParser();
944	944	
945	-	Chronology chrono = selectChronology(null);
945	+	Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
946	946	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
947	947	int newPos = parser.parseInto(bucket, text, 0);
948	948	if (newPos >= 0) {
✳		@@ -979,21 +979,5 @@ private InternalParser requireParser() {
979	979	}
980	980	
981	981	//-----
982	-	/**
983	-	* Determines the correct chronology to use.
984	-	*
985	-	* @param chrono the proposed chronology
986	-	* @return the actual chronology
987	-	*/
988	-	private Chronology selectChronology(Chronology chrono) {
989	-	chrono = DateTimeUtils.getChronology(chrono);
990	-	if (iChrono != null) {
991	-	chrono = iChrono;
992	-	}
993	-	if (iZone != null) {
994	-	chrono = chrono.withZone(iZone);
995	-	}
996	-	return chrono;
997	-	}
998	982	
999	983	}

Figure C.44: insert caption

I then saw an opportunity. The similarity of the bodies of the `parseDateTime`, `parseLocalDateTime`, and `parseMutableDateTime` methods suggested to me that a TEMPLATE METHOD was hidden in this class.

The use of a Design Pattern can increase the Germane Cognitive Load of an architecture, which can be more difficult for novices to understand. However, application of this pattern here likely resulted in a dramatically simpler `DateTimeFormatter` and better coherence in the architecture. Following Guideline 25: Write High Coherent Texts for Low Knowledge Readers, this strategic use of a design pattern seemed likely to make the code easier to understand for novices as well. Hence, I started working to make this pattern manifest. Through a series of commits, I realized that `DateTimeFormatter` had a lot of FEATURE ENVY with `DateTimeParserBucket`.

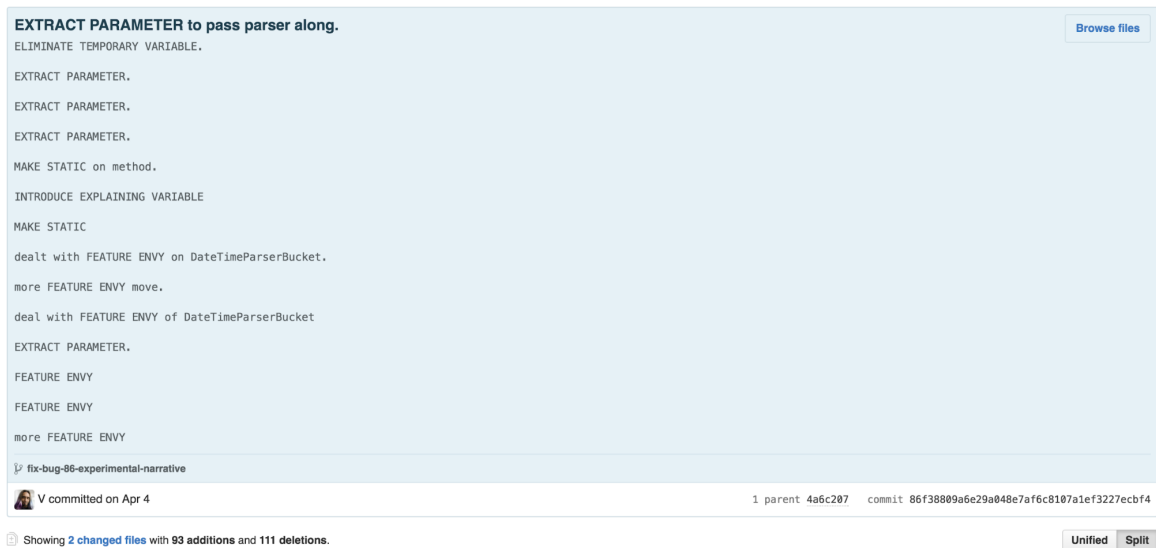


Figure C.45: insert caption

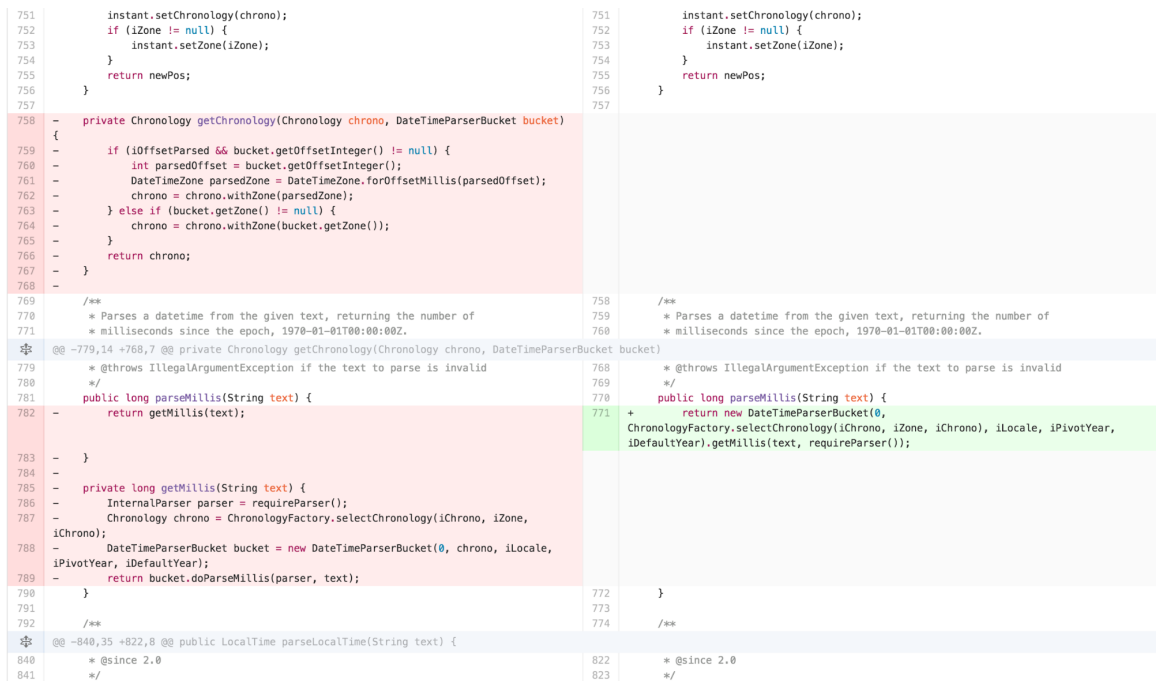


Figure C.46: insert caption

842	<code>public LocalDateTime parseLocalDateTime(String text) {</code>	824	<code>public LocalDateTime parseLocalDateTime(String text) {</code>
843	<code>- return getLocalDateTime(text);</code>	825	<code>+ Chronology chronology = ChronologyFactory.selectChronology(iChrono, iZone,</code>
844	<code>- }</code>	826	<code>null);</code>
845	<code>-</code>		<code>+ return new DateTimeParserBucket(0, chronology.withUTC(), iLocale, iPivotYear,</code>
846	<code>- private LocalDateTime getLocalDateTime(String text) {</code>		<code>iDefaultYear).getLocalDateTime(text, requireParser(), chronology.withUTC());</code>
847	<code>- InternalParser parser = requireParser();</code>		
848	<code>-</code>		
849	<code>- Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone,</code>		
850	<code>null).withUTC(); // always use UTC, avoiding DST gaps</code>		
851	<code>- DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale,</code>		
852	<code>iPivotYear, iDefaultYear);</code>		
853	<code>- int newPos = parser.parseInto(bucket, text, 0);</code>		
854	<code>- if (newPos >= 0) {</code>		
855	<code>- if (newPos >= text.length()) {</code>		
856	<code>- return getLocalDateTime(text, chrono, bucket);</code>		
857	<code>- }</code>		
858	<code>- } else {</code>		
859	<code>- newPos = ~newPos;</code>		
860	<code>- }</code>		
861	<code>- throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,</code>		
862	<code>newPos));</code>		
863	<code>- }</code>		
864	<code>- private LocalDateTime getLocalDateTime(String text, Chronology chrono,</code>		
865	<code>DateTimeParserBucket bucket) {</code>		
866	<code>- long millis = bucket.computeMillis(true, text);</code>		
867	<code>- if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() as</code>		
868	<code>- being true</code>		
869	<code>- int parsedOffset = bucket.getOffsetInteger();</code>		
870	<code>- DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);</code>		
871	<code>- chrono = chrono.withZone(parsedZone);</code>		
872	<code>- } else if (bucket.getZone() != null) {</code>		
873	<code>- chrono = chrono.withZone(bucket.getZone());</code>		
874	<code>- }</code>		
	<code>- return new LocalDateTime(millis, chrono);</code>	827	<code>}</code>
	<code>}</code>	828	
	<code>/**</code>	829	<code>/**</code>

Figure C.47: insert caption

889	<code>* @throws IllegalArgumentException if the text to parse is invalid</code>	844	<code>* @throws IllegalArgumentException if the text to parse is invalid</code>
890	<code>*/</code>	845	<code>*/</code>
891	<code>public DateTime parseDateTime(String text) {</code>	846	<code>public DateTime parseDateTime(String text) {</code>
892	<code>- return getDateTime(text);</code>		
893	<code>- }</code>		
894	<code>-</code>		
895	<code>- private DateTime getDateTime(String text) {</code>		
896	<code>- InternalParser parser = requireParser();</code>		
897	<code>-</code>		
898	<code>- Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);</code>	847	<code>Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);</code>
899	<code>- DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale,</code>	848	<code>+ return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear,</code>
900	<code>iPivotYear, iDefaultYear);</code>		<code>iDefaultYear).getDateTime(iOffsetParsed, iZone, text, requireParser(), chrono);</code>
901	<code>- int newPos = parser.parseInto(bucket, text, 0);</code>		
902	<code>- if (newPos >= 0) {</code>		
903	<code>- if (newPos >= text.length()) {</code>		
904	<code>- return getDateTime(text, chrono, bucket);</code>		
905	<code>- }</code>		
906	<code>- } else {</code>		
907	<code>- newPos = ~newPos;</code>		
908	<code>- }</code>		
909	<code>- throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,</code>		
910	<code>newPos));</code>		
911	<code>- }</code>		
912	<code>- private DateTime getDateTime(String text, Chronology chrono, DateTimeParserBucket</code>		
913	<code>bucket) {</code>		
914	<code>- long millis = bucket.computeMillis(true, text);</code>		
915	<code>- chrono = getChronology(chrono, bucket);</code>		
916	<code>- DateTime dt = new DateTime(millis, chrono);</code>		
917	<code>- if (iZone != null) {</code>		
918	<code>- dt = dt.withZone(iZone);</code>		
919	<code>- }</code>		
920	<code>- return dt;</code>	849	<code>}</code>
921	<code>}</code>	850	
922	<code>/**</code>	851	<code>/**</code>
936	<code>* @throws IllegalArgumentException if the text to parse is invalid</code>	866	<code>* @throws IllegalArgumentException if the text to parse is invalid</code>
937	<code>*/</code>	867	<code>*/</code>
938	<code>public MutableDateTime parseMutableDateTime(String text) {</code>	868	<code>public MutableDateTime parseMutableDateTime(String text) {</code>
939	<code>- return getMutableDateTime(text);</code>		
940	<code>- }</code>		
941	<code>-</code>		

Figure C.48: insert caption

942	- private MutableDateTime getMutableDateTime(String text) {		
943	- InternalParser parser = requireParser();		
944	-		
945	Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);	869	Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
946	- DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale,	870	+ return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear,
947	iPivotYear, iDefaultYear);		iDefaultYear).getMutableDateTime(iOffsetParsed, iZone, text, requireParser(), chrono);
948	- int newPos = parser.parseInto(bucket, text, 0);		
949	- if (newPos >= 0) {		
950	- if (newPos >= text.length()) {		
951	- return getMutableDateTime(text, chrono, bucket);		
952	- } else {		
953	- newPos = ~newPos;		
954	- }		
955	- throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,		
956	newPos));		
957	- }		
958	- private MutableDateTime getMutableDateTime(String text, Chronology chrono,		
959	DateTimeParserBucket bucket) {		
960	- long millis = bucket.computeMillis(true, text);		
961	- chrono = getChronology(chrono, bucket);		
962	- MutableDateTime dt = new MutableDateTime(millis, chrono);		
963	- if (iZone != null) {		
964	- dt.setZone(iZone);		
965	- return dt;		
966	}	871	}
967		872	
968	/**	873	/**
977	@@ -977,7 +882,6 @@ private InternalParser requireParser() {		
978	- return parser;	882	}
979	}	883	+ return parser;
980	-	884	}
981	//-----	885	//-----
982		886	
983	}	887	}

Figure C.49: insert caption

96 src/main/java/org/joda/time/format/DateTimeParserBucket.java				View
18	import java.util.Arrays;	18	import java.util.Arrays;	
19	import java.util.Locale;	19	import java.util.Locale;	
20		20		
21	-import org.joda.time.Chronology;	21	+import org.joda.time.*;	
22	-import org.joda.time.DateTimeField;			
23	-import org.joda.time.DateTimeFieldType;			
24	-import org.joda.time.DateTimeUtils;			
25	-import org.joda.time.DateTimeZone;			
26	-import org.joda.time.DurationField;			
27	-import org.joda.time.DurationFieldType;			
28	-import org.joda.time.IllegalFieldValueException;			
29	-import org.joda.time.IllegalInstantException;			
30		22		
31	/**	23	/**	
32	* DateTimeParserBucket is an advanced class, intended mainly for parser	24	* DateTimeParserBucket is an advanced class, intended mainly for parser	
137	@@ -137,6 +129,88 @@ public DateTimeParserBucket(long instantLocal, Chronology chrono,			
138	- iSavedFields = new SavedField[0];	129	iSavedFields = new SavedField[0];	
139	}	130	}	
		131		
		132	+ static MutableDateTime getMutableDateTime(DateTimeZone iZone, long l, Chronology	
		133	chrono) {	
		134	+ MutableDateTime dt = new MutableDateTime(l, chrono);	
		135	+ if (iZone != null) {	
		136	+ dt.setZone(iZone);	
		137	+ }	
		138	+ return dt;	
		139	+ }	
		140	+ LocalDateTime getLocalDateTime(String text, InternalParser parser, Chronology	
		141	chrono) {	
		142	+ int newPos = parser.parseInto(this, text, 0);	
		143	+ if (newPos >= 0) {	
		144	+ if (newPos >= text.length()) {	
		145	+ return getLocalDateTime(text, chrono);	
		146	+ }	
		147	+ } else {	
		148	+ newPos = ~newPos;	
		149	+ }	
		150	+ throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,	
		151	newPos));	
			+ }	

Figure C.50: insert caption

	<pre> 152 + 153 + LocalDateTime getLocalDateTime(String text, Chronology chrono) { 154 + long millis = computeMillis(true, text); 155 + if (getOffsetInteger() != null) { // treat withOffsetParsed() as being true 156 + int parsedOffset = getOffsetInteger(); 157 + DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset); 158 + chrono = chrono.withZone(parsedZone); 159 + } else if (getZone() != null) { 160 + chrono = chrono.withZone(getZone()); 161 + } 162 + return new LocalDateTime(millis, chrono); 163 + } 164 + 165 + MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, 166 + String text, InternalParser parser, Chronology chrono) { 167 + int newPos = parser.parseInto(this, text, 0); 168 + if (newPos >= 0) { 169 + if (newPos >= text.length()) { 170 + return getMutableDateTime(iOffsetParsed, iZone, text, chrono); 171 + } 172 + } else { 173 + newPos = ~newPos; 174 + } 175 + throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, 176 + newPos)); 177 + } 178 + 179 + MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, 180 + String text, Chronology chrono) { 181 + return getMutableDateTime(iZone, computeMillis(true, text), 182 + getChronology(iOffsetParsed, chrono)); 183 + } 184 + 185 + DateTime getDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, 186 + final InternalParser parser, Chronology chrono) { 187 + int newPos = parser.parseInto(this, text, 0); 188 + if (newPos >= 0) { 189 + if (newPos >= text.length()) { 190 + return getDateTime(iZone, getChronology(iOffsetParsed, chrono), 191 + computeMillis(true, text)); 192 + } 193 + } else { 194 + newPos = ~newPos; 195 + } 196 + } </pre>
--	--

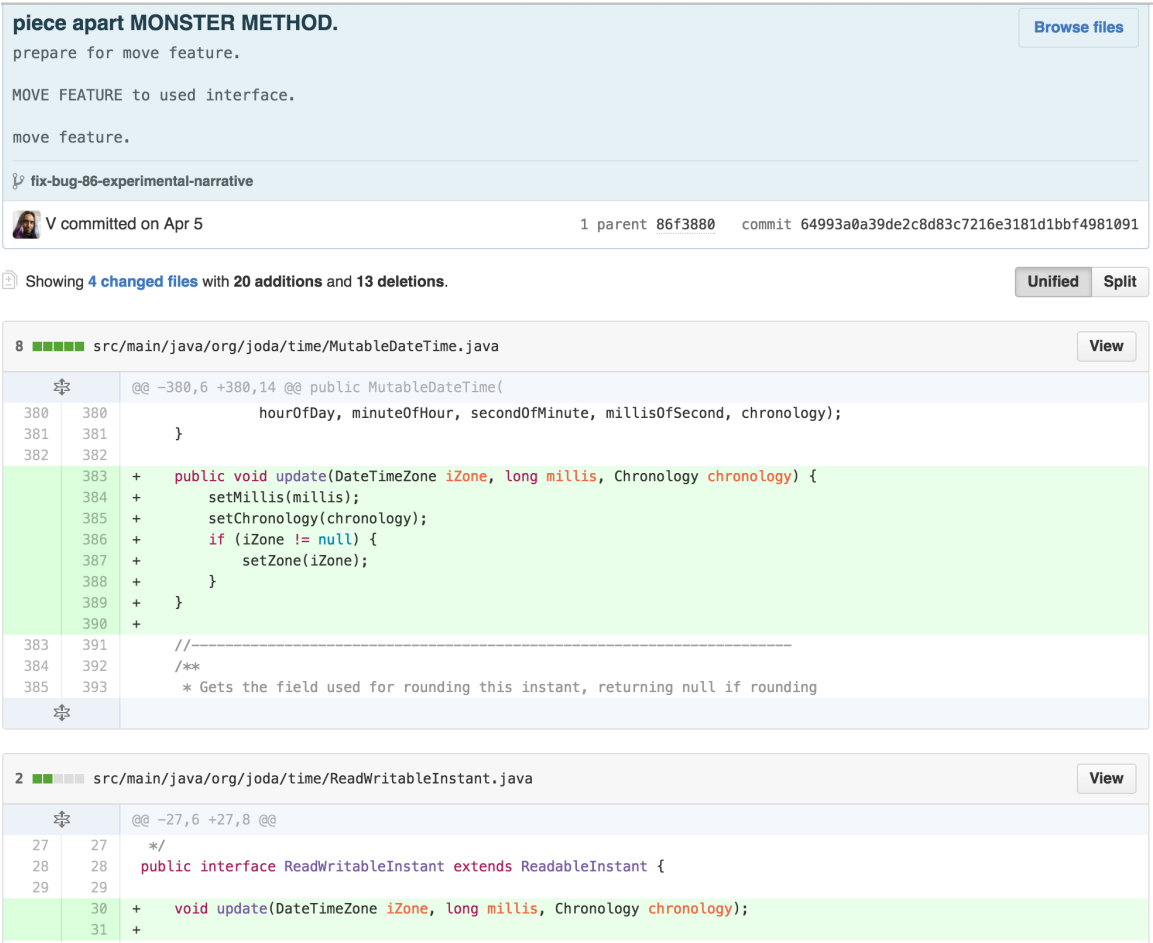
Figure C.51: insert caption

	<pre> 192 + throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, 193 + newPos)); 194 + } 195 + 196 + Chronology getChronology(boolean iOffsetParsed, Chronology chrono) { 197 + if (iOffsetParsed && getOffsetInteger() != null) { 198 + int parsedOffset = getOffsetInteger(); 199 + DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset); 200 + chrono = chrono.withZone(parsedZone); 201 + } else if (getZone() != null) { 202 + chrono = chrono.withZone(getZone()); 203 + } 204 + return chrono; 205 + } 206 + 207 + private static DateTime getDateTime(DateTimeZone iZone, Chronology chronology, 208 + long millis) { 209 + DateTime dt = new DateTime(millis, chronology); 210 + if (iZone != null) { 211 + dt = dt.withZone(iZone); 212 + } 213 + return dt; 214 + } </pre>
<pre> 140 //----- 141 /** 142 * Resets the state back to that when the object was constructed. 143 */ 144 @ -507,6 +581,10 @@ private static void sort(SavedField[] array, int high) { 145 } 146 } 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 </pre>	<pre> 581 + } 582 + } 583 + 584 + long getMillis(String text, InternalParser parser) { 585 + return doParseMillis(parser, text); 586 + } 587 + 588 + 589 + 590 + 591 + 592 + 593 + 594 + 595 + 596 + 597 + 598 + 599 + 600 + 601 + 602 + 603 + 604 + 605 + 606 + 607 + 608 + 609 + 610 + 611 + 612 + 613 + 614 + 615 + 616 + 617 + 618 + 619 + 620 + 621 + 622 + 623 + 624 + 625 + 626 + 627 + 628 + 629 + 630 + 631 + 632 + 633 + 634 + 635 + 636 + 637 + 638 + 639 + 640 + 641 + 642 + 643 + 644 + 645 + 646 + 647 + 648 + 649 + 650 + 651 + 652 + 653 + 654 + 655 + 656 + 657 + 658 + 659 + 660 + 661 + 662 + 663 + 664 + 665 + 666 + 667 + 668 + 669 + 670 + 671 + 672 + 673 + 674 + 675 + 676 + 677 + 678 + 679 + 680 + 681 + 682 + 683 + 684 + 685 + 686 + 687 + 688 + 689 + 690 + 691 + 692 + 693 + 694 + 695 + 696 + 697 + 698 + 699 + 700 + 701 + 702 + 703 + 704 + 705 + 706 + 707 + 708 + 709 + 710 + 711 + 712 + 713 + 714 + 715 + 716 + 717 + 718 + 719 + 720 + 721 + 722 + 723 + 724 + 725 + 726 + 727 + 728 + 729 + 730 + 731 + 732 + 733 + 734 + 735 + 736 + 737 + 738 + 739 + 740 + 741 + 742 + 743 + 744 + 745 + 746 + 747 + 748 + 749 + 750 + 751 + 752 + 753 + 754 + 755 + 756 + 757 + 758 + 759 + 760 + 761 + 762 + 763 + 764 + 765 + 766 + 767 + 768 + 769 + 770 + 771 + 772 + 773 + 774 + 775 + 776 + 777 + 778 + 779 + 780 + 781 + 782 + 783 + 784 + 785 + 786 + 787 + 788 + 789 + 790 + 791 + 792 + 793 + 794 + 795 + 796 + 797 + 798 + 799 + 800 + 801 + 802 + 803 + 804 + 805 + 806 + 807 + 808 + 809 + 810 + 811 + 812 + 813 + 814 + 815 + 816 + 817 + 818 + 819 + 820 + 821 + 822 + 823 + 824 + 825 + 826 + 827 + 828 + 829 + 830 + 831 + 832 + 833 + 834 + 835 + 836 + 837 + 838 + 839 + 840 + 841 + 842 + 843 + 844 + 845 + 846 + 847 + 848 + 849 + 850 + 851 + 852 + 853 + 854 + 855 + 856 + 857 + 858 + 859 + 860 + 861 + 862 + 863 + 864 + 865 + 866 + 867 + 868 + 869 + 870 + 871 + 872 + 873 + 874 + 875 + 876 + 877 + 878 + 879 + 880 + 881 + 882 + 883 + 884 + 885 + 886 + 887 + 888 + 889 + 890 + 891 + 892 + 893 + 894 + 895 + 896 + 897 + 898 + 899 + 900 + 901 + 902 + 903 + 904 + 905 + 906 + 907 + 908 + 909 + 910 + 911 + 912 + 913 + 914 + 915 + 916 + 917 + 918 + 919 + 920 + 921 + 922 + 923 + 924 + 925 + 926 + 927 + 928 + 929 + 930 + 931 + 932 + 933 + 934 + 935 + 936 + 937 + 938 + 939 + 940 + 941 + 942 + 943 + 944 + 945 + 946 + 947 + 948 + 949 + 950 + 951 + 952 + 953 + 954 + 955 + 956 + 957 + 958 + 959 + 960 + 961 + 962 + 963 + 964 + 965 + 966 + 967 + 968 + 969 + 970 + 971 + 972 + 973 + 974 + 975 + 976 + 977 + 978 + 979 + 980 + 981 + 982 + 983 + 984 + 985 + 986 + 987 + 988 + 989 + 990 + 991 + 992 + 993 + 994 + 995 + 996 + 997 + 998 + 999 </pre>

Figure C.52: insert caption

At this point, I recognized that much of the logic in `parseIntoInstant` in `DateTimeFormatter` was similar to the other parsing methods. `DateTimeFormatter` also found itself doing

more work in its internals than necessary because the `ReadWriteableInstant` interface was missing an update feature. As that functionality seems implied by the name, I added it to the interface and pushed some responsibility onto `MutableDateTime`.



30	32	/**
31	33	* Sets the value as the number of milliseconds since
32	34	* the epoch, 1970-01-01T00:00:00Z.
✚		

17 ■■■■ src/main/java/org/joda/time/format/DateTimeFormatter.java			View	
		@@ -712,47 +712,38 @@		
712	712	* fields before calling this method, or use {@link #parseDateTime(String)}		
713	713	* or {@link #parseMutableDateTime(String)}.		
714	714	* <p>		
715	715	* If it fails, the return value is negative, but the instant may still be		
716	716	* modified. To determine the position where the parse failed, apply the		
717	717	* one's complement operator (~) on the return value.		
718	718	* <p>		
719	719	* This parse method ignores the {@link #getDefaultYear() default year} and		
720	720	* parses using the year from the supplied instant based on the chronology		
721	721	* and time-zone of the supplied instant.		
722	722	* <p>		
723	723	* The parse will use the chronology of the instant.		
724	724	*		
725	725	* @param instant an instant that will be modified, not null		
726	726	* @param text the text to parse		
727	727	* @param position position to start parsing from		
728	728	* @return new position, negative value means parse failed -		
729	729	* apply complement operator (~) to get position of failure		
730	730	* @throws UnsupportedOperationException if parsing is not supported		
731	731	* @throws IllegalArgumentException if the instant is null		
732	732	* @throws IllegalArgumentException if any field is out of range		
733	733	*/		
734	734	public int parseInto(ReadWritableInstant instant, String text, int position) {		
735	735	InternalParser parser = requireParser();	-	
736	735	if (instant == null) {		
737	736	throw new IllegalArgumentException("Instant must not be null");		
738	737	}		
739		-	-	
740		- long instantMillis = instant.getMillis();		
741	738	Chronology chrono = instant.getChronology();		
739		+ long instantMillis = instant.getMillis();	+	
742	740	int defaultYear = DateTimeUtils.getChronology(chrono).year().get(instantMillis);		
743	741	long instantLocal = instantMillis + chrono.getZone().getOffset(instantMillis);		
744	742	chrono = ChronologyFactory.selectChronology(iChrono, iZone, chrono);		
745		-	-	
	743	+	+	

Figure C.54: insert caption

746	744	DateTimeParserBucket bucket = new DateTimeParserBucket(747 - instantLocal, chrono, iLocale, iPivotYear, defaultYear); 748 - int newPos = parser.parseInto(bucket, text, position); 749 - instant.setMillis(bucket.computeMillis(false, text)); 750 - chrono = bucket.getChronology(iOffsetParsed, chrono); 751 - instant.setChronology(chrono); 752 - if (iZone != null) { 753 - instant.setZone(iZone); 754 - } 755 - return newPos; 745 + instantLocal, chrono, iLocale, iPivotYear, defaultYear); 746 + return bucket.parseIntoInstant(iOffsetParsed, iZone, instant, text, position, requireParser(), chrono); 756 747 } 757 748 758 749 /** ✚
-----	-----	--

6■■■■■src/main/java/org/joda/time/format/DateTimeParserBucket.javaView

✚	@@ -137,6 +137,12 @@ static MutableDateTime getMutableDateTime(DateTimeZone iZone, long l, Chronology
137	137 return dt;
138	138 }
139	139
	140 + int parseIntoInstant(boolean iOffsetParsed, DateTimeZone iZone, ReadWritableInstant instant, String text, int posit
	141 + int newPos = parser.parseInto(this, text, position);
	142 + instant.update(iZone, computeMillis(false, text), getChronology(iOffsetParsed, chrono));
	143 + return newPos;
	144 + }
	145 + }
140	146 LocalDateTime getLocalDateTime(String text, InternalParser parser, Chronology chrono) {
141	147
142	148 int newPos = parser.parseInto(this, text, 0);
✚	

Figure C.55: insert caption

At this point, I was able to push the entire responsibility of parsing into the `Instant` into `DateTimeParserBucket`.

INLINE VARIABLE.
INLINE TEMP VARIABLE.

INLINE TEMP VARIABLE.

INLINE TEMP VARIABLE.

move body of method to new constructor to localize effect of scope into DateTimeParserBucket.

remove unnecessary parameter.

REMOVE PARAMETER.

[Browse files](#)

 V committed 27 days ago

1 parent 64993a0 commit b48f8c9ee7f10059c16b1f1b45915b0b247baca9

Showing 2 changed files with 17 additions and 11 deletions.

Unified

Split

13  src/main/java/org/joda/time/format/DateTimeFormatter.java

View

	@@ -735,15 +735,10 @@ public int parseInto(ReadWritableInstant instant, String text, int position) {
735	735
736	736
737	737
738	- Chronology chrono = instant.getChronology();
739	- long instantMillis = instant.getMillis();
740	- int defaultYear = DateTimeUtils.getChronology(chrono).year().get(instantMillis);
741	- long instantLocal = instantMillis + chrono.getZone().getOffset(instantMillis);
742	- chrono = ChronologyFactory.selectChronology(iChrono, iZone, chrono);
743	-
744	- DateTimeParserBucket bucket = new DateTimeParserBucket(
745	- instantLocal, chrono, iLocale, iPivotYear, defaultYear);
746	- return bucket.parseIntoInstant(iOffsetParsed, iZone, instant, text, position, requireParser(), chrono);
738	+ DateTimeParserBucket bucket = new DateTimeParserBucket(instant,
739	+ ChronologyFactory.selectChronology(iChrono, iZone, instant.getChronology()),
740	+ iLocale, iPivotYear) ;
741	+ return bucket.parseIntoInstant(iOffsetParsed, iZone, instant, text, position, requireParser());
747	742
748	743

Figure C.56: insert caption

749	744	/**
✱		
15 ■■■■ src/main/java/org/joda/time/format/DateTimeParserBucket.java View		
✱		@ -102,6 +102,17 @@ public DateTimeParserBucket(long instantLocal, Chronology chrono, Locale locale,
102	102	this(instantLocal, chrono, locale, pivotYear, 2000);
103	103	}
104	104	
	105	+ public DateTimeParserBucket(ReadWritableInstant instant,
	106	+ Chronology iChrono,
	107	+ Locale iLocale,
	108	+ Integer iPivotYear) {
	109	+ this(instant.getMillis() + instant.getChronology().getZone().getOffset(instant.getMillis()),
	110	+ iChrono,
	111	+ iLocale,
	112	+ iPivotYear,
	113	+ DateTimeUtils.getChronology(instant.getChronology()).year().get(instant.getMillis())
	114	+);
	115	+ }
105	116	/**
106	117	* Constructs a bucket, with the option of specifying the pivot year for
107	118	* two-digit year parsing.
✱		@ -137,9 +148,9 @@ static MutableDateTime getMutableDateTime(DateTimeZone iZone, long l, Chronology
137	148	return dt;
138	149	}
139	150	
140	151	- int parseIntoInstant(boolean iOffsetParsed, DateTimeZone iZone, ReadWritableInstant instant, String text, int posit
	152	+ int parseIntoInstant(boolean iOffsetParsed, DateTimeZone iZone, ReadWritableInstant instant, String text, int posit
141	152	int newPos = parser.parseInto(this, text, position);
142	153	- instant.update(iZone, computeMillis(false, text), getChronology(iOffsetParsed, chrono));
	154	+ instant.update(iZone, computeMillis(false, text), getChronology(iOffsetParsed, iChrono));
143	154	return newPos;
144	155	}
145	156	
✱		

Figure C.57: insert caption

`DateTimeFormatter`'s parse methods were now doing very little besides using its state to set up a `DateTimeParserBucket`. It seemed like `DateTimeParserBucket` could be the permanent home of much of this functionality. I could move the functionality over. Along the way, I could start by removing some of the cognitive complexity of `ChronologyFactory`'s reassignment to input parameters simultaneously.

apply MOVE METHOD to encapsulate functionality in DateTimeParserBucket.
[Browse files](#)

move related private method to DateTimeParserBucket.

fix-bug-86-experimental-narrative


 V committed 27 days ago

1 parent 23659e6 commit dc19ea0e92099ca6faee8b907731bc5f3f963514

Showing 2 changed files with 51 additions and 32 deletions.

Unified Split

31 src/main/java/org/joda/time/format/DateTimeFormatter.java [View](#)

```

@@ -732,13 +732,7 @@ private InternalPrinter requirePrinter() {
732 732     * @throws IllegalArgumentException if any field is out of range
733 733     */
734 734     public int parseInto(ReadWritableInstant instant, String text, int position) {
735         - if (instant == null) {
736         -     throw new IllegalArgumentException("Instant must not be null");
737         - }
738         - DateTimeParserBucket bucket = new DateTimeParserBucket(instant,
739         -     ChronologyFactory.selectChronology(iChrono, iZone, instant.getChronology()),
740         -     iLocale, iPivotYear);
741         - return bucket.parseIntoInstant(iOffsetParsed, iZone, instant, text, position, requireParser());
742         + return DateTimeParserBucket.parseIntoReadWritableInstant(iChrono, iLocale, iOffsetParsed, iPivotYear, iZone, i
743         }
744         /**
@@ -754,7 +748,7 @@ public int parseInto(ReadWritableInstant instant, String text, int position) {
754 748     * @throws IllegalArgumentException if the text to parse is invalid
755 749     */
756 750     public long parseMillis(String text) {
757         - return new DateTimeParserBucket(0, ChronologyFactory.selectChronology(iChrono, iZone, iChrono), iLocale, iPivot
758         + return DateTimeParserBucket.parseMillis(iChrono, iDefaultYear, iLocale, iPivotYear, iZone, text, this.iParser);
759         }
760         /**
@@ -808,8 +802,7 @@ public LocalTime parseLocalTime(String text) {
808 802     * @since 2.0
809 803     */

```

Figure C.59: insert caption

810	804	<code>public LocalDateTime parseLocalDateTime(String text) {</code>
811		<code>- Chronology chronology = ChronologyFactory.selectChronology(iChrono, iZone, null);</code>
812		<code>- return new DateTimeParserBucket(0, chronology.withUTC(), iLocale, iPivotYear, iDefaultYear).getLocalDateTime(text);</code>
	805	<code>+ return DateTimeParserBucket.parseLocalDateTime(iChrono, iDefaultYear, iLocale, iPivotYear, iZone, text, this.iParser);</code>
813	806	<code>}</code>
814	807	
815	808	<code>/**</code>
		<code>@@ -830,8 +823,7 @@ public LocalDateTime parseLocalDateTime(String text) {</code>
830	823	<code>* @throws IllegalArgumentException if the text to parse is invalid</code>
831	824	<code>*/</code>
832	825	<code>public DateTime parseDateTime(String text) {</code>
833		<code>- Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);</code>
834		<code>- return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear).getDateTime(iOffsetParsed, iZone, text);</code>
	826	<code>+ return DateTimeParserBucket.parseDateTime(iChrono, iDefaultYear, iLocale, iOffsetParsed, iPivotYear, iZone, text);</code>
835	827	<code>}</code>
836	828	
837	829	<code>/**</code>
		<code>@@ -852,22 +844,9 @@ public DateTime parseDateTime(String text) {</code>
852	844	<code>* @throws IllegalArgumentException if the text to parse is invalid</code>
853	845	<code>*/</code>
854	846	<code>public MutableDateTime parseMutableDateTime(String text) {</code>
855		<code>- Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);</code>
856		<code>- return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear).getMutableDateTime(iOffsetParsed, iZone, text);</code>
	847	<code>+ return DateTimeParserBucket.parseMutableDateTime(iChrono, iDefaultYear, iLocale, iOffsetParsed, iPivotYear, iZone, text);</code>
857	848	<code>}</code>
858	849	
859		<code>/**</code>
860		<code>- * Checks whether parsing is supported.</code>
861		<code>- *</code>
862		<code>- * @throws UnsupportedOperationException if parsing is not supported</code>
863		<code>*/</code>
864		<code>private InternalParser requireParser() {</code>
865		<code>- InternalParser parser = iParser;</code>
866		<code>- if (parser == null) {</code>
867		<code>- throw new UnsupportedOperationException("Parsing not supported");</code>
868		<code>- }</code>
869		<code>- return parser;</code>
870		<code>- }</code>
871	850	<code>//-----</code>
872	851	
873	852	<code>}</code>

Figure C.60: insert caption

52
src/main/java/org/joda/time/format/DateTimeParserBucket.java
View

```

148 148      return dt;
149 149    }
150 150
151 - int parseIntoInstant(boolean iOffsetParsed, DateTimeZone iZone, ReadWritableInstant instant, String text, int posit
152 - int newPos = parser.parseInto(this, text, position);
153 - instant.update(iZone, computeMillis(false, text), getChronology(iOffsetParsed, iChrono));
151 + static int parseIntoReadWritableInstant(Chronology iChrono, Locale iLocale, boolean iOffsetParsed, Integer iPivotY
152 + if (instant == null) {
153 +     throw new IllegalArgumentException("Instant must not be null");
154 + }
155 + DateTimeParserBucket bucket = new DateTimeParserBucket(instant,
156 +     ChronologyFactory.selectChronology(iChrono, iZone, instant.getChronology()),
157 +     iLocale, iPivotYear);
158 + int newPos = requireParser(parser).parseInto(bucket, text, position);
159 + instant.update(iZone, bucket.computeMillis(false, text), bucket.getChronology(iOffsetParsed, bucket.iChrono));
154 160      return newPos;
155 161    }
156 162
163 + static long parseMillis(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZon
164 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, iChrono);
165 + return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear).getMillis(text, parser);
166 + }
167 +
168 + static LocalDateTime parseLocalDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, D
169 + Chronology chronology = ChronologyFactory.selectChronology(iChrono, iZone, null);
170 + return new DateTimeParserBucket(0, chronology.withUTC(), iLocale, iPivotYear, iDefaultYear).getLocalDateTime(te
171 + }
172 +
173 + static DateTime parseDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, boolean iOffsetParsed, Integer
174 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
175 + return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear).getDateTime(iOffsetParsed, iZone,
176 + }
177 +
178 + static MutableDateTime parseMutableDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, boolean iOffsetPa
179 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
180 + return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear).getMutableDateTime(iOffsetParsed,
181 + }
182 +

```

Figure C.61: insert caption

183	183	+	/**
184	184	+	* Checks whether parsing is supported.
185	185	+	*
186	186	+	* @throws UnsupportedOperationException if parsing is not supported
187	187	+	* @param iParser
188	188	+	*/
189	189	+	private static InternalParser requireParser(InternalParser iParser) {
190	190	+	InternalParser parser = iParser;
191	191	+	if (parser == null) {
192	192	+	throw new UnsupportedOperationException("Parsing not supported");
193	193	+	}
194	194	+	return parser;
195	195	+	}
196	196	+	
157	197		LocalDateTime getLocalDateTime(String text, InternalParser parser, Chronology chrono) {
158	198		
159	199	-	int newPos = parser.parseInto(this, text, 0);
199	200	+	int newPos = requireParser(parser).parseInto(this, text, 0);
160	200		if (newPos >= 0) {
161	201		if (newPos >= text.length()) {
162	202		return getLocalDateTime(text, chrono);
✂			@@ -181,7 +221,7 @@ LocalDateTime getLocalDateTime(String text, Chronology chrono) {
181	221		
182	222		MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, InternalParser parser, C
183	223		
184	224	-	int newPos = parser.parseInto(this, text, 0);
224	224	+	int newPos = requireParser(parser).parseInto(this, text, 0);
185	225		if (newPos >= 0) {
186	226		if (newPos >= text.length()) {
187	227		return getMutableDateTime(iOffsetParsed, iZone, text, chrono);
✂			@@ -198,7 +238,7 @@ MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, St
198	238		
199	239		
200	240		DateTime getDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, final InternalParser parser, Chronolog
201	241	-	int newPos = parser.parseInto(this, text, 0);
241	241	+	int newPos = requireParser(parser).parseInto(this, text, 0);
202	242		if (newPos >= 0) {
203	243		if (newPos >= text.length()) {
204	244		return getDateTime(iZone, getChronology(iOffsetParsed, chrono), computeMillis(true, text));
✂			

Figure C.62: insert caption

At this point, it felt like I was succeeding more in moving complexity around than removing it. `DateTimeParserBucket` had most of the functionality related to parsing moved to it, as its internal state was a big part of most of the calculations. But I was no closer to my originally envisioned TEMPLATE METHOD. I also worried that more responsibility on an originally used intermediate calculation scratchpad object may belie its importance in the process. I sought further simplification. I started by reducing the scope of some methods of `DateTimeParserBucket` and re-arranging them according to the Newspaper Metaphor.

match symmetry.
simplify.

narrow visibility of methods.

Newspaper metaphor.

[Browse files](#)

fix-bug-86-experimental-narrative

V committed 27 days ago

1 parent `dc19ea0`
commit `8180ce6f848a15292aa14eeec49c74418fb08739`

Showing 1 changed file with 32 additions and 34 deletions.

Unified
Split

66
src/main/java/org/joda/time/format/DateTimeParserBucket.java

View

		@@ -102,17 +102,6 @@ public DateTimeParserBucket(long instantLocal, Chronology chrono, Locale locale,
102	102	this(instantLocal, chrono, locale, pivotYear, 2000);
103	103	}
104	104	
105		- public DateTimeParserBucket(ReadWritableInstant instant,
106		- Chronology iChrono,
107		- Locale iLocale,
108		- Integer iPivotYear) {
109		- this(instant.getMillis() + instant.getChronology().getZone().getOffset(instant.getMillis()),
110		- iChrono,
111		- iLocale,
112		- iPivotYear,
113		- DateTimeUtils.getChronology(instant.getChronology()).year().get(instant.getMillis())
114		-);
115		- }
116	105	/**
117	106	* Constructs a bucket, with the option of specifying the pivot year for
118	107	* two-digit year parsing.
		@@ -140,44 +129,45 @@ public DateTimeParserBucket(long instantLocal, Chronology chrono,
140	129	iSavedFields = new SavedField[8];
141	130	}
142	131	
143		- static MutableDateTime getMutableDateTime(DateTimeZone iZone, long l, Chronology chronology) {
144		- MutableDateTime dt = new MutableDateTime(l, chronology);
145		- if (iZone != null) {
146		- dt.setZone(iZone);
147		- }

Figure C.63: insert caption

```

148 -         return dt;
149 -     }
150 -
151 132     static int parseIntoReadWritableInstant(Chronology iChrono, Locale iLocale, boolean iOffsetParsed, Integer iPivotYear,
152 133     if (instant == null) {
153 134         throw new IllegalArgumentException("Instant must not be null");
154 135     }
155 -     DateTimeParserBucket bucket = new DateTimeParserBucket(instant,
156 -         ChronologyFactory.selectChronology(iChrono, iZone, instant.getChronology()),
157 -         iLocale, iPivotYear) ;
158 -     int newPos = requireParser(parser).parseInto(bucket, text, position);
159 -     instant.update(iZone, bucket.computeMillis(false, text), bucket.getChronology(iOffsetParsed, bucket.iChrono));
160 +     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, instant.getChronology());
161 +     long millis = instant.getMillis() + instant.getChronology().getZone().getOffset(instant.getMillis());
162 +     int defaultYear = DateTimeUtils.getChronology(instant.getChronology()).year().get(instant.getMillis());
163 +     DateTimeParserBucket bucket = new DateTimeParserBucket(millis, chrono, iLocale, iPivotYear, defaultYear);
164 +     return bucket.parseIntoInstantAndGetPosition(iOffsetParsed, iZone, instant, text, position, parser);
165 + }
166 +
167 + private int parseIntoInstantAndGetPosition(boolean iOffsetParsed, DateTimeZone iZone, ReadWritableInstant instant, String text,
168 +     int newPos = requireParser(parser).parseInto(this, text, position);
169 +     instant.update(iZone, computeMillis(false, text), getChronology(iOffsetParsed, iChrono));
170 +     return newPos;
171 + }
172 +
173 + static long parseMillis(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZone,
174 +     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, iChrono);
175 -     return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear).getMillis(text, parser);
176 +     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
177 +     return bucket.getMillis(text, parser);
178 + }
179 +
180 + static LocalDateTime parseLocalDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZone,
181 +     Chronology chronology = ChronologyFactory.selectChronology(iChrono, iZone, null);
182 -     return new DateTimeParserBucket(0, chronology.withUTC(), iLocale, iPivotYear, iDefaultYear).getLocalDateTime(text, parser);
183 +     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chronology.withUTC(), iLocale, iPivotYear, iDefaultYear);
184 +     return bucket.getLocalDateTime(text, parser, chronology.withUTC());
185 + }
186 +
187 + static DateTime parseDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, boolean iOffsetParsed, Integer iPivotYear,
188 +     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
189 -     return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear).getDateTime(iOffsetParsed, iZone, text, parser);
190 +     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
191 +     return bucket.getDateTime(iOffsetParsed, iZone, text, parser, chrono);
192 + }
193 +
194 + }

```

Figure C.64: insert caption

178	167	static MutableDateTime parseMutableDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, boolean iOffsetPars
179	168	Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
180	-	return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear).getMutableDateTime(iOffsetParsed, i
	169	DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
	170	return bucket.getMutableDateTime(iOffsetParsed, iZone, text, parser, chrono);
181	171	}
182	172	
183	173	/**
✳		@@ -194,7 +184,7 @@ private static InternalParser requireParser(InternalParser iParser) {
194	184	return parser;
195	185	}
196	186	
197	-	LocalDateTime getLocalDateTime(String text, InternalParser parser, Chronology chrono) {
	187	+ private LocalDateTime getLocalDateTime(String text, InternalParser parser, Chronology chrono) {
198	188	
199	189	int newPos = requireParser(parser).parseInto(this, text, 0);
200	190	if (newPos >= 0) {
✳		@@ -207,7 +197,7 @@ LocalDateTime getLocalDateTime(String text, InternalParser parser, Chronology ch
207	197	throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
208	198	}
209	199	
210	-	LocalDateTime getLocalDateTime(String text, Chronology chrono) {
	200	+ private LocalDateTime getLocalDateTime(String text, Chronology chrono) {
211	201	long millis = computeMillis(true, text);
212	202	if (getOffsetInteger() != null) { // treat withOffsetParsed() as being true
213	203	int parsedOffset = getOffsetInteger();
✳		@@ -219,7 +209,7 @@ LocalDateTime getLocalDateTime(String text, Chronology chrono) {
219	209	return new LocalDateTime(millis, chrono);
220	210	}
221	211	
222	-	MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, InternalParser parser, Ch
	212	+ private MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, InternalParser pai
223	213	
224	214	int newPos = requireParser(parser).parseInto(this, text, 0);
225	215	if (newPos >= 0) {
✳		@@ -232,12 +222,20 @@ MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, St
232	222	throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
233	223	}
234	224	
235	-	MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, Chronology chrono) {
	225	+ private MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, Chronology chrono;
236	226	return getMutableDateTime(iZone, computeMillis(true, text), getChronology(iOffsetParsed, chrono));
237	227	}
238	228	

Figure C.65: insert caption

229	229	+	static MutableDateTime getMutableDateTime(DateTimeZone iZone, long l, Chronology chronology) {
230	230	+	MutableDateTime dt = new MutableDateTime(l, chronology);
231	231	+	if (iZone != null) {
232	232	+	dt.setZone(iZone);
233	233	+	}
234	234	+	return dt;
235	235	+	}
236	236	+	}
239	237	-	DateTime getDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, final InternalParser parser, Chronology
240	238	+	private DateTime getDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, final InternalParser parser, Chronology
241	239		int newPos = requireParser(parser).parseInto(this, text, 0);
242	240		if (newPos >= 0) {
243	241		if (newPos >= text.length()) {
✱			@@ -249,7 +247,7 @@ DateTime getDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, fin
249	247		throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
250	248		}
251	249		}
252		-	Chronology getChronology(boolean iOffsetParsed, Chronology chrono) {
250	250	+	private Chronology getChronology(boolean iOffsetParsed, Chronology chrono) {
253	251		if (iOffsetParsed && getOffsetInteger() != null) {
254	252		int parsedOffset = getOffsetInteger();
255	253		DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
✱			

Figure C.66: insert caption

This helped me identify that the best approach for getting to the desired architecture was to split apart the functionality that had migrated into `DateTimeParserBucket` into another collaborating object.

broke out delegate for parsing.
[Browse files](#)

move Chronology related stuff to ChronologyFactory.

fix-bug-86-experimental-narrative

 V committed 27 days ago
 1 parent 8180ce6 commit e2b0b5cf38a47045c908e01c0c9db6a2a574beed

Showing 4 changed files with 140 additions and 145 deletions.
 Unified Split

6







 src/main/java/org/joda/time/format/ChronologyFactory.java
 View

		@@ -24,4 +24,10 @@ static Chronology getChronologyWithDefaultValue(Chronology defaultChronology, Ch
24	24	static Chronology getChronologyWithTimeZone(Chronology chrono, DateTimeZone defaultTimeZone) {
25	25	return (defaultTimeZone != null) ? chrono.withZone(defaultTimeZone) : chrono;
26	26	}
	27	+
	28	static Chronology getChronology(boolean iOffsetParsed, Chronology chrono, Integer offsetInteger, DateTimeZone zone)
	29	return (iOffsetParsed && offsetInteger != null) ?
	30	chrono.withZone(DateTimeZone.forOffsetMillis(offsetInteger)) :
	31	(zone != null) ? chrono.withZone(zone) : chrono;
	32	+
	33	}
27	33	} 

10







 src/main/java/org/joda/time/format/DateTimeFormatter.java
 View

		@@ -732,7 +732,7 @@ private InternalPrinter requirePrinter() {
		* @throws IllegalArgumentException if any field is out of range
		*/
732	732	public int parseInto(ReadWritableInstant instant, String text, int position) {
733	733	
734	734	
735	-	return DateTimeParserBucket.parseIntoReadWriteableInstant(iChrono, iLocale, iOffsetParsed, iPivotYear, iZone, i
	735	+ return SimpleParser.parseIntoReadWriteableInstant(iChrono, iLocale, iOffsetParsed, iPivotYear, iZone, instant,
	736	}
737	737	
738	738	/**
		@@ -748,7 +748,7 @@ public int parseInto(ReadWritableInstant instant, String text, int position) {
		* @throws IllegalArgumentException if the text to parse is invalid
		*/
748	748	public long parseMillis(String text) {
749	749	
750	750	
751	-	return DateTimeParserBucket.parseMillis(iChrono, iDefaultYear, iLocale, iPivotYear, iZone, text, this.iParser);
	751	+ return SimpleParser.parseMillis(iChrono, iDefaultYear, iLocale, iPivotYear, iZone, text, this.iParser);
752	752	}

Figure C.67: insert caption

753	753	
754	754	/**
✚		@@ -802,7 +802,7 @@ public LocalTime parseLocalTime(String text) {
802	802	* @since 2.0
803	803	*/
804	804	public LocalDateTime parseLocalDateTime(String text) {
805		- return DateTimeParserBucket.parseLocalDateTime(iChrono, iDefaultYear, iLocale, iPivotYear, iZone, text, this.iP
	805	+ return SimpleParser.parseLocalDateTime(iChrono, iDefaultYear, iLocale, iPivotYear, iZone, text, this.iParser);
806	806	}
807	807	
808	808	/**
✚		@@ -823,7 +823,7 @@ public LocalDateTime parseLocalDateTime(String text) {
823	823	* @throws IllegalArgumentException if the text to parse is invalid
824	824	*/
825	825	public DateTime parseDateTime(String text) {
826		- return DateTimeParserBucket.parseDateTime(iChrono, iDefaultYear, iLocale, iOffsetParsed, iPivotYear, iZone, tex
	826	+ return SimpleParser.parseDateTime(iChrono, iDefaultYear, iLocale, iOffsetParsed, iPivotYear, iZone, text, this.
827	827	}
828	828	
829	829	/**
✚		@@ -844,7 +844,7 @@ public DateTime parseDateTime(String text) {
844	844	* @throws IllegalArgumentException if the text to parse is invalid
845	845	*/
846	846	public MutableDateTime parseMutableDateTime(String text) {
847		- return DateTimeParserBucket.parseMutableDateTime(iChrono, iDefaultYear, iLocale, iOffsetParsed, iPivotYear, iZo
	847	+ return SimpleParser.parseMutableDateTime(iChrono, iDefaultYear, iLocale, iOffsetParsed, iPivotYear, iZone, text
848	848	}
849	849	
850	850	//-----
✚		

Figure C.68: insert caption

140
src/main/java/org/joda/time/format/DateTimeParserBucket.java
View

```

129 129      iSavedFields = new SavedField[8];
130 130  }
131 131
132 - static int parseIntoReadWriteableInstant(Chronology iChrono, Locale iLocale, boolean iOffsetParsed, Integer iPivotY
133 -     if (instant == null) {
134 -         throw new IllegalArgumentException("Instant must not be null");
135 -     }
136 -     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, instant.getChronology());
137 -     long millis = instant.getMillis() + instant.getChronology().getZone().getOffset(instant.getMillis());
138 -     int defaultYear = DateTimeUtils.getChronology(instant.getChronology()).year().get(instant.getMillis());
139 -     DateTimeParserBucket bucket = new DateTimeParserBucket(millis, chrono, iLocale, iPivotYear, defaultYear);
140 -     return bucket.parseIntoInstantAndGetPosition(iOffsetParsed, iZone, instant, text, position, parser);
141 - }
142 -
143 - private int parseIntoInstantAndGetPosition(boolean iOffsetParsed, DateTimeZone iZone, ReadWritableInstant instant,
144 -     int newPos = requireParser(parser).parseInto(this, text, position);
145 -     instant.update(iZone, computeMillis(false, text), getChronology(iOffsetParsed, iChrono));
146 -     return newPos;
147 - }
148 -
149 - static long parseMillis(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZon
150 -     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, iChrono);
151 -     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
152 -     return bucket.getMillis(text, parser);
153 - }
154 -
155 - static LocalDateTime parseLocalDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, D
156 -     Chronology chronology = ChronologyFactory.selectChronology(iChrono, iZone, null);
157 -     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chronology.withUTC(), iLocale, iPivotYear, iDefaultYe
158 -     return bucket.getLocalDateTime(text, parser, chronology.withUTC());
159 - }
160 -
161 - static DateTime parseDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, boolean iOffsetParsed, Integer
162 -     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
163 -     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
164 -     return bucket.getDateTime(iOffsetParsed, iZone, text, parser, chrono);
165 - }
166 -
167 - static MutableDateTime parseMutableDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, boolean iOffsetPa
168 -     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
169 -     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
170 -     return bucket.getMutableDateTime(iOffsetParsed, iZone, text, parser, chrono);

```

Figure C.69: insert caption

```

171 - }
172 -
173 - /**
174 -  * Checks whether parsing is supported.
175 -  *
176 -  * @throws UnsupportedOperationException if parsing is not supported
177 -  * @param iParser
178 -  */
179 - private static InternalParser requireParser(InternalParser iParser) {
180 -     InternalParser parser = iParser;
181 -     if (parser == null) {
182 -         throw new UnsupportedOperationException("Parsing not supported");
183 -     }
184 -     return parser;
185 - }
186 -
187 - private LocalDateTime getLocalDateTime(String text, InternalParser parser, Chronology chrono) {
188 -
189 -     int newPos = requireParser(parser).parseInto(this, text, 0);
190 -     if (newPos >= 0) {
191 -         if (newPos >= text.length()) {
192 -             return getLocalDateTime(text, chrono);
193 -         }
194 -     } else {
195 -         newPos = ~newPos;
196 -     }
197 -     throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
198 - }
199 -
200 - private LocalDateTime getLocalDateTime(String text, Chronology chrono) {
201 -     long millis = computeMillis(true, text);
202 -     if (getOffsetInteger() != null) { // treat withOffsetParsed() as being true
203 -         int parsedOffset = getOffsetInteger();
204 -         DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
205 -         chrono = chrono.withZone(parsedZone);
206 -     } else if (getZone() != null) {
207 -         chrono = chrono.withZone(getZone());
208 -     }
209 -     return new LocalDateTime(millis, chrono);
210 - }
211 -
212 - private MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, InternalParser p
213 -
214 -     int newPos = requireParser(parser).parseInto(this, text, 0);
215 -     if (newPos >= 0) {
216 -         if (newPos >= text.length()) {

```

Figure C.70: insert caption

```

217 -         return getMutableDateTime(iOffsetParsed, iZone, text, chrono);
218 -     }
219 - } else {
220 -     newPos = ~newPos;
221 - }
222 - throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
223 - }
224 -
225 - private MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, Chronology chrono) {
226 -     return getMutableDateTime(iZone, computeMillis(true, text), getChronology(iOffsetParsed, chrono));
227 - }
228 -
229 - static MutableDateTime getMutableDateTime(DateTimeZone iZone, long l, Chronology chronology) {
230 -     MutableDateTime dt = new MutableDateTime(l, chronology);
231 -     if (iZone != null) {
232 -         dt.setZone(iZone);
233 -     }
234 -     return dt;
235 - }
236 -
237 -
238 - private DateTime getDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, final InternalParser parser, C
239 -     int newPos = requireParser(parser).parseInto(this, text, 0);
240 -     if (newPos >= 0) {
241 -         if (newPos >= text.length()) {
242 -             return getDateTime(iZone, getChronology(iOffsetParsed, chrono), computeMillis(true, text));
243 -         }
244 -     } else {
245 -         newPos = ~newPos;
246 -     }
247 -     throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
248 - }
249 -
250 - private Chronology getChronology(boolean iOffsetParsed, Chronology chrono) {
251 -     if (iOffsetParsed && getOffsetInteger() != null) {
252 -         int parsedOffset = getOffsetInteger();
253 -         DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
254 -         chrono = chrono.withZone(parsedZone);
255 -     } else if (getZone() != null) {
256 -         chrono = chrono.withZone(getZone());
257 -     }
258 -     return chrono;
259 - }
260 -
261 - private static DateTime getDateTime(DateTimeZone iZone, Chronology chronology, long millis) {
262 -     DateTime dt = new DateTime(millis, chronology);

```

Figure C.71: insert caption

263		-	if (iZone != null) {
264		-	dt = dt.withZone(iZone);
265		-	}
266		-	return dt;
267		-	}
268	132		
269	133		//-----
270	134		/**
✚			@@ -636,10 +500,6 @@ private static void sort(SavedField[] array, int high) {
636	500		}
637	501		}
638	502		
639		-	long getMillis(String text, InternalParser parser) {
640		-	return doParseMillis(parser, text);
641		-	}
642		-	
643	503		class SavedState {
644	504		final DateTimeZone iZone;
645	505		final Integer iOffset;
✚			

129		src/main/java/org/joda/time/format/SimpleParser.java		View
...	...		@@ -0,0 +1,129 @@	
	1	+	package org.joda.time.format;	
	2	+		
	3	+	import org.joda.time.*;	
	4	+		
	5	+	import java.util.Locale;	
	6	+		
	7	+	final class SimpleParser {	
	8	+		
	9	+	static int parseIntoReadableInstant(Chronology iChrono, Locale iLocale, boolean iOffsetParsed, Integer iPivotY	
	10	+	if (instant == null) {	
	11	+	throw new IllegalArgumentException("Instant must not be null");	
	12	+	}	
	13	+	Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, instant.getChronology());	
	14	+	long millis = instant.getMillis() + instant.getChronology().getZone().getOffset(instant.getMillis());	
	15	+	int defaultYear = DateTimeUtils.getChronology(instant.getChronology()).year().get(instant.getMillis());	
	16	+	DateTimeParserBucket bucket = new DateTimeParserBucket(millis, chrono, iLocale, iPivotYear, defaultYear);	
	17	+	int newPos = requireParser(parser).parseInto(bucket, text, position);	
	18	+	instant.update(iZone, bucket.computeMillis(false, text), ChronologyFactory.getChronology(iOffsetParsed, bucket.	
	19	+	return newPos;	
	20	+	}	

Figure C.72: insert caption

```

21 +
22 + static long parseMillis(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZone) {
23 +     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, iChrono);
24 +     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
25 +     return bucket.doParseMillis(parser, text);
26 + }
27 +
28 + static LocalDateTime parseLocalDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZone) {
29 +     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
30 +     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono.withUTC(), iLocale, iPivotYear, iDefaultYear);
31 +     return getLocalDateTime(bucket, text, parser, chrono.withUTC());
32 + }
33 +
34 + static DateTime parseDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, boolean iOffsetParsed, Integer iPivotYear, DateTimeZone iZone) {
35 +     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
36 +     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
37 +     return getDateTime(bucket, iOffsetParsed, iZone, text, parser, chrono);
38 + }
39 +
40 + static MutableDateTime parseMutableDateTime(Chronology iChrono, int iDefaultYear, Locale iLocale, boolean iOffsetParsed, Integer iPivotYear, DateTimeZone iZone) {
41 +     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);
42 +     DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear);
43 +     return getMutableDateTime(bucket, iOffsetParsed, iZone, text, parser, chrono);
44 + }
45 +
46 + /**
47 +  * Checks whether parsing is supported.
48 +  *
49 +  * @param iParser
50 +  * @throws UnsupportedOperationException if parsing is not supported
51 +  */
52 + static InternalParser requireParser(InternalParser iParser) {
53 +     InternalParser parser = iParser;
54 +     if (parser == null) {
55 +         throw new UnsupportedOperationException("Parsing not supported");
56 +     }
57 +     return parser;
58 + }
59 +
60 + static LocalDateTime getLocalDateTime(DateTimeParserBucket bucket, String text, InternalParser parser, Chronology chrono) {
61 +
62 +     int newPos = requireParser(parser).parseInto(bucket, text, 0);
63 +     if (newPos >= 0) {
64 +         if (newPos >= text.length()) {
65 +             return getLocalDateTime(bucket, text, chrono);
66 +         }

```

Figure C.73: insert caption

```

67 +     } else {
68 +         newPos = ~newPos;
69 +     }
70 +     throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
71 + }
72 +
73 + static LocalDateTime getLocalDateTime(DateTimeParserBucket bucket, String text, Chronology chrono) {
74 +     long millis = bucket.computeMillis(true, text);
75 +     if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() as being true
76 +         int parsedOffset = bucket.getOffsetInteger();
77 +         DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
78 +         chrono = chrono.withZone(parsedZone);
79 +     } else if (bucket.getZone() != null) {
80 +         chrono = chrono.withZone(bucket.getZone());
81 +     }
82 +     return new LocalDateTime(millis, chrono);
83 + }
84 +
85 + static MutableDateTime getMutableDateTime(DateTimeParserBucket bucket, boolean iOffsetParsed, DateTimeZone iZone, S
86 +
87 +     int newPos = requireParser(parser).parseInto(bucket, text, 0);
88 +     if (newPos >= 0) {
89 +         if (newPos >= text.length()) {
90 +             return getMutableDateTime(bucket, iOffsetParsed, iZone, text, chrono);
91 +         }
92 +     } else {
93 +         newPos = ~newPos;
94 +     }
95 +     throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
96 + }
97 +
98 + static MutableDateTime getMutableDateTime(DateTimeParserBucket bucket, boolean iOffsetParsed, DateTimeZone iZone, S
99 +     return getMutableDateTime(iZone, bucket.computeMillis(true, text), ChronologyFactory.getChronology(iOffsetParse
100 + }
101 +
102 + static MutableDateTime getMutableDateTime(DateTimeZone iZone, long l, Chronology chronology) {
103 +     MutableDateTime dt = new MutableDateTime(l, chronology);
104 +     if (iZone != null) {
105 +         dt.setZone(iZone);
106 +     }
107 +     return dt;
108 + }
109 +
110 + static DateTime getDateTime(DateTimeParserBucket bucket, boolean iOffsetParsed, DateTimeZone iZone, String text, fi
111 +     int newPos = requireParser(parser).parseInto(bucket, text, 0);
112 +     if (newPos >= 0) {

```

Figure C.74: insert caption

```

113 +         if (newPos >= text.length()) {
114 +             return getDateTime(iZone, ChronologyFactory.getChronology(iOffsetParsed, chrono, bucket.getOffsetInteger(), buck
115 +         }
116 +     } else {
117 +         newPos = ~newPos;
118 +     }
119 +     throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
120 + }
121 +
122 + static DateTime getDateTime(DateTimeZone iZone, Chronology chronology, long millis) {
123 +     DateTime dt = new DateTime(millis, chronology);
124 +     if (iZone != null) {
125 +         dt = dt.withZone(iZone);
126 +     }
127 +     return dt;
128 + }
129 +}

```

Figure C.75: insert caption

With the new delegate `SimpleParser` in place, I had a target to move towards the desired TEMPLATE METHOD hierarchy. First, I started by inlining some meth-

ods I'd previously broken out. Refactoring operations are often well-known to be reversible, as exploring in one direction can lead to realization that the best path was another. I also applying some EXTRACT PARAMETER to reduce the length of method signatures, as the cognitive complexity of methods rises with the length its signature.

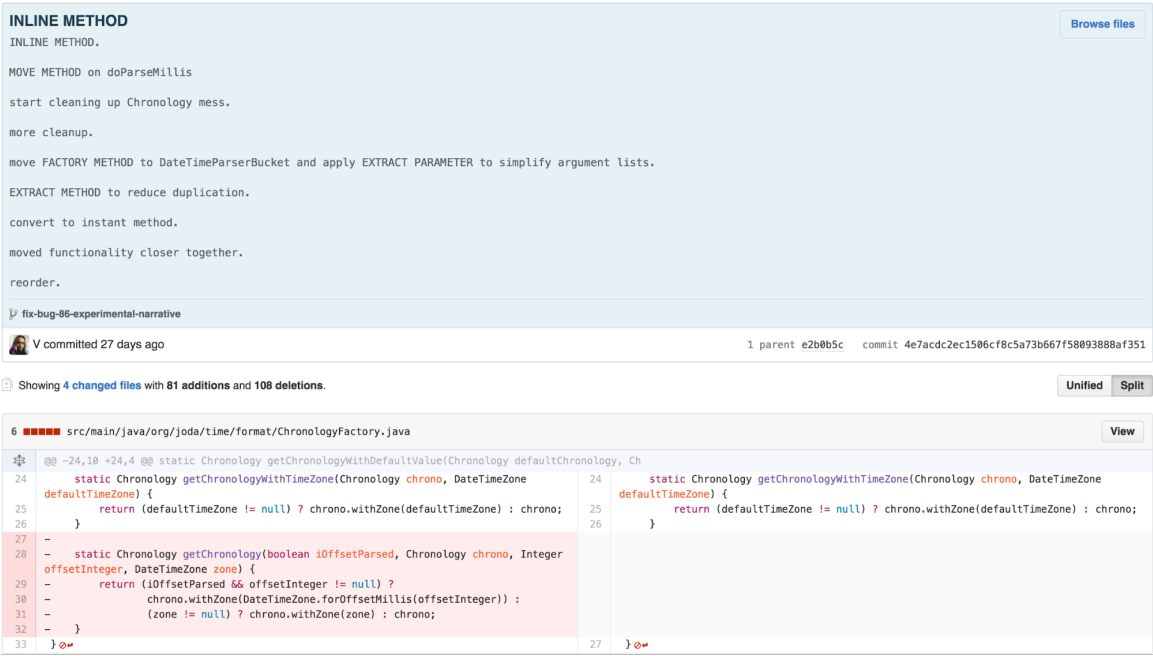


Figure C.76: insert caption

10  src/main/java/org/joda/time/format/DateTimeFormatter.java		View	
<pre> 732 * @throws IllegalArgumentException if any field is out of range 733 */ 734 public int parseInto(ReadWritableInstant instant, String text, int position) { 735 - return SimpleParser.parseIntoReadWritableInstant(iChrono, iLocale, iOffsetParsed, iPivotYear, iZone, instant, text, position, iParser); </pre>		<pre> 732 * @throws IllegalArgumentException if any field is out of range 733 */ 734 public int parseInto(ReadWritableInstant instant, String text, int position) { 735 + return SimpleParser.parseIntoReadWritableInstant(iOffsetParsed, iZone, instant, text, position, iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iLocale, iPivotYear, iZone, instant)); </pre>	
<pre> 736 } 737 /** 738 */ </pre>		<pre> 736 } 737 /** 738 */ </pre>	
<pre> 748 * @throws IllegalArgumentException if the text to parse is invalid 749 */ 750 public long parseMillis(String text) { 751 - return SimpleParser.parseMillis(iChrono, iDefaultYear, iLocale, iPivotYear, iZone, text, this.iParser); </pre>		<pre> 748 * @throws IllegalArgumentException if the text to parse is invalid 749 */ 750 public long parseMillis(String text) { 751 + return SimpleParser.parseMillis(text, this.iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale, iPivotYear, iZone)); </pre>	
<pre> 752 } 753 /** 754 */ </pre>		<pre> 752 } 753 /** 754 */ </pre>	
<pre> 802 * @since 2.0 803 */ 804 public LocalDateTime parseLocalDateTime(String text) { 805 - return SimpleParser.parseLocalDateTime(iChrono, iDefaultYear, iLocale, iPivotYear, iZone, text, this.iParser); </pre>		<pre> 802 * @since 2.0 803 */ 804 public LocalDateTime parseLocalDateTime(String text) { 805 + return SimpleParser.parseLocalDateTime(text, this.iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale, iPivotYear, iZone)); </pre>	
<pre> 806 } 807 /** 808 */ </pre>		<pre> 806 } 807 /** 808 */ </pre>	
<pre> 823 * @throws IllegalArgumentException if the text to parse is invalid 824 */ 825 public DateTime parseDateTime(String text) { 826 - return SimpleParser.parseDateTime(iChrono, iDefaultYear, iLocale, iOffsetParsed, iPivotYear, iZone, text, this.iParser); </pre>		<pre> 823 * @throws IllegalArgumentException if the text to parse is invalid 824 */ 825 public DateTime parseDateTime(String text) { 826 + return SimpleParser.parseDateTime(iOffsetParsed, iZone, text, this.iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale, iPivotYear, iZone)); </pre>	
<pre> 827 } 828 /** 829 */ </pre>		<pre> 827 } 828 /** 829 */ </pre>	

Figure C.77: insert caption

<pre> 844 * @throws IllegalArgumentException if the text to parse is invalid 845 */ 846 public MutableDateTime parseMutableDateTime(String text) { 847 - return SimpleParser.parseMutableDateTime(iChrono, iDefaultYear, iLocale, iOffsetParsed, iPivotYear, iZone, text, this.iParser); </pre>		<pre> 844 * @throws IllegalArgumentException if the text to parse is invalid 845 */ 846 public MutableDateTime parseMutableDateTime(String text) { 847 + return SimpleParser.parseMutableDateTime(iOffsetParsed, iZone, text, this.iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale, iPivotYear, iZone)); </pre>	
<pre> 848 } 849 //----- 850 */ </pre>		<pre> 848 } 849 //----- 850 */ </pre>	

40  src/main/java/org/joda/time/format/DateTimeParserBucket.java		View	
<pre> 129 @ -129,6 +129,30 @ public DateTimeParserBucket(long instantLocal, Chronology chrono, 130 iSavedFields = new SavedField[8]; 131 } </pre>		<pre> 129 iSavedFields = new SavedField[8]; 130 } 131 } </pre>	
<pre> 132 + static DateTimeParserBucket getDateTimeParserBucket(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZone) { 133 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, iChrono); 134 + return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear); 135 + } 136 + 137 + static DateTimeParserBucket getDateTimeParserBucket(Chronology iChrono, Locale iLocale, Integer iPivotYear, DateTimeZone iZone, ReadWritableInstant instant) { 138 + if (instant == null) { 139 + throw new IllegalArgumentException("Instant must not be null"); 140 + } 141 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, instant.getChronology()); 142 + long millis = instant.getMillis() + instant.getChronology().getZone().getOffset(instant.getMillis()); 143 + int defaultYear = DateTimeUtils.getChronology(instant.getChronology()).year().get(instant.getMillis()); 144 + return new DateTimeParserBucket(millis, chrono, iLocale, iPivotYear, defaultYear); 145 + } 146 + 147 + Chronology getBucketChronology(boolean iOffsetParsed) { 148 + Chronology chrono = getChronology(); 149 + Integer offsetInteger = getOffsetInteger(); 150 + DateTimeZone zone = getZone(); </pre>		<pre> 132 + static DateTimeParserBucket getDateTimeParserBucket(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZone) { 133 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, iChrono); 134 + return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear, iDefaultYear); 135 + } 136 + 137 + static DateTimeParserBucket getDateTimeParserBucket(Chronology iChrono, Locale iLocale, Integer iPivotYear, DateTimeZone iZone, ReadWritableInstant instant) { 138 + if (instant == null) { 139 + throw new IllegalArgumentException("Instant must not be null"); 140 + } 141 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, instant.getChronology()); 142 + long millis = instant.getMillis() + instant.getChronology().getZone().getOffset(instant.getMillis()); 143 + int defaultYear = DateTimeUtils.getChronology(instant.getChronology()).year().get(instant.getMillis()); 144 + return new DateTimeParserBucket(millis, chrono, iLocale, iPivotYear, defaultYear); 145 + } 146 + 147 + Chronology getBucketChronology(boolean iOffsetParsed) { 148 + Chronology chrono = getChronology(); 149 + Integer offsetInteger = getOffsetInteger(); 150 + DateTimeZone zone = getZone(); </pre>	

Figure C.78: insert caption

		151 + return (offsetParsed && offsetInteger != null) ?	
		152 + chrono.withZone(DateTimeZone.forOffsetMillis(offsetInteger)) :	
		153 + (zone != null) ? chrono.withZone(zone) : chrono;	
		154 + }	
		155 +	
132	//-----	156	
133	/**	157	//-----
134	*/	158	/**
135	@@ -165,19 +189,9 @@ public void reset() {		
165	*/	189	*/
166	public long parseMillis(DateTimeParser parser, CharSequence text) {	190	public long parseMillis(DateTimeParser parser, CharSequence text) {
167	reset();	191	reset();
168	- return doParseMillis(DateTimeParserInternalParser.of(parser), text);	192	+ return SimpleParser.parseMillis(
169	- }	193	+ text,
170	- }	194	+ DateTimeParserInternalParser.of(parser),
			getTimeParserBucket(iChrono, iDefaultYear, iLocale, iPivotYear, iZone));
171	- long doParseMillis(InternalParser parser, CharSequence text) {		
172	- int newPos = parser.parseInto(this, text, 0);		
173	- if (newPos >= 0) {		
174	- if (newPos >= text.length()) {		
175	- return computeMillis(true, text);		
176	- }		
177	- } else {		
178	- newPos = ~newPos;		
179	- }		
180	- throw new		
	IllegalArgumentException(FormatUtils.createErrorMessage(text.toString(), newPos));		
181	- }	195	}
182	//-----	196	
183	/**	197	//-----
133	src/main/java/org/joda/time/format/SimpleParser.java		
134	@@ -2,128 +2,93 @@		
2		2	
3	import org.joda.time.*;	3	import org.joda.time.*;
4		4	
5	-import java.util.Locale;		
6	-		
7	final class SimpleParser {	5	final class SimpleParser {

Figure C.79: insert caption

9	- static int parseIntoReadableInstant(Chronology iChrono, Locale iLocale,	7	+ static int parseIntoReadableInstant(boolean iOffsetParsed, DateTimeZone
	boolean iOffsetParsed, Integer iPivotYear, DateTimeZone iZone, ReadWritableInstant		iZone, ReadWritableInstant instant, String text, int position, InternalParser parser,
	instant, String text, int position, InternalParser parser) {		DateTimeParserBucket bucket) {
10	- if (instant == null) {	8	+ if (parser == null) {
11	- throw new IllegalArgumentException("Instant must not be null");	9	+ throw new UnsupportedOperationException("Parsing not supported");
12	- }	10	- }
13	- Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone,	11	+ int newPos = parser.parseInto(bucket, text, position);
	instant.getChronology());	12	+ instant.update(iZone, bucket.computeMillis(false, text),
14	- long millis = instant.getMillis() +		bucket.getBucketChronology(iOffsetParsed));
	instant.getChronology().getZone().getOffset(instant.getMillis());		
15	- int defaultYear =		
	DateTimeUtils.getChronology(instant.getChronology()).year().get(instant.getMillis());		
16	- DateTimeParserBucket bucket = new DateTimeParserBucket(millis, chrono,		
	iLocale, iPivotYear, defaultYear);		
17	- int newPos = requireParser(parser).parseInto(bucket, text, position);		
18	- instant.update(iZone, bucket.computeMillis(false, text),		
	ChronologyFactory.getChronology(iOffsetParsed, bucket.getChronology(),		
	bucket.getOffsetInteger(), bucket.getZone());		
19	- return newPos;	13	- return newPos;
20	- }	14	- }
21		15	
22	- static long parseMillis(Chronology iChrono, int iDefaultYear, Locale iLocale,	16	+ static long parseMillis(CharSequence text, InternalParser parser,
	Integer iPivotYear, DateTimeZone iZone, String text, InternalParser parser) {		DateTimeParserBucket bucket) {
23	- Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone,		
	iChrono);		
24	- DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale,		
	iPivotYear, iDefaultYear);		
25	- return bucket.doParseMillis(parser, text);		
26	- }		
27	-		
28	- static LocalDateTime parseLocalDateTime(Chronology iChrono, int iDefaultYear,		
	Locale iLocale, Integer iPivotYear, DateTimeZone iZone, String text, InternalParser		
	parser) {		
29	- Chronology chronology = ChronologyFactory.selectChronology(iChrono, iZone,		
	null);		
30	- DateTimeParserBucket bucket = new DateTimeParserBucket(0,		
	chronology.withUTC(), iLocale, iPivotYear, iDefaultYear);		
31	- return getLocalDateTime(bucket, text, parser, chronology.withUTC());		
32	- }		
33	-		
34	- static DateTime parseDateTime(Chronology iChrono, int iDefaultYear, Locale		
	iLocale, boolean iOffsetParsed, Integer iPivotYear, DateTimeZone iZone, String text,		
	InternalParser parser) {		
35	- Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null);		
36	- DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale,		
	iPivotYear, iDefaultYear);		

Figure C.80: insert caption

<pre> 37 return getDateTime(bucket, iOffsetParsed, iZone, text, parser, chrono); 38 } 39 40 static MutableDateTime parseMutableDateTime(Chronology iChrono, int iDefaultYear, 41 Locale iLocale, boolean iOffsetParsed, Integer iPivotYear, DateTimeZone iZone, String 42 text, InternalParser parser) { 43 Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, null); 44 DateTimeParserBucket bucket = new DateTimeParserBucket(0, chrono, iLocale, 45 iPivotYear, iDefaultYear); 46 return getMutableDateTime(bucket, iOffsetParsed, iZone, text, parser, 47 chrono); 48 } 49 50 /** 51 * Checks whether parsing is supported. 52 * 53 * @param iParser 54 * @throws UnsupportedOperationException if parsing is not supported 55 */ 56 static InternalParser requireParser(InternalParser iParser) { 57 InternalParser parser = iParser; 58 if (parser == null) { 59 throw new UnsupportedOperationException("Parsing not supported"); 60 } 61 return parser; 62 } </pre>	<pre> 17 if (parser == null) { 18 throw new UnsupportedOperationException("Parsing not supported"); 19 } 20 21 int newPos = parser.parseInto(bucket, text, 0); 22 if (newPos >= 0) { 23 if (newPos >= text.length()) { 24 return bucket.computeMillis(true, text); 25 } else { 26 newPos = ~newPos; 27 } 28 throw new 29 IllegalArgumentException(FormatUtils.createErrorMessage(text.toString(), newPos)); 30 } 31 32 static LocalDateTime parseLocalDateTime(CharSequence text, InternalParser parser, 33 DateTimeParserBucket bucket) { 34 if (parser == null) { 35 throw new UnsupportedOperationException("Parsing not supported"); 36 } 37 38 int newPos = parser.parseInto(bucket, text, 0); 39 if (newPos >= 0) { 40 if (newPos >= text.length()) { 41 long millis = bucket.computeMillis(true, text); </pre>
---	---

Figure C.81: insert caption

<pre> 66 } 67 } else { 68 newPos = ~newPos; 69 } 70 throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, 71 newPos)); 72 } 73 74 static LocalDateTime getLocalDateTime(DateTimeParserBucket bucket, String text, 75 Chronology chrono) { 76 long millis = bucket.computeMillis(true, text); 77 if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() as 78 being true 79 int parsedOffset = bucket.getOffsetInteger(); 80 DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset); 81 chrono = chrono.withZone(parsedZone); 82 } else if (bucket.getZone() != null) { 83 chrono = chrono.withZone(bucket.getZone()); 84 } 85 return new LocalDateTime(millis, chrono); 86 } 87 88 static MutableDateTime getMutableDateTime(DateTimeParserBucket bucket, boolean 89 iOffsetParsed, DateTimeZone iZone, String text, InternalParser parser, Chronology 90 chrono) { 91 int newPos = requireParser(parser).parseInto(bucket, text, 0); 92 if (newPos >= 0) { 93 if (newPos >= text.length()) { 94 return getMutableDateTime(bucket, iOffsetParsed, iZone, text, 95 chrono); </pre>	<pre> 40 Chronology chrono = bucket.getChronology(); 41 if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() 42 as being true 43 int parsedOffset = bucket.getOffsetInteger(); 44 DateTimeZone parsedZone = 45 DateTimeZone.forOffsetMillis(parsedOffset); 46 chrono = chrono.withZone(parsedZone); 47 } else if (bucket.getZone() != null) { 48 chrono = chrono.withZone(bucket.getZone()); 49 } 50 return new LocalDateTime(millis, chrono); 51 } else { 52 newPos = ~newPos; 53 } 54 throw new 55 IllegalArgumentException(FormatUtils.createErrorMessage(text.toString(), newPos)); 56 } 57 58 static DateTime parseDateTime(boolean iOffsetParsed, DateTimeZone iZone, String 59 text, InternalParser parser, DateTimeParserBucket bucket) { 60 if (parser == null) { 61 throw new UnsupportedOperationException("Parsing not supported"); 62 } 63 64 int newPos = parser.parseInto(bucket, text, 0); 65 66 if (newPos >= 0) { 67 if (newPos >= text.length()) { 68 DateTime dt = new DateTime(bucket.computeMillis(true, text), 69 bucket.getBucketChronology(iOffsetParsed)); 70 if (iZone != null) { 71 dt = dt.withZone(iZone); 72 } 73 return dt; </pre>
---	--

Figure C.82: insert caption

91	}	68	}
92	} else {	69	} else {
93	newPos = ~newPos;	70	newPos = ~newPos;
94	}	71	}
95	throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,	72	throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,
96	newPos));	73	newPos));
97	}	74	}
98	- static MutableDateTime getMutableDateTime(DateTimeParserBucket bucket, boolean	75	+ static MutableDateTime parseMutableDateTime(boolean iOffsetParsed, DateTimeZone
99	iOffsetParsed, DateTimeZone iZone, String text, Chronology chrono) {	76	iZone, String text, InternalParser parser, DateTimeParserBucket bucket) {
100	- return getMutableDateTime(iZone, bucket.computeMillis(true, text),		
101	ChronologyFactory.getChronology(iOffsetParsed, chrono, bucket.getOffsetInteger(),		
102	bucket.getZone());		
103	- }		
104	- static MutableDateTime getMutableDateTime(DateTimeZone iZone, long l, Chronology	77	+ if (parser == null) {
105	chrono) {	78	+ throw new UnsupportedOperationException("Parsing not supported");
106	- MutableDateTime dt = new MutableDateTime(l, chrono);		
107	- if (iZone != null) {		
108	- dt.setZone(iZone);		
109	- }	79	}
110	- return dt;	80	+ int newPos = parser.parseInto(bucket, text, 0);
111	- }		
112	- static DateTime getDateTime(DateTimeParserBucket bucket, boolean iOffsetParsed,		
113	DateTimeZone iZone, String text, final InternalParser parser, Chronology chrono) {		
114	- int newPos = requireParser(parser).parseInto(bucket, text, 0);		
115	- if (newPos >= 0) {	81	if (newPos >= 0) {
116	- if (newPos >= text.length()) {	82	if (newPos >= text.length()) {
117	- return getDateTime(iZone,	83	+ MutableDateTime dt = new MutableDateTime(bucket.computeMillis(true,
118	ChronologyFactory.getChronology(iOffsetParsed, chrono, bucket.getOffsetInteger(),	84	text), bucket.getBucketChronology(iOffsetParsed));
119	bucket.getZone()), bucket.computeMillis(true, text));	85	+ if (iZone != null) {
120	- }	86	+ dt.setZone(iZone);
		87	+ return dt;
		88	}
		89	} else {
		90	newPos = ~newPos;
		91	}
		92	throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,
		93	newPos));
			}

Figure C.83: insert caption

121	-		
122	- static DateTime getDateTime(DateTimeZone iZone, Chronology		
123	chrono, long millis) {		
124	- DateTime dt = new DateTime(millis, chrono);		
125	- if (iZone != null) {		
126	- dt = dt.withZone(iZone);		
127	- }		
128	- return dt;		
129	- }	94	}

Figure C.84: insert caption

Now there was obvious duplication and an outline structure in the body of the `parse*` methods. I was now able to apply EXTRACT METHOD to identify the points that varied.

apply EXTRACT METHOD to identify duplication in methods.
[Browse files](#)

 V committed 13 days ago
 1 parent 4e7acdc commit f78249f0a9017519b3b10fbe58ea69fdf1217cad

Showing 1 changed file with 37 additions and 21 deletions.

UnifiedSplit

58

 src/main/java/org/joda/time/format/SimpleParser.java

View

		@@ -20,14 +20,18 @@ static long parseMillis(CharSequence text, InternalParser parser, DateTimeParser
20	20	int newPos = parser.parseInto(bucket, text, 0);
21	21	if (newPos >= 0) {
22	22	if (newPos >= text.length()) {
23	-	return bucket.computeMillis(true, text);
	23	+ return getMillis(text, bucket);
24	24	}
25	25	} else {
26	26	newPos = ~newPos;
27	27	}
28	28	throw new IllegalArgumentException(FormatUtils.createErrorMessage(text.toString(), newPos));
29	29	}
30	30	
	31	+ private static long getMillis(CharSequence text, DateTimeParserBucket bucket) {
	32	+ return bucket.computeMillis(true, text);
	33	+ }
	34	+ }
31	35	static LocalDateTime parseLocalDateTime(CharSequence text, InternalParser parser, DateTimeParserBucket bucket) {
32	36	if (parser == null) {
33	37	throw new UnsupportedOperationException("Parsing not supported");
		@@ -36,42 +40,50 @@ static LocalDateTime parseLocalDateTime(CharSequence text, InternalParser parser
36	40	int newPos = parser.parseInto(bucket, text, 0);
37	41	if (newPos >= 0) {
38	42	if (newPos >= text.length()) {
39	-	long millis = bucket.computeMillis(true, text);
40	-	Chronology chrono = bucket.getChronology();
41	-	if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() as being true
42	-	int parsedOffset = bucket.getOffsetInteger();
43	-	DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
44	-	chrono = chrono.withZone(parsedZone);
45	-	} else if (bucket.getZone() != null) {
46	-	chrono = chrono.withZone(bucket.getZone());
47	-	}
48	-	return new LocalDateTime(millis, chrono);

Figure C.85: insert caption

	43	+	return getLocalDateTime(text, bucket);
49	44		}
50	45		} else {
51	46		newPos = ~newPos;
52	47		}
53	48		throw new IllegalArgumentException(FormatUtils.createErrorMessage(text.toString(), newPos));
54	49		}
55	50		}
	51	+	private static LocalDateTime getLocalDateTime(CharSequence text, DateTimeParserBucket bucket) {
	52	+	long millis = getMillis(text, bucket);
	53	+	Chronology chrono = bucket.getChronology();
	54	+	if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() as being true
	55	+	int parsedOffset = bucket.getOffsetInteger();
	56	+	DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
	57	+	chrono = chrono.withZone(parsedZone);
	58	+	} else if (bucket.getZone() != null) {
	59	+	chrono = chrono.withZone(bucket.getZone());
	60	+	}
	61	+	return new LocalDateTime(millis, chrono);
	62	+	}
	63	+	}
56	64		static DateTime parseDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, InternalParser parser, DateTi
57	65		if (parser == null) {
58	66		throw new UnsupportedOperationException("Parsing not supported");
59	67		}
60	68		int newPos = parser.parseInto(bucket, text, 0);
61	69		if (newPos >= 0) {
62	70		if (newPos >= text.length()) {
63		-	DateTime dt = new DateTime(bucket.computeMillis(true, text), bucket.getBucketChronology(iOffsetParsed))
64		-	if (iZone != null) {
65		-	dt = dt.withZone(iZone);
66		-	}
67		-	return dt;
	71	+	return getDateTime(iOffsetParsed, iZone, text, bucket);
68	72		}
69	73		} else {
70	74		newPos = ~newPos;
71	75		}
72	76		throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
73	77		}
74	78		}
	79	+	private static DateTime getDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, DateTimeParserBucket bu
	80	+	DateTime dt = new DateTime(bucket.computeMillis(true, text), bucket.getBucketChronology(iOffsetParsed));
	81	+	if (iZone != null) {
	82	+	dt = dt.withZone(iZone);
	83	+	}

Figure C.86: insert caption

	84	+	return dt;
	85	+	}
	86	+	
75	87		static MutableDateTime parseMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, InternalParser
76	88		
77	89		if (parser == null) {
✚		@@ -80,15 +92,19 @@	static MutableDateTime parseMutableDateTime(boolean iOffsetParsed, DateTimeZone
80	92		int newPos = parser.parseInto(bucket, text, 0);
81	93		if (newPos >= 0) {
82	94		if (newPos >= text.length()) {
83		-	MutableDateTime dt = new MutableDateTime(bucket.computeMillis(true, text), bucket.getBucketChronology(i
84		-	if (iZone != null) {
85		-	dt.setZone(iZone);
86		-	}
87		-	return dt;
	95	+	return getMutableDateTime(iOffsetParsed, iZone, text, bucket);
88	96		}
89	97		} else {
90	98		newPos = ~newPos;
91	99		}
92	100		throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
93	101		}
	102	+	
	103	+	private static MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, DateTimePa
	104	+	MutableDateTime dt = new MutableDateTime(bucket.computeMillis(true, text), bucket.getBucketChronology(iOffsetPa
	105	+	if (iZone != null) {
	106	+	dt.setZone(iZone);
	107	+	}
	108	+	return dt;
	109	+	}
94	110		}

Figure C.87: insert caption

DRY on DateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis and deal with exceptions correctly.

EXTRACT PARAMETER.

DRY LocalDateTime.

DRY LocalDateTime.

DRY parseMillis

Figure C.88: insert caption

		740 + int newPos = iParser.parseInto(bucket, text, position);
		741 + instant.update(iZone, bucket.computeMillis(false, text),
		742 bucket.getBucketChronology(iOffsetParsed));
		743 + return newPos;
736 }	743 }	
737 /**	744 /**	
738	745	
✱ @@ -747,8 +754,14 @@ public int parseInto(ReadableInstant instant, String text, int position) {	754 {	
747 * @throws UnsupportedOperationException if parsing is not supported	755 * @throws UnsupportedOperationException if parsing is not supported	
748 * @throws IllegalArgumentException if the text to parse is invalid	756 * @throws IllegalArgumentException if the text to parse is invalid	
749 */	757 */	
750 - public long parseMillis(String text) {	757 + public long parseMillis(final String text) {	
751 - return SimpleParser.parseMillis(text, this.iParser,	758 + final DateTimeParserBucket bucket =	
DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale,	DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale,	
iPivotYear, iZone));	iPivotYear, iZone);	
	759 + final Callable<Long> callback = new Callable<Long>() {	
	760 + public Long call() throws Exception {	
	761 + return bucket.computeMillis(true, text);	
	762 + }	
	763 + }	
	764 + return SimpleParser.parseMillis(text, this.iParser, bucket, callback);	
752 }	765 }	
753 /**	766 /**	
754	767	
✱ @@ -801,8 +814,23 @@ public LocalTime parseLocalTime(String text) {	814 {	
801 * @throws IllegalArgumentException if the text to parse is invalid	815 * @throws IllegalArgumentException if the text to parse is invalid	
802 * @since 2.0	816 * @since 2.0	
803 */	817 */	
804 - public LocalDateTime parseLocalDateTime(String text) {	817 + public LocalDateTime parseLocalDateTime(final String text) {	
805 - return SimpleParser.parseLocalDateTime(text, this.iParser,	818 + final DateTimeParserBucket bucket =	
DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale,	DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale,	
iPivotYear, iZone));	iPivotYear, iZone);	
	819 + final Callable<LocalDateTime> callback = new Callable<LocalDateTime>() {	
	820 + public LocalDateTime call() throws Exception {	
	821 + long millis = bucket.computeMillis(true, text);	
	822 + Chronology chrono = bucket.getChronology();	
	823 + if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed()	
	as being true	
	824 + int parsedOffset = bucket.getOffsetInteger();	
	825 + DateTimeZone parsedZone =	
	DateTimeZone.forOffsetMillis(parsedOffset);	
	826 + chrono = chrono.withZone(parsedZone);	
	827 + } else if (bucket.getZone() != null) {	
	828 + chrono = chrono.withZone(bucket.getZone());	
	829 + }	

Figure C.89: insert caption

		830 + return new LocalDateTime(millis, chrono);
		831 + }
		832 + };
		833 + return SimpleParser.parseLocalDateTime(text, this.iParser, bucket, callback);
806 }	834 }	
807 /**	835 /**	
808	836	
✱ @@ -822,8 +850,18 @@ public LocalDateTime parseLocalDateTime(String text) {	850 {	
822 * @throws UnsupportedOperationException if parsing is not supported	851 * @throws UnsupportedOperationException if parsing is not supported	
823 * @throws IllegalArgumentException if the text to parse is invalid	852 * @throws IllegalArgumentException if the text to parse is invalid	
824 */	853 */	
825 - public DateTime parseDateTime(String text) {	853 + public DateTime parseDateTime(final String text) {	
826 - return SimpleParser.parseDateTime(iOffsetParsed, iZone, text, this.iParser,	854 + final DateTimeParserBucket bucket =	
DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale,	DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale,	
iPivotYear, iZone));	iPivotYear, iZone);	
	855 + final Callable<DateTime> callback = new Callable<DateTime>() {	
	856 + public DateTime call() throws Exception {	
	857 + DateTime dt = new DateTime(bucket.computeMillis(true, text),	
	bucket.getBucketChronology(iOffsetParsed));	
	858 + if (iZone != null) {	
	859 + dt = dt.withZone(iZone);	
	860 + }	
	861 + return dt;	
	862 + }	
	863 + }	
	864 + return SimpleParser.parseDateTime(text, this.iParser, bucket, callback);	
827 }	865 }	
828 /**	866 /**	
829	867	
✱ @@ -843,8 +881,18 @@ public DateTime parseDateTime(String text) {	881 {	
843 * @throws UnsupportedOperationException if parsing is not supported	882 * @throws UnsupportedOperationException if parsing is not supported	
844 * @throws IllegalArgumentException if the text to parse is invalid	883 * @throws IllegalArgumentException if the text to parse is invalid	
845 */	884 */	
846 - public MutableDateTime parseMutableDateTime(String text) {	884 + public MutableDateTime parseMutableDateTime(final String text) {	
847 - return SimpleParser.parseMutableDateTime(iOffsetParsed, iZone, text,	885 + final DateTimeParserBucket bucket =	
this.iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear,	DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale,	
iLocale, iPivotYear, iZone));	iPivotYear, iZone);	
	886 + final Callable<MutableDateTime> callback = new Callable<MutableDateTime>() {	
	887 + public MutableDateTime call() throws Exception {	
	888 + MutableDateTime dt = new MutableDateTime(bucket.computeMillis(true,	
	text), bucket.getBucketChronology(iOffsetParsed));	
	889 + if (iZone != null) {	
	890 + dt.setZone(iZone);	
	891 + }	
	892 + return dt;	

Figure C.90: insert caption

38	- }		
39		8	
40	- int newPos = parser.parseInto(bucket, text, 0);	9	+ static long parseMillis(final CharSequence text, final InternalParser parser, final DateTimeParserBucket bucket, Callable<Long> callback) {
41	- if (newPos >= 0) {	10	+ return getResult(text.toString(), parser, bucket, callback);
42	- if (newPos >= text.length()) {		
43	- return getLocalDateTime(text, bucket);		
44	- }		
45	- } else {		
46	- newPos = ~newPos;		
47	- }		
48	- throw new		
49	IllegalArgumentException(FormatUtils.createErrorMessage(text.toString(), newPos));		
50	- }	11	}
51	- private static LocalDateTime getLocalDateTime(CharSequence text,	12	
52	DateTimeParserBucket bucket) {	13	+ static LocalDateTime parseLocalDateTime(final CharSequence text,
53	- long millis = getMillis(text, bucket);	14	+ final InternalParser parser,
54	- Chronology chrono = bucket.getChronology();	15	+ final DateTimeParserBucket bucket,
55	- if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() as	16	+ final Callable<LocalDateTime> callback) {
56	being true	17	+ return getResult(text.toString(), parser, bucket, callback);
57	- int parsedOffset = bucket.getOffsetInteger();		
58	- DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);		
59	- chrono = chrono.withZone(parsedZone);		
60	- } else if (bucket.getZone() != null) {		
61	- chrono = chrono.withZone(bucket.getZone());		
62	- }		
63	- return new LocalDateTime(millis, chrono);	18	}
64	- }	19	
65	- static DateTime parseDateTime(boolean iOffsetParsed, DateTimeZone iZone, String	20	+ static DateTime parseDateTime(final String text,
66	text, InternalParser parser, DateTimeParserBucket bucket) {	21	+ final InternalParser parser,
67	- if (parser == null) {	22	+ final DateTimeParserBucket bucket,
68	- throw new UnsupportedOperationException("Parsing not supported");	23	+ final Callable<DateTime> callback) {
69	- }	24	+ return getResult(text, parser, bucket, callback);
70	- int newPos = parser.parseInto(bucket, text, 0);		
71	- if (newPos >= 0) {		
72	- if (newPos >= text.length()) {		
73	- return getDate(iOffsetParsed, iZone, text, bucket);		
74	- }		
75	- } else {		
76	- newPos = ~newPos;		
77	- }		
78	- throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,		
79	newPos));		
80	- }	25	}

Figure C.93: insert caption

78		26	
79	- private static DateTime getDateTime(boolean iOffsetParsed, DateTimeZone iZone,	27	+ static MutableDateTime parseMutableDateTime(final String text, InternalParser
80	String text, DateTimeParserBucket bucket) {	28	+ parser, final DateTimeParserBucket bucket, Callable<MutableDateTime> callback) {
81	- DateTime dt = new DateTime(bucket.computeMillis(true, text),		+ return getResult(text, parser, bucket, callback);
82	bucket.getBucketChronology(iOffsetParsed));		
83	- if (iZone != null) {		
84	- dt = dt.withZone(iZone);		
85	- }		
86	- return dt;	29	}
87	- }	30	
88	- static MutableDateTime parseMutableDateTime(boolean iOffsetParsed, DateTimeZone	31	+ private static <T> T getResult(String text, InternalParser parser,
89	iZone, String text, InternalParser parser, DateTimeParserBucket bucket) {	32	DateTimeParserBucket bucket, Callable<T> callable) {
90	- if (parser == null) {	33	- if (parser == null) {
91	- throw new UnsupportedOperationException("Parsing not supported");	34	- throw new UnsupportedOperationException("Parsing not supported");
92	- }	35	- }
93	- int newPos = parser.parseInto(bucket, text, 0);	36	- int newPos = parser.parseInto(bucket, text, 0);
94	- if (newPos >= 0) {	37	- if (newPos >= 0) {
95	- if (newPos >= text.length()) {	38	- if (newPos >= text.length()) {
96	- return getMutableDateTime(iOffsetParsed, iZone, text, bucket);	39	- try {
97	- }	40	- return callable.call();
98	- }	41	- } catch (Exception e) {
99	- }	42	- if (e instanceof RuntimeException) {
100	- throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,	43	- throw (RuntimeException)e;
101	newPos));	44	- }
102	- }	45	- }
103	- }	46	- } else {
104	- private static MutableDateTime getMutableDateTime(boolean iOffsetParsed,	47	- newPos = ~newPos;
105	DateTimeZone iZone, String text, DateTimeParserBucket bucket) {	48	- }
106	- MutableDateTime dt = new MutableDateTime(bucket.computeMillis(true, text),	49	- throw new IllegalArgumentException(FormatUtils.createErrorMessage(text,
107	bucket.getBucketChronology(iOffsetParsed));	50	newPos));
108	- if (iZone != null) {		- }
109	- dt.setZone(iZone);		
110	- }		
111	- return dt;		
112	- }		
113	- }	51	}

Figure C.94: insert caption

I then had a couple of options. The traditional TEMPLATE METHOD approach

would require refactoring the `SimpleParser` to be a fully-fledged object instead of a utility class full of static methods. This approach required some intensive structural modifications, though is fairly straightforward. However, it is an approach most optimally suited for a traditional object-oriented programming paradigm.

Functional Programming is experiencing a revival in the software engineering community. As Moore’s Law runs into physical nanoscale limitations, chip manufacturers have sought to improve speeds by increasing the number of cores on a chip and promoting parallel computing. Pure, stateless functions are easier to parallelize. Hence, languages like Ruby, Python, Javascript, Scala, and Clojure began supporting functional paradigms. Eventually, both Java 8 and C++ 11 added functional support features. Many young programmers learning today may have more experience with functional idioms than in the past.

One common functional idiom is the notion of a callback function. When using callbacks, one passes a function to be invoked to another function. The other function is then able to execute and invoke the passed function at its leisure. I explored this approach with a subsequent set of refactoring.

With some slight caveats to handle the difference in exception handling introduced by Java’s `Callable`, this approach compiles and executes successfully. The question is, is it simpler?

Callbacks are often argued to make code harder to understand. Some argue that they can be used effectively and comprehensibly if the number is limited. There is no data comparing the cognitive load induced by following such an approach versus traditional OOP. This seems like ground rife for future research.

Returning our attention to simplifying the usage patterns, an obvious opportunity to reduce the Split Attention Effect is to use constructor chaining when the initialization semantics are the same and the only difference is a variable number of arguments. Hence, we can simplify one of the `DateTimeFormatter` constructors as follows.

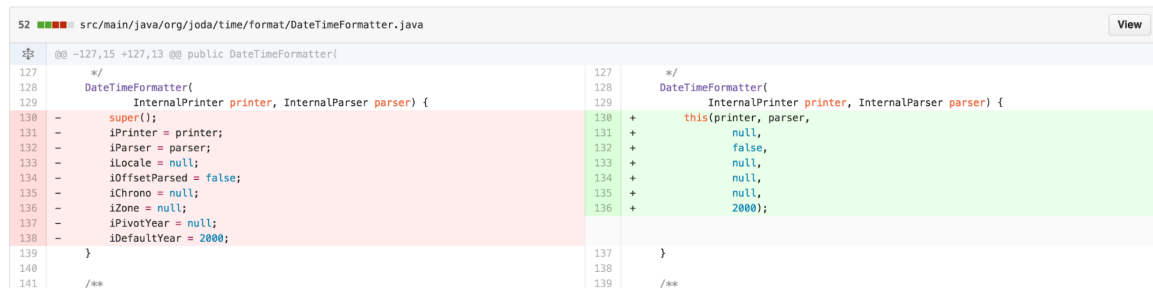


Figure C.95: insert caption

We can also significantly simplify the signatures of `SimpleParser` when we realize that most of its arguments are pulled from `DateTimeFormatter`.

747	@@ -747,8 +760,8 @@ public int parseInto(ReadableInstant instant, String text, int position) {	760	* @throws UnsupportedOperationException if parsing is not supported
748	* @throws IllegalArgumentException if the text to parse is invalid	761	* @throws IllegalArgumentException if the text to parse is invalid
749	*/	762	*/
750	- public long parseMillis(String text) {	763	+ public long parseMillis(final String text) {
751	- return SimpleParser.parseMillis(text, this.iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale, iPivotYear, iZone));	764	+ return SimpleParser.parseMillisFrom(this, text);
752	}	765	}
753	/**	766	/**
754	@@ -801,8 +814,8 @@ public LocalTime parseLocalTime(String text) {	767	/**
801	* @throws IllegalArgumentException if the text to parse is invalid	814	* @throws IllegalArgumentException if the text to parse is invalid
802	* @since 2.0	815	* @since 2.0
803	*/	816	*/
804	- public LocalDateTime parseLocalDateTime(String text) {	817	+ public LocalDateTime parseLocalDateTime(final String text) {
805	- return SimpleParser.parseLocalDateTime(text, this.iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale, iPivotYear, iZone));	818	+ return SimpleParser.parseIntoLocalDateTime(this, text);
806	}	819	}
807	/**	820	/**
808	@@ -822,8 +835,8 @@ public LocalDateTime parseLocalDateTime(String text) {	821	/**
822	* @throws UnsupportedOperationException if parsing is not supported	835	* @throws UnsupportedOperationException if parsing is not supported
823	* @throws IllegalArgumentException if the text to parse is invalid	836	* @throws IllegalArgumentException if the text to parse is invalid
824	*/	837	*/
825	- public DateTime parseDateTime(String text) {	838	+ public DateTime parseDateTime(final String text) {
826	- return SimpleParser.parseDateTime(iOffsetParsed, iZone, text, this.iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale, iPivotYear, iZone));	839	+ return SimpleParser.getDateTime(this, text);
827	}	840	}
828	/**	841	/**
829	@@ -843,10 +856,7 @@ public DateTime parseDateTime(String text) {	842	/**
843	* @throws UnsupportedOperationException if parsing is not supported	856	* @throws UnsupportedOperationException if parsing is not supported
844	* @throws IllegalArgumentException if the text to parse is invalid	857	* @throws IllegalArgumentException if the text to parse is invalid
845	*/	858	*/
846	- public MutableDateTime parseMutableDateTime(String text) {	859	+ public MutableDateTime parseMutableDateTime(final String text) {
847	- return SimpleParser.parseMutableDateTime(iOffsetParsed, iZone, text, this.iParser, DateTimeParserBucket.getDateTimeParserBucket(iChrono, iDefaultYear, iLocale, iPivotYear, iZone));	860	+ return SimpleParser.getMutableDateTime(this, text);
848	}	861	}
849	-		
850	- //-----		

Figure C.96: insert caption

This significantly simplifies DateTimeParserBucket:

25	src/main/java/org/joda/time/format/DateTimeParserBucket.java	View	
129	@@ -129,21 +129,6 @@ public DateTimeParserBucket(long instantLocal, Chronology chrono,	129	iSavedFields = new SavedField[0];
130	}	130	}
131		131	
132	- static DateTimeParserBucket getDateTimeParserBucket(Chronology iChrono, int		
133	- iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZone) {		
134	- Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone,		
135	- iChrono);		
136	- return new DateTimeParserBucket(0, chrono, iLocale, iPivotYear,		
137	- iDefaultYear);		
138	- }		
139	- static DateTimeParserBucket getDateTimeParserBucket(Chronology iChrono, Locale		
140	- iLocale, Integer iPivotYear, DateTimeZone iZone, ReadWritableInstant instant) {		
141	- if (instant == null) {		
142	- throw new IllegalArgumentException("Instant must not be null");		
143	- }		
144	- Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone,		
145	- instant.getChronology());		
146	- long millis = instant.getMillis() +		
147	- instant.getChronology().getZone().getOffset(instant.getMillis());		
148	- int defaultYear =		
149	- DateTimeUtils.getChronology(instant.getChronology()).year().get(instant.getMillis());		
150	- return new DateTimeParserBucket(millis, chrono, iLocale, iPivotYear,		
151	- defaultYear);		
152	- }		
153	- }		
154	Chronology getBucketChronology(boolean iOffsetParsed) {	132	Chronology getBucketChronology(boolean iOffsetParsed) {
155	Chronology chrono = getChronology();	133	Chronology chrono = getChronology();
156	Integer offsetInteger = getOffsetInteger();	134	Integer offsetInteger = getOffsetInteger();
157			
158			
159			
160			
161			
162			
163			
164			
165			
166			
167			
168			
169			
170			
171			
172			
173			
174			
175			
176			
177			
178			
179			
180			
181			
182			
183			
184			
185			
186			
187			
188			
189			
190			
191			
192			
193			
194			
195			
196			
197			
198			
199			
200			
201			
202			
203			
204			
205			
206			
207			
208			
209			
210			
211			
212			
213			
214			
215			
216			
217			
218			
219			
220			
221			
222			
223			
224			
225			
226			
227			
228			
229			
230			
231			
232			
233			
234			
235			
236			
237			
238			
239			
240			
241			
242			
243			
244			
245			
246			
247			
248			
249			
250			
251			
252			
253			
254			
255			
256			
257			
258			
259			
260			
261			
262			
263			
264			
265			
266			
267			
268			
269			
270			
271			
272			
273			
274			
275			
276			
277			
278			
279			
280			
281			
282			
283			
284			
285			
286			
287			
288			
289			
290			
291			
292			
293			
294			
295			
296			
297			
298			
299			
300			
301			
302			
303			
304			
305			
306			
307			
308			
309			
310			
311			
312			
313			
314			
315			
316			
317			
318			
319			
320			
321			
322			
323			
324			
325			
326			
327			
328			
329			
330			
331			
332			
333			
334			
335			
336			
337			
338			
339			
340			
341			
342			
343			
344			
345			
346			
347			
348			
349			
350			
351			
352			
353			
354			
355			
356			
357			
358			
359			
360			
361			
362			
363			
364			
365			
366			
367			
368			
369			
370			
371			
372			
373			
374			
375			
376			
377			
378			
379			
380			
381			
382			
383			
384			
385			
386			
387			
388			
389			
390			
391			
392			
393			
394			
395			
396			
397			
398			
399			
400			
401			
402			
403			
404			
405			
406			
407			
408			
409			
410			
411			
412			
413			
414			
415			
416			
417			
418			
419			
420			
421			
422			
423			
424			
425			
426			
427			
428			
429			
430			
431			
432			
433			
434			
435			
436			
437			
438			
439			
440			
441			
442			
443			
444			
445			
446			
447			
448			
449			
450			
451			
452			
453			
454			
455			
456			
457			
458			
459			
460			
461			
462			
463			
464			
465			
466			
467			
468			
469			
470			
471			
472			
473			
474			
475			
476			
477			
478			
479			
480			
481			
482			
483			
484			
485			
486			
487			
488			
489			
490			
491			
492			
493			
494			
495			
496			
497			
498			
499			
500			
501			
502			
503			
504			
505			
506			
507			
508			
509			
510			
511			
512			
513			
514			
515			
516			
517			
518			
519			
520			
521			
522			
523			
524			
525			
526			
527			
528			
529			
530			
531			
532			
533			
534			
535			
536			
537			
538			
539			
540			
541			
542			
543			
544			
545			
546			
547			
548			
549			
550			
551			
552			
553			
554			
555			
556			
557			
558			
559			
560			
561			
562			
563			
564			
565			
566			
567			
568			
569			
570			
571			
572			
573			
574			
575			
576			
577			
578			
579			
580			
581			
582			
583			
584			
585			
586			
587			
588			
589			
590			
591			
592			
593			
594			
595			
596			
597			
598			
599			
600			
601			
602			
603			
604			
605			
606			
607			
608			
609			
610			
611			
612			
613			
614			
615			
616			
617			
618			
619			
620			
621			
622			
623			
624			
625			
626			
627			

SimpleParser then provides a FACADE that wraps its callback generation and generalized getResult method.

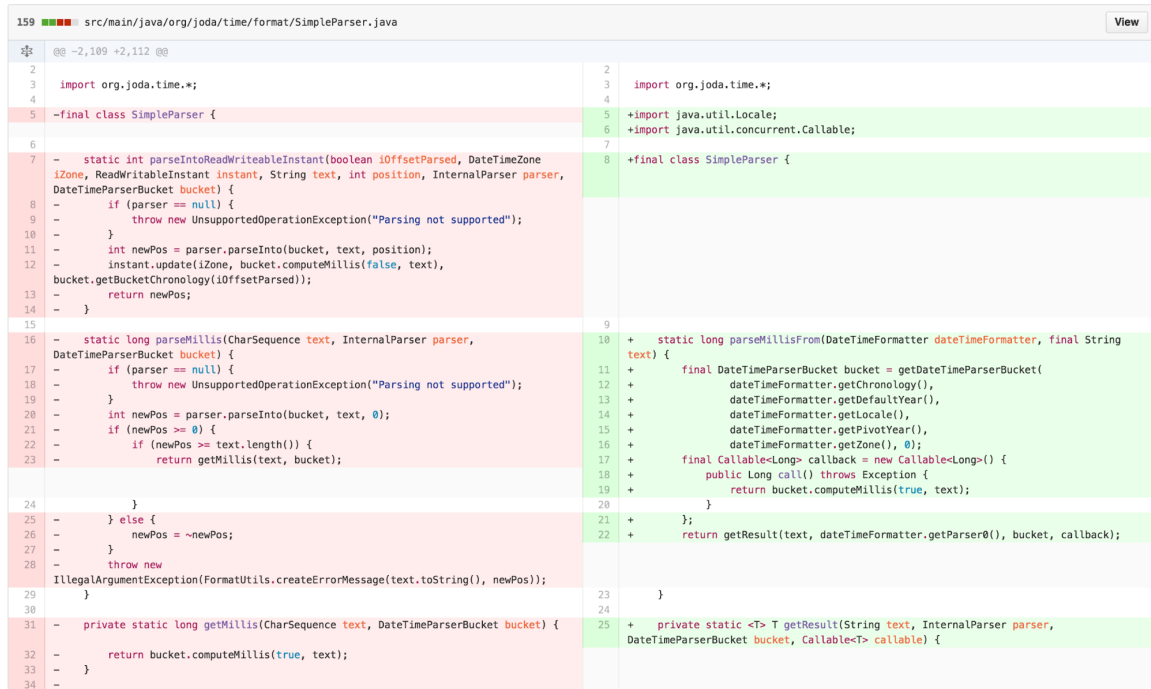


Figure C.98: insert caption

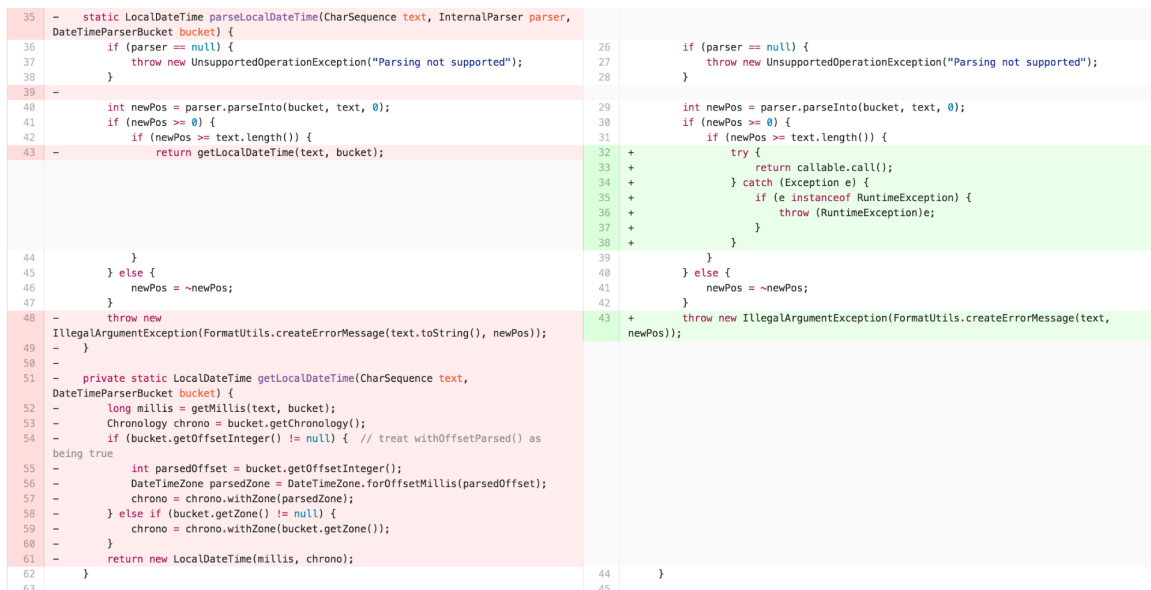


Figure C.99: insert caption

<pre> 64 - static DateTime parseDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, InternalParser parser, DateTimeParserBucket bucket) { 65 - if (parser == null) { 66 - throw new UnsupportedOperationException("Parsing not supported"); 67 - } 68 - int newPos = parser.parseInto(bucket, text, 0); 69 - if (newPos >= 0) { 70 - if (newPos >= text.length()) { 71 - return getDateTime(iOffsetParsed, iZone, text, bucket); 72 - } 73 - } else { 74 - newPos = ~newPos; 75 - } 76 - throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, 77 newPos)); 78 - } 79 - private static DateTime getDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, DateTimeParserBucket bucket) { 80 - DateTime dt = new DateTime(bucket.computeMillis(true, text), 81 bucket.getBucketChronology(iOffsetParsed)); 82 - if (iZone != null) { 83 - dt = dt.withZone(iZone); 84 - } 85 - return dt; </pre>	<pre> 46 + static LocalDateTime parseIntoLocalDateTime(DateTimeFormatter dateTimeFormatter, final String text) { 47 + final DateTimeParserBucket bucket = getDateTimeParserBucket(48 + dateTimeFormatter.getChronology(), 49 + dateTimeFormatter.getDefaultYear(), 50 + dateTimeFormatter.getLocale(), 51 + dateTimeFormatter.getPivotYear(), 52 + dateTimeFormatter.getZone(), 0); 53 + final Callable<LocalDateTime> callback = new Callable<LocalDateTime>() { 54 + public LocalDateTime call() throws Exception { 55 + long millis = bucket.computeMillis(true, text); 56 + Chronology chrono = bucket.getChronology(); 57 + if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() 58 + as being true 59 + int parsedOffset = bucket.getOffsetInteger(); 60 + DateTimeZone parsedZone = 61 + DateTimeZone.forOffsetMillis(parsedOffset); 62 + chrono = chrono.withZone(parsedZone); 63 + } else if (bucket.getZone() != null) { 64 + chrono = chrono.withZone(bucket.getZone()); 65 + } 66 + return new LocalDateTime(millis, chrono); 67 + } 68 + }; 69 + return getResult(text, dateTimeFormatter.getParser0(), bucket, callback); 70 - } 71 + static DateTime getDateTime(final DateTimeFormatter dateTimeFormatter, final String text) { 72 + final DateTimeParserBucket bucket = getDateTimeParserBucket(73 + dateTimeFormatter.getChronology(), 74 + dateTimeFormatter.getDefaultYear(), 75 + dateTimeFormatter.getLocale(), 76 + dateTimeFormatter.getPivotYear(), 77 + dateTimeFormatter.getZone(), 0); 78 + final Callable<DateTime> callback = new Callable<DateTime>() { 79 + public DateTime call() throws Exception { 80 + DateTime dt = new DateTime(bucket.computeMillis(true, text), 81 bucket.getBucketChronology(dateTimeFormatter.isOffsetParsed())); 82 + if (dateTimeFormatter.getZone() != null) { 83 + dt = dt.withZone(dateTimeFormatter.getZone()); 84 + } </pre>
--	---

Figure C.100: insert caption

<pre> 85 - } 86 - } 87 - static MutableDateTime parseMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, InternalParser parser, DateTimeParserBucket bucket) { 88 - if (parser == null) { 89 - throw new UnsupportedOperationException("Parsing not supported"); 90 - } 91 - int newPos = parser.parseInto(bucket, text, 0); 92 - if (newPos >= 0) { 93 - if (newPos >= text.length()) { 94 - return getMutableDateTime(iOffsetParsed, iZone, text, bucket); 95 - } 96 - } else { 97 - newPos = ~newPos; 98 - } 99 - throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, 100 newPos)); 101 - } 102 - } 103 - private static MutableDateTime getMutableDateTime(boolean iOffsetParsed, DateTimeZone iZone, String text, DateTimeParserBucket bucket) { 104 - MutableDateTime dt = new MutableDateTime(bucket.computeMillis(true, text), 105 bucket.getBucketChronology(iOffsetParsed)); 106 - if (iZone != null) { 107 - dt.setZone(iZone); 108 - } 109 - return dt; 110 - } </pre>	<pre> 83 + return dt; 84 + } 85 + }; 86 + return getResult(text, dateTimeFormatter.getParser0(), bucket, callback); 87 - } 88 - } 89 + static MutableDateTime getMutableDateTime(final DateTimeFormatter dateTimeFormatter, final String text) { 90 + final DateTimeParserBucket bucket = getDateTimeParserBucket(91 + dateTimeFormatter.getChronology(), 92 + dateTimeFormatter.getDefaultYear(), 93 + dateTimeFormatter.getLocale(), 94 + dateTimeFormatter.getPivotYear(), 95 + dateTimeFormatter.getZone(), 0); 96 + final Callable<MutableDateTime> callback = new Callable<MutableDateTime>() { 97 + public MutableDateTime call() throws Exception { 98 + MutableDateTime dt = new MutableDateTime(bucket.computeMillis(true, 99 text), bucket.getBucketChronology(dateTimeFormatter.isOffsetParsed())); 100 + if (dateTimeFormatter.getZone() != null) { 101 + dt.setZone(dateTimeFormatter.getZone()); 102 + } 103 + return dt; 104 + } 105 + }; 106 + return getResult(text, dateTimeFormatter.getParser0(), bucket, callback); 107 - } 108 - } 109 + private static DateTimeParserBucket getDateTimeParserBucket(Chronology iChrono, int iDefaultYear, Locale iLocale, Integer iPivotYear, DateTimeZone iZone, long millis) { 110 + Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, 111 iChrono); 112 + return new DateTimeParserBucket(millis, chrono, iLocale, iPivotYear, 113 iDefaultYear); </pre>
--	---

Figure C.101: insert caption

At this point, we are ready to apply our instance transformation to form a TEMPLATE METHOD and have individual STRATEGY implementations for each return

type.

convert methods to instance methods in preparation for STRATEGY refac...

Browse files

...tor.

Replace callback with inheritance as it is a more recognizable java programming model.

fix-bug-86-experimental-narrative

 V committed 8 days ago

1 parent e117eb0 commit 4cdb7d8bca4bf594ef2aa1e1feb91ee54bb692cb

Showing 10 changed files with 117 additions and 122 deletions.

UnifiedSplit

8 src/main/java/org/joda/time/format/ChronologyFactory.java

View

4	4	import org.joda.time.DateTimeUtils;
5	5	import org.joda.time.DateTimeZone;
6	6	
7	7	-final class ChronologyFactory {
8	8	+public final class ChronologyFactory {
9	9	private ChronologyFactory() {}
10	10	/**
11	11	* Determines the correct chronology to use.
12	12	*/
13	13	@param defaultTZ @param chrono the proposed chronology
14	14	@return the actual chronology
15	15	*/
16	16	- static Chronology selectChronology(Chronology defaultChronology, DateTimeZone defaultTZ, Chronology chrono) {
17	17	+ public static Chronology selectChronology(Chronology defaultChronology, DateTimeZone defaultTZ, Chronology chrono)
18	18	return getChronologyWithTimeZone(getChronologyWithDefaultValue(defaultChronology, chrono), defaultTZ);
19	19	}
20	20	- static Chronology getChronologyWithDefaultValue(Chronology defaultChronology, Chronology chrono) {
21	21	+ public static Chronology getChronologyWithDefaultValue(Chronology defaultChronology, Chronology chrono) {
22	22	return (defaultChronology != null) ? defaultChronology : DateTimeUtils.getChronology(chono);
23	23	}
24	24	- static Chronology getChronologyWithTimeZone(Chronology chrono, DateTimeZone defaultTimeZone) {
25	25	+ public static Chronology getChronologyWithTimeZone(Chronology chrono, DateTimeZone defaultTimeZone) {
26	26	return (defaultTimeZone != null) ? chrono.withZone(defaultTimeZone) : chrono;
27	27	}

Figure C.102: insert caption

8		src/main/java/org/joda/time/format/DateTimeFormatter.java		View	
✳		@@ -761,7 +761,7 @@ public int parseInto(ReadWritableInstant instant, String text, int position) {			
761	761	* @throws IllegalArgumentException if the text to parse is invalid			
762	762	*/			
763	763	public long parseMillis(final String text) {			
764		-	return SimpleParser.parseMillisFrom(this, text);		
	764	+	return new MillisecondParsingStrategy(this).parse(text);		
765	765	}			
766	766				
767	767	/**			
✳		@@ -815,7 +815,7 @@ public LocalTime parseLocalTime(String text) {			
815	815	* @since 2.0			
816	816	*/			
817	817	public LocalDateTime parseLocalDateTime(final String text) {			
818		-	return SimpleParser.parseIntoLocalDateTime(this, text);		
	818	+	return new LocalDateTimeParsingStrategy(this).parse(text);		
819	819	}			
820	820				
821	821	/**			
✳		@@ -836,7 +836,7 @@ public LocalDateTime parseLocalDateTime(final String text) {			
836	836	* @throws IllegalArgumentException if the text to parse is invalid			
837	837	*/			
838	838	public DateTime parseDateTime(final String text) {			
839		-	return SimpleParser.getDateTime(this, text);		
	839	+	return new DateTimeParsingStrategy(this).parse(text);		
840	840	}			
841	841				
842	842	/**			
✳		@@ -857,6 +857,6 @@ public DateTime parseDateTime(final String text) {			
857	857	* @throws IllegalArgumentException if the text to parse is invalid			
858	858	*/			
859	859	public MutableDateTime parseMutableDateTime(final String text) {			
860		-	return SimpleParser.getMutableDateTime(this, text);		
	860	+	return new MutableDateTimeParsingStrategy(this).parse(text);		
861	861	}			
862	862	}			

Figure C.103: insert caption

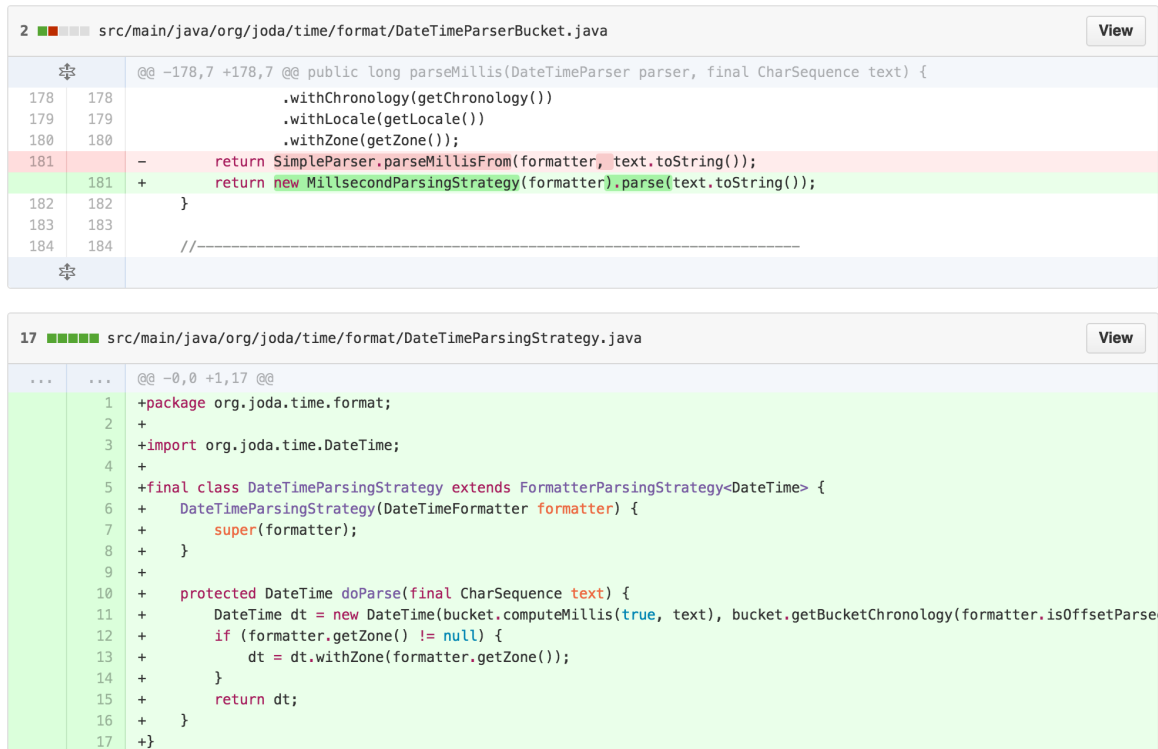


Figure C.104: insert caption

```

31 src/main/java/org/joda/time/format/FormatterParsingStrategy.java
...  ... @@ -0,0 +1,31 @@
1  +package org.joda.time.format;
2  +
3  +import org.joda.time.Chronology;
4  +
5  +abstract class FormatterParsingStrategy<T> implements ParsingStrategy<T> {
6  +    protected final DateTimeFormatter formatter;
7  +    protected final DateTimeParserBucket bucket;
8  +
9  +    FormatterParsingStrategy(final DateTimeFormatter formatter) {
10 +        this.formatter = formatter;
11 +        Chronology chrono = ChronologyFactory.selectChronology(formatter.getChronology(), formatter.getZone(), formate
12 +        this.bucket = new DateTimeParserBucket((long) 0, chrono, formatter.getLocale(), formatter.getPivotYear(), forma
13 +    }
14 +
15 +    public T parse(CharSequence text) {
16 +        if (formatter.getParser0() == null) {
17 +            throw new UnsupportedOperationException("Parsing not supported");
18 +        }
19 +        int newPos = formatter.getParser0().parseInto(bucket, text, 0);
20 +        if (newPos >= 0) {
21 +            if (newPos >= text.length()) {
22 +                return doParse(text);
23 +            }
24 +        } else {
25 +            newPos = ~newPos;
26 +        }
27 +        throw new IllegalArgumentException(FormatUtils.createErrorMessage(text.toString(), newPos));
28 +    }
29 +
30 +    protected abstract T doParse(CharSequence text);
31 +}

```

Figure C.105: insert caption

```

25 src/main/java/org/joda/time/format/LocalDateTimeParsingStrategy.java
...  ... @@ -0,0 +1,25 @@
1  +package org.joda.time.format;
2  +
3  +import org.joda.time.Chronology;
4  +import org.joda.time.DateTimeZone;
5  +import org.joda.time.LocalDateTime;
6  +
7  +final class LocalDateTimeParsingStrategy extends FormatterParsingStrategy<LocalDateTime> {
8  +
9  +    LocalDateTimeParsingStrategy(DateTimeFormatter formatter) {
10 +        super(formatter);
11 +    }
12 +
13 +    protected LocalDateTime doParse(final CharSequence text) {
14 +        long millis = bucket.computeMillis(true, text);
15 +        Chronology chrono = bucket.getChronology();
16 +        if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() as being true
17 +            int parsedOffset = bucket.getOffsetInteger();
18 +            DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
19 +            chrono = chrono.withZone(parsedZone);
20 +        } else if (bucket.getZone() != null) {
21 +            chrono = chrono.withZone(bucket.getZone());
22 +        }
23 +        return new LocalDateTime(millis, chrono);
24 +    }
25 +}

```

Figure C.106: insert caption



Figure C.107: insert caption

5 src/main/java/org/joda/time/format/ParsingStrategy.javaView

...
1 +package org.joda.time.format;
2 +
3 +public interface ParsingStrategy<T> {
4 + T parse(CharSequence text);
5 +}

113 src/main/java/org/joda/time/format/SimpleParser.javaView

...
1 -package org.joda.time.format;
2 -
3 -import org.joda.time.*;
4 -
5 -import java.util.Locale;
6 -import java.util.concurrent.Callable;
7 -
8 -final class SimpleParser {
9 -
10 - static long parseMillisFrom(DateTimeFormatter dateTimeFormatter, final String text) {
11 - final DateTimeParserBucket bucket = getDateTimeParserBucket(
12 - dateTimeFormatter.getChronology(),
13 - dateTimeFormatter.getDefaultYear(),
14 - dateTimeFormatter.getLocale(),
15 - dateTimeFormatter.getPivotYear(),
16 - dateTimeFormatter.getZone(), 0);
17 - final Callable<Long> callback = new Callable<Long>() {
18 - public Long call() throws Exception {
19 - return bucket.computeMillis(true, text);
20 - }
21 - };
22 - return getResult(text, dateTimeFormatter.getParser0(), bucket, callback);
23 - }
24 -
25 - private static <T> T getResult(String text, InternalParser parser, DateTimeParserBucket bucket, Callable<T> callabl
26 - if (parser == null) {
27 - throw new UnsupportedOperationException("Parsing not supported");
28 - }
29 - int newPos = parser.parseInto(bucket, text, 0);
30 - if (newPos >= 0) {
31 - if (newPos >= text.length()) {
32 - try {

Figure C.108: insert caption

```

33 -         return callable.call();
34 -     } catch (Exception e) {
35 -         if (e instanceof RuntimeException) {
36 -             throw (RuntimeException)e;
37 -         }
38 -     }
39 - }
40 - } else {
41 -     newPos = ~newPos;
42 - }
43 - throw new IllegalArgumentException(FormatUtils.createErrorMessage(text, newPos));
44 - }
45 -
46 - static LocalDateTime parseIntoLocalDateTime(DateTimeFormatter dateTimeFormatter, final String text) {
47 -     final DateTimeParserBucket bucket = getDateTimeParserBucket(
48 -         dateTimeFormatter.getChronology(),
49 -         dateTimeFormatter.getDefaultYear(),
50 -         dateTimeFormatter.getLocale(),
51 -         dateTimeFormatter.getPivotYear(),
52 -         dateTimeFormatter.getZone(), 0);
53 -     final Callable<LocalDateTime> callback = new Callable<LocalDateTime>() {
54 -         public LocalDateTime call() throws Exception {
55 -             long millis = bucket.computeMillis(true, text);
56 -             Chronology chrono = bucket.getChronology();
57 -             if (bucket.getOffsetInteger() != null) { // treat withOffsetParsed() as being true
58 -                 int parsedOffset = bucket.getOffsetInteger();
59 -                 DateTimeZone parsedZone = DateTimeZone.forOffsetMillis(parsedOffset);
60 -                 chrono = chrono.withZone(parsedZone);
61 -             } else if (bucket.getZone() != null) {
62 -                 chrono = chrono.withZone(bucket.getZone());
63 -             }
64 -             return new LocalDateTime(millis, chrono);
65 -         }
66 -     };
67 -     return getResult(text, dateTimeFormatter.getParser0(), bucket, callback);
68 - }
69 -
70 - static DateTime getDateTime(final DateTimeFormatter dateTimeFormatter, final String text) {
71 -     final DateTimeParserBucket bucket = getDateTimeParserBucket(
72 -         dateTimeFormatter.getChronology(),
73 -         dateTimeFormatter.getDefaultYear(),
74 -         dateTimeFormatter.getLocale(),
75 -         dateTimeFormatter.getPivotYear(),
76 -         dateTimeFormatter.getZone(), 0);
77 -     final Callable<DateTime> callback = new Callable<DateTime>() {
78 -         public DateTime call() throws Exception {

```

Figure C.109: insert caption

```

79 -         DateTime dt = new DateTime(bucket.computeMillis(true, text), bucket.getBucketChronology(dateTimeFormatt
80 -         if (dateTimeFormatter.getZone() != null) {
81 -             dt = dt.withZone(dateTimeFormatter.getZone());
82 -         }
83 -         return dt;
84 -     }
85 - };
86 - return getResult(text, dateTimeFormatter.getParser0(), bucket, callback);
87 - }
88 -
89 - static MutableDateTime getMutableDateTime(final DateTimeFormatter dateTimeFormatter, final String text) {
90 -     final DateTimeParserBucket bucket = getDateTimeParserBucket(
91 -         dateTimeFormatter.getChronology(),
92 -         dateTimeFormatter.getDefaultYear(),
93 -         dateTimeFormatter.getLocale(),
94 -         dateTimeFormatter.getPivotYear(),
95 -         dateTimeFormatter.getZone(), 0);
96 -
97 -     final Callable<MutableDateTime> callback = new Callable<MutableDateTime>() {
98 -         public MutableDateTime call() throws Exception {
99 -             MutableDateTime dt = new MutableDateTime(bucket.computeMillis(true, text), bucket.getBucketChronology(d
100 -             if (dateTimeFormatter.getZone() != null) {
101 -                 dt.setZone(dateTimeFormatter.getZone());
102 -             }
103 -             return dt;
104 -         }
105 -     };
106 -     return getResult(text, dateTimeFormatter.getParser0(), bucket, callback);
107 - }
108 -
109 - private static DateTimeParserBucket getDateTimeParserBucket(Chronology iChrono, int iDefaultYear, Locale iLocale, I
110 -     Chronology chrono = ChronologyFactory.selectChronology(iChrono, iZone, iChrono);
111 -     return new DateTimeParserBucket(millis, chrono, iLocale, iPivotYear, iDefaultYear);
112 - }
113 -}

```

Figure C.110: insert caption

At this point, we've replaced the `SimpleParser` with a `ParsingStrategy` and a `FormatterParsingStrategy` that holds the guts of the `getResult` method, with subclass implementations for each return type.

I now also had a straightforward point to simplify common logic. I also had an interface that I could use to unify the differences between the API design of `parseInto` which mutates the input parameter and returns the resulting position, and the other `parse` methods. This reduces the architectural complexity of `DateTimeFormatter` by making all of the parsing follow a common pattern.

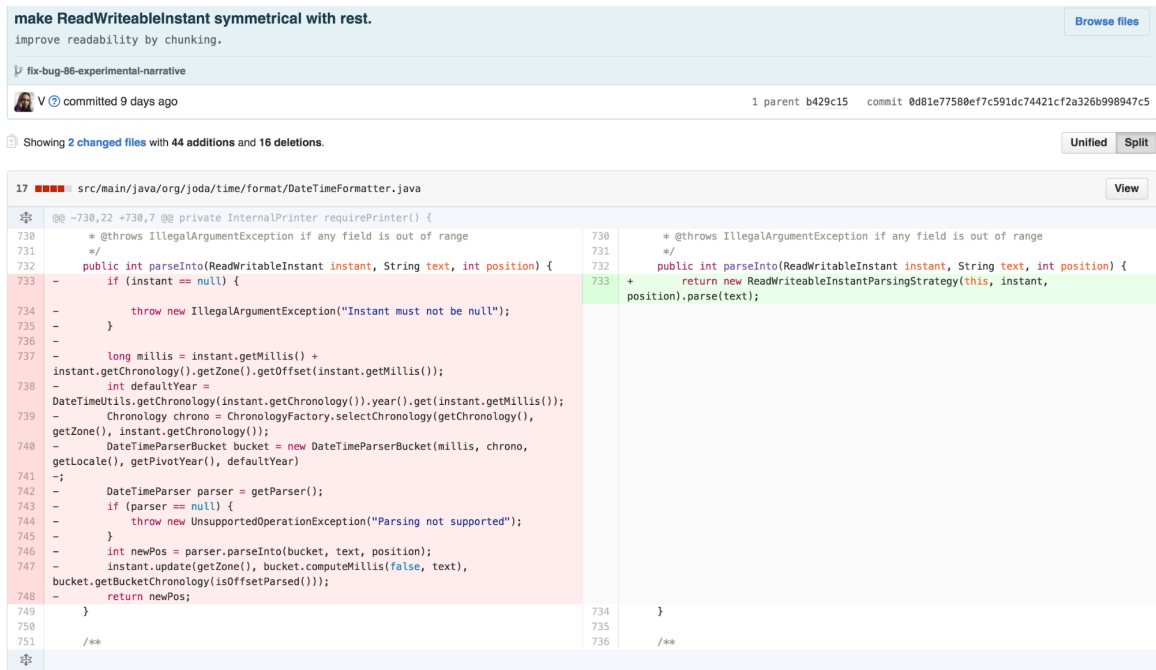


Figure C.111: insert caption

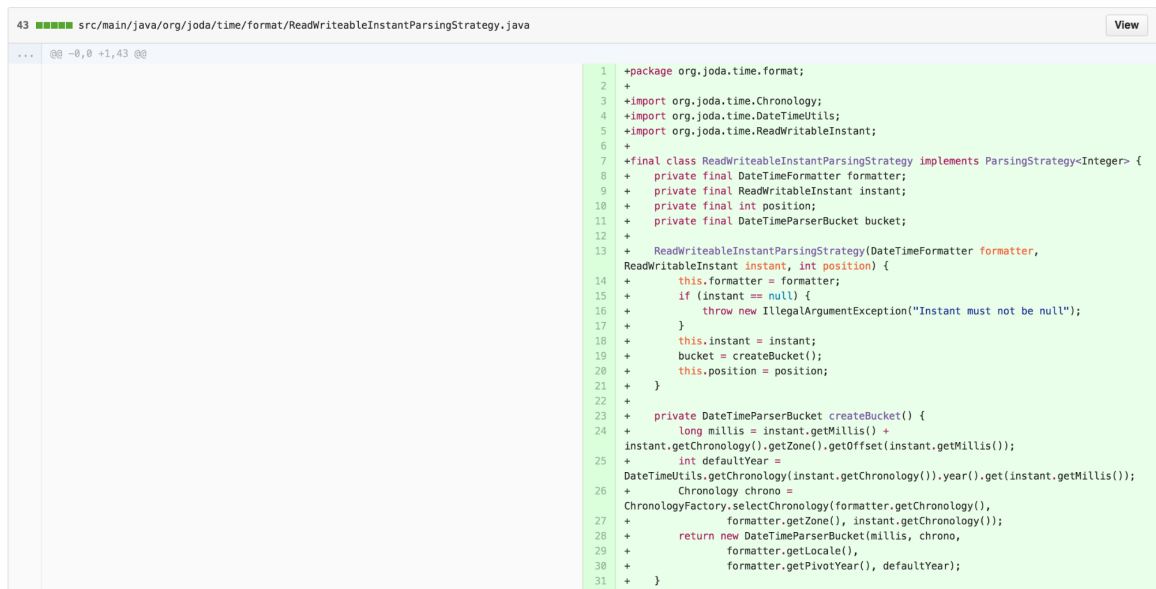


Figure C.112: insert caption


```

32 +
33 + public Integer parse(CharSequence text) {
34 +     DateTimeParser parser = formatter.getParser();
35 +     if (parser == null) {
36 +         throw new UnsupportedOperationException("Parsing not supported");
37 +     }
38 +     int newPos = parser.parseInto(bucket, text.toString(), position);
39 +     instant.update(formatter.getZone(), bucket.computeMillis(false, text),
40 +         bucket.getBucketChronology(formatter.isOffsetParsed()));
41 +     return newPos;
42 + }
43 +}

```

Figure C.113: insert caption

When I first tried to make the `ReadWritableInstantParsingStrategy`, I tried directly extending `FormatterParsingStrategy` like the other implementations. When I ran the tests, they failed. Something was different about the way it worked. Luckily, having a separate interface for `ParsingStrategy` from the abstract class implementation of `FormatterParsingStrategy` allowed me to move forward to advance architectural similarity without being blocked. I was able to create the implementation with only slight duplication of work done in the `FormatterParsingStrategy`. When I think about it further, it seems like the primary differences between `ReadWritableInstantParsingStrategy` and the other implementations is that the starting number of milliseconds for the `DateTimeParserBucket` is not 0, and neither is the position for the `DateTimeParser`. I could've spent some time refactoring `FormatterParsingStrategy` to parametrize it such that `ReadWritableInstantParsingStrategy` could extend it. I decided not to because this code path is unlikely to get explored during the study and the goal of this work was to simplify `DateTimeFormatter`.

C.6.3 Signaling architecture by organization – using packages as chunks

I now had a simpler `DateTimeFormatter` whose parsing methods delegated to implementations of parsing STRATEGY. However, the package `org.joda.time.format` had grown more imposing by adding all of these new classes. We can use packages to organize classes, again applying the concepts of sequencing and chunking to manage how much context a programmer needs to keep in their head at one time. Hence, we add a new package `org.joda.time.format.parsing` and move the new classes over.

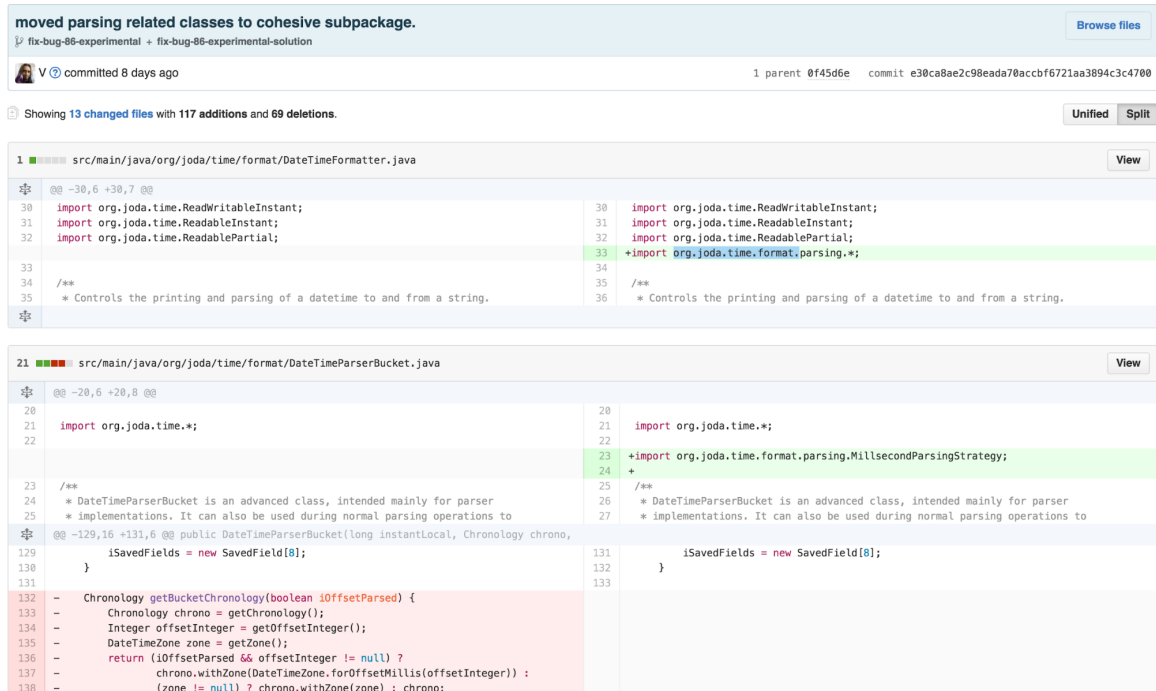


Figure C.114: insert caption

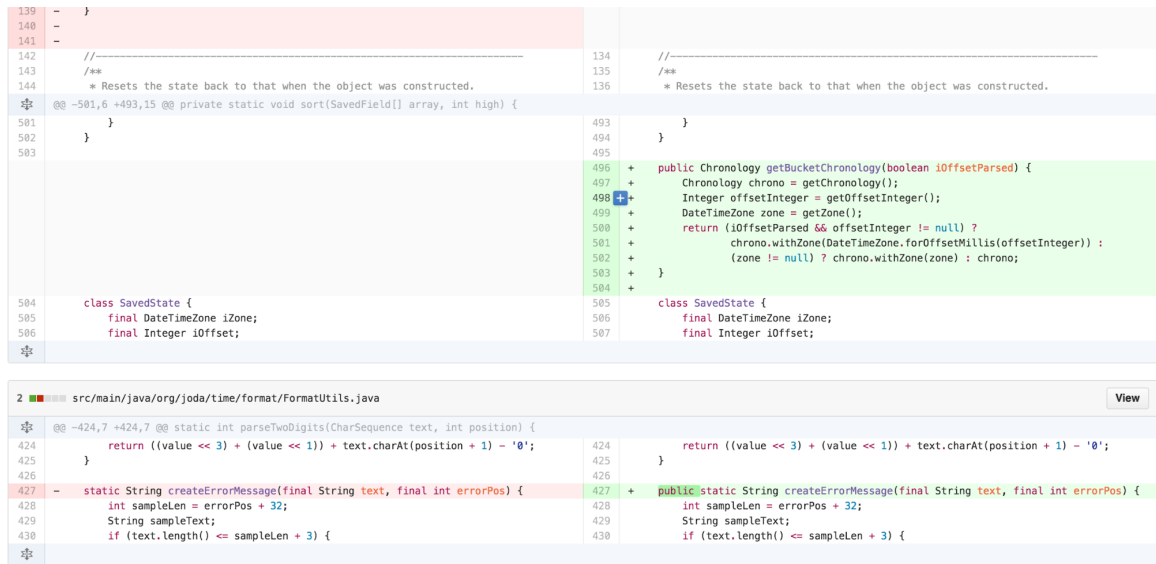


Figure C.115: insert caption

```

32 ■■■■ src/main/java/org/joda/time/format/InternalParser.java
View
@@ -23,34 +23,8 @@
23 *
24 * @author Stephen Colebourne
25 * @since 2.4
26 */
27 -interface InternalParser {
28 -    /**
29 -     * Returns the expected maximum number of characters consumed.
30 -     * The actual amount should rarely exceed this estimate.
31 -     *
32 -     * @return the estimated length
33 -     */
34 -    int estimateParsedLength();
35 -
36 -    /**
37 -     * Parse an element from the given text, saving any fields into the given
38 -     * DateTimeParserBucket. If the parse succeeds, the return value is the new
39 -     * text position. Note that the parse may succeed without fully reading the
40 -     * text.
41 -     *
42 -     * @param bucket field are saved into this, not null
43 -     * @param text the text to parse, not null
44 -     * @param position position to start parsing from
45 -     * @return new position, negative value means parse failed -
46 -     * apply complement operator (~) to get position of failure
47 -     * @throws IllegalArgumentException if any field is out of range
48 -     */
49 -    int parseInto(DateTimeParserBucket bucket, CharSequence text, int position);
50 -}
51
23 *
24 * @author Stephen Colebourne
25 * @since 2.4
26 + @deprecated Use org.joda.time.format.parsing.InternalParser instead.
27 + */
28 +@Deprecated
29 +interface InternalParser extends org.joda.time.format.parsing.InternalParser {
30 }

```

Figure C.116: insert caption

```

12 ■■■■ src/main/java/org/joda/time/format/MillisecondParsingStrategy.java
View
... @@ -1,12 +0,0 @@
1 -package org.joda.time.format;
2 -
3 -final class MillisecondParsingStrategy extends FormatterParsingStrategy<Long> {
4 -    MillisecondParsingStrategy(DateTimeFormatter formatter) {
5 -        super(formatter);
6 -    }
7 -
8 -    protected Long doParse(final CharSequence text) {
9 -        return bucket.computeMillis(true, text);
10 -    }
11 -}
12 -}

7 ■■■■ .../time/format/DateTimeParsingStrategy.java - ...mat/parsing/DateTimeParsingStrategy.java
View
... @@ -1,9 +1,10 @@
1 -package org.joda.time.format;
2 -
3 -import org.joda.time.DateTime;
4 -
5 -final class DateTimeParsingStrategy extends FormatterParsingStrategy<DateTime> {
6 -    DateTimeParsingStrategy(DateTimeFormatter formatter) {
7 -        super(formatter);
8 -    }
9 -}

1 +package org.joda.time.format.parsing;
2 +import org.joda.time.DateTime;
3 +import org.joda.time.format.DateTimeFormatter;
4 +
5 +public final class DateTimeParsingStrategy extends FormatterParsingStrategy<DateTime> {
6 +    public DateTimeParsingStrategy(final DateTimeFormatter formatter) {
7 +        super(formatter);
8 +    }
9 +}

8 ■■■■ ...time/format/FormatterParsingStrategy.java - ...mat/parsing/FormatterParsingStrategy.java
View
... @@ -1,6 +1,7 @@
1 -package org.joda.time.format;
2 -
3 -import org.joda.time.Chronology;
4 -
5 -abstract class FormatterParsingStrategy<T> implements ParsingStrategy<T> {
6 -    protected final DateTimeFormatter formatter;
7 -}

1 +package org.joda.time.format.parsing;
2 +import org.joda.time.Chronology;
3 +import org.joda.time.format.*;
4 +
5 +abstract class FormatterParsingStrategy<T> implements ParsingStrategy<T> {
6 +    protected final DateTimeFormatter formatter;
7 +}

```

Figure C.117: insert caption

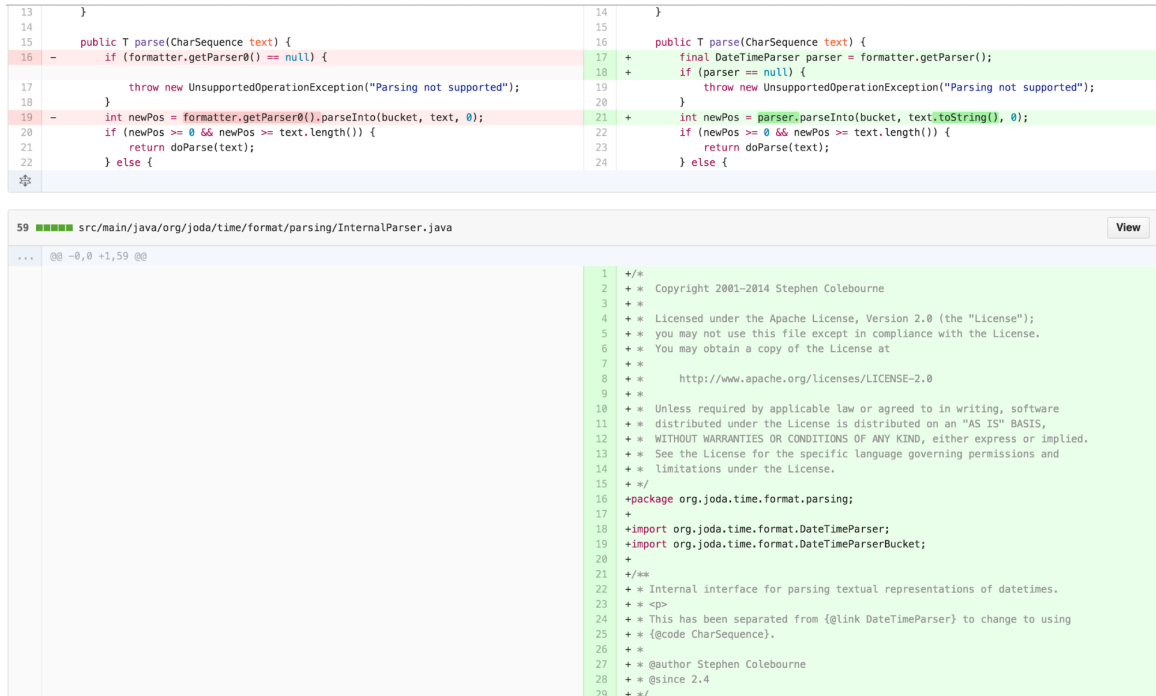


Figure C.118: insert caption

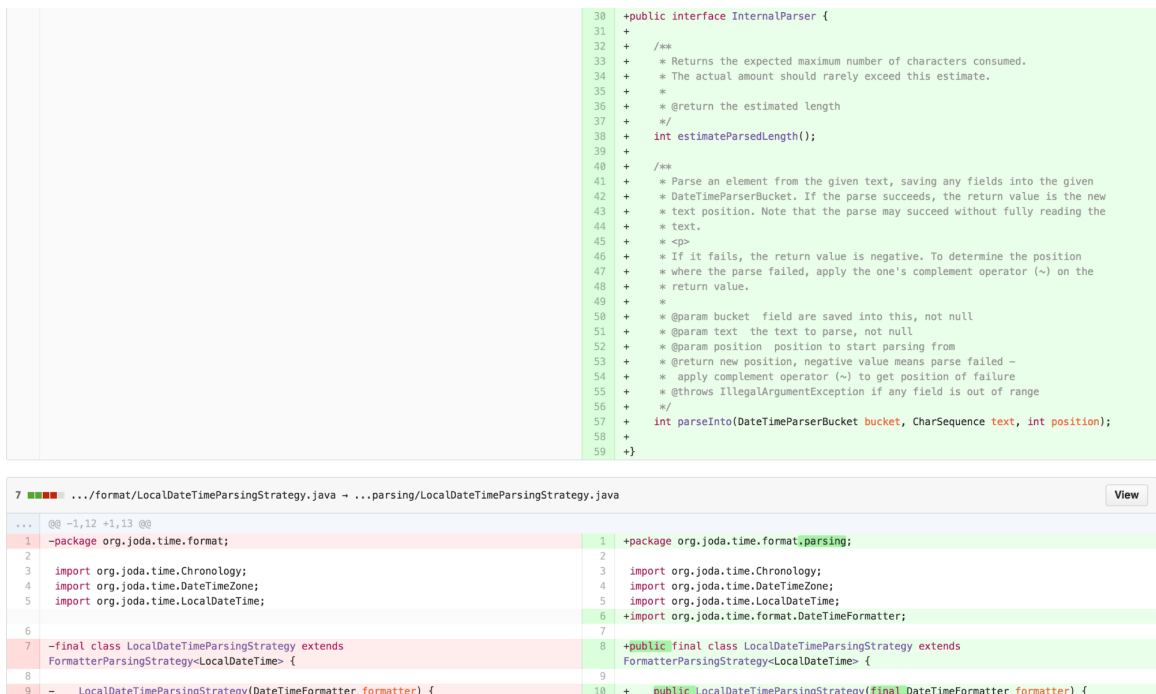


Figure C.119: insert caption

10 11 12	<code>super(formatter);</code> <code>}</code>	11 12 13	<code>super(formatter);</code> <code>}</code>
14 <code>src/main/java/org/joda/time/format/parsing/MillisecondParsingStrategy.java</code> View			
... @@ -8,8 +1,14 @@		1 2 3 4 5 6 7 8 9 10 11 12 13 14	<code>+package org.joda.time.format.parsing;</code> <code>+</code> <code>+import org.joda.time.format.DateTimeFormatter;</code> <code>+</code> <code>+public final class MillisecondParsingStrategy extends FormatterParsingStrategy<Long> {</code> <code>+</code> <code>public MillisecondParsingStrategy(final DateTimeFormatter formatter) {</code> <code>super(formatter);</code> <code>+</code> <code>+</code> <code>protected Long doParse(final CharSequence text) {</code> <code>return bucket.computeMillis(true, text);</code> <code>+</code> <code>+</code>
7 <code>format/MutableDateTimeParsingStrategy.java -> ...rsing/MutableDateTimeParsingStrategy.java</code> View			
... @@ -1,10 +1,11 @@		1 2 3 4 5 6 7 8 9 10 11	<code>-package org.joda.time.format;</code> <code>import org.joda.time.MutableDateTime;</code> <code>-final class MutableDateTimeParsingStrategy extends</code> <code>FormatterParsingStrategy<MutableDateTime> {</code> <code>- MutableDateTimeParsingStrategy(DateTimeFormatter formatter) {</code> <code>super(formatter);</code> <code>}</code>
		1 2 3 4 5 6 7 8 9 10 11	<code>+package org.joda.time.format.parsing;</code> <code>import org.joda.time.MutableDateTime;</code> <code>+import org.joda.time.format.DateTimeFormatter;</code> <code>+public final class MutableDateTimeParsingStrategy extends</code> <code>FormatterParsingStrategy<MutableDateTime> {</code> <code>+ public MutableDateTimeParsingStrategy(final DateTimeFormatter formatter) {</code> <code>super(formatter);</code> <code>}</code>

Figure C.120: insert caption

2 <code>...org/joda/time/format/ParsingStrategy.java -> .../time/format/parsing/ParsingStrategy.java</code> View	
... @@ -1,4 +1,4 @@	
1 2 3 4	<code>-package org.joda.time.format;</code> <code>public interface ParsingStrategy<T> {</code> <code>T parse(CharSequence text);</code>
	<code>+package org.joda.time.format.parsing;</code> <code>public interface ParsingStrategy<T> {</code> <code>T parse(CharSequence text);</code>
14 <code>.../ReadWriteableInstantParsingStrategy.java -> .../ReadWriteableInstantParsingStrategy.java</code> View	
... @@ -1,16 +1,22 @@	
1 2 3 4 5	<code>-package org.joda.time.format;</code> <code>import org.joda.time.Chronology;</code> <code>import org.joda.time.DateTimeUtils;</code> <code>import org.joda.time.ReadWritableInstant;</code>
6 7 8 9 10 11 12 13 14 15 16	<code>-final class ReadWritableInstantParsingStrategy implements ParsingStrategy<Integer> {</code> <code>- private final DateTimeFormatter formatter;</code> <code>private final ReadWritableInstant instant;</code> <code>private final int position;</code> <code>private final DateTimeParserBucket bucket;</code> <code>- ReadWritableInstantParsingStrategy(DateTimeFormatter formatter,</code> <code>ReadWritableInstant instant, int position) {</code> <code>this.formatter = formatter;</code> <code>if (instant == null) {</code> <code>throw new IllegalArgumentException("Instant must not be null");</code>
	<code>+package org.joda.time.format.parsing;</code> <code>import org.joda.time.Chronology;</code> <code>+import org.joda.time.DateTimeUtils;</code> <code>import org.joda.time.ReadWritableInstant;</code> <code>+import org.joda.time.format.ChronologyFactory;</code> <code>+import org.joda.time.format.DateTimeFormatter;</code> <code>+import org.joda.time.format.DateTimeParser;</code> <code>+import org.joda.time.format.DateTimeParserBucket;</code> <code>+public final class ReadWritableInstantParsingStrategy implements</code> <code>ParsingStrategy<Integer> {</code> <code>private final ReadWritableInstant instant;</code> <code>private final int position;</code> <code>+ private final DateTimeFormatter formatter;</code> <code>private final DateTimeParserBucket bucket;</code> <code>+ public ReadWritableInstantParsingStrategy(final DateTimeFormatter formatter,</code> <code>final ReadWritableInstant instant,</code> <code>final int position) {</code> <code>this.formatter = formatter;</code> <code>if (instant == null) {</code> <code>throw new IllegalArgumentException("Instant must not be null");</code>

Figure C.121: insert caption

Another heuristic to reduce the number of classes is to find LAZY CLASS and investigate whether we can MOVE METHOD. As `ChronologyFactory` is merely exposing a method that could be a STATIC FACTORY METHOD on `Chronology`, we move the functionality over.

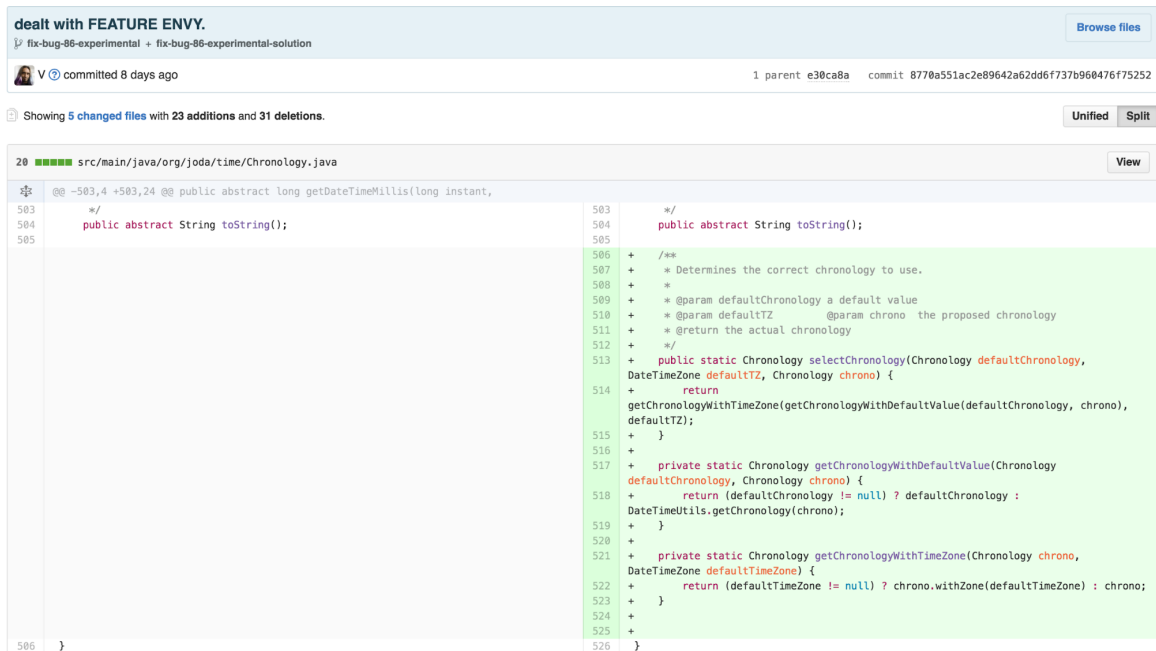


Figure C.122: insert caption

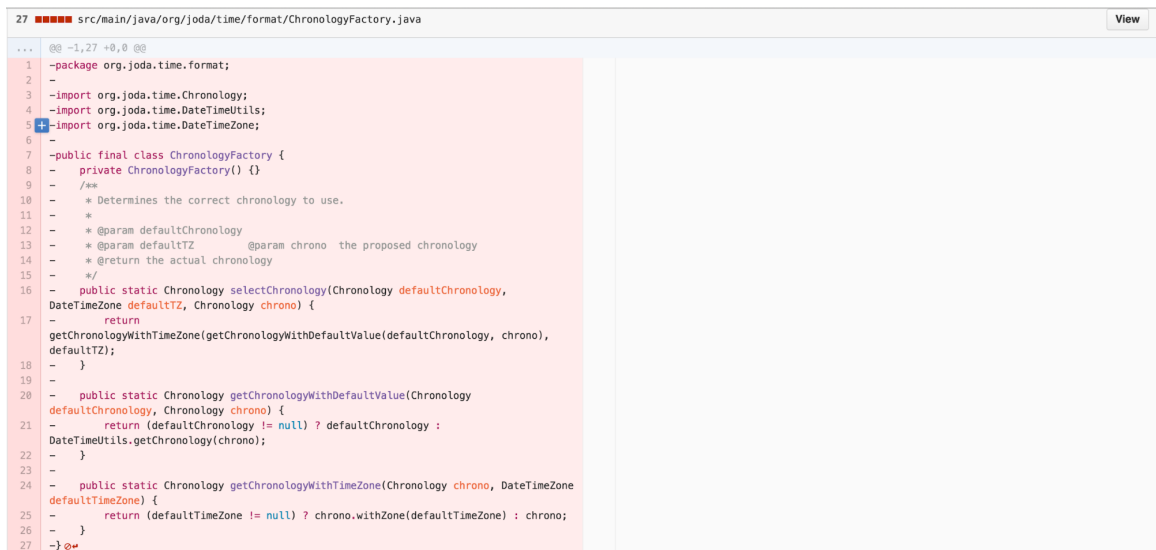


Figure C.123: insert caption

2 src/main/java/org/joda/time/format/DateTimeFormatter.java		View
<pre> 668 @@ -668,7 +668,7 @@ public String print(ReadablePartial partial) { 669 private void printTo(Appendable appendable, long instant, Chronology chrono) 670 throws IOException { 671 InternalPrinter printer = requirePrinter(); 672 chrono = ChronologyFactory.selectChronology(iChrono, iZone, chrono); 673 // Shift instant into local time (UTC) to avoid excessive offset 674 // calculations when printing multiple fields in a composite printer. 675 DateTimeZone zone = chrono.getZone(); </pre>		
<pre> 668 private void printTo(Appendable appendable, long instant, Chronology chrono) 669 throws IOException { 670 InternalPrinter printer = requirePrinter(); 671 chrono = Chronology.selectChronology(iChrono, iZone, chrono); 672 // Shift instant into local time (UTC) to avoid excessive offset 673 // calculations when printing multiple fields in a composite printer. 674 DateTimeZone zone = chrono.getZone(); </pre>		
2 src/main/java/org/joda/time/format/parsing/FormatterParsingStrategy.java		View
<pre> 9 @@ -9,7 +9,7 @@ 10 FormatterParsingStrategy(final DateTimeFormatter formatter) { 11 this.formatter = formatter; 12 Chronology chrono = ChronologyFactory.selectChronology(formatter.getChronology(), formatter.getZone(), 13 ChronologyFactory.selectChronology(formatter.getChronology(), formatter.getZone(), 14 ChronologyFactory.selectChronology(formatter.getChronology(), formatter.getZone(), 15 ChronologyFactory.selectChronology(formatter.getChronology(), formatter.getZone(), </pre>		
<pre> 9 FormatterParsingStrategy(final DateTimeFormatter formatter) { 10 this.formatter = formatter; 11 Chronology chrono = Chronology.selectChronology(formatter.getChronology(), 12 Chronology.selectChronology(formatter.getChronology(), 13 Chronology.selectChronology(formatter.getChronology(), 14 Chronology.selectChronology(formatter.getChronology(), </pre>		
3 src/main/java/org/joda/time/format/parsing/ReadWritableInstantParsingStrategy.java		View
<pre> 3 @@ -3,7 +3,6 @@ 4 import org.joda.time.Chronology; 5 import org.joda.time.DateTimeUtils; 6 import org.joda.time.ReadWritableInstant; 7 import org.joda.time.format.ChronologyFactory; 8 import org.joda.time.format.DateTimeFormatter; 9 import org.joda.time.format.DateTimeParser; 10 import org.joda.time.format.DateTimeParserBucket; </pre>		
<pre> 29 @@ -29,7 +28,7 @@ public ReadWritableInstantParsingStrategy(final DateTimeFormatter formatter, 30 private DateTimeParserBucket createBucket() { 31 long millis = instant.getMillis() + 32 instant.getChronology().getZone().getOffset(instant.getMillis()); </pre>		
<pre> 3 import org.joda.time.Chronology; 4 import org.joda.time.DateTimeUtils; 5 import org.joda.time.ReadWritableInstant; 6 import org.joda.time.format.DateTimeFormatter; 7 import org.joda.time.format.DateTimeParser; 8 import org.joda.time.format.DateTimeParserBucket; </pre>		
<pre> 28 private DateTimeParserBucket createBucket() { 29 long millis = instant.getMillis() + 30 instant.getChronology().getZone().getOffset(instant.getMillis()); </pre>		

Figure C.124: insert caption

C.6.4 Making Derived Components Expressive, the value of names

Are we done? The resultant code is structurally simpler than before. The outstanding question is, does it communicate the design intent of the original algorithm in a concise and clear way? Without access to the direct authors of the code and given the complexity of the original, it's difficult to say. Debugging through the code flow into the area of the bug, we see much of the code is now delegation: `DateTimeFormatter` -> `DateTimeParsingStrategy` -> `FormatterParsingStrategy` -> `NumberFormatter` -> `OffsetCalculator`. Most of the work involved in this bug is in the `OffsetCalculator`...could the code be clearer?

Consultation with other engineers suggested that the `OffsetCalculator` has variable names that were mostly consequential to its original sources. Revisiting those names may enhance clarity. Additionally, much of the work of `calculateLength` is done once, ultimately to be used by `calculateValue`. This work can be moved to the constructor, with expressive variables assigned. I can clean this up further.

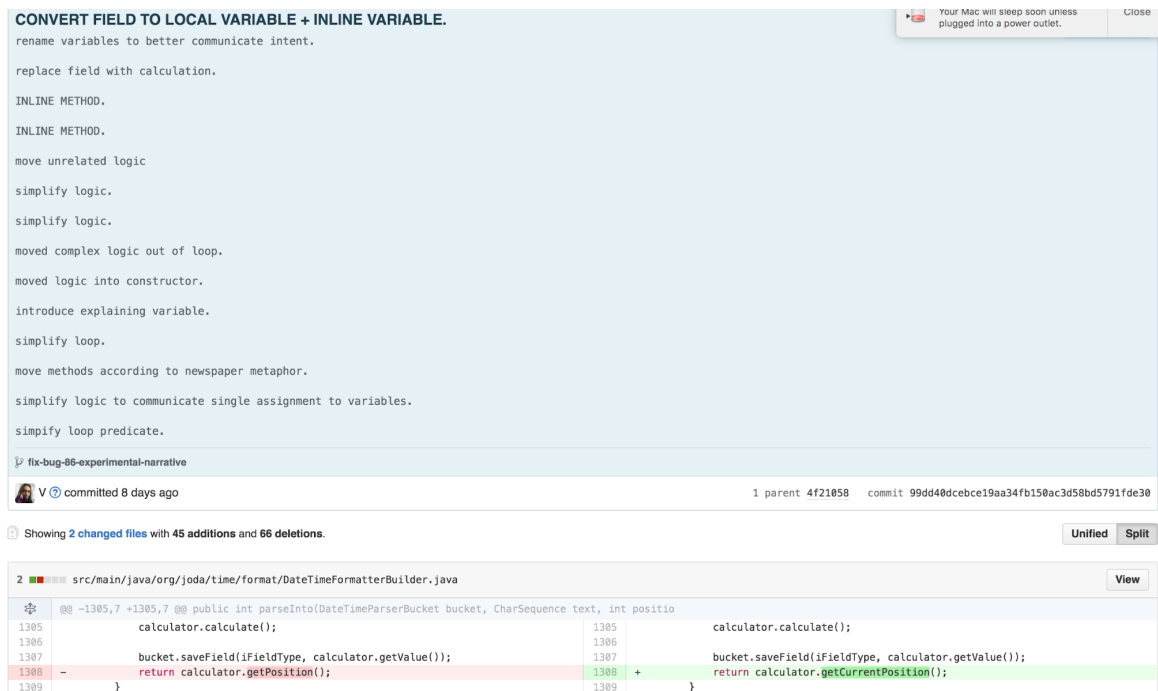


Figure C.125: insert caption

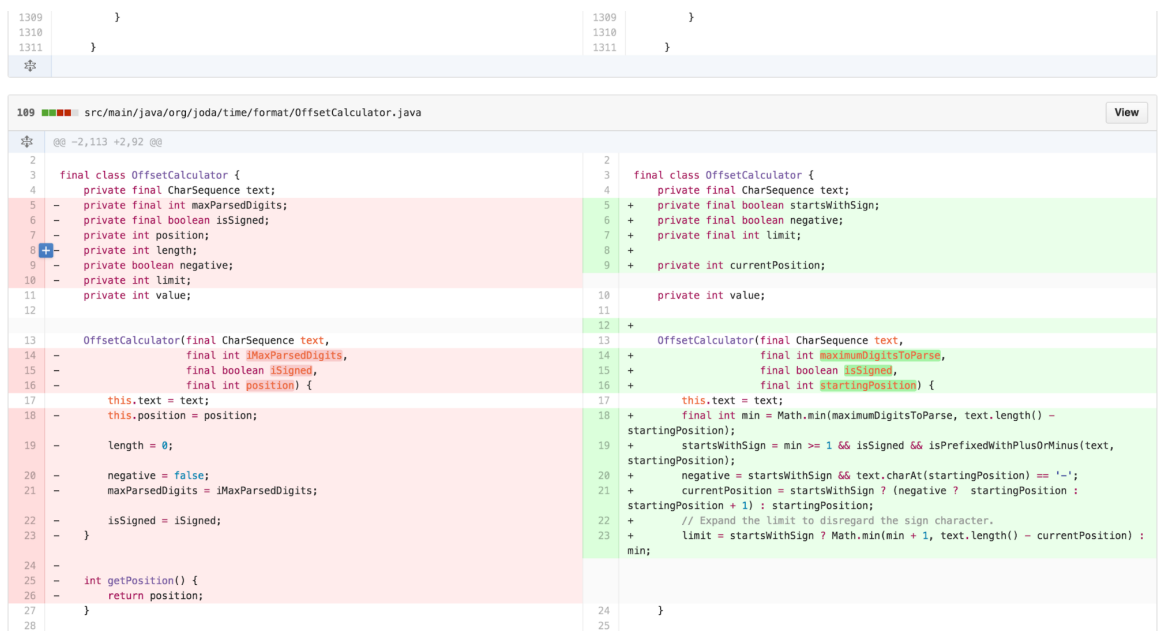


Figure C.126: insert caption

29	- int getValue() {	26	+ private static boolean isPrefixedWithPlusOrMinus(final CharSequence text, final
30	- return value;	27	+ int startingPosition) {
		28	+ final boolean isFirstCharacterOperator =
		29	+ isCharacterOperator(text.charAt(startingPosition));
		30	+ final boolean hasNextDigitCharacter = startingPosition < text.length() - 1 &&
		31	+ Character.isDigit(text.charAt(startingPosition + 1));
		32	+ return isFirstCharacterOperator && hasNextDigitCharacter;
31	}	33	}
32			
33	- void calculate() {	34	+ private static boolean isCharacterOperator(final char currentCharacter) {
34	- calculateLength();	35	+ return currentCharacter == '-' currentCharacter == '+';
35	- updatePositionAndValue();		
36	- }		
37	- }		
38	- private void calculateLength() {		
39	- limit = Math.min(maxParsedDigits, text.length() - position);		
40	- while (length < limit && shouldContinue()) {		
41	- updateBasedOnSign();		
42	- length = length + 1;		
43	- }		
44	- }		
45	- }		
46	- private boolean shouldContinue() {		
47	- final boolean hasSign = isPrefixedWithPlusOrMinus() && isSigned;		
48	- return Character.isDigit(text.charAt(position + length)) hasSign;		
49	- }		
50	- }		
51	- private boolean isPrefixedWithPlusOrMinus() {	34	}
52	- final int index = position + length;	35	
53	- final char currentCharacter = text.charAt(index);	36	+ int getCurrentPosition() {
54	- final boolean isFirstCharacterOperator = length == 0 &&	37	+ return currentPosition;
55	- isCharacterOperator(currentCharacter);		
56	- final boolean hasNextDigitCharacter = index < text.length() - 1 &&		
57	- Character.isDigit(text.charAt(index + 1));		
58	- return isFirstCharacterOperator && isBeforeBoundary() &&		
59	- hasNextDigitCharacter;		
60	- }		
61	- }		
62	- }		
63	- private boolean isBeforeBoundary() {		
64	- return length + 1 <= limit;		
65	- }		
66	- }		

Figure C.127: insert caption

67	- private void updateBasedOnSign() {	48	+ private int calculateLength() {
68	- if (isPrefixedWithPlusOrMinus()) {	49	+ int length = startsWithSign ? 1 : 0;
69	- negative = text.charAt(position + length) == '-';	50	+ while (length + 1 <= limit && Character.isDigit(text.charAt(currentPosition +
70	- length = negative ? length + 1 : length;	51	+ length))) {
71	- position = negative ? position : position + 1;		+ length = length + 1;
72	- // Expand the limit to disregard the sign character.		
73	- limit = Math.min(limit + 1, text.length() - position);		
74	- }		
75	- }		
76	- }		
77	- private void updatePositionAndValue() {	52	}
78	- if (length == 0) {	53	+ return length;
79	- position = ~position;	54	}
80	- } else if (length >= 9) {	55	
81	- useDefaultParser();	56	+ private void updatePositionAndValue(int length) {
82	- } else {	57	- if (length == 0) {
83	- useFastParser();	58	- currentPosition = ~currentPosition;
84	- }	59	- } else if (length >= 9) {
85	- }	60	- useDefaultParser(length);
86	- }	61	- } else {
87	- private void useDefaultParser() {	62	- useFastParser(length);
88	- // Since value may exceed integer limits, use stock parser	63	- }
89	- // which checks for this.	64	- }
90	- final String toParse = text.subSequence(position, position +	65	
91	- length).toString();	66	+ private void useDefaultParser(int length) {
92	- value = Integer.parseInt(toParse);	67	- // Since value may exceed integer limits, use stock parser
93	- position += length;	68	- // which checks for this.
94	- }	69	+ final String toParse = text.subSequence(currentPosition, currentPosition +
95	- private void useFastParser() {	70	+ length).toString();
96	- int i = negative ? position + 1 : position;	71	- value = Integer.parseInt(toParse);
97	-	72	- currentPosition += length;
98	- final int index = i++;	73	- }
99	- if (index < text.length()) {	74	+ private void useFastParser(int length) {
100	- position += length;	75	- int i = negative ? currentPosition + 1 : currentPosition;
101	- value = negative ? -calculateValue(i, index) : calculateValue(i, index);	76	-
102	- } else {	77	- final int index = i++;
103	- position = ~position;	78	- if (index < text.length()) {
104	- }	79	- currentPosition += length;
105	- }	80	- value = negative ? -calculateValue(i, index) : calculateValue(i, index);
106	- }	81	- } else {
107	- private int calculateValue(final int i, final int index) {	82	- currentPosition = ~currentPosition;
108	- int startingIndex = i;	83	- }
109	- int calculated = getAsciiCharacterFor(index);	84	- }
		85	
		86	+ private int calculateValue(final int i, final int index) {
		87	- int startingIndex = i;
		88	- int calculated = getAsciiCharacterFor(index);

Figure C.128: insert caption

110		89	
111	-	90	+
112	while (startingIndex < position) {	91	while (startingIndex < currentPosition) {
	calculated = ((calculated << 3) + (calculated << 1)) +		calculated = ((calculated << 3) + (calculated << 1)) +
	getAsciiCharacterFor(startingIndex++);		getAsciiCharacterFor(startingIndex++);
113	}	92	}
114	return calculated;	93	return calculated;

Figure C.129: insert caption

The clearer variable names and re-arranged methods lead me to realize there is a clean separation between two distinct concepts missing in the OffsetCalculator. Much of the work it does results from FEATURE ENVY of the CharSequence. Procedural style code that asks questions such as “does the text begin with a + or -?” would be better answered independently from the calculation.

C.6.5 The Final Form – calculate in the NumberFormatter, NumericSequence

I can decompose the OffsetCalculator back into a data type that provides the necessary functionality on top of CharSequence and a calculation that uses that data type to calculate a value in the NumberFormatter implementation of InternalParser. This aligns most well with the Cognitive Load Theory principle of “write high cohesive content for low-knowledge learners.”

EXTRACT PARAMETER OBJECT.	Browse files
move negative to NumericSequence.	
move currentPosition to NumericSequence.	
encapsulate CharSequence in NumericSequence.	
added Query to see if character at an index is a digit to NumericSequence.	
move limit to NumericSequence.	
COMMAND QUERY SEPARATION on NumericSequence construction and value calculation.	
move getAsciiCharacterAtIndex to NumericSequence.	
move length calculation to NumericSequence.	
replace low-level details with query.	
cleanup.	
pull up check.	
simplify.:	
)	
move new NumericSequence class to top-level.	
separate command and query.	
remove OffsetCalculator and move calculate() back to NumberFormatter.	
simplify fields/	
moved NumericSequence to parsing package.	
move index knowledge to NumericSequence.	
cache calculation of length in constructor.	
simplify.	
Newspaper metaphor.	
realign.	

Figure C.130: insert caption

```
scope cleanup.

Decompose Conditional.

fix-bug-86-experimental-narrative

V committed 4 hours ago 1 parent 99dd48d commit 7b827d49279c4073e623fd85d67ce1ddae3d25ed

Showing 3 changed files with 121 additions and 103 deletions. Unified Split

41 src/main/java/org/joda/time/format/DateTimeFormatterBuilder.java View

@@ -34,6 +34,7 @@
34 import org.joda.time.ReadablePartial;
35 import org.joda.time.field.MillisDurationField;
36 import org.joda.time.field.PreciseDateTimeField;
37
38 /**
39  * Factory that creates complex instances of DateTimeFormatter via method calls.
40
@@ -1301,11 +1302,43 @@ public int estimateParsedLength() {
1301 }
1302
1303 public int parseInto(DateTimeParserBucket bucket, CharSequence text, int
1304 position) {
1305     final OffsetCalculator calculator = new OffsetCalculator(text,
1306     iMaxParsedDigits, iSigned, position);
1307     calculator.calculate();
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
37
```

```

16 -         final int startingPosition) {
17 -             this.text = text;
18 -             final int min = Math.min(maximumDigitsToParse, text.length() -
19 -                 startingPosition);
20 -             startsWithSign = min >= 1 && isSigned && isPrefixedWithPlusOrMinus(text,
21 -                 startingPosition);
22 -             negative = startsWithSign && text.charAt(startingPosition) == '-';
23 -             currentPosition = startsWithSign ? (negative ? startingPosition :
24 -                 startingPosition + 1) : startingPosition;
25 -             // Expand the limit to disregard the sign character.
26 -             limit = startsWithSign ? Math.min(min + 1, text.length() - currentPosition) :
27 -                 min;
28 -         }
29 -
30 -         private static boolean isPrefixedWithPlusOrMinus(final CharSequence text, final
31 -             int startingPosition) {
32 -             final boolean isFirstCharacterOperator =
33 -                 isCharacterOperator(text.charAt(startingPosition));
34 -             final boolean hasNextDigitCharacter = startingPosition < text.length() - 1 &&
35 -                 Character.isDigit(text.charAt(startingPosition + 1));
36 -             return isFirstCharacterOperator && hasNextDigitCharacter;
37 -         }
38 -
39 -         private static boolean isCharacterOperator(final char currentCharacter) {
40 -             return currentCharacter == '-' || currentCharacter == '+';
41 -         }
42 -
43 -         int getCurrentPosition() {
44 -             return currentPosition;
45 -         }
46 -
47 -         int getValue() {
48 -             return value;
49 -         }
50 -
51 -         void calculate() {
52 -             updatePositionAndValue(calculateLength());
53 -         }
54 -
55 -         private int calculateLength() {
56 -             int length = startsWithSign ? 1 : 0;
57 -             while (length + 1 <= limit && Character.isDigit(text.charAt(currentPosition +
58 -                 length))) {
59 -                 length = length + 1;
60 -             }
61 -             return length;

```

Figure C.133: insert caption

```

54 -     }
55 -
56 -     private void updatePositionAndValue(int length) {
57 -         if (length == 0) {
58 -             currentPosition = ~currentPosition;
59 -         } else if (length == 1) {
60 -             useDefaultParser(length);
61 -         } else {
62 -             useFastParser(length);
63 -         }
64 -     }
65 -
66 -     private void useDefaultParser(int length) {
67 -         // Since value may exceed integer limits, use stock parser
68 -         // which checks for this.
69 -         final String toParse = text.subSequence(currentPosition, currentPosition +
70 -             length).toString();
71 -         value = Integer.parseInt(toParse);
72 -         currentPosition += length;
73 -     }
74 -
75 -     private void useFastParser(int length) {
76 -         int i = negative ? currentPosition + 1 : currentPosition;
77 -
78 -         final int index = i++;
79 -         if (index < text.length()) {
80 -             currentPosition += length;
81 -             value = negative ? -calculateValue(i, index) : calculateValue(i, index);
82 -         } else {
83 -             currentPosition = ~currentPosition;
84 -         }
85 -     }
86 -
87 -     private int calculateValue(final int i, final int index) {
88 -         int startingIndex = i;
89 -         int calculated = getAsciiCharacterFor(index);
90 -
91 -         while (startingIndex < currentPosition) {
92 -             calculated = ((calculated << 3) + (calculated << 1)) +
93 -                 getAsciiCharacterFor(startingIndex++);
94 -         }
95 -         return calculated;
96 -     }
97 -
98 -     private int getAsciiCharacterFor(final int index) {
99 -         return text.charAt(index) - '0';

```

Figure C.134: insert caption

```

98 - }
99 -}

84 src/main/java/org/joda/time/format/parsing/NumericSequence.java View
... @@ -0,0 +1,84 @@

1 +package org.joda.time.format.parsing;
2 +
3 +public final class NumericSequence {
4 +    private final CharSequence text;
5 +    private final boolean startsWithSign;
6 +    private final boolean negative;
7 +    private final int limit;
8 +    private final int length;
9 +
10 +    private int currentPosition;
11 +
12 +    public NumericSequence(final CharSequence text, final int maximumDigitsToParse, final boolean isSigned, final int startingPosition) {
13 +        this.text = text;
14 +        startsWithSign = isPrefixedWithPlusOrMinus(maximumDigitsToParse, isSigned, startingPosition);
15 +        negative = startsWithSign && charAt(startingPosition) == '-';
16 +        // Expand the limit to disregard the sign character.
17 +        currentPosition = startsWithSign ? (isNegative() ? startingPosition : startingPosition + 1) : startingPosition;
18 +        limit = Math.min(startsWithSign ? maximumDigitsToParse + 1 : maximumDigitsToParse, text.length() - currentPosition());
19 +        length = calculateLength();
20 +    }
21 +
22 +    private boolean isPrefixedWithPlusOrMinus(final int maximumDigitsToParse, final boolean isSigned, final int startingPosition) {
23 +        final int min = Math.min(maximumDigitsToParse, text.length() - startingPosition);
24 +        return min >= 1 && isSigned && isCharacterOperator(charAt(startingPosition)) && hasNextDigitCharacter(startingPosition);
25 +    }
26 +
27 +    private char charAt(int index) {
28 +        return text.charAt(index);
29 +    }
30 +
31 +    private static boolean isCharacterOperator(final char currentCharacter) {
32 +        return currentCharacter == '-' || currentCharacter == '+';

```

Figure C.135: insert caption

```

33 +    }
34 +
35 +    private boolean hasNextDigitCharacter(int startingPosition) {
36 +        return startingPosition < text.length() - 1 && isDigitAt(startingPosition + 1);
37 +    }
38 +
39 +    private boolean isDigitAt(final int index) {
40 +        return Character.isDigit(charAt(index));
41 +    }
42 +
43 +    private int calculateLength() {
44 +        int length = startsWithSign ? 1 : 0;
45 +        while (length + 1 <= limit && isDigitAt(getCurrentPosition() + length)) {
46 +            length = length + 1;
47 +        }
48 +        return length;
49 +    }
50 +
51 +    public int getIndexOffsetFirstDigit() {
52 +        return isNegative() ? getCurrentPosition() + 1 : getCurrentPosition();
53 +    }
54 +
55 +    public boolean isNegative() {
56 +        return negative;
57 +    }
58 +
59 +    public int getCurrentPosition() {
60 +        return currentPosition;
61 +    }
62 +
63 +    public void addLengthToPosition() {
64 +        currentPosition = getCurrentPosition() + length;
65 +    }
66 +
67 +    public void invertPosition() {
68 +        currentPosition = ~getCurrentPosition();
69 +    }
70 +
71 +    public String getNumberAsString() {
72 +        return text.subSequence(getCurrentPosition(), getCurrentPosition() + length).toString();
73 +    }
74 +
75 +    public boolean hasMoreThanOneDigit() {
76 +        return (isNegative() ? getCurrentPosition() + 1 : getCurrentPosition()) <

```

Figure C.136: insert caption

```
text.length();
+ }
77 +
78 +
79 + public int getAsciiCharacterFor(final int index) {
80 +     return charAt(index) - '0';
81 + }
82 +
83 + public int getLength() { return length; }
84 +}
```

Figure C.137: insert caption

C.7 The Experimental Solution

This has been a lot of work! But the resulting solution is very satisfying. In fact, it requires adding no additional code. The correct solution to fixing the bug actually involves removing some existing confusing code.

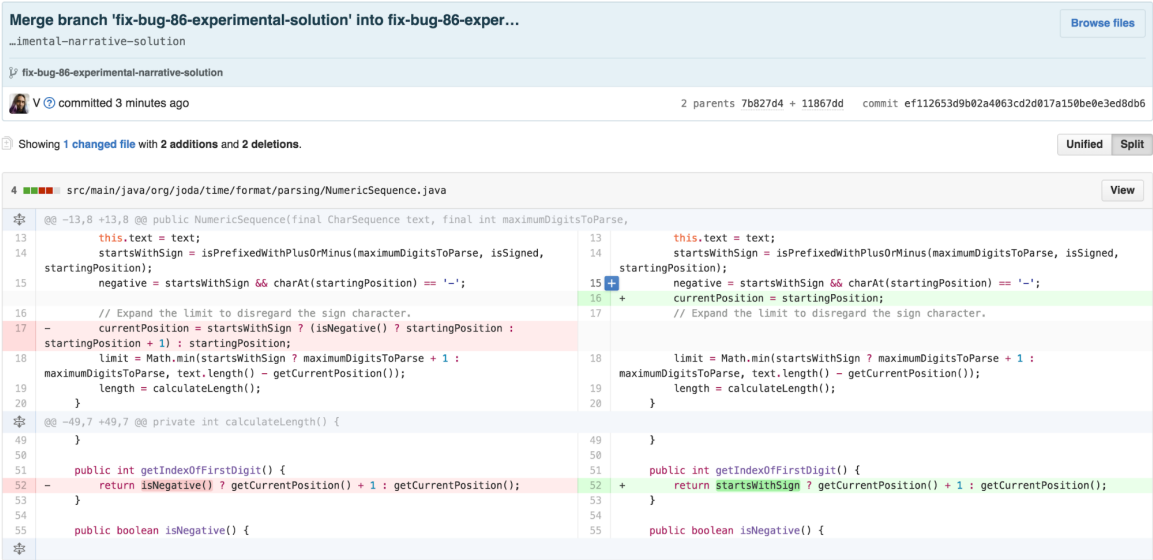


Figure C.138: insert caption

Just like that, the bug is resolved.

C.7.1 Side-by-Side comparison

According to SonarQube 5.4, the metrics for the experimental version are shown below.

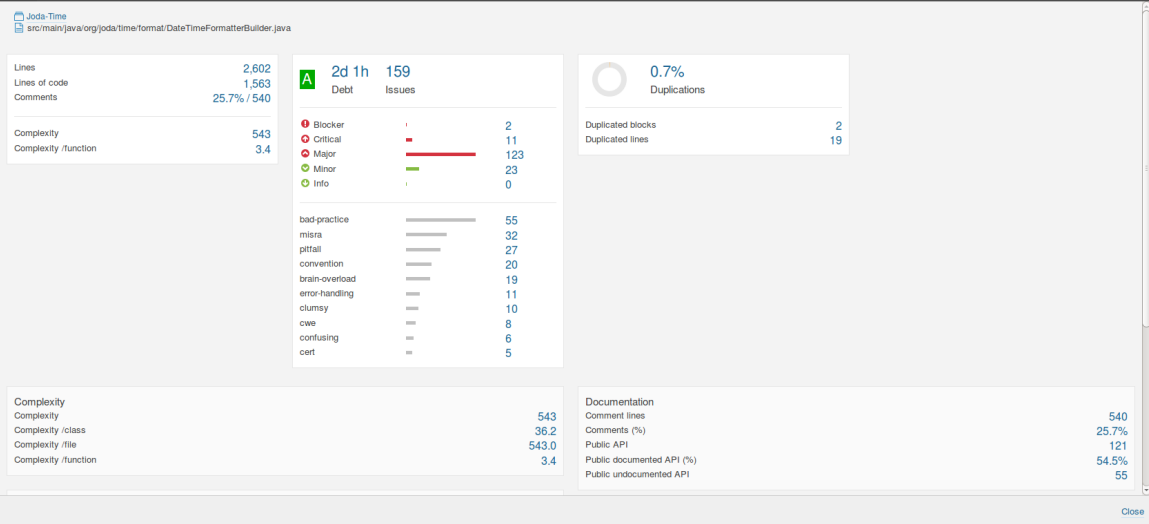


Figure C.139: insert caption

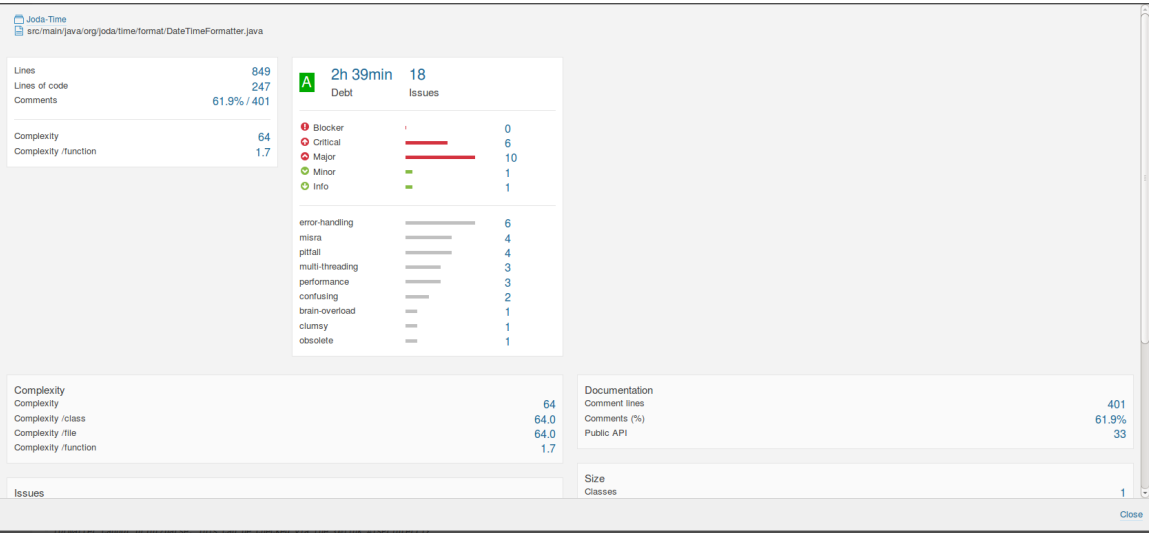


Figure C.140: insert caption

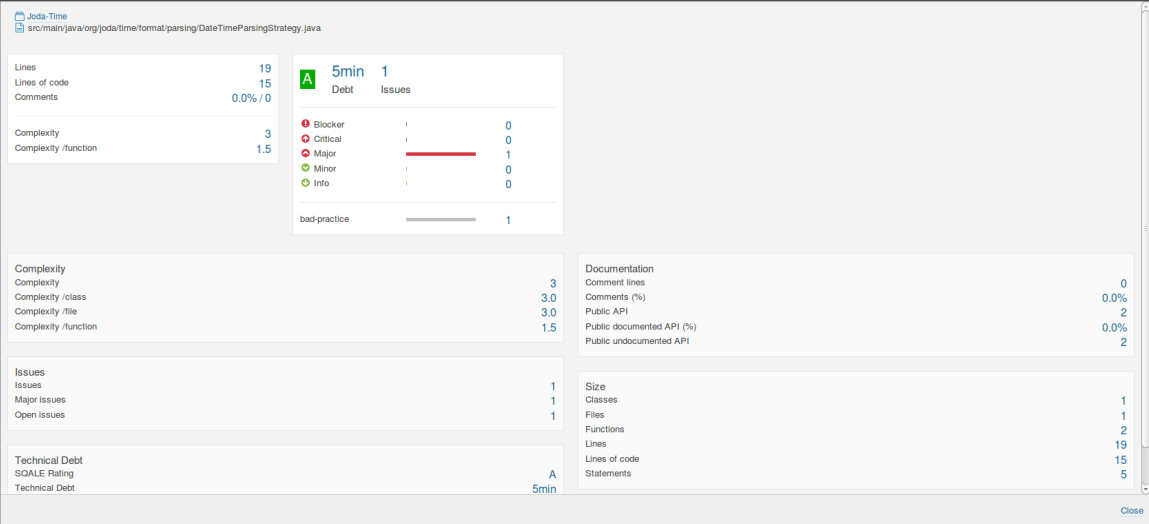


Figure C.141: insert caption

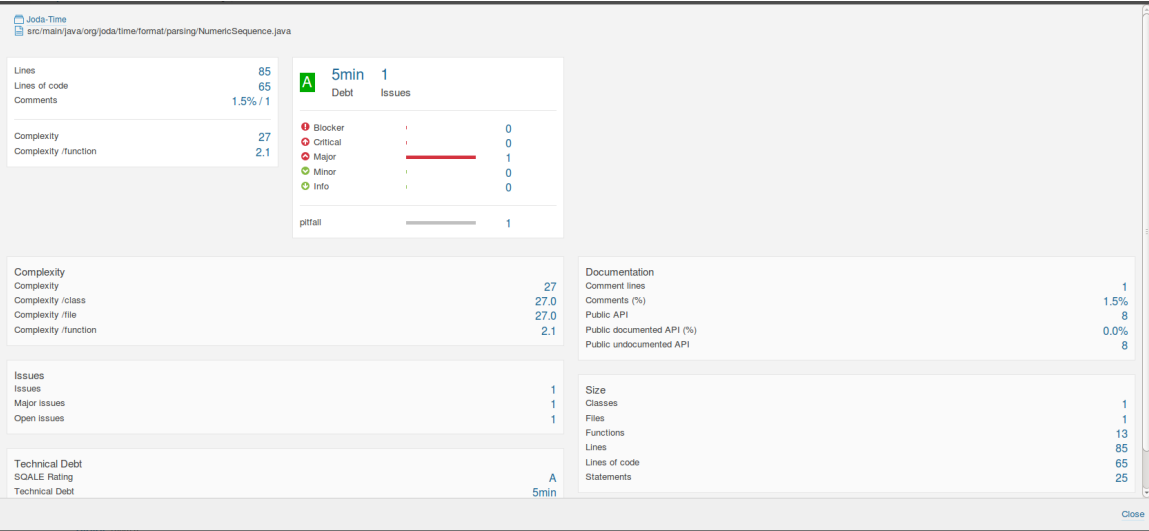


Figure C.142: insert caption

Comparing these with the original `DateTimeFormatter` and `DateTimeFormatterBuilder`;

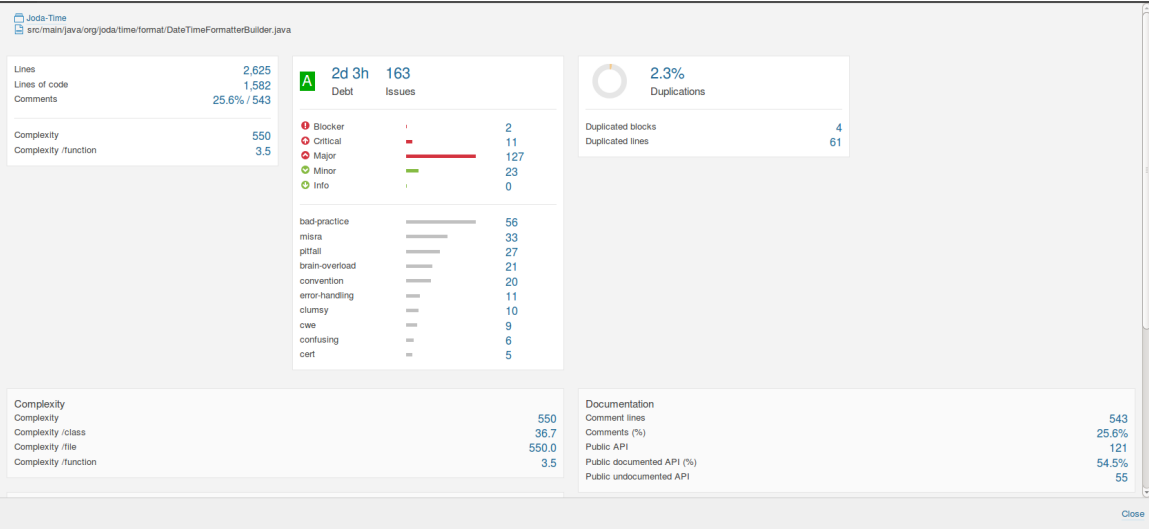


Figure C.143: insert caption

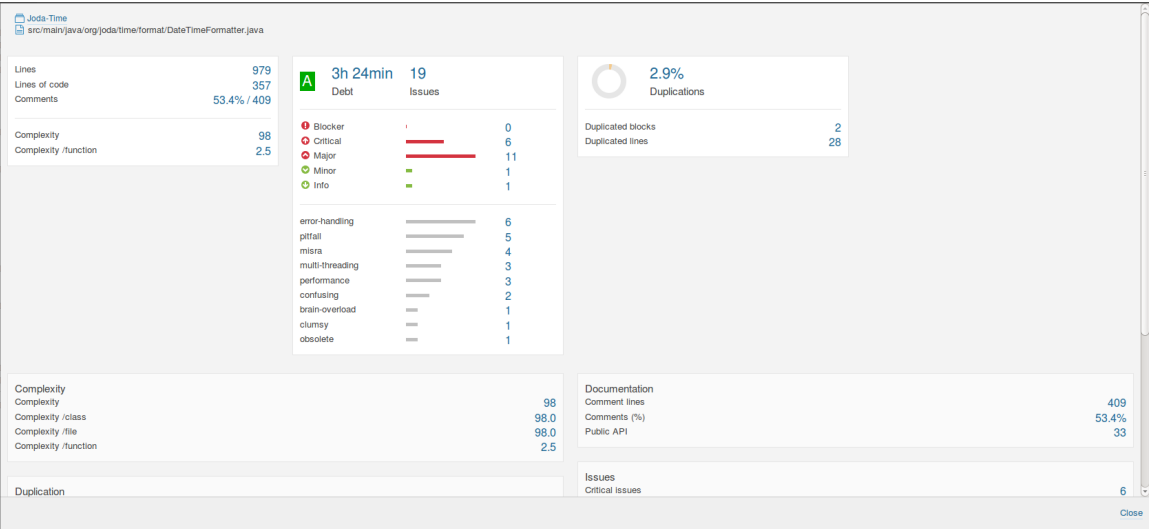


Figure C.144: insert caption

From a pure complexity metrics perspective, it doesn't look like much has changed. `DateTimeFormatterBuilder` went from a control complexity score of 550 to 543 in the experimental. The complexity per function dropped from 3.5 from to 3.4. The number of lines of code dropped by 21. The number of duplications dropped from 2.3% to 0.7%.

`DateTimeFormatter` shows a slightly more pronounced effect. The complexity score dropped from 98 to 64. The complexity per function went from 2.5 to 1.7. The number of lines of code dropped by 110. The 2.9% duplications dropped to 0.

Are these changes enough to show a measurable difference in comprehensibility and debugging time? If current best practice software engineering complexity metrics

are sufficient, I do not expect to find a statistically significant difference. Thus, I am ready to conduct a study.