Technical Report CS17-02

An Empirical Study on the Effects of Code Formatting on Manual Code Proofreading

Matthew Clive

Submitted to the Faculty of
The Department of Computer Science

Project Director: Dr. Kapfhammer
Second Reader: Dr. Jumadinova

Allegheny College
2016

I hereby recognize and pledge to fulfill my responsibilities as defined in the Honor Code, and to maintain the integrity of both myself and the college community as a whole.

Matthew Clive

MATTHEW CLIVE. An Empirical Study on the Effects of Code Formatting on Manual Code Proofreading.
(Under the direction of Dr. Kapfhammer.)

ABSTRACT

There are many different coding styles and guidelines, advanced by different groups of engineers or companies for various reasons, such as promoting mutual understanding and improving version-control consistency. While it is well-known that illogically or haphazardly formatted code is "difficult to read", the exact nature of "good" formatting is often subjective and ill-defined across differing coding styles. This paper presents the results of a small-scale study of code formatting and its effect on human debugging tasks, in which we found anecdotal evidence to suggest that code formatting has little to no effect on code reading ability.

Contents

List of Figures **ix**

1 Introduction **1**

1.1 Motivation 2

1.2 Goals of the Project 4

1.3 Thesis Outline 6

2 Related Work **7**

2.1 Primary Sources 8

2.2 Recent Results 10

3 Method of Approach **13**

3.1 Experiment Overview 13

3.2 Detailed Experimental Considerations 19

3.2.1 Code Formatting Style Configuration Selection 19

3.2.2 Debugging Task Selection Criteria 22

3.2.3 Debugging Task Creation 24

3.2.4 Study Participant Selection 25

3.2.5 Task Administration and Data Collection 25

3.2.6 Data Analysis 26

3.3 The SkriptIV System 28

3.3.1 System Architecture Overview 28

3.3.2 Running the System 29
3.3.3 Remote Research Database 30
3.3.4 Test Administration 30
3.4 Threats to Validity 33

4 Experimental Results **35**
4.1 Summary 35
4.2 Detail 35

5 Discussion and Future Work **39**
5.1 Summary of Results 39
5.2 Future Work 40
5.3 Conclusion 41

Bibliography **43**

A Complete Experimental Dataset **45**

B Miscellaneous Source Files **48**
B.1 Mathematica^{TM} Data Analysis Workbook 48
B.2 defects-compactify.pl 50
B.3 reportbuilder.js 51

C SkriptIV **54**
C.1 Skriptiv License 54
C.2 package.json 56
C.3 bower.json 57
C.4 main.js 58
C.5 integrityChecker.js 59

C.6 index.html 65
C.7 my_components/appincludes.html 66
C.8 my_components/accurate-timer/accurate-timer.html 68
C.9 my_components/app-frame/app-frame.html 71
C.10 my_components/consent-form/consent-form.html 80
C.11 my_components/database-input-page/database-input-page.html . . . 83
C.12 my_components/demographics-survey/demographics-survey.html . . . 86
C.13 my_components/error-display-box/error-display-box.html 90
C.14 my_components/multi-radio-question/multi-radio-question.html . . . 94
C.15 my_components/pre-test-page/pre-test-page.html 96
C.16 my_components/test-question/test-question.html 100
C.17 my_components/thank-page/thank-page.html 114

D SkriptIV Data Files **116**

D.1 Defects-Sources/tasks.json 116
D.2 Defects-Sources/Chart-5/info.json 117
D.3 Defects-Sources/Chart-8/info.json 118
D.4 Defects-Sources/Lang-1/info.json 119
D.5 Defects-Sources/Math-26/info.json 120
D.6 Defects-Sources/Math-59/info.json 121
D.7 Defects-Sources/Mockito-18/info.json 122
D.8 Defects-Sources/Time-15/info.json 123

E Manually Formatted Source Files **124**

E.1 Manually Transformed Sources for Task Chart-5 124
E.1.1 Defects-Sources/Chart-5/XYSeries-bugged-A.java 124
E.1.2 Defects-Sources/Chart-5/XYSeries-fixed-A.java 145

E.1.3 Defects-Sources/Chart-5/XYSeries-bugged-B.java 165
E.1.4 Defects-Sources/Chart-5/XYSeries-fixed-B.java 187
E.2 Manually Transformed Sources for Task Chart-8 209
E.2.1 Defects-Sources/Chart-8/Week-bugged-A.java 209
E.2.2 Defects-Sources/Chart-8/Week-fixed-A.java 227
E.2.3 Defects-Sources/Chart-8/Week-bugged-B.java 245
E.2.4 Defects-Sources/Chart-8/Week-fixed-B.java 264
E.3 Manually Transformed Sources for Task Lang-1 283
E.3.1 Defects-Sources/Lang-1/NumberUtils-bugged-A.java 283
E.3.2 Defects-Sources/Lang-1/NumberUtils-fixed-A.java 324
E.3.3 Defects-Sources/Lang-1/NumberUtils-bugged-B.java 366
E.3.4 Defects-Sources/Lang-1/NumberUtils-fixed-B.java 409
E.4 Manually Transformed Sources for Task Math-26 452
E.4.1 Defects-Sources/Math-26/Fraction-bugged-A.java 452
E.4.2 Defects-Sources/Math-26/Fraction-fixed-A.java 471
E.4.3 Defects-Sources/Math-26/Fraction-bugged-B.java 490
E.4.4 Defects-Sources/Math-26/Fraction-fixed-B.java 510
E.5 Manually Transformed Sources for Task Mockito-18 530
E.5.1 Defects-Sources/Mockito-18/ReturnsEmptyValues-bugged-A.java 530
E.5.2 Defects-Sources/Mockito-18/ReturnsEmptyValues-fixed-A.java 534
E.5.3 Defects-Sources/Mockito-18/ReturnsEmptyValues-bugged-B.java 538
E.5.4 Defects-Sources/Mockito-18/ReturnsEmptyValues-fixed-B.java 543
E.6 Manually Transformed Sources for Task Time-15 548
E.6.1 Defects-Sources/Time-15/FieldUtils-bugged-A.java 548
E.6.2 Defects-Sources/Time-15/FieldUtils-fixed-A.java 558
E.6.3 Defects-Sources/Time-15/FieldUtils-bugged-B.java 568

E.6.4 Defects-Sources/Time-15/FieldUtils-fixed-B.java 578

List of Figures

3.1 "Allman" style braces 17

3.2 "Egyptian" style braces 18

3.3 Task Assignment Algorithm 18

3.4 `SkriptIV` technical architecture overview 28

3.5 Commands to install `SkriptIV` dependencies 29

3.6 Command to launch `SkriptIV` 30

3.7 The database input page in `SkriptIV` 31

3.8 The survey administered as part of our experiment 31

3.9 An example of the prompt that appears before a task is shown. . . . 32

3.10 The testing interface of `SkriptIV` 32

4.1 Result of the Friedman-Rank test applied to participant response time 36

4.2 Participant response time by formatting configuration 37

4.3 Participant correct responses by formatting configuration 38

Chapter 1

Introduction

Computer scientists are frequently concerned with the "correct" way to format, read, write, and style program source code. Although any syntactically correct program will function, even early texts concerning the operation of the then newly-invented EDSAC machine suggested that some sequences of instructions were easier to comprehend than others [18]. Throughout the history of computer science, there are published recommendations concerning programming best-practices, such as the influential 1982 text "The Elements of Programming Style" by Kernighan and Plauger [6]. Today it is possible to locate any number of publicly available, published styles for a given programming language, such as the Sun Microsystems Code Conventions for the Java^{TM} Programming Language document [17], or the Google Style Guide for Java^{TM} document [3].

Previous work has demonstrated that program readability and formatting style plays a role in comprehension by human readers [1], but different studies have come to contradictory conclusions about what best practices to follow in order to increase comprehension [9, 16]. Some academics have even called these prior studies' methodologies into question [12].

This paper presents the results of an empirical study on the effects of code formatting on manual code proofreading (i.e. debugging) tasks. After performing our

experiment, we have been able to show anecdotal evidence to suggest that code formatting style does *not* affect the speed or correctness of such code proofreading tasks.

1.1 Motivation

Human minds operate quite differently from a computer, and while it has always been necessary to transform human ideas into a language suitable for machines since the introduction of the modern digital computer, much thought has been put into easing the burden of human workers by making programs easier to read. Even early texts concerning some of the first modern digital computers noted the importance of readable and relatable source code, such as this aside from the 1951 text, "The Preparation of Programs for an Electronic Digital Computer" by Wilkes et al. [18] *emphasis added.*

> It is important to realize that the relation between the form in which orders are punched on the tape and the form in which they appear in the store is determined solely by the initial input routine. By making the two forms more similar [...], it would be possible to simplify the initial input routine. There would, however, be no advantage in doing this, and it would mean that **more work would be left to the programmer**. [...]
>
> It follows that the choice of initial input routine, and thus of the form in which orders are punched, is a matter for careful consideration, since **upon it depends the ease with which all programs are constructed**. [18]

This particular passage concerns itself with the form that input takes on a program tape or punch card stack in relation to the original machine instructions for the EDSAC computer, yet even this early text underscores an important theme in

software development knowledge: program style and readability matter. Therefore, in the interests of supporting program portability, maintenance, and compatibility, industry professionals have continuously published standards and conventions for the written presentation of program source code [3][8][17].

The trouble with these existing, competing styles is that they offer little to no justification for why any particular stylistic choice should be preferred over any other. These style guides offer their statements and prescriptions as though they were normative, with little rationale or evidence to suggest how choosing one particular choice affects the overall quality of any piece of code. As an example, the previously-mentioned Sun Code Conventions states that source code should be indented in four spaces with tabs eight spaces wide [17], whereas Google's document prescribes a strict two space indentation amount, and tab characters are not allowed [3]. Sometimes these formatting decisions go unjustified even if and when disparate style guides *agree* on the exact implementation of a certain stylistic feature. Such guides may reference historical traditions — for example, in the use of the phrase "conventional indentation" in the Sun style guide [17] — or aesthetic reasoning for their decisions. Additionally, these style guides also frequently disagree without any apparent justification, such as insisting on different indentation styles such as 8 column tab characters [17] versus 2 column spaces [3].

Additionally, the requirements of these guides should tend to evolve over time because of the continuously changing landscape of software development. For example, the Sun Code Conventions for the Java^{TM} Programming Language, originally published in 1999, mentions specifically a hard column limit of 80 characters in printed source code due to the limitations of character terminals at the time [17]. This limitation is a problem which is unlikely to hinder a programmer working on a computer today, as most modern computers feature displays which can comfortably render

many more than 80 characters per row.

These findings should be concerning, because the existing body of research which has investigated the efficacy of code formatting as a means to improve the form and function of a written program has been shown to be frequently contradictory, flawed, or incomplete [12]. Some aspects of code formatting that have been stipulated in existing code style guides have never been adequately studied in a scientific context, while other aspects such as code indentation have been studied repeatedly, but each study has come to a completely contradictory conclusion [9, 12, 16]. Despite these contradictory results, it happens to be widely-held knowledge that poorly formatted source code is more difficult to read than "conventionally" formatted source code [7].

1.2 Goals of the Project

Because code readability significantly impacts programming tasks which require reader comprehension, it follows that unit test comprehension and code debugging tasks should be among those strongly affected by code readability [2]. Additionally, it has been found that even when debugging code with the assistance of an Integrated Development Environment (IDE) which shows multiple representations of the code (such as a pane showing program variables and a diagram of program dependencies), the best performers of debugging tasks referenced only the code and some dynamic visualizations much more often than other types of visualizations or debugging aides [4]. Together, these findings show that code comprehension, and therefore the presentation of written code, is vital to the task of debugging faulty programs.

For the purposes of this work, we shall define the term *code formatting*, referring to the context where code is read and written by human programmers, to be the amount and method of using horizontal indentation, the use of line wrapping to maintain a maximum line length, the placement of blank lines and comments, and use

of superfluous horizontal and vertical whitespace. We chose this definition carefully to collect together those details of program presentation which are related only to layout and whitespace. Whitespace is not syntactically significant in many programming languages, with the notable exception of Python (and other languages like it), and therefore the semantic meaning of a program in these languages is not changed based on the presence, absence, or modification of this code formatting. Code formatting, as defined here, is then only significant to a human reader, who may use regular formatting as a design aid, and is *not* significant to the compiler or to the semantics of the program.

This paper aims to answer the following two research questions:

RQ.1 We seek to verify that regular code formatting impacts performance of debugging tasks by identifying a statistically significant decrease in both speed and accuracy of participants when performing our debugging tasks when an irregular 'control' format is introduced.

RQ.2 After the finding of RQ.1, we seek to identify any significant increase in speed and accuracy of participants when viewing one of two separate formatting variants based on popular coding standards. Reference section 3.1 for a comprehensive discussion of the format variants.

Therefore, we hypothesize the following:

RH.1 We will find that participants perform significantly slower when presented with the irregular 'control' code format as compared to the other two formats.

RH.2 We will find that participants tend to favor one of the code formats based on popular coding standards, performing slightly faster and with improved correctness when presented with this format as compared to the other two formats.

1.3 Thesis Outline

Chapter 2 reviews a number of past approaches to rigorously studying code formatting and style, and summarizes their strengths and weaknesses. Chapter 3 outlines the method of approach used to establish the results, while chapter 4 provides a detailed and in-depth analysis of the experimental results obtained. Chapter 5 concludes the work by presenting a concise summary of the results, the anticipated impact of the work, and a discussion of areas for future research.

Chapter 2

Related Work

Code style was and is a concern for developers, as poorly written code presents barriers in both the learning and development of new applications. In addition, more readable code may promote the effectiveness of human proofreading and verification, increasing a program's maintainability over time. The overall legibility of code, and the effects that increased legibility might have on the performance of both programmers and the quality of the code itself have been studied with a passing interest, however often the focus in research remains on structural differences in code, rather than its typographical presentation.

Research which does analyze typographical presentation is both old and contradictory. First, it is significant that the research is old, because the tools and languages involved in programming have changed significantly since the time that these studies were conducted — the question arises whether these findings are still applicable today with modern tools and languages. Second, the fact that some of these studies contradict one another shows that more research should be conducted in this area in order to determine fact from fiction. If code formatting can be shown to be significant in some way, it would behoove us as Computer Scientists to empirically verify the type and nature of this significance.

2.1 Primary Sources

A small body of work evaluating the significance and precise nature of code was published in the 1980s. A summary of this work by authors Oman and Cook primarily serves as a listing and critique of existing research performed at the time [12]. The work by Oman and Cook identifies two strong trends, where code as a typographical work was being studied by some, and others were creating tools to measure the "quality" or "style" of a given piece of code. The authors note that studies performed on the style grading tools could not find significant advantages to employing those tools, and studies focused on typographical code formatting — of the type we are focusing on for this particular study — utilized questionable methodology and were frequently contradictory.

Of particular note for the purposes of our study were two papers specifically cited by the Oman and Cook critique: one by Miara et al., which found that the ideal code indentation length was between 2 to 4 spaces of indentation [9], and another paper edited by Sheil, which found that indentation does not affect code review performance at all [16].

The paper by Miara et al. was a specific user case study conducted as a survey, collecting subjective and objective metrics [9]. The study collected both subjective user scores of program readability, as well as responses from study participants to general comprehension questions, as well as responses to a bug-identification task. In summary, Computer Science students who were proficient in Pascal were categorized into two groups, novices and experts, based on their level of prior programming experience. These students were then given a paper-based quiz consisting of two programs in Pascal, each presented with varying levels of indentation, and some specific comprehension questions. Experimental data collected included the results of the comprehension quiz, specifically including a question involving the identification

of a bug in the presented programs, and a subjective report by the participants of the programs' readability. The study found that 2 to 4 spaces of indentation significantly improved their program comprehension metric, while no indentation lowered this metric. This result seems to indicate that pure typographical presentation of code can help or hinder overall program comprehension.

The other work is instead a specific section of a much larger survey of programming practices approached from the perspective of the psychological sciences [16]. The study of indentation is merely a small portion of the findings collected by this research group, some of which is a review of other available research. In a section summarizing research concerning indentation, this particular review finds evidence that suggests that while subjective measures of program readability are improved via indentation, there were prior studies that found no supporting evidence that indentation affects the performance specific programming tasks, such as identifying a bug or modifying a program. This result seems to indicate that pure typographical presentation of code has no material effect on any type of program comprehension, merely that it enhances the aesthetic of the code.

The fact that the results of these experiments can be completely contradictory should raise a number of questions. Oman and Cook state that more work may be required in this area in order to determine any useful conclusions, and hypothesize that the earlier studies' methodology may be flawed in some way [12]. The precise nature of the hypothesized flaw, however, is never explicitly discussed.

In a completely separate work originating from around the same time period, a study conducted by Newsted et al. in 1981 shows that metrics relating to code readability and style do correlate well with improved performance in code debugging [11]. The authors investigated the impact of COBOL programming style on debugging tasks by asking university students to participate in a short survey after they had

taken a programming class utilizing the COBOL language. The students were asked questions such as whether they used additional diagrams as part of their development process, how often they commented their code, how many errors they had encountered when compiling a program for the first time, and how long they took to work on a coding problem in general. The survey responses were then correlated with students' grades to try and determine if any coding style features were indicative of an individual student's success. The study concluded by finding that design aids did correlate with better performance, and that based on their evidence, future studies would likely have success demonstrating that the practice of good style correlated strongly with debugging ability.

This result seems to again suggest that code formatting has a strong correlation with better human performance when comprehending code. In our opinion, it is interesting that this study finds a strong correlation between coders who write with good style and higher performing debugging ability, but this does not inform us precisely whether it is the case that good style enhances debugging ability, or whether good style is a behavior that exceptionally performing programmers merely *exhibit*. It is our belief that this is an important distinction that should be made, and whose verification one way or another may prove useful to the discipline of Computer Science as a whole.

2.2 Recent Results

Though the primary examples of studies involving typographical code formatting are older works, there is plenty of evidence that software readability and style remains a topical concern in the field of Computer Science today. Though recent research does not focus specifically on whitespace or purely typographical formatting as we do, there is clear evidence that readability is a consideration that should be attended

to in current and future research.

Software readability is an important consideration when it affects the learning and adoption of new technology, as evidenced by a 2012 work by Santos and Macedo [14]. Throughout this paper, the authors cite an increase to the readability of code as one of the major motivations for developing their improvements to the CUDA^{TM} API interface. CUDA^{TM} is a technology which allows for general purpose programming on a graphics processing unit, which enables highly parallel and performant operations on large vectors and matrices. However, the type of programming required to use CUDA^{TM} effectively is very different than traditional programming for a CPU. The authors reference CUDA^{TM}'s current lack of readability as a significant barrier to developers writing applications in CUDA^{TM}, and the authors speculate that improvements to readability would also benefit the learning curve of those attempting to learn CUDA^{TM} programming.

In another example, authors Daka et al. modeled code readability in unit tests, with the hypothesis that illegible or otherwise obtuse unit tests would be harder to maintain [2]. In their study, they modeled readability in a generally broad sense, utilizing a machine learning metric for code readability which was tuned by human input taken from a survey. The authors collected a set of readability data from the survey data, They then applied their machine learning model to the EvoSuite automatic unit test generation framework, and concluded that in general users preferred the authors' more readable tests compared with other automatically generated tests. However, their model for code readability was quite limited, the study only focused on automatically generated unit tests, and there were quite a few open questions remaining after the study had concluded. One of those remaining questions was what specific factors influenced human understanding of the generated tests specifically, rather than just a general sense of "readability".

Though the work on readability in unit tests does not specifically reference typographical style, it does find that developers prefer code which is more readable than code which is less so, even when both types are equally correct. It remains to be seen whether such a preference translates into better overall programmer performance.

A different paper by Posnet et al. presents an analysis of data on code readability gathered from large survey data [13]. In this study, subjects were presented with code segments and asked to rate how readable they thought they were. The results were then subjected to statistical analysis and used to create models for predicting readability scores. The goal of the study was to isolate those features of code segments which were most predictive of high readability. The authors found a correlation between code size, code entropy, and human readability.

This study also did not focus primarily on typographical formatting as a signifier of high readability, but does find that the size and complexity of code matters in terms of readability. It may be interesting to investigate in a future work whether or not typographical formatting influences how significant the effects of code size or entropy can be on a subjective readability metric.

Chapter 3

Method of Approach

Our approach focused primarily on the construction and administration of a user study, whose parameters were carefully chosen to test a number of key formatting variables. This chapter provides a complete overview of the experimental methods, beginning with a high-level overview of the experiment as a whole (found in section 3.1), then proceeding with a full and detailed discussion of the chosen experimental parameters, their impact on this work, and any particular limitations imposed (found in section 3.2), then a discussion of merits and potential threats to validity of the chosen method (found in section 3.4).

The experiment was administered electronically by a program written in HTML5 named `SkriptIV`. An overview of the `SkriptIV` system itself is found in section 3.3, while the complete `SkriptIV` source code can be examined as Appendix C.

3.1 Experiment Overview

The experiment conducted was a user study which collected data from human participants engaged in performing a number of debugging tasks. Before completing any tasks, test subjects took a short questionnaire which sorted their code reading ability into the generally subjective categories of 'novice', 'intermediate', and 'expert'.

Subjects then proceeded to complete 7 of our debugging tasks, taking no more than 35 minutes to do so, with 5 minutes allotted per task. Both the survey and the testing portion were administered by a computer application which subjects used to record their responses, and each test was proctored by a researcher who ensured that participants followed the experimental procedure properly.

Each task was a fault identification activity, in which participants were shown side-by-side two code segments written in the Java programming language, one of which contained a software fault, while the other did not contain a fault. Participants were presented with a human-readable description of the type of fault, including an example of code that may trigger the fault, and then participants were asked to identify which of the two code segments was faulty, as well as to identify the line of code on which the fault appeared. Participants were timed in the completion of their tasks, and were allowed no more than 5 minutes per task. A participant who met the 5 minute time limit was considered to have delivered an incorrect response, and moved on to the next task.

Our independent variable in this experiment was the code formatting of the side-by-side code segments which were part of a particular task. Participants completed a total of 7 tasks, each of which could have been shown with code formatting in one of three particular formatting styles, or *configurations*. For the ease of discussion, we shall label these configurations as A, B, and C. Configurations A and B were derived by choosing several conflicting elements of well-known coding style guidelines, while Configuration C was specifically manufactured for the purposes of the experiment, to be used as a control configuration. The very first task shown to participants was always the same and presented in configuration C. This first task's purpose was to familiarize participants with the computer-administered testing interface, and hence its results were discarded, while the remaining 6 tasks were assigned configurations

randomly such that each participant completed 2 more tasks of each configuration, e.g. 2 tasks in Configuration A, 2 tasks in Configuration B, and 2 tasks in Configuration C. A summary of the particular elements of the formatting style used in Configurations A and B is shown in Table 3.1.

All of the source code used as part of the fault identification activities, in each configuration A, B, or C, were crafted beforehand by the researchers. Configurations A and B were generated by hand for each original source set for each task, while configuration C was the result of running an automated script.

	Configuration A	**Configuration B**
Line-Length	70 characters in block comments, otherwise 80 characters	100 characters
Vertical Whitespace	Two blank lines: 1. Between sections of a source file 2. Between class and interface definitions One blank line: 3. Between methods 4. Between the local variables in a method and its first statement 5. Before a block or single-line comment 6. Between logical sections inside a method to improve readability.	One single blank line appears: 7. Between statements, *as needed* to organize the code into logical subsections. 8. Before the first member or after the last member of the class. 9. Between consecutive members (or initializers) of a class 10. **Exception:** There is no blank line between two consecutive fields (having no other code between them). 11. For the purposes of rule 10, a Javadoc comment shall count as "code" between fields, therefore requiring a separating blank line as per rule 9.
Placement of Braces (a.k.a. "curly-brackets")	"Egyptian Style" braces which appear with opening braces on the same line as the preceding statement and closing braces on their own line, as shown in the example in Figure 3.2.	"Allman Style" braces, which appear on their own line, at the same indentation level as the *surrounding* code, as shown in the example in Figure 3.1.

Table 3.1: Formatting Configuration Summary

```
class Main
{
    public static void main(String[] args)
    {
        int x = 3;

        if(x == 3)
        {
            //do something
        }
        else
        {
            //do something else
        }
    }
}
```

Figure 3.1: "Allman" style braces

Configuration C was specifically designed as a control to determine whether or not variance in response time or accuracy was due to code formatting. Configuration C was implemented as a Perl script, the complete source of which is given in appendix B.2. In summary, the Perl script strips leading and trailing whitespace from the original source files, and if an even-numbered line ends in an opening bracket ('{') character, it inserts a newline character before the bracket, moving the it down onto its own line.

```
class Main {
    public static void main(String[] args) {
        int x = 3;

        if(x == 3) {
            //do something
        } else {
            //do something else
        }
    }
}
```

Figure 3.2: "Egyptian" style braces

For each participant, the assignment of tasks for the testing portion proceeded according to the algorithm presented as Figure 3.3. In summary, we start with a desired list of tasks we wish to run and a list of configurations we wish to test. Then, we shuffle both lists and assign completely configured tasks as pairs of both a task and a configuration. The particular implementation used for this experiment, written in JavaScript, can be found as part of the `ready()` function found in the App-Frame source file, included as Appendix C.9

Input: A list T of desired tasks and a list C of desired task configurations, both of length N.
Output: A final list O of configured tasks to be administered as a test.

$t \leftarrow \text{shuffle}(T)$;
$l \leftarrow \text{shuffle}(L)$;
$O = \emptyset$;

for $i \leftarrow range\,[1, N]$ **do**
  O append (t_i, l_i);
end

Figure 3.3: Task Assignment Algorithm

Our dependent variables were the speed of test subjects' responses and the accuracy of those responses. Speed of response was measured in seconds to a resolution of one second, while accuracy was measured in the following way: either the response was correct by identifying both the correct code segment containing the fault *and* the precise location of the fault, receiving a value of 1, or the response was incorrect and did not identify the correct location or code segment containing the fault, receiving a value of 0. Participants who did not answer in the allotted time of five minutes were considered to have delivered an incorrect response.

3.2 Detailed Experimental Considerations

While the previous section provided a summary view of the experiment as a whole, several important experimental considerations were omitted from the summary in the interests of brevity and simplicity for the summary itself. This section, then, expands upon the earlier summary and provides a complete description of each and every stage of the experimental process.

3.2.1 Code Formatting Style Configuration Selection

Code formatting, and by extension the more broad category of code style, is central to the research questions put forth by this study. Therefore, careful consideration of what particular elements of style should be examined during this study is an essential part of our experimental design. In an ideal world, it would be possible to closely examine a vast myriad of different formatting style configurations, yet sadly time and study participants are relatively constrained resources. Therefore, our study needed to strike a delicate balance, producing useful data while working within this constrained environment.

One of the primary critiques of previous research in this area by the earlier Oman and Cook paper was that programming style was in itself ill-defined [12]. For this reason, we chose to carefully examine available published work on the subject to determine stylistic variables which were likely to be good candidates for empirical research. Upon searching related academic work available to us (see chapter 2), it was apparent that several formatting variables had been previously identified. From this research we discovered a great number of potential code formatting variables, however, it was ultimately determined that we would focus our study on a narrow subset of these variables. Those remaining variables chosen were: Line Length, Vertical Whitespace, and Placement of Braces.

With the resources available to us, it was determined before the experiment began that 45 minutes was approximately the maximum amount of time that it would be reasonable to engage a particular study participant, and as such, five minutes would be about the maximum amount of time we could reasonably expect a participant to spend on any particular task. Assuming that participants' consent forms and any other demographic data that we collected before beginning the study proper took about 10 minutes to complete, this left us with an estimated 35 minutes of time to conduct an experiment, allowing us only 7 tasks per study participant according to these constraints. It was this time constraint in particular that required us to limit ourselves to studying only three possible variations for each of our code formatting variables. Thus, we needed to construct two competing style types, as well as a relevant control style.

Additionally, we faced the challenge of needing to carefully select code formatting configurations which would be relatively representative of real code for the first two styles. We did not want to conduct a study by manufacturing possible variations of each of our formatting variables; rather, we attempted to create a study focusing on

situations that might be encountered outside of an academic or research setting. For this reason, we chose to select relevant formatting variable values only from well–known, published coding style guides. These guides were selected based on their prevalence, public availability, and relative completeness in describing as many elements of a program's style as possible. For the Line–Length and Vertical Whitespace categories, we chose styles reminiscent of the Sun Microsystems Code Conventions for the Java^{TM} Programming Language published coding standard [17] and the the Google Style Guide for Java^{TM} published coding standard [3].

Unfortunately, both the Sun Coding Conventions and the Google Style Guide agreed on the "Egyptian" style placement of braces, and so a third source was required. The choice of including "Allman" Style braces in task configuration B was ultimately influenced by programming guidelines published by Microsoft for the C# programming language [10].

The control formatting style configuration, configuration C, was later implemented in the form of a Perl script which strips most of the whitespace from a single source code file input, and which moves opening bracket characters on the same line as other code onto their own line if the line number is even. The source for the Perl script is presented as Appendix B.2, and for convenience and consistency, configuration C is defined to be the output of this script.

Finally, it is important to the purposes of the study to note that while we have affirmatively set several code formatting variables within each task configuration, we explicitly do *not* take a stance on any other code formatting concerns within each task. When a source file being used for a task is transformed into one of the configurations that we have presented, and a situation arises which is not explicitly described in the description of the task configuration, a *best effort* is made to remain consistent with whatever stylistic decision was present in the original source file. In future work, it

may be useful to make a more rigorous definition of each style configuration with even less ambiguity.

3.2.2 Debugging Task Selection Criteria

Another primary concern for the design of this study was the representativeness of the software faults which would be presented as tasks.

As part of the experimental design, we intended to focus as much as possible on tasks which would be representative of real working conditions for a software developer or programmer. To this end, we leveraged the Defects4J database of existing software faults for curating the fault-finding tasks to use as part of our study. The Defects4J database, first presented in a paper by Just et. al., is a curated database of software faults in the Java programming language sourced from various open-source projects. The database contains specifically curated information about software faults which have been found in real open source code as the result of normal software development, and contains faults for six separate open source projects [5].

Each fault documented in the Defects4J database has several properties which are useful for the purposes of this study. Each fault must have been found in the version-control tree for the original open-source project, must have been documented as part of a bug-tracking system for the project, and must have a unit test available which documents the behavior of the fault. The unit test must fail when the fault is present, and be made to pass by a single patch to the source code, resulting in a fixed version of the software.

These properties are extremely useful to the construction of our study, as we can leverage this database to have immediate access to software faults which:

1. Are representative of faults which could be encountered in the course of real software development. Each fault which forms the basis of a task in this study

has been found published within the version control system of an open source software project.

2. Have a verifiable "faulty" and "fixed" version. Available to us as part of the Defects4J database are full source-code examples of code which contains a fault, and a version of the source with a *minimal* patch applied which fixes the fault. Therefore, the only difference between such a "faulty" and "fixed" version of code stored within the Defects4J database is a patch which directly addresses the fault in question. No code is changed which is not directly relevant to the fault.

3. Have documented "correct" and "incorrect" behavior. As a result of the requirement that there is a unit test available for a fault to be included in the Defects4J database, we have access to clear and *often*[1] unambiguous documentation of the desired behavior of the code in question.

While it was useful to begin with the Defects4J database for these reasons, a further curation of the available faults in the database was necessary for our work. In addition to the properties already mentioned, which all faults in the Defects4J database share, we also desired several other important properties for faults to include in our study. Our additional criteria were:

4. Each fault which would be selected for a task should only be present in one source file only. That is, the complete patch fixing a documented fault should be contained within a single source file.

[1] As mentioned by a separate paper published by Shamshiri et. al. investigating the Defects4J Database, some software faults' desired behavior may be *under–specified* by a simple test suite, and thus the test does not completely describe the required behavior [15]. However, for the purposes of this study, we are not concerned with this effect.

5. Each fault which would be selected for a task should not require domain–specific knowledge to understand. For example, we expect that test participants would understand simple functionality such as a *min* function, or would understand how to debug code which generated `NullPointerException`, but that faults involving more complex behavior reliant on the specific problem domain of the given open-source project would not be good candidates for research.

Eventually, we were able to select 7 faults from the Defects4J database, each fault matching the complete set of criteria, sampled roughly evenly across the open source projects included in the database. We selected two faults from the JFreeChart project, two faults from the Apache Commons–Math project, one fault from the Apache Commons–Lang project, one fault from the Mockito project, and one fault from the Joda-Time project.

3.2.3 Debugging Task Creation

After the initial task selection process, it was still necessary to construct additional source files for each configuration of every task. The Defects4J database provided us with the original faulty and fixed source files, but to run our study we needed to present the files formatted according to our differing format configurations.

During initial research for this study, we could not locate an automated code formatting tool which was flexible enough to format each source file according to our selected formatting rules while leaving the remaining formatting intact. For this reason, the transformation of task source code into the different code configurations was performed manually, generating several additional source code files for each possible task configuration.

Although some of the stylistic rules defined in our code configurations are subjective and therefore cannot be easily expressed in a machine-readable form, as an aid

to consistency in our manual source code transformation process, some machine verification was performed. A JavaScript file, `integrityChecker.js`, was incorporated into the `SkriptIV` test-administration program which verifies several of the properties of each given configuration and ensures that all source code files pass these selected tests before the test is administered. The source code for the `integrityChecker.js` program is presented as Appendix C.5

3.2.4 Study Participant Selection

As discussed in detail later as part of subsection 3.2.5, in an effort to maintain the reliability of the data collected during the experiment, the tests were proctored by a human researcher. This requirement to have a test proctor directly limited the ability of the researchers to solicit test participants. Due to these time and resource constraints, study participants were solicited from the students and faculty of the Computer Science department at Allegheny College as a matter of feasibility.

In future work, it might be useful to consider distributing the study over the Internet where more data could be gathered from participants who might be located elsewhere. However, the benefit of widely distributing the test comes at a cost of reliability, as self-reporting, anonymous participants may not be guaranteed to follow the experimental procedure correctly.

3.2.5 Task Administration and Data Collection

After study participants were gathered, the survey and test were administered by the automated `SkriptIV` system. Participants completed their survey and testing under the supervision of a test administrator, who ensured that the participants did not cheat, followed the experimental procedure correctly, and was available to answer any questions a participant might have.

The `SkriptIV` system automatically tracked all relevant testing metrics throughout the test procedure. `SkriptIV` was responsible for timing the individual tasks during the test, as well as reporting complete participant responses by posting results to a remote database. See section 3.3 for a more detailed explanation of the `SkriptIV` system.

Data collection involved tracking study participants' survey responses as well as their recorded responses to the test section. However, aside from the responses to the demographics survey, all data was anonymous. No unique identifiers were created to match data with our individual participants.

3.2.6 Data Analysis

As mentioned in the previous section, upon completion each participant's responses were entered into a remote database. As explained more fully in section 3.3, the database in question was implemented with Apache CouchDB, which provided a simple interface for the `SkriptIV` system, but which was not immediately amenable to data analysis. To this end, a short script (whose source is available as Appendix B.3) was created to fetch all of the data from the database and transform it into a `CSV` file.

This `CSV` file, containing all of the data collected from the experiment, was then imported into Wolfram Mathematica^{TM}, which was used to prepare the figures found in chapter 4. In addition to creating the figures, charts and plots, Mathematica^{TM} was also used to compute the result of the Friedman–Rank statistical test on our data.

To determine statistical significance of our findings, based on our experimental design, we selected the Friedman-Rank statistical test. This test was chosen because we divide individual results by category of formatting shown to the user, but users

are shown multiple types of formatting. The Friedman-Rank test is appropriate to analyze this type of data, because it is both a *non-parametric* test, meaning that it allows us to assume that our data will not fit a normal distribution, and also it is a test for data from single participants who participated across multiple test attempts. That second distinction is important, because the fact that our individual participants responded for each of the different categories of code formatting means we cannot use similar statistical tests such as the Mann–Whitney U test.

3.3 The SkriptIV System

A major contribution of our work involves the creation and the implementation of an automated tool for performing our research and research like it, a tool we have named `SkriptIV`. This tool is capable of automatically administering the entire experiment for a study participant, except for filling the role of a human test proctor. The system electronically administers the required informed consent forms, provides directions to test participants, collects responses to the demographics survey, and automatically collects participant timing and response data for the testing portion of the experiment.

The complete source code for the system is given in Appendix C. We hope that by releasing the source code for the complete system and making it readily available, researchers who might attempt to review, reproduce or modify our study will find it relatively easy to do so.

3.3.1 System Architecture Overview

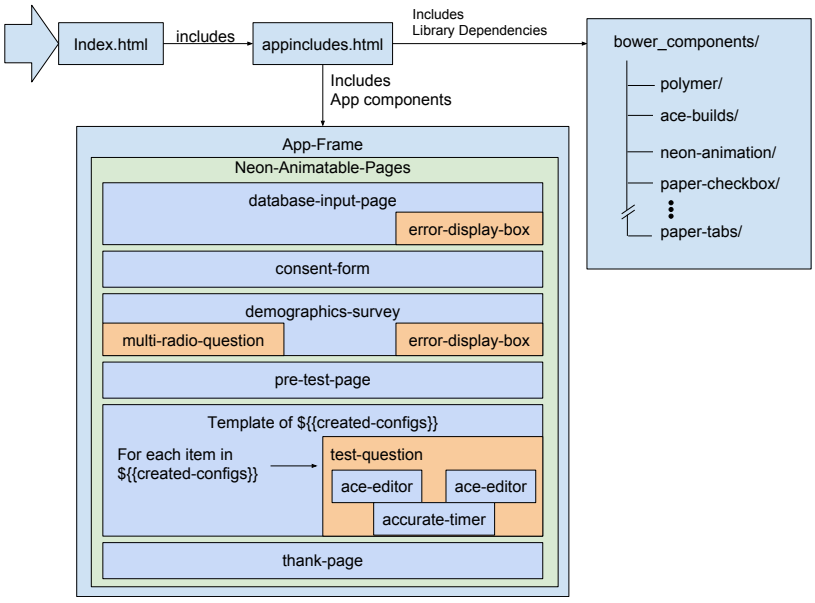

Figure 3.4: `SkriptIV` technical architecture overview

As we can see in Figure 3.4, the the software system is designed in the form

of several modular components, each contained in their own source file. Additionally, the tool leverages several open-source components and libraries in order to function correctly. These libraries include several open source projects, notably both the `Ace Editor` component, for displaying a high quality interface for viewing syntax-highlighted source code, and the `Polymer` open-source project for creating self-contained modules of HTML code called *Web Components.* A number of open-source Web Components published by the `Polymer` developers and community are also included as dependencies in order to provide a pleasant and well designed user experience, such as the `paper-button` and `paper-tabs` components.

3.3.2 Running the System

The process to build and run the `SkriptIV` system from source is relatively simple. First, you will require a working `Node.JS` and corresponding `npm` installation, as well as an installation of the `bower` command line tool. Then, to gather the required (but not provided) dependencies, one must run the commands found in Figure 3.5 in your terminal:

```
> npm install
> bower install
> npm install -g electron
```

Figure 3.5: Commands to install `SkriptIV` dependencies

These commands install the necessary library dependencies for `SkriptIV` from the NPM registry, as well as installing the `electron` command, which is the platform that allows `SkriptIV` to be run as a desktop application rather than a web page.

To start `SkriptIV` after correctly installing the dependencies, one can perform the command found in Figure 3.6. This command launches `SkriptIV` when run.

When launched, the `SkriptIV` system prompts the user to connect to the remote

```
> npm start
```

Figure 3.6: Command to launch `SkriptIV`

research database, then proceeds to administer both the survey and test. The program reads from a separate data directory containing task descriptions and source files, so the test can be configured by modifying the configuration files or by changing the contents of the included data files. Copies of the configuration files are presented as Appendix D. Copies of source data files originally used for this study are presented as Appendix E.

3.3.3 Remote Research Database

The `SkriptIV` system expects to connect to a remotely administered research database to display the participant consent document as well as to post experimental data for later review. For our implementation, we chose to set up a remote instance of `Apache CouchDB`, however, any database which implements the `CouchDB` protocol would be suitable for this purpose. The user credentials which the system uses to connect can be modified within the `database-input-page` source code; see Figure 3.4, the technical architecture diagram, or Figure 3.7, a screenshot of the relevant UI.

3.3.4 Test Administration

The most important component of the `SkriptIV` software system, however, was the actual test administration component. `SkriptIV` was capable of displaying fully syntax-highlighted source code within an environment which we could carefully control. For example, while the visual style of the testing component closely resembled a normal text editor, participants had no access to features such as find-in-file or the ability to automatically locate matching braces in programs. While we are aware

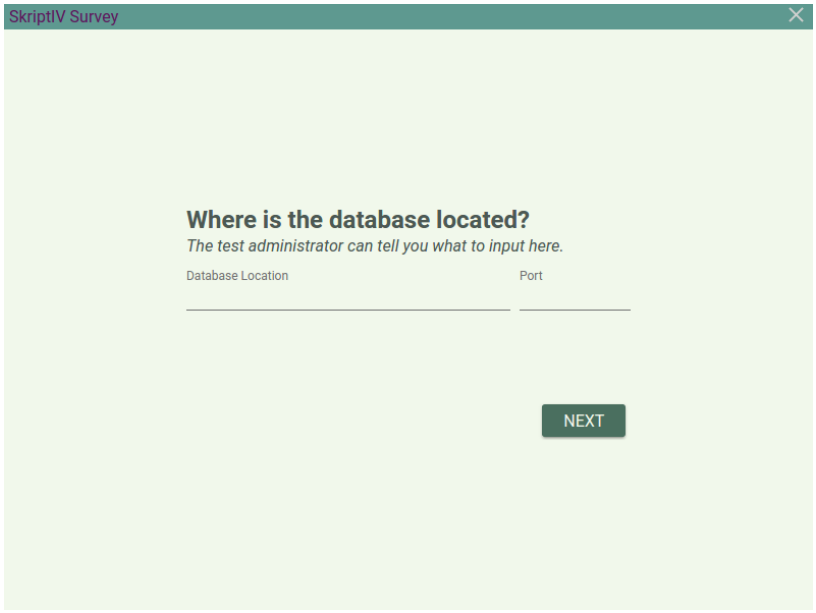

Figure 3.7: The database input page in `SkriptIV`

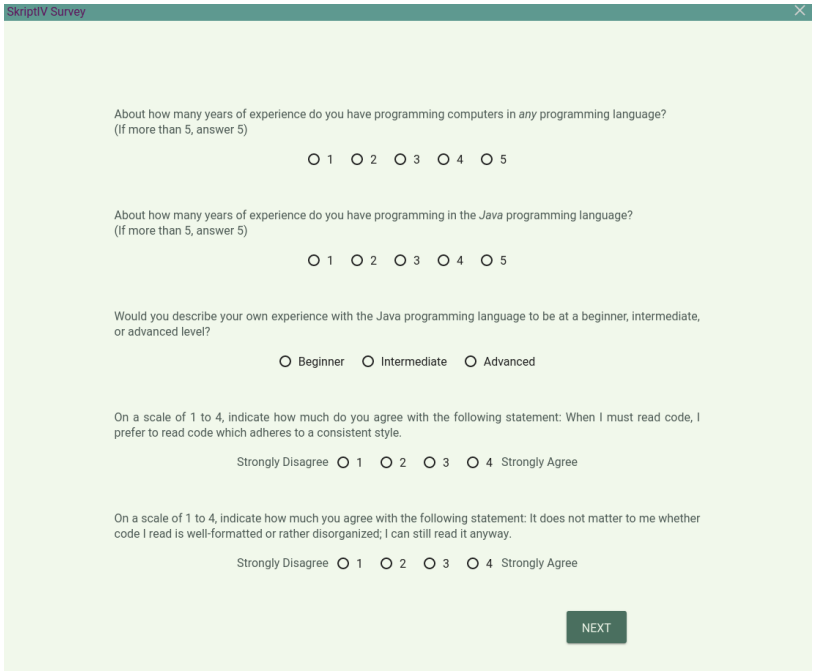

Figure 3.8: The survey administered as part of our experiment

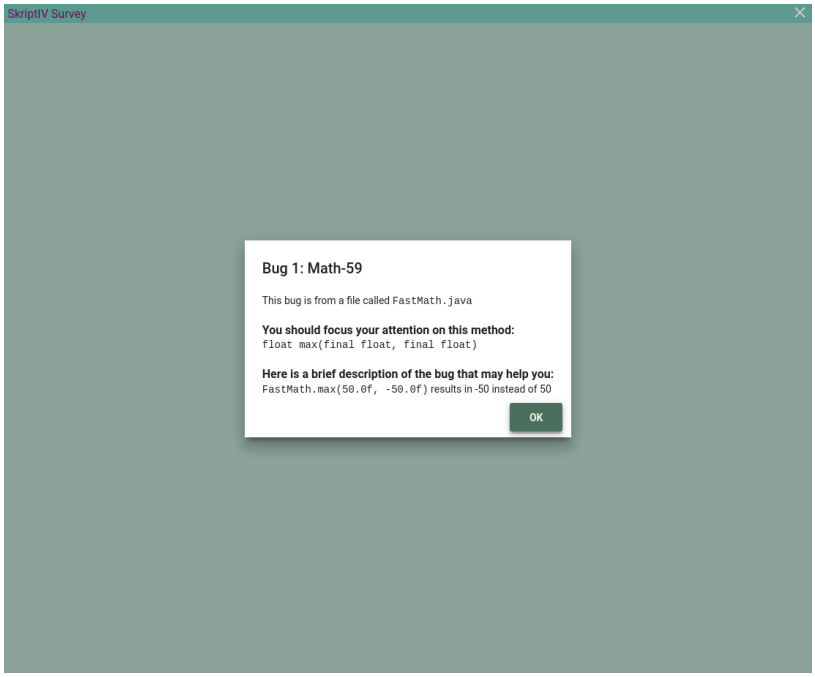

Figure 3.9: An example of the prompt that appears before a task is shown.

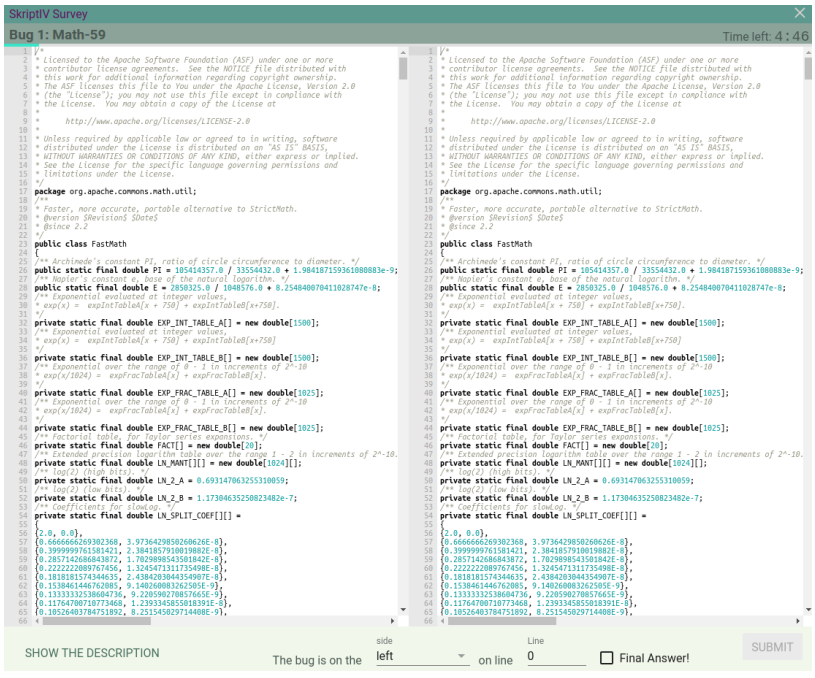

Figure 3.10: The testing interface of `SkriptIV`

that in a normal software-development environment, a programmer who wanted to use these features would have access to them, constraining the testing environment in this way for our experiment enabled us to require that timing data we received from participants was based solely on participants' abilities to read and navigate differently formatted source files.

The test administration system has been carefully architected to meet the exact requirements of our experimental methodology. For example, when randomizing task distribution, the implementation uses a cryptogoraphy module instead of the built-in JavaScript pseudo-random generator in an effort to eliminate perceived bias in the random generator. Random numbers from the cryptogoraphy module are strong enough to be used for secure key generation, and therefore should be strong enough for the experimental procedure. As another example, the procedure to post participant results back to the database is careful not to attach any unique identifiers to the dataset, and therefore there are no keys which may link a dataset to the participant who produced the data; even the IDs of the resulting documents stored in the database are randomly generated UUIDs instead of a sequential index based on when the documents were added.

3.4 Threats to Validity

Due to the nature of our study, there are a number of threats to validity that we have identified.

1. It should be noted that the number of volunteer study participants who successfully completed our experiment was very low. Only 12 volunteers participated at all, and of these, two participants submitted unusable data by not following experimental procedure correctly. These two participants merely clicked

through the entire test, resulting in default task responses where less than 10 seconds had passed, and the indicated faulty line of code was line 0, a line which does not exist. These responses were removed from the full dataset, leaving only 10 remaining data points.

2. During our analysis, we found that the correctness of participant responses was much lower than expected — less than 50% of responses were correct — while the number of responses which timed–out was higher than expected, at ∼25%. As a result, we suspect that the time limit of 5 minutes per task was perhaps too short. It is possible that this short time constraint caused some participants to guess as time ran out, skewing our results.

3. It should be noted that while best efforts were made to isolate variables during the experiment, the nature of the experiment requires that some outside influences may interfere with the results. For example, the methodology that we chose for this experiment involved presenting participants with the two formatted source code files side-by-side. Because of the errors that we chose and the way that code was presented to participants, it is quite possible that at least some of the participants merely attempted to scan through the files to search for differences. To at least partially mitigate this measurement error, we ensured that some files were very long, thereby requiring more reading, as well as to ask participants for a specific line number of the code which triggers the bug. In order to answer our comprehension question, test participants would still be required to read and comprehend the code they are presented with, even if they were able to isolate the bug location by a mere side-by-side comparison of the two files. Discussed in more detail in chapter 5, we believe that future work may be greatly hindered by this and similar effects while attempting to isolate code formatting as a variable in a study.

Chapter 4

Experimental Results

4.1 Summary

After performing an analysis of our experimental data, we have concluded that our (admittedly, anecdotal) evidence suggests that there is no correlation between code formatting and human debugging ability, successfully refuting both hypotheses **RH.1** and **RH.2** as described earlier in chapter 1. The complete experimental dataset is disclosed as Appendix A, while the Wolfram Language^{TM} code used to prepare figures and perform the analysis of experimental results is presented as Appendix B.1.

4.2 Detail

We began our analysis by separating participant responses by task configuration only. As we can see by the box-and whisker plot in Figure 4.2 showing participant response time variance, there is very little discernible difference in response time of participants based on the particular code formatting configuration that was presented. Similarly, in Figure 4.3 there is very little variation in the number of correct responses to tasks reported based only on configuration.

To determine statistical significance of our findings, based on our experimental

design, we selected the Friedman-Rank statistical test. This test was chosen because we divide individual results by category of formatting shown to the user, but users are shown multiple types of formatting. The Friedman-Rank test is appropriate to analyze this type of data, because it is both a *non-parametric* test, meaning that it allows us to assume that our data will not fit a normal distribution, and also it is a test for data from single participants who participated across multiple test attempts. That second distinction is important, because the fact that our individual participants responded for each of the different categories of code formatting means we cannot use similar statistical tests such as the Mann–Whitney U test. The result of the Friedman-Rank statistical test applied to the same data from Figure 4.2 confirms our original intuition about the variation of the data with a very high P-Value.

	Statistic	P-Value
Friedman Rank	0.0902919	0.91386

Figure 4.1: Result of the Friedman-Rank test applied to participant response time

The Friedman-Rank statistical test is a measure which attempts to determine with statistical significance whether or not the means of sets of data are distinct. The test does not determine *which* set may have a distinct mean, only that one of them may be different. In order to conclude that any set of data is distinct, we must be able to find a very low P-Value, typically $p < 0.05$. Here, we find that the P-Value is very large, and so we cannot conclude with confidence that there is any significant variation in the means of typical response times of task participants when the data is categorized only by task formatting configuration.

Based on our data and analysis, then, we can immediately reject the hypothesis **RH.1**, as we could not determine with statistical significance that participants who were exposed to code formatting configuration C spent a longer amount of time performing debugging tasks when compared to the other formats.

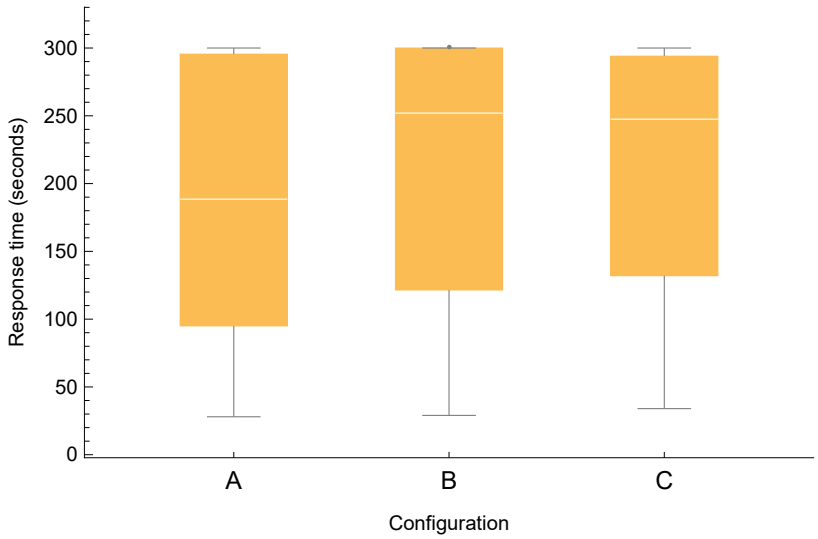

Figure 4.2: Participant response time by formatting configuration

Similarly, we can immediately reject the hypothesis **RH.2** based on the same result, as we could not determine with statistical significance that there was any difference in participant speed when comparing only configurations A or B.

Although we had originally intended to also determine significance of results across smaller categories such as for a particular task or within a stated demographic of participant, due to the incredibly small sample size of our data we are unable to draw any useful, generalizable conclusions based on any smaller subdivisions of our data; conclusions drawn about the data categorized only by task configuration are already based upon a very small sample size.

As mentioned previously in section 3.4, because the sample size of our study was very small, these results may yet have very little significance. It is important to keep in mind that that the rejection of **RH.1** and **RH.2** may be incorrect as a result of small sample size.

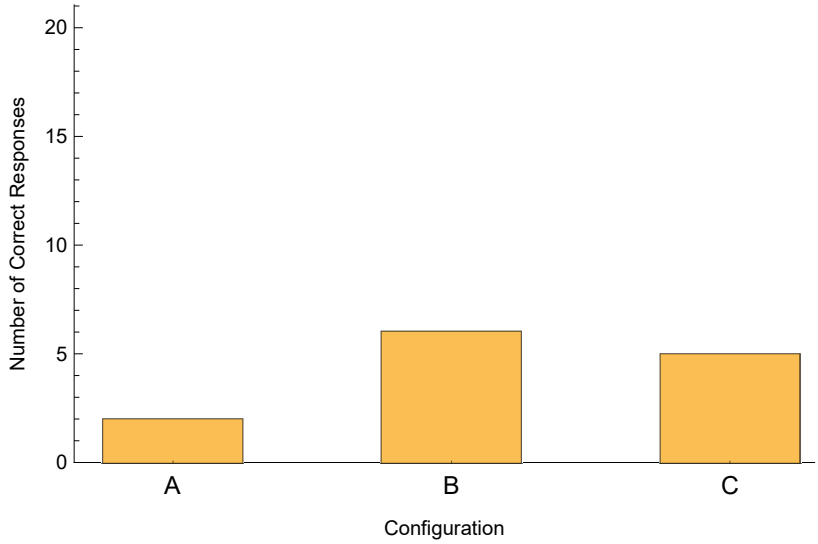

Figure 4.3: Participant correct responses by formatting configuration

Chapter 5

Discussion and Future Work

5.1 Summary of Results

In chapter 1, we set forth two research questions, **RQ.1** and **RQ.2**, and two relevant hypotheses concerning those questions, **RH.1** and **RH.2**. These were:

RQ.1 We seek to verify that regular code formatting impacts performance of debugging tasks by identifying a statistically significant decrease in both speed and accuracy of participants when performing our debugging tasks when an irregular 'control' format is introduced.

RQ.2 After the finding of RQ.1, we seek to identify any significant increase in speed and accuracy of participants when viewing one of two separate formatting variants based on popular coding standards. Reference section 3.1 for a comprehensive discussion of the format variants.

RH.1 We will find that participants perform significantly slower when presented with the irregular 'control' code format as compared to the other two formats.

RH.2 We will find that participants tend to favor one of the code formats based on popular coding standards, performing slightly faster and with improved correctness when presented with this format as compared to the other two formats.

To test these hypotheses and answer the research questions, we presented a detailed research method in chapter 3, and presented `SkriptIV`, an automated tool that implements much of our experimental procedure. This tool was then run to collect our experimental data, presented in full as Appendix A and analyzed in chapter 4.

As the result of our analysis, we have been able to reject both initial hypotheses; however, as discussed previously in section 3.4, "Threats to Validity", our dataset was relatively small. Because of this, our assertion cannot be viewed as based on anything more than anecdotal evidence, however, if our findings are replicated and confirmed with a larger sample size, based on this evidence we can believe that research would find there is be little or no effect that code formatting has on human debugging performance.

Despite our difficulties with experiment sample size, however, we believe that we have presented a sound experimental methodology for the evaluation of code readability and formatting metrics in a controlled setting. Our `SkriptIV` software system for automatic test generation we feel is an important contribution to this area of research, enabling our study to be completely replicable with minimal effort, while also providing a flexible platform for extending and modifying the experimental procedure to study types which may require evaluating human performance of advanced coding tasks.

5.2 Future Work

Based on our findings, we have identified a number of areas of future work for research in this area:

1. A larger replication study to verify our findings among a larger population of participants would be a simple extension to our work. While we strongly be-

lieve that we have developed a useful fundamental platform for doing this type of research comparing differing coding formatting styles, including the development and implementation of the automated `SkriptIV` test-administration tool, the size of the study that we were able to conduct leaves much to be desired. Because the entire `SkriptIV` tool has been released alongside this paper, replication of our study would be a relatively simple affair as the `SkriptIV` tool automates nearly all of our stated research method.

2. Although we were unable to find that code formatting significantly impacted human debugging performance, this still leaves the question of whether code formatting affects any other human task performance. Our claim that code formatting has little or no effect on human debugging performance is a relatively limited one, and research into other types of tasks or other types of formatting variables that we did not study may yet find a useful result.

5.3 Conclusion

Although our research may be inconclusive due to small sample size, if confirmed, we believe that this work may be applicable rather broadly to the field of Computer Science.

First, the result may demonstrate that although some programmers may have an preference for certain formatting features such as line-length, vertical whitespace, or the particular layout of the placement of brackets, these individual preferences have limited utility when actually performing one of the major activities of programming, namely isolating and identifying a software fault in code. Therefore, this finding is a direct example of the fact that code attractiveness does not inherently equate to "better" code, in the sense that "better" code easier to work with or to maintain.

Second, it may be useful in the field of programming education to note that code which is formatted incorrectly may not necessarily be more difficult to debug in the context of an exam or laboratory setting. Our finding instead directly suggests that students should be just as effective when presented with haphazardly formatted code as they would be when presented with regular formatting of source code.

However, based on the limited size of our study, as well as the narrow nature of the claim that we are able to make, we strongly believe that more research in this area is required. Whether or not code formatting or style has an effect on programming effectiveness, it should be equally important to refute as well as to support such hypotheses.

Despite this prominent setback, we believe that the presentation and subsequent release of our `SkriptIV` software system may help to advance this important area of research in the future by providing a platform for reproducible research focusing on human interaction with written code. As a result, we believe that our contribution will enable other researchers to more easily craft research studies evaluating human performance when controlling for different aspects of written programming code.

Bibliography

[1] Simon Butler. The effect of identifier naming on source code readability and quality. In *Proceedings of the doctoral symposium for ESEC/FSE on Doctoral symposium*, 2009.

[2] Ermira Daka, José Campos, Gordon Fraser, Jonathan Dorn, and Westley Weimer. Modeling readability to improve unit tests. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015.

[3] Google. styleguide. `https://github.com/google/styleguide`, June 2016.

[4] Prateek Hejmady and N. Hari Narayanan. Visual attention patterns during program debugging with an IDE. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, 2012.

[5] René Just, Darioush Jalali, and Michael D. Ernst. Defects4j: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, 2014.

[6] Brian W. Kernighan and P. J. Plauger. *The Elements of Programming Style*. McGraw-Hill, Inc., New York, NY, USA, 2nd edition, 1982.

[7] John Lewis and William Loftus. *Java software solutions : Foundations of program design*. Pearson, 6 edition, 2009.

[8] M. Metcalf. Fortran 77 coding conventions. *SIGPLAN Fortran Forum*, 2(4):10–15, December 1983.

[9] Richard J. Miara, Joyce A. Musselman, Juan A. Navarro, and Ben Shneiderman. Program indentation and comprehensibility. *Commun. ACM*, 26(11):861–867, November 1983.

[10] Microsoft. C# coding conventions (c# programming guide). `https://msdn.microsoft.com/en-us/library/ff926074.aspx`, July 2015.

[11] Peter R. Newsted, Wing-Keen Leong, and Joanna Yeung. The impact of programming styles on debugging efficiency. *SIGSOFT Software Engineering Notes*, 6(5):14–18, October 1981.

[12] P. W. Oman and C. R. Cook. A paradigm for programming style research. *SIGPLAN Notices*, 23(12):69–78, December 1988.

[13] Daryl Posnett, Abram HIndle, and Premkumar Devanbu. A simpler model of software readability. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 73–82, Waikiki, Honolulu, Hawaii, 2011. ACM.

[14] Brono F.L. Santos and Hendrik T. Macedo. Improving $CUDA^{TM}$ C/C++ encoding readability to foster parallel application development. *SIGSOFT Software Engineering Notes*, 37(1):1 – 5, 2012.

[15] Sina Shamshiri, Rene Just, Jose Miguel Rojas, Gordon Fraser, Phil McMinn, and Andrea Arcuri. Do automatically generated unit tests find real faults? an empirical study of effectiveness and challenges (t). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 201–211, Washington, DC, USA, 2015. IEEE Computer Society.

[16] B. A. Sheil. The psychological study of programming. *ACM Computing Surveys*, 13(1):101–120, March 1981.

[17] Sun. Code conventions for the Java(TM) programming language. `http://www.oracle.com/technetwork/java/codeconvtoc-136057.html`, April 1999.

[18] Maurice V. Wilkes, David J. Wheeler, and Stanley Gill. *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley Publishing Company, Inc., 1951, 1957.

Appendix A

Complete Experimental Dataset

participant	consented	consentVersion	dem1	dem2	dem3	dem4	dem5	task	config	flipped	timeRemain	side	line	result
0	1	6	5	3	Advanced	4	1	Math-59	C	false	0:00		-1	Timed-Out
0	1	6	5	3	Advanced	4	1	Lang-1	C	true	0:00		-1	Timed-Out
0	1	6	5	3	Advanced	4	1	Chart-5	B	false	0:00		-1	Timed-Out
0	1	6	5	3	Advanced	4	1	Chart-8	C	true	2:31	left	0	Incorrect
0	1	6	5	3	Advanced	4	1	Time-15	A	false	4:32	left	129	Incorrect
0	1	6	5	3	Advanced	4	1	Math-26	B	false	4:31	left	120	Incorrect
0	1	6	5	3	Advanced	4	1	Mockito-18	A	true	4:07	left	57	Incorrect
1	1	6	2	2	Intermediate	3	3	Math-59	C	false	0:00		-1	Timed-Out
1	1	6	2	2	Intermediate	3	3	Lang-1	A	true	0:04	left	661	Incorrect
1	1	6	2	2	Intermediate	3	3	Chart-8	B	true	0:00		-1	Timed-Out
1	1	6	2	2	Intermediate	3	3	Math-26	C	false	0:31	right	160	Incorrect
1	1	6	2	2	Intermediate	3	3	Chart-5	B	true	1:50	left	600	Incorrect
1	1	6	2	2	Intermediate	3	3	Time-15	C	false	3:57	left	141	Correct
1	1	6	2	2	Intermediate	3	3	Mockito-18	A	false	3:23	right	99	Incorrect
2	1	6	4	2	Beginner	4	1	Math-59	C	false	0:00		-1	Timed-Out
2	1	6	4	2	Beginner	4	1	Time-15	C	true	0:44	right	148	Incorrect
2	1	6	4	2	Beginner	4	1	Chart-8	B	true	2:24	right	183	Correct
2	1	6	4	2	Beginner	4	1	Math-26	A	false	0:05	left	193	Correct
2	1	6	4	2	Beginner	4	1	Lang-1	B	false	0:00		-1	Timed-Out
2	1	6	4	2	Beginner	4	1	Mockito-18	A	true	2:23	left	100	Incorrect
2	1	6	4	2	Beginner	4	1	Chart-5	C	false	0:00	left	540	Incorrect
3	1	6	3	2	Intermediate	4	3	Math-59	C	false	0:00		-1	Timed-Out
3	1	6	3	2	Intermediate	4	3	Chart-5	B	true	0:00		-1	Timed-Out
3	1	6	3	2	Intermediate	4	3	Time-15	C	false	0:00		-1	Timed-Out
3	1	6	3	2	Intermediate	4	3	Mockito-18	A	true	3:27	left	93	Incorrect
3	1	6	3	2	Intermediate	4	3	Lang-1	C	false	0:00		-1	Timed-Out
3	1	6	3	2	Intermediate	4	3	Chart-8	B	false	0:00		-1	Timed-Out
3	1	6	3	2	Intermediate	4	3	Math-26	A	false	3:29	left	483	Incorrect
4	1	6	3	3	Intermediate	3	2	Math-59	C	false	0:49	left	3013	Correct
4	1	6	3	3	Intermediate	3	2	Chart-5	A	true	0:22	right	568	Incorrect
4	1	6	3	3	Intermediate	3	2	Chart-8	C	false	2:54	right	175	Incorrect
4	1	6	3	3	Intermediate	3	2	Mockito-18	B	false	0:22	left	104	Incorrect
4	1	6	3	3	Intermediate	3	2	Time-15	C	false	3:21	left	141	Correct
4	1	6	3	3	Intermediate	3	2	Math-26	A	true	0:00		-1	Timed-Out
4	1	6	3	3	Intermediate	3	2	Lang-1	B	false	0:22	right	512	Incorrect
5	1	6	4	3	Intermediate	3	3	Math-59	C	false	1:14	left	3013	Correct
5	1	6	4	3	Intermediate	3	3	Time-15	A	true	1:24	left	151	Incorrect
5	1	6	4	3	Intermediate	3	3	Chart-8	C	false	2:42	right	175	Incorrect
5	1	6	4	3	Intermediate	3	3	Lang-1	A	false	0:33	left	514	Correct
5	1	6	4	3	Intermediate	3	3	Chart-5	B	true	1:14	right	595	Correct
5	1	6	4	3	Intermediate	3	3	Mockito-18	C	true	1:46	right	83	Incorrect
5	1	6	4	3	Intermediate	3	3	Math-26	B	true	3:04	right	123	Incorrect
6	1	6	3	3	Advanced	4	1	Math-59	C	false	0:00		-1	Timed-Out
6	1	6	3	3	Advanced	4	1	Math-26	A	true	0:00		-1	Timed-Out
6	1	6	3	3	Advanced	4	1	Lang-1	A	false	0:00		-1	Timed-Out
6	1	6	3	3	Advanced	4	1	Mockito-18	B	false	0:17	left	0100	Incorrect
6	1	6	3	3	Advanced	4	1	Chart-8	C	true	1:01	right	175	Correct
6	1	6	3	3	Advanced	4	1	Chart-5	C	false	0:14	left	542	Incorrect
6	1	6	3	3	Advanced	4	1	Time-15	B	true	3:16	right	152	Correct
7	1	6	4	4	Intermediate	4	2	Math-59	C	false	0:20	left	3013	Correct
7	1	6	4	4	Intermediate	4	2	Chart-5	C	false	0:00		-1	Timed-Out
7	1	6	4	4	Intermediate	4	2	Math-26	C	false	1:02	left	186	Correct
7	1	6	4	4	Intermediate	4	2	Lang-1	A	true	0:00		-1	Timed-Out
7	1	6	4	4	Intermediate	4	2	Mockito-18	A	false	1:53	right	100	Incorrect
7	1	6	4	4	Intermediate	4	2	Chart-8	B	true	0:18	right	183	Correct
7	1	6	4	4	Intermediate	4	2	Time-15	B	false	4:20	left	152	Correct
8	1	6	5	4	Intermediate	4	1	Math-59	C	false	1:45	left	3013	Correct
8	1	6	5	4	Intermediate	4	1	Math-26	B	false	0:00		-1	Timed-Out
8	1	6	5	4	Intermediate	4	1	Chart-8	A	false	1:50	left	191	Incorrect
8	1	6	5	4	Intermediate	4	1	Mockito-18	A	false	3:19	right	100	Incorrect

participant	consented	consentVersion	dem1	dem2	dem3	dem4	dem5	task	config	flipped	timeRemain	side	line	result
8	1	6	5	4	Intermediate	4	1	Time-15	C	false	4:26	left	141	Correct
8	1	6	5	4	Intermediate	4	1	Chart-5	B	false	1:47	left	597	Correct
8	1	6	5	4	Intermediate	4	1	Lang-1	C	false	0:35	right	483	Incorrect
9	1	6	1	1	Beginner	1	4	Math-59	C	false	3:07	right	168	Incorrect
9	1	6	1	1	Beginner	1	4	Chart-8	A	true	2:01	left	560	Incorrect
9	1	6	1	1	Beginner	1	4	Mockito-18	C	false	3:42	right	88	Incorrect
9	1	6	1	1	Beginner	1	4	Math-26	B	true	3:30	left	100	Incorrect
9	1	6	1	1	Beginner	1	4	Chart-5	C	false	0:12	left	44	Incorrect
9	1	6	1	1	Beginner	1	4	Time-15	A	false	3:48	left	154	Incorrect
9	1	6	1	1	Beginner	1	4	Lang-1	B	false	2:53	right	516	Incorrect

Appendix B

Miscellaneous Source Files

B.1 Mathematica^{TM} Data Analysis Workbook

```
1  In[1]:= SetDirectory[NotebookDirectory[]];
2  rawdata = Import["data-out.csv"];
3  headers=rawdata[[1]];
4  Assoc=Function[x,(#[[1]]->#[[2]])&/@Partition[x,2]];
5  data = Dataset[Association/@((Assoc[Riffle[headers,#]])&/
       @rawdata[[2;;]])] ;
6  data=Function[x, Association[Flatten[{KeyDrop[x, "
       timeRemain"]//Normal,{"timeRemain"->TimeObject[Flatten
       [{0,Interpreter[Number]/@StringSplit[x["timeRemain
       "],":"]}]] - TimeObject[{0,0,0}], "timeUsed"-> (
       TimeObject[{0,5,0}]-TimeObject[Flatten[{0,Interpreter[
       Number]/@StringSplit[x["timeRemain"],":"]}]])}]]]/@data
7  In[382]:= yy = KeyTake[#, {"config", "task","timeUsed", "
       result"}]&/@Select[data, #["task"] != "Math-59"&];
8  byConfig= Function[x,Select[yy, #["config"]==x&]]/@{"A","B
       ","C"};
9  timeByConfig = Normal[#["timeUsed"]&/@#]&/@byConfig;
10 correctnessByConfig = Normal[If[#["result"]=="Correct
       ",1,0]&/@#]&/@byConfig;
11 timeByConfigAndTaskRaws = (GroupBy[#, (#["task"])&])&/@ (
       KeyTake[#, {"task","timeUsed"}]&/@byConfig);
12 timeByConfigAndTask = KeySort[Map[Function[item,UnitConvert
       [item["timeUsed"],"seconds"]],#,{2}]]&/
       @timeByConfigAndTaskRaws
13 correctnessByConfigAndTaskRaws=(GroupBy[#, (#["task"])&])&/
       @Function[item, Map[(Association[{"task"-> #["task"],"
       result"-> If[#["result"]=="Correct", 1, 0]}])&,item]]/
       @byConfig;
14 correctnessByConfigAndTask = KeySort[Map[Function[item,item
       ["result"]],#,{2}]]&/@correctnessByConfigAndTaskRaws
```

```
15 In[402]:= Export["correctness-by-config-and-task-A.eps",
      correctnessByConfigAndTask[[1]]](*Correct responses by
      Task for Configuration A*)
16 Export["correctness-by-config-and-task-B.eps",
      correctnessByConfigAndTask[[2]]] (*"Correct responses by
       Task for Configuration B*)
17 Export["correctness-by-config-and-task-C.eps",
      correctnessByConfigAndTask[[3]]](* "Correct responses by
       Task for Configuration C*)
18 Export["time-by-config-and-task-A.eps",timeByConfigAndTask
      [[1]]] (*Time to response by Task for Configuration A*)
19 Export["time-by-config-and-task-B.eps",timeByConfigAndTask
      [[2]]] (*Time to response by Task for Configuration B*)
20 Export["time-by-config-and-task-C.eps",timeByConfigAndTask
      [[3]]] (*Time to response by Task for Configuration C*)
21 Export["boxwhisker-time-by-config.eps",BoxWhiskerChart[
      timeByConfig,{Frame -> {{True, False}, {True, False}},
      FrameLabel-> {"Configuration","Response time (seconds)
      "},ChartLabels->Placed[{"A", "B","C"},Axis,Style[#,11,
      Black]&]}]] (*Time to response by Configuration, in
      Seconds*)
22 Export["barchart-correctness-by-config.eps",BarChart[Map[
      Total,correctnessByConfig],BarSpacing->1,Ticks
      ->{{0,1,2,3,4,5,6},{}},Frame-> {{True, False},{True,
      False}},FrameLabel-> {"Configuration","Number of Correct
       Responses"},ChartLabels->Placed[{"A", "B","C"},Axis,
      Style[#,11,Black]&],PlotRange-> {0,Length[
      correctnessByConfig[[1]]]} ]] (*Correct responses amount
       by configuration*)
23 Export["stat-test-time-by-config.eps",
      LocationEquivalenceTest[timeByConfig, {"TestDataTable","
      FriedmanRank"}]]
```

B.2 defects-compactify.pl

The script used to create the control source code formatting type "C". It strips whitespace from the file input on the command line, and on certain lines, and on lines ending with curly-braces, will move ending braces to the next line if the line is even-numbered.

```
#!/usr/bin/env perl
$i = 0;
while(<>) {
    chop;
    if($_ ne "") {
        s/^\s+//;
        $a = $_;
        if($a =~ m/([^{]*){$/ && $i % 2 == 0 ) {
            print $1 . "\n{\n";
        } else {
            print $a . "\n";
        }
         $i = $i + 1;
    }
}
```

B.3 reportbuilder.js

```
1  "use strict";
2
3  const cradle = require('cradle')
4  const fs = require('fs')
5
6  function primitiveCollect(howMany, callback) {
7      let sharedObject = {results: []}
8
9      return (function(result) {
10         console.log("Debug: got response " + howMany)
11         howMany--;
12         if(result) {
13             this.results.push(result);
14         } else {
15             console.log("Warn: threw away response " +
                   howMany)
16         }
17         if(howMany == 0) {
18             callback(this.results);
19         }
20     }).bind(sharedObject);
21 }
22
23
24 var db;
25 try {
26     db = new(cradle.Connection)(process.argv[2], process.
          argv[3], {
27          auth: {
28              username: 'researcher',
29              password: '12345'
30          }
31     }).database('research')
32 } catch(e) {
33     console.log("Error!")
34     console.log(e);
35     process.exit(1);
36 }
37
38 db.all(function(err, reply) {
39     if(err) {
40         console.log(err)
41     } else {
42         let responder = primitiveCollect(reply.length,
```

```
            output)
43
44          for(var i = 0; i < reply.length; i++) {
45              db.get(reply[i].id, function(err, doc) {
46                  if(err) {
47                      console.log(err)
48                  } else {
49                      if(doc.consent) {
50                          responder(doc)
51                      } else {
52                          responder(null)
53                      }
54                  }
55              })
56          }
57      }
58  })
59
60  function output(x) {
61      console.log("%%%%%%%%%% BEGIN CSV %%%%%%%%%%%");
62      let header = 'participant,consented,consentVersion,dem1
            ,dem2,dem3,dem4,dem5,task,config,flipped,timeRemain,
            side,line,result'
63      console.log('participant,consented,consentVersion,dem1,
            dem2,dem3,dem4,dem5,task,config,flipped,timeRemain,
            side,line,result')
64
65      let results = [header]
66      for(var i = 0; i < x.length; i++) {
67          let temp = csvify(x[i],i);
68
69          for(var z = 0; z < temp.length; z++) {
70              results.push(temp[z])
71          }
72      }
73
74      let outString = "";
75      for(var i = 0; i < results.length; i++) {
76          outString = outString + results[i] + "\n"
77      }
78
79      fs.writeFileSync("./data-out.csv", outString, {
80          encoding: "utf-8"
81      })
82  }
83
```

```
84 function csvify(object, index) {
85     let rows = []
86     for(var i = 0; i < object.questionResults.length; i++)
          {
87         let c = object.consent;
88         let d = object.demographicsResults;
89         let q = object.questionResults[i];
90         let lv = `${index},${c.consented},${c.
              consentVersion},${d.q1},${d.q2},${d.q3},${d.q4},
              ${d.q5},${q.task},${q.config.bugVersion},${q.
              config.flipped},${q.timeRemaining},${q.
              rawResponse.side},${q.rawResponse.line},${q.
              result}`;
91         console.log(lv)
92         rows.push(lv)
93     }
94     return rows;
95 }
```

Appendix C

SkriptIV

The complete source of the test administration software, `SkriptIV`. Be careful of the difference between single-quote, double-quote, and backtick characters in the source, as these differences are potentially significant in JavaScript and HTML, but are not made particularly clear when reaing print.

`SkriptIV` has been released free of charge under the MIT license. See section C.1

C.1 Skriptiv License

```
1  .NOTICE.
2
3  Skriptiv itself is licensed under the MIT License, which is
4  reproduced below. Skriptiv may make use of a number
5  of open source components which may be required to build or
6  use Skriptiv. These components have their own license terms
       ,
7  and are not included directly in this source repository.
8
9  Using the build script to produce a working binary of
       Scriptiv
10 may install these components, and thus the resulting binary
        may
11 or may not be compatible with the permissions granted by
       the MIT
12 license, which is applicable only to the original **source
       code**
13 contained here.
14
15 -----------------
16  The MIT License
17 -----------------
18
```

```
19 Copyright (c) 2016 Matthew Clive
      <arcticlight@arcticlight.me>
20
21 Permission is hereby granted, free of charge, to any person
       obtaining a copy
22 of this software and associated documentation files (the "
      Software"), to deal
23 in the Software without restriction, including without
      limitation the rights
24 to use, copy, modify, merge, publish, distribute,
      sublicense, and/or sell
25 copies of the Software, and to permit persons to whom the
      Software is
26 furnished to do so, subject to the following conditions:
27
28 The above copyright notice and this permission notice shall
       be included in all
29 copies or substantial portions of the Software.
30
31 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
      KIND, EXPRESS OR
32 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
      MERCHANTABILITY,
33 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
       EVENT SHALL THE
34 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
      DAMAGES OR OTHER
35 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
      OTHERWISE, ARISING FROM,
36 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
      OTHER DEALINGS IN THE
37 SOFTWARE.
```

C.2 package.json

```
1  {
2    "name": "skriptiv",
3    "version": "0.1.0",
4    "description": "",
5    "main": "main.js",
6    "scripts": {
7      "start": "electron ."
8    },
9    "dependencies": {
10     "bower": "^1.7.9",
11     "cradle": "^0.7.1",
12     "electron": "^1.4.1"
13   }
14 }
```

C.3 bower.json

```
{
  "name": "skriptiv",
  "private": true,
  "dependencies": {
    "iron-icons": "polymerelements/iron-icons#^1.0.0",
    "iron-iconset-svg": "polymerelements/iron-iconset-svg
      #^1.0.0",
    "paper-input": "PolymerElements/paper-input#^1.0.0",
    "paper-button": "PolymerElements/paper-button#^1.0.0",
    "paper-dropdown-menu": "PolymerElements/paper-dropdown-
      menu#^1.4.2",
    "neon-animation": "PolymerElements/neon-animation
      #^1.0.0",
    "paper-checkbox": "PolymerElements/paper-checkbox
      #^1.0.0",
    "paper-radio-group": "PolymerElements/paper-radio-group
      #^1.2.1",
    "ace-builds": "^1.2.5",
    "iron-ajax": "PolymerElements/iron-ajax#^1.4.3",
    "paper-progress": "PolymerElements/paper-progress
      #^1.0.11",
    "paper-dialog": "PolymerElements/paper-dialog#^1.1.0",
    "paper-dialog-scrollable": "PolymerElements/paper-
      dialog-scrollable#^1.1.5",
    "paper-tabs": "PolymerElements/paper-tabs#^1.7.0"
  }
}
```

C.4 main.js

```
1  //require the things
2  const integrityChecker = require(`${__dirname}/
       integrityChecker.js`)
3  const {app, BrowserWindow, dialog} = require('electron')
4
5  //Begin by verifying that the required datafiles are all
       present and accounted for.
6  if(!integrityChecker.check()) {
7      dialog.showErrorBox("FATAL ERROR", "Some required data
           files failed to verify!")
8      app.exit(1)
9  }
10
11 global.tasks = require(`${__dirname}/../Defects-Sources/
       tasks.json`)
12
13 var mainWindow = null
14
15 app.on('ready', () => {
16     mainWindow = new BrowserWindow({
17         height: 600,
18         frame: false,
19         'min-width': 950
20     })
21
22     mainWindow.on('closed', () => {
23         mainWindow = null
24     })
25     //mainWindow.webContents.openDevTools()
26     mainWindow.loadURL(`file://${__dirname}/index.html`)
27
28 })
29
30 app.on('window-all-closed', () => {
31     app.quit()
32 })
```

C.5 integrityChecker.js

```
1  let fs = require('fs')
2  let pathroot = `${__dirname}/../Defects-Sources`
3
4  function log(s) {
5      console.log("[CHECKER] " + s)
6  }
7
8  function getInfoFor(bug) {
9      let info = require(`${pathroot}/${bug}/info.json`)
10
11     if (!info
12      || !info.bugTitle
13      || !info.bugLocation
14      || !info.bugDescription
15      || !info.fileName) {
16          throw("The info object for this task was
                incomplete!")
17      }
18
19     return info;
20 }
21
22 function checkResponses(bug, config) {
23     return !(!bug
24             || !bug.correctResponses
25             || !bug.correctResponses[config]
26             || !bug.correctResponses[config].length
27             || bug.correctResponses[config].length == 0)
28 }
29
30 function checkConfigExists(bug, conf) {
31     let a = fs.existsSync(`${pathroot}/${bug.bugTitle}/${
           bug.fileName}-bugged-${conf}.java`)
32     let b = fs.existsSync(`${pathroot}/${bug.bugTitle}/${
           bug.fileName}-fixed-${conf}.java`)
33     return (a && b)
34 }
35
36 function checkLicenseExists(bug) {
37     return fs.existsSync(`${pathroot}/${bug.bugTitle}/
           LICENSE.txt`)
38 }
39
40 function checkConfigBWhitespace(bug) {
```

```
41      if(!checkConfigExists(bug, "B")) {
42          throw("Config B does not exist for this file!")
43      }
44
45      let sv = function(bug, bv) {
46          let file = fs.readFileSync(`${pathroot}/${bug.
               bugTitle}/${bug.fileName}-${bv}-B.java`, 'utf-8'
               )
47              .split("\n")
48
49          let previousLineWasBlank = true
50
51          for(let i = 0; i < file.length; i++) {
52              if(file[i].length > 100) {
53                  throw(`${bug.bugTitle} conf ${bv}-B has a
                       >100 column long line at ${i}!`)
54              }
55
56              let thisLine = file[i].trim()
57              let thisLineIsBlank = (thisLine === "")
58
59              if(thisLineIsBlank && previousLineWasBlank) {
60                  throw(`${bug.bugTitle} conf ${bv}-B has two
                           consecutive blank lines at ${i}!`)
61              }
62
63              previousLineWasBlank = thisLineIsBlank
64          }
65      }
66
67      return sv(bug, "bugged") && sv(bug, "fixed")
68  }
69
70  function checkConfigAWhitespace(bug) {
71      if(!checkConfigExists(bug, "A")) {
72          throw("Config A does not exist for this file!")
73      }
74
75      let sv = function(bug, bv) {
76          let file = fs.readFileSync(`${pathroot}/${bug.
               bugTitle}/${bug.fileName}-${bv}-A.java`, 'utf-8'
               )
77              .split("\n")
78
79          let previousLineWasBlank = true
80          let twoLinesAgoWereBlank = true
```

```
81          let previousLineWasComment = false
82
83          for(let i = 0; i < file.length; i++) {
84              let thisLine = file[i].trim()
85              //technically untrimmed, need to remove \r on
                    windows though to get accurate counts.
86              let untrimmedLine = file[i].replace(new RegExp(
                    "\r", "g"), "")
87              let thisLineIsBlank = (thisLine === "")
88              let thisLineStartsComment = (thisLine.
                    startsWith("//") || thisLine.startsWith("/*"
                    ))
89              let thisLineEndsComment = (thisLine.startsWith(
                    "//") || thisLine.endsWith("*/"))
90
91              //Assume that lines which - when trimmed start
                    with *, and where previousLineWasComment is
                    true - are inside Javadoc comments.
92              let thisLineInBlockComment = (thisLine.
                    startsWith("/*") || thisLine.endsWith("*/")
93                   || (thisLine.startsWith("*") &&
                        previousLineWasComment))
94
95              let thisLineIsComment = (thisLineStartsComment
                    || thisLineEndsComment ||
                    thisLineInBlockComment)
96
97              if(thisLineIsBlank && previousLineWasBlank &&
                    twoLinesAgoWereBlank) {
98                  throw(`${bug.bugTitle} conf ${bv}-A has
                        three blank lines in a row at ${i}!`)
99              }
100
101             if(thisLineStartsComment) {
102                 if(!previousLineWasBlank && !
                        previousLineWasComment) {
103                     throw(`${bug.bugTitle} conf ${bv}-A has
                            a comment which is not preceeded by
                            a blank line or comment at ${i}!`)
104                 }
105             }
106
107             if(thisLineInBlockComment) {
108                 //Need to use original (untrimmed) length
109                 if(untrimmedLine.length > 70) {
110                     throw(`${bug.bugTitle} conf ${bv}-A has
```

```
                        a >70 column line in a block
                       comment at ${i}!\n"${file[i]}"`)
111                 }
112             } else if(untrimmedLine.length > 80) {
113                 throw(`${bug.bugTitle} conf ${bv}-A has a
                       >80 column line outside a block comment
                       at ${i}!\n"${file[i]}"`)
114             }
115
116             if(thisLine.startsWith("}") &&
                   previousLineWasBlank) {
117                 throw(`Blocks shouldn't end with blank
                       lines in ${bug.bugTitle}-${bv}-A at ${i
                       }!`)
118             }
119
120             previousLineWasComment = thisLineIsComment
121             twoLinesAgoWereBlank = previousLineWasBlank
122             previousLineWasBlank = thisLineIsBlank
123         }
124
125         return true;
126     }
127
128     return sv(bug, "bugged") && sv(bug, "fixed")
129 }
130
131 exports.check = function() {
132     log("Beginning sanity check of task files...")
133     let tasks = {}
134
135     try {
136         tasks = require(`${pathroot}/tasks.json`)
137     } catch(e) {
138         log("!!! Failed to get task description !!!")
139         return false;
140     }
141
142     log("Verifying control task...")
143     try {
144         let controlTask = getInfoFor(tasks.control.bugTitle
                )
145
146         if(tasks.control.config === "A")
                checkConfigAWhitespace(controlTask)
147         if(tasks.control.config === "B")
```

```
            checkConfigBWhitespace(controlTask)
148
149         if (!checkConfigExists(controlTask, tasks.control.
                config)
150          || !checkResponses(controlTask, tasks.control.
                config)) {
151             throw("Something was wrong with the task's info
                    .json!")
152         }
153
154         if(!checkLicenseExists(controlTask)) {
155             throw("This task doesn't contain the original
                    sourcecode license!")
156         }
157
158         if(typeof controlTask.flipped !== "undefined") {
159             throw("Control task must declare if it is
                    flipped or not!")
160         }
161     } catch(e) {
162         log(`!!! Failed to verify (control) task "${tasks.
                control.bugTitle}" !!!`)
163         log(`The error: ${e}`)
164         return false;
165     }
166
167     log("Verifying remaining task(s) ...")
168     for(let i = 0; i < tasks.taskList.length; i++) {
169         log(`Verifying task "${tasks.taskList[i]}" ...`)
170         let taskName = tasks.taskList[i]
171         try {
172             let task = getInfoFor(taskName)
173
174             checkConfigAWhitespace(task)
175             checkConfigBWhitespace(task)
176
177             if(!checkConfigExists(task, "A")
178                     || !checkConfigExists(task, "B")
179                     || !checkConfigExists(task, "C")) {
180                 throw("A config was missing!")
181             } else if(!checkResponses(task, "A")
182                     || !checkResponses(task, "B")
183                     || !checkResponses(task, "C")) {
184                 throw("Some task responses were missing!")
185             }
186
```

```
187             if(!checkLicenseExists(task)) {
188                 throw("This task doesn't contain the
                        original sourcecode license!")
189             }
190         } catch(e) {
191             log(`!!! Failed to verify task "${taskName}"
                    !!!`)
192             log(`The error: ${e}`)
193             return false;
194         }
195     }
196
197     return true;
198 }
```

C.6 index.html

```
<html>
    <head>
        <title>Asdf</title>
        <style>
            html, body {
                width: 100%;
                height: 100%;
                font-family: 'Roboto', sans-serif;
                margin: 0;
                padding: 0;
            }
        </style>
        <script>
            window.Polymer = {lazyRegister: true, dom: '
               shadow'};
            window.electron = require('electron').remote;
            window.appExit = window.electron.app.quit;
            window.appMaximize = window.electron.
               getCurrentWindow().maximize;
            window.scienceTasks = window.electron.getGlobal
               ('tasks');
        </script>
        <link href="https://fonts.googleapis.com/css?
           family=Roboto" rel="stylesheet">
        <link rel="import" href="./my_components/
           appincludes.html">
    </head>
    <body>
        <app-frame title="SkriptIV Survey"></app-frame>
    </body>
</html>
```

C.7 my_components/appincludes.html

```
1  <link rel="import" href="../bower_components/polymer/
       polymer.html">
2  <link rel="import" href="../bower_components/paper-input/
       paper-input.html">
3  <link rel="import" href="../bower_components/paper-checkbox
       /paper-checkbox.html">
4  <link rel="import" href="../bower_components/paper-button/
       paper-button.html">
5  <link rel="import" href="../bower_components/
       paper-radio-group/paper-radio-group.html">
6  <link rel="import" href="../bower_components/
       paper-radio-button/paper-radio-button.html">
7  <link rel="import" href="../bower_components/iron-icons/
       iron-icons.html">
8  <link rel="import" href="../bower_components/iron-ajax/
       iron-ajax.html">
9  <link rel="import" href="../bower_components/neon-animation
       /neon-animation.html">
10 <link rel="import" href="../bower_components/paper-progress
       /paper-progress.html">
11 <link rel="import" href="../bower_components/paper-dialog/
       paper-dialog.html">
12 <link rel="import" href="../bower_components/
       paper-dialog-scrollable/paper-dialog-scrollable.html">
13 <link rel="import" href="../bower_components/
       paper-dropdown-menu/paper-dropdown-menu.html">
14 <link rel="import" href="../bower_components/paper-tabs/
       paper-tabs.html">
15
16 <link rel="import" href="./app-frame/app-frame.html">
17 <link rel="import" href="./database-input-page/
       database-input-page.html">
18 <link rel="import" href="./error-display-box/
       error-display-box.html">
19 <link rel="import" href="./demographics-survey/
       demographics-survey.html">
20 <link rel="import" href="./consent-form/consent-form.html">
21 <link rel="import" href="./multi-radio-question/
       multi-radio-question.html">
22 <link rel="import" href="./pre-test-page/pre-test-page.html
       ">
23 <link rel="import" href="./test-question/test-question.html
       ">
24 <link rel="import" href="./accurate-timer/accurate-timer.
```

```
    html">
25 <link rel="import" href="./thank-page/thank-page.html">
```

C.8 my_components/accurate-timer/accurate-timer.html

```
<dom-module id="accurate-timer">
    <script>
        Polymer({
            is: "accurate-timer",
            properties: {
                timerDuration: {
                    type: Number,
                    value: 0,
                },
                timeNow: {
                    type: Number,
                    value: 0,
                    readOnly: true,
                    notify: true
                },
                isRunning: {
                    type: Boolean,
                    value: false,
                    readOnly: true,
                    notify: true
                },
                percentageComplete: {
                    type: Number,
                    computed: 'computePercentage(timeNow,
                       timerDuration)',
                    notify: true
                }
            },
            start: function() {
                //Use integer milliseconds only: We don't
                   want higher precision
                //as the computer might be lying/
                   inconsistent about higher precision time
                   .
                //
                //See https://developer.mozilla.org/en-US/
                   docs/Web/API/DOMHighResTimeStamp
                this._timeStarted = Math.ceil(window.
                   performance.now())
                this._setTimeNow(0)

                if(!this.isRunning) {
                    window.requestAnimationFrame(this.
                       _update.bind(this))
```

```
38                      this._setIsRunning(true)
39                  } else {
40                      //don't make multiple
                           requestAnimationFrame calls
41                      console.warn("Timer was started before
                           it was finished! The timer has been
                           **RESET**"
42                              + "and the first time-out **
                                   MIGHT NOT** ever happen!
                                   Don't call start() twice!
                                   ")
43                  }
44              },
45              _update: function(z) {
46                  if(!this.isRunning) {
47                      //quit if we're not supposed to be
                           running the timer
48                      return;
49                  }
50
51                  if(!this.timerDuration || !this.
                       _timeStarted) {
52                      console.warn("Something went wrong with
                            the timer! It was cancelled as a
                           result.")
53                      this._setIsRunning(false)
54                      return;
55                  }
56
57                  //Use integer milliseconds only: We don't
                       want higher precision
58                  //as the computer might be lying/
                       inconsistent about higher precision time
                       .
59                  //
60                  //See https://developer.mozilla.org/en-US/
                       docs/Web/API/DOMHighResTimeStamp
61                  var timestampNow = Math.ceil(z)
62
63                  //compute the number of milliseconds that
                       has passed since the timer was
64                  //started with start()
65                  var now = timestampNow - this._timeStarted
66
67                  //if we are done...
68                  if(now >= this.timerDuration) {
```

```
69                      //results in exactly 100%
70                      this._setTimeNow(this.timerDuration)
71                      //fire clock event
72                      this.fire('timeout')
73                      this._setIsRunning(false)
74                      return;
75                  }
76
77                  //do every 3rd frame
78                  if(now % 3 == 0) {
79                      this._setTimeNow(now)
80                  }
81
82                  window.requestAnimationFrame(this._update.
                        bind(this))
83              },
84              computePercentage: function(timeNow,
                    timerDuration) {
85                  if(!(!timerDuration) && timerDuration > 0)
                        {
86                      return Math.ceil((timeNow/timerDuration
                            )*10000)/100;
87                  }
88
89                  return 0;
90              }
91          })
92      </script>
93  </dom-module>
```

C.9 my_components/app-frame/app-frame.html

```
<dom-module id="app-frame">
    <template>
        <style>
            :host , #fullbleed-container {
                display: block;
                position: absolute;
                left: 0;
                top: 0;
                width: 100%;
                height: 100%;
                box-sizing: border-box;
                padding: 0px;
                margin: 0px;
                background-color: #aaa;

                --button-theme: {
                    color: #F1F8EB;
                    background-color: #4A6F5F;
                }
            }

            #title-bar {
                position: absolute;
                left: 0;
                top: 0;
                width: 100%;
                height: 27px;
                background-color: #5E9990;
                padding-left: 8px;
                box-sizing: border-box;
            }

            .title {
                position: relative;
                top: 3px;
                display: inline-block;
                color: #5E195F;
            }

            .dragregion {
                position: absolute;
                width: calc(100% - 40px);
                height: 27px;
                left: 0;
```

```
45                  top: 0;
46                  background-color: transparent;
47                  -webkit-app-region: drag;
48              }
49
50              .closebutton {
51                  position: absolute;
52                  width: 36px;
53                  height: 25px;
54                  right: 2px;
55                  top: 0;
56                  background-color: #5E9990;
57                  color: #CFD0CE;
58                  overflow:none;
59                  transition: background-color .25s, color
                        .25s;
60              }
61
62              .closebutton:hover {
63                  background-color: #B5453F;
64                  color: white;
65              }
66
67              #content-area {
68                  position: absolute;
69                  left: 0;
70                  top: 27px;
71                  width: 100%;
72                  height: calc(100% - 27px);
73                  background-color: #F1F8EB;
74                  overflow: none;
75                  color: #4C5954;
76                  transform: translate(0,0);
77              }
78
79              #contentShade {
80                  position: absolute;
81                  left: 0;
82                  top: 27px;
83                  width: 100%;
84                  height: calc(100% - 27px);
85                  background-color: #F1F8EB;
86                  overflow: none;
87                  color: #4C5954;
88                  display: flex;
89                  justify-content: center;
```

```
90                  align-items: center;
91                  transition: height 3s, opacity 1s;
92                  transition-timing-function: ease;
93              }
94
95              #contentShade.hidden {
96                  height: 0;
97                  opacity: 0;
98              }
99
100             #closeicon {
101                 position: absolute;
102                 height: 25px;
103                 width: 25px;
104                 left: 6px;
105                 top: 0;
106             }
107
108             neon-animated-pages {
109                 width: 100%;
110                 height: 100%;
111                 overflow: hidden;
112             }
113
114             neon-animatable {
115                 @apply(--layout-horizontal);
116             }
117         </style>
118
119         <div id="fullbleed-container">
120             <div id="title-bar">
121                 <span class="title">{{title}}</span>
122                 <div class="dragregion"></div>
123                 <div class="closebutton" onclick="window.
                        appExit()">
124                     <iron-icon id="closeicon" icon="icons:
                            close"></iron-icon>
125                 </div>
126             </div>
127             <div id="content-area">
128                 <neon-animated-pages id="pages" selected="
                        [[selectedPage]]" entry-animation="
                        slide-from-right-animation"
                        exit-animation="slide-left-animation">
129                     <neon-animatable><database-input-page
                            shared-data="{{sharedData}}"></
```

```
                        database-input-page></
                        neon-animatable>
130                     <neon-animatable><consent-form
                        shared-data="{{sharedData}}"></
                        consent-form></neon-animatable>
131                     <neon-animatable><demographics-survey
                        shared-data="{{sharedData}}"></
                        demographics-survey></
                        neon-animatable>
132                     <neon-animatable><pre-test-page></
                        pre-test-page></neon-animatable>
133                     <template is="dom-repeat" items="{{
                        _createdConfigs}}">
134                      <neon-animatable>
135                          <test-question
136                              bug-title="[[item.bugTitle
                                ]]"
137                              bug-config="[[item.
                                bugConfig]]"
138                              flipped="[[item.flipped]]"
139                              shared-data="{{sharedData}}
                                "
140                              bug-number=[[_addOne(index)
                                ]]>
141                          </test-question>
142                      </neon-animatable>
143                     </template>
144                     <neon-animatable><thank-page
                        shared-data="{{sharedData}}"></
                        thank-page></neon-animatable>
145                 </neon-animated-pages>
146                 <div id="contentShade"><div>App is loading.
                       </div></div>
147             </div>
148         </div>
149     </template>
150     <script>
151         Polymer({
152             is: 'app-frame',
153             properties: {
154                 title: {
155                     type: String,
156                     value: "[No Title?]"
157                 },
158                 selectedPage: {
159                     type: Number,
```

```
160                     value: 0,
161                     notify: true
162                 },
163                 sharedData: {
164                     type: Object,
165                     value: {}
166                 }
167             },
168             ready: function() {
169                 window.appMaximize();
170                 this.sharedData = {
171                     db: {},
172                     results: {}
173                 };
174 
175                 window.setTimeout((function (){
176                     //This is from the MAIN PROCESS, set at
                        launch time as a global.
177                     //It is captured under the name
                        scienceTasks in index.html so
178                     //we can use it here. Because this
                        object is from the MAIN PROCESS,
179                     //it is guaranteed to have passed the
                        integrity checker already.
180                     let taskObj = window.scienceTasks;
181 
182                     //Shuffle the "decks" of configs and
                        tasks
183                     let tmpA = this._SecureShuffleArray(
                        taskObj.taskList);
184                     let tmpB = this._SecureShuffleArray(
                        taskObj.configSet);
185                     let tmpC = this.
                        _MakeSecureCoinTossArray(taskObj.
                        configSet.length);
186                     //all three of these resulting arrays
                        should be of the same length;
187                     //this is a property that (should) be
                        guaranteed
188 
189                     //an array, starting with the
                        description of the control task.
190                     let tmpD = [{
191                         bugTitle: taskObj.control.bugTitle,
192                         bugConfig: taskObj.control.config,
193                         flipped: (taskObj.control.flipped
```

```
                        === "true")
194                 }];
195
196                 //Riffle through the decks and make
                        coherent configs.
197                 //Add them all to tmpD.
198                 for(let i = 0; i < tmpA.length; i++) {
199                     tmpD.push({
200                         bugTitle: tmpA[i],
201                         bugConfig: tmpB[i],
202                         flipped: tmpC[i]
203                     })
204                 }
205
206                 //Force Polymer to notice we are
                        editing _createdConfigs
207                 this._createdConfigs = [];
208                 //Publish the full config list in one
                        giant lump.
209                 //This cascades into the DOM, and now
                        we should have
210                 //a full set of <test-question>
                        elements
211                 this._createdConfigs = tmpD;
212
213                 window.setTimeout((function() {
214                     this.pages = Polymer.dom(this.root)
                            .querySelectorAll("
                            neon-animatable");
215                     for(var i = 0; i < this.pages.
                            length; i++) {
216                         this.pages[i].children[0].
                                addEventListener('advance',
                                this._onAdvance.bind(this));
217                     }
218                     this.toggleClass("hidden", true,
                            this.$.contentShade);
219                 }).bind(this), 2000);
220             }).bind(this), 500);
221         },
222         _onAdvance: function(e, detail) {
223             if(this.selectedPage < this.pages.length -
                    1) {
224                 this.selectedPage++;
225                 this.fire('activate', {}, {node: this.
                        pages[this.selectedPage].children
```

```
                    [0]});
226             }
227         },
228         _SecureShuffleArray: function(array) {
229             //This method uses the window.crypto module
                    to shuffle an array.
230             //The use of the window.crypto module
                    ensures that the randomness
231             //would be suitable enough for cryptography
                    ; so we should assume
232             //that if the random numbers are good for
                    crypto they are random
233             //enough for science.
234             //
235             //Note: because we make a Uint8Array, the
                    array length must be less
236             //than 2^8 elements long for this to work
                    correctly. I doubt this
237             //will be a problem for MY purposes,
238             //but keep it in mind if you reuse this
                    code.
239             //
240             //incidentally, since we are using Modulo
                    in this algorithm for shuffling,
241             //there may be a possible bias towards one
                    end of the list or the other.
242             //We don't care about this bias *
                    particularly* much? but it would be
243             //very hard to write this function
                    differently.
244
245             //copy out the original array contents into
                    a new array we control.
246             let result = [];
247             for(let i = 0; i < array.length; i++) {
248                 result.push(array[i])
249             }
250
251             //generate 32x more potential array swaps
                    than there are items
252             //in the array
253             let shuffleTargetsA = new Uint8Array(array.
                    length * 32);
254             window.crypto.getRandomValues(
                    shuffleTargetsA);
255
```

```
256                 let shuffleTargetsB = new Uint8Array(array.
                        length * 32);
257                 window.crypto.getRandomValues(
                        shuffleTargetsB);
258
259                 //Swap Party!
260                 for(let i = 0; i < shuffleTargetsA.length;
                        i++) {
261                     let A = result[shuffleTargetsA[i] %
                            result.length];
262                     let B = result[shuffleTargetsB[i] %
                            result.length];
263                     result[shuffleTargetsA[i] % result.
                            length] = B;
264                     result[shuffleTargetsB[i] % result.
                            length] = A;
265                 }
266
267                 //return the result of doing all this.
268                 return result;
269             },
270             _MakeSecureCoinTossArray: function(arrayLength)
                    {
271                 //This method uses the window.crypto module
                        to simulate a lot of coin tosses.
272                 //The use of the window.crypto module
                        ensures that the randomness
273                 //would be suitable enough for cryptography
                        ; so we should assume
274                 //that if the random numbers are good for
                        crypto they are random
275                 //enough for science.
276
277                 let cryptoTargets = new Uint8Array(
                        arrayLength);
278                 window.crypto.getRandomValues(cryptoTargets
                        );
279                 let result = [];
280                 for(let i = 0; i < arrayLength; i++) {
281                     result.push(((cryptoTargets[i] % 2) ===
                            0));
282                 }
283                 return result;
284             },
285             _addOne: function(x) {
286                 return x + 1;
```

```
            }
        })
    </script>
</dom-module>
```

C.10 my_components/consent-form/consent-form.html

```
<dom-module id="consent-form">
    <template>
        <style>
            :host {
                display: flex;
                width: 100%;
                height: 100%;
                justify-content: center;
                align-items: center;
                flex-direction: column;
            }

            #doublecontain {
                display: flex;
                width: 100%;
                max-width: 66em;
                height: 100%;
                justify-content: center;
                align-items: center;
                flex-direction: column;
            }

            #consenttext {
                width: 100%;
                max-height: calc(100% - 5em);
                overflow: auto;
                box-sizing: border-box;
                margin-bottom: 1em;
                padding-left: 4em;
                padding-right: 4em;
            }

            paper-button {
                @apply(--button-theme);
                margin-right: 4em;
                visibility: hidden;
            }

            #agreebox {
                margin-right: 2em;
            }
        </style>
        <div id="doublecontain">
            <error-display-box centered id="error"></
```

```
            error-display-box>
45          <div id="consenttext"></div>
46          <div style="display:flex;justify-content:
               flex-end;align-items:baseline;width:calc
               (100% - 100px);padding-bottom:1em">
47           <div style="height:1em"><paper-checkbox
                no-ripple checked="{{checked}}" id="
                agreebox">I Agree</paper-checkbox></div>
48           <paper-button id="nextbutton" disabled="{{!
                checked}}" raised on-tap="_doConsent"
                >next</paper-button>
49          </div>
50      </div>
51  </template>
52  <script>
53      Polymer({
54          is: 'consent-form',
55          properties: {
56              sharedData: {
57                  type: Object,
58                  value: {},
59                  notify: true
60              }
61          },
62          listeners: {
63              'activate': 'activatePage'
64          },
65          activatePage: function() {
66              this.sharedData.db.get('consent', function(
                    err, doc) {
67                  if(err) {
68                      this.$.error.setText("Something
                            went wrong: " + String(err));
69                  } else {
70                      this.$.consenttext.innerHTML = doc.
                            formHTML;
71                      this.consentVersion = doc.
                            formVersion;
72                      this.$.nextbutton.style.visibility=
                            "visible";
73                  }
74              }.bind(this));
75          },
76          _doConsent: function() {
77              this.sharedData.results.consent = {
78                  consented: 1,
```

```
79                      consentVersion: this.consentVersion
80                  };
81
82                  this.fire('advance');
83              }
84          })
85      </script>
86  </dom-module>
```

C.11 my_components/database-input-page/database-input-page.html

```
<dom-module id="database-input-page">
    <template>
        <style>
            :host {
                display: flex;
                width: 100%;
                height: 100%;
                justify-content: center;
                align-items: center;
            }

            #locfield {
                display: inline-block;
                width: 73%;
                margin-right: 2%;
            }

            #portfield {
                display: inline-block;
                width: 25%
            }

            paper-button {
                @apply(--button-theme);
            }
        </style>
        <div>
            <h2 style="margin:0">Where is the database
               located?</h2>
            <i>The test administrator can tell you what to
               input here.</i>
            <br>
            <paper-input always-float-label id="locfield"
               value="" label="Database Location"></
               paper-input><paper-input always-float-label
               id="portfield" value="" label="Port"></
               paper-input>
            <error-display-box id="error"></
               error-display-box>
            <div style="display:flex;width:100%;height:2em;
               justify-content:flex-end;margin-top:4em">
                <paper-button id="submit" raised on-tap="
```

```
                    _tryLocation">Next</paper-button>
35              </div>
36          </div>
37      </template>
38      <script>
39          Polymer({
40              is: 'database-input-page',
41              properties: {
42                  sharedData: {
43                      type: Object,
44                      value: {},
45                      notify: true
46                  }
47              },
48              ready: function() {
49                  try {
50                      this.cradle = require('cradle');
51                  } catch(e) {
52                      this._setDisabled();
53                      this.$.error.setText("Something went
                            wrong: " + String(e));
54                  }
55              },
56              _tryLocation: function() {
57                  var db;
58                  this._setDisabled();
59                  this.$.error.setText("Working...");
60                  try {
61                      db = new(this.cradle.Connection)(this.$
                            .locfield.value,
62                          this.$.portfield.value, {auth: {
63                              username: 'researcher',
64                              password: '12345'
65                          }}).database('research')
66                  } catch(e) {
67                      this._setFailure(String(e));
68                      return;
69                  }
70
71                  db.exists(function(err,exists) {
72                      if(err) {
73                          this._setFailure(String(err));
74                      } else {
75                          if(!exists) {
76                              this._setFailure("The database
                                    couldn't be found.");
```

```
                        } else {
                            this._setSuccess(db);
                        }
                    }
                }.bind(this))
            },
            _setDisabled: function() {
                this.$.locfield.disabled = true;
                this.$.portfield.disabled = true;
                this.$.submit.disabled = true;
            },
            _setFailure: function(str) {
                this.$.locfield.disabled = false;
                this.$.portfield.disabled = false;
                this.$.submit.disabled = false;
                this.$.error.setText(str);
            },
            _setSuccess: function(db) {
                this.sharedData.db = db;
                this.$.error.displayText="";
                this.fire('advance');
            }
        })
    </script>
</dom-module>
```

C.12 my_components/demographics-survey/demographics-survey.html

```
<dom-module id="demographics-survey">
    <template>
        <style>
            :host {
                display: flex;
                width: 100%;
                height: 100%;

                justify-content: center;
                align-items: stretch;
            }

            #container {
                display: flex;
                width: 100%;
                max-width: 66em;
                height: 100%;
                padding-left: 8em;
                padding-right: 8em;
                box-sizing: border-box;
                flex-direction: column;
                justify-content: center;

                overflow: auto;
                transform: translate(0,0);
            }

            #next-holder {
                width: 100%;
                position: relative;
                top: 1em;
            }

            #next-holder paper-button {
                position: absolute;
                right: 6em;
            }

            paper-button {
                @apply(--button-theme);
            }

```

```
43              p {
44                  margin: 0;
45                  text-align: justify;
46              }
47
48              span {
49                  position: relative;
50                  top: 1px;
51              }
52          </style>
53
54          <div id="container">
55              <multi-radio-question num-items="5" selected="
                    {{q1Answer}}">
56                  <p>About how many years of experience do
                        you have programming computers in
                        <i>any</i> programming language?<br>(If
                        more than 5, answer 5)</p>
57              </multi-radio-question>
58              <error-display-box centered id="q1error"></
                    error-display-box>
59              <multi-radio-question num-items="5" selected="
                    {{q2Answer}}">
60                  <p>About how many years of experience do
                        you have programming in the <i>Java</i>
                        programming language?<br>(If more than
                        5, answer 5)</p>
61              </multi-radio-question>
62              <error-display-box centered id="q2error"></
                    error-display-box>
63              <multi-radio-question num-items="0" menu-items=
                    '["Beginner","Intermediate","Advanced"]'
                    selected="{{q3Answer}}">
64                  <p>Would you describe your own experience
                        with the Java programming language to be
                         at a beginner, intermediate, or
                        advanced level?</p>
65              </multi-radio-question>
66              <error-display-box centered id="q3error"></
                    error-display-box>
67              <multi-radio-question num-items="4" selected="
                    {{q4Answer}}">
68                  <p>On a scale of 1 to 4, indicate how much
                        do you agree with the following
                        statement: When I must read code, I
                        prefer to read code which adheres to a
```

```
                    consistent style.</p>
                <span left-label>Strongly Disagree</span>
                <span right-label>Strongly Agree</span>
            </multi-radio-question>
            <error-display-box centered id="q4error"></
                error-display-box>
            <multi-radio-question num-items="4" selected="
                {{q5Answer}}">
                <p>On a scale of 1 to 4, indicate how much
                    you agree with the following statement:
                    It does not matter to me whether code I
                    read is well-formatted or rather
                    disorganized; I can still read it anyway
                    .</p>
                <span left-label>Strongly Disagree</span>
                <span right-label>Strongly Agree</span>
            </multi-radio-question>
            <error-display-box centered id="q5error"></
                error-display-box>
            <div id="next-holder"><paper-button raised
                on-tap="_validateForm">Next</paper-button></
                div>
        </div>

    </template>
    <script>
        Polymer({
            is: 'demographics-survey',
            properties: {
                sharedData: {
                    type: Object,
                    notify: true,
                    value: {}
                }
            },
            _validateForm: function() {
                var error = false;
                for(var i = 1; i <= 5; i++) {
                    if(!this["q" + i + "Answer"]) {
                        error = true;
                        this.$["q" + i + "error"].setText('
                            This question is required.');
                    } else {
                        this.$["q" + i + "error"].setText
                            ('');
                    }
```

```
102                 }
103
104                 if(!error) {
105                     this.sharedData.results.
                           demographicsResults = {
106                         q1: this.q1Answer,
107                         q2: this.q2Answer,
108                         q3: this.q3Answer,
109                         q4: this.q4Answer,
110                         q5: this.q5Answer
111                     };
112
113                     console.log(this.sharedData);
114
115                     this.fire('advance');
116                 }
117             }
118         })
119     </script>
120 </dom-module>
```

C.13 my_components/error-display-box/error-display-box.html

```
<dom-module id="error-display-box">
    <template>
        <style>
            :host {
                display: block;
                position: relative;
                left: 0;
                top: 0;
                width: 100%;
                height: 1.2em;
                transition: height .225s;
            }

            :host([centered]) {
                text-align: center;
            }

            #shade {
                position: absolute;
                left: 0;
                top: 0;
                height: 0;
                width: 100%;
                overflow: hidden;
                transition: height .225s;
                background: transparent;
                margin: 0;
                padding: 0;
                text-align: inherit;
            }

            #content {
                position: absolute;
                left: 0;
                top: 0;
                width: 100%;
                color: #B5453F;
                margin: 0;
                padding: 0;
                text-align: inherit;
            }

```

```
43              #content.blink {
44                  animation-name: highlight-blink;
45                  animation-duration: 2s;
46              }
47
48              @keyframes highlight-blink {
49                  0%   {
50                      animation-timing-function: cubic-bezier
                           (0.4, 0.0, 1, 1);
51                      text-shadow: transparent 0px 0px 2px;
52                  }
53
54                  7%  {
55                      animation-timing-function: cubic-bezier
                           (0.4, 0.0, 0.6, 1);
56                      text-shadow: rgba(255,0,0,.7) 0px 0px
                           11px;
57                  }
58
59                  100% {
60                      text-shadow: transparent 0px 0px 2px;
61                  }
62              }
63          </style>
64
65          <div id="shade">
66              <div id="content"></div>
67          </div>
68
69      </template>
70
71      <script>
72          Polymer({
73              is: 'error-display-box',
74              properties: {
75                  displayText: {
76                      type: String,
77                      value: "",
78                      notify: true,
79                      observer: '_textChanged'
80                  },
81                  centered: {
82                      type: Boolean,
83                      value: false,
84                      reflectToAttribute: true,
85                      notify: true
```

```
86                  }
87              },
88              listeners: {
89                  'blink': '_blink'
90              },
91              setText: function(text) {
92                  if(this.displayText !== String(text)) {
93                      this.displayText = String(text);
94                  } else {
95                      this.fire('blink');
96                  }
97              },
98              _textChanged: function(newValue, oldValue) {
99                  if(newValue === "") {
100                     this.$.shade.style.height = "0px";
101                     Polymer.dom(this.root).node.host.style.
                           height = "1.2em";
102                 } else {
103                     this.$.content.innerHTML = String(
                           newValue);
104                     var height = this.$.content.
                           getBoundingClientRect().height;
105                     if(Polymer.dom(this.root).node.host.
                           getBoundingClientRect().height !==
                           height) {
106                         Polymer.dom(this.root).node.host.
                               style.height = String(height) +
                               "px";
107                     } else {
108                         Polymer.dom(this.root).node.host.
                               style.height = "1.2em";
109                     }
110                     this.$.shade.style.height = String(
                           height) + "px";
111                     this.fire('blink');
112                 }
113             },
114             _blink: function() {
115                 this.toggleClass('blink', false, this.$.
                       content);
116                 setTimeout(function(){
117                     this.toggleClass('blink', true, this.$.
                           content);
118                 }.bind(this), 5);
119             }
120         })
```

```
121 |     </script>
122 | </dom-module>
```

C.14 my_components/multi-radio-question/multi-radio-question.html

```
<dom-module id="multi-radio-question">
    <template>
        <style>
            :host {
                display: flex;
                width: 100%;
                flex-direction: column;
                margin-bottom: .5em;
                margin-top: 1em;
            }

            #textcontainer {
                width: 100%;
                padding-bottom: .5em;
            }

            #radiocontainer {
                display: flex;
                justify-content: center;
                width: 100%;
                align-items: baseline;
            }

            #radiocontainer ::content div {
                position: relative;
                top: 2px;
            }
        </style>

        <div id="textcontainer">
            <content select="p"></content>
        </div>

        <div id="radiocontainer">
            <content select="[left-label]"></content>
            <paper-radio-group selected="{{selected}}">
                <template is="dom-repeat" items="[[
                   menuItems]]">
                    <paper-radio-button name="[[item]]">[[
                       item]]</paper-radio-button>
                </template>
            </paper-radio-group>
```

```
41              <content select="[right-label]"></content>
42          </div>
43
44      </template>
45
46      <script>
47          Polymer({
48              is: 'multi-radio-question',
49              properties: {
50                  selected: {
51                      type: Number,
52                      value: 0,
53                      notify: true
54                  },
55                  numItems: {
56                      type: Number,
57                      value: 1,
58                  },
59                  menuItems: {
60                      type: Array
61                  }
62              },
63              ready: function() {
64                  if(this.numItems > 0) {
65                      this.menuItems = [];
66                      for(var i = 0; i < this.numItems; i++)
                            {
67                          this.push('menuItems', i+1);
68                      }
69                  }
70              }
71          })
72      </script>
73  </dom-module>
```

C.15 my_components/pre-test-page/pre-test-page.html

```
<dom-module id="pre-test-page">
    <template>
        <style>
            :host {
                display: flex;
                width: 100%;
                height: 100%;
                justify-content: center;
                align-items: center;
                flex-direction: column;
            }

            #container {
                box-sizing: border-box;
                padding: 4em 2em 2em 2em;
                max-width: 60em;
                height: 100%;
                overflow: auto;
            }

            paper-button {
                @apply(--button-theme);
            }
        </style>
        <div id="container">
            <h3>Survey complete.</h3>
            <p>
                You are about to move on to the testing
                   portion of the study.
                <strong>Please be sure to thoroughly read
                   through these instrucitons, and that you
                    understand them completely before
                   continuing.</strong>
                Make sure you scroll down to view all of
                   the instructions. If you have any
                   questions after reading them, you can
                   ask your test administrator.
            </p>
            <p>
                For the testing portion of the study, you
                   will complete seven (7) different
                   debugging and code comprehension tasks.
                For each task, you will be given a brief
                   description of a real-world bug which
```

```
                has been found in an open-source project
                .
35              The bug description contains the kind of
                information you might be able to
                discover about the bug if you ran the
                code in a debugger,
36              such as the method which contains the bug,
                and an example of code which triggers
                the bug. Please read the descriptions
                carefully,
37              as they will help you to debug the code.
38          </p>
39          <p>
40              Next, when you are ready, you will be shown
                 two source-code files side-by-side. One
                 of these files contains the bug you are
                 looking for,
41              and the other file is a fixed version of
                the same code, meaning that its code is
                <em>correct</em>, and does not contain
                the bug.
42              Your task is to read through the files,
                locate the method you were directed to
                from the bug description, and determine
                which code
43              has the bug and which has been fixed.
                Remember that you will only have
                <strong>five (5)</strong> minutes to
                complete each task,
44              however the five minute timer does not
                start until you have finished reading
                the bug descriptions. A timer and a
                progress bar
45              will be at the top of the page to show you
                how much time you have left.
46          </p>
47          <p>
48              Finally, you should make a note of which
                line number the bug was found on
                <strong>in the buggy file</strong>. As
                an example, you
49              may be reading some code and have
                determined that the buggy file is the
                one on the left. You should write in the
                 line number of
50              where the bug was found in the
```

```
                <strong>left</strong> file, and
                <strong>not</strong> the line number of
                the fixed code on the right.
            There may be some cases in which there is
                more than one correct answer, because
                multiple lines of code are incorrect. If
                 that is the
            case, you can pick any one of those
                incorrect lines. When you are done,
                check the box indicating that this is
                your final answer, and click
            "Submit"
        </p>
        <p>
            What makes these tasks special is that the
                code which you will be reading will be
                formatted slightly differently as you go
                 along.
            You may encounter code which is formatted
                so that it is almost impossible to read,
                 code which is rather well-formatted,
            and everything in between. We ask that you
                try your best to complete each of the
                tasks as best as you can, even
            when the code is hard to read.
        </p>
        <p>
            In summary, for each of the seven tasks,
                you will:
        </p>
        <ol>
            <li>Read the description of a bug</li>
            <li>Start a five minute timer, and search
                for the bug before the time is up</li>
            <li>Record which file the bug was in, and
                one of the line numbers in the file you
                found the bug</li>
            <li>Check the checkbox indicating your
                final answer has been chosen, and then
                press the "Submit" button.</li>
        </ol>
        <p>When you are ready, click "Next" to begin.</
            p>
        <br>
        <br>
        <div style="width:100%;display:flex;
```

```
                flex-direction:row-reverse">
74                <paper-button id="button" disabled="true"
                      on-tap="_doAdvance">Next</paper-button>
75            </div>
76        </div>
77    </template>
78    <script>
79        Polymer({
80            is: 'pre-test-page',
81            listeners: {
82                'activate': 'activatePage'
83            },
84            activatePage: function() {
85                window.setTimeout((function() {
86                    this.$.button.disabled = false;
87                    this.$.button.raised = true;
88                }).bind(this), 10000);
89            },
90            _doAdvance: function() {
91                this.fire('advance')
92            }
93        })
94    </script>
95</dom-module>
```

C.16 my_components/test-question/test-question.html

```
<dom-module id="test-question">
    <template>
        <style>
            :host {
                display: flex;
                position: relative;
                width: 100%;
                height: 100%;
                justify-content: center;
                align-items: center;
            }

            paper-button {
                @apply(--button-theme);
            }

            paper-button.desc {
                position: relative;
                top: 1.2em;
                height: 2em;
                color: #4A6F5F;
                background-color: #F1F8EB;
                left: 1em;
            }

            paper-progress {
                display: block;
                width: 100%;
                position: absolute;
                bottom: 0;
                left: 0;
                --paper-progress-active-color: #39E5C8;
                --paper-progress-secondary-color: #93F0E1;
            }

            #header {
                position: absolute;
                top: 0;
                left: 0;
                height: 2em;
                width: 100%;
                line-height: 2em;
                vertical-align: center;
                text-align: left;
```

```
                box-sizing: border-box;
                padding-left: 8px;
                background: #8BA399;
            }

            #footer {
                position: absolute;
                bottom: 0;
                left: 0;
                right: 0;
                height: 4em;
                background-color: #F1F8EB;

                display: flex;
                flex-direction: row;
                flex-wrap: wrap;
                justify-content: space-between;
                align-items: flex-start;
                vertical-align: bottom;

                box-sizing: border-box;
            }

            #footer div {
                display: inline-block;
                height: 3.5em;
                line-height: 3.2em;
                vertical-align: bottom;
            }

            h3 {
                margin: 0;
                padding: 0;
            }

            #sizer {
                position: absolute;
                top: 2em;
                left: 0;
                bottom: 4em;
                right: 0;
            }

            #leftEditor {
                position: absolute;
                top: 0;
```

```
91                  left: 0;
92                  height: 100%;
93                  width: 50%;
94                  background: white;
95              }
96
97              #rightEditor {
98                  position: absolute;
99                  top: 0;
100                 left: 50%;
101                 height: 100%;
102                 width: 50%;
103                 background: white;
104             }
105
106             #responseDiv {
107                 padding-left: 1em;
108                 padding-bottom: .5em;
109                 height: 3em;
110             }
111
112             #lineAnswer {
113                 display:inline-block;
114                 width:5em;
115                 margin-left:1em;
116             }
117
118             #timeRemaining {
119                 height: 1em;
120                 position: absolute;
121                 bottom: .4em;
122                 right: .4em;
123                 line-height: 1em;
124                 vertical-align: bottom;
125             }
126
127             #timeRemaining code {
128                 font-size: 1.5em;
129             }
130
131             #wholeThing {
132                 position: absolute;
133                 top: 0;
134                 left: 0;
135                 width: 100%;
136                 height: 100%;
```

```
137             }
138
139             #sizerShield {
140                 position: absolute;
141                 top: 0;
142                 left: 0;
143                 width: 100%;
144                 height: 100%;
145                 background: #8BA399;
146                 z-index: 900;
147             }
148
149             #sizerShield[hidden] {
150                 display: none;
151             }
152
153             paper-tab {
154                 min-width: 100px;
155             }
156
157             #about {
158                 z-index: 901;
159             }
160
161             #outOfTimeBox {
162                 z-index: 901;
163             }
164         </style>
165         <div id="wholeThing">
166             <iron-ajax id="AJAXInfo" url="[[
                    _getURLForBugInfoObject(bugTitle)]]"
167                 on-error="_error" on-response="_activate2"
                        handle-as="json" last-response="{{
                        _bugInfoObject}}"></iron-ajax>
168             <iron-ajax id="AJAXBugged" url="[[
                    _getURLForBuggedSource(bugTitle,
                    _bugInfoObject.fileName,bugConfig)]]"
169                 on-error="_error" on-response="_activate3"
                        handle-as="text" last-response="{{
                        _buggedSource}}"></iron-ajax>
170             <iron-ajax id="AJAXFixed" url="[[
                    _getURLForFixedSource(bugTitle,
                    _bugInfoObject.fileName,bugConfig)]]"
171                 on-error="_error" on-response="_activate4"
                        handle-as="text" last-response="{{
                        _fixedSource}}"></iron-ajax>
```

```
172             <!-- This is a timer set for 300,000 millis, or
                    5 minutes. -->
173             <accurate-timer id="timer" timer-duration="
                   300000"
174                             percentage-complete="{{
                                   _timePercent}}"
175                             time-now="{{_timeNow}}"
176                             is-running="{{_timerRunning}}"
177                             on-timeout="_timeout"></
                                   accurate-timer>
178             <div id="header">
179                  <paper-progress id="progress"
180                     value="{{_timeNow}}"
181                     min="0"
182                     max="300000"
183                     step="1"></paper-progress>
184
185                 <h3>Bug <span>{{bugNumber}}</span>: <span>
                       {{bugTitle}}</span></h3>
186
187                 <div id="timeRemaining">Time left: <code>{{
                       formatTimeRemaining(_timeNow)}}</code></
                       div>
188             </div>
189             <div id="sizer">
190                 <div id="leftEditor"></div>
191                 <div id="rightEditor"></div>
192             </div>
193             <div id="footer">
194                 <paper-button on-tap="_openDesc" class="
                       desc">Show the description</
                       paper-button>
195                 <div>
196                 The bug is on the
197                 <paper-dropdown-menu style="margin-left:1em
                       ;margin-right:.5em;width:8em;" id="
                       sideAnswer"
198                                      label="side" vertical-align="bottom
                                            " horizontal-align="left"
                                            disabled="[[!_timerRunning]]">
199                      <paper-tabs class="dropdown-content"
                            selected="0">
200                          <paper-tab>left</paper-tab>
201                          <paper-tab>right</paper-tab>
202                      </paper-tabs>
203                 </paper-dropdown-menu>
```

```
204                 on line
205                 <paper-input style="margin-right: 1em;"
206                     id="lineAnswer" value="0" type="number"
                            label="Line" min="0"
207                     auto-validate allowed-pattern="[0-9]+"
                        disabled="[[!_timerRunning]]">
208                 </paper-input>
209                 <paper-checkbox style="position:relative;
                        top:-.3em;" disabled="[[!_timerRunning]]
                        " no-ripple checked="{{_checked}}" id="
                        agreebox">Final Answer!</paper-checkbox>
210                 <paper-button style="position:relative;
                        right:.7em;top:-1.2em;height:2.3em;
                        margin-left:5em;"
211                             disabled="[[_calcButton(
                                _timerRunning,_checked)]]"
212                             class="next"
213                             on-tap="_completeMission"
                                >Submit</paper-button>
214             </div>
215         </div>
216
217         <div id="sizerShield"
218                 hidden$="[[_timerRunning]]"></div>
219         <paper-dialog id="about"
220                       no-cancel-on-outside-click="true"
221                       opened="{{_showingDialog}}"
222                       on-iron-overlay-closed="
                             _closedDialog">
223             <h2>Bug <span>{{bugNumber}}</span>: <span
                    id="bugTitle"></span></h2>
224             <paper-dialog-scrollable>
225                 This bug is from a file called <code><
                        span>{{_bugInfoObject.fileName}}</
                        span>.java</code><br/>
226                 <br/>
227                 <h3>You should focus your attention on
                        this method:</h3>
228                 <code id="bugLocation"></code><br/>
229                 <br/>
230                 <h3>Here is a brief description of the
                        bug that may help you:</h3>
231                 <div id="bugDescription">
232                 </div>
233             </paper-dialog-scrollable>
234             <div class="buttons">
```

```
235                     <paper-button raised class="next"
                           dialog-confirm autofocus>OK</
                           paper-button>
236                 </div>
237             </paper-dialog>
238             <paper-dialog id="outOfTimeBox"
239                           no-cancel-on-outside-click="true"
240                           on-iron-overlay-closed="
                                 _failMission">
241                 <h2>Out of time!</h2>
242                 <paper-dialog-scrollable>
243                 You are out of time to work on this problem
                       . Press OK to move on to the next one.
244                 </paper-dialog-scrollable>
245                 <div class="buttons">
246                     <paper-button class="next"
                           dialog-confirm autofocus>OK</
                           paper-button>
247                 </div>
248             </paper-dialog>
249         </div>
250     </template>
251     <script src="../../bower_components/ace-builds/
           src-min-noconflict/ace.js"></script>
252     <script src="../../bower_components/ace-builds/
           src-min-noconflict/theme-github.js"></script>
253     <script>
254         Polymer({
255             is: "test-question",
256             properties: {
257                 bugTitle: {
258                     type: String,
259                     value: "",
260                     notify: true
261                 },
262                 bugConfig: {
263                     type: String,
264                     value: "",
265                     notify: true
266                 },
267                 sharedData: {
268                     type: Object,
269                     value: {},
270                     notify: true
271                 },
272                 flipped: {
```

```
273                     type: Boolean,
274                     value: false,
275                     notify: true
276                 },
277                 bugNumber: {
278                     type: Number,
279                     value: 0,
280                     notify: true
281                 }
282             },
283             listeners: {
284                 'activate': 'activatePage'
285             },
286             activatePage: function() {
287                 this._checked = false; //ensure this
                        property is set.
288                 window.setTimeout((function() {
289                     this.$.AJAXInfo.generateRequest();
290                 }).bind(this), 500);
291             },
292             _activate2: function() {
293                 this.$.AJAXBugged.generateRequest();
294             },
295             _activate3: function() {
296                 this.$.AJAXFixed.generateRequest();
297             },
298             _activate4: function() {
299                 //need this for later, so we only start the
                        timer once.
300                 this._started = false;
301
302                 if(this.flipped) {
303                     this.$.leftEditor.textContent = this.
                            _fixedSource;
304                     this.$.rightEditor.textContent = this.
                            _buggedSource;
305                 } else {
306                     this.$.leftEditor.textContent = this.
                            _buggedSource;
307                     this.$.rightEditor.textContent = this.
                            _fixedSource;
308                 }
309
310                 this._leftEditor = ace.edit(this.$.
                        leftEditor);
311                 this._rightEditor = ace.edit(this.$.
```

```
                rightEditor);
312             this._leftEditor.setTheme("ace/theme/github
                ");
313             this._rightEditor.setTheme("ace/theme/
                github");
314             this._leftEditor.getSession().setMode("ace/
                mode/java");
315             this._rightEditor.getSession().setMode("ace
                /mode/java");
316             this._leftEditor.setReadOnly(true);
317             this._rightEditor.setReadOnly(true);
318             this._leftEditor.setBehavioursEnabled(false
                );
319             this._rightEditor.setBehavioursEnabled(
                false);
320             this._leftEditor.setShowPrintMargin(false);
321             this._rightEditor.setShowPrintMargin(false)
                ;
322             this._leftEditor.setShowFoldWidgets(false);
323             this._rightEditor.setShowFoldWidgets(false)
                ;
324             this._leftEditor.setDisplayIndentGuides(
                false);
325             this._rightEditor.setDisplayIndentGuides(
                false);
326             this._leftEditor.setHighlightActiveLine(
                false);
327             this._rightEditor.setHighlightActiveLine(
                false);
328
329             var acss;
330             var tm;
331             var gh;
332
333             //Ace-Editor doesn't put its styles in the
                correct places when using Shadow Dom.
334             //We need to manually move them out of the
                document's <head> and into our scope.
335             //
336             //We're sharing references to the dom nodes
                 via SharedData but only because I'm
337             //not sure whether Ace-Editor updates the
                styles correctly after the first time
338             //it has been instantiated. Better to be
                safe (and use the same ones) than sorry
339             if(!this.sharedData.styles) {
```

```
340                     acss = document.querySelector("head").
                           querySelector("#ace_editor\\.css");
341                     tm = document.querySelector("head").
                           querySelector("#ace-tm");
342                     gh = document.querySelector("head").
                           querySelector("#ace-github");
343                     this.sharedData.styles = {
344                         a: acss,
345                         b: tm,
346                         c: gh
347                     }
348                 } else {
349                     acss = this.sharedData.styles.a
350                     tm = this.sharedData.styles.b
351                     gh = this.sharedData.styles.c
352                 }
353                 this.$.sizer.appendChild(acss);
354                 this.$.sizer.appendChild(tm);
355                 this.$.sizer.appendChild(gh);
356                 this.$.bugTitle.textContent = this.
                       _bugInfoObject.bugTitle;
357                 this.$.bugLocation.innerHTML = this.
                       _bugInfoObject.bugLocation;
358                 this.$.bugDescription.innerHTML = this.
                       _bugInfoObject.bugDescription;
359                 this._openDesc();
360             },
361             _openDesc: function() {
362                 this.$.about.open();
363             },
364             _closedDialog: function(x) {
365                 if(!this._started) {
366                     this.$.timer.start();
367                     console.log("Start timer...");
368                     this._started = true;
369                 } else {
370                     console.warn("Not starting the timer!")
                           ;
371                 }
372             },
373             _timeout: function(x) {
374                 if(this.$.about.opened) {
375                     this.$.about.close();
376                 }
377                 this.$.outOfTimeBox.open();
378             },
```

```
379             _failMission: function(x) {
380                 this._publishResultAndContinue({
381                     task: this._bugInfoObject.bugTitle,
382                     config: {
383                         bugVersion: this.bugConfig,
384                         flipped: this.flipped
385                     },
386                     result: "Timed-Out",
387                     timeRemaining: "0:00",
388                     rawResponse: {
389                         side: "",
390                         line: -1
391                     }
392                 });
393             },
394             _completeMission: function(x) {
395                 this._publishResultAndContinue({
396                     task: this._bugInfoObject.bugTitle,
397                     config: {
398                         bugVersion: this.bugConfig,
399                         flipped: this.flipped
400                     },
401                     result: this._getResult(),
402                     timeRemaining: this.formatTimeRemaining
                        (this._timeNow),
403                     rawResponse: {
404                         side: this.$.sideAnswer.value,
405                         line: this.$.lineAnswer.value
406                     }
407                 })
408             },
409             _getResult: function() {
410                 var correctResponses = this._bugInfoObject.
                        correctResponses[this.bugConfig];
411                 var responseLR = this.$.sideAnswer.value;
412                 var responseLine = this.$.lineAnswer.value;
413 
414                 console.log(this.flipped, responseLR, ((
                        this.flipped === true || this.flipped
                        === "true") && responseLR == "left"), ((
                        this.flipped === false || this.flipped
                        === "false") && responseLR == "right"))
415 
416                 //For some incredibly strange reason, this.
                        flipped
417                 //is sometimes a string "true"/"false".
```

```
418                 //
419                 //Instead of boolean-value true/false.
420                 //
421                 //wat.
422                 if((this.flipped === true || this.flipped
                       === "true") && responseLR == "left") {
423                     //if it's flipped, the FIXED code is on
                           the LEFT.
424                     return "Incorrect";
425                 } else if((this.flipped === false || this.
                       flipped === "false") && responseLR == "
                       right") {
426                     //if it's NOT flipped, the FIXED code
                           is on the RIGHT.
427                     return "Incorrect";
428                 }
429
430                 //So now we know that the LR response is
                       correct.
431                 for(var i = 0; i < correctResponses.length;
                        i++) {
432                     if(correctResponses[i] == responseLine)
                           {
433                         //We have found the line typed in
                               by the user in the
434                         //list of acceptable answers, we
                               know their answer is right.
435                         return "Correct";
436                     }
437                 }
438
439                 //We couldn't find the user's entered line
                       number in the list
440                 //of acceptable answers.
441                 return "Incorrect";
442             },
443             _nodecimals: function(x) {
444                 return Math.floor(x) == x;
445             },
446             _getURLForBugInfoObject: function(bugTitle,
                   bugConfig) {
447                 return(`../Defects-Sources/${bugTitle}/info
                       .json`)
448             },
449             _getURLForBuggedSource: function(bugTitle,
                   fileName, bugConfig) {
```

```
450                 return(`../Defects-Sources/${bugTitle}/${
                        fileName}-bugged-${bugConfig}.java`)
451             },
452             _getURLForFixedSource: function(bugTitle,
                    fileName, bugConfig) {
453                 return(`../Defects-Sources/${bugTitle}/${
                        fileName}-fixed-${bugConfig}.java`)
454             },
455             _error: function(x) {
456                 alert("ERROR!!!\n\nSomething has gone very
                        wrong and the test has crashed.\nPlease
                        inform the test operator!");
457                 console.log(x);
458             },
459             _calcButton: function(timerRunning, checked) {
460                 return !timerRunning || !checked;
461             },
462             formatTimeRemaining: function(timeNow) {
463                 var remainingMillis = 300000 - timeNow;
464                 var mins = Math.floor(Math.floor(
                        remainingMillis / 1000) / 60);
465                 var secs = String(Math.floor(Math.floor(
                        remainingMillis / 1000) % 60));
466                 if(secs.length < 2) {
467                     secs = "0" + secs;
468                 }
469                 return (mins + ":" + secs);
470             },
471             _publishResultAndContinue: function(
                    resultObject) {
472                 var s = this.sharedData;
473
474                 if(!s.results) {
475                     s.results = {}
476                 }
477                 if(!s.results.questionResults) {
478                     s.results.questionResults = [];
479                 }
480
481                 s.results.questionResults.push(resultObject
                        );
482
483                 this.fire('advance')
484             }
485         })
486     </script>
```

```
487 </dom-module>
```

C.17 my_components/thank-page/thank-page.html

```
<dom-module id="thank-page">
    <template>
        <style>
            :host {
                display: flex;
                position: relative;
                width: 100%;
                height: 100%;
                flex-direction: column;
                align-items: center;
                justify-content: center;
            }

            h3 {
                display: block;
            }

            p {
                max-width:40em;
                display: block;
            }
        </style>
        <h3>Thank you for all of your hard work!</h3>
        <p>Your contributions to science are highly
           appreciated.</p>
        <br/>
        <p>Your responses are being submitted...</p>
        <div id="result"></div>
    </template>
    <script>
        Polymer({
            is: "thank-page",
            properties: {
                sharedData: {
                    type: Object,
                    value: {},
                    notify: true
                }
            },
            listeners: {
                'activate': 'activatePage'
            },
            activatePage: function() {
                if(!this.sharedData.db) {
```

```
44                      this._failFallback("something,
                           somewhere went horribly wrong.");
45                  }
46
47                  this.sharedData.db.save(this.sharedData.
                       results, function(err, result) {
48                      if(err) {
49                          this._failFallback(err)
50                      } else {
51                          this._success()
52                      }
53                  }.bind(this))
54              },
55              _failFallback: function(error) {
56                  this.$.result.innerHTML =
57                      "<p>We're sorry, but something went
                           wrong when submitting your response
                           .\n"
58                      +"Please alert the test administrator
                           and have them help you.\n"
59                      +"Don't worry though, your contribution
                            is not lost, because here is your
                           data:</p><textarea "
60                      +"style='width:40em;height:8em'>"
61                      +(JSON.stringify(this.sharedData.
                           results))+"</textarea><p>(The exact
                           nature of the error was: "
62                      +(String(error)) + ")</p>";
63              },
64              _success: function() {
65                  this.$.result.innerHTML =
66                      "<p>Your results have been submitted
                           successfully.</p>"
67                      +"<p>You are now free to close the
                           survey application. You're all done!
                            And thanks again.</p>";
68              }
69          })
70      </script>
71  </dom-module>
```

Appendix D

SkriptIV Data Files

D.1 Defects-Sources/tasks.json

```
{
    "control": {
        "bugTitle": "Math-59",
        "config": "C",
        "flipped": "false"
    },

    "configSet": ["A", "A", "B", "B", "C", "C"],
    "taskList": ["Lang-1", "Chart-5", "Chart-8", "Math-26",
        "Mockito-18", "Time-15"]
}
```

D.2 Defects-Sources/Chart-5/info.json

```
{
    "bugTitle": "Chart-5",
    "bugLocation": "XYDataItem addOrUpdate(Number, Number)"
       ,
    "bugDescription": "With the following sequence of
       events, an <code>IndexOutOfBoundsException</code> is
        thrown in the <code>XYSeries</code> code:<br/><br/>
       <code style=\"white-space:pre\">    //Setup:\n
       XYSeries series = new XYSeries(\"Series\", true,
       true);\n    series.addOrUpdate(1.0, 1.0); // This
       works fine\n    \n    series.addOrUpdate(1.0, 2.0);
       //This triggers the bug</code>",
    "fileName": "XYSeries",
    "correctResponses": {
        "A": [570, 571],
        "B": [595, 597],
        "C": [548, 549]
    }
}
```

D.3 Defects-Sources/Chart-8/info.json

```
{
    "bugTitle": "Chart-8",
    "bugLocation": "&lt;Constructor&gt; Week(Date, TimeZone)",
    "bugDescription": "Given the following sequence of code, we get a Week that is the 34th week instead of the expected 35th.<br/><br/><code style=\"white-space: pre\">    //This is the setup:\n GregorianCalendar cal = (GregorianCalendar) Calendar.getInstance(TimeZone.getDefault());\n    cal.set(2007, Calendar.AUGUST, 26, 1, 0, 0);\n    cal.set(Calendar.MILLISECOND, 0);\n    Week w = new Week(cal.getTime(), TimeZone.getTimeZone(\"Europe/Copenhagen\"));\n\n    w.getWeek(); // This result is 34 instead of 35.</code>",
    "fileName": "Week",
    "correctResponses": {
        "A": [194],
        "B": [183],
        "C": [175]
    }
}
```

D.4 Defects-Sources/Lang-1/info.json

```
{
    "bugTitle": "Lang-1",
    "bugLocation": "Number createNumber(String)",
    "bugDescription": "Calling <code>createNumber
       (\"80000000\")</code> incorrectly throws <code>
       NumberFormatException</code> instead of producing a
       result.",
    "fileName": "NumberUtils",
    "correctResponses": {
        "A": [512,513,514,515,516,517,518,519,520,521],
        "B":
           [509,510,511,512,513,514,515,516,517,518,519,520,521,522,523

        "C": [469,470,471,472,473,474,475,476,477,478]
    }
}
```

D.5 Defects-Sources/Math-26/info.json

```
{
    "bugTitle": "Math-26",
    "bugLocation": "&lt;Constructor&gt; Fraction(double,
       double, int)",
    "bugDescription": "calling <code>new Fraction(-1.0e10,
       1.0e-12, 1000);</code> <strong>doesn't</strong>
       throw a <code>ConvergenceException</code>, which it
       should throw.",
    "fileName": "Fraction",
    "correctResponses": {
        "A": [193, 221],
        "B": [183, 214],
        "C": [160, 186]
    }
}
```

D.6 Defects-Sources/Math-59/info.json

```
{
    "bugTitle": "Math-59",
    "bugLocation": "float max(final float, final float)",
    "bugDescription": "<code>FastMath.max(50.0f, -50.0f)</
       code> results in -50 instead of 50",
    "fileName": "FastMath",
    "correctResponses": {
        "C": [3013]
    }
}
```

D.7 Defects-Sources/Mockito-18/info.json

```
{
    "bugTitle": "Mockito-18",
    "bugLocation": "Object returnValueFor(Class&lt;?&gt;)",
    "bugDescription": "Calling<br/><br/><code style=\"white
       -space:pre\">    ((Iterable) (new ReturnsEmptyValues
       ()).returnValueFor(Iterable.class)).iterator().
       hasNext</code><br/><br/>results in a <code>
       NullPointerException</code>",
    "fileName": "ReturnsEmptyValues",
    "correctResponses": {
        "A": [130],
        "B": [161],
        "C": [130]
    }
}
```

D.8 Defects-Sources/Time-15/info.json

```
{
    "bugTitle": "Time-15",
    "bugLocation": "long safeMultiply(long, int)",
    "bugDescription": "Calling <code>safeMultiply(Long.
       MIN_VALUE, -1)</code> does not throw an <code>
       ArithmeticException</code> as it should.",
    "fileName": "FieldUtils",
    "correctResponses": {
        "A": [148],
        "B": [152],
        "C": [141]
    }
}
```

Appendix E

Manually Formatted Source Files

The following attachments are those files which were manually edited, then used as part of our experiment. These files are originally from open source projects, each with their own licensing, but have had manual code formatting transformations applied to them in order to make them compatible with our experimental procedure. They retain their original licenses, but are reproduced here for the convenience of anyone who may need to inspect them in reference to our experiment.

E.1 Manually Transformed Sources for Task Chart-5

E.1.1 Defects-Sources/Chart-5/XYSeries-bugged-A.java

```
1  /* ===========================================================
2   * JFreeChart : a free chart library for the Java(tm) platform
3   *              ===========================================================
4   *
5   * (C) Copyright 2000-2008, by Object Refinery Limited and
6   * Contributors.
7   *
8   * Project Info:  http://www.jfree.org/jfreechart/index.html
9   *
10  * This library is free software; you can redistribute it and/or
11  * modify it under the terms of the GNU Lesser General Public License
```

```
12  * as published by the Free Software Foundation; either
        version 2.1 of
13  * the License, or (at your option) any later version.
14  *
15  * This library is distributed in the hope that it will be
        useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty
        of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
        the GNU
18  * Lesser General Public License for more details.
19  *
20  * You should have received a copy of the GNU Lesser
        General Public
21  * License along with this library; if not, write to the
        Free Software
22  * Foundation, Inc.,
23  * 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
        USA.
24  *
25  * [Java is a trademark or registered trademark of
26  * Sun Microsystems, Inc. in the United States and other
        countries.]
27  *
28  * -------------
29  * XYSeries.java
30  * -------------
31  * (C) Copyright 2001-2008, Object Refinery Limited and
        Contributors.
32  *
33  * Original Author:  David Gilbert (for Object Refinery
        Limited);
34  * Contributor(s):   Aaron Metzger;
35  *                   Jonathan Gabbai;
36  *                   Richard Atkinson;
37  *                   Michel Santos;
38  *                   Ted Schwartz (fix for bug 1955483);
39  *
40  * Changes
41  * -------
42  * 15-Nov-2001 : Version 1 (DG);
43  * 03-Apr-2002 : Added an add(double, double) method (DG);
44  * 29-Apr-2002 : Added a clear() method (ARM);
45  * 06-Jun-2002 : Updated Javadoc comments (DG);
46  * 29-Aug-2002 : Modified to give user control over whether
        or not
```

```
47  *                duplicate x-values are allowed (DG);
48  * 07-Oct-2002 : Fixed errors reported by Checkstyle (DG);
49  * 11-Nov-2002 : Added maximum item count, code contributed
        by
50  *                Jonathan Gabbai (DG);
51  * 26-Mar-2003 : Implemented Serializable (DG);
52  * 04-Aug-2003 : Added getItems() method (DG);
53  * 15-Aug-2003 : Changed 'data' from private to protected,
        added new
54  *                add() methods with a 'notify' argument (DG
        );
55  * 22-Sep-2003 : Added getAllowDuplicateXValues() method (
        RA);
56  * 29-Jan-2004 : Added autoSort attribute, based on a
        contribution by
57  *                Michel Santos - see patch 886740 (DG);
58  * 03-Feb-2004 : Added indexOf() method (DG);
59  * 16-Feb-2004 : Added remove() method (DG);
60  * 18-Aug-2004 : Moved from org.jfree.data --> org.jfree.
        data.xy (DG);
61  * 21-Feb-2005 : Added update(Number, Number) and
62  *                addOrUpdate(Number, Number) methods (DG);
63  * 03-May-2005 : Added a new constructor, fixed the
64  *                setMaximumItemCount() method to remove
        items (and
65  *                notify listeners) if necessary, fixed the
        add() and
66  *                addOrUpdate() methods to handle
        unsortedseries (DG);
67  * ------------- JFreeChart 1.0.x
        ------------------------------------
68  * 11-Jan-2005 : Renamed update(int, Number) -->
        updateByIndex() (DG);
69  * 15-Jan-2007 : Added toArray() method (DG);
70  * 20-Jun-2007 : Removed deprecated code and JCommon
        dependencies
71  *                (DG);
72  * 31-Oct-2007 : Implemented faster hashCode() (DG);
73  * 22-Nov-2007 : Reimplemented clone() (DG);
74  * 01-May-2008 : Fixed bug 1955483 in addOrUpdate() method,
        thanks to
75  *                Ted Schwartz (DG);
76  * 24-Nov-2008 : Further fix for 1955483 (DG);
77  *
78  */
79
```

```
80
81 package org.jfree.data.xy;
82
83
84 import java.io.Serializable;
85 import java.util.Collections;
86 import java.util.List;
87 import org.jfree.chart.util.ObjectUtilities;
88 import org.jfree.data.general.Series;
89 import org.jfree.data.general.SeriesChangeEvent;
90 import org.jfree.data.general.SeriesException;
91
92
93 /**
94  * Represents a sequence of zero or more data items in the
       form
95  * (x, y).  By default, items in the series will be sorted
       into
96  * ascending order by x-value, and duplicate x-values are
       permitted.
97  * Both the sorting and duplicate defaults can be changed
       in the
98  * constructor.  Y-values can be <code>null</code> to
       represent
99  * missing values.
100  */
101 public class XYSeries extends Series implements Cloneable,
       Serializable {
102
103     /** For serialization. */
104     static final long serialVersionUID =
           -5908509288197150436L;
105
106     // In version 0.9.12, in response to several developer
           requests, I changed
107     // the 'data' attribute from 'private' to 'protected',
           so that others can
108     // make subclasses that work directly with the
           underlying data structure.
109     /** Storage for the data items in the series. */
110     protected List data;
111
112     /** The maximum number of items for the series. */
113     private int maximumItemCount = Integer.MAX_VALUE;
114
115     /**
```

```
116         * A flag that controls whether the items are
                automatically
117         * sorted.
118         */
119        private boolean autoSort;
120
121        /**
122         * A flag that controls whether or not duplicate x-
                values are
123         * allowed.
124         */
125        private boolean allowDuplicateXValues;
126
127
128        /**
129         * Creates a new empty series.  By default, items added
                to the
130         * series will be sorted into ascending order by x-
                value, and
131         * duplicate x-values will be allowed (these defaults
                can be
132         * modified with another constructor.
133         *
134         * @param key  the series key (<code>null</code> not
                permitted).
135         */
136        public XYSeries(Comparable key) {
137            this(key, true, true);
138        }
139
140        /**
141         * Constructs a new empty series, with the auto-sort
                flag set as
142         * requested, and duplicate values allowed.
143         *
144         * @param key  the series key (<code>null</code> not
                permitted).
145         * @param autoSort  a flag that controls whether or not
                the items
146         *                  in the series are sorted.
147         */
148        public XYSeries(Comparable key, boolean autoSort) {
149            this(key, autoSort, true);
150        }
151
152        /**
```

```
153         * Constructs a new xy-series that contains no data.
               You can
154         * specify whether or not duplicate x-values are
               allowed for the
155         * series.
156         *
157         * @param key  the series key (<code>null</code> not
               permitted).
158         * @param autoSort  a flag that controls whether or not
                the items
159         *                  in the series are sorted.
160         * @param allowDuplicateXValues  a flag that controls
               whether
161         *                               duplicate x-values are
                allowed.
162         */
163        public XYSeries(Comparable key,
164                        boolean autoSort,
165                        boolean allowDuplicateXValues) {
166            super(key);
167            this.data = new java.util.ArrayList();
168            this.autoSort = autoSort;
169            this.allowDuplicateXValues = allowDuplicateXValues;
170        }
171
172
173        /**
174         * Returns the flag that controls whether the items in
               the series
175         * are automatically sorted.  There is no setter for
               this flag, it
176         * must be defined in the series constructor.
177         *
178         * @return A boolean.
179         */
180        public boolean getAutoSort() {
181            return this.autoSort;
182        }
183
184        /**
185         * Returns a flag that controls whether duplicate x-
               values are
186         * allowed. This flag can only be set in the
               constructor.
187         *
188         * @return A boolean.
```

```
189         */
190        public boolean getAllowDuplicateXValues() {
191            return this.allowDuplicateXValues;
192        }
193
194        /**
195         * Returns the number of items in the series.
196         *
197         * @return The item count.
198         */
199        public int getItemCount() {
200            return this.data.size();
201        }
202
203        /**
204         * Returns the list of data items for the series (the
                list
205         * contains {@link XYDataItem} objects and is
                unmodifiable).
206         *
207         * @return The list of data items.
208         */
209        public List getItems() {
210            return Collections.unmodifiableList(this.data);
211        }
212
213        /**
214         * Returns the maximum number of items that will be
                retained in
215         * the series. The default value is
216         * <code>Integer.MAX_VALUE</code>.
217         *
218         * @return The maximum item count.
219         * @see #setMaximumItemCount(int)
220         */
221        public int getMaximumItemCount() {
222            return this.maximumItemCount;
223        }
224
225        /**
226         * Sets the maximum number of items that will be
                retained in the
227         * series. If you add a new item to the series such
                that the
228         * number of items will exceed the maximum item count,
                then the
```

```
229         * first element in the series is automatically removed
                , ensuring
230         * that the maximum item count is not exceeded.
231         * <p>
232         * Typically this value is set before the series is
                populated with
233         * data, but if it is applied later, it may cause some
                items to be
234         * removed from the series (in which case a
235         * {@link SeriesChangeEvent} will be sent to all
                registered
236         * listeners.
237         *
238         * @param maximum  the maximum number of items for the
                series.
239         */
240        public void setMaximumItemCount(int maximum) {
241            this.maximumItemCount = maximum;
242            boolean dataRemoved = false;
243            while (this.data.size() > maximum) {
244                this.data.remove(0);
245                dataRemoved = true;
246            }
247            if (dataRemoved) {
248                fireSeriesChanged();
249            }
250        }
251
252        /**
253         * Adds a data item to the series and sends a
254         * {@link SeriesChangeEvent} to all registered
                listeners.
255         *
256         * @param item  the (x, y) item (<code>null</code> not
                permitted).
257         */
258        public void add(XYDataItem item) {
259
260            // argument checking delegated...
261            add(item, true);
262        }
263
264        /**
265         * Adds a data item to the series and sends a
266         * {@link SeriesChangeEvent} to all registered
                listeners.
```

```
267         *
268         * @param x  the x value.
269         * @param y  the y value.
270         */
271        public void add(double x, double y) {
272            add(new Double(x), new Double(y), true);
273        }
274
275        /**
276         * Adds a data item to the series and, if requested,
               sends a
277         * {@link SeriesChangeEvent} to all registered
               listeners.
278         *
279         * @param x  the x value.
280         * @param y  the y value.
281         * @param notify  a flag that controls whether or not a
282         *                {@link SeriesChangeEvent} is sent to
               all
283         *                registered listeners.
284         */
285        public void add(double x, double y, boolean notify) {
286            add(new Double(x), new Double(y), notify);
287        }
288
289        /**
290         * Adds a data item to the series and sends a
291         * {@link SeriesChangeEvent} to all registered
               listeners.  The
292         * unusual pairing of parameter types is to make it
               easier to add
293         * <code>null</code> y-values.
294         *
295         * @param x  the x value.
296         * @param y  the y value (<code>null</code> permitted).
297         */
298        public void add(double x, Number y) {
299            add(new Double(x), y);
300        }
301
302        /**
303         * Adds a data item to the series and, if requested,
               sends a
304         * {@link SeriesChangeEvent} to all registered
               listeners.  The
```

```
305         * unusual pairing of parameter types is to make it
                easier to add
306         * null y-values.
307         *
308         * @param x  the x value.
309         * @param y  the y value (<code>null</code> permitted).
310         * @param notify  a flag that controls whether or not a
311         *                {@link SeriesChangeEvent} is sent to
                all
312         *                registered listeners.
313         */
314        public void add(double x, Number y, boolean notify) {
315            add(new Double(x), y, notify);
316        }
317
318        /**
319         * Adds a new data item to the series (in the correct
                position if
320         * the <code>autoSort</code> flag is set for the series
                ) and sends
321         * a {@link SeriesChangeEvent} to all registered
                listeners.
322         * <P>
323         * Throws an exception if the x-value is a duplicate
                AND the
324         * allowDuplicateXValues flag is false.
325         *
326         * @param x  the x-value (<code>null</code> not
                permitted).
327         * @param y  the y-value (<code>null</code> permitted).
328         *
329         * @throws SeriesException if the x-value is a
                duplicate and the
330         *     <code>allowDuplicateXValues</code> flag is not
                set for this
331         *     series.
332         */
333        public void add(Number x, Number y) {
334
335            // argument checking delegated...
336            add(x, y, true);
337        }
338
339        /**
340         * Adds new data to the series and, if requested, sends
                a
```

```
341         * {@link SeriesChangeEvent} to all registered
                listeners.
342         * <P>
343         * Throws an exception if the x-value is a duplicate
                AND the
344         * allowDuplicateXValues flag is false.
345         *
346         * @param x  the x-value (<code>null</code> not
                permitted).
347         * @param y  the y-value (<code>null</code> permitted).
348         * @param notify  a flag the controls whether or not a
349         *                {@link SeriesChangeEvent} is sent to
                all
350         *                registered listeners.
351         */
352        public void add(Number x, Number y, boolean notify) {
353
354            // delegate argument checking to XYDataItem...
355            XYDataItem item = new XYDataItem(x, y);
356
357            add(item, notify);
358        }
359
360        /**
361         * Adds a data item to the series and, if requested,
                sends a
362         * {@link SeriesChangeEvent} to all registered
                listeners.
363         *
364         * @param item  the (x, y) item (<code>null</code> not
                permitted).
365         * @param notify  a flag that controls whether or not a
366         *                {@link SeriesChangeEvent} is sent to
                all
367         *                registered listeners.
368         */
369        public void add(XYDataItem item, boolean notify) {
370            if (item == null) {
371                throw new IllegalArgumentException("Null 'item'
                        argument.");
372            }
373
374            if (this.autoSort) {
375                int index = Collections.binarySearch(this.data,
                        item);
376                if (index < 0) {
```

```
377                 this.data.add(-index - 1, item);
378             } else {
379                 if (this.allowDuplicateXValues) {
380
381                     // need to make sure we are adding *
                        after* any duplicates
382                     int size = this.data.size();
383                     while (index < size
384                             && item.compareTo(this.data.get(
                                index)) == 0) {
385                         index++;
386                     }
387                     if (index < this.data.size()) {
388                         this.data.add(index, item);
389                     } else {
390                         this.data.add(item);
391                     }
392                 } else {
393                     throw new SeriesException("X-value
                        already exists.");
394                 }
395             }
396         } else {
397             if (!this.allowDuplicateXValues) {
398
399                 // can't allow duplicate values, so we need
                    to check whether
400                 // there is an item with the given x-value
                    already
401                 int index = indexOf(item.getX());
402                 if (index >= 0) {
403                     throw new SeriesException("X-value
                        already exists.");
404                 }
405             }
406             this.data.add(item);
407         }
408         if (getItemCount() > this.maximumItemCount) {
409             this.data.remove(0);
410         }
411         if (notify) {
412             fireSeriesChanged();
413         }
414     }
415
416     /**
```

```
417         * Deletes a range of items from the series and sends a
418         * {@link SeriesChangeEvent} to all registered
                listeners.
419         *
420         * @param start  the start index (zero-based).
421         * @param end  the end index (zero-based).
422         */
423        public void delete(int start, int end) {
424            for (int i = start; i <= end; i++) {
425                this.data.remove(start);
426            }
427            fireSeriesChanged();
428        }
429
430        /**
431         * Removes the item at the specified index and sends a
432         * {@link SeriesChangeEvent} to all registered
                listeners.
433         *
434         * @param index  the index.
435         *
436         * @return The item removed.
437         */
438        public XYDataItem remove(int index) {
439            XYDataItem result = (XYDataItem) this.data.remove(
                    index);
440
441            fireSeriesChanged();
442            return result;
443        }
444
445        /**
446         * Removes the item with the specified x-value and
                sends a
447         * {@link SeriesChangeEvent} to all registered
                listeners.
448         *
449         * @param x  the x-value.
450
451         * @return The item removed.
452         */
453        public XYDataItem remove(Number x) {
454            return remove(indexOf(x));
455        }
456
457        /**
```

```
458         * Removes all data items from the series.
459         */
460        public void clear() {
461            if (this.data.size() > 0) {
462                this.data.clear();
463                fireSeriesChanged();
464            }
465        }
466
467        /**
468         * Return the data item with the specified index.
469         *
470         * @param index  the index.
471         *
472         * @return The data item with the specified index.
473         */
474        public XYDataItem getDataItem(int index) {
475            return (XYDataItem) this.data.get(index);
476        }
477
478        /**
479         * Returns the x-value at the specified index.
480         *
481         * @param index  the index (zero-based).
482         *
483         * @return The x-value (never <code>null</code>).
484         */
485        public Number getX(int index) {
486            return getDataItem(index).getX();
487        }
488
489        /**
490         * Returns the y-value at the specified index.
491         *
492         * @param index  the index (zero-based).
493         *
494         * @return The y-value (possibly <code>null</code>).
495         */
496        public Number getY(int index) {
497            return getDataItem(index).getY();
498        }
499
500        /**
501         * Updates the value of an item in the series and sends
                a
```

```
502         * {@link SeriesChangeEvent} to all registered
                listeners.
503         *
504         * @param index  the item (zero based index).
505         * @param y  the new value (<code>null</code> permitted
                ).
506         *
507         * @since 1.0.1
508         */
509        public void updateByIndex(int index, Number y) {
510            XYDataItem item = getDataItem(index);
511
512            item.setY(y);
513            fireSeriesChanged();
514        }
515
516        /**
517         * Updates an item in the series.
518         *
519         * @param x  the x-value (<code>null</code> not
                permitted).
520         * @param y  the y-value (<code>null</code> permitted).
521         *
522         * @throws SeriesException if there is no existing item
                    with the
523         *         specified x-value.
524         */
525        public void update(Number x, Number y) {
526            int index = indexOf(x);
527
528            if (index < 0) {
529                throw new SeriesException("No observation for x
                        = " + x);
530            } else {
531                XYDataItem item = getDataItem(index);
532                item.setY(y);
533                fireSeriesChanged();
534            }
535        }
536
537        /**
538         * Adds or updates an item in the series and sends a
539         * {@link SeriesChangeEvent} to all registered
                listeners.
540         *
541         * @param x  the x-value.
```

```
542          * @param y  the y-value.
543          *
544          * @return The item that was overwritten, if any.
545          *
546          * @since 1.0.10
547          */
548         public XYDataItem addOrUpdate(double x, double y) {
549             return addOrUpdate(new Double(x), new Double(y));
550         }
551
552         /**
553          * Adds or updates an item in the series and sends a
554          * {@link SeriesChangeEvent} to all registered
                listeners.
555          *
556          * @param x  the x-value (<code>null</code> not
                permitted).
557          * @param y  the y-value (<code>null</code> permitted).
558          *
559          * @return A copy of the overwritten data item, or
560          *         <code>null</code> if noitem was overwritten.
561          */
562         public XYDataItem addOrUpdate(Number x, Number y) {
563             if (x == null) {
564                 throw new IllegalArgumentException("Null 'x'
                        argument.");
565             }
566
567             // if we get to here, we know that duplicate X
                    values are not permitted
568             XYDataItem overwritten = null;
569             int index = indexOf(x);
570             if (index >= 0 && !this.allowDuplicateXValues) {
571                 XYDataItem existing = (XYDataItem) this.data.
                        get(index);
572                 try {
573                     overwritten = (XYDataItem) existing.clone()
                            ;
574                 } catch (CloneNotSupportedException e) {
575                     throw new SeriesException("Couldn't clone
                            XYDataItem!");
576                 }
577                 existing.setY(y);
578             } else {
579
```

```
580             // if the series is sorted, the negative index
                    is a result from
581             // Collections.binarySearch() and tells us
                    where to insert the
582             // new item...otherwise it will be just -1 and
                    we should just
583             // append the value to the list...
584             if (this.autoSort) {
585                 this.data.add(-index - 1, new XYDataItem(x,
                        y));
586             } else {
587                 this.data.add(new XYDataItem(x, y));
588             }
589
590             // check if this addition will exceed the
                    maximum item count...
591             if (getItemCount() > this.maximumItemCount) {
592                 this.data.remove(0);
593             }
594         }
595         fireSeriesChanged();
596         return overwritten;
597     }
598
599     /**
600      * Returns the index of the item with the specified x-
                value, or a
601      * negative index if the series does not contain an
                item with that
602      * x-value.  Be aware that for an unsorted series, the
                index is
603      * found by iterating through all items in the series.
604      *
605      * @param x  the x-value (<code>null</code> not
                permitted).
606      *
607      * @return The index.
608      */
609     public int indexOf(Number x) {
610         if (this.autoSort) {
611             return Collections.binarySearch(this.data, new
                        XYDataItem(x, null));
612         } else {
613             for (int i = 0; i < this.data.size(); i++) {
614                 XYDataItem item = (XYDataItem) this.data.
                            get(i);
```

```
615                 if (item.getX().equals(x)) {
616                     return i;
617                 }
618             }
619             return -1;
620         }
621     }
622
623     /**
624      * Returns a new array containing the x and y values
           from this
625      * series.
626      *
627      * @return A new array containing the x and y values
           from this
628      * series.
629      *
630      * @since 1.0.4
631      */
632     public double[][] toArray() {
633         int itemCount = getItemCount();
634         double[][] result = new double[2][itemCount];
635
636         for (int i = 0; i < itemCount; i++) {
637             result[0][i] = this.getX(i).doubleValue();
638             Number y = getY(i);
639             if (y != null) {
640                 result[1][i] = y.doubleValue();
641             } else {
642                 result[1][i] = Double.NaN;
643             }
644         }
645         return result;
646     }
647
648     /**
649      * Returns a clone of the series.
650      *
651      * @return A clone of the series.
652      *
653      * @throws CloneNotSupportedException if there is a
           cloning
654      *                                    problem.
655      */
656     public Object clone() throws CloneNotSupportedException
           {
```

```
657         XYSeries clone = (XYSeries) super.clone();
658
659         clone.data = (List) ObjectUtilities.deepClone(this.
               data);
660         return clone;
661     }
662
663     /**
664      * Creates a new series by copying a subset of the data
665      * time series.
666      *
667      * @param start  the index of the first item to copy.
668      * @param end  the index of the last item to copy.
669      *
670      * @return A series containing a copy of this series
671      *         until end.
672      *
673      * @throws CloneNotSupportedException if there is a
674      *                                    problem.
675      */
676     public XYSeries createCopy(int start, int end)
677         throws CloneNotSupportedException {
678         XYSeries copy = (XYSeries) super.clone();
679
680         copy.data = new java.util.ArrayList();
681         if (this.data.size() > 0) {
682             for (int index = start; index <= end; index++)
                   {
683                 XYDataItem item = (XYDataItem) this.data.
                       get(index);
684                 XYDataItem clone = (XYDataItem) item.clone
                       ();
685                 try {
686                     copy.add(clone);
687                 }
688                 catch (SeriesException e) {
689                     System.err.println("Unable to add
                           cloned data item.");
690                 }
691             }
692         }
693         return copy;
694     }
```

```
695
696     /**
697      * Tests this series for equality with an arbitrary
            object.
698      *
699      * @param obj  the object to test against for equality
700      *             (<code>null</code> permitted).
701      *
702      * @return A boolean.
703      */
704     public boolean equals(Object obj) {
705         if (obj == this) {
706             return true;
707         }
708         if (!(obj instanceof XYSeries)) {
709             return false;
710         }
711         if (!super.equals(obj)) {
712             return false;
713         }
714         XYSeries that = (XYSeries) obj;
715         if (this.maximumItemCount != that.maximumItemCount)
                {
716             return false;
717         }
718         if (this.autoSort != that.autoSort) {
719             return false;
720         }
721         if (this.allowDuplicateXValues != that.
               allowDuplicateXValues) {
722             return false;
723         }
724         if (!ObjectUtilities.equal(this.data, that.data)) {
725             return false;
726         }
727         return true;
728     }
729
730     /**
731      * Returns a hash code.
732      *
733      * @return A hash code.
734      */
735     public int hashCode() {
736         int result = super.hashCode();
737
```

```
738             // it is too slow to look at every data item, so
                   let's just look at
739             // the first, middle and last items...
740             int count = getItemCount();
741
742             if (count > 0) {
743                 XYDataItem item = getDataItem(0);
744                 result = 29 * result + item.hashCode();
745             }
746             if (count > 1) {
747                 XYDataItem item = getDataItem(count - 1);
748                 result = 29 * result + item.hashCode();
749             }
750             if (count > 2) {
751                 XYDataItem item = getDataItem(count / 2);
752                 result = 29 * result + item.hashCode();
753             }
754             result = 29 * result + this.maximumItemCount;
755             result = 29 * result + (this.autoSort ? 1 : 0);
756             result = 29 * result + (this.allowDuplicateXValues
                   ? 1 : 0);
757             return result;
758         }
759 }
```

E.1.2 Defects-Sources/Chart-5/XYSeries-fixed-A.java

```
1  /*
     ===========================================================

2   * JFreeChart : a free chart library for the Java(tm)
       platform
3   *
       ===========================================================

4   *
5   * (C) Copyright 2000-2008, by Object Refinery Limited and
6   * Contributors.
7   *
8   * Project Info:  http://www.jfree.org/jfreechart/index.
       html
9   *
10  * This library is free software; you can redistribute it
       and/or
11  * modify it under the terms of the GNU Lesser General
       Public License
12  * as published by the Free Software Foundation; either
       version 2.1 of
13  * the License, or (at your option) any later version.
14  *
15  * This library is distributed in the hope that it will be
       useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty
       of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
         the GNU
18  * Lesser General Public License for more details.
19  *
20  * You should have received a copy of the GNU Lesser
       General Public
21  * License along with this library; if not, write to the
       Free Software
22  * Foundation, Inc.,
23  * 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
         USA.
24  *
25  * [Java is a trademark or registered trademark of
26  * Sun Microsystems, Inc. in the United States and other
       countries.]
27  *
28  * -------------
```

```
29 * XYSeries.java
30 * -------------
31 * (C) Copyright 2001-2008, Object Refinery Limited and
     Contributors.
32 *
33 * Original Author:  David Gilbert (for Object Refinery
     Limited);
34 * Contributor(s):   Aaron Metzger;
35 *                   Jonathan Gabbai;
36 *                   Richard Atkinson;
37 *                   Michel Santos;
38 *                   Ted Schwartz (fix for bug 1955483);
39 *
40 * Changes
41 * -------
42 * 15-Nov-2001 : Version 1 (DG);
43 * 03-Apr-2002 : Added an add(double, double) method (DG);
44 * 29-Apr-2002 : Added a clear() method (ARM);
45 * 06-Jun-2002 : Updated Javadoc comments (DG);
46 * 29-Aug-2002 : Modified to give user control over whether
       or not
47 *               duplicate x-values are allowed (DG);
48 * 07-Oct-2002 : Fixed errors reported by Checkstyle (DG);
49 * 11-Nov-2002 : Added maximum item count, code contributed
       by
50 *               Jonathan Gabbai (DG);
51 * 26-Mar-2003 : Implemented Serializable (DG);
52 * 04-Aug-2003 : Added getItems() method (DG);
53 * 15-Aug-2003 : Changed 'data' from private to protected,
     added new
54 *               add() methods with a 'notify' argument (DG
     );
55 * 22-Sep-2003 : Added getAllowDuplicateXValues() method (
     RA);
56 * 29-Jan-2004 : Added autoSort attribute, based on a
     contribution by
57 *               Michel Santos - see patch 886740 (DG);
58 * 03-Feb-2004 : Added indexOf() method (DG);
59 * 16-Feb-2004 : Added remove() method (DG);
60 * 18-Aug-2004 : Moved from org.jfree.data --> org.jfree.
     data.xy (DG);
61 * 21-Feb-2005 : Added update(Number, Number) and
62 *               addOrUpdate(Number, Number) methods (DG);
63 * 03-May-2005 : Added a new constructor, fixed the
64 *               setMaximumItemCount() method to remove
     items (and
```

```
65  *                notify listeners) if necessary, fixed the
       add() and
66  *                addOrUpdate() methods to handle
       unsortedseries (DG);
67  * ------------- JFreeChart 1.0.x
       ------------------------------------
68  * 11-Jan-2005 : Renamed update(int, Number) -->
       updateByIndex() (DG);
69  * 15-Jan-2007 : Added toArray() method (DG);
70  * 20-Jun-2007 : Removed deprecated code and JCommon
       dependencies
71  *                (DG);
72  * 31-Oct-2007 : Implemented faster hashCode() (DG);
73  * 22-Nov-2007 : Reimplemented clone() (DG);
74  * 01-May-2008 : Fixed bug 1955483 in addOrUpdate() method,
       thanks to
75  *                Ted Schwartz (DG);
76  * 24-Nov-2008 : Further fix for 1955483 (DG);
77  *
78  */
79
80
81 package org.jfree.data.xy;
82
83
84 import java.io.Serializable;
85 import java.util.Collections;
86 import java.util.List;
87 import org.jfree.chart.util.ObjectUtilities;
88 import org.jfree.data.general.Series;
89 import org.jfree.data.general.SeriesChangeEvent;
90 import org.jfree.data.general.SeriesException;
91
92
93 /**
94  * Represents a sequence of zero or more data items in the
       form
95  * (x, y).  By default, items in the series will be sorted
       into
96  * ascending order by x-value, and duplicate x-values are
       permitted.
97  * Both the sorting and duplicate defaults can be changed
       in the
98  * constructor.  Y-values can be <code>null</code> to
       represent
99  * missing values.
```

```
100  */
101 public class XYSeries extends Series implements Cloneable,
       Serializable {
102
103     /** For serialization. */
104     static final long serialVersionUID =
           -5908509288197150436L;
105
106     // In version 0.9.12, in response to several developer
           requests, I changed
107     // the 'data' attribute from 'private' to 'protected',
           so that others can
108     // make subclasses that work directly with the
           underlying data structure.
109     /** Storage for the data items in the series. */
110     protected List data;
111
112     /** The maximum number of items for the series. */
113     private int maximumItemCount = Integer.MAX_VALUE;
114
115     /**
116      * A flag that controls whether the items are
            automatically
117      * sorted.
118      */
119     private boolean autoSort;
120
121     /**
122      * A flag that controls whether or not duplicate x-
            values are
123      * allowed.
124      */
125     private boolean allowDuplicateXValues;
126
127
128     /**
129      * Creates a new empty series.  By default, items added
             to the
130      * series will be sorted into ascending order by x-
            value, and
131      * duplicate x-values will be allowed (these defaults
            can be
132      * modified with another constructor.
133      *
134      * @param key  the series key (<code>null</code> not
            permitted).
```

```
135         */
136        public XYSeries(Comparable key) {
137            this(key, true, true);
138        }
139
140        /**
141         * Constructs a new empty series, with the auto-sort
               flag set as
142         * requested, and duplicate values allowed.
143         *
144         * @param key  the series key (<code>null</code> not
               permitted).
145         * @param autoSort  a flag that controls whether or not
                the items
146         *                  in the series are sorted.
147         */
148        public XYSeries(Comparable key, boolean autoSort) {
149            this(key, autoSort, true);
150        }
151
152        /**
153         * Constructs a new xy-series that contains no data.
               You can
154         * specify whether or not duplicate x-values are
               allowed for the
155         * series.
156         *
157         * @param key  the series key (<code>null</code> not
               permitted).
158         * @param autoSort  a flag that controls whether or not
                the items
159         *                  in the series are sorted.
160         * @param allowDuplicateXValues  a flag that controls
               whether
161         *                               duplicate x-values are
                allowed.
162         */
163        public XYSeries(Comparable key,
164                        boolean autoSort,
165                        boolean allowDuplicateXValues) {
166            super(key);
167            this.data = new java.util.ArrayList();
168            this.autoSort = autoSort;
169            this.allowDuplicateXValues = allowDuplicateXValues;
170        }
171
```

```
172
173     /**
174      * Returns the flag that controls whether the items in
            the series
175      * are automatically sorted.  There is no setter for
            this flag, it
176      * must be defined in the series constructor.
177      *
178      * @return A boolean.
179      */
180     public boolean getAutoSort() {
181         return this.autoSort;
182     }
183
184     /**
185      * Returns a flag that controls whether duplicate x-
            values are
186      * allowed. This flag can only be set in the
            constructor.
187      *
188      * @return A boolean.
189      */
190     public boolean getAllowDuplicateXValues() {
191         return this.allowDuplicateXValues;
192     }
193
194     /**
195      * Returns the number of items in the series.
196      *
197      * @return The item count.
198      */
199     public int getItemCount() {
200         return this.data.size();
201     }
202
203     /**
204      * Returns the list of data items for the series (the
            list
205      * contains {@link XYDataItem} objects and is
            unmodifiable).
206      *
207      * @return The list of data items.
208      */
209     public List getItems() {
210         return Collections.unmodifiableList(this.data);
211     }
```

```
212
213     /**
214      * Returns the maximum number of items that will be
             retained in
215      * the series. The default value is
216      * <code>Integer.MAX_VALUE</code>.
217      *
218      * @return The maximum item count.
219      * @see #setMaximumItemCount(int)
220      */
221     public int getMaximumItemCount() {
222         return this.maximumItemCount;
223     }
224
225     /**
226      * Sets the maximum number of items that will be
             retained in the
227      * series. If you add a new item to the series such
             that the
228      * number of items will exceed the maximum item count,
             then the
229      * first element in the series is automatically removed
             , ensuring
230      * that the maximum item count is not exceeded.
231      * <p>
232      * Typically this value is set before the series is
             populated with
233      * data, but if it is applied later, it may cause some
             items to be
234      * removed from the series (in which case a
235      * {@link SeriesChangeEvent} will be sent to all
             registered
236      * listeners.
237      *
238      * @param maximum  the maximum number of items for the
             series.
239      */
240     public void setMaximumItemCount(int maximum) {
241         this.maximumItemCount = maximum;
242         boolean dataRemoved = false;
243         while (this.data.size() > maximum) {
244             this.data.remove(0);
245             dataRemoved = true;
246         }
247         if (dataRemoved) {
248             fireSeriesChanged();
```

```
249         }
250     }
251
252     /**
253      * Adds a data item to the series and sends a
254      * {@link SeriesChangeEvent} to all registered
            listeners.
255      *
256      * @param item  the (x, y) item (<code>null</code> not
            permitted).
257      */
258     public void add(XYDataItem item) {
259
260         // argument checking delegated...
261         add(item, true);
262     }
263
264     /**
265      * Adds a data item to the series and sends a
266      * {@link SeriesChangeEvent} to all registered
            listeners.
267      *
268      * @param x  the x value.
269      * @param y  the y value.
270      */
271     public void add(double x, double y) {
272         add(new Double(x), new Double(y), true);
273     }
274
275     /**
276      * Adds a data item to the series and, if requested,
            sends a
277      * {@link SeriesChangeEvent} to all registered
            listeners.
278      *
279      * @param x  the x value.
280      * @param y  the y value.
281      * @param notify  a flag that controls whether or not a
282      *                {@link SeriesChangeEvent} is sent to
            all
283      *                registered listeners.
284      */
285     public void add(double x, double y, boolean notify) {
286         add(new Double(x), new Double(y), notify);
287     }
288
```

```
289     /**
290      * Adds a data item to the series and sends a
291      * {@link SeriesChangeEvent} to all registered
            listeners.  The
292      * unusual pairing of parameter types is to make it
            easier to add
293      * <code>null</code> y-values.
294      *
295      * @param x  the x value.
296      * @param y  the y value (<code>null</code> permitted).
297      */
298     public void add(double x, Number y) {
299         add(new Double(x), y);
300     }
301
302     /**
303      * Adds a data item to the series and, if requested,
            sends a
304      * {@link SeriesChangeEvent} to all registered
            listeners.  The
305      * unusual pairing of parameter types is to make it
            easier to add
306      * null y-values.
307      *
308      * @param x  the x value.
309      * @param y  the y value (<code>null</code> permitted).
310      * @param notify  a flag that controls whether or not a
311      *                {@link SeriesChangeEvent} is sent to
            all
312      *                registered listeners.
313      */
314     public void add(double x, Number y, boolean notify) {
315         add(new Double(x), y, notify);
316     }
317
318     /**
319      * Adds a new data item to the series (in the correct
            position if
320      * the <code>autoSort</code> flag is set for the series
            ) and sends
321      * a {@link SeriesChangeEvent} to all registered
            listeners.
322      * <P>
323      * Throws an exception if the x-value is a duplicate
            AND the
324      * allowDuplicateXValues flag is false.
```

```
325          *
326          * @param x  the x-value (<code>null</code> not
                permitted).
327          * @param y  the y-value (<code>null</code> permitted).
328          *
329          * @throws SeriesException if the x-value is a
                duplicate and the
330          *     <code>allowDuplicateXValues</code> flag is not
                set for this
331          *     series.
332          */
333         public void add(Number x, Number y) {
334
335             // argument checking delegated...
336             add(x, y, true);
337         }
338
339         /**
340          * Adds new data to the series and, if requested, sends
                 a
341          * {@link SeriesChangeEvent} to all registered
                listeners.
342          * <P>
343          * Throws an exception if the x-value is a duplicate
                AND the
344          * allowDuplicateXValues flag is false.
345          *
346          * @param x  the x-value (<code>null</code> not
                permitted).
347          * @param y  the y-value (<code>null</code> permitted).
348          * @param notify  a flag the controls whether or not a
349          *                {@link SeriesChangeEvent} is sent to
                all
350          *                registered listeners.
351          */
352         public void add(Number x, Number y, boolean notify) {
353
354             // delegate argument checking to XYDataItem...
355             XYDataItem item = new XYDataItem(x, y);
356
357             add(item, notify);
358         }
359
360         /**
361          * Adds a data item to the series and, if requested,
                sends a
```

```
362         * {@link SeriesChangeEvent} to all registered
                listeners.
363         *
364         * @param item  the (x, y) item (<code>null</code> not
                permitted).
365         * @param notify  a flag that controls whether or not a
366         *                {@link SeriesChangeEvent} is sent to
                all
367         *                registered listeners.
368         */
369        public void add(XYDataItem item, boolean notify) {
370            if (item == null) {
371                throw new IllegalArgumentException("Null 'item'
                        argument.");
372            }
373
374            if (this.autoSort) {
375                int index = Collections.binarySearch(this.data,
                        item);
376                if (index < 0) {
377                    this.data.add(-index - 1, item);
378                } else {
379                    if (this.allowDuplicateXValues) {
380
381                        // need to make sure we are adding *
                            after* any duplicates
382                        int size = this.data.size();
383                        while (index < size
384                                && item.compareTo(this.data.get(
                                    index)) == 0) {
385                            index++;
386                        }
387                        if (index < this.data.size()) {
388                            this.data.add(index, item);
389                        } else {
390                            this.data.add(item);
391                        }
392                    } else {
393                        throw new SeriesException("X-value
                            already exists.");
394                    }
395                }
396            } else {
397                if (!this.allowDuplicateXValues) {
398
399                    // can't allow duplicate values, so we need
```

```
                    to check whether
400                 // there is an item with the given x-value
                       already
401                 int index = indexOf(item.getX());
402                 if (index >= 0) {
403                     throw new SeriesException("X-value
                           already exists.");
404                 }
405             }
406             this.data.add(item);
407         }
408         if (getItemCount() > this.maximumItemCount) {
409             this.data.remove(0);
410         }
411         if (notify) {
412             fireSeriesChanged();
413         }
414     }
415
416     /**
417      * Deletes a range of items from the series and sends a
418      * {@link SeriesChangeEvent} to all registered
           listeners.
419      *
420      * @param start  the start index (zero-based).
421      * @param end  the end index (zero-based).
422      */
423     public void delete(int start, int end) {
424         for (int i = start; i <= end; i++) {
425             this.data.remove(start);
426         }
427         fireSeriesChanged();
428     }
429
430     /**
431      * Removes the item at the specified index and sends a
432      * {@link SeriesChangeEvent} to all registered
           listeners.
433      *
434      * @param index  the index.
435      *
436      * @return The item removed.
437      */
438     public XYDataItem remove(int index) {
439         XYDataItem result = (XYDataItem) this.data.remove(
               index);
```

```
440
441             fireSeriesChanged();
442             return result;
443         }
444
445         /**
446          * Removes the item with the specified x-value and
                sends a
447          * {@link SeriesChangeEvent} to all registered
                listeners.
448          *
449          * @param x  the x-value.
450
451          * @return The item removed.
452          */
453         public XYDataItem remove(Number x) {
454             return remove(indexOf(x));
455         }
456
457         /**
458          * Removes all data items from the series.
459          */
460         public void clear() {
461             if (this.data.size() > 0) {
462                 this.data.clear();
463                 fireSeriesChanged();
464             }
465         }
466
467         /**
468          * Return the data item with the specified index.
469          *
470          * @param index  the index.
471          *
472          * @return The data item with the specified index.
473          */
474         public XYDataItem getDataItem(int index) {
475             return (XYDataItem) this.data.get(index);
476         }
477
478         /**
479          * Returns the x-value at the specified index.
480          *
481          * @param index  the index (zero-based).
482          *
483          * @return The x-value (never <code>null</code>).
```

```
484         */
485        public Number getX(int index) {
486            return getDataItem(index).getX();
487        }
488
489        /**
490         * Returns the y-value at the specified index.
491         *
492         * @param index  the index (zero-based).
493         *
494         * @return The y-value (possibly <code>null</code>).
495         */
496        public Number getY(int index) {
497            return getDataItem(index).getY();
498        }
499
500        /**
501         * Updates the value of an item in the series and sends
                a
502         * {@link SeriesChangeEvent} to all registered
                listeners.
503         *
504         * @param index  the item (zero based index).
505         * @param y  the new value (<code>null</code> permitted
                ).
506         *
507         * @since 1.0.1
508         */
509        public void updateByIndex(int index, Number y) {
510            XYDataItem item = getDataItem(index);
511
512            item.setY(y);
513            fireSeriesChanged();
514        }
515
516        /**
517         * Updates an item in the series.
518         *
519         * @param x  the x-value (<code>null</code> not
                permitted).
520         * @param y  the y-value (<code>null</code> permitted).
521         *
522         * @throws SeriesException if there is no existing item
                  with the
523         *         specified x-value.
524         */
```

```
525     public void update(Number x, Number y) {
526         int index = indexOf(x);
527
528         if (index < 0) {
529             throw new SeriesException("No observation for x
                    = " + x);
530         } else {
531             XYDataItem item = getDataItem(index);
532             item.setY(y);
533             fireSeriesChanged();
534         }
535     }
536
537     /**
538      * Adds or updates an item in the series and sends a
539      * {@link SeriesChangeEvent} to all registered
           listeners.
540      *
541      * @param x  the x-value.
542      * @param y  the y-value.
543      *
544      * @return The item that was overwritten, if any.
545      *
546      * @since 1.0.10
547      */
548     public XYDataItem addOrUpdate(double x, double y) {
549         return addOrUpdate(new Double(x), new Double(y));
550     }
551
552     /**
553      * Adds or updates an item in the series and sends a
554      * {@link SeriesChangeEvent} to all registered
           listeners.
555      *
556      * @param x  the x-value (<code>null</code> not
           permitted).
557      * @param y  the y-value (<code>null</code> permitted).
558      *
559      * @return A copy of the overwritten data item, or
560      *         <code>null</code> if noitem was overwritten.
561      */
562     public XYDataItem addOrUpdate(Number x, Number y) {
563         if (x == null) {
564             throw new IllegalArgumentException("Null 'x'
                    argument.");
565         }
```

```
566         if (this.allowDuplicateXValues) {
567             add(x, y);
568             return null;
569         }
570 
571         // if we get to here, we know that duplicate X
                values are not permitted
572         XYDataItem overwritten = null;
573         int index = indexOf(x);
574         if (index >= 0) {
575             XYDataItem existing = (XYDataItem) this.data.
                    get(index);
576             try {
577                 overwritten = (XYDataItem) existing.clone()
                        ;
578             } catch (CloneNotSupportedException e) {
579                 throw new SeriesException("Couldn't clone
                        XYDataItem!");
580             }
581             existing.setY(y);
582         } else {
583 
584             // if the series is sorted, the negative index
                    is a result from
585             // Collections.binarySearch() and tells us
                    where to insert the
586             // new item...otherwise it will be just -1 and
                    we should just
587             // append the value to the list...
588             if (this.autoSort) {
589                 this.data.add(-index - 1, new XYDataItem(x,
                        y));
590             } else {
591                 this.data.add(new XYDataItem(x, y));
592             }
593 
594             // check if this addition will exceed the
                    maximum item count...
595             if (getItemCount() > this.maximumItemCount) {
596                 this.data.remove(0);
597             }
598         }
599         fireSeriesChanged();
600         return overwritten;
601     }
602 
```

```
603     /**
604      * Returns the index of the item with the specified x-
            value, or a
605      * negative index if the series does not contain an
            item with that
606      * x-value.  Be aware that for an unsorted series, the
            index is
607      * found by iterating through all items in the series.
608      *
609      * @param x  the x-value (<code>null</code> not
            permitted).
610      *
611      * @return The index.
612      */
613     public int indexOf(Number x) {
614         if (this.autoSort) {
615             return Collections.binarySearch(this.data, new
                XYDataItem(x, null));
616         } else {
617             for (int i = 0; i < this.data.size(); i++) {
618                 XYDataItem item = (XYDataItem) this.data.
                    get(i);
619                 if (item.getX().equals(x)) {
620                     return i;
621                 }
622             }
623             return -1;
624         }
625     }
626
627     /**
628      * Returns a new array containing the x and y values
            from this
629      * series.
630      *
631      * @return A new array containing the x and y values
            from this
632      * series.
633      *
634      * @since 1.0.4
635      */
636     public double[][] toArray() {
637         int itemCount = getItemCount();
638         double[][] result = new double[2][itemCount];
639
640         for (int i = 0; i < itemCount; i++) {
```

```
641             result[0][i] = this.getX(i).doubleValue();
642             Number y = getY(i);
643             if (y != null) {
644                 result[1][i] = y.doubleValue();
645             } else {
646                 result[1][i] = Double.NaN;
647             }
648         }
649         return result;
650     }
651 
652     /**
653      * Returns a clone of the series.
654      *
655      * @return A clone of the series.
656      *
657      * @throws CloneNotSupportedException if there is a
           cloning
658      *                                    problem.
659      */
660     public Object clone() throws CloneNotSupportedException
           {
661         XYSeries clone = (XYSeries) super.clone();
662 
663         clone.data = (List) ObjectUtilities.deepClone(this.
              data);
664         return clone;
665     }
666 
667     /**
668      * Creates a new series by copying a subset of the data
           in this
669      * time series.
670      *
671      * @param start  the index of the first item to copy.
672      * @param end  the index of the last item to copy.
673      *
674      * @return A series containing a copy of this series
           from start
675      *         until end.
676      *
677      * @throws CloneNotSupportedException if there is a
           cloning
678      *                                    problem.
679      */
680     public XYSeries createCopy(int start, int end)
```

```
681         throws CloneNotSupportedException {
682         XYSeries copy = (XYSeries) super.clone();
683 
684         copy.data = new java.util.ArrayList();
685         if (this.data.size() > 0) {
686             for (int index = start; index <= end; index++)
                    {
687                 XYDataItem item = (XYDataItem) this.data.
                        get(index);
688                 XYDataItem clone = (XYDataItem) item.clone
                        ();
689                 try {
690                     copy.add(clone);
691                 }
692                 catch (SeriesException e) {
693                     System.err.println("Unable to add
                            cloned data item.");
694                 }
695             }
696         }
697         return copy;
698     }
699 
700     /**
701      * Tests this series for equality with an arbitrary
            object.
702      *
703      * @param obj  the object to test against for equality
704      *             (<code>null</code> permitted).
705      *
706      * @return A boolean.
707      */
708     public boolean equals(Object obj) {
709         if (obj == this) {
710             return true;
711         }
712         if (!(obj instanceof XYSeries)) {
713             return false;
714         }
715         if (!super.equals(obj)) {
716             return false;
717         }
718         XYSeries that = (XYSeries) obj;
719         if (this.maximumItemCount != that.maximumItemCount)
                {
720             return false;
```

```
721         }
722         if (this.autoSort != that.autoSort) {
723             return false;
724         }
725         if (this.allowDuplicateXValues != that.
               allowDuplicateXValues) {
726             return false;
727         }
728         if (!ObjectUtilities.equal(this.data, that.data)) {
729             return false;
730         }
731         return true;
732     }
733 
734     /**
735      * Returns a hash code.
736      *
737      * @return A hash code.
738      */
739     public int hashCode() {
740         int result = super.hashCode();
741 
742         // it is too slow to look at every data item, so
               let's just look at
743         // the first, middle and last items...
744         int count = getItemCount();
745 
746         if (count > 0) {
747             XYDataItem item = getDataItem(0);
748             result = 29 * result + item.hashCode();
749         }
750         if (count > 1) {
751             XYDataItem item = getDataItem(count - 1);
752             result = 29 * result + item.hashCode();
753         }
754         if (count > 2) {
755             XYDataItem item = getDataItem(count / 2);
756             result = 29 * result + item.hashCode();
757         }
758         result = 29 * result + this.maximumItemCount;
759         result = 29 * result + (this.autoSort ? 1 : 0);
760         result = 29 * result + (this.allowDuplicateXValues
               ? 1 : 0);
761         return result;
762     }
763 }
```

E.1.3 Defects-Sources/Chart-5/XYSeries-bugged-B.java

```
1  /*
     ===========================================================

2   * JFreeChart : a free chart library for the Java(tm)
       platform
3   *
       ===========================================================

4   *
5   * (C) Copyright 2000-2008, by Object Refinery Limited and
       Contributors.
6   *
7   * Project Info:  http://www.jfree.org/jfreechart/index.
       html
8   *
9   * This library is free software; you can redistribute it
       and/or modify it
10  * under the terms of the GNU Lesser General Public License
        as published by
11  * the Free Software Foundation; either version 2.1 of the
       License, or
12  * (at your option) any later version.
13  *
14  * This library is distributed in the hope that it will be
       useful, but
15  * WITHOUT ANY WARRANTY; without even the implied warranty
       of MERCHANTABILITY
16  * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser
       General Public
17  * License for more details.
18  *
19  * You should have received a copy of the GNU Lesser
       General Public
20  * License along with this library; if not, write to the
       Free Software
21  * Foundation, Inc., 51 Franklin Street, Fifth Floor,
       Boston, MA  02110-1301,
22  * USA.
23  *
24  * [Java is a trademark or registered trademark of Sun
       Microsystems, Inc.
25  * in the United States and other countries.]
26  *
27  * -------------
```

```
28  * XYSeries.java
29  * -------------
30  * (C) Copyright 2001-2008, Object Refinery Limited and
        Contributors.
31  *
32  * Original Author:  David Gilbert (for Object Refinery
        Limited);
33  * Contributor(s):   Aaron Metzger;
34  *                   Jonathan Gabbai;
35  *                   Richard Atkinson;
36  *                   Michel Santos;
37  *                   Ted Schwartz (fix for bug 1955483);
38  *
39  * Changes
40  * -------
41  * 15-Nov-2001 : Version 1 (DG);
42  * 03-Apr-2002 : Added an add(double, double) method (DG);
43  * 29-Apr-2002 : Added a clear() method (ARM);
44  * 06-Jun-2002 : Updated Javadoc comments (DG);
45  * 29-Aug-2002 : Modified to give user control over whether
        or not duplicate
46  *               x-values are allowed (DG);
47  * 07-Oct-2002 : Fixed errors reported by Checkstyle (DG);
48  * 11-Nov-2002 : Added maximum item count, code contributed
        by Jonathan
49  *               Gabbai (DG);
50  * 26-Mar-2003 : Implemented Serializable (DG);
51  * 04-Aug-2003 : Added getItems() method (DG);
52  * 15-Aug-2003 : Changed 'data' from private to protected,
        added new add()
53  *               methods with a 'notify' argument (DG);
54  * 22-Sep-2003 : Added getAllowDuplicateXValues() method (
        RA);
55  * 29-Jan-2004 : Added autoSort attribute, based on a
        contribution by
56  *               Michel Santos - see patch 886740 (DG);
57  * 03-Feb-2004 : Added indexOf() method (DG);
58  * 16-Feb-2004 : Added remove() method (DG);
59  * 18-Aug-2004 : Moved from org.jfree.data --> org.jfree.
        data.xy (DG);
60  * 21-Feb-2005 : Added update(Number, Number) and
        addOrUpdate(Number, Number)
61  *               methods (DG);
62  * 03-May-2005 : Added a new constructor, fixed the
        setMaximumItemCount()
63  *               method to remove items (and notify
```

```
    listeners) if necessary,
64  *                  fixed the add() and addOrUpdate() methods
    to handle unsorted
65  *                  series (DG);
66  * -------------- JFreeChart 1.0.x
    ---------------------------------------------
67  * 11-Jan-2005 : Renamed update(int, Number) -->
    updateByIndex() (DG);
68  * 15-Jan-2007 : Added toArray() method (DG);
69  * 20-Jun-2007 : Removed deprecated code and JCommon
    dependencies (DG);
70  * 31-Oct-2007 : Implemented faster hashCode() (DG);
71  * 22-Nov-2007 : Reimplemented clone() (DG);
72  * 01-May-2008 : Fixed bug 1955483 in addOrUpdate() method,
      thanks to
73  *                  Ted Schwartz (DG);
74  * 24-Nov-2008 : Further fix for 1955483 (DG);
75  *
76  */
77
78 package org.jfree.data.xy;
79
80 import java.io.Serializable;
81 import java.util.Collections;
82 import java.util.List;
83
84 import org.jfree.chart.util.ObjectUtilities;
85 import org.jfree.data.general.Series;
86 import org.jfree.data.general.SeriesChangeEvent;
87 import org.jfree.data.general.SeriesException;
88
89 /**
90  * Represents a sequence of zero or more data items in the
    form (x, y).  By
91  * default, items in the series will be sorted into
    ascending order by x-value,
92  * and duplicate x-values are permitted.  Both the sorting
    and duplicate
93  * defaults can be changed in the constructor.  Y-values
    can be
94  * <code>null</code> to represent missing values.
95  */
96 public class XYSeries extends Series implements Cloneable,
    Serializable
97 {
98
```

```
99      /** For serialization. */
100     static final long serialVersionUID =
           -5908509288197150436L;
101
102     // In version 0.9.12, in response to several developer
           requests, I changed
103     // the 'data' attribute from 'private' to 'protected',
           so that others can
104     // make subclasses that work directly with the
           underlying data structure.
105
106     /** Storage for the data items in the series. */
107     protected List data;
108
109     /** The maximum number of items for the series. */
110     private int maximumItemCount = Integer.MAX_VALUE;
111
112     /** A flag that controls whether the items are
           automatically sorted. */
113     private boolean autoSort;
114
115     /** A flag that controls whether or not duplicate x-
           values are allowed. */
116     private boolean allowDuplicateXValues;
117
118     /**
119      * Creates a new empty series.  By default, items added
             to the series will
120      * be sorted into ascending order by x-value, and
           duplicate x-values will
121      * be allowed (these defaults can be modified with
           another constructor.
122      *
123      * @param key  the series key (<code>null</code> not
           permitted).
124      */
125     public XYSeries(Comparable key)
126     {
127         this(key, true, true);
128     }
129
130     /**
131      * Constructs a new empty series, with the auto-sort
           flag set as requested,
132      * and duplicate values allowed.
133      *
```

```
134         * @param key  the series key (<code>null</code> not
                permitted).
135         * @param autoSort  a flag that controls whether or not
                the items in the
136         *                  series are sorted.
137         */
138        public XYSeries(Comparable key, boolean autoSort)
139        {
140            this(key, autoSort, true);
141        }
142
143        /**
144         * Constructs a new xy-series that contains no data.
                You can specify
145         * whether or not duplicate x-values are allowed for
                the series.
146         *
147         * @param key  the series key (<code>null</code> not
                permitted).
148         * @param autoSort  a flag that controls whether or not
                the items in the
149         *                  series are sorted.
150         * @param allowDuplicateXValues  a flag that controls
                whether duplicate
151         *                               x-values are allowed.
152         */
153        public XYSeries(Comparable key,
154                        boolean autoSort,
155                        boolean allowDuplicateXValues)
156        {
157            super(key);
158            this.data = new java.util.ArrayList();
159            this.autoSort = autoSort;
160            this.allowDuplicateXValues = allowDuplicateXValues;
161        }
162
163        /**
164         * Returns the flag that controls whether the items in
                the series are
165         * automatically sorted.  There is no setter for this
                flag, it must be
166         * defined in the series constructor.
167         *
168         * @return A boolean.
169         */
170        public boolean getAutoSort()
```

```
171     {
172         return this.autoSort;
173     }
174
175     /**
176      * Returns a flag that controls whether duplicate x-
            values are allowed.
177      * This flag can only be set in the constructor.
178      *
179      * @return A boolean.
180      */
181     public boolean getAllowDuplicateXValues()
182     {
183         return this.allowDuplicateXValues;
184     }
185
186     /**
187      * Returns the number of items in the series.
188      *
189      * @return The item count.
190      */
191     public int getItemCount()
192     {
193         return this.data.size();
194     }
195
196     /**
197      * Returns the list of data items for the series (the
            list contains
198      * {@link XYDataItem} objects and is unmodifiable).
199      *
200      * @return The list of data items.
201      */
202     public List getItems()
203     {
204         return Collections.unmodifiableList(this.data);
205     }
206
207     /**
208      * Returns the maximum number of items that will be
            retained in the series.
209      * The default value is <code>Integer.MAX_VALUE</code>.
210      *
211      * @return The maximum item count.
212      * @see #setMaximumItemCount(int)
213      */
```

```
214     public int getMaximumItemCount()
215     {
216         return this.maximumItemCount;
217     }
218
219     /**
220      * Sets the maximum number of items that will be
           retained in the series.
221      * If you add a new item to the series such that the
           number of items will
222      * exceed the maximum item count, then the first
           element in the series is
223      * automatically removed, ensuring that the maximum
           item count is not
224      * exceeded.
225      * <p>
226      * Typically this value is set before the series is
           populated with data,
227      * but if it is applied later, it may cause some items
           to be removed from
228      * the series (in which case a {@link SeriesChangeEvent
           } will be sent to
229      * all registered listeners.
230      *
231      * @param maximum  the maximum number of items for the
           series.
232      */
233     public void setMaximumItemCount(int maximum)
234     {
235         this.maximumItemCount = maximum;
236         boolean dataRemoved = false;
237         while (this.data.size() > maximum)
238         {
239             this.data.remove(0);
240             dataRemoved = true;
241         }
242         if (dataRemoved)
243         {
244             fireSeriesChanged();
245         }
246     }
247
248     /**
249      * Adds a data item to the series and sends a {@link
           SeriesChangeEvent} to
250      * all registered listeners.
```

```
251         *
252         * @param item  the (x, y) item (<code>null</code> not
                permitted).
253         */
254        public void add(XYDataItem item)
255        {
256            // argument checking delegated...
257            add(item, true);
258        }
259
260        /**
261         * Adds a data item to the series and sends a {@link
                SeriesChangeEvent} to
262         * all registered listeners.
263         *
264         * @param x  the x value.
265         * @param y  the y value.
266         */
267        public void add(double x, double y)
268        {
269            add(new Double(x), new Double(y), true);
270        }
271
272        /**
273         * Adds a data item to the series and, if requested,
                sends a
274         * {@link SeriesChangeEvent} to all registered
                listeners.
275         *
276         * @param x  the x value.
277         * @param y  the y value.
278         * @param notify  a flag that controls whether or not a
279         *                {@link SeriesChangeEvent} is sent to
                all registered
280         *                listeners.
281         */
282        public void add(double x, double y, boolean notify)
283        {
284            add(new Double(x), new Double(y), notify);
285        }
286
287        /**
288         * Adds a data item to the series and sends a {@link
                SeriesChangeEvent} to
289         * all registered listeners.  The unusual pairing of
                parameter types is to
```

```
290         * make it easier to add <code>null</code> y-values.
291         *
292         * @param x  the x value.
293         * @param y  the y value (<code>null</code> permitted).
294         */
295        public void add(double x, Number y)
296        {
297            add(new Double(x), y);
298        }
299
300        /**
301         * Adds a data item to the series and, if requested,
               sends a
302         * {@link SeriesChangeEvent} to all registered
               listeners.  The unusual
303         * pairing of parameter types is to make it easier to
               add null y-values.
304         *
305         * @param x  the x value.
306         * @param y  the y value (<code>null</code> permitted).
307         * @param notify  a flag that controls whether or not a
308         *                {@link SeriesChangeEvent} is sent to
               all registered
309         *                listeners.
310         */
311        public void add(double x, Number y, boolean notify)
312        {
313            add(new Double(x), y, notify);
314        }
315
316        /**
317         * Adds a new data item to the series (in the correct
               position if the
318         * <code>autoSort</code> flag is set for the series)
               and sends a
319         * {@link SeriesChangeEvent} to all registered
               listeners.
320         * <P>
321         * Throws an exception if the x-value is a duplicate
               AND the
322         * allowDuplicateXValues flag is false.
323         *
324         * @param x  the x-value (<code>null</code> not
               permitted).
325         * @param y  the y-value (<code>null</code> permitted).
326         *
```

```
327         * @throws SeriesException if the x-value is a
                duplicate and the
328         *     <code>allowDuplicateXValues</code> flag is not
                set for this series.
329         */
330        public void add(Number x, Number y)
331        {
332            // argument checking delegated...
333            add(x, y, true);
334        }
335
336        /**
337         * Adds new data to the series and, if requested, sends
                a
338         * {@link SeriesChangeEvent} to all registered
                listeners.
339         * <P>
340         * Throws an exception if the x-value is a duplicate
                AND the
341         * allowDuplicateXValues flag is false.
342         *
343         * @param x  the x-value (<code>null</code> not
                permitted).
344         * @param y  the y-value (<code>null</code> permitted).
345         * @param notify  a flag the controls whether or not a
346         *                {@link SeriesChangeEvent} is sent to
                all registered
347         *                listeners.
348         */
349        public void add(Number x, Number y, boolean notify)
350        {
351            // delegate argument checking to XYDataItem...
352            XYDataItem item = new XYDataItem(x, y);
353            add(item, notify);
354        }
355
356        /**
357         * Adds a data item to the series and, if requested,
                sends a
358         * {@link SeriesChangeEvent} to all registered
                listeners.
359         *
360         * @param item  the (x, y) item (<code>null</code> not
                permitted).
361         * @param notify  a flag that controls whether or not a
362         *                {@link SeriesChangeEvent} is sent to
```

```
         all registered
363       *                listeners.
364       */
365      public void add(XYDataItem item, boolean notify)
366      {
367
368          if (item == null)
369          {
370              throw new IllegalArgumentException("Null 'item'
                     argument.");
371          }
372
373          if (this.autoSort)
374          {
375              int index = Collections.binarySearch(this.data,
                     item);
376              if (index < 0)
377              {
378                  this.data.add(-index - 1, item);
379              }
380              else
381              {
382                  if (this.allowDuplicateXValues)
383                  {
384                      // need to make sure we are adding *
                             after* any duplicates
385                      int size = this.data.size();
386                      while (index < size
387                             && item.compareTo(this.data.get(
                                index)) == 0)
388                      {
389                          index++;
390                      }
391                      if (index < this.data.size())
392                      {
393                          this.data.add(index, item);
394                      }
395                      else
396                      {
397                          this.data.add(item);
398                      }
399                  }
400                  else
401                  {
402                      throw new SeriesException("X-value
                             already exists.");
```

```
403                 }
404             }
405         }
406         else
407         {
408             if (!this.allowDuplicateXValues)
409             {
410                 // can't allow duplicate values, so we need
                        to check whether
411                 // there is an item with the given x-value
                        already
412                 int index = indexOf(item.getX());
413                 if (index >= 0)
414                 {
415                     throw new SeriesException("X-value
                            already exists.");
416                 }
417             }
418             this.data.add(item);
419         }
420         if (getItemCount() > this.maximumItemCount)
421         {
422             this.data.remove(0);
423         }
424         if (notify)
425         {
426             fireSeriesChanged();
427         }
428     }
429
430     /**
431      * Deletes a range of items from the series and sends a
432      * {@link SeriesChangeEvent} to all registered
            listeners.
433      *
434      * @param start  the start index (zero-based).
435      * @param end  the end index (zero-based).
436      */
437     public void delete(int start, int end)
438     {
439         for (int i = start; i <= end; i++)
440         {
441             this.data.remove(start);
442         }
443         fireSeriesChanged();
444     }
```

```
445
446     /**
447      * Removes the item at the specified index and sends a
448      * {@link SeriesChangeEvent} to all registered
           listeners.
449      *
450      * @param index  the index.
451      *
452      * @return The item removed.
453      */
454     public XYDataItem remove(int index)
455     {
456         XYDataItem result = (XYDataItem) this.data.remove(
               index);
457         fireSeriesChanged();
458         return result;
459     }
460
461     /**
462      * Removes the item with the specified x-value and
           sends a
463      * {@link SeriesChangeEvent} to all registered
           listeners.
464      *
465      * @param x  the x-value.
466
467      * @return The item removed.
468      */
469     public XYDataItem remove(Number x)
470     {
471         return remove(indexOf(x));
472     }
473
474     /**
475      * Removes all data items from the series.
476      */
477     public void clear()
478     {
479         if (this.data.size() > 0)
480         {
481             this.data.clear();
482             fireSeriesChanged();
483         }
484     }
485
486     /**
```

```
487         * Return the data item with the specified index.
488         *
489         * @param index  the index.
490         *
491         * @return The data item with the specified index.
492         */
493        public XYDataItem getDataItem(int index)
494        {
495            return (XYDataItem) this.data.get(index);
496        }
497
498        /**
499         * Returns the x-value at the specified index.
500         *
501         * @param index  the index (zero-based).
502         *
503         * @return The x-value (never <code>null</code>).
504         */
505        public Number getX(int index)
506        {
507            return getDataItem(index).getX();
508        }
509
510        /**
511         * Returns the y-value at the specified index.
512         *
513         * @param index  the index (zero-based).
514         *
515         * @return The y-value (possibly <code>null</code>).
516         */
517        public Number getY(int index)
518        {
519            return getDataItem(index).getY();
520        }
521
522        /**
523         * Updates the value of an item in the series and sends
                a
524         * {@link SeriesChangeEvent} to all registered
                listeners.
525         *
526         * @param index  the item (zero based index).
527         * @param y  the new value (<code>null</code> permitted
                ).
528         *
529         * @since 1.0.1
```

```
530         */
531        public void updateByIndex(int index, Number y)
532        {
533            XYDataItem item = getDataItem(index);
534            item.setY(y);
535            fireSeriesChanged();
536        }
537
538        /**
539         * Updates an item in the series.
540         *
541         * @param x  the x-value (<code>null</code> not
               permitted).
542         * @param y  the y-value (<code>null</code> permitted).
543         *
544         * @throws SeriesException if there is no existing item
                with the specified
545         *         x-value.
546         */
547        public void update(Number x, Number y)
548        {
549            int index = indexOf(x);
550            if (index < 0)
551            {
552                throw new SeriesException("No observation for x
                       = " + x);
553            }
554            else
555            {
556                XYDataItem item = getDataItem(index);
557                item.setY(y);
558                fireSeriesChanged();
559            }
560        }
561
562        /**
563         * Adds or updates an item in the series and sends a
564         * {@link SeriesChangeEvent} to all registered
               listeners.
565         *
566         * @param x  the x-value.
567         * @param y  the y-value.
568         *
569         * @return The item that was overwritten, if any.
570         *
571         * @since 1.0.10
```

```
572          */
573         public XYDataItem addOrUpdate(double x, double y)
574         {
575             return addOrUpdate(new Double(x), new Double(y));
576         }
577
578         /**
579          * Adds or updates an item in the series and sends a
580          * {@link SeriesChangeEvent} to all registered
                listeners.
581          *
582          * @param x  the x-value (<code>null</code> not
                permitted).
583          * @param y  the y-value (<code>null</code> permitted).
584          *
585          * @return A copy of the overwritten data item, or <
                code>null</code> if no
586          *         item was overwritten.
587          */
588         public XYDataItem addOrUpdate(Number x, Number y)
589         {
590             if (x == null)
591             {
592                 throw new IllegalArgumentException("Null 'x'
                        argument.");
593             }
594
595             // if we get to here, we know that duplicate X
                    values are not permitted
596             XYDataItem overwritten = null;
597             int index = indexOf(x);
598             if (index >= 0 && !this.allowDuplicateXValues)
599             {
600                 XYDataItem existing = (XYDataItem) this.data.
                        get(index);
601                 try
602                 {
603                     overwritten = (XYDataItem) existing.clone()
                            ;
604                 }
605                 catch (CloneNotSupportedException e)
606                 {
607                     throw new SeriesException("Couldn't clone
                            XYDataItem!");
608                 }
609                 existing.setY(y);
```

```
610         }
611         else
612         {
613             // if the series is sorted, the negative index
                is a result from
614             // Collections.binarySearch() and tells us
                where to insert the
615             // new item...otherwise it will be just -1 and
                we should just
616             // append the value to the list...
617             if (this.autoSort)
618             {
619                 this.data.add(-index - 1, new XYDataItem(x,
                        y));
620             }
621             else
622             {
623                 this.data.add(new XYDataItem(x, y));
624             }
625             // check if this addition will exceed the
                maximum item count...
626             if (getItemCount() > this.maximumItemCount)
627             {
628                 this.data.remove(0);
629             }
630         }
631         fireSeriesChanged();
632         return overwritten;
633     }
634
635     /**
636      * Returns the index of the item with the specified x-
            value, or a negative
637      * index if the series does not contain an item with
            that x-value.  Be
638      * aware that for an unsorted series, the index is
            found by iterating
639      * through all items in the series.
640      *
641      * @param x  the x-value (<code>null</code> not
            permitted).
642      *
643      * @return The index.
644      */
645     public int indexOf(Number x)
646     {
```

```
647         if (this.autoSort)
648         {
649             return Collections.binarySearch(this.data, new
                    XYDataItem(x, null));
650         }
651         else
652         {
653             for (int i = 0; i < this.data.size(); i++)
654             {
655                 XYDataItem item = (XYDataItem) this.data.
                        get(i);
656                 if (item.getX().equals(x))
657                 {
658                     return i;
659                 }
660             }
661             return -1;
662         }
663     }
664
665     /**
666      * Returns a new array containing the x and y values
            from this series.
667      *
668      * @return A new array containing the x and y values
            from this series.
669      *
670      * @since 1.0.4
671      */
672     public double[][] toArray()
673     {
674         int itemCount = getItemCount();
675         double[][] result = new double[2][itemCount];
676         for (int i = 0; i < itemCount; i++)
677         {
678             result[0][i] = this.getX(i).doubleValue();
679             Number y = getY(i);
680             if (y != null)
681             {
682                 result[1][i] = y.doubleValue();
683             }
684             else
685             {
686                 result[1][i] = Double.NaN;
687             }
688         }
```

```
689             return result;
690         }
691
692         /**
693          * Returns a clone of the series.
694          *
695          * @return A clone of the series.
696          *
697          * @throws CloneNotSupportedException if there is a
                cloning problem.
698          */
699         public Object clone() throws CloneNotSupportedException
700         {
701             XYSeries clone = (XYSeries) super.clone();
702             clone.data = (List) ObjectUtilities.deepClone(this.
                data);
703             return clone;
704         }
705
706         /**
707          * Creates a new series by copying a subset of the data
                in this time series.
708          *
709          * @param start  the index of the first item to copy.
710          * @param end  the index of the last item to copy.
711          *
712          * @return A series containing a copy of this series
                from start until end.
713          *
714          * @throws CloneNotSupportedException if there is a
                cloning problem.
715          */
716         public XYSeries createCopy(int start, int end)
717             throws CloneNotSupportedException
718         {
719
720             XYSeries copy = (XYSeries) super.clone();
721             copy.data = new java.util.ArrayList();
722             if (this.data.size() > 0)
723             {
724                 for (int index = start; index <= end; index++)
725                 {
726                     XYDataItem item = (XYDataItem) this.data.
                        get(index);
727                     XYDataItem clone = (XYDataItem) item.clone
                        ();
```

```
728                 try
729                 {
730                     copy.add(clone);
731                 }
732                 catch (SeriesException e)
733                 {
734                     System.err.println("Unable to add
                        cloned data item.");
735                 }
736             }
737         }
738         return copy;
739
740     }
741
742     /**
743      * Tests this series for equality with an arbitrary
           object.
744      *
745      * @param obj  the object to test against for equality
746      *             (<code>null</code> permitted).
747      *
748      * @return A boolean.
749      */
750     public boolean equals(Object obj)
751     {
752         if (obj == this)
753         {
754             return true;
755         }
756         if (!(obj instanceof XYSeries))
757         {
758             return false;
759         }
760         if (!super.equals(obj))
761         {
762             return false;
763         }
764         XYSeries that = (XYSeries) obj;
765         if (this.maximumItemCount != that.maximumItemCount)
766         {
767             return false;
768         }
769         if (this.autoSort != that.autoSort)
770         {
771             return false;
```

```
772         }
773         if (this.allowDuplicateXValues != that.
               allowDuplicateXValues)
774         {
775             return false;
776         }
777         if (!ObjectUtilities.equal(this.data, that.data))
778         {
779             return false;
780         }
781         return true;
782     }
783 
784     /**
785      * Returns a hash code.
786      *
787      * @return A hash code.
788      */
789     public int hashCode()
790     {
791         int result = super.hashCode();
792         // it is too slow to look at every data item, so
               let's just look at
793         // the first, middle and last items...
794         int count = getItemCount();
795         if (count > 0)
796         {
797             XYDataItem item = getDataItem(0);
798             result = 29 * result + item.hashCode();
799         }
800         if (count > 1)
801         {
802             XYDataItem item = getDataItem(count - 1);
803             result = 29 * result + item.hashCode();
804         }
805         if (count > 2)
806         {
807             XYDataItem item = getDataItem(count / 2);
808             result = 29 * result + item.hashCode();
809         }
810         result = 29 * result + this.maximumItemCount;
811         result = 29 * result + (this.autoSort ? 1 : 0);
812         result = 29 * result + (this.allowDuplicateXValues
               ? 1 : 0);
813         return result;
814     }
```

```
815 |
816 | }
```

E.1.4 Defects-Sources/Chart-5/XYSeries-fixed-B.java

```
1  /*
     ===========================================================

2   * JFreeChart : a free chart library for the Java(tm)
       platform
3   *
       ===========================================================

4   *
5   * (C) Copyright 2000-2008, by Object Refinery Limited and
       Contributors.
6   *
7   * Project Info:  http://www.jfree.org/jfreechart/index.
       html
8   *
9   * This library is free software; you can redistribute it
       and/or modify it
10  * under the terms of the GNU Lesser General Public License
        as published by
11  * the Free Software Foundation; either version 2.1 of the
       License, or
12  * (at your option) any later version.
13  *
14  * This library is distributed in the hope that it will be
       useful, but
15  * WITHOUT ANY WARRANTY; without even the implied warranty
       of MERCHANTABILITY
16  * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser
       General Public
17  * License for more details.
18  *
19  * You should have received a copy of the GNU Lesser
       General Public
20  * License along with this library; if not, write to the
       Free Software
21  * Foundation, Inc., 51 Franklin Street, Fifth Floor,
       Boston, MA  02110-1301,
22  * USA.
23  *
24  * [Java is a trademark or registered trademark of Sun
       Microsystems, Inc.
25  * in the United States and other countries.]
26  *
27  * -------------
```

```
28  * XYSeries.java
29  * -------------
30  * (C) Copyright 2001-2008, Object Refinery Limited and
        Contributors.
31  *
32  * Original Author:  David Gilbert (for Object Refinery
        Limited);
33  * Contributor(s):   Aaron Metzger;
34  *                   Jonathan Gabbai;
35  *                   Richard Atkinson;
36  *                   Michel Santos;
37  *                   Ted Schwartz (fix for bug 1955483);
38  *
39  * Changes
40  * -------
41  * 15-Nov-2001 : Version 1 (DG);
42  * 03-Apr-2002 : Added an add(double, double) method (DG);
43  * 29-Apr-2002 : Added a clear() method (ARM);
44  * 06-Jun-2002 : Updated Javadoc comments (DG);
45  * 29-Aug-2002 : Modified to give user control over whether
        or not duplicate
46  *               x-values are allowed (DG);
47  * 07-Oct-2002 : Fixed errors reported by Checkstyle (DG);
48  * 11-Nov-2002 : Added maximum item count, code contributed
        by Jonathan
49  *               Gabbai (DG);
50  * 26-Mar-2003 : Implemented Serializable (DG);
51  * 04-Aug-2003 : Added getItems() method (DG);
52  * 15-Aug-2003 : Changed 'data' from private to protected,
        added new add()
53  *               methods with a 'notify' argument (DG);
54  * 22-Sep-2003 : Added getAllowDuplicateXValues() method (
        RA);
55  * 29-Jan-2004 : Added autoSort attribute, based on a
        contribution by
56  *               Michel Santos - see patch 886740 (DG);
57  * 03-Feb-2004 : Added indexOf() method (DG);
58  * 16-Feb-2004 : Added remove() method (DG);
59  * 18-Aug-2004 : Moved from org.jfree.data --> org.jfree.
        data.xy (DG);
60  * 21-Feb-2005 : Added update(Number, Number) and
        addOrUpdate(Number, Number)
61  *               methods (DG);
62  * 03-May-2005 : Added a new constructor, fixed the
        setMaximumItemCount()
63  *               method to remove items (and notify
```

```
    listeners) if necessary,
64  *                  fixed the add() and addOrUpdate() methods
    to handle unsorted
65  *                  series (DG);
66  * ------------- JFreeChart 1.0.x
    ---------------------------------------------
67  * 11-Jan-2005 : Renamed update(int, Number) -->
    updateByIndex() (DG);
68  * 15-Jan-2007 : Added toArray() method (DG);
69  * 20-Jun-2007 : Removed deprecated code and JCommon
    dependencies (DG);
70  * 31-Oct-2007 : Implemented faster hashCode() (DG);
71  * 22-Nov-2007 : Reimplemented clone() (DG);
72  * 01-May-2008 : Fixed bug 1955483 in addOrUpdate() method,
      thanks to
73  *                  Ted Schwartz (DG);
74  * 24-Nov-2008 : Further fix for 1955483 (DG);
75  *
76  */
77
78 package org.jfree.data.xy;
79
80 import java.io.Serializable;
81 import java.util.Collections;
82 import java.util.List;
83
84 import org.jfree.chart.util.ObjectUtilities;
85 import org.jfree.data.general.Series;
86 import org.jfree.data.general.SeriesChangeEvent;
87 import org.jfree.data.general.SeriesException;
88
89 /**
90  * Represents a sequence of zero or more data items in the
    form (x, y).  By
91  * default, items in the series will be sorted into
    ascending order by x-value,
92  * and duplicate x-values are permitted.  Both the sorting
    and duplicate
93  * defaults can be changed in the constructor.  Y-values
    can be
94  * <code>null</code> to represent missing values.
95  */
96 public class XYSeries extends Series implements Cloneable,
    Serializable
97 {
98
```

```
99      /** For serialization. */
100     static final long serialVersionUID =
           -5908509288197150436L;
101
102     // In version 0.9.12, in response to several developer
           requests, I changed
103     // the 'data' attribute from 'private' to 'protected',
           so that others can
104     // make subclasses that work directly with the
           underlying data structure.
105
106     /** Storage for the data items in the series. */
107     protected List data;
108     /** The maximum number of items for the series. */
109     private int maximumItemCount = Integer.MAX_VALUE;
110     /** A flag that controls whether the items are
           automatically sorted. */
111     private boolean autoSort;
112     /** A flag that controls whether or not duplicate x-
           values are allowed. */
113     private boolean allowDuplicateXValues;
114
115     /**
116      * Creates a new empty series.  By default, items added
             to the series will
117      * be sorted into ascending order by x-value, and
            duplicate x-values will
118      * be allowed (these defaults can be modified with
            another constructor.
119      *
120      * @param key  the series key (<code>null</code> not
            permitted).
121      */
122     public XYSeries(Comparable key)
123     {
124         this(key, true, true);
125     }
126
127     /**
128      * Constructs a new empty series, with the auto-sort
            flag set as requested,
129      * and duplicate values allowed.
130      *
131      * @param key  the series key (<code>null</code> not
            permitted).
132      * @param autoSort  a flag that controls whether or not
```

```
         *  the items in the
133      *                  series are sorted.
134      */
135     public XYSeries(Comparable key, boolean autoSort)
136     {
137         this(key, autoSort, true);
138     }
139
140     /**
141      * Constructs a new xy-series that contains no data.
            You can specify
142      * whether or not duplicate x-values are allowed for
            the series.
143      *
144      * @param key  the series key (<code>null</code> not
            permitted).
145      * @param autoSort  a flag that controls whether or not
             the items in the
146      *                  series are sorted.
147      * @param allowDuplicateXValues  a flag that controls
            whether duplicate
148      *                               x-values are allowed.
149      */
150     public XYSeries(Comparable key,
151                     boolean autoSort,
152                     boolean allowDuplicateXValues)
153     {
154         super(key);
155         this.data = new java.util.ArrayList();
156         this.autoSort = autoSort;
157         this.allowDuplicateXValues = allowDuplicateXValues;
158     }
159
160     /**
161      * Returns the flag that controls whether the items in
            the series are
162      * automatically sorted.  There is no setter for this
            flag, it must be
163      * defined in the series constructor.
164      *
165      * @return A boolean.
166      */
167     public boolean getAutoSort()
168     {
169         return this.autoSort;
170     }
```

```
171
172     /**
173      * Returns a flag that controls whether duplicate x-
            values are allowed.
174      * This flag can only be set in the constructor.
175      *
176      * @return A boolean.
177      */
178     public boolean getAllowDuplicateXValues()
179     {
180         return this.allowDuplicateXValues;
181     }
182
183     /**
184      * Returns the number of items in the series.
185      *
186      * @return The item count.
187      */
188     public int getItemCount()
189     {
190         return this.data.size();
191     }
192
193     /**
194      * Returns the list of data items for the series (the
            list contains
195      * {@link XYDataItem} objects and is unmodifiable).
196      *
197      * @return The list of data items.
198      */
199     public List getItems()
200     {
201         return Collections.unmodifiableList(this.data);
202     }
203
204     /**
205      * Returns the maximum number of items that will be
            retained in the series.
206      * The default value is <code>Integer.MAX_VALUE</code>.
207      *
208      * @return The maximum item count.
209      * @see #setMaximumItemCount(int)
210      */
211     public int getMaximumItemCount()
212     {
213         return this.maximumItemCount;
```

```
214     }
215
216     /**
217      * Sets the maximum number of items that will be
            retained in the series.
218      * If you add a new item to the series such that the
            number of items will
219      * exceed the maximum item count, then the first
            element in the series is
220      * automatically removed, ensuring that the maximum
            item count is not
221      * exceeded.
222      * <p>
223      * Typically this value is set before the series is
            populated with data,
224      * but if it is applied later, it may cause some items
            to be removed from
225      * the series (in which case a {@link SeriesChangeEvent
            } will be sent to
226      * all registered listeners.
227      *
228      * @param maximum  the maximum number of items for the
            series.
229      */
230     public void setMaximumItemCount(int maximum)
231     {
232         this.maximumItemCount = maximum;
233         boolean dataRemoved = false;
234         while (this.data.size() > maximum)
235         {
236             this.data.remove(0);
237             dataRemoved = true;
238         }
239         if (dataRemoved)
240         {
241             fireSeriesChanged();
242         }
243     }
244
245     /**
246      * Adds a data item to the series and sends a {@link
            SeriesChangeEvent} to
247      * all registered listeners.
248      *
249      * @param item  the (x, y) item (<code>null</code> not
            permitted).
```

```
250          */
251         public void add(XYDataItem item)
252         {
253             // argument checking delegated...
254             add(item, true);
255         }
256
257         /**
258          * Adds a data item to the series and sends a {@link
                SeriesChangeEvent} to
259          * all registered listeners.
260          *
261          * @param x  the x value.
262          * @param y  the y value.
263          */
264         public void add(double x, double y)
265         {
266             add(new Double(x), new Double(y), true);
267         }
268
269         /**
270          * Adds a data item to the series and, if requested,
                sends a
271          * {@link SeriesChangeEvent} to all registered
                listeners.
272          *
273          * @param x  the x value.
274          * @param y  the y value.
275          * @param notify  a flag that controls whether or not a
276          *                {@link SeriesChangeEvent} is sent to
                all registered
277          *                listeners.
278          */
279         public void add(double x, double y, boolean notify)
280         {
281             add(new Double(x), new Double(y), notify);
282         }
283
284         /**
285          * Adds a data item to the series and sends a {@link
                SeriesChangeEvent} to
286          * all registered listeners.  The unusual pairing of
                parameter types is to
287          * make it easier to add <code>null</code> y-values.
288          *
289          * @param x  the x value.
```

```
290         * @param y  the y value (<code>null</code> permitted).
291         */
292        public void add(double x, Number y)
293        {
294            add(new Double(x), y);
295        }
296
297        /**
298         * Adds a data item to the series and, if requested,
               sends a
299         * {@link SeriesChangeEvent} to all registered
               listeners.  The unusual
300         * pairing of parameter types is to make it easier to
               add null y-values.
301         *
302         * @param x  the x value.
303         * @param y  the y value (<code>null</code> permitted).
304         * @param notify  a flag that controls whether or not a
305         *                {@link SeriesChangeEvent} is sent to
               all registered
306         *                listeners.
307         */
308        public void add(double x, Number y, boolean notify)
309        {
310            add(new Double(x), y, notify);
311        }
312
313        /**
314         * Adds a new data item to the series (in the correct
               position if the
315         * <code>autoSort</code> flag is set for the series)
               and sends a
316         * {@link SeriesChangeEvent} to all registered
               listeners.
317         * <P>
318         * Throws an exception if the x-value is a duplicate
               AND the
319         * allowDuplicateXValues flag is false.
320         *
321         * @param x  the x-value (<code>null</code> not
               permitted).
322         * @param y  the y-value (<code>null</code> permitted).
323         *
324         * @throws SeriesException if the x-value is a
               duplicate and the
325         *     <code>allowDuplicateXValues</code> flag is not
```

```
         set for this series.
326       */
327      public void add(Number x, Number y)
328      {
329          // argument checking delegated...
330          add(x, y, true);
331      }
332
333      /**
334       * Adds new data to the series and, if requested, sends
              a
335       * {@link SeriesChangeEvent} to all registered
             listeners.
336       * <P>
337       * Throws an exception if the x-value is a duplicate
             AND the
338       * allowDuplicateXValues flag is false.
339       *
340       * @param x  the x-value (<code>null</code> not
             permitted).
341       * @param y  the y-value (<code>null</code> permitted).
342       * @param notify  a flag the controls whether or not a
343       *                {@link SeriesChangeEvent} is sent to
             all registered
344       *                listeners.
345       */
346      public void add(Number x, Number y, boolean notify)
347      {
348          // delegate argument checking to XYDataItem...
349          XYDataItem item = new XYDataItem(x, y);
350          add(item, notify);
351      }
352
353      /**
354       * Adds a data item to the series and, if requested,
             sends a
355       * {@link SeriesChangeEvent} to all registered
             listeners.
356       *
357       * @param item  the (x, y) item (<code>null</code> not
             permitted).
358       * @param notify  a flag that controls whether or not a
359       *                {@link SeriesChangeEvent} is sent to
             all registered
360       *                listeners.
361       */
```

```
362     public void add(XYDataItem item, boolean notify)
363     {
364
365         if (item == null)
366         {
367             throw new IllegalArgumentException("Null 'item'
                    argument.");
368         }
369
370         if (this.autoSort)
371         {
372             int index = Collections.binarySearch(this.data,
                    item);
373             if (index < 0)
374             {
375                 this.data.add(-index - 1, item);
376             }
377             else
378             {
379                 if (this.allowDuplicateXValues)
380                 {
381                     // need to make sure we are adding *
                           after* any duplicates
382                     int size = this.data.size();
383                     while (index < size
384                             && item.compareTo(this.data.get(
                                index)) == 0)
385                     {
386                         index++;
387                     }
388                     if (index < this.data.size())
389                     {
390                         this.data.add(index, item);
391                     }
392                     else
393                     {
394                         this.data.add(item);
395                     }
396                 }
397                 else
398                 {
399                     throw new SeriesException("X-value
                            already exists.");
400                 }
401             }
402         }
```

```
403         else
404         {
405             if (!this.allowDuplicateXValues)
406             {
407                 // can't allow duplicate values, so we need
                        to check whether
408                 // there is an item with the given x-value
                        already
409                 int index = indexOf(item.getX());
410                 if (index >= 0)
411                 {
412                     throw new SeriesException("X-value
                            already exists.");
413                 }
414             }
415             this.data.add(item);
416         }
417         if (getItemCount() > this.maximumItemCount)
418         {
419             this.data.remove(0);
420         }
421         if (notify)
422         {
423             fireSeriesChanged();
424         }
425     }
426
427     /**
428      * Deletes a range of items from the series and sends a
429      * {@link SeriesChangeEvent} to all registered
            listeners.
430      *
431      * @param start  the start index (zero-based).
432      * @param end  the end index (zero-based).
433      */
434     public void delete(int start, int end)
435     {
436         for (int i = start; i <= end; i++)
437         {
438             this.data.remove(start);
439         }
440         fireSeriesChanged();
441     }
442
443     /**
444      * Removes the item at the specified index and sends a
```

```
445         * {@link SeriesChangeEvent} to all registered
                listeners.
446         *
447         * @param index  the index.
448         *
449         * @return The item removed.
450         */
451        public XYDataItem remove(int index)
452        {
453            XYDataItem result = (XYDataItem) this.data.remove(
                   index);
454            fireSeriesChanged();
455            return result;
456        }
457
458        /**
459         * Removes the item with the specified x-value and
                sends a
460         * {@link SeriesChangeEvent} to all registered
                listeners.
461         *
462         * @param x  the x-value.
463
464         * @return The item removed.
465         */
466        public XYDataItem remove(Number x)
467        {
468            return remove(indexOf(x));
469        }
470
471        /**
472         * Removes all data items from the series.
473         */
474        public void clear()
475        {
476            if (this.data.size() > 0)
477            {
478                this.data.clear();
479                fireSeriesChanged();
480            }
481        }
482
483        /**
484         * Return the data item with the specified index.
485         *
486         * @param index  the index.
```

```
487         *
488         * @return The data item with the specified index.
489         */
490        public XYDataItem getDataItem(int index)
491        {
492            return (XYDataItem) this.data.get(index);
493        }
494
495        /**
496         * Returns the x-value at the specified index.
497         *
498         * @param index  the index (zero-based).
499         *
500         * @return The x-value (never <code>null</code>).
501         */
502        public Number getX(int index)
503        {
504            return getDataItem(index).getX();
505        }
506
507        /**
508         * Returns the y-value at the specified index.
509         *
510         * @param index  the index (zero-based).
511         *
512         * @return The y-value (possibly <code>null</code>).
513         */
514        public Number getY(int index)
515        {
516            return getDataItem(index).getY();
517        }
518
519        /**
520         * Updates the value of an item in the series and sends
                a
521         * {@link SeriesChangeEvent} to all registered
                listeners.
522         *
523         * @param index  the item (zero based index).
524         * @param y  the new value (<code>null</code> permitted
                ).
525         *
526         * @since 1.0.1
527         */
528        public void updateByIndex(int index, Number y)
529        {
```

```
530         XYDataItem item = getDataItem(index);
531         item.setY(y);
532         fireSeriesChanged();
533     }
534 
535     /**
536      * Updates an item in the series.
537      *
538      * @param x  the x-value (<code>null</code> not
             permitted).
539      * @param y  the y-value (<code>null</code> permitted).
540      *
541      * @throws SeriesException if there is no existing item
                 with the specified
542      *         x-value.
543      */
544     public void update(Number x, Number y)
545     {
546         int index = indexOf(x);
547         if (index < 0)
548         {
549             throw new SeriesException("No observation for x
                        = " + x);
550         }
551         else
552         {
553             XYDataItem item = getDataItem(index);
554             item.setY(y);
555             fireSeriesChanged();
556         }
557     }
558 
559     /**
560      * Adds or updates an item in the series and sends a
561      * {@link SeriesChangeEvent} to all registered
             listeners.
562      *
563      * @param x  the x-value.
564      * @param y  the y-value.
565      *
566      * @return The item that was overwritten, if any.
567      *
568      * @since 1.0.10
569      */
570     public XYDataItem addOrUpdate(double x, double y)
571     {
```

```
572         return addOrUpdate(new Double(x), new Double(y));
573     }
574
575     /**
576      * Adds or updates an item in the series and sends a
577      * {@link SeriesChangeEvent} to all registered
            listeners.
578      *
579      * @param x  the x-value (<code>null</code> not
            permitted).
580      * @param y  the y-value (<code>null</code> permitted).
581      *
582      * @return A copy of the overwritten data item, or <
            code>null</code> if no
583      *         item was overwritten.
584      */
585     public XYDataItem addOrUpdate(Number x, Number y)
586     {
587         if (x == null)
588         {
589             throw new IllegalArgumentException("Null 'x'
                argument.");
590         }
591         if (this.allowDuplicateXValues)
592         {
593             add(x, y);
594             return null;
595         }
596
597         // if we get to here, we know that duplicate X
            values are not permitted
598         XYDataItem overwritten = null;
599         int index = indexOf(x);
600         if (index >= 0)
601         {
602             XYDataItem existing = (XYDataItem) this.data.
                get(index);
603             try
604             {
605                 overwritten = (XYDataItem) existing.clone()
                    ;
606             }
607             catch (CloneNotSupportedException e)
608             {
609                 throw new SeriesException("Couldn't clone
                    XYDataItem!");
```

```
610             }
611             existing.setY(y);
612         }
613         else
614         {
615             // if the series is sorted, the negative index
                    is a result from
616             // Collections.binarySearch() and tells us
                    where to insert the
617             // new item...otherwise it will be just -1 and
                    we should just
618             // append the value to the list...
619             if (this.autoSort)
620             {
621                 this.data.add(-index - 1, new XYDataItem(x,
                        y));
622             }
623             else
624             {
625                 this.data.add(new XYDataItem(x, y));
626             }
627             // check if this addition will exceed the
                    maximum item count...
628             if (getItemCount() > this.maximumItemCount)
629             {
630                 this.data.remove(0);
631             }
632         }
633         fireSeriesChanged();
634         return overwritten;
635     }
636
637     /**
638      * Returns the index of the item with the specified x-
            value, or a negative
639      * index if the series does not contain an item with
            that x-value.  Be
640      * aware that for an unsorted series, the index is
            found by iterating
641      * through all items in the series.
642      *
643      * @param x  the x-value (<code>null</code> not
            permitted).
644      *
645      * @return The index.
646      */
```

```
647     public int indexOf(Number x)
648     {
649         if (this.autoSort)
650         {
651             return Collections.binarySearch(this.data, new
                    XYDataItem(x, null));
652         }
653         else
654         {
655             for (int i = 0; i < this.data.size(); i++)
656             {
657                 XYDataItem item = (XYDataItem) this.data.
                        get(i);
658                 if (item.getX().equals(x))
659                 {
660                     return i;
661                 }
662             }
663             return -1;
664         }
665     }
666
667     /**
668      * Returns a new array containing the x and y values
            from this series.
669      *
670      * @return A new array containing the x and y values
            from this series.
671      *
672      * @since 1.0.4
673      */
674     public double[][] toArray()
675     {
676         int itemCount = getItemCount();
677         double[][] result = new double[2][itemCount];
678         for (int i = 0; i < itemCount; i++)
679         {
680             result[0][i] = this.getX(i).doubleValue();
681             Number y = getY(i);
682             if (y != null)
683             {
684                 result[1][i] = y.doubleValue();
685             }
686             else
687             {
688                 result[1][i] = Double.NaN;
```

```
689             }
690         }
691         return result;
692     }
693 
694     /**
695      * Returns a clone of the series.
696      *
697      * @return A clone of the series.
698      *
699      * @throws CloneNotSupportedException if there is a
           cloning problem.
700      */
701     public Object clone() throws CloneNotSupportedException
702     {
703         XYSeries clone = (XYSeries) super.clone();
704         clone.data = (List) ObjectUtilities.deepClone(this.
               data);
705         return clone;
706     }
707 
708     /**
709      * Creates a new series by copying a subset of the data
           in this time series.
710      *
711      * @param start  the index of the first item to copy.
712      * @param end  the index of the last item to copy.
713      *
714      * @return A series containing a copy of this series
           from start until end.
715      *
716      * @throws CloneNotSupportedException if there is a
           cloning problem.
717      */
718     public XYSeries createCopy(int start, int end)
719         throws CloneNotSupportedException
720     {
721 
722         XYSeries copy = (XYSeries) super.clone();
723         copy.data = new java.util.ArrayList();
724         if (this.data.size() > 0)
725         {
726             for (int index = start; index <= end; index++)
727             {
728                 XYDataItem item = (XYDataItem) this.data.
                       get(index);
```

```
729                 XYDataItem clone = (XYDataItem) item.clone
                        ();
730                 try
731                 {
732                     copy.add(clone);
733                 }
734                 catch (SeriesException e)
735                 {
736                     System.err.println("Unable to add
                            cloned data item.");
737                 }
738             }
739         }
740         return copy;
741
742     }
743
744     /**
745      * Tests this series for equality with an arbitrary
             object.
746      *
747      * @param obj  the object to test against for equality
748      *             (<code>null</code> permitted).
749      *
750      * @return A boolean.
751      */
752     public boolean equals(Object obj)
753     {
754         if (obj == this)
755         {
756             return true;
757         }
758         if (!(obj instanceof XYSeries))
759         {
760             return false;
761         }
762         if (!super.equals(obj))
763         {
764             return false;
765         }
766         XYSeries that = (XYSeries) obj;
767         if (this.maximumItemCount != that.maximumItemCount)
768         {
769             return false;
770         }
771         if (this.autoSort != that.autoSort)
```

```
772         {
773             return false;
774         }
775         if (this.allowDuplicateXValues != that.
               allowDuplicateXValues)
776         {
777             return false;
778         }
779         if (!ObjectUtilities.equal(this.data, that.data))
780         {
781             return false;
782         }
783         return true;
784     }
785 
786     /**
787      * Returns a hash code.
788      *
789      * @return A hash code.
790      */
791     public int hashCode()
792     {
793         int result = super.hashCode();
794         // it is too slow to look at every data item, so
               let's just look at
795         // the first, middle and last items...
796         int count = getItemCount();
797         if (count > 0)
798         {
799             XYDataItem item = getDataItem(0);
800             result = 29 * result + item.hashCode();
801         }
802         if (count > 1)
803         {
804             XYDataItem item = getDataItem(count - 1);
805             result = 29 * result + item.hashCode();
806         }
807         if (count > 2)
808         {
809             XYDataItem item = getDataItem(count / 2);
810             result = 29 * result + item.hashCode();
811         }
812         result = 29 * result + this.maximumItemCount;
813         result = 29 * result + (this.autoSort ? 1 : 0);
814         result = 29 * result + (this.allowDuplicateXValues
               ? 1 : 0);
```

```
815 |           return result;
816 |       }
817 |
818 | }
```

E.2 Manually Transformed Sources for Task Chart-8

E.2.1 Defects-Sources/Chart-8/Week-bugged-A.java

```
1  /*
     ===========================================================

2   * JFreeChart : a free chart library for the Java(tm)
       platform
3   *
       ===========================================================

4   *
5   * (C) Copyright 2000-2008, by Object Refinery Limited and
6   * Contributors.
7   *
8   * Project Info:  http://www.jfree.org/jfreechart/index.
       html
9   *
10  * This library is free software; you can redistribute it
       and/or
11  * modify it under the terms of the GNU Lesser General
       Public License
12  * as published by the Free Software Foundation; either
       version 2.1 of
13  * the License, or (at your option) any later version.
14  *
15  * This library is distributed in the hope that it will be
       useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty
       of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
         the GNU
18  * Lesser General Public License for more details.
19  *
20  * You should have received a copy of the GNU Lesser
       General Public
21  * License along with this library; if not, write to the
       Free Software
22  * Foundation, Inc.,
23  * 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
         USA.
24  *
25  * [Java is a trademark or registered trademark of
26  * Sun Microsystems, Inc. in the United States and other
```

```
    countries.]
27  *
28  * ---------
29  * Week.java
30  * ---------
31  * (C) Copyright 2001-2008, by Object Refinery Limited and
32  * Contributors.
33  *
34  * Original Author:  David Gilbert (for Object Refinery
    Limited);
35  * Contributor(s):   Aimin Han;
36  *
37  * Changes
38  * -------
39  * 11-Oct-2001 : Version 1 (DG);
40  * 18-Dec-2001 : Changed order of parameters in constructor
    (DG);
41  * 19-Dec-2001 : Added a new constructor as suggested by
    Paul English
42  *               (DG);
43  * 29-Jan-2002 : Worked on the parseWeek() method (DG);
44  * 13-Feb-2002 : Fixed bug in Week(Date) constructor (DG);
45  * 26-Feb-2002 : Changed getStart(), getMiddle() and getEnd
    () methods
46  *               to evaluate with reference to a particular
    time zone
47  *               (DG);
48  * 05-Apr-2002 : Reinstated this class to the JCommon
    library (DG);
49  * 24-Jun-2002 : Removed unnecessary main method (DG);
50  * 10-Sep-2002 : Added getSerialIndex() method (DG);
51  * 06-Oct-2002 : Fixed errors reported by Checkstyle (DG);
52  * 18-Oct-2002 : Changed to observe 52 or 53 weeks per year
    ,
53  *               consistent with GregorianCalendar. Thanks
    to
54  *               Aimin Han for the code (DG);
55  * 02-Jan-2003 : Removed debug code (DG);
56  * 13-Mar-2003 : Moved to com.jrefinery.data.time package,
    and
57  *               implemented Serializable (DG);
58  * 21-Oct-2003 : Added hashCode() method (DG);
59  * 24-May-2004 : Modified getFirstMillisecond() and
60  *               getLastMillisecond() to take account of
61  *               firstDayOfWeek setting in Java's Calendar
    class (DG);
```

```
62  * 30-Sep-2004 : Replaced getTime().getTime() with
        getTimeInMillis()
63  *                (DG);
64  * 04-Nov-2004 : Reverted change of 30-Sep-2004, because it
        won't work
65  *                for JDK 1.3 (DG);
66  * ------------- JFREECHART 1.0.x
        ---------------------------------
67  * 06-Mar-2006 : Fix for bug 1448828, incorrect calculation
        of week
68  *                and year for the first few days of some
        years (DG);
69  * 05-Oct-2006 : Updated API docs (DG);
70  * 06-Oct-2006 : Refactored to cache first and last
        millisecond values
71  *                (DG);
72  * 09-Jan-2007 : Fixed bug in next() (DG);
73  * 28-Aug-2007 : Added new constructor to avoid problem in
        creating
74  *                new instances (DG);
75  * 19-Dec-2007 : Fixed bug in deprecated constructor (DG);
76  *
77  */
78
79
80 package org.jfree.data.time;
81
82
83 import java.io.Serializable;
84 import java.util.Calendar;
85 import java.util.Date;
86 import java.util.Locale;
87 import java.util.TimeZone;
88
89
90 /**
91  * A calendar week.  All years are considered to have 53
        weeks,
92  * numbered from 1 to 53, although in many cases the 53rd
        week is
93  * empty.  Most of the time, the 1st week of the year *
        begins* in the
94  * previous calendar year, but it always finishes in the
        current year
95  * (this behaviour matches the workings of the
96  * <code>GregorianCalendar</code> class).
```

```
97  * <P>
98  * This class is immutable, which is a requirement for all
99  * {@link RegularTimePeriod} subclasses.
100 */
101 public class Week extends RegularTimePeriod implements
        Serializable {
102
103
104     /** For serialization. */
105     private static final long serialVersionUID =
            1856387786939865061L;
106
107     /** Constant for the first week in the year. */
108     public static final int FIRST_WEEK_IN_YEAR = 1;
109
110     /** Constant for the last week in the year. */
111     public static final int LAST_WEEK_IN_YEAR = 53;
112
113     /** The year in which the week falls. */
114     private short year;
115
116     /** The week (1-53). */
117     private byte week;
118
119     /** The first millisecond. */
120     private long firstMillisecond;
121
122     /** The last millisecond. */
123     private long lastMillisecond;
124
125
126     /**
127      * Creates a new time period for the week in which the
            current
128      * system date/time falls.
129      */
130     public Week() {
131         this(new Date());
132     }
133
134     /**
135      * Creates a time period representing the week in the
            specified
136      * year.
137      *
138      * @param week  the week (1 to 53).
```

```
139         * @param year  the year (1900 to 9999).
140         */
141        public Week(int week, int year) {
142            if ((week < FIRST_WEEK_IN_YEAR) && (week >
                   LAST_WEEK_IN_YEAR)) {
143                throw new IllegalArgumentException(
144                        "The 'week' argument must be in the
                           range 1 - 53.");
145            }
146            this.week = (byte) week;
147            this.year = (short) year;
148            peg(Calendar.getInstance());
149        }
150
151        /**
152         * Creates a time period representing the week in the
               specified
153         * year.
154         *
155         * @param week  the week (1 to 53).
156         * @param year  the year (1900 to 9999).
157         */
158        public Week(int week, Year year) {
159            if ((week < FIRST_WEEK_IN_YEAR) && (week >
                   LAST_WEEK_IN_YEAR)) {
160                throw new IllegalArgumentException(
161                        "The 'week' argument must be in the
                           range 1 - 53.");
162            }
163            this.week = (byte) week;
164            this.year = (short) year.getYear();
165            peg(Calendar.getInstance());
166        }
167
168        /**
169         * Creates a time period for the week in which the
               specified
170         * date/time falls.
171         *
172         * @param time  the time (<code>null</code> not
               permitted).
173         */
174        public Week(Date time) {
175
176            // defer argument checking...
177            this(time, RegularTimePeriod.DEFAULT_TIME_ZONE,
```

```
            Locale.getDefault());
178     }
179
180     /**
181      * Creates a time period for the week in which the
            specified
182      * date/time falls, calculated relative to the
            specified time
183      * zone.
184      *
185      * @param time  the date/time (<code>null</code> not
            permitted).
186      * @param zone  the time zone (<code>null</code> not
            permitted).
187      *
188      * @deprecated As of 1.0.7, use
189      *             {@link #Week(Date, TimeZone, Locale)}.
190      */
191     public Week(Date time, TimeZone zone) {
192
193         // defer argument checking...
194         this(time, RegularTimePeriod.DEFAULT_TIME_ZONE,
               Locale.getDefault());
195     }
196
197     /**
198      * Creates a time period for the week in which the
            specified
199      * date/time falls, calculated relative to the
            specified time
200      * zone.
201      *
202      * @param time  the date/time (<code>null</code> not
            permitted).
203      * @param zone  the time zone (<code>null</code> not
            permitted).
204      * @param locale  the locale (<code>null</code> not
            permitted).
205      *
206      * @since 1.0.7
207      */
208     public Week(Date time, TimeZone zone, Locale locale) {
209         if (time == null) {
210             throw new IllegalArgumentException("Null 'time'
                    argument.");
211         }
```

```
212         if (zone == null) {
213             throw new IllegalArgumentException("Null 'zone'
                    argument.");
214         }
215         if (locale == null) {
216             throw new IllegalArgumentException("Null '
                   locale' argument.");
217         }
218         Calendar calendar = Calendar.getInstance(zone,
                locale);
219         calendar.setTime(time);
220 
221         // sometimes the last few days of the year are
                considered to fall in
222         // the *first* week of the following year.  Refer
                to the Javadocs for
223         // GregorianCalendar.
224         int tempWeek = calendar.get(Calendar.WEEK_OF_YEAR);
225         if (tempWeek == 1
226                 && calendar.get(Calendar.MONTH) == Calendar
                       .DECEMBER) {
227             this.week = 1;
228             this.year = (short) (calendar.get(Calendar.YEAR
                    ) + 1);
229         } else {
230             this.week = (byte) Math.min(tempWeek,
                    LAST_WEEK_IN_YEAR);
231             int yyyy = calendar.get(Calendar.YEAR);
232 
233             // alternatively, sometimes the first few days
                    of the year are
234             // considered to fall in the *last* week of the
                     previous year...
235             if (calendar.get(Calendar.MONTH) == Calendar.
                    JANUARY
236                     && this.week >= 52) {
237                 yyyy--;
238             }
239             this.year = (short) yyyy;
240         }
241         peg(calendar);
242     }
243 
244 
245     /**
246      * Returns the year in which the week falls.
```

```
247         *
248         * @return The year (never <code>null</code>).
249         */
250        public Year getYear() {
251            return new Year(this.year);
252        }
253
254        /**
255         * Returns the year in which the week falls, as an
               integer value.
256         *
257         * @return The year.
258         */
259        public int getYearValue() {
260            return this.year;
261        }
262
263        /**
264         * Returns the week.
265         *
266         * @return The week.
267         */
268        public int getWeek() {
269            return this.week;
270        }
271
272        /**
273         * Returns the first millisecond of the week.  This
               will be
274         * determined relative to the time zone specified in
               the
275         * constructor, or in the calendar instance passed in
               the most
276         * recent call to the {@link #peg(Calendar)} method.
277         *
278         * @return The first millisecond of the week.
279         *
280         * @see #getLastMillisecond()
281         */
282        public long getFirstMillisecond() {
283            return this.firstMillisecond;
284        }
285
286        /**
287         * Returns the last millisecond of the week.  This will
                be
```

```
288         * determined relative to the time zone specified in
                the
289         * constructor, or in the calendar instance passed in
                the most
290         * recent call to the {@link #peg(Calendar)} method.
291         *
292         * @return The last millisecond of the week.
293         *
294         * @see #getFirstMillisecond()
295         */
296        public long getLastMillisecond() {
297            return this.lastMillisecond;
298        }
299
300        /**
301         * Recalculates the start date/time and end date/time
                for this
302         * time period relative to the supplied calendar (which
303         * incorporates a time zone).
304         *
305         * @param calendar  the calendar (<code>null</code> not
306         *                  permitted).
307         *
308         * @since 1.0.3
309         */
310        public void peg(Calendar calendar) {
311            this.firstMillisecond = getFirstMillisecond(
                    calendar);
312            this.lastMillisecond = getLastMillisecond(calendar)
                    ;
313        }
314
315        /**
316         * Returns the week preceding this one.  This method
                will return
317         * <code>null</code> for some lower limit on the range
                of weeks
318         * (currently week 1, 1900).  For week 1 of any year,
                the previous
319         * week is always week 53, but week 53 may not contain
                any days
320         * (you should check for this).
321         *
322         * @return The preceding week (possibly <code>null</
                code>).
323         */
```

```
324     public RegularTimePeriod previous() {
325         Week result;
326
327         if (this.week != FIRST_WEEK_IN_YEAR) {
328             result = new Week(this.week - 1, this.year);
329         } else {
330
331             // we need to work out if the previous year has
                    52 or 53 weeks...
332             if (this.year > 1900) {
333                 int yy = this.year - 1;
334                 Calendar prevYearCalendar = Calendar.
                        getInstance();
335                 prevYearCalendar.set(yy, Calendar.DECEMBER,
                        31);
336                 result = new Week(prevYearCalendar.
                        getActualMaximum(
337                         Calendar.WEEK_OF_YEAR), yy);
338             } else {
339                 result = null;
340             }
341         }
342         return result;
343     }
344
345     /**
346      * Returns the week following this one.  This method
            will return
347      * <code>null</code> for some upper limit on the range
            of weeks
348      * (currently week 53, 9999).  For week 52 of any year,
             the
349      * following week is always week 53, but week 53 may
            not contain
350      * any days (you should check for this).
351      *
352      * @return The following week (possibly <code>null</
            code>).
353      */
354     public RegularTimePeriod next() {
355         Week result;
356
357         if (this.week < 52) {
358             result = new Week(this.week + 1, this.year);
359         } else {
360             Calendar calendar = Calendar.getInstance();
```

```
361             calendar.set(this.year, Calendar.DECEMBER, 31);
362             int actualMaxWeek
363                 = calendar.getActualMaximum(Calendar.
                       WEEK_OF_YEAR);
364             if (this.week < actualMaxWeek) {
365                 result = new Week(this.week + 1, this.year)
                       ;
366             } else {
367                 if (this.year < 9999) {
368                     result = new Week(FIRST_WEEK_IN_YEAR,
                           this.year + 1);
369                 } else {
370                     result = null;
371                 }
372             }
373         }
374         return result;
375     }
376 
377     /**
378      * Returns a serial index number for the week.
379      *
380      * @return The serial index number.
381      */
382     public long getSerialIndex() {
383         return this.year * 53L + this.week;
384     }
385 
386     /**
387      * Returns the first millisecond of the week, evaluated
           using the
388      * supplied calendar (which determines the time zone).
389      *
390      * @param calendar  the calendar (<code>null</code> not
391      *                  permitted).
392      *
393      * @return The first millisecond of the week.
394      *
395      * @throws NullPointerException if <code>calendar</code
           > is
396      *     <code>null</code>.
397      */
398     public long getFirstMillisecond(Calendar calendar) {
399         Calendar c = (Calendar) calendar.clone();
400 
401         c.clear();
```

```
402         c.set(Calendar.YEAR, this.year);
403         c.set(Calendar.WEEK_OF_YEAR, this.week);
404         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
405         c.set(Calendar.HOUR, 0);
406         c.set(Calendar.MINUTE, 0);
407         c.set(Calendar.SECOND, 0);
408         c.set(Calendar.MILLISECOND, 0);
409
410         //return c.getTimeInMillis();  // this won't work
                for JDK 1.3
411         return c.getTime().getTime();
412     }
413
414     /**
415      * Returns the last millisecond of the week, evaluated
            using the
416      * supplied calendar (which determines the time zone).
417      *
418      * @param calendar  the calendar (<code>null</code> not
419      *                  permitted).
420      *
421      * @return The last millisecond of the week.
422      *
423      * @throws NullPointerException if <code>calendar</code
            > is
424      *     <code>null</code>.
425      */
426     public long getLastMillisecond(Calendar calendar) {
427         Calendar c = (Calendar) calendar.clone();
428
429         c.clear();
430         c.set(Calendar.YEAR, this.year);
431         c.set(Calendar.WEEK_OF_YEAR, this.week + 1);
432         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
433         c.set(Calendar.HOUR, 0);
434         c.set(Calendar.MINUTE, 0);
435         c.set(Calendar.SECOND, 0);
436         c.set(Calendar.MILLISECOND, 0);
437
438         //return c.getTimeInMillis();  // this won't work
                for JDK 1.3
439         return c.getTime().getTime() - 1;
440     }
441
442     /**
443      * Returns a string representing the week (e.g. "Week
```

```
         9, 2002").
444      *
445      * TODO: look at internationalisation.
446      *
447      * @return A string representing the week.
448      */
449     public String toString() {
450         return "Week " + this.week + ", " + this.year;
451     }
452
453     /**
454      * Tests the equality of this Week object to an
            arbitrary object.
455      * Returns true if the target is a Week instance
            representing the
456      * same week as this object.  In all other cases,
            returns false.
457      *
458      * @param obj  the object (<code>null</code> permitted)
            .
459      *
460      * @return <code>true</code> if week and year of this
            and object
461      *         are the same.
462      */
463     public boolean equals(Object obj) {
464         if (obj == this) {
465             return true;
466         }
467         if (!(obj instanceof Week)) {
468             return false;
469         }
470         Week that = (Week) obj;
471         if (this.week != that.week) {
472             return false;
473         }
474         if (this.year != that.year) {
475             return false;
476         }
477         return true;
478     }
479
480     /**
481      * Returns a hash code for this object instance.  The
            approach
482      * described by Joshua Bloch in "Effective Java" has
```

```
         been used
483      * here:
484      * <p>
485      * <code>
486      * http://developer.java.sun.com/developer/Books/
             effectivejava
487      * /Chapter3.pdf</code>
488      *
489      * @return A hash code.
490      */
491     public int hashCode() {
492         int result = 17;
493
494         result = 37 * result + this.week;
495         result = 37 * result + this.year;
496         return result;
497     }
498
499     /**
500      * Returns an integer indicating the order of this Week
             object
501      * relative to the specified object:
502      *
503      * negative == before, zero == same, positive == after.
504      *
505      * @param o1  the object to compare.
506      *
507      * @return negative == before, zero == same, positive
             == after.
508      */
509     public int compareTo(Object o1) {
510         int result;
511
512         // CASE 1 : Comparing to another Week object
513         // -----------------------------------------
514         if (o1 instanceof Week) {
515             Week w = (Week) o1;
516             result = this.year - w.getYear().getYear();
517             if (result == 0) {
518                 result = this.week - w.getWeek();
519             }
520         } else if (o1 instanceof RegularTimePeriod) {
521
522             // CASE 2 : Comparing to another TimePeriod
                    object
523             //
```

```
            ------------------------------------------------

            // more difficult case - evaluate later...
            result = 0;
        } else {

            // CASE 3 : Comparing to a non-TimePeriod
               object
            //
               ------------------------------------------

            // consider time periods to be ordered after
               general objects
            result = 1;
        }

        return result;
    }

    /**
     * Parses the string argument as a week.
     * <P>
     * This method is required to accept the format "YYYY-
        Wnn".  It
     * will also accept "Wnn-YYYY". Anything else, at the
        moment, is a
     * bonus.
     *
     * @param s  string to parse.
     *
     * @return <code>null</code> if the string is not
        parseable, the
     *         week otherwise.
     */
    public static Week parseWeek(String s) {
        Week result = null;

        if (s != null) {

            // trim whitespace from either end of the
               string
            s = s.trim();

            int i = Week.findSeparator(s);
            if (i != -1) {
                String s1 = s.substring(0, i).trim();
```

```
560                 String s2 = s.substring(i + 1, s.length()).
                        trim();
561
562                 Year y = Week.evaluateAsYear(s1);
563                 int w;
564                 if (y != null) {
565                     w = Week.stringToWeek(s2);
566                     if (w == -1) {
567                         throw new TimePeriodFormatException
                                (
568                                 "Can't evaluate the week.")
                                    ;
569                     }
570                     result = new Week(w, y);
571                 } else {
572                     y = Week.evaluateAsYear(s2);
573                     if (y != null) {
574                         w = Week.stringToWeek(s1);
575                         if (w == -1) {
576                             throw new
                                    TimePeriodFormatException(
577                                         "Can't evaluate the
                                            week.");
578                         }
579                         result = new Week(w, y);
580                     } else {
581                         throw new TimePeriodFormatException
                                (
582                                 "Can't evaluate the year.")
                                    ;
583                     }
584                 }
585             } else {
586                 throw new TimePeriodFormatException(
587                         "Could not find separator.");
588             }
589         }
590         return result;
591     }
592
593     /**
594      * Finds the first occurrence of ' ', '-', ',' or '.'
595      *
596      * @param s  the string to parse.
597      *
598      * @return <code>-1</code> if none of the characters
```

```
         was found,
599      *         the index of the first occurrence otherwise.
600      */
601     private static int findSeparator(String s) {
602         int result = s.indexOf('-');
603 
604         if (result == -1) {
605             result = s.indexOf(',');
606         }
607         if (result == -1) {
608             result = s.indexOf(' ');
609         }
610         if (result == -1) {
611             result = s.indexOf('.');
612         }
613         return result;
614     }
615 
616     /**
617      * Creates a year from a string, or returns null (
            format
618      * exceptions suppressed).
619      *
620      * @param s  string to parse.
621      *
622      * @return <code>null</code> if the string is not
            parseable, the
623      *         year otherwise.
624      */
625     private static Year evaluateAsYear(String s) {
626         Year result = null;
627 
628         try {
629             result = Year.parseYear(s);
630         } catch (TimePeriodFormatException e) {
631 
632             // suppress
633         }
634         return result;
635     }
636 
637     /**
638      * Converts a string to a week.
639      *
640      * @param s  the string to parse.
641      * @return <code>-1</code> if the string does not
```

```
         *          contain a week
642      *          number, the number of the week otherwise.
643      */
644     private static int stringToWeek(String s) {
645         int result = -1;
646 
647         s = s.replace('W', ' ');
648         s = s.trim();
649         try {
650             result = Integer.parseInt(s);
651             if ((result < 1) || (result > LAST_WEEK_IN_YEAR
                    )) {
652                 result = -1;
653             }
654         } catch (NumberFormatException e) {
655 
656             // suppress
657         }
658         return result;
659     }
660 }
```

E.2.2 Defects-Sources/Chart-8/Week-fixed-A.java

```
1  /*
     ===========================================================

2   * JFreeChart : a free chart library for the Java(tm)
       platform
3   *
       ===========================================================

4   *
5   * (C) Copyright 2000-2008, by Object Refinery Limited and
6   * Contributors.
7   *
8   * Project Info:  http://www.jfree.org/jfreechart/index.
       html
9   *
10  * This library is free software; you can redistribute it
       and/or
11  * modify it under the terms of the GNU Lesser General
       Public License
12  * as published by the Free Software Foundation; either
       version 2.1 of
13  * the License, or (at your option) any later version.
14  *
15  * This library is distributed in the hope that it will be
       useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty
       of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
         the GNU
18  * Lesser General Public License for more details.
19  *
20  * You should have received a copy of the GNU Lesser
       General Public
21  * License along with this library; if not, write to the
       Free Software
22  * Foundation, Inc.,
23  * 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
         USA.
24  *
25  * [Java is a trademark or registered trademark of
26  * Sun Microsystems, Inc. in the United States and other
       countries.]
27  *
28  * ---------
```

```
29 * Week.java
30 * ---------
31 * (C) Copyright 2001-2008, by Object Refinery Limited and
32 * Contributors.
33 *
34 * Original Author:  David Gilbert (for Object Refinery
     Limited);
35 * Contributor(s):   Aimin Han;
36 *
37 * Changes
38 * -------
39 * 11-Oct-2001 : Version 1 (DG);
40 * 18-Dec-2001 : Changed order of parameters in constructor
       (DG);
41 * 19-Dec-2001 : Added a new constructor as suggested by
     Paul English
42 *               (DG);
43 * 29-Jan-2002 : Worked on the parseWeek() method (DG);
44 * 13-Feb-2002 : Fixed bug in Week(Date) constructor (DG);
45 * 26-Feb-2002 : Changed getStart(), getMiddle() and getEnd
     () methods
46 *               to evaluate with reference to a particular
       time zone
47 *               (DG);
48 * 05-Apr-2002 : Reinstated this class to the JCommon
     library (DG);
49 * 24-Jun-2002 : Removed unnecessary main method (DG);
50 * 10-Sep-2002 : Added getSerialIndex() method (DG);
51 * 06-Oct-2002 : Fixed errors reported by Checkstyle (DG);
52 * 18-Oct-2002 : Changed to observe 52 or 53 weeks per year
     ,
53 *               consistent with GregorianCalendar. Thanks
     to
54 *               Aimin Han for the code (DG);
55 * 02-Jan-2003 : Removed debug code (DG);
56 * 13-Mar-2003 : Moved to com.jrefinery.data.time package,
     and
57 *               implemented Serializable (DG);
58 * 21-Oct-2003 : Added hashCode() method (DG);
59 * 24-May-2004 : Modified getFirstMillisecond() and
60 *               getLastMillisecond() to take account of
61 *               firstDayOfWeek setting in Java's Calendar
     class (DG);
62 * 30-Sep-2004 : Replaced getTime().getTime() with
     getTimeInMillis()
63 *               (DG);
```

```
64  * 04-Nov-2004 : Reverted change of 30-Sep-2004, because it
        won't work
65  *                for JDK 1.3 (DG);
66  * ------------- JFREECHART 1.0.x
        ------------------------------------
67  * 06-Mar-2006 : Fix for bug 1448828, incorrect calculation
        of week
68  *                and year for the first few days of some
        years (DG);
69  * 05-Oct-2006 : Updated API docs (DG);
70  * 06-Oct-2006 : Refactored to cache first and last
        millisecond values
71  *                (DG);
72  * 09-Jan-2007 : Fixed bug in next() (DG);
73  * 28-Aug-2007 : Added new constructor to avoid problem in
        creating
74  *                new instances (DG);
75  * 19-Dec-2007 : Fixed bug in deprecated constructor (DG);
76  *
77  */
78
79
80 package org.jfree.data.time;
81
82
83 import java.io.Serializable;
84 import java.util.Calendar;
85 import java.util.Date;
86 import java.util.Locale;
87 import java.util.TimeZone;
88
89
90 /**
91  * A calendar week.  All years are considered to have 53
        weeks,
92  * numbered from 1 to 53, although in many cases the 53rd
        week is
93  * empty.  Most of the time, the 1st week of the year *
        begins* in the
94  * previous calendar year, but it always finishes in the
        current year
95  * (this behaviour matches the workings of the
96  * <code>GregorianCalendar</code> class).
97  * <P>
98  * This class is immutable, which is a requirement for all
99  * {@link RegularTimePeriod} subclasses.
```

```
100  */
101 public class Week extends RegularTimePeriod implements
       Serializable {
102
103
104     /** For serialization. */
105     private static final long serialVersionUID =
           1856387786939865061L;
106
107     /** Constant for the first week in the year. */
108     public static final int FIRST_WEEK_IN_YEAR = 1;
109
110     /** Constant for the last week in the year. */
111     public static final int LAST_WEEK_IN_YEAR = 53;
112
113     /** The year in which the week falls. */
114     private short year;
115
116     /** The week (1-53). */
117     private byte week;
118
119     /** The first millisecond. */
120     private long firstMillisecond;
121
122     /** The last millisecond. */
123     private long lastMillisecond;
124
125
126     /**
127      * Creates a new time period for the week in which the
           current
128      * system date/time falls.
129      */
130     public Week() {
131         this(new Date());
132     }
133
134     /**
135      * Creates a time period representing the week in the
           specified
136      * year.
137      *
138      * @param week  the week (1 to 53).
139      * @param year  the year (1900 to 9999).
140      */
141     public Week(int week, int year) {
```

```
142         if ((week < FIRST_WEEK_IN_YEAR) && (week >
               LAST_WEEK_IN_YEAR)) {
143             throw new IllegalArgumentException(
144                     "The 'week' argument must be in the
                           range 1 - 53.");
145         }
146         this.week = (byte) week;
147         this.year = (short) year;
148         peg(Calendar.getInstance());
149     }
150 
151     /**
152      * Creates a time period representing the week in the
               specified
153      * year.
154      *
155      * @param week  the week (1 to 53).
156      * @param year  the year (1900 to 9999).
157      */
158     public Week(int week, Year year) {
159         if ((week < FIRST_WEEK_IN_YEAR) && (week >
               LAST_WEEK_IN_YEAR)) {
160             throw new IllegalArgumentException(
161                     "The 'week' argument must be in the
                           range 1 - 53.");
162         }
163         this.week = (byte) week;
164         this.year = (short) year.getYear();
165         peg(Calendar.getInstance());
166     }
167 
168     /**
169      * Creates a time period for the week in which the
               specified
170      * date/time falls.
171      *
172      * @param time  the time (<code>null</code> not
               permitted).
173      */
174     public Week(Date time) {
175 
176         // defer argument checking...
177         this(time, RegularTimePeriod.DEFAULT_TIME_ZONE,
               Locale.getDefault());
178     }
179 
```

```
180     /**
181      * Creates a time period for the week in which the
             specified
182      * date/time falls, calculated relative to the
             specified time
183      * zone.
184      *
185      * @param time  the date/time (<code>null</code> not
             permitted).
186      * @param zone  the time zone (<code>null</code> not
             permitted).
187      *
188      * @deprecated As of 1.0.7, use
189      *             {@link #Week(Date, TimeZone, Locale)}.
190      */
191     public Week(Date time, TimeZone zone) {
192
193         // defer argument checking...
194         this(time, zone, Locale.getDefault());
195     }
196
197     /**
198      * Creates a time period for the week in which the
             specified
199      * date/time falls, calculated relative to the
             specified time
200      * zone.
201      *
202      * @param time  the date/time (<code>null</code> not
             permitted).
203      * @param zone  the time zone (<code>null</code> not
             permitted).
204      * @param locale  the locale (<code>null</code> not
             permitted).
205      *
206      * @since 1.0.7
207      */
208     public Week(Date time, TimeZone zone, Locale locale) {
209         if (time == null) {
210             throw new IllegalArgumentException("Null 'time'
                    argument.");
211         }
212         if (zone == null) {
213             throw new IllegalArgumentException("Null 'zone'
                    argument.");
214         }
```

```
215         if (locale == null) {
216             throw new IllegalArgumentException("Null '
                locale' argument.");
217         }
218         Calendar calendar = Calendar.getInstance(zone,
               locale);
219         calendar.setTime(time);
220 
221         // sometimes the last few days of the year are
               considered to fall in
222         // the *first* week of the following year.  Refer
               to the Javadocs for
223         // GregorianCalendar.
224         int tempWeek = calendar.get(Calendar.WEEK_OF_YEAR);
225         if (tempWeek == 1
226                 && calendar.get(Calendar.MONTH) == Calendar
                       .DECEMBER) {
227             this.week = 1;
228             this.year = (short) (calendar.get(Calendar.YEAR
                    ) + 1);
229         } else {
230             this.week = (byte) Math.min(tempWeek,
                    LAST_WEEK_IN_YEAR);
231             int yyyy = calendar.get(Calendar.YEAR);
232 
233             // alternatively, sometimes the first few days
                    of the year are
234             // considered to fall in the *last* week of the
                     previous year...
235             if (calendar.get(Calendar.MONTH) == Calendar.
                    JANUARY
236                     && this.week >= 52) {
237                 yyyy--;
238             }
239             this.year = (short) yyyy;
240         }
241         peg(calendar);
242     }
243 
244 
245     /**
246      * Returns the year in which the week falls.
247      *
248      * @return The year (never <code>null</code>).
249      */
250     public Year getYear() {
```

```
251         return new Year(this.year);
252     }
253 
254     /**
255      * Returns the year in which the week falls, as an
            integer value.
256      *
257      * @return The year.
258      */
259     public int getYearValue() {
260         return this.year;
261     }
262 
263     /**
264      * Returns the week.
265      *
266      * @return The week.
267      */
268     public int getWeek() {
269         return this.week;
270     }
271 
272     /**
273      * Returns the first millisecond of the week.  This
            will be
274      * determined relative to the time zone specified in
            the
275      * constructor, or in the calendar instance passed in
            the most
276      * recent call to the {@link #peg(Calendar)} method.
277      *
278      * @return The first millisecond of the week.
279      *
280      * @see #getLastMillisecond()
281      */
282     public long getFirstMillisecond() {
283         return this.firstMillisecond;
284     }
285 
286     /**
287      * Returns the last millisecond of the week.  This will
             be
288      * determined relative to the time zone specified in
            the
289      * constructor, or in the calendar instance passed in
            the most
```

```
290         * recent call to the {@link #peg(Calendar)} method.
291         *
292         * @return The last millisecond of the week.
293         *
294         * @see #getFirstMillisecond()
295         */
296        public long getLastMillisecond() {
297            return this.lastMillisecond;
298        }
299
300        /**
301         * Recalculates the start date/time and end date/time
                for this
302         * time period relative to the supplied calendar (which
303         * incorporates a time zone).
304         *
305         * @param calendar  the calendar (<code>null</code> not
306         *                  permitted).
307         *
308         * @since 1.0.3
309         */
310        public void peg(Calendar calendar) {
311            this.firstMillisecond = getFirstMillisecond(
                    calendar);
312            this.lastMillisecond = getLastMillisecond(calendar)
                    ;
313        }
314
315        /**
316         * Returns the week preceding this one.  This method
                will return
317         * <code>null</code> for some lower limit on the range
                of weeks
318         * (currently week 1, 1900).  For week 1 of any year,
                the previous
319         * week is always week 53, but week 53 may not contain
                any days
320         * (you should check for this).
321         *
322         * @return The preceding week (possibly <code>null</
                code>).
323         */
324        public RegularTimePeriod previous() {
325            Week result;
326
327            if (this.week != FIRST_WEEK_IN_YEAR) {
```

```
328             result = new Week(this.week - 1, this.year);
329         } else {
330
331             // we need to work out if the previous year has
                    52 or 53 weeks...
332             if (this.year > 1900) {
333                 int yy = this.year - 1;
334                 Calendar prevYearCalendar = Calendar.
                        getInstance();
335                 prevYearCalendar.set(yy, Calendar.DECEMBER,
                        31);
336                 result = new Week(prevYearCalendar.
                        getActualMaximum(
337                             Calendar.WEEK_OF_YEAR), yy);
338             } else {
339                 result = null;
340             }
341         }
342         return result;
343     }
344
345     /**
346      * Returns the week following this one.  This method
            will return
347      * <code>null</code> for some upper limit on the range
            of weeks
348      * (currently week 53, 9999).  For week 52 of any year,
             the
349      * following week is always week 53, but week 53 may
            not contain
350      * any days (you should check for this).
351      *
352      * @return The following week (possibly <code>null</
            code>).
353      */
354     public RegularTimePeriod next() {
355         Week result;
356
357         if (this.week < 52) {
358             result = new Week(this.week + 1, this.year);
359         } else {
360             Calendar calendar = Calendar.getInstance();
361             calendar.set(this.year, Calendar.DECEMBER, 31);
362             int actualMaxWeek
363                 = calendar.getActualMaximum(Calendar.
                        WEEK_OF_YEAR);
```

```
364             if (this.week < actualMaxWeek) {
365                 result = new Week(this.week + 1, this.year)
                        ;
366             } else {
367                 if (this.year < 9999) {
368                     result = new Week(FIRST_WEEK_IN_YEAR,
                            this.year + 1);
369                 } else {
370                     result = null;
371                 }
372             }
373         }
374         return result;
375     }
376 
377     /**
378      * Returns a serial index number for the week.
379      *
380      * @return The serial index number.
381      */
382     public long getSerialIndex() {
383         return this.year * 53L + this.week;
384     }
385 
386     /**
387      * Returns the first millisecond of the week, evaluated
             using the
388      * supplied calendar (which determines the time zone).
389      *
390      * @param calendar  the calendar (<code>null</code> not
391      *                  permitted).
392      *
393      * @return The first millisecond of the week.
394      *
395      * @throws NullPointerException if <code>calendar</code
             > is
396      *     <code>null</code>.
397      */
398     public long getFirstMillisecond(Calendar calendar) {
399         Calendar c = (Calendar) calendar.clone();
400 
401         c.clear();
402         c.set(Calendar.YEAR, this.year);
403         c.set(Calendar.WEEK_OF_YEAR, this.week);
404         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
405         c.set(Calendar.HOUR, 0);
```

```
406         c.set(Calendar.MINUTE, 0);
407         c.set(Calendar.SECOND, 0);
408         c.set(Calendar.MILLISECOND, 0);
409 
410         //return c.getTimeInMillis();  // this won't work
                for JDK 1.3
411         return c.getTime().getTime();
412     }
413 
414     /**
415      * Returns the last millisecond of the week, evaluated
            using the
416      * supplied calendar (which determines the time zone).
417      *
418      * @param calendar  the calendar (<code>null</code> not
419      *                  permitted).
420      *
421      * @return The last millisecond of the week.
422      *
423      * @throws NullPointerException if <code>calendar</code
            > is
424      *     <code>null</code>.
425      */
426     public long getLastMillisecond(Calendar calendar) {
427         Calendar c = (Calendar) calendar.clone();
428 
429         c.clear();
430         c.set(Calendar.YEAR, this.year);
431         c.set(Calendar.WEEK_OF_YEAR, this.week + 1);
432         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
433         c.set(Calendar.HOUR, 0);
434         c.set(Calendar.MINUTE, 0);
435         c.set(Calendar.SECOND, 0);
436         c.set(Calendar.MILLISECOND, 0);
437 
438         //return c.getTimeInMillis();  // this won't work
                for JDK 1.3
439         return c.getTime().getTime() - 1;
440     }
441 
442     /**
443      * Returns a string representing the week (e.g. "Week
            9, 2002").
444      *
445      * TODO: look at internationalisation.
446      *
```

```
447         * @return A string representing the week.
448         */
449        public String toString() {
450            return "Week " + this.week + ", " + this.year;
451        }
452
453        /**
454         * Tests the equality of this Week object to an
               arbitrary object.
455         * Returns true if the target is a Week instance
               representing the
456         * same week as this object.  In all other cases,
               returns false.
457         *
458         * @param obj  the object (<code>null</code> permitted)
               .
459         *
460         * @return <code>true</code> if week and year of this
               and object
461         *         are the same.
462         */
463        public boolean equals(Object obj) {
464            if (obj == this) {
465                return true;
466            }
467            if (!(obj instanceof Week)) {
468                return false;
469            }
470            Week that = (Week) obj;
471            if (this.week != that.week) {
472                return false;
473            }
474            if (this.year != that.year) {
475                return false;
476            }
477            return true;
478        }
479
480        /**
481         * Returns a hash code for this object instance.  The
               approach
482         * described by Joshua Bloch in "Effective Java" has
               been used
483         * here:
484         * <p>
485         * <code>
```

```
486         * http://developer.java.sun.com/developer/Books/
                effectivejava
487         * /Chapter3.pdf</code>
488         *
489         * @return A hash code.
490         */
491        public int hashCode() {
492            int result = 17;
493
494            result = 37 * result + this.week;
495            result = 37 * result + this.year;
496            return result;
497        }
498
499        /**
500         * Returns an integer indicating the order of this Week
                object
501         * relative to the specified object:
502         *
503         * negative == before, zero == same, positive == after.
504         *
505         * @param o1  the object to compare.
506         *
507         * @return negative == before, zero == same, positive
                == after.
508         */
509        public int compareTo(Object o1) {
510            int result;
511
512            // CASE 1 : Comparing to another Week object
513            // -----------------------------------------
514            if (o1 instanceof Week) {
515                Week w = (Week) o1;
516                result = this.year - w.getYear().getYear();
517                if (result == 0) {
518                    result = this.week - w.getWeek();
519                }
520            } else if (o1 instanceof RegularTimePeriod) {
521
522                // CASE 2 : Comparing to another TimePeriod
                       object
523                //
                       ------------------------------------------------

524                // more difficult case - evaluate later...
525                result = 0;
```

```
526         } else {
527
528             // CASE 3 : Comparing to a non-TimePeriod
                    object
529             //
                    ---------------------------------------------

530             // consider time periods to be ordered after
                    general objects
531             result = 1;
532         }
533
534         return result;
535     }
536
537     /**
538      * Parses the string argument as a week.
539      * <P>
540      * This method is required to accept the format "YYYY-
                Wnn".  It
541      * will also accept "Wnn-YYYY". Anything else, at the
                moment, is a
542      * bonus.
543      *
544      * @param s  string to parse.
545      *
546      * @return <code>null</code> if the string is not
                parseable, the
547      *         week otherwise.
548      */
549     public static Week parseWeek(String s) {
550         Week result = null;
551
552         if (s != null) {
553
554             // trim whitespace from either end of the
                    string
555             s = s.trim();
556
557             int i = Week.findSeparator(s);
558             if (i != -1) {
559                 String s1 = s.substring(0, i).trim();
560                 String s2 = s.substring(i + 1, s.length()).
                        trim();
561
562                 Year y = Week.evaluateAsYear(s1);
```

```
563                 int w;
564                 if (y != null) {
565                     w = Week.stringToWeek(s2);
566                     if (w == -1) {
567                         throw new TimePeriodFormatException
                               (
568                                 "Can't evaluate the week.")
                                    ;
569                     }
570                     result = new Week(w, y);
571                 } else {
572                     y = Week.evaluateAsYear(s2);
573                     if (y != null) {
574                         w = Week.stringToWeek(s1);
575                         if (w == -1) {
576                             throw new
                                   TimePeriodFormatException(
577                                     "Can't evaluate the
                                        week.");
578                         }
579                         result = new Week(w, y);
580                     } else {
581                         throw new TimePeriodFormatException
                               (
582                                 "Can't evaluate the year.")
                                    ;
583                     }
584                 }
585             } else {
586                 throw new TimePeriodFormatException(
587                         "Could not find separator.");
588             }
589         }
590         return result;
591     }
592
593     /**
594      * Finds the first occurrence of ' ', '-', ',' or '.'
595      *
596      * @param s  the string to parse.
597      *
598      * @return <code>-1</code> if none of the characters
           was found,
599      *       the index of the first occurrence otherwise.
600      */
601     private static int findSeparator(String s) {
```

```
602         int result = s.indexOf('-');
603 
604         if (result == -1) {
605             result = s.indexOf(',');
606         }
607         if (result == -1) {
608             result = s.indexOf(' ');
609         }
610         if (result == -1) {
611             result = s.indexOf('.');
612         }
613         return result;
614     }
615 
616     /**
617      * Creates a year from a string, or returns null (
            format
618      * exceptions suppressed).
619      *
620      * @param s  string to parse.
621      *
622      * @return <code>null</code> if the string is not
            parseable, the
623      *         year otherwise.
624      */
625     private static Year evaluateAsYear(String s) {
626         Year result = null;
627 
628         try {
629             result = Year.parseYear(s);
630         } catch (TimePeriodFormatException e) {
631 
632             // suppress
633         }
634         return result;
635     }
636 
637     /**
638      * Converts a string to a week.
639      *
640      * @param s  the string to parse.
641      * @return <code>-1</code> if the string does not
            contain a week
642      *         number, the number of the week otherwise.
643      */
644     private static int stringToWeek(String s) {
```

```
645         int result = -1;
646 
647         s = s.replace('W', ' ');
648         s = s.trim();
649         try {
650             result = Integer.parseInt(s);
651             if ((result < 1) || (result > LAST_WEEK_IN_YEAR
                    )) {
652                 result = -1;
653             }
654         } catch (NumberFormatException e) {
655 
656             // suppress
657         }
658         return result;
659     }
660 }
```

E.2.3 Defects-Sources/Chart-8/Week-bugged-B.java

```
1  /*
     ===========================================================

2   * JFreeChart : a free chart library for the Java(tm)
       platform
3   *
       ===========================================================

4   *
5   * (C) Copyright 2000-2008, by Object Refinery Limited and
       Contributors.
6   *
7   * Project Info:  http://www.jfree.org/jfreechart/index.
       html
8   *
9   * This library is free software; you can redistribute it
       and/or modify it
10  * under the terms of the GNU Lesser General Public License
        as published by
11  * the Free Software Foundation; either version 2.1 of the
       License, or
12  * (at your option) any later version.
13  *
14  * This library is distributed in the hope that it will be
       useful, but
15  * WITHOUT ANY WARRANTY; without even the implied warranty
       of MERCHANTABILITY
16  * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser
       General Public
17  * License for more details.
18  *
19  * You should have received a copy of the GNU Lesser
       General Public
20  * License along with this library; if not, write to the
       Free Software
21  * Foundation, Inc., 51 Franklin Street, Fifth Floor,
       Boston, MA  02110-1301,
22  * USA.
23  *
24  * [Java is a trademark or registered trademark of Sun
       Microsystems, Inc.
25  * in the United States and other countries.]
26  *
27  * ---------
```

```
28 * Week.java
29 * ---------
30 * (C) Copyright 2001-2008, by Object Refinery Limited and
       Contributors.
31 *
32 * Original Author:  David Gilbert (for Object Refinery
       Limited);
33 * Contributor(s):   Aimin Han;
34 *
35 * Changes
36 * -------
37 * 11-Oct-2001 : Version 1 (DG);
38 * 18-Dec-2001 : Changed order of parameters in constructor
       (DG);
39 * 19-Dec-2001 : Added a new constructor as suggested by
       Paul English (DG);
40 * 29-Jan-2002 : Worked on the parseWeek() method (DG);
41 * 13-Feb-2002 : Fixed bug in Week(Date) constructor (DG);
42 * 26-Feb-2002 : Changed getStart(), getMiddle() and getEnd
       () methods to
43 *               evaluate with reference to a particular
       time zone (DG);
44 * 05-Apr-2002 : Reinstated this class to the JCommon
       library (DG);
45 * 24-Jun-2002 : Removed unnecessary main method (DG);
46 * 10-Sep-2002 : Added getSerialIndex() method (DG);
47 * 06-Oct-2002 : Fixed errors reported by Checkstyle (DG);
48 * 18-Oct-2002 : Changed to observe 52 or 53 weeks per year
       , consistent with
49 *               GregorianCalendar. Thanks to Aimin Han for
       the code (DG);
50 * 02-Jan-2003 : Removed debug code (DG);
51 * 13-Mar-2003 : Moved to com.jrefinery.data.time package,
       and implemented
52 *               Serializable (DG);
53 * 21-Oct-2003 : Added hashCode() method (DG);
54 * 24-May-2004 : Modified getFirstMillisecond() and
       getLastMillisecond() to
55 *               take account of firstDayOfWeek setting in
       Java's Calendar
56 *               class (DG);
57 * 30-Sep-2004 : Replaced getTime().getTime() with
       getTimeInMillis() (DG);
58 * 04-Nov-2004 : Reverted change of 30-Sep-2004, because it
       won't work for
59 *               JDK 1.3 (DG);
```

```
60  * ------------- JFREECHART 1.0.x
        ---------------------------------------------
61  * 06-Mar-2006 : Fix for bug 1448828, incorrect calculation
        of week and year
62  *                for the first few days of some years (DG);
63  * 05-Oct-2006 : Updated API docs (DG);
64  * 06-Oct-2006 : Refactored to cache first and last
        millisecond values (DG);
65  * 09-Jan-2007 : Fixed bug in next() (DG);
66  * 28-Aug-2007 : Added new constructor to avoid problem in
        creating new
67  *                instances (DG);
68  * 19-Dec-2007 : Fixed bug in deprecated constructor (DG);
69  *
70  */
71
72 package org.jfree.data.time;
73
74 import java.io.Serializable;
75 import java.util.Calendar;
76 import java.util.Date;
77 import java.util.Locale;
78 import java.util.TimeZone;
79
80 /**
81  * A calendar week.  All years are considered to have 53
        weeks, numbered from 1
82  * to 53, although in many cases the 53rd week is empty.
        Most of the time, the
83  * 1st week of the year *begins* in the previous calendar
        year, but it always
84  * finishes in the current year (this behaviour matches the
        workings of the
85  * <code>GregorianCalendar</code> class).
86  * <P>
87  * This class is immutable, which is a requirement for all
88  * {@link RegularTimePeriod} subclasses.
89  */
90 public class Week extends RegularTimePeriod implements
       Serializable
91 {
92
93     /** For serialization. */
94     private static final long serialVersionUID =
           1856387786939865061L;
95
```

```
96      /** Constant for the first week in the year. */
97      public static final int FIRST_WEEK_IN_YEAR = 1;
98
99      /** Constant for the last week in the year. */
100     public static final int LAST_WEEK_IN_YEAR = 53;
101
102     /** The year in which the week falls. */
103     private short year;
104
105     /** The week (1-53). */
106     private byte week;
107
108     /** The first millisecond. */
109     private long firstMillisecond;
110
111     /** The last millisecond. */
112     private long lastMillisecond;
113
114     /**
115      * Creates a new time period for the week in which the
            current system
116      * date/time falls.
117      */
118     public Week()
119     {
120         this(new Date());
121     }
122
123     /**
124      * Creates a time period representing the week in the
            specified year.
125      *
126      * @param week  the week (1 to 53).
127      * @param year  the year (1900 to 9999).
128      */
129     public Week(int week, int year)
130     {
131         if ((week < FIRST_WEEK_IN_YEAR) && (week >
               LAST_WEEK_IN_YEAR))
132         {
133             throw new IllegalArgumentException(
134                     "The 'week' argument must be in the
                           range 1 - 53.");
135         }
136         this.week = (byte) week;
137         this.year = (short) year;
```

```
138         peg(Calendar.getInstance());
139     }
140
141     /**
142      * Creates a time period representing the week in the
            specified year.
143      *
144      * @param week  the week (1 to 53).
145      * @param year  the year (1900 to 9999).
146      */
147     public Week(int week, Year year)
148     {
149         if ((week < FIRST_WEEK_IN_YEAR) && (week >
                LAST_WEEK_IN_YEAR))
150         {
151             throw new IllegalArgumentException(
152                     "The 'week' argument must be in the
                        range 1 - 53.");
153         }
154         this.week = (byte) week;
155         this.year = (short) year.getYear();
156         peg(Calendar.getInstance());
157     }
158
159     /**
160      * Creates a time period for the week in which the
            specified date/time
161      * falls.
162      *
163      * @param time  the time (<code>null</code> not
            permitted).
164      */
165     public Week(Date time)
166     {
167         // defer argument checking...
168         this(time, RegularTimePeriod.DEFAULT_TIME_ZONE,
                Locale.getDefault());
169     }
170
171     /**
172      * Creates a time period for the week in which the
            specified date/time
173      * falls, calculated relative to the specified time
            zone.
174      *
175      * @param time  the date/time (<code>null</code> not
```

```
         permitted).
176      * @param zone  the time zone (<code>null</code> not
             permitted).
177      *
178      * @deprecated As of 1.0.7, use {@link #Week(Date,
             TimeZone, Locale)}.
179      */
180     public Week(Date time, TimeZone zone)
181     {
182         // defer argument checking...
183         this(time, RegularTimePeriod.DEFAULT_TIME_ZONE,
                Locale.getDefault());
184     }
185
186     /**
187      * Creates a time period for the week in which the
             specified date/time
188      * falls, calculated relative to the specified time
             zone.
189      *
190      * @param time  the date/time (<code>null</code> not
             permitted).
191      * @param zone  the time zone (<code>null</code> not
             permitted).
192      * @param locale  the locale (<code>null</code> not
             permitted).
193      *
194      * @since 1.0.7
195      */
196     public Week(Date time, TimeZone zone, Locale locale)
197     {
198         if (time == null)
199         {
200             throw new IllegalArgumentException("Null 'time'
                    argument.");
201         }
202         if (zone == null)
203         {
204             throw new IllegalArgumentException("Null 'zone'
                    argument.");
205         }
206         if (locale == null)
207         {
208             throw new IllegalArgumentException("Null '
                   locale' argument.");
209         }
```

```
210         Calendar calendar = Calendar.getInstance(zone,
               locale);
211         calendar.setTime(time);
212
213         // sometimes the last few days of the year are
               considered to fall in
214         // the *first* week of the following year.  Refer
               to the Javadocs for
215         // GregorianCalendar.
216         int tempWeek = calendar.get(Calendar.WEEK_OF_YEAR);
217         if (tempWeek == 1
218                 && calendar.get(Calendar.MONTH) == Calendar
                       .DECEMBER)
219         {
220             this.week = 1;
221             this.year = (short) (calendar.get(Calendar.YEAR
                   ) + 1);
222         }
223         else
224         {
225             this.week = (byte) Math.min(tempWeek,
                   LAST_WEEK_IN_YEAR);
226             int yyyy = calendar.get(Calendar.YEAR);
227             // alternatively, sometimes the first few days
                   of the year are
228             // considered to fall in the *last* week of the
                    previous year...
229             if (calendar.get(Calendar.MONTH) == Calendar.
                   JANUARY
230                     && this.week >= 52)
231             {
232                 yyyy--;
233             }
234             this.year = (short) yyyy;
235         }
236         peg(calendar);
237     }
238
239     /**
240      * Returns the year in which the week falls.
241      *
242      * @return The year (never <code>null</code>).
243      */
244     public Year getYear()
245     {
246         return new Year(this.year);
```

```
247     }
248
249     /**
250      * Returns the year in which the week falls, as an
            integer value.
251      *
252      * @return The year.
253      */
254     public int getYearValue()
255     {
256         return this.year;
257     }
258
259     /**
260      * Returns the week.
261      *
262      * @return The week.
263      */
264     public int getWeek()
265     {
266         return this.week;
267     }
268
269     /**
270      * Returns the first millisecond of the week.  This
            will be determined
271      * relative to the time zone specified in the
            constructor, or in the
272      * calendar instance passed in the most recent call to
            the
273      * {@link #peg(Calendar)} method.
274      *
275      * @return The first millisecond of the week.
276      *
277      * @see #getLastMillisecond()
278      */
279     public long getFirstMillisecond()
280     {
281         return this.firstMillisecond;
282     }
283
284     /**
285      * Returns the last millisecond of the week.  This will
             be
286      * determined relative to the time zone specified in
            the constructor, or
```

```
287         * in the calendar instance passed in the most recent
                call to the
288         * {@link #peg(Calendar)} method.
289         *
290         * @return The last millisecond of the week.
291         *
292         * @see #getFirstMillisecond()
293         */
294        public long getLastMillisecond()
295        {
296            return this.lastMillisecond;
297        }
298
299        /**
300         * Recalculates the start date/time and end date/time
                for this time period
301         * relative to the supplied calendar (which
                incorporates a time zone).
302         *
303         * @param calendar  the calendar (<code>null</code> not
                permitted).
304         *
305         * @since 1.0.3
306         */
307        public void peg(Calendar calendar)
308        {
309            this.firstMillisecond = getFirstMillisecond(
                    calendar);
310            this.lastMillisecond = getLastMillisecond(calendar)
                    ;
311        }
312
313        /**
314         * Returns the week preceding this one.  This method
                will return
315         * <code>null</code> for some lower limit on the range
                of weeks (currently
316         * week 1, 1900).  For week 1 of any year, the previous
                week is always week
317         * 53, but week 53 may not contain any days (you should
                check for this).
318         *
319         * @return The preceding week (possibly <code>null</
                code>).
320         */
321        public RegularTimePeriod previous()
```

```
322     {
323
324         Week result;
325         if (this.week != FIRST_WEEK_IN_YEAR)
326         {
327             result = new Week(this.week - 1, this.year);
328         }
329         else
330         {
331             // we need to work out if the previous year has
                    52 or 53 weeks...
332             if (this.year > 1900)
333             {
334                 int yy = this.year - 1;
335                 Calendar prevYearCalendar = Calendar.
                       getInstance();
336                 prevYearCalendar.set(yy, Calendar.DECEMBER,
                        31);
337                 result = new Week(prevYearCalendar.
                       getActualMaximum(
338                         Calendar.WEEK_OF_YEAR), yy);
339             }
340             else
341             {
342                 result = null;
343             }
344         }
345         return result;
346
347     }
348
349     /**
350      * Returns the week following this one.  This method
            will return
351      * <code>null</code> for some upper limit on the range
            of weeks (currently
352      * week 53, 9999).  For week 52 of any year, the
            following week is always
353      * week 53, but week 53 may not contain any days (you
            should check for
354      * this).
355      *
356      * @return The following week (possibly <code>null</
            code>).
357      */
358     public RegularTimePeriod next()
```

```
359     {
360
361         Week result;
362         if (this.week < 52)
363         {
364             result = new Week(this.week + 1, this.year);
365         }
366         else
367         {
368             Calendar calendar = Calendar.getInstance();
369             calendar.set(this.year, Calendar.DECEMBER, 31);
370             int actualMaxWeek
371                 = calendar.getActualMaximum(Calendar.
                       WEEK_OF_YEAR);
372             if (this.week < actualMaxWeek)
373             {
374                 result = new Week(this.week + 1, this.year)
                       ;
375             }
376             else
377             {
378                 if (this.year < 9999)
379                 {
380                     result = new Week(FIRST_WEEK_IN_YEAR,
                           this.year + 1);
381                 }
382                 else
383                 {
384                     result = null;
385                 }
386             }
387         }
388         return result;
389
390     }
391
392     /**
393      * Returns a serial index number for the week.
394      *
395      * @return The serial index number.
396      */
397     public long getSerialIndex()
398     {
399         return this.year * 53L + this.week;
400     }
401
```

```
402     /**
403      * Returns the first millisecond of the week, evaluated
             using the supplied
404      * calendar (which determines the time zone).
405      *
406      * @param calendar  the calendar (<code>null</code> not
             permitted).
407      *
408      * @return The first millisecond of the week.
409      *
410      * @throws NullPointerException if <code>calendar</code
             > is
411      *     <code>null</code>.
412      */
413     public long getFirstMillisecond(Calendar calendar)
414     {
415         Calendar c = (Calendar) calendar.clone();
416         c.clear();
417         c.set(Calendar.YEAR, this.year);
418         c.set(Calendar.WEEK_OF_YEAR, this.week);
419         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
420         c.set(Calendar.HOUR, 0);
421         c.set(Calendar.MINUTE, 0);
422         c.set(Calendar.SECOND, 0);
423         c.set(Calendar.MILLISECOND, 0);
424         //return c.getTimeInMillis();  // this won't work
             for JDK 1.3
425         return c.getTime().getTime();
426     }
427
428     /**
429      * Returns the last millisecond of the week, evaluated
             using the supplied
430      * calendar (which determines the time zone).
431      *
432      * @param calendar  the calendar (<code>null</code> not
             permitted).
433      *
434      * @return The last millisecond of the week.
435      *
436      * @throws NullPointerException if <code>calendar</code
             > is
437      *     <code>null</code>.
438      */
439     public long getLastMillisecond(Calendar calendar)
440     {
```

```
441         Calendar c = (Calendar) calendar.clone();
442         c.clear();
443         c.set(Calendar.YEAR, this.year);
444         c.set(Calendar.WEEK_OF_YEAR, this.week + 1);
445         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
446         c.set(Calendar.HOUR, 0);
447         c.set(Calendar.MINUTE, 0);
448         c.set(Calendar.SECOND, 0);
449         c.set(Calendar.MILLISECOND, 0);
450         //return c.getTimeInMillis();  // this won't work
                for JDK 1.3
451         return c.getTime().getTime() - 1;
452     }
453
454     /**
455      * Returns a string representing the week (e.g. "Week
            9, 2002").
456      *
457      * TODO: look at internationalisation.
458      *
459      * @return A string representing the week.
460      */
461     public String toString()
462     {
463         return "Week " + this.week + ", " + this.year;
464     }
465
466     /**
467      * Tests the equality of this Week object to an
            arbitrary object.  Returns
468      * true if the target is a Week instance representing
            the same week as this
469      * object.  In all other cases, returns false.
470      *
471      * @param obj  the object (<code>null</code> permitted)
            .
472      *
473      * @return <code>true</code> if week and year of this
            and object are the
474      *         same.
475      */
476     public boolean equals(Object obj)
477     {
478
479         if (obj == this)
480         {
```

```
481             return true;
482         }
483         if (!(obj instanceof Week))
484         {
485             return false;
486         }
487         Week that = (Week) obj;
488         if (this.week != that.week)
489         {
490             return false;
491         }
492         if (this.year != that.year)
493         {
494             return false;
495         }
496         return true;
497 
498     }
499 
500     /**
501      * Returns a hash code for this object instance.  The
            approach described by
502      * Joshua Bloch in "Effective Java" has been used here:
503      * <p>
504      * <code>http://developer.java.sun.com/developer/Books/
            effectivejava
505      * /Chapter3.pdf</code>
506      *
507      * @return A hash code.
508      */
509     public int hashCode()
510     {
511         int result = 17;
512         result = 37 * result + this.week;
513         result = 37 * result + this.year;
514         return result;
515     }
516 
517     /**
518      * Returns an integer indicating the order of this Week
              object relative to
519      * the specified object:
520      *
521      * negative == before, zero == same, positive == after.
522      *
523      * @param o1  the object to compare.
```

```
524         *
525         * @return negative == before, zero == same, positive
                == after.
526         */
527        public int compareTo(Object o1)
528        {
529
530            int result;
531
532            // CASE 1 : Comparing to another Week object
533            // -----------------------------------------
534            if (o1 instanceof Week)
535            {
536                Week w = (Week) o1;
537                result = this.year - w.getYear().getYear();
538                if (result == 0)
539                {
540                    result = this.week - w.getWeek();
541                }
542            }
543
544            // CASE 2 : Comparing to another TimePeriod object
545            // -----------------------------------------------
546            else if (o1 instanceof RegularTimePeriod)
547            {
548                // more difficult case - evaluate later...
549                result = 0;
550            }
551
552            // CASE 3 : Comparing to a non-TimePeriod object
553            // ---------------------------------------------
554            else
555            {
556                // consider time periods to be ordered after
                       general objects
557                result = 1;
558            }
559
560            return result;
561
562        }
563
564        /**
565         * Parses the string argument as a week.
566         * <P>
567         * This method is required to accept the format "YYYY-
```

```
         Wnn".  It will also
568       * accept "Wnn-YYYY". Anything else, at the moment, is
             a bonus.
569       *
570       * @param s  string to parse.
571       *
572       * @return <code>null</code> if the string is not
             parseable, the week
573       *         otherwise.
574       */
575      public static Week parseWeek(String s)
576      {
577
578          Week result = null;
579          if (s != null)
580          {
581
582              // trim whitespace from either end of the
                    string
583              s = s.trim();
584
585              int i = Week.findSeparator(s);
586              if (i != -1)
587              {
588                  String s1 = s.substring(0, i).trim();
589                  String s2 = s.substring(i + 1, s.length()).
                        trim();
590
591                  Year y = Week.evaluateAsYear(s1);
592                  int w;
593                  if (y != null)
594                  {
595                      w = Week.stringToWeek(s2);
596                      if (w == -1)
597                      {
598                          throw new TimePeriodFormatException
                                (
599                                       "Can't evaluate the week.")
                                          ;
600                      }
601                      result = new Week(w, y);
602                  }
603                  else
604                  {
605                      y = Week.evaluateAsYear(s2);
606                      if (y != null)
```

```
607                     {
608                         w = Week.stringToWeek(s1);
609                         if (w == -1)
610                         {
611                             throw new
                                    TimePeriodFormatException(
612                                         "Can't evaluate the
                                            week.");
613                         }
614                         result = new Week(w, y);
615                     }
616                     else
617                     {
618                         throw new TimePeriodFormatException
                            (
619                                  "Can't evaluate the year.")
                                     ;
620                     }
621                 }
622
623             }
624             else
625             {
626                 throw new TimePeriodFormatException(
627                         "Could not find separator.");
628             }
629
630         }
631         return result;
632
633     }
634
635     /**
636      * Finds the first occurrence of ' ', '-', ',' or '.'
637      *
638      * @param s  the string to parse.
639      *
640      * @return <code>-1</code> if none of the characters
                was found, the
641      *         index of the first occurrence otherwise.
642      */
643     private static int findSeparator(String s)
644     {
645
646         int result = s.indexOf('-');
647         if (result == -1)
```

```
648         {
649             result = s.indexOf(',');
650         }
651         if (result == -1)
652         {
653             result = s.indexOf(' ');
654         }
655         if (result == -1)
656         {
657             result = s.indexOf('.');
658         }
659         return result;
660     }
661 
662     /**
663      * Creates a year from a string, or returns null (
           format exceptions
664      * suppressed).
665      *
666      * @param s  string to parse.
667      *
668      * @return <code>null</code> if the string is not
           parseable, the year
669      *         otherwise.
670      */
671     private static Year evaluateAsYear(String s)
672     {
673 
674         Year result = null;
675         try
676         {
677             result = Year.parseYear(s);
678         }
679         catch (TimePeriodFormatException e)
680         {
681             // suppress
682         }
683         return result;
684 
685     }
686 
687     /**
688      * Converts a string to a week.
689      *
690      * @param s  the string to parse.
691      * @return <code>-1</code> if the string does not
```

```
         contain a week number,
692       *          the number of the week otherwise.
693       */
694      private static int stringToWeek(String s)
695      {
696
697          int result = -1;
698          s = s.replace('W', ' ');
699          s = s.trim();
700          try
701          {
702              result = Integer.parseInt(s);
703              if ((result < 1) || (result > LAST_WEEK_IN_YEAR
                     ))
704              {
705                  result = -1;
706              }
707          }
708          catch (NumberFormatException e)
709          {
710              // suppress
711          }
712          return result;
713
714      }
715
716  }
```

E.2.4 Defects-Sources/Chart-8/Week-fixed-B.java

```
1  /*
     ===========================================================

2   * JFreeChart : a free chart library for the Java(tm)
       platform
3   *
       ===========================================================

4   *
5   * (C) Copyright 2000-2008, by Object Refinery Limited and
       Contributors.
6   *
7   * Project Info:  http://www.jfree.org/jfreechart/index.
       html
8   *
9   * This library is free software; you can redistribute it
       and/or modify it
10  * under the terms of the GNU Lesser General Public License
        as published by
11  * the Free Software Foundation; either version 2.1 of the
       License, or
12  * (at your option) any later version.
13  *
14  * This library is distributed in the hope that it will be
       useful, but
15  * WITHOUT ANY WARRANTY; without even the implied warranty
       of MERCHANTABILITY
16  * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser
       General Public
17  * License for more details.
18  *
19  * You should have received a copy of the GNU Lesser
       General Public
20  * License along with this library; if not, write to the
       Free Software
21  * Foundation, Inc., 51 Franklin Street, Fifth Floor,
       Boston, MA  02110-1301,
22  * USA.
23  *
24  * [Java is a trademark or registered trademark of Sun
       Microsystems, Inc.
25  * in the United States and other countries.]
26  *
27  * ---------
```

```
28 * Week.java
29 * ---------
30 * (C) Copyright 2001-2008, by Object Refinery Limited and
       Contributors.
31 *
32 * Original Author:  David Gilbert (for Object Refinery
       Limited);
33 * Contributor(s):   Aimin Han;
34 *
35 * Changes
36 * -------
37 * 11-Oct-2001 : Version 1 (DG);
38 * 18-Dec-2001 : Changed order of parameters in constructor
       (DG);
39 * 19-Dec-2001 : Added a new constructor as suggested by
       Paul English (DG);
40 * 29-Jan-2002 : Worked on the parseWeek() method (DG);
41 * 13-Feb-2002 : Fixed bug in Week(Date) constructor (DG);
42 * 26-Feb-2002 : Changed getStart(), getMiddle() and getEnd
       () methods to
43 *               evaluate with reference to a particular
       time zone (DG);
44 * 05-Apr-2002 : Reinstated this class to the JCommon
       library (DG);
45 * 24-Jun-2002 : Removed unnecessary main method (DG);
46 * 10-Sep-2002 : Added getSerialIndex() method (DG);
47 * 06-Oct-2002 : Fixed errors reported by Checkstyle (DG);
48 * 18-Oct-2002 : Changed to observe 52 or 53 weeks per year
       , consistent with
49 *               GregorianCalendar. Thanks to Aimin Han for
       the code (DG);
50 * 02-Jan-2003 : Removed debug code (DG);
51 * 13-Mar-2003 : Moved to com.jrefinery.data.time package,
       and implemented
52 *               Serializable (DG);
53 * 21-Oct-2003 : Added hashCode() method (DG);
54 * 24-May-2004 : Modified getFirstMillisecond() and
       getLastMillisecond() to
55 *               take account of firstDayOfWeek setting in
       Java's Calendar
56 *               class (DG);
57 * 30-Sep-2004 : Replaced getTime().getTime() with
       getTimeInMillis() (DG);
58 * 04-Nov-2004 : Reverted change of 30-Sep-2004, because it
       won't work for
59 *               JDK 1.3 (DG);
```

```
60  * ------------- JFREECHART 1.0.x
        ---------------------------------------------
61  * 06-Mar-2006 : Fix for bug 1448828, incorrect calculation
        of week and year
62  *                for the first few days of some years (DG);
63  * 05-Oct-2006 : Updated API docs (DG);
64  * 06-Oct-2006 : Refactored to cache first and last
        millisecond values (DG);
65  * 09-Jan-2007 : Fixed bug in next() (DG);
66  * 28-Aug-2007 : Added new constructor to avoid problem in
        creating new
67  *                instances (DG);
68  * 19-Dec-2007 : Fixed bug in deprecated constructor (DG);
69  *
70  */
71
72 package org.jfree.data.time;
73
74 import java.io.Serializable;
75 import java.util.Calendar;
76 import java.util.Date;
77 import java.util.Locale;
78 import java.util.TimeZone;
79
80 /**
81  * A calendar week.  All years are considered to have 53
        weeks, numbered from 1
82  * to 53, although in many cases the 53rd week is empty.
        Most of the time, the
83  * 1st week of the year *begins* in the previous calendar
        year, but it always
84  * finishes in the current year (this behaviour matches the
        workings of the
85  * <code>GregorianCalendar</code> class).
86  * <P>
87  * This class is immutable, which is a requirement for all
88  * {@link RegularTimePeriod} subclasses.
89  */
90 public class Week extends RegularTimePeriod implements
       Serializable
91 {
92
93     /** For serialization. */
94     private static final long serialVersionUID =
           1856387786939865061L;
95
```

```
96      /** Constant for the first week in the year. */
97      public static final int FIRST_WEEK_IN_YEAR = 1;
98
99      /** Constant for the last week in the year. */
100     public static final int LAST_WEEK_IN_YEAR = 53;
101
102     /** The year in which the week falls. */
103     private short year;
104
105     /** The week (1-53). */
106     private byte week;
107
108     /** The first millisecond. */
109     private long firstMillisecond;
110
111     /** The last millisecond. */
112     private long lastMillisecond;
113
114     /**
115      * Creates a new time period for the week in which the
            current system
116      * date/time falls.
117      */
118     public Week()
119     {
120         this(new Date());
121     }
122
123     /**
124      * Creates a time period representing the week in the
            specified year.
125      *
126      * @param week  the week (1 to 53).
127      * @param year  the year (1900 to 9999).
128      */
129     public Week(int week, int year)
130     {
131         if ((week < FIRST_WEEK_IN_YEAR) && (week >
               LAST_WEEK_IN_YEAR))
132         {
133             throw new IllegalArgumentException(
134                     "The 'week' argument must be in the
                           range 1 - 53.");
135         }
136         this.week = (byte) week;
137         this.year = (short) year;
```

```
138         peg(Calendar.getInstance());
139     }
140 
141     /**
142      * Creates a time period representing the week in the
             specified year.
143      *
144      * @param week  the week (1 to 53).
145      * @param year  the year (1900 to 9999).
146      */
147     public Week(int week, Year year)
148     {
149         if ((week < FIRST_WEEK_IN_YEAR) && (week >
                LAST_WEEK_IN_YEAR))
150         {
151             throw new IllegalArgumentException(
152                     "The 'week' argument must be in the
                        range 1 - 53.");
153         }
154         this.week = (byte) week;
155         this.year = (short) year.getYear();
156         peg(Calendar.getInstance());
157     }
158 
159     /**
160      * Creates a time period for the week in which the
             specified date/time
161      * falls.
162      *
163      * @param time  the time (<code>null</code> not
             permitted).
164      */
165     public Week(Date time)
166     {
167         // defer argument checking...
168         this(time, RegularTimePeriod.DEFAULT_TIME_ZONE,
                Locale.getDefault());
169     }
170 
171     /**
172      * Creates a time period for the week in which the
             specified date/time
173      * falls, calculated relative to the specified time
             zone.
174      *
175      * @param time  the date/time (<code>null</code> not
```

```
         permitted).
176       * @param zone  the time zone (<code>null</code> not
             permitted).
177       *
178       * @deprecated As of 1.0.7, use {@link #Week(Date,
             TimeZone, Locale)}.
179       */
180      public Week(Date time, TimeZone zone)
181      {
182          // defer argument checking...
183          this(time, zone, Locale.getDefault());
184      }
185
186      /**
187       * Creates a time period for the week in which the
             specified date/time
188       * falls, calculated relative to the specified time
             zone.
189       *
190       * @param time  the date/time (<code>null</code> not
             permitted).
191       * @param zone  the time zone (<code>null</code> not
             permitted).
192       * @param locale  the locale (<code>null</code> not
             permitted).
193       *
194       * @since 1.0.7
195       */
196      public Week(Date time, TimeZone zone, Locale locale)
197      {
198          if (time == null)
199          {
200              throw new IllegalArgumentException("Null 'time'
                     argument.");
201          }
202          if (zone == null)
203          {
204              throw new IllegalArgumentException("Null 'zone'
                     argument.");
205          }
206          if (locale == null)
207          {
208              throw new IllegalArgumentException("Null '
                    locale' argument.");
209          }
210          Calendar calendar = Calendar.getInstance(zone,
```

```
            locale);
211         calendar.setTime(time);
212
213         // sometimes the last few days of the year are
                considered to fall in
214         // the *first* week of the following year.  Refer
                to the Javadocs for
215         // GregorianCalendar.
216         int tempWeek = calendar.get(Calendar.WEEK_OF_YEAR);
217         if (tempWeek == 1
218                 && calendar.get(Calendar.MONTH) == Calendar
                        .DECEMBER)
219         {
220             this.week = 1;
221             this.year = (short) (calendar.get(Calendar.YEAR
                        ) + 1);
222         }
223         else
224         {
225             this.week = (byte) Math.min(tempWeek,
                        LAST_WEEK_IN_YEAR);
226             int yyyy = calendar.get(Calendar.YEAR);
227             // alternatively, sometimes the first few days
                        of the year are
228             // considered to fall in the *last* week of the
                         previous year...
229             if (calendar.get(Calendar.MONTH) == Calendar.
                        JANUARY
230                     && this.week >= 52)
231             {
232                 yyyy--;
233             }
234             this.year = (short) yyyy;
235         }
236         peg(calendar);
237     }
238
239     /**
240      * Returns the year in which the week falls.
241      *
242      * @return The year (never <code>null</code>).
243      */
244     public Year getYear()
245     {
246         return new Year(this.year);
247     }
```

```
248
249     /**
250      * Returns the year in which the week falls, as an
            integer value.
251      *
252      * @return The year.
253      */
254     public int getYearValue()
255     {
256         return this.year;
257     }
258
259     /**
260      * Returns the week.
261      *
262      * @return The week.
263      */
264     public int getWeek()
265     {
266         return this.week;
267     }
268
269     /**
270      * Returns the first millisecond of the week.  This
            will be determined
271      * relative to the time zone specified in the
            constructor, or in the
272      * calendar instance passed in the most recent call to
            the
273      * {@link #peg(Calendar)} method.
274      *
275      * @return The first millisecond of the week.
276      *
277      * @see #getLastMillisecond()
278      */
279     public long getFirstMillisecond()
280     {
281         return this.firstMillisecond;
282     }
283
284     /**
285      * Returns the last millisecond of the week.  This will
             be
286      * determined relative to the time zone specified in
            the constructor, or
287      * in the calendar instance passed in the most recent
```

```
         call to the
288      * {@link #peg(Calendar)} method.
289      *
290      * @return The last millisecond of the week.
291      *
292      * @see #getFirstMillisecond()
293      */
294     public long getLastMillisecond()
295     {
296         return this.lastMillisecond;
297     }
298
299     /**
300      * Recalculates the start date/time and end date/time
            for this time period
301      * relative to the supplied calendar (which
            incorporates a time zone).
302      *
303      * @param calendar  the calendar (<code>null</code> not
             permitted).
304      *
305      * @since 1.0.3
306      */
307     public void peg(Calendar calendar)
308     {
309         this.firstMillisecond = getFirstMillisecond(
               calendar);
310         this.lastMillisecond = getLastMillisecond(calendar)
               ;
311     }
312
313     /**
314      * Returns the week preceding this one.  This method
            will return
315      * <code>null</code> for some lower limit on the range
            of weeks (currently
316      * week 1, 1900).  For week 1 of any year, the previous
             week is always week
317      * 53, but week 53 may not contain any days (you should
             check for this).
318      *
319      * @return The preceding week (possibly <code>null</
            code>).
320      */
321     public RegularTimePeriod previous()
322     {
```

```
323
324         Week result;
325         if (this.week != FIRST_WEEK_IN_YEAR)
326         {
327             result = new Week(this.week - 1, this.year);
328         }
329         else
330         {
331             // we need to work out if the previous year has
                    52 or 53 weeks...
332             if (this.year > 1900)
333             {
334                 int yy = this.year - 1;
335                 Calendar prevYearCalendar = Calendar.
                       getInstance();
336                 prevYearCalendar.set(yy, Calendar.DECEMBER,
                        31);
337                 result = new Week(prevYearCalendar.
                       getActualMaximum(
338                         Calendar.WEEK_OF_YEAR), yy);
339             }
340             else
341             {
342                 result = null;
343             }
344         }
345         return result;
346
347     }
348
349     /**
350      * Returns the week following this one.  This method
            will return
351      * <code>null</code> for some upper limit on the range
            of weeks (currently
352      * week 53, 9999).  For week 52 of any year, the
            following week is always
353      * week 53, but week 53 may not contain any days (you
            should check for
354      * this).
355      *
356      * @return The following week (possibly <code>null</
            code>).
357      */
358     public RegularTimePeriod next()
359     {
```

```
360
361         Week result;
362         if (this.week < 52)
363         {
364             result = new Week(this.week + 1, this.year);
365         }
366         else
367         {
368             Calendar calendar = Calendar.getInstance();
369             calendar.set(this.year, Calendar.DECEMBER, 31);
370             int actualMaxWeek
371                 = calendar.getActualMaximum(Calendar.
                       WEEK_OF_YEAR);
372             if (this.week < actualMaxWeek)
373             {
374                 result = new Week(this.week + 1, this.year)
                       ;
375             }
376             else
377             {
378                 if (this.year < 9999)
379                 {
380                     result = new Week(FIRST_WEEK_IN_YEAR,
                           this.year + 1);
381                 }
382                 else
383                 {
384                     result = null;
385                 }
386             }
387         }
388         return result;
389
390     }
391
392     /**
393      * Returns a serial index number for the week.
394      *
395      * @return The serial index number.
396      */
397     public long getSerialIndex()
398     {
399         return this.year * 53L + this.week;
400     }
401
402     /**
```

```
403         * Returns the first millisecond of the week, evaluated
                using the supplied
404         * calendar (which determines the time zone).
405         *
406         * @param calendar  the calendar (<code>null</code> not
                permitted).
407         *
408         * @return The first millisecond of the week.
409         *
410         * @throws NullPointerException if <code>calendar</code
                > is
411         *     <code>null</code>.
412         */
413        public long getFirstMillisecond(Calendar calendar)
414        {
415            Calendar c = (Calendar) calendar.clone();
416            c.clear();
417            c.set(Calendar.YEAR, this.year);
418            c.set(Calendar.WEEK_OF_YEAR, this.week);
419            c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
420            c.set(Calendar.HOUR, 0);
421            c.set(Calendar.MINUTE, 0);
422            c.set(Calendar.SECOND, 0);
423            c.set(Calendar.MILLISECOND, 0);
424            //return c.getTimeInMillis();  // this won't work
                   for JDK 1.3
425            return c.getTime().getTime();
426        }
427
428        /**
429         * Returns the last millisecond of the week, evaluated
                using the supplied
430         * calendar (which determines the time zone).
431         *
432         * @param calendar  the calendar (<code>null</code> not
                permitted).
433         *
434         * @return The last millisecond of the week.
435         *
436         * @throws NullPointerException if <code>calendar</code
                > is
437         *     <code>null</code>.
438         */
439        public long getLastMillisecond(Calendar calendar)
440        {
441            Calendar c = (Calendar) calendar.clone();
```

```
442         c.clear();
443         c.set(Calendar.YEAR, this.year);
444         c.set(Calendar.WEEK_OF_YEAR, this.week + 1);
445         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
446         c.set(Calendar.HOUR, 0);
447         c.set(Calendar.MINUTE, 0);
448         c.set(Calendar.SECOND, 0);
449         c.set(Calendar.MILLISECOND, 0);
450         //return c.getTimeInMillis();  // this won't work
                for JDK 1.3
451         return c.getTime().getTime() - 1;
452     }
453
454     /**
455      * Returns a string representing the week (e.g. "Week
            9, 2002").
456      *
457      * TODO: look at internationalisation.
458      *
459      * @return A string representing the week.
460      */
461     public String toString()
462     {
463         return "Week " + this.week + ", " + this.year;
464     }
465
466     /**
467      * Tests the equality of this Week object to an
            arbitrary object.  Returns
468      * true if the target is a Week instance representing
            the same week as this
469      * object.  In all other cases, returns false.
470      *
471      * @param obj  the object (<code>null</code> permitted)
            .
472      *
473      * @return <code>true</code> if week and year of this
            and object are the
474      *         same.
475      */
476     public boolean equals(Object obj)
477     {
478
479         if (obj == this)
480         {
481             return true;
```

```
482         }
483         if (!(obj instanceof Week))
484         {
485             return false;
486         }
487         Week that = (Week) obj;
488         if (this.week != that.week)
489         {
490             return false;
491         }
492         if (this.year != that.year)
493         {
494             return false;
495         }
496         return true;
497 
498     }
499 
500     /**
501      * Returns a hash code for this object instance.  The
             approach described by
502      * Joshua Bloch in "Effective Java" has been used here:
503      * <p>
504      * <code>http://developer.java.sun.com/developer/Books/
             effectivejava
505      * /Chapter3.pdf</code>
506      *
507      * @return A hash code.
508      */
509     public int hashCode()
510     {
511         int result = 17;
512         result = 37 * result + this.week;
513         result = 37 * result + this.year;
514         return result;
515     }
516 
517     /**
518      * Returns an integer indicating the order of this Week
               object relative to
519      * the specified object:
520      *
521      * negative == before, zero == same, positive == after.
522      *
523      * @param o1  the object to compare.
524      *
```

```
525         * @return negative == before, zero == same, positive
                == after.
526         */
527        public int compareTo(Object o1)
528        {
529
530            int result;
531
532            // CASE 1 : Comparing to another Week object
533            // -----------------------------------------
534            if (o1 instanceof Week)
535            {
536                Week w = (Week) o1;
537                result = this.year - w.getYear().getYear();
538                if (result == 0)
539                {
540                    result = this.week - w.getWeek();
541                }
542            }
543
544            // CASE 2 : Comparing to another TimePeriod object
545            // -----------------------------------------------
546            else if (o1 instanceof RegularTimePeriod)
547            {
548                // more difficult case - evaluate later...
549                result = 0;
550            }
551
552            // CASE 3 : Comparing to a non-TimePeriod object
553            // ---------------------------------------------
554            else
555            {
556                // consider time periods to be ordered after
                       general objects
557                result = 1;
558            }
559
560            return result;
561
562        }
563
564        /**
565         * Parses the string argument as a week.
566         * <P>
567         * This method is required to accept the format "YYYY-
                Wnn".  It will also
```

```
568         * accept "Wnn-YYYY". Anything else, at the moment, is
                a bonus.
569         *
570         * @param s  string to parse.
571         *
572         * @return <code>null</code> if the string is not
                parseable, the week
573         *         otherwise.
574         */
575        public static Week parseWeek(String s)
576        {
577
578            Week result = null;
579            if (s != null)
580            {
581
582                // trim whitespace from either end of the
                       string
583                s = s.trim();
584
585                int i = Week.findSeparator(s);
586                if (i != -1)
587                {
588                    String s1 = s.substring(0, i).trim();
589                    String s2 = s.substring(i + 1, s.length()).
                           trim();
590
591                    Year y = Week.evaluateAsYear(s1);
592                    int w;
593                    if (y != null)
594                    {
595                        w = Week.stringToWeek(s2);
596                        if (w == -1)
597                        {
598                            throw new TimePeriodFormatException
                                   (
599                                    "Can't evaluate the week.")
                                       ;
600                        }
601                        result = new Week(w, y);
602                    }
603                    else
604                    {
605                        y = Week.evaluateAsYear(s2);
606                        if (y != null)
607                        {
```

```
608                         w = Week.stringToWeek(s1);
609                         if (w == -1)
610                         {
611                             throw new
                                    TimePeriodFormatException(
612                                         "Can't evaluate the
                                            week.");
613                         }
614                         result = new Week(w, y);
615                     }
616                     else
617                     {
618                         throw new TimePeriodFormatException
                                (
619                                      "Can't evaluate the year.")
                                         ;
620                     }
621                 }
622
623             }
624             else
625             {
626                 throw new TimePeriodFormatException(
627                         "Could not find separator.");
628             }
629
630         }
631         return result;
632
633     }
634
635     /**
636      * Finds the first occurrence of ' ', '-', ',' or '.'
637      *
638      * @param s  the string to parse.
639      *
640      * @return <code>-1</code> if none of the characters
                    was found, the
641      *         index of the first occurrence otherwise.
642      */
643     private static int findSeparator(String s)
644     {
645
646         int result = s.indexOf('-');
647         if (result == -1)
648         {
```

```
649             result = s.indexOf(',');
650         }
651         if (result == -1)
652         {
653             result = s.indexOf(' ');
654         }
655         if (result == -1)
656         {
657             result = s.indexOf('.');
658         }
659         return result;
660     }
661 
662     /**
663      * Creates a year from a string, or returns null (
           format exceptions
664      * suppressed).
665      *
666      * @param s  string to parse.
667      *
668      * @return <code>null</code> if the string is not
           parseable, the year
669      *         otherwise.
670      */
671     private static Year evaluateAsYear(String s)
672     {
673 
674         Year result = null;
675         try
676         {
677             result = Year.parseYear(s);
678         }
679         catch (TimePeriodFormatException e)
680         {
681             // suppress
682         }
683         return result;
684 
685     }
686 
687     /**
688      * Converts a string to a week.
689      *
690      * @param s  the string to parse.
691      * @return <code>-1</code> if the string does not
           contain a week number,
```

```
692         *          the number of the week otherwise.
693         */
694        private static int stringToWeek(String s)
695        {
696
697            int result = -1;
698            s = s.replace('W', ' ');
699            s = s.trim();
700            try
701            {
702                result = Integer.parseInt(s);
703                if ((result < 1) || (result > LAST_WEEK_IN_YEAR
                       ))
704                {
705                    result = -1;
706                }
707            }
708            catch (NumberFormatException e)
709            {
710                // suppress
711            }
712            return result;
713
714        }
715
716    }
```

E.3 Manually Transformed Sources for Task Lang-1

E.3.1 Defects-Sources/Lang-1/NumberUtils-bugged-A.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under
       one or more
3   * contributor license agreements.  See the NOTICE file
       distributed
4   * with this work for additional information regarding
       copyright
5   * ownership. The ASF licenses this file to You under the
       Apache
6   * License, Version 2.0 (the "License"); you may not use
       this file
7   * except in compliance with the License.  You may obtain a
        copy of
8   * the License at
9   *
10  *      http://www.apache.org/licenses/LICENSE-2.0
11  *
12  * Unless required by applicable law or agreed to in
       writing,
13  * software distributed under the License is distributed on
        an "AS
14  * IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
        either
15  * express or implied. See the License for the specific
       language
16  * governing permissions and limitations under the License.
17  */
18 package org.apache.commons.lang3.math;
19
20
21 import java.lang.reflect.Array;
22 import java.math.BigDecimal;
23 import java.math.BigInteger;
24 import org.apache.commons.lang3.StringUtils;
25
26
27 /**
28  * <p>Provides extra functionality for Java Number classes
       .</p>
29  *
30  * @since 2.0
```

```
31  * @version $Id$
32  */
33 public class NumberUtils {
34 
35     /** Reusable Long constant for zero. */
36     public static final Long LONG_ZERO = Long.valueOf(0L);
37 
38     /** Reusable Long constant for one. */
39     public static final Long LONG_ONE = Long.valueOf(1L);
40 
41     /** Reusable Long constant for minus one. */
42     public static final Long LONG_MINUS_ONE = Long.valueOf
       (-1L);
43 
44     /** Reusable Integer constant for zero. */
45     public static final Integer INTEGER_ZERO = Integer.
       valueOf(0);
46 
47     /** Reusable Integer constant for one. */
48     public static final Integer INTEGER_ONE = Integer.
       valueOf(1);
49 
50     /** Reusable Integer constant for minus one. */
51     public static final Integer INTEGER_MINUS_ONE = Integer
       .valueOf(-1);
52 
53     /** Reusable Short constant for zero. */
54     public static final Short SHORT_ZERO = Short.valueOf((
       short) 0);
55 
56     /** Reusable Short constant for one. */
57     public static final Short SHORT_ONE = Short.valueOf((
       short) 1);
58 
59     /** Reusable Short constant for minus one. */
60     public static final Short SHORT_MINUS_ONE = Short.
       valueOf((short) -1);
61 
62     /** Reusable Byte constant for zero. */
63     public static final Byte BYTE_ZERO = Byte.valueOf((byte
       ) 0);
64 
65     /** Reusable Byte constant for one. */
66     public static final Byte BYTE_ONE = Byte.valueOf((byte)
        1);
67 
```

```
68      /** Reusable Byte constant for minus one. */
69      public static final Byte BYTE_MINUS_ONE = Byte.valueOf
           ((byte) -1);
70
71      /** Reusable Double constant for zero. */
72      public static final Double DOUBLE_ZERO = Double.valueOf
           (0.0d);
73
74      /** Reusable Double constant for one. */
75      public static final Double DOUBLE_ONE = Double.valueOf
           (1.0d);
76
77      /** Reusable Double constant for minus one. */
78      public static final Double DOUBLE_MINUS_ONE = Double.
           valueOf(-1.0d);
79
80      /** Reusable Float constant for zero. */
81      public static final Float FLOAT_ZERO = Float.valueOf
           (0.0f);
82
83      /** Reusable Float constant for one. */
84      public static final Float FLOAT_ONE = Float.valueOf(1.0
           f);
85
86      /** Reusable Float constant for minus one. */
87      public static final Float FLOAT_MINUS_ONE = Float.
           valueOf(-1.0f);
88
89
90      /**
91       * <p><code>NumberUtils</code> instances should NOT be
            constructed
92       * in standard programming. Instead, the class should
            be used as
93       * <code>NumberUtils.toInt("6");</code>.</p>
94       *
95       * <p>This constructor is public to permit tools that
            require a
96       * JavaBean instance to operate.</p>
97       */
98      public NumberUtils() {
99          super();
100     }
101
102
103     //
```

```
    //-----------------------------------------------------------------------

104     /**
105      * <p>Convert a <code>String</code> to an <code>int</code>,
106      * returning <code>zero</code> if the conversion fails.</p>
107      *
108      * <p>If the string is <code>null</code>, <code>zero</code> is
109      * returned.</p>
110      *
111      * <pre>
112      *   NumberUtils.toInt(null) = 0
113      *   NumberUtils.toInt("")   = 0
114      *   NumberUtils.toInt("1")  = 1
115      * </pre>
116      *
117      * @param str  the string to convert, may be null
118      * @return the int represented by the string, or <code>zero</code>
119      *  if conversion fails
120      * @since 2.1
121      */
122     public static int toInt(final String str) {
123         return toInt(str, 0);
124     }
125
126     /**
127      * <p>Convert a <code>String</code> to an <code>int</code>,
128      * returning a default value if the conversion fails.</p>
129      *
130      * <p>If the string is <code>null</code>, the default value is
131      * returned.</p>
132      *
133      * <pre>
134      *   NumberUtils.toInt(null, 1) = 1
135      *   NumberUtils.toInt("", 1)   = 1
136      *   NumberUtils.toInt("1", 0)  = 1
137      * </pre>
138      *
139      * @param str  the string to convert, may be null
140      * @param defaultValue  the default value
```

```
141         * @return the int represented by the string, or the
                default if
142         *  conversion fails
143         * @since 2.1
144         */
145        public static int toInt(final String str, final int
               defaultValue) {
146            if(str == null) {
147                return defaultValue;
148            }
149            try {
150                return Integer.parseInt(str);
151            } catch (final NumberFormatException nfe) {
152                return defaultValue;
153            }
154        }
155
156        /**
157         * <p>Convert a <code>String</code> to a <code>long</
                code>,
158         * returning <code>zero</code> if the conversion fails
                .</p>
159         *
160         * <p>If the string is <code>null</code>, <code>zero</
                code> is
161         * returned.</p>
162         *
163         * <pre>
164         *   NumberUtils.toLong(null) = 0L
165         *   NumberUtils.toLong("")   = 0L
166         *   NumberUtils.toLong("1")  = 1L
167         * </pre>
168         *
169         * @param str  the string to convert, may be null
170         * @return the long represented by the string, or <code
                >0</code>
171         *  if conversion fails
172         * @since 2.1
173         */
174        public static long toLong(final String str) {
175            return toLong(str, 0L);
176        }
177
178        /**
179         * <p>Convert a <code>String</code> to a <code>long</
                code>,
```

```
180         * returning a default value if the conversion fails.</
               p>
181         *
182         * <p>If the string is <code>null</code>, the default
               value is
183         * returned.</p>
184         *
185         * <pre>
186         *   NumberUtils.toLong(null, 1L) = 1L
187         *   NumberUtils.toLong("", 1L)   = 1L
188         *   NumberUtils.toLong("1", 0L)  = 1L
189         * </pre>
190         *
191         * @param str  the string to convert, may be null
192         * @param defaultValue  the default value
193         * @return the long represented by the string, or the
               default if
194         *  conversion fails
195         * @since 2.1
196         */
197        public static long toLong(final String str, final long
               defaultValue) {
198            if (str == null) {
199                return defaultValue;
200            }
201            try {
202                return Long.parseLong(str);
203            } catch (final NumberFormatException nfe) {
204                return defaultValue;
205            }
206        }
207
208        /**
209         * <p>Convert a <code>String</code> to a <code>float</
               code>,
210         * returning <code>0.0f</code> if the conversion fails
               .</p>
211         *
212         * <p>If the string <code>str</code> is <code>null</
               code>,
213         * <code>0.0f</code> is returned.</p>
214         *
215         * <pre>
216         *   NumberUtils.toFloat(null)   = 0.0f
217         *   NumberUtils.toFloat("")     = 0.0f
218         *   NumberUtils.toFloat("1.5")  = 1.5f
```

```
219         * </pre>
220         *
221         * @param str the string to convert, may be <code>null
                </code>
222         * @return the float represented by the string, or
223         *  <code>0.0f</code> if conversion fails
224         * @since 2.1
225         */
226        public static float toFloat(final String str) {
227            return toFloat(str, 0.0f);
228        }
229
230        /**
231         * <p>Convert a <code>String</code> to a <code>float</
                code>,
232         * returning a default value if the conversion fails.</
                p>
233         *
234         * <p>If the string <code>str</code> is <code>null</
                code>, the
235         * default value is returned.</p>
236         *
237         * <pre>
238         *   NumberUtils.toFloat(null, 1.1f)   = 1.0f
239         *   NumberUtils.toFloat("", 1.1f)     = 1.1f
240         *   NumberUtils.toFloat("1.5", 0.0f)  = 1.5f
241         * </pre>
242         *
243         * @param str the string to convert, may be <code>null
                </code>
244         * @param defaultValue the default value
245         * @return the float represented by the string, or
                defaultValue
246         *  if conversion fails
247         * @since 2.1
248         */
249        public static float toFloat(final String str, final
               float defaultValue) {
250            if (str == null) {
251                return defaultValue;
252            }
253            try {
254                return Float.parseFloat(str);
255            } catch (final NumberFormatException nfe) {
256                return defaultValue;
257            }
```

```
258     }
259
260     /**
261      * <p>Convert a <code>String</code> to a <code>double</
            code>,
262      * returning <code>0.0d</code> if the conversion fails
            .</p>
263      *
264      * <p>If the string <code>str</code> is <code>null</
            code>,
265      * <code>0.0d</code> is returned.</p>
266      *
267      * <pre>
268      *   NumberUtils.toDouble(null)   = 0.0d
269      *   NumberUtils.toDouble("")     = 0.0d
270      *   NumberUtils.toDouble("1.5")  = 1.5d
271      * </pre>
272      *
273      * @param str the string to convert, may be <code>null
            </code>
274      * @return the double represented by the string, or
275      *  <code>0.0d</code> if conversion fails
276      * @since 2.1
277      */
278     public static double toDouble(final String str) {
279         return toDouble(str, 0.0d);
280     }
281
282     /**
283      * <p>Convert a <code>String</code> to a <code>double</
            code>,
284      * returning a default value if the conversion fails.</
            p>
285      *
286      * <p>If the string <code>str</code> is <code>null</
            code>, the
287      * default value is returned.</p>
288      *
289      * <pre>
290      *   NumberUtils.toDouble(null, 1.1d)   = 1.1d
291      *   NumberUtils.toDouble("", 1.1d)     = 1.1d
292      *   NumberUtils.toDouble("1.5", 0.0d)  = 1.5d
293      * </pre>
294      *
295      * @param str the string to convert, may be <code>null
            </code>
```

```
296       * @param defaultValue the default value
297       * @return the double represented by the string, or
              defaultValue
298       *  if conversion fails
299       * @since 2.1
300       */
301      public static double toDouble(final String str, final
             double defaultValue) {
302        if (str == null) {
303            return defaultValue;
304        }
305        try {
306            return Double.parseDouble(str);
307        } catch (final NumberFormatException nfe) {
308            return defaultValue;
309        }
310      }
311
312
313       //
              -----------------------------------------------------------------------
314       /**
315       * <p>Convert a <code>String</code> to a <code>byte</
              code>,
316       * returning <code>zero</code> if the conversion fails
              .</p>
317       *
318       * <p>If the string is <code>null</code>, <code>zero</
              code> is
319       * returned.</p>
320       *
321       * <pre>
322       *   NumberUtils.toByte(null) = 0
323       *   NumberUtils.toByte("")   = 0
324       *   NumberUtils.toByte("1")  = 1
325       * </pre>
326       *
327       * @param str  the string to convert, may be null
328       * @return the byte represented by the string, or
329       *  <code>zero</code> if conversion fails
330       * @since 2.5
331       */
332      public static byte toByte(final String str) {
333          return toByte(str, (byte) 0);
334      }
```

```
335
336     /**
337      * <p>Convert a <code>String</code> to a <code>byte</
             code>,
338      * returning a default value if the conversion fails.</
             p>
339      *
340      * <p>If the string is <code>null</code>, the default
             value is
341      * returned.</p>
342      *
343      * <pre>
344      *   NumberUtils.toByte(null, 1) = 1
345      *   NumberUtils.toByte("", 1)   = 1
346      *   NumberUtils.toByte("1", 0)  = 1
347      * </pre>
348      *
349      * @param str  the string to convert, may be null
350      * @param defaultValue  the default value
351      * @return the byte represented by the string, or the
             default if
352      *  conversion fails
353      * @since 2.5
354      */
355     public static byte toByte(final String str, final byte
           defaultValue) {
356         if(str == null) {
357             return defaultValue;
358         }
359         try {
360             return Byte.parseByte(str);
361         } catch (final NumberFormatException nfe) {
362             return defaultValue;
363         }
364     }
365
366     /**
367      * <p>Convert a <code>String</code> to a <code>short</
             code>,
368      * returning <code>zero</code> if the conversion fails
             .</p>
369      *
370      * <p>If the string is <code>null</code>, <code>zero</
             code> is
371      * returned.</p>
372      *
```

```
373         * <pre>
374         *   NumberUtils.toShort(null) = 0
375         *   NumberUtils.toShort("")   = 0
376         *   NumberUtils.toShort("1")  = 1
377         * </pre>
378         *
379         * @param str  the string to convert, may be null
380         * @return the short represented by the string, or
381         *  <code>zero</code> if conversion fails
382         * @since 2.5
383         */
384        public static short toShort(final String str) {
385            return toShort(str, (short) 0);
386        }
387
388        /**
389         * <p>Convert a <code>String</code> to an <code>short</
               code>,
390         * returning a default value if the conversion fails.</
               p>
391         *
392         * <p>If the string is <code>null</code>, the default
               value is
393         * returned.</p>
394         *
395         * <pre>
396         *   NumberUtils.toShort(null, 1) = 1
397         *   NumberUtils.toShort("", 1)   = 1
398         *   NumberUtils.toShort("1", 0)  = 1
399         * </pre>
400         *
401         * @param str  the string to convert, may be null
402         * @param defaultValue  the default value
403         * @return the short represented by the string, or the
               default if
404         *  conversion fails
405         * @since 2.5
406         */
407        public static short toShort(final String str, final
               short defaultValue) {
408            if(str == null) {
409                return defaultValue;
410            }
411            try {
412                return Short.parseShort(str);
413            } catch (final NumberFormatException nfe) {
```

```
414             return defaultValue;
415         }
416     }
417
418
419     //
        ----------------------------------------------------------------

420     // must handle Long, Float, Integer, Float, Short,
421     //                  BigDecimal, BigInteger and Byte
422     // useful methods:
423     // Byte.decode(String)
424     // Byte.valueOf(String,int radix)
425     // Byte.valueOf(String)
426     // Double.valueOf(String)
427     // Float.valueOf(String)
428     // Float.valueOf(String)
429     // Integer.valueOf(String,int radix)
430     // Integer.valueOf(String)
431     // Integer.decode(String)
432     // Integer.getInteger(String)
433     // Integer.getInteger(String,int val)
434     // Integer.getInteger(String,Integer val)
435     // Integer.valueOf(String)
436     // Double.valueOf(String)
437     // new Byte(String)
438     // Long.valueOf(String)
439     // Long.getLong(String)
440     // Long.getLong(String,int)
441     // Long.getLong(String,Integer)
442     // Long.valueOf(String,int)
443     // Long.valueOf(String)
444     // Short.valueOf(String)
445     // Short.decode(String)
446     // Short.valueOf(String,int)
447     // Short.valueOf(String)
448     // new BigDecimal(String)
449     // new BigInteger(String)
450     // new BigInteger(String,int radix)
451     // Possible inputs:
452     // 45 45.5 45E7 4.5E7 Hex Oct Binary xxxF xxxD xxxf
        xxxd
453     // plus minus everything. Prolly more. A lot are not
        separable.
454
455     /**
```

```
456       * <p>Turns a string value into a java.lang.Number.</p>
457       *
458       * <p>If the string starts with {@code 0x} or {@code -0
              x} (lower
459       * or upper case) or {@code #} or {@code -#}, it will
              be
460       * interpreted as a hexadecimal Integer - or Long, if
              the number
461       * of digits after the prefix is more than 8 - or
              BigInteger if
462       * there are more than 16 digits. </p>
463       *
464       * <p>Then, the value is examined for a type qualifier
              on the end,
465       * i.e. one of <code>'f','F','d','D','l','L'</code>.
              If it is
466       * found, it starts trying to create successively
              larger types
467       * from the type specified until one is found that can
              represent
468       * the value.</p>
469       *
470       * <p>If a type specifier is not found, it will check
              for a
471       * decimal point and then try successively larger types
               from
472       * <code>Integer</code> to <code>BigInteger</code> and
              from
473       * <code>Float</code> to <code>BigDecimal</code>.</p>
474       *
475       * <p>
476       * Integral values with a leading {@code 0} will be
              interpreted as
477       * octal; the returned number will be Integer, Long or
              BigDecimal
478       * as appropriate.
479       * </p>
480       *
481       * <p>Returns <code>null</code> if the string is
482       * <code>null</code>.</p>
483       *
484       * <p>This method does not trim the input string, i.e.,
               strings
485       * with leading or trailing spaces will generate
486       * NumberFormatExceptions.</p>
487       *
```

```
488         * @param str  String containing a number, may be null
489         * @return Number created from the string (or null if
                the input is
490         *  null)
491         * @throws NumberFormatException if the value cannot be
                converted
492         */
493        public static Number createNumber(final String str)
494                throws NumberFormatException {
495            if (str == null) {
496                return null;
497            }
498            if (StringUtils.isBlank(str)) {
499                throw new NumberFormatException(
500                    "A blank string is not a valid number");
501            }
502
503            // Need to deal with all possible hex prefixes here
504            final String[] hex_prefixes = {"0x", "0X", "-0x", "
                   -0X", "#", "-#"};
505            int pfxLen = 0;
506            for(final String pfx : hex_prefixes) {
507                if (str.startsWith(pfx)) {
508                    pfxLen += pfx.length();
509                    break;
510                }
511            }
512            if (pfxLen > 0) { // we have a hex number
513                final int hexDigits = str.length() - pfxLen;
514                if (hexDigits > 16) { // too many for Long
515                    return createBigInteger(str);
516                }
517                if (hexDigits > 8) { // too many for an int
518                    return createLong(str);
519                }
520                return createInteger(str);
521            }
522            final char lastChar = str.charAt(str.length() - 1);
523            String mant;
524            String dec;
525            String exp;
526            final int decPos = str.indexOf('.');
527            final int expPos = str.indexOf('e') + str.indexOf('
                   E') + 1;
528
529            // assumes both not present
```

```
530         // if both e and E are present, this is caught by
                the checks
531         // on expPos (which prevent IOOBE) and the parsing
                which will
532         // detect if e or E appear in a number due to using
                 the wrong
533         // offset
534
535         int numDecimals = 0; // Check required precision (
                LANG-693)
536         if (decPos > -1) { // there is a decimal point
537
538             if (expPos > -1) { // there is an exponent
539                 if (expPos < decPos || expPos > str.length
                        ()) {
540
541                     // prevents double exponent causing
                            IOOBE
542                     throw new NumberFormatException(str
543                         + " is not a valid number.");
544                 }
545                 dec = str.substring(decPos + 1, expPos);
546             } else {
547                 dec = str.substring(decPos + 1);
548             }
549             mant = str.substring(0, decPos);
550             numDecimals = dec.length();
551
552             // gets number of digits past
553             // the decimal to ensure no loss of precision
                    for
554             // floating point numbers.
555         } else {
556             if (expPos > -1) {
557                 if (expPos > str.length()) {
558
559                     // prevents double exponent causing
                            IOOBE
560                     throw new NumberFormatException(str
561                         + " is not a valid number.");
562                 }
563                 mant = str.substring(0, expPos);
564             } else {
565                 mant = str;
566             }
567             dec = null;
```

```
568             }
569             if (!Character.isDigit(lastChar) && lastChar != '.'
                    ) {
570                 if (expPos > -1 && expPos < str.length() - 1) {
571                     exp = str.substring(expPos + 1, str.length
                            () - 1);
572                 } else {
573                     exp = null;
574                 }
575 
576                 //Requesting a specific type..
577                 final String numeric = str.substring(0, str.
                        length() - 1);
578                 final boolean allZeros = isAllZeros(mant) &&
                        isAllZeros(exp);
579                 switch (lastChar) {
580                     case 'l' :
581                     case 'L' :
582                         if (dec == null
583                             && exp == null
584                             && (numeric.charAt(0) == '-'
585                             && isDigits(numeric.substring(1))
586                             || isDigits(numeric))) {
587                             try {
588                                 return createLong(numeric);
589                             } catch (final
                                    NumberFormatException nfe) {
590 
591                                 // NOPMD
592                                 // Too big for a long
593                             }
594                             return createBigInteger(numeric);
595                         }
596                         throw new NumberFormatException(str
597                             + " is not a valid number.");
598                     case 'f' :
599                     case 'F' :
600                         try {
601                             final Float f = NumberUtils.
                                    createFloat(numeric);
602                             if (!(f.isInfinite() || (f.
                                    floatValue() == 0.0F
603                                 && !allZeros))) {
604 
605                                 // If it's too big for a float
                                    or the
```

```
606                             // float value = 0 and the
                                   string has
607                             // non-zeros in it, then float
                                   does not
608                             // have the precision we want
609                             return f;
610                         }
611                     } catch (final NumberFormatException
                           nfe) {
612
613                         // NOPMD
614                         // ignore the bad number
615                     }
616
617                     //$FALL-THROUGH$
618                 case 'd' :
619                 case 'D' :
620                     try {
621                         final Double d = NumberUtils.
                               createDouble(numeric);
622                         if (!(d.isInfinite() || (d.
                               floatValue() == 0.0D
623                             && !allZeros))) {
624                             return d;
625                         }
626                     } catch (final NumberFormatException
                           nfe) {
627
628                         // NOPMD
629                         // ignore the bad number
630                     }
631                     try {
632                         return createBigDecimal(numeric);
633                     } catch (final NumberFormatException e)
                           {
634
635                         // NOPMD
636                         // ignore the bad number
637                     }
638
639                     //$FALL-THROUGH$
640                 default :
641                     throw new NumberFormatException(str
642                         + " is not a valid number.");
643             }
644         }
```

```
645
646         //User doesn't have a preference on the return type
                , so let's
647         //start small and go from there...
648         if (expPos > -1 && expPos < str.length() - 1) {
649             exp = str.substring(expPos + 1, str.length());
650         } else {
651             exp = null;
652         }
653         if (dec == null && exp == null) {
654
655             // no decimal point and no exponent
656             //Must be an Integer, Long, Biginteger
657             try {
658                 return createInteger(str);
659             } catch (final NumberFormatException nfe) {
660
661                 // NOPMD
662                 // ignore the bad number
663             }
664             try {
665                 return createLong(str);
666             } catch (final NumberFormatException nfe) {
667
668                 // NOPMD
669                 // ignore the bad number
670             }
671             return createBigInteger(str);
672         }
673
674         //Must be a Float, Double, BigDecimal
675         final boolean allZeros = isAllZeros(mant) &&
                isAllZeros(exp);
676         try {
677             if(numDecimals <= 7){
678
679                 // If number has 7 or fewer digits past the
                        decimal
680                 // point then make it a float
681                 final Float f = createFloat(str);
682                 if (!(f.isInfinite()
683                         || (f.floatValue() == 0.0F && !allZeros
                            ))) {
684                     return f;
685                 }
686             }
```

```
687         } catch (final NumberFormatException nfe) {
688 
689             // NOPMD
690             // ignore the bad number
691         }
692         try {
693             if(numDecimals <= 16){
694 
695                 // If number has between 8 and 16 digits
                       past the
696                 // decimal point then make it a double
697                 final Double d = createDouble(str);
698                 if (!(d.isInfinite()
699                     || (d.doubleValue() == 0.0D && !
                           allZeros))) {
700                     return d;
701                 }
702             }
703         } catch (final NumberFormatException nfe) {
704 
705             // NOPMD
706             // ignore the bad number
707         }
708 
709         return createBigDecimal(str);
710     }
711 
712     /**
713      * <p>Utility method for
714      * {@link #createNumber(java.lang.String)}.</p>
715      *
716      * <p>Returns <code>true</code> if s is <code>null</
           code>.</p>
717      *
718      * @param str  the String to check
719      * @return if it is all zeros or <code>null</code>
720      */
721     private static boolean isAllZeros(final String str) {
722         if (str == null) {
723             return true;
724         }
725         for (int i = str.length() - 1; i >= 0; i--) {
726             if (str.charAt(i) != '0') {
727                 return false;
728             }
729         }
```

```
730         return str.length() > 0;
731     }
732 
733 
734     //-----------------------------------------------------------------------
735     /**
736      * <p>Convert a <code>String</code> to a <code>Float</code>.</p>
737      *
738      * <p>Returns <code>null</code> if the string is
739      * <code>null</code>.</p>
740      *
741      * @param str  a <code>String</code> to convert, may be null
742      * @return converted <code>Float</code> (or null if the input is
743      *  null)
744      * @throws NumberFormatException if the value cannot be converted
745      */
746     public static Float createFloat(final String str) {
747         if (str == null) {
748             return null;
749         }
750         return Float.valueOf(str);
751     }
752 
753     /**
754      * <p>Convert a <code>String</code> to a <code>Double</code>.</p>
755      *
756      * <p>Returns <code>null</code> if the string is
757      * <code>null</code>.</p>
758      *
759      * @param str  a <code>String</code> to convert, may be null
760      * @return converted <code>Double</code> (or null if the input is
761      *  null)
762      * @throws NumberFormatException if the value cannot be converted
763      */
764     public static Double createDouble(final String str) {
765         if (str == null) {
```

```
766             return null;
767         }
768         return Double.valueOf(str);
769     }
770 
771     /**
772      * <p>Convert a <code>String</code> to a <code>Integer
            </code>,
773      * handling hex and octal notations.</p>
774      *
775      * <p>Returns <code>null</code> if the string is
776      * <code>null</code>.</p>
777      *
778      * @param str  a <code>String</code> to convert, may be
            null
779      * @return converted <code>Integer</code> (or null if
            the input is
780      *  null)
781      * @throws NumberFormatException if the value cannot be
            converted
782      */
783     public static Integer createInteger(final String str) {
784         if (str == null) {
785             return null;
786         }
787 
788         // decode() handles 0xAABD and 0777 (hex and octal)
                as well.
789         return Integer.decode(str);
790     }
791 
792     /**
793      * <p>Convert a <code>String</code> to a <code>Long</
            code>;
794      * since 3.1 it handles hex and octal notations.</p>
795      *
796      * <p>Returns <code>null</code> if the string is
797      * <code>null</code>.</p>
798      *
799      * @param str  a <code>String</code> to convert, may be
            null
800      * @return converted <code>Long</code> (or null if the
            input is
801      *  null)
802      * @throws NumberFormatException if the value cannot be
            converted
```

```
803         */
804         public static Long createLong(final String str) {
805             if (str == null) {
806                 return null;
807             }
808             return Long.decode(str);
809         }
810 
811         /**
812          * <p>Convert a <code>String</code> to a <code>
                BigInteger</code>;
813          * since 3.2 it handles hex (0x or #) and octal (0)
                notations.</p>
814          *
815          * <p>Returns <code>null</code> if the string is
816          * <code>null</code>.</p>
817          *
818          * @param str  a <code>String</code> to convert, may be
                 null
819          * @return converted <code>BigInteger</code> (or null
                if the input
820          *  is null)
821          * @throws NumberFormatException if the value cannot be
                 converted
822          */
823         public static BigInteger createBigInteger(final String
               str) {
824             if (str == null) {
825                 return null;
826             }
827             int pos = 0; // offset within string
828             int radix = 10;
829             boolean negate = false; // need to negate later?
830             if (str.startsWith("-")) {
831                 negate = true;
832                 pos = 1;
833             }
834             if (str.startsWith("0x", pos) || str.startsWith("0x
                   ", pos)) {
835 
836                 // hex
837                 radix = 16;
838                 pos += 2;
839             } else if (str.startsWith("#", pos)) {
840 
841                 // alternative hex (allowed by Long/Integer)
```

```
842             radix = 16;
843             pos ++;
844         } else if (str.startsWith("0", pos) && str.length()
                > pos + 1) {
845
846             // octal; so long as there are additional
                   digits
847             radix = 8;
848             pos ++;
849         } // default is to treat as decimal
850
851         final BigInteger value = new BigInteger(str.
               substring(pos), radix);
852         return negate ? value.negate() : value;
853     }
854
855     /**
856      * <p>Convert a <code>String</code> to a
857      * <code>BigDecimal</code>.</p>
858      *
859      * <p>Returns <code>null</code> if the string is
860      * <code>null</code>.</p>
861      *
862      * @param str  a <code>String</code> to convert, may be
                null
863      * @return converted <code>BigDecimal</code> (or null
                if the input
864      *  is null)
865      * @throws NumberFormatException if the value cannot be
                converted
866      */
867     public static BigDecimal createBigDecimal(final String
           str) {
868         if (str == null) {
869             return null;
870         }
871
872         // handle JDK1.3.1 bug where "" throws
873         // IndexOutOfBoundsException
874         if (StringUtils.isBlank(str)) {
875             throw new NumberFormatException(
876                 "A blank string is not a valid number");
877         }
878         if (str.trim().startsWith("--")) {
879
880             // this is protection for poorness in
```

```
881             // java.lang.BigDecimal. it accepts this as a
                   legal value,
882             // but it does not appear to be in
                   specification of class.
883             // OS X Java parses it to a wrong value.
884             throw new NumberFormatException(str + " is not
                   a valid number.");
885         }
886         return new BigDecimal(str);
887     }
888 
889 
890     // Min in array
891     //
           ----------------------------------------------------------------

892     /**
893      * <p>Returns the minimum value in an array.</p>
894      *
895      * @param array  an array, must not be null or empty
896      * @return the minimum value in the array
897      * @throws IllegalArgumentException if <code>array</
                code> is
898      *  <code>null</code>
899      * @throws IllegalArgumentException if <code>array</
                code> is empty
900      */
901     public static long min(final long[] array) {
902 
903         // Validates input
904         validateArray(array);
905 
906         // Finds and returns min
907         long min = array[0];
908         for (int i = 1; i < array.length; i++) {
909             if (array[i] < min) {
910                 min = array[i];
911             }
912         }
913 
914         return min;
915     }
916 
917     /**
918      * <p>Returns the minimum value in an array.</p>
919      *
```

```
920         * @param array  an array, must not be null or empty
921         * @return the minimum value in the array
922         * @throws IllegalArgumentException if <code>array</
                code> is
923         *  <code>null</code>
924         * @throws IllegalArgumentException if <code>array</
                code> is empty
925         */
926        public static int min(final int[] array) {
927
928            // Validates input
929            validateArray(array);
930
931            // Finds and returns min
932            int min = array[0];
933            for (int j = 1; j < array.length; j++) {
934                if (array[j] < min) {
935                    min = array[j];
936                }
937            }
938
939            return min;
940        }
941
942        /**
943         * <p>Returns the minimum value in an array.</p>
944         *
945         * @param array  an array, must not be null or empty
946         * @return the minimum value in the array
947         * @throws IllegalArgumentException if <code>array</
                code> is
948         *  <code>null</code>
949         * @throws IllegalArgumentException if <code>array</
                code> is empty
950         */
951        public static short min(final short[] array) {
952
953            // Validates input
954            validateArray(array);
955
956            // Finds and returns min
957            short min = array[0];
958            for (int i = 1; i < array.length; i++) {
959                if (array[i] < min) {
960                    min = array[i];
961                }
```

```
962         }
963 
964         return min;
965     }
966 
967     /**
968      * <p>Returns the minimum value in an array.</p>
969      *
970      * @param array  an array, must not be null or empty
971      * @return the minimum value in the array
972      * @throws IllegalArgumentException if <code>array</
         code> is
973      *  <code>null</code>
974      * @throws IllegalArgumentException if <code>array</
         code> is empty
975      */
976     public static byte min(final byte[] array) {
977 
978         // Validates input
979         validateArray(array);
980 
981         // Finds and returns min
982         byte min = array[0];
983         for (int i = 1; i < array.length; i++) {
984             if (array[i] < min) {
985                 min = array[i];
986             }
987         }
988 
989         return min;
990     }
991 
992      /**
993      * <p>Returns the minimum value in an array.</p>
994      *
995      * @param array  an array, must not be null or empty
996      * @return the minimum value in the array
997      * @throws IllegalArgumentException if <code>array</
         code> is
998      *  <code>null</code>
999      * @throws IllegalArgumentException if <code>array</
         code> is empty
1000     * @see IEEE754rUtils#min(double[]) IEEE754rUtils for a
          version of
1001     *  this method that handles NaN differently
1002     */
```

```
1003        public static double min(final double[] array) {
1004
1005            // Validates input
1006            validateArray(array);
1007
1008            // Finds and returns min
1009            double min = array[0];
1010            for (int i = 1; i < array.length; i++) {
1011                if (Double.isNaN(array[i])) {
1012                    return Double.NaN;
1013                }
1014                if (array[i] < min) {
1015                    min = array[i];
1016                }
1017            }
1018
1019            return min;
1020        }
1021
1022        /**
1023         * <p>Returns the minimum value in an array.</p>
1024         *
1025         * @param array  an array, must not be null or empty
1026         * @return the minimum value in the array
1027         * @throws IllegalArgumentException if <code>array</
                code> is
1028         *  <code>null</code>
1029         * @throws IllegalArgumentException if <code>array</
                code> is empty
1030         * @see IEEE754rUtils#min(float[]) IEEE754rUtils for a
                version of
1031         *  this method that handles NaN differently
1032         */
1033        public static float min(final float[] array) {
1034
1035            // Validates input
1036            validateArray(array);
1037
1038            // Finds and returns min
1039            float min = array[0];
1040            for (int i = 1; i < array.length; i++) {
1041                if (Float.isNaN(array[i])) {
1042                    return Float.NaN;
1043                }
1044                if (array[i] < min) {
1045                    min = array[i];
```

```
1046            }
1047        }
1048
1049        return min;
1050    }
1051
1052
1053    // Max in array
1054    //
       -----------------------------------------------------------------

1055    /**
1056     * <p>Returns the maximum value in an array.</p>
1057     *
1058     * @param array  an array, must not be null or empty
1059     * @return the minimum value in the array
1060     * @throws IllegalArgumentException if <code>array</
          code> is
1061     *  <code>null</code>
1062     * @throws IllegalArgumentException if <code>array</
          code> is empty
1063     */
1064    public static long max(final long[] array) {
1065
1066        // Validates input
1067        validateArray(array);
1068
1069        // Finds and returns max
1070        long max = array[0];
1071        for (int j = 1; j < array.length; j++) {
1072            if (array[j] > max) {
1073                max = array[j];
1074            }
1075        }
1076
1077        return max;
1078    }
1079
1080    /**
1081     * <p>Returns the maximum value in an array.</p>
1082     *
1083     * @param array  an array, must not be null or empty
1084     * @return the minimum value in the array
1085     * @throws IllegalArgumentException if <code>array</
          code> is
1086     *  <code>null</code>
```

```
1087      * @throws IllegalArgumentException if <code>array</
              code> is empty
1088      */
1089     public static int max(final int[] array) {
1090 
1091         // Validates input
1092         validateArray(array);
1093 
1094         // Finds and returns max
1095         int max = array[0];
1096         for (int j = 1; j < array.length; j++) {
1097             if (array[j] > max) {
1098                 max = array[j];
1099             }
1100         }
1101 
1102         return max;
1103     }
1104 
1105     /**
1106      * <p>Returns the maximum value in an array.</p>
1107      *
1108      * @param array  an array, must not be null or empty
1109      * @return the minimum value in the array
1110      * @throws IllegalArgumentException if <code>array</
              code> is
1111      *  <code>null</code>
1112      * @throws IllegalArgumentException if <code>array</
              code> is empty
1113      */
1114     public static short max(final short[] array) {
1115 
1116         // Validates input
1117         validateArray(array);
1118 
1119         // Finds and returns max
1120         short max = array[0];
1121         for (int i = 1; i < array.length; i++) {
1122             if (array[i] > max) {
1123                 max = array[i];
1124             }
1125         }
1126 
1127         return max;
1128     }
1129 
```

```
1130        /**
1131         * <p>Returns the maximum value in an array.</p>
1132         *
1133         * @param array  an array, must not be null or empty
1134         * @return the minimum value in the array
1135         * @throws IllegalArgumentException if <code>array</
                code> is
1136         *  <code>null</code>
1137         * @throws IllegalArgumentException if <code>array</
                code> is empty
1138         */
1139        public static byte max(final byte[] array) {
1140
1141            // Validates input
1142            validateArray(array);
1143
1144            // Finds and returns max
1145            byte max = array[0];
1146            for (int i = 1; i < array.length; i++) {
1147                if (array[i] > max) {
1148                    max = array[i];
1149                }
1150            }
1151
1152            return max;
1153        }
1154
1155        /**
1156         * <p>Returns the maximum value in an array.</p>
1157         *
1158         * @param array  an array, must not be null or empty
1159         * @return the minimum value in the array
1160         * @throws IllegalArgumentException if <code>array</
                code> is
1161         *  <code>null</code>
1162         * @throws IllegalArgumentException if <code>array</
                code> is empty
1163         * @see IEEE754rUtils#max(double[]) IEEE754rUtils for a
                 version of
1164         *  this method that handles NaN differently
1165         */
1166        public static double max(final double[] array) {
1167
1168            // Validates input
1169            validateArray(array);
1170
```

```
1171            // Finds and returns max
1172            double max = array[0];
1173            for (int j = 1; j < array.length; j++) {
1174                if (Double.isNaN(array[j])) {
1175                    return Double.NaN;
1176                }
1177                if (array[j] > max) {
1178                    max = array[j];
1179                }
1180            }
1181
1182            return max;
1183        }
1184
1185        /**
1186         * <p>Returns the maximum value in an array.</p>
1187         *
1188         * @param array  an array, must not be null or empty
1189         * @return the minimum value in the array
1190         * @throws IllegalArgumentException if <code>array</
                code> is
1191         *  <code>null</code>
1192         * @throws IllegalArgumentException if <code>array</
                code> is empty
1193         * @see IEEE754rUtils#max(float[]) IEEE754rUtils for a
                version of
1194         *  this method that handles NaN differently
1195         */
1196        public static float max(final float[] array) {
1197
1198            // Validates input
1199            validateArray(array);
1200
1201            // Finds and returns max
1202            float max = array[0];
1203            for (int j = 1; j < array.length; j++) {
1204                if (Float.isNaN(array[j])) {
1205                    return Float.NaN;
1206                }
1207                if (array[j] > max) {
1208                    max = array[j];
1209                }
1210            }
1211
1212            return max;
1213        }
```

```
1214
1215        /**
1216         * Checks if the specified array is neither null nor
                empty.
1217         *
1218         * @param array  the array to check
1219         * @throws IllegalArgumentException if {@code array} is
                either
1220         *  {@code null} or empty
1221         */
1222        private static void validateArray(final Object array) {
1223            if (array == null) {
1224                throw new IllegalArgumentException("The Array
                        must not be null");
1225            } else if (Array.getLength(array) == 0) {
1226                throw new IllegalArgumentException("Array
                        cannot be empty.");
1227            }
1228        }
1229
1230
1231        // 3 param min
1232        //
               -----------------------------------------------------------------

1233        /**
1234         * <p>Gets the minimum of three <code>long</code>
                values.</p>
1235         *
1236         * @param a  value 1
1237         * @param b  value 2
1238         * @param c  value 3
1239         * @return  the smallest of the values
1240         */
1241        public static long min(long a, final long b, final long
                c) {
1242            if (b < a) {
1243                a = b;
1244            }
1245            if (c < a) {
1246                a = c;
1247            }
1248            return a;
1249        }
1250
1251        /**
```

```
1252        * <p>Gets the minimum of three <code>int</code> values
               .</p>
1253        *
1254        * @param a  value 1
1255        * @param b  value 2
1256        * @param c  value 3
1257        * @return  the smallest of the values
1258        */
1259       public static int min(int a, final int b, final int c)
               {
1260           if (b < a) {
1261               a = b;
1262           }
1263           if (c < a) {
1264               a = c;
1265           }
1266           return a;
1267       }
1268
1269       /**
1270        * <p>Gets the minimum of three <code>short</code>
               values.</p>
1271        *
1272        * @param a  value 1
1273        * @param b  value 2
1274        * @param c  value 3
1275        * @return  the smallest of the values
1276        */
1277       public static short min(short a, final short b, final
               short c) {
1278           if (b < a) {
1279               a = b;
1280           }
1281           if (c < a) {
1282               a = c;
1283           }
1284           return a;
1285       }
1286
1287       /**
1288        * <p>Gets the minimum of three <code>byte</code>
               values.</p>
1289        *
1290        * @param a  value 1
1291        * @param b  value 2
1292        * @param c  value 3
```

```
1293        * @return  the smallest of the values
1294        */
1295       public static byte min(byte a, final byte b, final byte
               c) {
1296           if (b < a) {
1297               a = b;
1298           }
1299           if (c < a) {
1300               a = c;
1301           }
1302           return a;
1303       }
1304
1305       /**
1306        * <p>Gets the minimum of three <code>double</code>
               values.</p>
1307        *
1308        * <p>If any value is <code>NaN</code>, <code>NaN</code
               > is
1309        * returned. Infinity is handled.</p>
1310        *
1311        * @param a  value 1
1312        * @param b  value 2
1313        * @param c  value 3
1314        * @return  the smallest of the values
1315        * @see IEEE754rUtils#min(double, double, double) for a
                version of
1316        *  this method that handles NaN differently
1317        */
1318       public static double min(final double a, final double b
               , final double c) {
1319           return Math.min(Math.min(a, b), c);
1320       }
1321
1322       /**
1323        * <p>Gets the minimum of three <code>float</code>
               values.</p>
1324        *
1325        * <p>If any value is <code>NaN</code>, <code>NaN</code
               > is
1326        * returned. Infinity is handled.</p>
1327        *
1328        * @param a  value 1
1329        * @param b  value 2
1330        * @param c  value 3
1331        * @return  the smallest of the values
```

```
1332      * @see IEEE754rUtils#min(float, float, float) for a
              version of
1333      *  this method that handles NaN differently
1334      */
1335     public static float min(final float a, final float b,
              final float c) {
1336         return Math.min(Math.min(a, b), c);
1337     }
1338 
1339 
1340     // 3 param max
1341     //
              -----------------------------------------------------------------
1342     /**
1343      * <p>Gets the maximum of three <code>long</code>
              values.</p>
1344      *
1345      * @param a  value 1
1346      * @param b  value 2
1347      * @param c  value 3
1348      * @return  the largest of the values
1349      */
1350     public static long max(long a, final long b, final long
              c) {
1351         if (b > a) {
1352             a = b;
1353         }
1354         if (c > a) {
1355             a = c;
1356         }
1357         return a;
1358     }
1359 
1360     /**
1361      * <p>Gets the maximum of three <code>int</code> values
              .</p>
1362      *
1363      * @param a  value 1
1364      * @param b  value 2
1365      * @param c  value 3
1366      * @return  the largest of the values
1367      */
1368     public static int max(int a, final int b, final int c)
              {
1369         if (b > a) {
```

```
1370            a = b;
1371        }
1372        if (c > a) {
1373            a = c;
1374        }
1375        return a;
1376    }
1377
1378    /**
1379     * <p>Gets the maximum of three <code>short</code>
            values.</p>
1380     *
1381     * @param a  value 1
1382     * @param b  value 2
1383     * @param c  value 3
1384     * @return  the largest of the values
1385     */
1386    public static short max(short a, final short b, final
           short c) {
1387        if (b > a) {
1388            a = b;
1389        }
1390        if (c > a) {
1391            a = c;
1392        }
1393        return a;
1394    }
1395
1396    /**
1397     * <p>Gets the maximum of three <code>byte</code>
            values.</p>
1398     *
1399     * @param a  value 1
1400     * @param b  value 2
1401     * @param c  value 3
1402     * @return  the largest of the values
1403     */
1404    public static byte max(byte a, final byte b, final byte
           c) {
1405        if (b > a) {
1406            a = b;
1407        }
1408        if (c > a) {
1409            a = c;
1410        }
1411        return a;
```

```
1412        }
1413
1414        /**
1415         * <p>Gets the maximum of three <code>double</code>
                values.</p>
1416         *
1417         * <p>If any value is <code>NaN</code>, <code>NaN</code
                > is
1418         * returned. Infinity is handled.</p>
1419         *
1420         * @param a  value 1
1421         * @param b  value 2
1422         * @param c  value 3
1423         * @return  the largest of the values
1424         * @see IEEE754rUtils#max(double, double, double) for a
                 version of
1425         *  this method that handles NaN differently
1426         */
1427        public static double max(final double a, final double b
               , final double c) {
1428            return Math.max(Math.max(a, b), c);
1429        }
1430
1431        /**
1432         * <p>Gets the maximum of three <code>float</code>
                values.</p>
1433         *
1434         * <p>If any value is <code>NaN</code>, <code>NaN</code
                > is
1435         * returned. Infinity is handled.</p>
1436         *
1437         * @param a  value 1
1438         * @param b  value 2
1439         * @param c  value 3
1440         * @return  the largest of the values
1441         * @see IEEE754rUtils#max(float, float, float) for a
                version of
1442         *  this method that handles NaN differently
1443         */
1444        public static float max(final float a, final float b,
               final float c) {
1445            return Math.max(Math.max(a, b), c);
1446        }
1447
1448
1449        //
```

```
    -----------------------------------------------------------------------

1450    /**
1451     * <p>Checks whether the <code>String</code> contains
            only
1452     * digit characters.</p>
1453     *
1454     * <p><code>Null</code> and empty String will return
1455     * <code>false</code>.</p>
1456     *
1457     * @param str  the <code>String</code> to check
1458     * @return <code>true</code> if str contains only
            Unicode numeric
1459     */
1460    public static boolean isDigits(final String str) {
1461        if (StringUtils.isEmpty(str)) {
1462            return false;
1463        }
1464        for (int i = 0; i < str.length(); i++) {
1465            if (!Character.isDigit(str.charAt(i))) {
1466                return false;
1467            }
1468        }
1469        return true;
1470    }
1471
1472    /**
1473     * <p>Checks whether the String a valid Java number.</p
            >
1474     *
1475     * <p>Valid numbers include hexadecimal marked with the
1476     * <code>0x</code> qualifier, scientific notation and
            numbers
1477     * marked with a type qualifier (e.g. 123L).</p>
1478     *
1479     * <p><code>Null</code> and empty String will return
1480     * <code>false</code>.</p>
1481     *
1482     * @param str  the <code>String</code> to check
1483     * @return <code>true</code> if the string is a
            correctly
1484     *  formatted number
1485     */
1486    public static boolean isNumber(final String str) {
1487        if (StringUtils.isEmpty(str)) {
1488            return false;
```

```
1489            }
1490            final char[] chars = str.toCharArray();
1491            int sz = chars.length;
1492            boolean hasExp = false;
1493            boolean hasDecPoint = false;
1494            boolean allowSigns = false;
1495            boolean foundDigit = false;
1496
1497            // deal with any possible sign up front
1498            final int start = (chars[0] == '-') ? 1 : 0;
1499            if (sz > start + 1 && chars[start] == '0' && chars[
                   start + 1] == 'x') {
1500                int i = start + 2;
1501                if (i == sz) {
1502                    return false; // str == "0x"
1503                }
1504
1505                // checking hex (it can't be anything else)
1506                for (; i < chars.length; i++) {
1507                    if ((chars[i] < '0' || chars[i] > '9')
1508                        && (chars[i] < 'a' || chars[i] > 'f')
1509                        && (chars[i] < 'A' || chars[i] > 'F'))
                               {
1510                        return false;
1511                    }
1512                }
1513                return true;
1514            }
1515            sz--;
1516
1517            // don't want to loop to the last char, check it
1518            // afterwords for type qualifiers
1519
1520            int i = start;
1521
1522            // loop to the next to last char or to the last
                   char if we
1523            // need another digit to make a valid number
1524            // (e.g. chars[0..5] = "1234E")
1525            while (i < sz || (i < sz + 1 && allowSigns && !
                   foundDigit)) {
1526                if (chars[i] >= '0' && chars[i] <= '9') {
1527                    foundDigit = true;
1528                    allowSigns = false;
1529                } else if (chars[i] == '.') {
1530                    if (hasDecPoint || hasExp) {
```

```
1531 |
1532 |                     // two decimal points or dec in
     |                        exponent
1533 |                     return false;
1534 |                 }
1535 |                 hasDecPoint = true;
1536 |             } else if (chars[i] == 'e' || chars[i] == 'E')
     |                {
1537 |
1538 |                 // we've already taken care of hex.
1539 |                 if (hasExp) {
1540 |
1541 |                     // two E's
1542 |                     return false;
1543 |                 }
1544 |                 if (!foundDigit) {
1545 |                     return false;
1546 |                 }
1547 |                 hasExp = true;
1548 |                 allowSigns = true;
1549 |             } else if (chars[i] == '+' || chars[i] == '-')
     |                {
1550 |                 if (!allowSigns) {
1551 |                     return false;
1552 |                 }
1553 |                 allowSigns = false;
1554 |                 foundDigit = false; // we need a digit
     |                    after the E
1555 |             } else {
1556 |                 return false;
1557 |             }
1558 |             i++;
1559 |         }
1560 |         if (i < chars.length) {
1561 |             if (chars[i] >= '0' && chars[i] <= '9') {
1562 |
1563 |                 // no type qualifier, OK
1564 |                 return true;
1565 |             }
1566 |             if (chars[i] == 'e' || chars[i] == 'E') {
1567 |
1568 |                 // can't have an E at the last byte
1569 |                 return false;
1570 |             }
1571 |             if (chars[i] == '.') {
1572 |                 if (hasDecPoint || hasExp) {
```

```
1573 |
1574 |                     // two decimal points or dec in
     |                        exponent
1575 |                     return false;
1576 |                 }
1577 |
1578 |                 // single trailing decimal point after non-
     |                    exponent is
1579 |                 // ok
1580 |                 return foundDigit;
1581 |             }
1582 |             if (!allowSigns
1583 |                 && (chars[i] == 'd'
1584 |                     || chars[i] == 'D'
1585 |                     || chars[i] == 'f'
1586 |                     || chars[i] == 'F')) {
1587 |                 return foundDigit;
1588 |             }
1589 |             if (chars[i] == 'l'
1590 |                 || chars[i] == 'L') {
1591 |
1592 |                 // not allowing L with an exponent or
     |                    decimal point
1593 |                 return foundDigit && !hasExp && !
     |                    hasDecPoint;
1594 |             }
1595 |
1596 |             // last character is illegal
1597 |             return false;
1598 |         }
1599 |
1600 |         // allowSigns is true iff the val ends in 'E'
1601 |         // found digit it to make sure weird stuff like '.'
     |             and '1E-'
1602 |         // doesn't pass
1603 |         return !allowSigns && foundDigit;
1604 |     }
1605 | }
```

E.3.2 Defects-Sources/Lang-1/NumberUtils-fixed-A.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under
       one or more
3   * contributor license agreements.  See the NOTICE file
       distributed
4   * with this work for additional information regarding
       copyright
5   * ownership. The ASF licenses this file to You under the
       Apache
6   * License, Version 2.0 (the "License"); you may not use
       this file
7   * except in compliance with the License.  You may obtain a
        copy of
8   * the License at
9   *
10  *      http://www.apache.org/licenses/LICENSE-2.0
11  *
12  * Unless required by applicable law or agreed to in
       writing,
13  * software distributed under the License is distributed on
        an "AS
14  * IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
        either
15  * express or implied. See the License for the specific
       language
16  * governing permissions and limitations under the License.
17  */
18 package org.apache.commons.lang3.math;
19
20
21 import java.lang.reflect.Array;
22 import java.math.BigDecimal;
23 import java.math.BigInteger;
24 import org.apache.commons.lang3.StringUtils;
25
26
27 /**
28  * <p>Provides extra functionality for Java Number classes
       .</p>
29  *
30  * @since 2.0
31  * @version $Id$
32  */
33 public class NumberUtils {
```

```
34
35      /** Reusable Long constant for zero. */
36      public static final Long LONG_ZERO = Long.valueOf(0L);
37
38      /** Reusable Long constant for one. */
39      public static final Long LONG_ONE = Long.valueOf(1L);
40
41      /** Reusable Long constant for minus one. */
42      public static final Long LONG_MINUS_ONE = Long.valueOf
           (-1L);
43
44      /** Reusable Integer constant for zero. */
45      public static final Integer INTEGER_ZERO = Integer.
           valueOf(0);
46
47      /** Reusable Integer constant for one. */
48      public static final Integer INTEGER_ONE = Integer.
           valueOf(1);
49
50      /** Reusable Integer constant for minus one. */
51      public static final Integer INTEGER_MINUS_ONE = Integer
           .valueOf(-1);
52
53      /** Reusable Short constant for zero. */
54      public static final Short SHORT_ZERO = Short.valueOf((
           short) 0);
55
56      /** Reusable Short constant for one. */
57      public static final Short SHORT_ONE = Short.valueOf((
           short) 1);
58
59      /** Reusable Short constant for minus one. */
60      public static final Short SHORT_MINUS_ONE = Short.
           valueOf((short) -1);
61
62      /** Reusable Byte constant for zero. */
63      public static final Byte BYTE_ZERO = Byte.valueOf((byte
           ) 0);
64
65      /** Reusable Byte constant for one. */
66      public static final Byte BYTE_ONE = Byte.valueOf((byte)
            1);
67
68      /** Reusable Byte constant for minus one. */
69      public static final Byte BYTE_MINUS_ONE = Byte.valueOf
           ((byte) -1);
```

```
70
71      /** Reusable Double constant for zero. */
72      public static final Double DOUBLE_ZERO = Double.valueOf
            (0.0d);
73
74      /** Reusable Double constant for one. */
75      public static final Double DOUBLE_ONE = Double.valueOf
            (1.0d);
76
77      /** Reusable Double constant for minus one. */
78      public static final Double DOUBLE_MINUS_ONE = Double.
            valueOf(-1.0d);
79
80      /** Reusable Float constant for zero. */
81      public static final Float FLOAT_ZERO = Float.valueOf
            (0.0f);
82
83      /** Reusable Float constant for one. */
84      public static final Float FLOAT_ONE = Float.valueOf(1.0
            f);
85
86      /** Reusable Float constant for minus one. */
87      public static final Float FLOAT_MINUS_ONE = Float.
            valueOf(-1.0f);
88
89
90      /**
91       * <p><code>NumberUtils</code> instances should NOT be
             constructed
92       * in standard programming. Instead, the class should
             be used as
93       * <code>NumberUtils.toInt("6");</code>.</p>
94       *
95       * <p>This constructor is public to permit tools that
             require a
96       * JavaBean instance to operate.</p>
97       */
98      public NumberUtils() {
99          super();
100     }
101
102
103     //
            -----------------------------------------------------------------
104     /**
```

```
105         * <p>Convert a <code>String</code> to an <code>int</
                code>,
106         * returning <code>zero</code> if the conversion fails
                .</p>
107         *
108         * <p>If the string is <code>null</code>, <code>zero</
                code> is
109         * returned.</p>
110         *
111         * <pre>
112         *   NumberUtils.toInt(null) = 0
113         *   NumberUtils.toInt("")   = 0
114         *   NumberUtils.toInt("1")  = 1
115         * </pre>
116         *
117         * @param str  the string to convert, may be null
118         * @return the int represented by the string, or <code>
                zero</code>
119         *  if conversion fails
120         * @since 2.1
121         */
122        public static int toInt(final String str) {
123            return toInt(str, 0);
124        }
125
126        /**
127         * <p>Convert a <code>String</code> to an <code>int</
                code>,
128         * returning a default value if the conversion fails.</
                p>
129         *
130         * <p>If the string is <code>null</code>, the default
                value is
131         * returned.</p>
132         *
133         * <pre>
134         *   NumberUtils.toInt(null, 1) = 1
135         *   NumberUtils.toInt("", 1)   = 1
136         *   NumberUtils.toInt("1", 0)  = 1
137         * </pre>
138         *
139         * @param str  the string to convert, may be null
140         * @param defaultValue  the default value
141         * @return the int represented by the string, or the
                default if
142         *  conversion fails
```

```
143         * @since 2.1
144         */
145        public static int toInt(final String str, final int
               defaultValue) {
146            if(str == null) {
147                return defaultValue;
148            }
149            try {
150                return Integer.parseInt(str);
151            } catch (final NumberFormatException nfe) {
152                return defaultValue;
153            }
154        }
155
156        /**
157         * <p>Convert a <code>String</code> to a <code>long</
               code>,
158         * returning <code>zero</code> if the conversion fails
               .</p>
159         *
160         * <p>If the string is <code>null</code>, <code>zero</
               code> is
161         * returned.</p>
162         *
163         * <pre>
164         *   NumberUtils.toLong(null) = 0L
165         *   NumberUtils.toLong("")   = 0L
166         *   NumberUtils.toLong("1")  = 1L
167         * </pre>
168         *
169         * @param str  the string to convert, may be null
170         * @return the long represented by the string, or <code
               >0</code>
171         *  if conversion fails
172         * @since 2.1
173         */
174        public static long toLong(final String str) {
175            return toLong(str, 0L);
176        }
177
178        /**
179         * <p>Convert a <code>String</code> to a <code>long</
               code>,
180         * returning a default value if the conversion fails.</
               p>
181         *
```

```
182      * <p>If the string is <code>null</code>, the default
             value is
183      * returned.</p>
184      *
185      * <pre>
186      *   NumberUtils.toLong(null, 1L) = 1L
187      *   NumberUtils.toLong("", 1L)   = 1L
188      *   NumberUtils.toLong("1", 0L)  = 1L
189      * </pre>
190      *
191      * @param str  the string to convert, may be null
192      * @param defaultValue  the default value
193      * @return the long represented by the string, or the
             default if
194      *  conversion fails
195      * @since 2.1
196      */
197     public static long toLong(final String str, final long
           defaultValue) {
198         if (str == null) {
199             return defaultValue;
200         }
201         try {
202             return Long.parseLong(str);
203         } catch (final NumberFormatException nfe) {
204             return defaultValue;
205         }
206     }
207
208     /**
209      * <p>Convert a <code>String</code> to a <code>float</
             code>,
210      * returning <code>0.0f</code> if the conversion fails
             .</p>
211      *
212      * <p>If the string <code>str</code> is <code>null</
             code>,
213      * <code>0.0f</code> is returned.</p>
214      *
215      * <pre>
216      *   NumberUtils.toFloat(null)   = 0.0f
217      *   NumberUtils.toFloat("")     = 0.0f
218      *   NumberUtils.toFloat("1.5")  = 1.5f
219      * </pre>
220      *
221      * @param str the string to convert, may be <code>null
```

```
         </code>
222       * @return the float represented by the string, or
223       *  <code>0.0f</code> if conversion fails
224       * @since 2.1
225       */
226      public static float toFloat(final String str) {
227          return toFloat(str, 0.0f);
228      }
229
230      /**
231       * <p>Convert a <code>String</code> to a <code>float</
             code>,
232       * returning a default value if the conversion fails.</
             p>
233       *
234       * <p>If the string <code>str</code> is <code>null</
             code>, the
235       * default value is returned.</p>
236       *
237       * <pre>
238       *   NumberUtils.toFloat(null, 1.1f)   = 1.0f
239       *   NumberUtils.toFloat("", 1.1f)     = 1.1f
240       *   NumberUtils.toFloat("1.5", 0.0f)  = 1.5f
241       * </pre>
242       *
243       * @param str the string to convert, may be <code>null
             </code>
244       * @param defaultValue the default value
245       * @return the float represented by the string, or
             defaultValue
246       *  if conversion fails
247       * @since 2.1
248       */
249      public static float toFloat(final String str, final
           float defaultValue) {
250        if (str == null) {
251            return defaultValue;
252        }
253        try {
254            return Float.parseFloat(str);
255        } catch (final NumberFormatException nfe) {
256            return defaultValue;
257        }
258      }
259
260      /**
```

```
261         * <p>Convert a <code>String</code> to a <code>double</
                code>,
262         * returning <code>0.0d</code> if the conversion fails
                .</p>
263         *
264         * <p>If the string <code>str</code> is <code>null</
                code>,
265         * <code>0.0d</code> is returned.</p>
266         *
267         * <pre>
268         *   NumberUtils.toDouble(null)   = 0.0d
269         *   NumberUtils.toDouble("")     = 0.0d
270         *   NumberUtils.toDouble("1.5")  = 1.5d
271         * </pre>
272         *
273         * @param str the string to convert, may be <code>null
                </code>
274         * @return the double represented by the string, or
275         *  <code>0.0d</code> if conversion fails
276         * @since 2.1
277         */
278        public static double toDouble(final String str) {
279            return toDouble(str, 0.0d);
280        }
281
282        /**
283         * <p>Convert a <code>String</code> to a <code>double</
                code>,
284         * returning a default value if the conversion fails.</
                p>
285         *
286         * <p>If the string <code>str</code> is <code>null</
                code>, the
287         * default value is returned.</p>
288         *
289         * <pre>
290         *   NumberUtils.toDouble(null, 1.1d)   = 1.1d
291         *   NumberUtils.toDouble("", 1.1d)     = 1.1d
292         *   NumberUtils.toDouble("1.5", 0.0d)  = 1.5d
293         * </pre>
294         *
295         * @param str the string to convert, may be <code>null
                </code>
296         * @param defaultValue the default value
297         * @return the double represented by the string, or
                defaultValue
```

```
298         *  if conversion fails
299         * @since 2.1
300         */
301        public static double toDouble(final String str, final
               double defaultValue) {
302            if (str == null) {
303                return defaultValue;
304            }
305            try {
306                return Double.parseDouble(str);
307            } catch (final NumberFormatException nfe) {
308                return defaultValue;
309            }
310        }
311
312
313         //
               -----------------------------------------------------------------------
314         /**
315         * <p>Convert a <code>String</code> to a <code>byte</
               code>,
316         * returning <code>zero</code> if the conversion fails
               .</p>
317         *
318         * <p>If the string is <code>null</code>, <code>zero</
               code> is
319         * returned.</p>
320         *
321         * <pre>
322         *   NumberUtils.toByte(null) = 0
323         *   NumberUtils.toByte("")   = 0
324         *   NumberUtils.toByte("1")  = 1
325         * </pre>
326         *
327         * @param str  the string to convert, may be null
328         * @return the byte represented by the string, or
329         *  <code>zero</code> if conversion fails
330         * @since 2.5
331         */
332        public static byte toByte(final String str) {
333            return toByte(str, (byte) 0);
334        }
335
336        /**
337         * <p>Convert a <code>String</code> to a <code>byte</
```

```
         code>,
338      * returning a default value if the conversion fails.</
             p>
339      *
340      * <p>If the string is <code>null</code>, the default
             value is
341      * returned.</p>
342      *
343      * <pre>
344      *   NumberUtils.toByte(null, 1) = 1
345      *   NumberUtils.toByte("", 1)   = 1
346      *   NumberUtils.toByte("1", 0)  = 1
347      * </pre>
348      *
349      * @param str  the string to convert, may be null
350      * @param defaultValue  the default value
351      * @return the byte represented by the string, or the
             default if
352      *  conversion fails
353      * @since 2.5
354      */
355     public static byte toByte(final String str, final byte
           defaultValue) {
356         if(str == null) {
357             return defaultValue;
358         }
359         try {
360             return Byte.parseByte(str);
361         } catch (final NumberFormatException nfe) {
362             return defaultValue;
363         }
364     }
365
366     /**
367      * <p>Convert a <code>String</code> to a <code>short</
             code>,
368      * returning <code>zero</code> if the conversion fails
             .</p>
369      *
370      * <p>If the string is <code>null</code>, <code>zero</
             code> is
371      * returned.</p>
372      *
373      * <pre>
374      *   NumberUtils.toShort(null) = 0
375      *   NumberUtils.toShort("")   = 0
```

```
376         *   NumberUtils.toShort("1")  = 1
377         * </pre>
378         *
379         * @param str  the string to convert, may be null
380         * @return the short represented by the string, or
381         *  <code>zero</code> if conversion fails
382         * @since 2.5
383         */
384        public static short toShort(final String str) {
385            return toShort(str, (short) 0);
386        }
387
388        /**
389         * <p>Convert a <code>String</code> to an <code>short</
                code>,
390         * returning a default value if the conversion fails.</
                p>
391         *
392         * <p>If the string is <code>null</code>, the default
                value is
393         * returned.</p>
394         *
395         * <pre>
396         *   NumberUtils.toShort(null, 1) = 1
397         *   NumberUtils.toShort("", 1)   = 1
398         *   NumberUtils.toShort("1", 0)  = 1
399         * </pre>
400         *
401         * @param str  the string to convert, may be null
402         * @param defaultValue  the default value
403         * @return the short represented by the string, or the
                default if
404         *  conversion fails
405         * @since 2.5
406         */
407        public static short toShort(final String str, final
               short defaultValue) {
408            if(str == null) {
409                return defaultValue;
410            }
411            try {
412                return Short.parseShort(str);
413            } catch (final NumberFormatException nfe) {
414                return defaultValue;
415            }
416        }
```

```
417
418
419     //
            ------------------------------------------------------------
420     // must handle Long, Float, Integer, Float, Short,
421     //                  BigDecimal, BigInteger and Byte
422     // useful methods:
423     // Byte.decode(String)
424     // Byte.valueOf(String,int radix)
425     // Byte.valueOf(String)
426     // Double.valueOf(String)
427     // Float.valueOf(String)
428     // Float.valueOf(String)
429     // Integer.valueOf(String,int radix)
430     // Integer.valueOf(String)
431     // Integer.decode(String)
432     // Integer.getInteger(String)
433     // Integer.getInteger(String,int val)
434     // Integer.getInteger(String,Integer val)
435     // Integer.valueOf(String)
436     // Double.valueOf(String)
437     // new Byte(String)
438     // Long.valueOf(String)
439     // Long.getLong(String)
440     // Long.getLong(String,int)
441     // Long.getLong(String,Integer)
442     // Long.valueOf(String,int)
443     // Long.valueOf(String)
444     // Short.valueOf(String)
445     // Short.decode(String)
446     // Short.valueOf(String,int)
447     // Short.valueOf(String)
448     // new BigDecimal(String)
449     // new BigInteger(String)
450     // new BigInteger(String,int radix)
451     // Possible inputs:
452     // 45 45.5 45E7 4.5E7 Hex Oct Binary xxxF xxxD xxxf
            xxxd
453     // plus minus everything. Prolly more. A lot are not
            separable.
454
455     /**
456      * <p>Turns a string value into a java.lang.Number.</p>
457      *
458      * <p>If the string starts with {@code 0x} or {@code -0
```

```
         x} (lower
459      * or upper case) or {@code #} or {@code -#}, it will
             be
460      * interpreted as a hexadecimal Integer - or Long, if
             the number
461      * of digits after the prefix is more than 8 - or
             BigInteger if
462      * there are more than 16 digits. </p>
463      *
464      * <p>Then, the value is examined for a type qualifier
             on the end,
465      * i.e. one of <code>'f','F','d','D','l','L'</code>.
             If it is
466      * found, it starts trying to create successively
             larger types
467      * from the type specified until one is found that can
             represent
468      * the value.</p>
469      *
470      * <p>If a type specifier is not found, it will check
             for a
471      * decimal point and then try successively larger types
              from
472      * <code>Integer</code> to <code>BigInteger</code> and
             from
473      * <code>Float</code> to <code>BigDecimal</code>.</p>
474      *
475      * <p>
476      * Integral values with a leading {@code 0} will be
             interpreted as
477      * octal; the returned number will be Integer, Long or
             BigDecimal
478      * as appropriate.
479      * </p>
480      *
481      * <p>Returns <code>null</code> if the string is
482      * <code>null</code>.</p>
483      *
484      * <p>This method does not trim the input string, i.e.,
              strings
485      * with leading or trailing spaces will generate
486      * NumberFormatExceptions.</p>
487      *
488      * @param str  String containing a number, may be null
489      * @return Number created from the string (or null if
             the input is
```

```
490         *  null)
491         * @throws NumberFormatException if the value cannot be
                converted
492         */
493        public static Number createNumber(final String str)
494                throws NumberFormatException {
495            if (str == null) {
496                return null;
497            }
498            if (StringUtils.isBlank(str)) {
499                throw new NumberFormatException(
500                    "A blank string is not a valid number");
501            }
502
503            // Need to deal with all possible hex prefixes here
504            final String[] hex_prefixes = {"0x", "0X", "-0x", "
                   -0X", "#", "-#"};
505            int pfxLen = 0;
506            for(final String pfx : hex_prefixes) {
507                if (str.startsWith(pfx)) {
508                    pfxLen += pfx.length();
509                    break;
510                }
511            }
512            if (pfxLen > 0) { // we have a hex number
513                char firstSigDigit = 0; // strip leading zeroes
514                for(int i = pfxLen; i < str.length(); i++) {
515                    firstSigDigit = str.charAt(i);
516                    if (firstSigDigit == '0') { // count
                           leading zeroes
517                        pfxLen++;
518                    } else {
519                        break;
520                    }
521                }
522                final int hexDigits = str.length() - pfxLen;
523                if (hexDigits > 16 || (hexDigits == 16 &&
                       firstSigDigit > '7')) {
524
525                    // too many for Long
526                    return createBigInteger(str);
527                }
528                if (hexDigits > 8 || (hexDigits == 8 &&
                       firstSigDigit > '7')) {
529
530                    // too many for an int
```

```
531                 return createLong(str);
532             }
533             return createInteger(str);
534         }
535         final char lastChar = str.charAt(str.length() - 1);
536         String mant;
537         String dec;
538         String exp;
539         final int decPos = str.indexOf('.');
540         final int expPos = str.indexOf('e') + str.indexOf('
                E') + 1;
541 
542         // assumes both not present
543         // if both e and E are present, this is caught by
                the checks
544         // on expPos (which prevent IOOBE) and the parsing
                which will
545         // detect if e or E appear in a number due to using
                 the wrong
546         // offset
547 
548         int numDecimals = 0; // Check required precision (
                LANG-693)
549         if (decPos > -1) { // there is a decimal point
550 
551             if (expPos > -1) { // there is an exponent
552                 if (expPos < decPos || expPos > str.length
                        ()) {
553 
554                     // prevents double exponent causing
                            IOOBE
555                     throw new NumberFormatException(str
556                             + " is not a valid number.");
557                 }
558                 dec = str.substring(decPos + 1, expPos);
559             } else {
560                 dec = str.substring(decPos + 1);
561             }
562             mant = str.substring(0, decPos);
563             numDecimals = dec.length();
564 
565             // gets number of digits past
566             // the decimal to ensure no loss of precision
                    for
567             // floating point numbers.
568         } else {
```

```
569             if (expPos > -1) {
570                 if (expPos > str.length()) {
571 
572                     // prevents double exponent causing
                        IOOBE
573                     throw new NumberFormatException(str
574                         + " is not a valid number.");
575                 }
576                 mant = str.substring(0, expPos);
577             } else {
578                 mant = str;
579             }
580             dec = null;
581         }
582         if (!Character.isDigit(lastChar) && lastChar != '.'
               ) {
583             if (expPos > -1 && expPos < str.length() - 1) {
584                 exp = str.substring(expPos + 1, str.length
                       () - 1);
585             } else {
586                 exp = null;
587             }
588 
589             //Requesting a specific type..
590             final String numeric = str.substring(0, str.
                   length() - 1);
591             final boolean allZeros = isAllZeros(mant) &&
                   isAllZeros(exp);
592             switch (lastChar) {
593                 case 'l' :
594                 case 'L' :
595                     if (dec == null
596                         && exp == null
597                         && (numeric.charAt(0) == '-'
598                         && isDigits(numeric.substring(1))
599                         || isDigits(numeric))) {
600                         try {
601                             return createLong(numeric);
602                         } catch (final
                               NumberFormatException nfe) {
603 
604                             // NOPMD
605                             // Too big for a long
606                         }
607                         return createBigInteger(numeric);
608                     }
```

```
609                     throw new NumberFormatException(str
610                         + " is not a valid number.");
611                 case 'f' :
612                 case 'F' :
613                     try {
614                         final Float f = NumberUtils.
                               createFloat(numeric);
615                         if (!(f.isInfinite() || (f.
                               floatValue() == 0.0F
616                             && !allZeros))) {
617 
618                             // If it's too big for a float
                                or the
619                             // float value = 0 and the
                                string has
620                             // non-zeros in it, then float
                                does not
621                             // have the precision we want
622                             return f;
623                         }
624                     } catch (final NumberFormatException
                           nfe) {
625 
626                         // NOPMD
627                         // ignore the bad number
628                     }
629 
630                     //$FALL-THROUGH$
631                 case 'd' :
632                 case 'D' :
633                     try {
634                         final Double d = NumberUtils.
                               createDouble(numeric);
635                         if (!(d.isInfinite() || (d.
                               floatValue() == 0.0D
636                             && !allZeros))) {
637                             return d;
638                         }
639                     } catch (final NumberFormatException
                           nfe) {
640 
641                         // NOPMD
642                         // ignore the bad number
643                     }
644                     try {
645                         return createBigDecimal(numeric);
```

```
646                     } catch (final NumberFormatException e)
                            {
647
648                         // NOPMD
649                         // ignore the bad number
650                     }
651
652                     //$FALL-THROUGH$
653                 default :
654                     throw new NumberFormatException(str
655                         + " is not a valid number.");
656             }
657         }
658
659         //User doesn't have a preference on the return type
               , so let's
660         //start small and go from there...
661         if (expPos > -1 && expPos < str.length() - 1) {
662             exp = str.substring(expPos + 1, str.length());
663         } else {
664             exp = null;
665         }
666         if (dec == null && exp == null) {
667
668             // no decimal point and no exponent
669             //Must be an Integer, Long, Biginteger
670             try {
671                 return createInteger(str);
672             } catch (final NumberFormatException nfe) {
673
674                 // NOPMD
675                 // ignore the bad number
676             }
677             try {
678                 return createLong(str);
679             } catch (final NumberFormatException nfe) {
680
681                 // NOPMD
682                 // ignore the bad number
683             }
684             return createBigInteger(str);
685         }
686
687         //Must be a Float, Double, BigDecimal
688         final boolean allZeros = isAllZeros(mant) &&
               isAllZeros(exp);
```

```
689             try {
690                 if(numDecimals <= 7){
691
692                     // If number has 7 or fewer digits past the
                            decimal
693                     // point then make it a float
694                     final Float f = createFloat(str);
695                     if (!(f.isInfinite()
696                         || (f.floatValue() == 0.0F && !allZeros
                                ))) {
697                         return f;
698                     }
699                 }
700             } catch (final NumberFormatException nfe) {
701
702                 // NOPMD
703                 // ignore the bad number
704             }
705             try {
706                 if(numDecimals <= 16){
707
708                     // If number has between 8 and 16 digits
                            past the
709                     // decimal point then make it a double
710                     final Double d = createDouble(str);
711                     if (!(d.isInfinite()
712                         || (d.doubleValue() == 0.0D && !
                                allZeros))) {
713                         return d;
714                     }
715                 }
716             } catch (final NumberFormatException nfe) {
717
718                 // NOPMD
719                 // ignore the bad number
720             }
721
722             return createBigDecimal(str);
723         }
724
725         /**
726          * <p>Utility method for
727          * {@link #createNumber(java.lang.String)}.</p>
728          *
729          * <p>Returns <code>true</code> if s is <code>null</
                code>.</p>
```

```
730          *
731          * @param str  the String to check
732          * @return if it is all zeros or <code>null</code>
733          */
734         private static boolean isAllZeros(final String str) {
735             if (str == null) {
736                 return true;
737             }
738             for (int i = str.length() - 1; i >= 0; i--) {
739                 if (str.charAt(i) != '0') {
740                     return false;
741                 }
742             }
743             return str.length() > 0;
744         }
745 
746 
747         //
            ----------------------------------------------------------------------

748         /**
749          * <p>Convert a <code>String</code> to a <code>Float</
                code>.</p>
750          *
751          * <p>Returns <code>null</code> if the string is
752          * <code>null</code>.</p>
753          *
754          * @param str  a <code>String</code> to convert, may be
                 null
755          * @return converted <code>Float</code> (or null if the
                 input is
756          *  null)
757          * @throws NumberFormatException if the value cannot be
                 converted
758          */
759         public static Float createFloat(final String str) {
760             if (str == null) {
761                 return null;
762             }
763             return Float.valueOf(str);
764         }
765 
766         /**
767          * <p>Convert a <code>String</code> to a <code>Double</
                code>.</p>
768          *
```

```
769         * <p>Returns <code>null</code> if the string is
770         * <code>null</code>.</p>
771         *
772         * @param str  a <code>String</code> to convert, may be
                null
773         * @return converted <code>Double</code> (or null if
                the input is
774         *  null)
775         * @throws NumberFormatException if the value cannot be
                converted
776         */
777        public static Double createDouble(final String str) {
778            if (str == null) {
779                return null;
780            }
781            return Double.valueOf(str);
782        }
783
784        /**
785         * <p>Convert a <code>String</code> to a <code>Integer
                </code>,
786         * handling hex and octal notations.</p>
787         *
788         * <p>Returns <code>null</code> if the string is
789         * <code>null</code>.</p>
790         *
791         * @param str  a <code>String</code> to convert, may be
                null
792         * @return converted <code>Integer</code> (or null if
                the input is
793         *  null)
794         * @throws NumberFormatException if the value cannot be
                converted
795         */
796        public static Integer createInteger(final String str) {
797            if (str == null) {
798                return null;
799            }
800
801            // decode() handles 0xAABD and 0777 (hex and octal)
                    as well.
802            return Integer.decode(str);
803        }
804
805        /**
806         * <p>Convert a <code>String</code> to a <code>Long</
```

```
         code>;
807      * since 3.1 it handles hex and octal notations.</p>
808      *
809      * <p>Returns <code>null</code> if the string is
810      * <code>null</code>.</p>
811      *
812      * @param str  a <code>String</code> to convert, may be
             null
813      * @return converted <code>Long</code> (or null if the
             input is
814      *  null)
815      * @throws NumberFormatException if the value cannot be
             converted
816      */
817     public static Long createLong(final String str) {
818         if (str == null) {
819             return null;
820         }
821         return Long.decode(str);
822     }
823
824     /**
825      * <p>Convert a <code>String</code> to a <code>
             BigInteger</code>;
826      * since 3.2 it handles hex (0x or #) and octal (0)
             notations.</p>
827      *
828      * <p>Returns <code>null</code> if the string is
829      * <code>null</code>.</p>
830      *
831      * @param str  a <code>String</code> to convert, may be
             null
832      * @return converted <code>BigInteger</code> (or null
             if the input
833      *  is null)
834      * @throws NumberFormatException if the value cannot be
             converted
835      */
836     public static BigInteger createBigInteger(final String
        str) {
837         if (str == null) {
838             return null;
839         }
840         int pos = 0; // offset within string
841         int radix = 10;
842         boolean negate = false; // need to negate later?
```

```
843         if (str.startsWith("-")) {
844             negate = true;
845             pos = 1;
846         }
847         if (str.startsWith("0x", pos) || str.startsWith("0x
                ", pos)) {
848 
849             // hex
850             radix = 16;
851             pos += 2;
852         } else if (str.startsWith("#", pos)) {
853 
854             // alternative hex (allowed by Long/Integer)
855             radix = 16;
856             pos ++;
857         } else if (str.startsWith("0", pos) && str.length()
                > pos + 1) {
858 
859             // octal; so long as there are additional
                    digits
860             radix = 8;
861             pos ++;
862         } // default is to treat as decimal
863 
864         final BigInteger value = new BigInteger(str.
                substring(pos), radix);
865         return negate ? value.negate() : value;
866     }
867 
868     /**
869      * <p>Convert a <code>String</code> to a
870      * <code>BigDecimal</code>.</p>
871      *
872      * <p>Returns <code>null</code> if the string is
873      * <code>null</code>.</p>
874      *
875      * @param str  a <code>String</code> to convert, may be
                null
876      * @return converted <code>BigDecimal</code> (or null
                if the input
877      *  is null)
878      * @throws NumberFormatException if the value cannot be
                converted
879      */
880     public static BigDecimal createBigDecimal(final String
            str) {
```

```
881         if (str == null) {
882             return null;
883         }
884 
885         // handle JDK1.3.1 bug where "" throws
886         // IndexOutOfBoundsException
887         if (StringUtils.isBlank(str)) {
888             throw new NumberFormatException(
889                 "A blank string is not a valid number");
890         }
891         if (str.trim().startsWith("--")) {
892 
893             // this is protection for poorness in
894             // java.lang.BigDecimal. it accepts this as a
                   legal value,
895             // but it does not appear to be in
                   specification of class.
896             // OS X Java parses it to a wrong value.
897             throw new NumberFormatException(str + " is not
                   a valid number.");
898         }
899         return new BigDecimal(str);
900     }
901 
902 
903     // Min in array
904     //
           ----------------------------------------------------------------
905     /**
906      * <p>Returns the minimum value in an array.</p>
907      *
908      * @param array  an array, must not be null or empty
909      * @return the minimum value in the array
910      * @throws IllegalArgumentException if <code>array</
           code> is
911      *  <code>null</code>
912      * @throws IllegalArgumentException if <code>array</
           code> is empty
913      */
914     public static long min(final long[] array) {
915 
916         // Validates input
917         validateArray(array);
918 
919         // Finds and returns min
```

```
920         long min = array[0];
921         for (int i = 1; i < array.length; i++) {
922             if (array[i] < min) {
923                 min = array[i];
924             }
925         }
926 
927         return min;
928     }
929 
930     /**
931      * <p>Returns the minimum value in an array.</p>
932      *
933      * @param array  an array, must not be null or empty
934      * @return the minimum value in the array
935      * @throws IllegalArgumentException if <code>array</
         code> is
936      *  <code>null</code>
937      * @throws IllegalArgumentException if <code>array</
         code> is empty
938      */
939     public static int min(final int[] array) {
940 
941         // Validates input
942         validateArray(array);
943 
944         // Finds and returns min
945         int min = array[0];
946         for (int j = 1; j < array.length; j++) {
947             if (array[j] < min) {
948                 min = array[j];
949             }
950         }
951 
952         return min;
953     }
954 
955     /**
956      * <p>Returns the minimum value in an array.</p>
957      *
958      * @param array  an array, must not be null or empty
959      * @return the minimum value in the array
960      * @throws IllegalArgumentException if <code>array</
         code> is
961      *  <code>null</code>
962      * @throws IllegalArgumentException if <code>array</
```

```
         code> is empty
963       */
964      public static short min(final short[] array) {
965
966          // Validates input
967          validateArray(array);
968
969          // Finds and returns min
970          short min = array[0];
971          for (int i = 1; i < array.length; i++) {
972              if (array[i] < min) {
973                  min = array[i];
974              }
975          }
976
977          return min;
978      }
979
980      /**
981       * <p>Returns the minimum value in an array.</p>
982       *
983       * @param array  an array, must not be null or empty
984       * @return the minimum value in the array
985       * @throws IllegalArgumentException if <code>array</
             code> is
986       *  <code>null</code>
987       * @throws IllegalArgumentException if <code>array</
             code> is empty
988       */
989      public static byte min(final byte[] array) {
990
991          // Validates input
992          validateArray(array);
993
994          // Finds and returns min
995          byte min = array[0];
996          for (int i = 1; i < array.length; i++) {
997              if (array[i] < min) {
998                  min = array[i];
999              }
1000         }
1001
1002         return min;
1003     }
1004
1005      /**
```

```
1006        * <p>Returns the minimum value in an array.</p>
1007        *
1008        * @param array  an array, must not be null or empty
1009        * @return the minimum value in the array
1010        * @throws IllegalArgumentException if <code>array</
               code> is
1011        *  <code>null</code>
1012        * @throws IllegalArgumentException if <code>array</
               code> is empty
1013        * @see IEEE754rUtils#min(double[]) IEEE754rUtils for a
                version of
1014        *  this method that handles NaN differently
1015        */
1016       public static double min(final double[] array) {
1017
1018           // Validates input
1019           validateArray(array);
1020
1021           // Finds and returns min
1022           double min = array[0];
1023           for (int i = 1; i < array.length; i++) {
1024               if (Double.isNaN(array[i])) {
1025                   return Double.NaN;
1026               }
1027               if (array[i] < min) {
1028                   min = array[i];
1029               }
1030           }
1031
1032           return min;
1033       }
1034
1035       /**
1036        * <p>Returns the minimum value in an array.</p>
1037        *
1038        * @param array  an array, must not be null or empty
1039        * @return the minimum value in the array
1040        * @throws IllegalArgumentException if <code>array</
               code> is
1041        *  <code>null</code>
1042        * @throws IllegalArgumentException if <code>array</
               code> is empty
1043        * @see IEEE754rUtils#min(float[]) IEEE754rUtils for a
               version of
1044        *  this method that handles NaN differently
1045        */
```

```
1046        public static float min(final float[] array) {
1047
1048            // Validates input
1049            validateArray(array);
1050
1051            // Finds and returns min
1052            float min = array[0];
1053            for (int i = 1; i < array.length; i++) {
1054                if (Float.isNaN(array[i])) {
1055                    return Float.NaN;
1056                }
1057                if (array[i] < min) {
1058                    min = array[i];
1059                }
1060            }
1061
1062            return min;
1063        }
1064
1065
1066        // Max in array
1067        //
               ----------------------------------------------------------------

1068        /**
1069         * <p>Returns the maximum value in an array.</p>
1070         *
1071         * @param array  an array, must not be null or empty
1072         * @return the minimum value in the array
1073         * @throws IllegalArgumentException if <code>array</
                code> is
1074         *  <code>null</code>
1075         * @throws IllegalArgumentException if <code>array</
                code> is empty
1076         */
1077        public static long max(final long[] array) {
1078
1079            // Validates input
1080            validateArray(array);
1081
1082            // Finds and returns max
1083            long max = array[0];
1084            for (int j = 1; j < array.length; j++) {
1085                if (array[j] > max) {
1086                    max = array[j];
1087                }
```

```
1088            }
1089
1090            return max;
1091        }
1092
1093        /**
1094         * <p>Returns the maximum value in an array.</p>
1095         *
1096         * @param array  an array, must not be null or empty
1097         * @return the minimum value in the array
1098         * @throws IllegalArgumentException if <code>array</
                code> is
1099         *  <code>null</code>
1100         * @throws IllegalArgumentException if <code>array</
                code> is empty
1101         */
1102        public static int max(final int[] array) {
1103
1104            // Validates input
1105            validateArray(array);
1106
1107            // Finds and returns max
1108            int max = array[0];
1109            for (int j = 1; j < array.length; j++) {
1110                if (array[j] > max) {
1111                    max = array[j];
1112                }
1113            }
1114
1115            return max;
1116        }
1117
1118        /**
1119         * <p>Returns the maximum value in an array.</p>
1120         *
1121         * @param array  an array, must not be null or empty
1122         * @return the minimum value in the array
1123         * @throws IllegalArgumentException if <code>array</
                code> is
1124         *  <code>null</code>
1125         * @throws IllegalArgumentException if <code>array</
                code> is empty
1126         */
1127        public static short max(final short[] array) {
1128
1129            // Validates input
```

```
1130            validateArray(array);
1131
1132            // Finds and returns max
1133            short max = array[0];
1134            for (int i = 1; i < array.length; i++) {
1135                if (array[i] > max) {
1136                    max = array[i];
1137                }
1138            }
1139
1140            return max;
1141        }
1142
1143        /**
1144         * <p>Returns the maximum value in an array.</p>
1145         *
1146         * @param array  an array, must not be null or empty
1147         * @return the minimum value in the array
1148         * @throws IllegalArgumentException if <code>array</
                code> is
1149         *  <code>null</code>
1150         * @throws IllegalArgumentException if <code>array</
                code> is empty
1151         */
1152        public static byte max(final byte[] array) {
1153
1154            // Validates input
1155            validateArray(array);
1156
1157            // Finds and returns max
1158            byte max = array[0];
1159            for (int i = 1; i < array.length; i++) {
1160                if (array[i] > max) {
1161                    max = array[i];
1162                }
1163            }
1164
1165            return max;
1166        }
1167
1168        /**
1169         * <p>Returns the maximum value in an array.</p>
1170         *
1171         * @param array  an array, must not be null or empty
1172         * @return the minimum value in the array
1173         * @throws IllegalArgumentException if <code>array</
```

```
         code> is
1174      *  <code>null</code>
1175      * @throws IllegalArgumentException if <code>array</
             code> is empty
1176      * @see IEEE754rUtils#max(double[]) IEEE754rUtils for a
              version of
1177      *  this method that handles NaN differently
1178      */
1179     public static double max(final double[] array) {
1180 
1181         // Validates input
1182         validateArray(array);
1183 
1184         // Finds and returns max
1185         double max = array[0];
1186         for (int j = 1; j < array.length; j++) {
1187             if (Double.isNaN(array[j])) {
1188                 return Double.NaN;
1189             }
1190             if (array[j] > max) {
1191                 max = array[j];
1192             }
1193         }
1194 
1195         return max;
1196     }
1197 
1198     /**
1199      * <p>Returns the maximum value in an array.</p>
1200      *
1201      * @param array  an array, must not be null or empty
1202      * @return the minimum value in the array
1203      * @throws IllegalArgumentException if <code>array</
             code> is
1204      *  <code>null</code>
1205      * @throws IllegalArgumentException if <code>array</
             code> is empty
1206      * @see IEEE754rUtils#max(float[]) IEEE754rUtils for a
             version of
1207      *  this method that handles NaN differently
1208      */
1209     public static float max(final float[] array) {
1210 
1211         // Validates input
1212         validateArray(array);
1213 
```

```
1214            // Finds and returns max
1215            float max = array[0];
1216            for (int j = 1; j < array.length; j++) {
1217                if (Float.isNaN(array[j])) {
1218                    return Float.NaN;
1219                }
1220                if (array[j] > max) {
1221                    max = array[j];
1222                }
1223            }
1224
1225            return max;
1226        }
1227
1228        /**
1229         * Checks if the specified array is neither null nor
                empty.
1230         *
1231         * @param array  the array to check
1232         * @throws IllegalArgumentException if {@code array} is
                either
1233         *  {@code null} or empty
1234         */
1235        private static void validateArray(final Object array) {
1236            if (array == null) {
1237                throw new IllegalArgumentException("The Array
                        must not be null");
1238            } else if (Array.getLength(array) == 0) {
1239                throw new IllegalArgumentException("Array
                        cannot be empty.");
1240            }
1241        }
1242
1243
1244        // 3 param min
1245        //
                -----------------------------------------------------------------

1246        /**
1247         * <p>Gets the minimum of three <code>long</code>
                values.</p>
1248         *
1249         * @param a  value 1
1250         * @param b  value 2
1251         * @param c  value 3
1252         * @return  the smallest of the values
```

```
1253        */
1254       public static long min(long a, final long b, final long
               c) {
1255           if (b < a) {
1256               a = b;
1257           }
1258           if (c < a) {
1259               a = c;
1260           }
1261           return a;
1262       }
1263
1264       /**
1265        * <p>Gets the minimum of three <code>int</code> values
               .</p>
1266        *
1267        * @param a  value 1
1268        * @param b  value 2
1269        * @param c  value 3
1270        * @return  the smallest of the values
1271        */
1272       public static int min(int a, final int b, final int c)
              {
1273           if (b < a) {
1274               a = b;
1275           }
1276           if (c < a) {
1277               a = c;
1278           }
1279           return a;
1280       }
1281
1282       /**
1283        * <p>Gets the minimum of three <code>short</code>
               values.</p>
1284        *
1285        * @param a  value 1
1286        * @param b  value 2
1287        * @param c  value 3
1288        * @return  the smallest of the values
1289        */
1290       public static short min(short a, final short b, final
              short c) {
1291           if (b < a) {
1292               a = b;
1293           }
```

```
1294            if (c < a) {
1295                a = c;
1296            }
1297            return a;
1298        }
1299
1300        /**
1301         * <p>Gets the minimum of three <code>byte</code>
                values.</p>
1302         *
1303         * @param a  value 1
1304         * @param b  value 2
1305         * @param c  value 3
1306         * @return  the smallest of the values
1307         */
1308        public static byte min(byte a, final byte b, final byte
                c) {
1309            if (b < a) {
1310                a = b;
1311            }
1312            if (c < a) {
1313                a = c;
1314            }
1315            return a;
1316        }
1317
1318        /**
1319         * <p>Gets the minimum of three <code>double</code>
                values.</p>
1320         *
1321         * <p>If any value is <code>NaN</code>, <code>NaN</code
                > is
1322         * returned. Infinity is handled.</p>
1323         *
1324         * @param a  value 1
1325         * @param b  value 2
1326         * @param c  value 3
1327         * @return  the smallest of the values
1328         * @see IEEE754rUtils#min(double, double, double) for a
                    version of
1329         *  this method that handles NaN differently
1330         */
1331        public static double min(final double a, final double b
               , final double c) {
1332            return Math.min(Math.min(a, b), c);
1333        }
```

```
1334
1335        /**
1336         * <p>Gets the minimum of three <code>float</code>
                values.</p>
1337         *
1338         * <p>If any value is <code>NaN</code>, <code>NaN</code
                > is
1339         * returned. Infinity is handled.</p>
1340         *
1341         * @param a  value 1
1342         * @param b  value 2
1343         * @param c  value 3
1344         * @return  the smallest of the values
1345         * @see IEEE754rUtils#min(float, float, float) for a
                version of
1346         *  this method that handles NaN differently
1347         */
1348        public static float min(final float a, final float b,
               final float c) {
1349            return Math.min(Math.min(a, b), c);
1350        }
1351
1352
1353        // 3 param max
1354        //
               -----------------------------------------------------------------

1355        /**
1356         * <p>Gets the maximum of three <code>long</code>
                values.</p>
1357         *
1358         * @param a  value 1
1359         * @param b  value 2
1360         * @param c  value 3
1361         * @return  the largest of the values
1362         */
1363        public static long max(long a, final long b, final long
                c) {
1364            if (b > a) {
1365                a = b;
1366            }
1367            if (c > a) {
1368                a = c;
1369            }
1370            return a;
1371        }
```

```
1372
1373        /**
1374         * <p>Gets the maximum of three <code>int</code> values
                .</p>
1375         *
1376         * @param a  value 1
1377         * @param b  value 2
1378         * @param c  value 3
1379         * @return  the largest of the values
1380         */
1381        public static int max(int a, final int b, final int c)
               {
1382            if (b > a) {
1383                a = b;
1384            }
1385            if (c > a) {
1386                a = c;
1387            }
1388            return a;
1389        }
1390
1391        /**
1392         * <p>Gets the maximum of three <code>short</code>
                values.</p>
1393         *
1394         * @param a  value 1
1395         * @param b  value 2
1396         * @param c  value 3
1397         * @return  the largest of the values
1398         */
1399        public static short max(short a, final short b, final
               short c) {
1400            if (b > a) {
1401                a = b;
1402            }
1403            if (c > a) {
1404                a = c;
1405            }
1406            return a;
1407        }
1408
1409        /**
1410         * <p>Gets the maximum of three <code>byte</code>
                values.</p>
1411         *
1412         * @param a  value 1
```

```
1413        * @param b  value 2
1414        * @param c  value 3
1415        * @return  the largest of the values
1416        */
1417       public static byte max(byte a, final byte b, final byte
               c) {
1418           if (b > a) {
1419               a = b;
1420           }
1421           if (c > a) {
1422               a = c;
1423           }
1424           return a;
1425       }
1426
1427       /**
1428        * <p>Gets the maximum of three <code>double</code>
               values.</p>
1429        *
1430        * <p>If any value is <code>NaN</code>, <code>NaN</code
               > is
1431        * returned. Infinity is handled.</p>
1432        *
1433        * @param a  value 1
1434        * @param b  value 2
1435        * @param c  value 3
1436        * @return  the largest of the values
1437        * @see IEEE754rUtils#max(double, double, double) for a
                version of
1438        *  this method that handles NaN differently
1439        */
1440       public static double max(final double a, final double b
               , final double c) {
1441           return Math.max(Math.max(a, b), c);
1442       }
1443
1444       /**
1445        * <p>Gets the maximum of three <code>float</code>
               values.</p>
1446        *
1447        * <p>If any value is <code>NaN</code>, <code>NaN</code
               > is
1448        * returned. Infinity is handled.</p>
1449        *
1450        * @param a  value 1
1451        * @param b  value 2
```

```
1452        * @param c  value 3
1453        * @return  the largest of the values
1454        * @see IEEE754rUtils#max(float, float, float) for a
               version of
1455        *  this method that handles NaN differently
1456        */
1457       public static float max(final float a, final float b,
               final float c) {
1458           return Math.max(Math.max(a, b), c);
1459       }
1460
1461
1462       //
           -----------------------------------------------------------------------
1463       /**
1464        * <p>Checks whether the <code>String</code> contains
               only
1465        * digit characters.</p>
1466        *
1467        * <p><code>Null</code> and empty String will return
1468        * <code>false</code>.</p>
1469        *
1470        * @param str  the <code>String</code> to check
1471        * @return <code>true</code> if str contains only
               Unicode numeric
1472        */
1473       public static boolean isDigits(final String str) {
1474           if (StringUtils.isEmpty(str)) {
1475               return false;
1476           }
1477           for (int i = 0; i < str.length(); i++) {
1478               if (!Character.isDigit(str.charAt(i))) {
1479                   return false;
1480               }
1481           }
1482           return true;
1483       }
1484
1485       /**
1486        * <p>Checks whether the String a valid Java number.</p
               >
1487        *
1488        * <p>Valid numbers include hexadecimal marked with the
1489        * <code>0x</code> qualifier, scientific notation and
               numbers
```

```
1490        * marked with a type qualifier (e.g. 123L).</p>
1491        *
1492        * <p><code>Null</code> and empty String will return
1493        * <code>false</code>.</p>
1494        *
1495        * @param str  the <code>String</code> to check
1496        * @return <code>true</code> if the string is a
               correctly
1497        *  formatted number
1498        */
1499       public static boolean isNumber(final String str) {
1500           if (StringUtils.isEmpty(str)) {
1501               return false;
1502           }
1503           final char[] chars = str.toCharArray();
1504           int sz = chars.length;
1505           boolean hasExp = false;
1506           boolean hasDecPoint = false;
1507           boolean allowSigns = false;
1508           boolean foundDigit = false;
1509
1510           // deal with any possible sign up front
1511           final int start = (chars[0] == '-') ? 1 : 0;
1512           if (sz > start + 1 && chars[start] == '0' && chars[
                  start + 1] == 'x') {
1513               int i = start + 2;
1514               if (i == sz) {
1515                   return false; // str == "0x"
1516               }
1517
1518               // checking hex (it can't be anything else)
1519               for (; i < chars.length; i++) {
1520                   if ((chars[i] < '0' || chars[i] > '9')
1521                           && (chars[i] < 'a' || chars[i] > 'f')
1522                           && (chars[i] < 'A' || chars[i] > 'F'))
                              {
1523                       return false;
1524                   }
1525               }
1526               return true;
1527           }
1528           sz--;
1529
1530           // don't want to loop to the last char, check it
1531           // afterwords for type qualifiers
1532
```

```
1533        int i = start;
1534
1535        // loop to the next to last char or to the last
               char if we
1536        // need another digit to make a valid number
1537        // (e.g. chars[0..5] = "1234E")
1538        while (i < sz || (i < sz + 1 && allowSigns && !
               foundDigit)) {
1539            if (chars[i] >= '0' && chars[i] <= '9') {
1540                foundDigit = true;
1541                allowSigns = false;
1542            } else if (chars[i] == '.') {
1543                if (hasDecPoint || hasExp) {
1544
1545                    // two decimal points or dec in
                           exponent
1546                    return false;
1547                }
1548                hasDecPoint = true;
1549            } else if (chars[i] == 'e' || chars[i] == 'E')
                   {
1550
1551                // we've already taken care of hex.
1552                if (hasExp) {
1553
1554                    // two E's
1555                    return false;
1556                }
1557                if (!foundDigit) {
1558                    return false;
1559                }
1560                hasExp = true;
1561                allowSigns = true;
1562            } else if (chars[i] == '+' || chars[i] == '-')
                   {
1563                if (!allowSigns) {
1564                    return false;
1565                }
1566                allowSigns = false;
1567                foundDigit = false; // we need a digit
                       after the E
1568            } else {
1569                return false;
1570            }
1571            i++;
1572        }
```

```
1573            if (i < chars.length) {
1574                if (chars[i] >= '0' && chars[i] <= '9') {
1575
1576                    // no type qualifier, OK
1577                    return true;
1578                }
1579                if (chars[i] == 'e' || chars[i] == 'E') {
1580
1581                    // can't have an E at the last byte
1582                    return false;
1583                }
1584                if (chars[i] == '.') {
1585                    if (hasDecPoint || hasExp) {
1586
1587                        // two decimal points or dec in
                               exponent
1588                        return false;
1589                    }
1590
1591                    // single trailing decimal point after non-
                           exponent is
1592                    // ok
1593                    return foundDigit;
1594                }
1595                if (!allowSigns
1596                    && (chars[i] == 'd'
1597                        || chars[i] == 'D'
1598                        || chars[i] == 'f'
1599                        || chars[i] == 'F')) {
1600                    return foundDigit;
1601                }
1602                if (chars[i] == 'l'
1603                    || chars[i] == 'L') {
1604
1605                    // not allowing L with an exponent or
                           decimal point
1606                    return foundDigit && !hasExp && !
                           hasDecPoint;
1607                }
1608
1609                // last character is illegal
1610                return false;
1611            }
1612
1613            // allowSigns is true iff the val ends in 'E'
1614            // found digit it to make sure weird stuff like '.'
```

```
            and '1E-'
        // doesn't pass
        return !allowSigns && foundDigit;
    }
}
```

E.3.3 Defects-Sources/Lang-1/NumberUtils-bugged-B.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under
       one or more
3   * contributor license agreements.  See the NOTICE file
       distributed with
4   * this work for additional information regarding copyright
        ownership.
5   * The ASF licenses this file to You under the Apache
       License, Version 2.0
6   * (the "License"); you may not use this file except in
       compliance with
7   * the License.  You may obtain a copy of the License at
8   *
9   *      http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in
       writing, software
12  * distributed under the License is distributed on an "AS
       IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
       express or implied.
14  * See the License for the specific language governing
       permissions and
15  * limitations under the License.
16  */
17 package org.apache.commons.lang3.math;
18
19 import java.lang.reflect.Array;
20 import java.math.BigDecimal;
21 import java.math.BigInteger;
22
23 import org.apache.commons.lang3.StringUtils;
24
25 /**
26  * <p>Provides extra functionality for Java Number classes
       .</p>
27  *
28  * @since 2.0
29  * @version $Id$
30  */
31 public class NumberUtils
32 {
33
34     /** Reusable Long constant for zero. */
```

```
35      public static final Long LONG_ZERO = Long.valueOf(0L);
36      /** Reusable Long constant for one. */
37      public static final Long LONG_ONE = Long.valueOf(1L);
38      /** Reusable Long constant for minus one. */
39      public static final Long LONG_MINUS_ONE = Long.valueOf
            (-1L);
40      /** Reusable Integer constant for zero. */
41      public static final Integer INTEGER_ZERO = Integer.
            valueOf(0);
42      /** Reusable Integer constant for one. */
43      public static final Integer INTEGER_ONE = Integer.
            valueOf(1);
44      /** Reusable Integer constant for minus one. */
45      public static final Integer INTEGER_MINUS_ONE = Integer
            .valueOf(-1);
46      /** Reusable Short constant for zero. */
47      public static final Short SHORT_ZERO = Short.valueOf((
            short) 0);
48      /** Reusable Short constant for one. */
49      public static final Short SHORT_ONE = Short.valueOf((
            short) 1);
50      /** Reusable Short constant for minus one. */
51      public static final Short SHORT_MINUS_ONE = Short.
            valueOf((short) -1);
52      /** Reusable Byte constant for zero. */
53      public static final Byte BYTE_ZERO = Byte.valueOf((byte
            ) 0);
54      /** Reusable Byte constant for one. */
55      public static final Byte BYTE_ONE = Byte.valueOf((byte)
             1);
56      /** Reusable Byte constant for minus one. */
57      public static final Byte BYTE_MINUS_ONE = Byte.valueOf
            ((byte) -1);
58      /** Reusable Double constant for zero. */
59      public static final Double DOUBLE_ZERO = Double.valueOf
            (0.0d);
60      /** Reusable Double constant for one. */
61      public static final Double DOUBLE_ONE = Double.valueOf
            (1.0d);
62      /** Reusable Double constant for minus one. */
63      public static final Double DOUBLE_MINUS_ONE = Double.
            valueOf(-1.0d);
64      /** Reusable Float constant for zero. */
65      public static final Float FLOAT_ZERO = Float.valueOf
            (0.0f);
66      /** Reusable Float constant for one. */
```

```
67      public static final Float FLOAT_ONE = Float.valueOf(1.0
           f);
68      /** Reusable Float constant for minus one. */
69      public static final Float FLOAT_MINUS_ONE = Float.
           valueOf(-1.0f);
70
71      /**
72       * <p><code>NumberUtils</code> instances should NOT be
            constructed in standard programming.
73       * Instead, the class should be used as <code>
            NumberUtils.toInt("6");</code>.</p>
74       *
75       * <p>This constructor is public to permit tools that
            require a JavaBean instance
76       * to operate.</p>
77       */
78      public NumberUtils()
79      {
80          super();
81      }
82
83      //
           -----------------------------------------------------------------

84      /**
85       * <p>Convert a <code>String</code> to an <code>int</
            code>, returning
86       * <code>zero</code> if the conversion fails.</p>
87       *
88       * <p>If the string is <code>null</code>, <code>zero</
            code> is returned.</p>
89       *
90       * <pre>
91       *   NumberUtils.toInt(null) = 0
92       *   NumberUtils.toInt("")   = 0
93       *   NumberUtils.toInt("1")  = 1
94       * </pre>
95       *
96       * @param str  the string to convert, may be null
97       * @return the int represented by the string, or <code>
            zero</code> if
98       *  conversion fails
99       * @since 2.1
100      */
101     public static int toInt(final String str)
102     {
```

```
103         return toInt(str, 0);
104     }
105 
106     /**
107      * <p>Convert a <code>String</code> to an <code>int</
             code>, returning a
108      * default value if the conversion fails.</p>
109      *
110      * <p>If the string is <code>null</code>, the default
             value is returned.</p>
111      *
112      * <pre>
113      *   NumberUtils.toInt(null, 1) = 1
114      *   NumberUtils.toInt("", 1)   = 1
115      *   NumberUtils.toInt("1", 0)  = 1
116      * </pre>
117      *
118      * @param str  the string to convert, may be null
119      * @param defaultValue  the default value
120      * @return the int represented by the string, or the
             default if conversion fails
121      * @since 2.1
122      */
123     public static int toInt(final String str, final int
           defaultValue)
124     {
125         if(str == null)
126         {
127             return defaultValue;
128         }
129         try
130         {
131             return Integer.parseInt(str);
132         }
133         catch (final NumberFormatException nfe)
134         {
135             return defaultValue;
136         }
137     }
138 
139     /**
140      * <p>Convert a <code>String</code> to a <code>long</
             code>, returning
141      * <code>zero</code> if the conversion fails.</p>
142      *
143      * <p>If the string is <code>null</code>, <code>zero</
```

```
         code> is returned.</p>
144       *
145       * <pre>
146       *   NumberUtils.toLong(null) = 0L
147       *   NumberUtils.toLong("")   = 0L
148       *   NumberUtils.toLong("1")  = 1L
149       * </pre>
150       *
151       * @param str  the string to convert, may be null
152       * @return the long represented by the string, or <code
             >0</code> if
153       *  conversion fails
154       * @since 2.1
155       */
156      public static long toLong(final String str)
157      {
158          return toLong(str, 0L);
159      }
160
161      /**
162       * <p>Convert a <code>String</code> to a <code>long</
             code>, returning a
163       * default value if the conversion fails.</p>
164       *
165       * <p>If the string is <code>null</code>, the default
             value is returned.</p>
166       *
167       * <pre>
168       *   NumberUtils.toLong(null, 1L) = 1L
169       *   NumberUtils.toLong("", 1L)   = 1L
170       *   NumberUtils.toLong("1", 0L)  = 1L
171       * </pre>
172       *
173       * @param str  the string to convert, may be null
174       * @param defaultValue  the default value
175       * @return the long represented by the string, or the
             default if conversion fails
176       * @since 2.1
177       */
178      public static long toLong(final String str, final long
           defaultValue)
179      {
180          if (str == null)
181          {
182              return defaultValue;
183          }
```

```
184         try
185         {
186             return Long.parseLong(str);
187         }
188         catch (final NumberFormatException nfe)
189         {
190             return defaultValue;
191         }
192     }
193 
194     /**
195      * <p>Convert a <code>String</code> to a <code>float</
             code>, returning
196      * <code>0.0f</code> if the conversion fails.</p>
197      *
198      * <p>If the string <code>str</code> is <code>null</
             code>,
199      * <code>0.0f</code> is returned.</p>
200      *
201      * <pre>
202      *   NumberUtils.toFloat(null)   = 0.0f
203      *   NumberUtils.toFloat("")     = 0.0f
204      *   NumberUtils.toFloat("1.5")  = 1.5f
205      * </pre>
206      *
207      * @param str the string to convert, may be <code>null
             </code>
208      * @return the float represented by the string, or <
             code>0.0f</code>
209      *  if conversion fails
210      * @since 2.1
211      */
212     public static float toFloat(final String str)
213     {
214         return toFloat(str, 0.0f);
215     }
216 
217     /**
218      * <p>Convert a <code>String</code> to a <code>float</
             code>, returning a
219      * default value if the conversion fails.</p>
220      *
221      * <p>If the string <code>str</code> is <code>null</
             code>, the default
222      * value is returned.</p>
223      *
```

```
224         * <pre>
225         *   NumberUtils.toFloat(null, 1.1f)   = 1.0f
226         *   NumberUtils.toFloat("", 1.1f)     = 1.1f
227         *   NumberUtils.toFloat("1.5", 0.0f)  = 1.5f
228         * </pre>
229         *
230         * @param str the string to convert, may be <code>null
                </code>
231         * @param defaultValue the default value
232         * @return the float represented by the string, or
                defaultValue
233         *  if conversion fails
234         * @since 2.1
235         */
236        public static float toFloat(final String str, final
               float defaultValue)
237        {
238            if (str == null)
239            {
240                return defaultValue;
241            }
242            try
243            {
244                return Float.parseFloat(str);
245            }
246            catch (final NumberFormatException nfe)
247            {
248                return defaultValue;
249            }
250        }
251
252        /**
253         * <p>Convert a <code>String</code> to a <code>double</
                code>, returning
254         * <code>0.0d</code> if the conversion fails.</p>
255         *
256         * <p>If the string <code>str</code> is <code>null</
                code>,
257         * <code>0.0d</code> is returned.</p>
258         *
259         * <pre>
260         *   NumberUtils.toDouble(null)   = 0.0d
261         *   NumberUtils.toDouble("")     = 0.0d
262         *   NumberUtils.toDouble("1.5")  = 1.5d
263         * </pre>
264         *
```

```
265         * @param str the string to convert, may be <code>null
                </code>
266         * @return the double represented by the string, or <
                code>0.0d</code>
267         *  if conversion fails
268         * @since 2.1
269         */
270        public static double toDouble(final String str)
271        {
272            return toDouble(str, 0.0d);
273        }
274
275        /**
276         * <p>Convert a <code>String</code> to a <code>double</
                code>, returning a
277         * default value if the conversion fails.</p>
278         *
279         * <p>If the string <code>str</code> is <code>null</
                code>, the default
280         * value is returned.</p>
281         *
282         * <pre>
283         *   NumberUtils.toDouble(null, 1.1d)   = 1.1d
284         *   NumberUtils.toDouble("", 1.1d)     = 1.1d
285         *   NumberUtils.toDouble("1.5", 0.0d)  = 1.5d
286         * </pre>
287         *
288         * @param str the string to convert, may be <code>null
                </code>
289         * @param defaultValue the default value
290         * @return the double represented by the string, or
                defaultValue
291         *  if conversion fails
292         * @since 2.1
293         */
294        public static double toDouble(final String str, final
               double defaultValue)
295        {
296            if (str == null)
297            {
298                return defaultValue;
299            }
300            try
301            {
302                return Double.parseDouble(str);
303            }
```

```
304         catch (final NumberFormatException nfe)
305         {
306             return defaultValue;
307         }
308     }
309 
310     //-----------------------------------------------------------------------
311     /**
312      * <p>Convert a <code>String</code> to a <code>byte</code>, returning
313      * <code>zero</code> if the conversion fails.</p>
314      *
315      * <p>If the string is <code>null</code>, <code>zero</code> is returned.</p>
316      *
317      * <pre>
318      *   NumberUtils.toByte(null) = 0
319      *   NumberUtils.toByte("")   = 0
320      *   NumberUtils.toByte("1")  = 1
321      * </pre>
322      *
323      * @param str  the string to convert, may be null
324      * @return the byte represented by the string, or <code>zero</code> if
325      *  conversion fails
326      * @since 2.5
327      */
328     public static byte toByte(final String str)
329     {
330         return toByte(str, (byte) 0);
331     }
332 
333     /**
334      * <p>Convert a <code>String</code> to a <code>byte</code>, returning a
335      * default value if the conversion fails.</p>
336      *
337      * <p>If the string is <code>null</code>, the default value is returned.</p>
338      *
339      * <pre>
340      *   NumberUtils.toByte(null, 1) = 1
341      *   NumberUtils.toByte("", 1)   = 1
342      *   NumberUtils.toByte("1", 0)  = 1
```

```
343         * </pre>
344         *
345         * @param str  the string to convert, may be null
346         * @param defaultValue  the default value
347         * @return the byte represented by the string, or the
                default if conversion fails
348         * @since 2.5
349         */
350        public static byte toByte(final String str, final byte
               defaultValue)
351        {
352            if(str == null)
353            {
354                return defaultValue;
355            }
356            try
357            {
358                return Byte.parseByte(str);
359            }
360            catch (final NumberFormatException nfe)
361            {
362                return defaultValue;
363            }
364        }
365
366        /**
367         * <p>Convert a <code>String</code> to a <code>short</
                code>, returning
368         * <code>zero</code> if the conversion fails.</p>
369         *
370         * <p>If the string is <code>null</code>, <code>zero</
                code> is returned.</p>
371         *
372         * <pre>
373         *   NumberUtils.toShort(null) = 0
374         *   NumberUtils.toShort("")   = 0
375         *   NumberUtils.toShort("1")  = 1
376         * </pre>
377         *
378         * @param str  the string to convert, may be null
379         * @return the short represented by the string, or <
                code>zero</code> if
380         *  conversion fails
381         * @since 2.5
382         */
383        public static short toShort(final String str)
```

```
384     {
385         return toShort(str, (short) 0);
386     }
387 
388     /**
389      * <p>Convert a <code>String</code> to an <code>short</
            code>, returning a
390      * default value if the conversion fails.</p>
391      *
392      * <p>If the string is <code>null</code>, the default
            value is returned.</p>
393      *
394      * <pre>
395      *   NumberUtils.toShort(null, 1) = 1
396      *   NumberUtils.toShort("", 1)   = 1
397      *   NumberUtils.toShort("1", 0)  = 1
398      * </pre>
399      *
400      * @param str  the string to convert, may be null
401      * @param defaultValue  the default value
402      * @return the short represented by the string, or the
            default if conversion fails
403      * @since 2.5
404      */
405     public static short toShort(final String str, final
           short defaultValue)
406     {
407         if(str == null)
408         {
409             return defaultValue;
410         }
411         try
412         {
413             return Short.parseShort(str);
414         }
415         catch (final NumberFormatException nfe)
416         {
417             return defaultValue;
418         }
419     }
420 
421     //
           -----------------------------------------------------------------
422     // must handle Long, Float, Integer, Float, Short,
423     //                  BigDecimal, BigInteger and Byte
```

```
424     // useful methods:
425     // Byte.decode(String)
426     // Byte.valueOf(String,int radix)
427     // Byte.valueOf(String)
428     // Double.valueOf(String)
429     // Float.valueOf(String)
430     // Float.valueOf(String)
431     // Integer.valueOf(String,int radix)
432     // Integer.valueOf(String)
433     // Integer.decode(String)
434     // Integer.getInteger(String)
435     // Integer.getInteger(String,int val)
436     // Integer.getInteger(String,Integer val)
437     // Integer.valueOf(String)
438     // Double.valueOf(String)
439     // new Byte(String)
440     // Long.valueOf(String)
441     // Long.getLong(String)
442     // Long.getLong(String,int)
443     // Long.getLong(String,Integer)
444     // Long.valueOf(String,int)
445     // Long.valueOf(String)
446     // Short.valueOf(String)
447     // Short.decode(String)
448     // Short.valueOf(String,int)
449     // Short.valueOf(String)
450     // new BigDecimal(String)
451     // new BigInteger(String)
452     // new BigInteger(String,int radix)
453     // Possible inputs:
454     // 45 45.5 45E7 4.5E7 Hex Oct Binary xxxF xxxD xxxf
           xxxd
455     // plus minus everything. Prolly more. A lot are not
           separable.
456
457     /**
458      * <p>Turns a string value into a java.lang.Number.</p>
459      *
460      * <p>If the string starts with {@code 0x} or {@code -0
           x} (lower or upper case) or {@code #} or
461      * {@code -#}, it will be interpreted as a hexadecimal
           Integer - or Long, if the number of
462      * digits after the prefix is more than 8 - or
           BigInteger if there are more than 16 digits.
463      * </p>
464      * <p>Then, the value is examined for a type qualifier
```

```
         on the end, i.e. one of
465      * <code>'f','F','d','D','l','L'</code>.  If it is
             found, it starts
466      * trying to create successively larger types from the
             type specified
467      * until one is found that can represent the value.</p>
468      *
469      * <p>If a type specifier is not found, it will check
             for a decimal point
470      * and then try successively larger types from <code>
             Integer</code> to
471      * <code>BigInteger</code> and from <code>Float</code>
             to
472      * <code>BigDecimal</code>.</p>
473      *
474      * <p>
475      * Integral values with a leading {@code 0} will be
             interpreted as octal; the returned number
476      * will be Integer, Long or BigDecimal as appropriate.
477      * </p>
478      *
479      * <p>Returns <code>null</code> if the string is <code>
             null</code>.</p>
480      *
481      * <p>This method does not trim the input string, i.e.,
               strings with leading
482      * or trailing spaces will generate
             NumberFormatExceptions.</p>
483      *
484      * @param str  String containing a number, may be null
485      * @return Number created from the string (or null if
             the input is null)
486      * @throws NumberFormatException if the value cannot be
               converted
487      */
488     public static Number createNumber(final String str)
           throws NumberFormatException
489     {
490         if (str == null)
491         {
492             return null;
493         }
494         if (StringUtils.isBlank(str))
495         {
496             throw new NumberFormatException("A blank string
                    is not a valid number");
```

```
497         }
498         // Need to deal with all possible hex prefixes here
499         final String[] hex_prefixes = {"0x", "0X", "-0x", "
               -0X", "#", "-#"};
500         int pfxLen = 0;
501         for(final String pfx : hex_prefixes)
502         {
503             if (str.startsWith(pfx))
504             {
505                 pfxLen += pfx.length();
506                 break;
507             }
508         }
509         if (pfxLen > 0)
510         {
511             // we have a hex number
512             final int hexDigits = str.length() - pfxLen;
513             if (hexDigits > 16)
514             {
515                 // too many for Long
516                 return createBigInteger(str);
517             }
518             if (hexDigits > 8)
519             {
520                 // too many for an int
521                 return createLong(str);
522             }
523             return createInteger(str);
524         }
525         final char lastChar = str.charAt(str.length() - 1);
526         String mant;
527         String dec;
528         String exp;
529         final int decPos = str.indexOf('.');
530         final int expPos = str.indexOf('e') + str.indexOf('
               E') + 1; // assumes both not present
531         // if both e and E are present, this is caught by
               the checks on expPos (which prevent IOOBE)
532         // and the parsing which will detect if e or E
               appear in a number due to using the wrong
533         // offset
534 
535         int numDecimals = 0; // Check required precision (
               LANG-693)
536         if (decPos > -1)
537         {
```

```
538             // there is a decimal point
539             if (expPos > -1)
540             {
541                 // there is an exponent
542                 if (expPos < decPos || expPos > str.length
                        ())
543                 {
544                     //prevents double exponent causing
                            IOOBE
545                     throw new NumberFormatException(str + "
                            is not a valid number.");
546                 }
547                 dec = str.substring(decPos + 1, expPos);
548             }
549             else
550             {
551                 dec = str.substring(decPos + 1);
552             }
553             mant = str.substring(0, decPos);
554             numDecimals = dec.length(); // gets number of
                    digits past the decimal to ensure no loss
555                                         // of precision for
                                                floating point
                                               numbers.
556         }
557         else
558         {
559             if (expPos > -1)
560             {
561                 if (expPos > str.length())
562                 {
563                     // prevents double exponent causing
                            IOOBE
564                     throw new NumberFormatException(str + "
                            is not a valid number.");
565                 }
566                 mant = str.substring(0, expPos);
567             }
568             else
569             {
570                 mant = str;
571             }
572             dec = null;
573         }
574         if (!Character.isDigit(lastChar) && lastChar != '.'
                )
```

```
575         {
576             if (expPos > -1 && expPos < str.length() - 1)
577             {
578                 exp = str.substring(expPos + 1, str.length
                        () - 1);
579             }
580             else
581             {
582                 exp = null;
583             }
584             //Requesting a specific type..
585             final String numeric = str.substring(0, str.
                    length() - 1);
586             final boolean allZeros = isAllZeros(mant) &&
                    isAllZeros(exp);
587             switch (lastChar)
588             {
589                 case 'l' :
590                 case 'L' :
591                     if (dec == null
592                         && exp == null
593                         && (numeric.charAt(0) == '-' &&
                                isDigits(numeric.substring(1))
594                         || isDigits(numeric)))
595                     {
596                         try
597                         {
598                             return createLong(numeric);
599                         }
600                         catch (final NumberFormatException
                                nfe)
601                         {
602                             // NOPMD
603                             // Too big for a long
604                         }
605                         return createBigInteger(numeric);
606 
607                     }
608                     throw new NumberFormatException(str + "
                            is not a valid number.");
609                 case 'f' :
610                 case 'F' :
611                     try
612                     {
613                         final Float f = NumberUtils.
                                createFloat(numeric);
```

```
614                         if (!(f.isInfinite() || (f.
                               floatValue() == 0.0F && !
                               allZeros)))
615                         {
616                             //If it's too big for a float
                                   or the float value = 0 and
                                   the string
617                             //has non-zeros in it, then
                                   float does not have the
                                   precision we want
618                             return f;
619                         }
620
621                     }
622                     catch (final NumberFormatException nfe)
623                     {
624                         // NOPMD
625                         // ignore the bad number
626                     }
627                     //$FALL-THROUGH$
628                 case 'd' :
629                 case 'D' :
630                     try
631                     {
632                         final Double d = NumberUtils.
                               createDouble(numeric);
633                         if (!(d.isInfinite() || (d.
                               floatValue() == 0.0D && !
                               allZeros)))
634                         {
635                             return d;
636                         }
637                     }
638                     catch (final NumberFormatException nfe)
639                     {
640                         // NOPMD
641                         // ignore the bad number
642                     }
643                     try
644                     {
645                         return createBigDecimal(numeric);
646                     }
647                     catch (final NumberFormatException e)
648                     {
649                         // NOPMD
650                         // ignore the bad number
```

```
651                     }
652                     //$FALL-THROUGH$
653                 default :
654                     throw new NumberFormatException(str + "
                        is not a valid number.");
655
656             }
657         }
658         //User doesn't have a preference on the return type
               , so let's start
659         //small and go from there...
660         if (expPos > -1 && expPos < str.length() - 1)
661         {
662             exp = str.substring(expPos + 1, str.length());
663         }
664         else
665         {
666             exp = null;
667         }
668         if (dec == null && exp == null)
669         {
670             // no decimal point and no exponent
671             //Must be an Integer, Long, Biginteger
672             try
673             {
674                 return createInteger(str);
675             }
676             catch (final NumberFormatException nfe)
677             {
678                 // NOPMD
679                 // ignore the bad number
680             }
681             try
682             {
683                 return createLong(str);
684             }
685             catch (final NumberFormatException nfe)
686             {
687                 // NOPMD
688                 // ignore the bad number
689             }
690             return createBigInteger(str);
691         }
692
693         //Must be a Float, Double, BigDecimal
694         final boolean allZeros = isAllZeros(mant) &&
```

```
            isAllZeros(exp);
695         try
696         {
697             if(numDecimals <= 7)
698             {
699                 // If number has 7 or fewer digits past the
                        decimal point then make it a float
700                 final Float f = createFloat(str);
701                 if (!(f.isInfinite() || (f.floatValue() ==
                        0.0F && !allZeros)))
702                 {
703                     return f;
704                 }
705             }
706         }
707         catch (final NumberFormatException nfe)
708         {
709             // NOPMD
710             // ignore the bad number
711         }
712         try
713         {
714             if(numDecimals <= 16)
715             {
716                 // If number has between 8 and 16 digits
                        past the decimal point
717                 // then make it a double
718                 final Double d = createDouble(str);
719                 if (!(d.isInfinite() || (d.doubleValue() ==
                        0.0D && !allZeros)))
720                 {
721                     return d;
722                 }
723             }
724         }
725         catch (final NumberFormatException nfe)
726         {
727             // NOPMD
728             // ignore the bad number
729         }
730
731         return createBigDecimal(str);
732     }
733
734     /**
735      * <p>Utility method for {@link #createNumber(java.lang
```

```
         .String)}.</p>
736      *
737      * <p>Returns <code>true</code> if s is <code>null</
             code>.</p>
738      *
739      * @param str  the String to check
740      * @return if it is all zeros or <code>null</code>
741      */
742     private static boolean isAllZeros(final String str)
743     {
744         if (str == null)
745         {
746             return true;
747         }
748         for (int i = str.length() - 1; i >= 0; i--)
749         {
750             if (str.charAt(i) != '0')
751             {
752                 return false;
753             }
754         }
755         return str.length() > 0;
756     }
757
758     //
            -----------------------------------------------------------------------

759     /**
760      * <p>Convert a <code>String</code> to a <code>Float</
             code>.</p>
761      *
762      * <p>Returns <code>null</code> if the string is <code>
             null</code>.</p>
763      *
764      * @param str  a <code>String</code> to convert, may be
              null
765      * @return converted <code>Float</code> (or null if the
              input is null)
766      * @throws NumberFormatException if the value cannot be
              converted
767      */
768     public static Float createFloat(final String str)
769     {
770         if (str == null)
771         {
772             return null;
```

```
773         }
774         return Float.valueOf(str);
775     }
776 
777     /**
778      * <p>Convert a <code>String</code> to a <code>Double</
             code>.</p>
779      *
780      * <p>Returns <code>null</code> if the string is <code>
             null</code>.</p>
781      *
782      * @param str  a <code>String</code> to convert, may be
               null
783      * @return converted <code>Double</code> (or null if
             the input is null)
784      * @throws NumberFormatException if the value cannot be
               converted
785      */
786     public static Double createDouble(final String str)
787     {
788         if (str == null)
789         {
790             return null;
791         }
792         return Double.valueOf(str);
793     }
794 
795     /**
796      * <p>Convert a <code>String</code> to a <code>Integer
             </code>, handling
797      * hex and octal notations.</p>
798      *
799      * <p>Returns <code>null</code> if the string is <code>
             null</code>.</p>
800      *
801      * @param str  a <code>String</code> to convert, may be
               null
802      * @return converted <code>Integer</code> (or null if
             the input is null)
803      * @throws NumberFormatException if the value cannot be
               converted
804      */
805     public static Integer createInteger(final String str)
806     {
807         if (str == null)
808         {
```

```
809             return null;
810         }
811         // decode() handles 0xAABD and 0777 (hex and octal)
                as well.
812         return Integer.decode(str);
813     }
814 
815     /**
816      * <p>Convert a <code>String</code> to a <code>Long</
            code>;
817      * since 3.1 it handles hex and octal notations.</p>
818      *
819      * <p>Returns <code>null</code> if the string is <code>
            null</code>.</p>
820      *
821      * @param str  a <code>String</code> to convert, may be
             null
822      * @return converted <code>Long</code> (or null if the
            input is null)
823      * @throws NumberFormatException if the value cannot be
             converted
824      */
825     public static Long createLong(final String str)
826     {
827         if (str == null)
828         {
829             return null;
830         }
831         return Long.decode(str);
832     }
833 
834     /**
835      * <p>Convert a <code>String</code> to a <code>
            BigInteger</code>;
836      * since 3.2 it handles hex (0x or #) and octal (0)
            notations.</p>
837      *
838      * <p>Returns <code>null</code> if the string is <code>
            null</code>.</p>
839      *
840      * @param str  a <code>String</code> to convert, may be
             null
841      * @return converted <code>BigInteger</code> (or null
            if the input is null)
842      * @throws NumberFormatException if the value cannot be
             converted
```

```
843         */
844         public static BigInteger createBigInteger(final String
                str)
845         {
846             if (str == null)
847             {
848                 return null;
849             }
850             int pos = 0; // offset within string
851             int radix = 10;
852             boolean negate = false; // need to negate later?
853             if (str.startsWith("-"))
854             {
855                 negate = true;
856                 pos = 1;
857             }
858             if (str.startsWith("0x", pos) || str.startsWith("0x
                    ", pos))
859             {
860                 // hex
861                 radix = 16;
862                 pos += 2;
863             }
864             else if (str.startsWith("#", pos))
865             {
866                 // alternative hex (allowed by Long/Integer)
867                 radix = 16;
868                 pos ++;
869             }
870             else if (str.startsWith("0", pos) && str.length() >
                        pos + 1)
871             {
872                 // octal; so long as there are additional
                        digits
873                 radix = 8;
874                 pos ++;
875             } // default is to treat as decimal
876
877             final BigInteger value = new BigInteger(str.
                    substring(pos), radix);
878             return negate ? value.negate() : value;
879         }
880
881         /**
882          * <p>Convert a <code>String</code> to a <code>
                BigDecimal</code>.</p>
```

```
883         *
884         * <p>Returns <code>null</code> if the string is <code>
                null</code>.</p>
885         *
886         * @param str  a <code>String</code> to convert, may be
                null
887         * @return converted <code>BigDecimal</code> (or null
                if the input is null)
888         * @throws NumberFormatException if the value cannot be
                converted
889         */
890        public static BigDecimal createBigDecimal(final String
               str)
891        {
892            if (str == null)
893            {
894                return null;
895            }
896            // handle JDK1.3.1 bug where "" throws
                   IndexOutOfBoundsException
897            if (StringUtils.isBlank(str))
898            {
899                throw new NumberFormatException("A blank string
                       is not a valid number");
900            }
901            if (str.trim().startsWith("--"))
902            {
903                // this is protection for poorness in java.lang
                       .BigDecimal.
904                // it accepts this as a legal value, but it
                       does not appear
905                // to be in specification of class. OS X Java
                       parses it to
906                // a wrong value.
907                throw new NumberFormatException(str + " is not
                       a valid number.");
908            }
909            return new BigDecimal(str);
910        }
911
912        // Min in array
913        //
               ---------------------------------------------------------------------------

914        /**
915         * <p>Returns the minimum value in an array.</p>
```

```
916         *
917         * @param array  an array, must not be null or empty
918         * @return the minimum value in the array
919         * @throws IllegalArgumentException if <code>array</
                code> is <code>null</code>
920         * @throws IllegalArgumentException if <code>array</
                code> is empty
921         */
922        public static long min(final long[] array)
923        {
924            // Validates input
925            validateArray(array);
926
927            // Finds and returns min
928            long min = array[0];
929            for (int i = 1; i < array.length; i++)
930            {
931                if (array[i] < min)
932                {
933                    min = array[i];
934                }
935            }
936
937            return min;
938        }
939
940        /**
941         * <p>Returns the minimum value in an array.</p>
942         *
943         * @param array  an array, must not be null or empty
944         * @return the minimum value in the array
945         * @throws IllegalArgumentException if <code>array</
                code> is <code>null</code>
946         * @throws IllegalArgumentException if <code>array</
                code> is empty
947         */
948        public static int min(final int[] array)
949        {
950            // Validates input
951            validateArray(array);
952
953            // Finds and returns min
954            int min = array[0];
955            for (int j = 1; j < array.length; j++)
956            {
957                if (array[j] < min)
```

```
958             {
959                 min = array[j];
960             }
961         }
962 
963         return min;
964     }
965 
966     /**
967      * <p>Returns the minimum value in an array.</p>
968      *
969      * @param array  an array, must not be null or empty
970      * @return the minimum value in the array
971      * @throws IllegalArgumentException if <code>array</
         code> is <code>null</code>
972      * @throws IllegalArgumentException if <code>array</
         code> is empty
973      */
974     public static short min(final short[] array)
975     {
976         // Validates input
977         validateArray(array);
978 
979         // Finds and returns min
980         short min = array[0];
981         for (int i = 1; i < array.length; i++)
982         {
983             if (array[i] < min)
984             {
985                 min = array[i];
986             }
987         }
988 
989         return min;
990     }
991 
992     /**
993      * <p>Returns the minimum value in an array.</p>
994      *
995      * @param array  an array, must not be null or empty
996      * @return the minimum value in the array
997      * @throws IllegalArgumentException if <code>array</
         code> is <code>null</code>
998      * @throws IllegalArgumentException if <code>array</
         code> is empty
999      */
```

```
1000     public static byte min(final byte[] array)
1001     {
1002         // Validates input
1003         validateArray(array);
1004 
1005         // Finds and returns min
1006         byte min = array[0];
1007         for (int i = 1; i < array.length; i++)
1008         {
1009             if (array[i] < min)
1010             {
1011                 min = array[i];
1012             }
1013         }
1014 
1015         return min;
1016     }
1017 
1018      /**
1019      * <p>Returns the minimum value in an array.</p>
1020      *
1021      * @param array  an array, must not be null or empty
1022      * @return the minimum value in the array
1023      * @throws IllegalArgumentException if <code>array</
            code> is <code>null</code>
1024      * @throws IllegalArgumentException if <code>array</
            code> is empty
1025      * @see IEEE754rUtils#min(double[]) IEEE754rUtils for a
             version of this method that handles NaN
1026      *  differently
1027      */
1028     public static double min(final double[] array)
1029     {
1030         // Validates input
1031         validateArray(array);
1032 
1033         // Finds and returns min
1034         double min = array[0];
1035         for (int i = 1; i < array.length; i++)
1036         {
1037             if (Double.isNaN(array[i]))
1038             {
1039                 return Double.NaN;
1040             }
1041             if (array[i] < min)
1042             {
```

```
1043                 min = array[i];
1044             }
1045         }
1046 
1047         return min;
1048     }
1049 
1050     /**
1051      * <p>Returns the minimum value in an array.</p>
1052      *
1053      * @param array  an array, must not be null or empty
1054      * @return the minimum value in the array
1055      * @throws IllegalArgumentException if <code>array</
          code> is <code>null</code>
1056      * @throws IllegalArgumentException if <code>array</
          code> is empty
1057      * @see IEEE754rUtils#min(float[]) IEEE754rUtils for a
          version of this method that handles NaN
1058      *  differently
1059      */
1060     public static float min(final float[] array)
1061     {
1062         // Validates input
1063         validateArray(array);
1064 
1065         // Finds and returns min
1066         float min = array[0];
1067         for (int i = 1; i < array.length; i++)
1068         {
1069             if (Float.isNaN(array[i]))
1070             {
1071                 return Float.NaN;
1072             }
1073             if (array[i] < min)
1074             {
1075                 min = array[i];
1076             }
1077         }
1078 
1079         return min;
1080     }
1081 
1082     // Max in array
1083     //
        ---------------------------------------------------------------
```

```
1084        /**
1085         * <p>Returns the maximum value in an array.</p>
1086         *
1087         * @param array  an array, must not be null or empty
1088         * @return the minimum value in the array
1089         * @throws IllegalArgumentException if <code>array</
                code> is <code>null</code>
1090         * @throws IllegalArgumentException if <code>array</
                code> is empty
1091         */
1092        public static long max(final long[] array)
1093        {
1094            // Validates input
1095            validateArray(array);
1096
1097            // Finds and returns max
1098            long max = array[0];
1099            for (int j = 1; j < array.length; j++)
1100            {
1101                if (array[j] > max)
1102                {
1103                    max = array[j];
1104                }
1105            }
1106
1107            return max;
1108        }
1109
1110        /**
1111         * <p>Returns the maximum value in an array.</p>
1112         *
1113         * @param array  an array, must not be null or empty
1114         * @return the minimum value in the array
1115         * @throws IllegalArgumentException if <code>array</
                code> is <code>null</code>
1116         * @throws IllegalArgumentException if <code>array</
                code> is empty
1117         */
1118        public static int max(final int[] array)
1119        {
1120            // Validates input
1121            validateArray(array);
1122
1123            // Finds and returns max
1124            int max = array[0];
1125            for (int j = 1; j < array.length; j++)
```

```
1126            {
1127                if (array[j] > max)
1128                {
1129                    max = array[j];
1130                }
1131            }
1132
1133            return max;
1134        }
1135
1136        /**
1137         * <p>Returns the maximum value in an array.</p>
1138         *
1139         * @param array  an array, must not be null or empty
1140         * @return the minimum value in the array
1141         * @throws IllegalArgumentException if <code>array</
                code> is <code>null</code>
1142         * @throws IllegalArgumentException if <code>array</
                code> is empty
1143         */
1144        public static short max(final short[] array)
1145        {
1146            // Validates input
1147            validateArray(array);
1148
1149            // Finds and returns max
1150            short max = array[0];
1151            for (int i = 1; i < array.length; i++)
1152            {
1153                if (array[i] > max)
1154                {
1155                    max = array[i];
1156                }
1157            }
1158
1159            return max;
1160        }
1161
1162        /**
1163         * <p>Returns the maximum value in an array.</p>
1164         *
1165         * @param array  an array, must not be null or empty
1166         * @return the minimum value in the array
1167         * @throws IllegalArgumentException if <code>array</
                code> is <code>null</code>
1168         * @throws IllegalArgumentException if <code>array</
```

```
         code> is empty
1169      */
1170     public static byte max(final byte[] array)
1171     {
1172         // Validates input
1173         validateArray(array);
1174 
1175         // Finds and returns max
1176         byte max = array[0];
1177         for (int i = 1; i < array.length; i++)
1178         {
1179             if (array[i] > max)
1180             {
1181                 max = array[i];
1182             }
1183         }
1184 
1185         return max;
1186     }
1187 
1188     /**
1189      * <p>Returns the maximum value in an array.</p>
1190      *
1191      * @param array  an array, must not be null or empty
1192      * @return the minimum value in the array
1193      * @throws IllegalArgumentException if <code>array</
         code> is <code>null</code>
1194      * @throws IllegalArgumentException if <code>array</
         code> is empty
1195      * @see IEEE754rUtils#max(double[]) IEEE754rUtils for a
          version of this method that handles NaN
1196      *  differently
1197      */
1198     public static double max(final double[] array)
1199     {
1200         // Validates input
1201         validateArray(array);
1202 
1203         // Finds and returns max
1204         double max = array[0];
1205         for (int j = 1; j < array.length; j++)
1206         {
1207             if (Double.isNaN(array[j]))
1208             {
1209                 return Double.NaN;
1210             }
```

```
1211            if (array[j] > max)
1212            {
1213                max = array[j];
1214            }
1215        }
1216
1217        return max;
1218    }
1219
1220    /**
1221     * <p>Returns the maximum value in an array.</p>
1222     *
1223     * @param array  an array, must not be null or empty
1224     * @return the minimum value in the array
1225     * @throws IllegalArgumentException if <code>array</
            code> is <code>null</code>
1226     * @throws IllegalArgumentException if <code>array</
            code> is empty
1227     * @see IEEE754rUtils#max(float[]) IEEE754rUtils for a
            version of this method that handles NaN
1228     *  differently
1229     */
1230    public static float max(final float[] array)
1231    {
1232        // Validates input
1233        validateArray(array);
1234
1235        // Finds and returns max
1236        float max = array[0];
1237        for (int j = 1; j < array.length; j++)
1238        {
1239            if (Float.isNaN(array[j]))
1240            {
1241                return Float.NaN;
1242            }
1243            if (array[j] > max)
1244            {
1245                max = array[j];
1246            }
1247        }
1248
1249        return max;
1250    }
1251
1252    /**
1253     * Checks if the specified array is neither null nor
```

```
         empty.
1254      *
1255      * @param array  the array to check
1256      * @throws IllegalArgumentException if {@code array} is
              either {@code null} or empty
1257      */
1258     private static void validateArray(final Object array)
1259     {
1260         if (array == null)
1261         {
1262             throw new IllegalArgumentException("The Array
                    must not be null");
1263         }
1264         else if (Array.getLength(array) == 0)
1265         {
1266             throw new IllegalArgumentException("Array
                    cannot be empty.");
1267         }
1268     }
1269
1270     // 3 param min
1271     //
            -----------------------------------------------------------------

1272     /**
1273      * <p>Gets the minimum of three <code>long</code>
             values.</p>
1274      *
1275      * @param a  value 1
1276      * @param b  value 2
1277      * @param c  value 3
1278      * @return  the smallest of the values
1279      */
1280     public static long min(long a, final long b, final long
            c)
1281     {
1282         if (b < a)
1283         {
1284             a = b;
1285         }
1286         if (c < a)
1287         {
1288             a = c;
1289         }
1290         return a;
1291     }
```

```
1292
1293        /**
1294         * <p>Gets the minimum of three <code>int</code> values
                .</p>
1295         *
1296         * @param a  value 1
1297         * @param b  value 2
1298         * @param c  value 3
1299         * @return  the smallest of the values
1300         */
1301        public static int min(int a, final int b, final int c)
1302        {
1303            if (b < a)
1304            {
1305                a = b;
1306            }
1307            if (c < a)
1308            {
1309                a = c;
1310            }
1311            return a;
1312        }
1313
1314        /**
1315         * <p>Gets the minimum of three <code>short</code>
                values.</p>
1316         *
1317         * @param a  value 1
1318         * @param b  value 2
1319         * @param c  value 3
1320         * @return  the smallest of the values
1321         */
1322        public static short min(short a, final short b, final
               short c)
1323        {
1324            if (b < a)
1325            {
1326                a = b;
1327            }
1328            if (c < a)
1329            {
1330                a = c;
1331            }
1332            return a;
1333        }
1334
```

```
1335        /**
1336         * <p>Gets the minimum of three <code>byte</code>
                values.</p>
1337         *
1338         * @param a  value 1
1339         * @param b  value 2
1340         * @param c  value 3
1341         * @return  the smallest of the values
1342         */
1343        public static byte min(byte a, final byte b, final byte
                c)
1344        {
1345            if (b < a)
1346            {
1347                a = b;
1348            }
1349            if (c < a)
1350            {
1351                a = c;
1352            }
1353            return a;
1354        }
1355
1356        /**
1357         * <p>Gets the minimum of three <code>double</code>
                values.</p>
1358         *
1359         * <p>If any value is <code>NaN</code>, <code>NaN</code
                > is
1360         * returned. Infinity is handled.</p>
1361         *
1362         * @param a  value 1
1363         * @param b  value 2
1364         * @param c  value 3
1365         * @return  the smallest of the values
1366         * @see IEEE754rUtils#min(double, double, double) for a
                 version of this method that handles NaN
1367         *  differently
1368         */
1369        public static double min(final double a, final double b
               , final double c)
1370        {
1371            return Math.min(Math.min(a, b), c);
1372        }
1373
1374        /**
```

```
1375        * <p>Gets the minimum of three <code>float</code>
               values.</p>
1376        *
1377        * <p>If any value is <code>NaN</code>, <code>NaN</code
               > is
1378        * returned. Infinity is handled.</p>
1379        *
1380        * @param a  value 1
1381        * @param b  value 2
1382        * @param c  value 3
1383        * @return  the smallest of the values
1384        * @see IEEE754rUtils#min(float, float, float) for a
               version of this method that handles NaN
1385        *  differently
1386        */
1387       public static float min(final float a, final float b,
               final float c)
1388       {
1389           return Math.min(Math.min(a, b), c);
1390       }
1391
1392       // 3 param max
1393       //
               -----------------------------------------------------------------

1394       /**
1395        * <p>Gets the maximum of three <code>long</code>
               values.</p>
1396        *
1397        * @param a  value 1
1398        * @param b  value 2
1399        * @param c  value 3
1400        * @return  the largest of the values
1401        */
1402       public static long max(long a, final long b, final long
               c)
1403       {
1404           if (b > a)
1405           {
1406               a = b;
1407           }
1408           if (c > a)
1409           {
1410               a = c;
1411           }
1412           return a;
```

```
1413        }
1414
1415        /**
1416         * <p>Gets the maximum of three <code>int</code> values
                .</p>
1417         *
1418         * @param a  value 1
1419         * @param b  value 2
1420         * @param c  value 3
1421         * @return  the largest of the values
1422         */
1423        public static int max(int a, final int b, final int c)
1424        {
1425            if (b > a)
1426            {
1427                a = b;
1428            }
1429            if (c > a)
1430            {
1431                a = c;
1432            }
1433            return a;
1434        }
1435
1436        /**
1437         * <p>Gets the maximum of three <code>short</code>
                values.</p>
1438         *
1439         * @param a  value 1
1440         * @param b  value 2
1441         * @param c  value 3
1442         * @return  the largest of the values
1443         */
1444        public static short max(short a, final short b, final
              short c)
1445        {
1446            if (b > a)
1447            {
1448                a = b;
1449            }
1450            if (c > a)
1451            {
1452                a = c;
1453            }
1454            return a;
1455        }
```

```
1456
1457        /**
1458         * <p>Gets the maximum of three <code>byte</code>
                values.</p>
1459         *
1460         * @param a  value 1
1461         * @param b  value 2
1462         * @param c  value 3
1463         * @return  the largest of the values
1464         */
1465        public static byte max(byte a, final byte b, final byte
                c)
1466        {
1467            if (b > a)
1468            {
1469                a = b;
1470            }
1471            if (c > a)
1472            {
1473                a = c;
1474            }
1475            return a;
1476        }
1477
1478        /**
1479         * <p>Gets the maximum of three <code>double</code>
                values.</p>
1480         *
1481         * <p>If any value is <code>NaN</code>, <code>NaN</code
                > is
1482         * returned. Infinity is handled.</p>
1483         *
1484         * @param a  value 1
1485         * @param b  value 2
1486         * @param c  value 3
1487         * @return  the largest of the values
1488         * @see IEEE754rUtils#max(double, double, double) for a
                    version of this method that handles NaN
1489         *  differently
1490         */
1491        public static double max(final double a, final double b
               , final double c)
1492        {
1493            return Math.max(Math.max(a, b), c);
1494        }
1495
```

```
1496        /**
1497         * <p>Gets the maximum of three <code>float</code>
                values.</p>
1498         *
1499         * <p>If any value is <code>NaN</code>, <code>NaN</code
                > is
1500         * returned. Infinity is handled.</p>
1501         *
1502         * @param a  value 1
1503         * @param b  value 2
1504         * @param c  value 3
1505         * @return  the largest of the values
1506         * @see IEEE754rUtils#max(float, float, float) for a
                version of this method that handles NaN
1507         *  differently
1508         */
1509        public static float max(final float a, final float b,
               final float c)
1510        {
1511            return Math.max(Math.max(a, b), c);
1512        }
1513
1514        //
               -----------------------------------------------------------------------

1515        /**
1516         * <p>Checks whether the <code>String</code> contains
                only
1517         * digit characters.</p>
1518         *
1519         * <p><code>Null</code> and empty String will return
1520         * <code>false</code>.</p>
1521         *
1522         * @param str  the <code>String</code> to check
1523         * @return <code>true</code> if str contains only
                Unicode numeric
1524         */
1525        public static boolean isDigits(final String str)
1526        {
1527            if (StringUtils.isEmpty(str))
1528            {
1529                return false;
1530            }
1531            for (int i = 0; i < str.length(); i++)
1532            {
1533                if (!Character.isDigit(str.charAt(i)))
```

```
1534            {
1535                return false;
1536            }
1537        }
1538        return true;
1539    }
1540
1541    /**
1542     * <p>Checks whether the String a valid Java number.</p
            >
1543     *
1544     * <p>Valid numbers include hexadecimal marked with the
              <code>0x</code>
1545     * qualifier, scientific notation and numbers marked
            with a type
1546     * qualifier (e.g. 123L).</p>
1547     *
1548     * <p><code>Null</code> and empty String will return
1549     * <code>false</code>.</p>
1550     *
1551     * @param str  the <code>String</code> to check
1552     * @return <code>true</code> if the string is a
            correctly formatted number
1553     */
1554    public static boolean isNumber(final String str)
1555    {
1556        if (StringUtils.isEmpty(str))
1557        {
1558            return false;
1559        }
1560        final char[] chars = str.toCharArray();
1561        int sz = chars.length;
1562        boolean hasExp = false;
1563        boolean hasDecPoint = false;
1564        boolean allowSigns = false;
1565        boolean foundDigit = false;
1566        // deal with any possible sign up front
1567        final int start = (chars[0] == '-') ? 1 : 0;
1568        if (sz > start + 1 && chars[start] == '0' && chars[
              start + 1] == 'x')
1569        {
1570            int i = start + 2;
1571            if (i == sz)
1572            {
1573                return false; // str == "0x"
1574            }
```

```
1575            // checking hex (it can't be anything else)
1576            for (; i < chars.length; i++)
1577            {
1578                if ((chars[i] < '0' || chars[i] > '9')
1579                    && (chars[i] < 'a' || chars[i] > 'f')
1580                    && (chars[i] < 'A' || chars[i] > 'F'))
1581                    {
1582                    return false;
1583                }
1584            }
1585            return true;
1586        }
1587        sz--; // don't want to loop to the last char, check
                it afterwords
1588              // for type qualifiers
1589        int i = start;
1590        // loop to the next to last char or to the last
              char if we need another digit to
1591        // make a valid number (e.g. chars[0..5] = "1234E")
1592        while (i < sz || (i < sz + 1 && allowSigns && !
              foundDigit))
1593        {
1594            if (chars[i] >= '0' && chars[i] <= '9')
1595            {
1596                foundDigit = true;
1597                allowSigns = false;
1598
1599            }
1600            else if (chars[i] == '.')
1601            {
1602                if (hasDecPoint || hasExp)
1603                {
1604                    // two decimal points or dec in
                           exponent
1605                    return false;
1606                }
1607                hasDecPoint = true;
1608            }
1609            else if (chars[i] == 'e' || chars[i] == 'E')
1610            {
1611                // we've already taken care of hex.
1612                if (hasExp)
1613                {
1614                    // two E's
1615                    return false;
1616                }
```

```
1617                if (!foundDigit)
1618                {
1619                    return false;
1620                }
1621                hasExp = true;
1622                allowSigns = true;
1623            }
1624            else if (chars[i] == '+' || chars[i] == '-')
1625            {
1626                if (!allowSigns)
1627                {
1628                    return false;
1629                }
1630                allowSigns = false;
1631                foundDigit = false; // we need a digit
                       after the E
1632            }
1633            else
1634            {
1635                return false;
1636            }
1637            i++;
1638        }
1639        if (i < chars.length)
1640        {
1641            if (chars[i] >= '0' && chars[i] <= '9')
1642            {
1643                // no type qualifier, OK
1644                return true;
1645            }
1646            if (chars[i] == 'e' || chars[i] == 'E')
1647            {
1648                // can't have an E at the last byte
1649                return false;
1650            }
1651            if (chars[i] == '.')
1652            {
1653                if (hasDecPoint || hasExp)
1654                {
1655                    // two decimal points or dec in
                           exponent
1656                    return false;
1657                }
1658                // single trailing decimal point after non-
                       exponent is ok
1659                return foundDigit;
```

```
1660            }
1661            if (!allowSigns
1662                && (chars[i] == 'd'
1663                    || chars[i] == 'D'
1664                    || chars[i] == 'f'
1665                    || chars[i] == 'F'))
1666                    {
1667                return foundDigit;
1668            }
1669            if (chars[i] == 'l'
1670                || chars[i] == 'L')
1671                {
1672                // not allowing L with an exponent or
                       decimal point
1673                return foundDigit && !hasExp && !
                       hasDecPoint;
1674            }
1675            // last character is illegal
1676            return false;
1677        }
1678        // allowSigns is true iff the val ends in 'E'
1679        // found digit it to make sure weird stuff like '.'
               and '1E-' doesn't pass
1680        return !allowSigns && foundDigit;
1681    }
1682
1683 }
```

E.3.4 Defects-Sources/Lang-1/NumberUtils-fixed-B.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under
       one or more
3   * contributor license agreements.  See the NOTICE file
       distributed with
4   * this work for additional information regarding copyright
        ownership.
5   * The ASF licenses this file to You under the Apache
       License, Version 2.0
6   * (the "License"); you may not use this file except in
       compliance with
7   * the License.  You may obtain a copy of the License at
8   *
9   *      http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in
       writing, software
12  * distributed under the License is distributed on an "AS
       IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
       express or implied.
14  * See the License for the specific language governing
       permissions and
15  * limitations under the License.
16  */
17 package org.apache.commons.lang3.math;
18
19 import java.lang.reflect.Array;
20 import java.math.BigDecimal;
21 import java.math.BigInteger;
22
23 import org.apache.commons.lang3.StringUtils;
24
25 /**
26  * <p>Provides extra functionality for Java Number classes
       .</p>
27  *
28  * @since 2.0
29  * @version $Id$
30  */
31 public class NumberUtils
32 {
33
34     /** Reusable Long constant for zero. */
```

```
35      public static final Long LONG_ZERO = Long.valueOf(0L);
36      /** Reusable Long constant for one. */
37      public static final Long LONG_ONE = Long.valueOf(1L);
38      /** Reusable Long constant for minus one. */
39      public static final Long LONG_MINUS_ONE = Long.valueOf
            (-1L);
40      /** Reusable Integer constant for zero. */
41      public static final Integer INTEGER_ZERO = Integer.
            valueOf(0);
42      /** Reusable Integer constant for one. */
43      public static final Integer INTEGER_ONE = Integer.
            valueOf(1);
44      /** Reusable Integer constant for minus one. */
45      public static final Integer INTEGER_MINUS_ONE = Integer
            .valueOf(-1);
46      /** Reusable Short constant for zero. */
47      public static final Short SHORT_ZERO = Short.valueOf((
            short) 0);
48      /** Reusable Short constant for one. */
49      public static final Short SHORT_ONE = Short.valueOf((
            short) 1);
50      /** Reusable Short constant for minus one. */
51      public static final Short SHORT_MINUS_ONE = Short.
            valueOf((short) -1);
52      /** Reusable Byte constant for zero. */
53      public static final Byte BYTE_ZERO = Byte.valueOf((byte
            ) 0);
54      /** Reusable Byte constant for one. */
55      public static final Byte BYTE_ONE = Byte.valueOf((byte)
             1);
56      /** Reusable Byte constant for minus one. */
57      public static final Byte BYTE_MINUS_ONE = Byte.valueOf
            ((byte) -1);
58      /** Reusable Double constant for zero. */
59      public static final Double DOUBLE_ZERO = Double.valueOf
            (0.0d);
60      /** Reusable Double constant for one. */
61      public static final Double DOUBLE_ONE = Double.valueOf
            (1.0d);
62      /** Reusable Double constant for minus one. */
63      public static final Double DOUBLE_MINUS_ONE = Double.
            valueOf(-1.0d);
64      /** Reusable Float constant for zero. */
65      public static final Float FLOAT_ZERO = Float.valueOf
            (0.0f);
66      /** Reusable Float constant for one. */
```

```
67      public static final Float FLOAT_ONE = Float.valueOf(1.0
           f);
68      /** Reusable Float constant for minus one. */
69      public static final Float FLOAT_MINUS_ONE = Float.
           valueOf(-1.0f);
70
71      /**
72       * <p><code>NumberUtils</code> instances should NOT be
            constructed in standard programming.
73       * Instead, the class should be used as <code>
            NumberUtils.toInt("6");</code>.</p>
74       *
75       * <p>This constructor is public to permit tools that
            require a JavaBean instance
76       * to operate.</p>
77       */
78      public NumberUtils()
79      {
80          super();
81      }
82
83      //
           -----------------------------------------------------------------

84      /**
85       * <p>Convert a <code>String</code> to an <code>int</
            code>, returning
86       * <code>zero</code> if the conversion fails.</p>
87       *
88       * <p>If the string is <code>null</code>, <code>zero</
            code> is returned.</p>
89       *
90       * <pre>
91       *   NumberUtils.toInt(null) = 0
92       *   NumberUtils.toInt("")   = 0
93       *   NumberUtils.toInt("1")  = 1
94       * </pre>
95       *
96       * @param str  the string to convert, may be null
97       * @return the int represented by the string, or <code>
            zero</code> if
98       *  conversion fails
99       * @since 2.1
100      */
101     public static int toInt(final String str)
102     {
```

```
103         return toInt(str, 0);
104     }
105 
106     /**
107      * <p>Convert a <code>String</code> to an <code>int</
             code>, returning a
108      * default value if the conversion fails.</p>
109      *
110      * <p>If the string is <code>null</code>, the default
             value is returned.</p>
111      *
112      * <pre>
113      *   NumberUtils.toInt(null, 1) = 1
114      *   NumberUtils.toInt("", 1)   = 1
115      *   NumberUtils.toInt("1", 0)  = 1
116      * </pre>
117      *
118      * @param str  the string to convert, may be null
119      * @param defaultValue  the default value
120      * @return the int represented by the string, or the
             default if conversion fails
121      * @since 2.1
122      */
123     public static int toInt(final String str, final int
           defaultValue)
124     {
125         if(str == null)
126         {
127             return defaultValue;
128         }
129         try
130         {
131             return Integer.parseInt(str);
132         }
133         catch (final NumberFormatException nfe)
134         {
135             return defaultValue;
136         }
137     }
138 
139     /**
140      * <p>Convert a <code>String</code> to a <code>long</
             code>, returning
141      * <code>zero</code> if the conversion fails.</p>
142      *
143      * <p>If the string is <code>null</code>, <code>zero</
```

```
         code> is returned.</p>
144       *
145       * <pre>
146       *   NumberUtils.toLong(null) = 0L
147       *   NumberUtils.toLong("")   = 0L
148       *   NumberUtils.toLong("1")  = 1L
149       * </pre>
150       *
151       * @param str  the string to convert, may be null
152       * @return the long represented by the string, or <code
             >0</code> if
153       *  conversion fails
154       * @since 2.1
155       */
156      public static long toLong(final String str)
157      {
158          return toLong(str, 0L);
159      }
160
161      /**
162       * <p>Convert a <code>String</code> to a <code>long</
             code>, returning a
163       * default value if the conversion fails.</p>
164       *
165       * <p>If the string is <code>null</code>, the default
             value is returned.</p>
166       *
167       * <pre>
168       *   NumberUtils.toLong(null, 1L) = 1L
169       *   NumberUtils.toLong("", 1L)   = 1L
170       *   NumberUtils.toLong("1", 0L)  = 1L
171       * </pre>
172       *
173       * @param str  the string to convert, may be null
174       * @param defaultValue  the default value
175       * @return the long represented by the string, or the
             default if conversion fails
176       * @since 2.1
177       */
178      public static long toLong(final String str, final long
           defaultValue)
179      {
180          if (str == null)
181          {
182              return defaultValue;
183          }
```

```
184         try
185         {
186             return Long.parseLong(str);
187         }
188         catch (final NumberFormatException nfe)
189         {
190             return defaultValue;
191         }
192     }
193 
194     /**
195      * <p>Convert a <code>String</code> to a <code>float</
             code>, returning
196      * <code>0.0f</code> if the conversion fails.</p>
197      *
198      * <p>If the string <code>str</code> is <code>null</
             code>,
199      * <code>0.0f</code> is returned.</p>
200      *
201      * <pre>
202      *   NumberUtils.toFloat(null)   = 0.0f
203      *   NumberUtils.toFloat("")     = 0.0f
204      *   NumberUtils.toFloat("1.5")  = 1.5f
205      * </pre>
206      *
207      * @param str the string to convert, may be <code>null
             </code>
208      * @return the float represented by the string, or <
             code>0.0f</code>
209      *  if conversion fails
210      * @since 2.1
211      */
212     public static float toFloat(final String str)
213     {
214         return toFloat(str, 0.0f);
215     }
216 
217     /**
218      * <p>Convert a <code>String</code> to a <code>float</
             code>, returning a
219      * default value if the conversion fails.</p>
220      *
221      * <p>If the string <code>str</code> is <code>null</
             code>, the default
222      * value is returned.</p>
223      *
```

```
224         * <pre>
225         *   NumberUtils.toFloat(null, 1.1f)   = 1.0f
226         *   NumberUtils.toFloat("", 1.1f)     = 1.1f
227         *   NumberUtils.toFloat("1.5", 0.0f)  = 1.5f
228         * </pre>
229         *
230         * @param str the string to convert, may be <code>null
                </code>
231         * @param defaultValue the default value
232         * @return the float represented by the string, or
                defaultValue
233         *  if conversion fails
234         * @since 2.1
235         */
236        public static float toFloat(final String str, final
               float defaultValue)
237        {
238            if (str == null)
239            {
240                return defaultValue;
241            }
242            try
243            {
244                return Float.parseFloat(str);
245            }
246            catch (final NumberFormatException nfe)
247            {
248                return defaultValue;
249            }
250        }
251
252        /**
253         * <p>Convert a <code>String</code> to a <code>double</
                code>, returning
254         * <code>0.0d</code> if the conversion fails.</p>
255         *
256         * <p>If the string <code>str</code> is <code>null</
                code>,
257         * <code>0.0d</code> is returned.</p>
258         *
259         * <pre>
260         *   NumberUtils.toDouble(null)   = 0.0d
261         *   NumberUtils.toDouble("")     = 0.0d
262         *   NumberUtils.toDouble("1.5")  = 1.5d
263         * </pre>
264         *
```

```
265         * @param str the string to convert, may be <code>null
                </code>
266         * @return the double represented by the string, or <
                code>0.0d</code>
267         *  if conversion fails
268         * @since 2.1
269         */
270        public static double toDouble(final String str)
271        {
272            return toDouble(str, 0.0d);
273        }
274
275        /**
276         * <p>Convert a <code>String</code> to a <code>double</
                code>, returning a
277         * default value if the conversion fails.</p>
278         *
279         * <p>If the string <code>str</code> is <code>null</
                code>, the default
280         * value is returned.</p>
281         *
282         * <pre>
283         *   NumberUtils.toDouble(null, 1.1d)   = 1.1d
284         *   NumberUtils.toDouble("", 1.1d)     = 1.1d
285         *   NumberUtils.toDouble("1.5", 0.0d)  = 1.5d
286         * </pre>
287         *
288         * @param str the string to convert, may be <code>null
                </code>
289         * @param defaultValue the default value
290         * @return the double represented by the string, or
                defaultValue
291         *  if conversion fails
292         * @since 2.1
293         */
294        public static double toDouble(final String str, final
               double defaultValue)
295        {
296            if (str == null)
297            {
298                return defaultValue;
299            }
300            try
301            {
302                return Double.parseDouble(str);
303            }
```

```
304         catch (final NumberFormatException nfe)
305         {
306             return defaultValue;
307         }
308     }
309 
310     //
        -----------------------------------------------------------------

311     /**
312      * <p>Convert a <code>String</code> to a <code>byte</
            code>, returning
313      * <code>zero</code> if the conversion fails.</p>
314      *
315      * <p>If the string is <code>null</code>, <code>zero</
            code> is returned.</p>
316      *
317      * <pre>
318      *   NumberUtils.toByte(null) = 0
319      *   NumberUtils.toByte("")   = 0
320      *   NumberUtils.toByte("1")  = 1
321      * </pre>
322      *
323      * @param str  the string to convert, may be null
324      * @return the byte represented by the string, or <code
            >zero</code> if
325      *  conversion fails
326      * @since 2.5
327      */
328     public static byte toByte(final String str)
329     {
330         return toByte(str, (byte) 0);
331     }
332 
333     /**
334      * <p>Convert a <code>String</code> to a <code>byte</
            code>, returning a
335      * default value if the conversion fails.</p>
336      *
337      * <p>If the string is <code>null</code>, the default
            value is returned.</p>
338      *
339      * <pre>
340      *   NumberUtils.toByte(null, 1) = 1
341      *   NumberUtils.toByte("", 1)   = 1
342      *   NumberUtils.toByte("1", 0)  = 1
```

```
343         * </pre>
344         *
345         * @param str  the string to convert, may be null
346         * @param defaultValue  the default value
347         * @return the byte represented by the string, or the
                default if conversion fails
348         * @since 2.5
349         */
350        public static byte toByte(final String str, final byte
               defaultValue)
351        {
352            if(str == null)
353            {
354                return defaultValue;
355            }
356            try
357            {
358                return Byte.parseByte(str);
359            }
360            catch (final NumberFormatException nfe)
361            {
362                return defaultValue;
363            }
364        }
365
366        /**
367         * <p>Convert a <code>String</code> to a <code>short</
                code>, returning
368         * <code>zero</code> if the conversion fails.</p>
369         *
370         * <p>If the string is <code>null</code>, <code>zero</
                code> is returned.</p>
371         *
372         * <pre>
373         *   NumberUtils.toShort(null) = 0
374         *   NumberUtils.toShort("")   = 0
375         *   NumberUtils.toShort("1")  = 1
376         * </pre>
377         *
378         * @param str  the string to convert, may be null
379         * @return the short represented by the string, or <
                code>zero</code> if
380         *  conversion fails
381         * @since 2.5
382         */
383        public static short toShort(final String str)
```

```
384     {
385         return toShort(str, (short) 0);
386     }
387 
388     /**
389      * <p>Convert a <code>String</code> to an <code>short</
            code>, returning a
390      * default value if the conversion fails.</p>
391      *
392      * <p>If the string is <code>null</code>, the default
            value is returned.</p>
393      *
394      * <pre>
395      *   NumberUtils.toShort(null, 1) = 1
396      *   NumberUtils.toShort("", 1)   = 1
397      *   NumberUtils.toShort("1", 0)  = 1
398      * </pre>
399      *
400      * @param str  the string to convert, may be null
401      * @param defaultValue  the default value
402      * @return the short represented by the string, or the
            default if conversion fails
403      * @since 2.5
404      */
405     public static short toShort(final String str, final
           short defaultValue)
406     {
407         if(str == null)
408         {
409             return defaultValue;
410         }
411         try
412         {
413             return Short.parseShort(str);
414         }
415         catch (final NumberFormatException nfe)
416         {
417             return defaultValue;
418         }
419     }
420 
421     //
           -----------------------------------------------------------------------
422     // must handle Long, Float, Integer, Float, Short,
423     //                  BigDecimal, BigInteger and Byte
```

```
424     // useful methods:
425     // Byte.decode(String)
426     // Byte.valueOf(String,int radix)
427     // Byte.valueOf(String)
428     // Double.valueOf(String)
429     // Float.valueOf(String)
430     // Float.valueOf(String)
431     // Integer.valueOf(String,int radix)
432     // Integer.valueOf(String)
433     // Integer.decode(String)
434     // Integer.getInteger(String)
435     // Integer.getInteger(String,int val)
436     // Integer.getInteger(String,Integer val)
437     // Integer.valueOf(String)
438     // Double.valueOf(String)
439     // new Byte(String)
440     // Long.valueOf(String)
441     // Long.getLong(String)
442     // Long.getLong(String,int)
443     // Long.getLong(String,Integer)
444     // Long.valueOf(String,int)
445     // Long.valueOf(String)
446     // Short.valueOf(String)
447     // Short.decode(String)
448     // Short.valueOf(String,int)
449     // Short.valueOf(String)
450     // new BigDecimal(String)
451     // new BigInteger(String)
452     // new BigInteger(String,int radix)
453     // Possible inputs:
454     // 45 45.5 45E7 4.5E7 Hex Oct Binary xxxF xxxD xxxf
           xxxd
455     // plus minus everything. Prolly more. A lot are not
           separable.
456
457     /**
458      * <p>Turns a string value into a java.lang.Number.</p>
459      *
460      * <p>If the string starts with {@code 0x} or {@code -0
           x} (lower or upper case) or {@code #} or
461      * {@code -#}, it will be interpreted as a hexadecimal
            Integer - or Long, if the number of
462      * digits after the prefix is more than 8 - or
            BigInteger if there are more than 16 digits.
463      * </p>
464      * <p>Then, the value is examined for a type qualifier
```

```
         on the end, i.e. one of
465      * <code>'f','F','d','D','l','L'</code>.  If it is
             found, it starts
466      * trying to create successively larger types from the
             type specified
467      * until one is found that can represent the value.</p>
468      *
469      * <p>If a type specifier is not found, it will check
             for a decimal point
470      * and then try successively larger types from <code>
             Integer</code> to
471      * <code>BigInteger</code> and from <code>Float</code>
             to
472      * <code>BigDecimal</code>.</p>
473      *
474      * <p>
475      * Integral values with a leading {@code 0} will be
             interpreted as octal; the returned number
476      * will be Integer, Long or BigDecimal as appropriate.
477      * </p>
478      *
479      * <p>Returns <code>null</code> if the string is <code>
             null</code>.</p>
480      *
481      * <p>This method does not trim the input string, i.e.,
               strings with leading
482      * or trailing spaces will generate
             NumberFormatExceptions.</p>
483      *
484      * @param str  String containing a number, may be null
485      * @return Number created from the string (or null if
             the input is null)
486      * @throws NumberFormatException if the value cannot be
               converted
487      */
488     public static Number createNumber(final String str)
           throws NumberFormatException
489     {
490         if (str == null)
491         {
492             return null;
493         }
494         if (StringUtils.isBlank(str))
495         {
496             throw new NumberFormatException("A blank string
                    is not a valid number");
```

```
497         }
498         // Need to deal with all possible hex prefixes here
499         final String[] hex_prefixes = {"0x", "0X", "-0x", "
               -0X", "#", "-#"};
500         int pfxLen = 0;
501         for(final String pfx : hex_prefixes)
502         {
503             if (str.startsWith(pfx))
504             {
505                 pfxLen += pfx.length();
506                 break;
507             }
508         }
509         if (pfxLen > 0)
510         {
511             // we have a hex number
512             char firstSigDigit = 0; // strip leading zeroes
513             for(int i = pfxLen; i < str.length(); i++)
514             {
515                 firstSigDigit = str.charAt(i);
516                 if (firstSigDigit == '0')
517                 {
518                     // count leading zeroes
519                     pfxLen++;
520                 }
521                 else
522                 {
523                     break;
524                 }
525             }
526             final int hexDigits = str.length() - pfxLen;
527             if (hexDigits > 16 || (hexDigits == 16 &&
                   firstSigDigit > '7'))
528             {
529                 // too many for Long
530                 return createBigInteger(str);
531             }
532             if (hexDigits > 8 || (hexDigits == 8 &&
                   firstSigDigit > '7'))
533             {
534                 // too many for an int
535                 return createLong(str);
536             }
537             return createInteger(str);
538         }
539         final char lastChar = str.charAt(str.length() - 1);
```

```
540         String mant;
541         String dec;
542         String exp;
543         final int decPos = str.indexOf('.');
544         final int expPos = str.indexOf('e') + str.indexOf('
               E') + 1; // assumes both not present
545         // if both e and E are present, this is caught by
               the checks on expPos (which prevent IOOBE)
546         // and the parsing which will detect if e or E
               appear in a number due to using the wrong
547         // offset
548
549         int numDecimals = 0; // Check required precision (
               LANG-693)
550         if (decPos > -1)
551         {
552             // there is a decimal point
553             if (expPos > -1)
554             {
555                 // there is an exponent
556                 if (expPos < decPos || expPos > str.length
                       ())
557                 {
558                     //prevents double exponent causing
                           IOOBE
559                     throw new NumberFormatException(str + "
                           is not a valid number.");
560                 }
561                 dec = str.substring(decPos + 1, expPos);
562             }
563             else
564             {
565                 dec = str.substring(decPos + 1);
566             }
567             mant = str.substring(0, decPos);
568             numDecimals = dec.length(); // gets number of
                   digits past the decimal to ensure no loss
569                                         // of precision for
                                               floating point
                                              numbers.
570         }
571         else
572         {
573             if (expPos > -1)
574             {
575                 if (expPos > str.length())
```

```
576                 {
577                     // prevents double exponent causing
                        IOOBE
578                     throw new NumberFormatException(str + "
                            is not a valid number.");
579                 }
580                 mant = str.substring(0, expPos);
581             }
582             else
583             {
584                 mant = str;
585             }
586             dec = null;
587         }
588         if (!Character.isDigit(lastChar) && lastChar != '.'
            )
589         {
590             if (expPos > -1 && expPos < str.length() - 1)
591             {
592                 exp = str.substring(expPos + 1, str.length
                        () - 1);
593             }
594             else
595             {
596                 exp = null;
597             }
598             //Requesting a specific type..
599             final String numeric = str.substring(0, str.
                length() - 1);
600             final boolean allZeros = isAllZeros(mant) &&
                isAllZeros(exp);
601             switch (lastChar)
602             {
603                 case 'l' :
604                 case 'L' :
605                     if (dec == null
606                         && exp == null
607                         && (numeric.charAt(0) == '-' &&
                            isDigits(numeric.substring(1))
608                         || isDigits(numeric)))
609                     {
610                         try
611                         {
612                             return createLong(numeric);
613                         }
614                         catch (final NumberFormatException
```

```
                        nfe)
615                 {
616                     // NOPMD
617                     // Too big for a long
618                 }
619                 return createBigInteger(numeric);
620
621             }
622             throw new NumberFormatException(str + "
                    is not a valid number.");
623         case 'f' :
624         case 'F' :
625             try
626             {
627                 final Float f = NumberUtils.
                        createFloat(numeric);
628                 if (!(f.isInfinite() || (f.
                        floatValue() == 0.0F && !
                        allZeros)))
629                 {
630                     //If it's too big for a float
                            or the float value = 0 and
                            the string
631                     //has non-zeros in it, then
                            float does not have the
                            precision we want
632                     return f;
633                 }
634
635             }
636             catch (final NumberFormatException nfe)
637             {
638                 // NOPMD
639                 // ignore the bad number
640             }
641             //$FALL-THROUGH$
642         case 'd' :
643         case 'D' :
644             try
645             {
646                 final Double d = NumberUtils.
                        createDouble(numeric);
647                 if (!(d.isInfinite() || (d.
                        floatValue() == 0.0D && !
                        allZeros)))
648                 {
```

```
649                             return d;
650                         }
651                     }
652                     catch (final NumberFormatException nfe)
653                     {
654                         // NOPMD
655                         // ignore the bad number
656                     }
657                     try
658                     {
659                         return createBigDecimal(numeric);
660                     }
661                     catch (final NumberFormatException e)
662                     {
663                         // NOPMD
664                         // ignore the bad number
665                     }
666                     //$FALL-THROUGH$
667                 default :
668                     throw new NumberFormatException(str + "
                        is not a valid number.");
669 
670             }
671         }
672         //User doesn't have a preference on the return type
               , so let's start
673         //small and go from there...
674         if (expPos > -1 && expPos < str.length() - 1)
675         {
676             exp = str.substring(expPos + 1, str.length());
677         }
678         else
679         {
680             exp = null;
681         }
682         if (dec == null && exp == null)
683         {
684             // no decimal point and no exponent
685             //Must be an Integer, Long, Biginteger
686             try
687             {
688                 return createInteger(str);
689             }
690             catch (final NumberFormatException nfe)
691             {
692                 // NOPMD
```

```
693                 // ignore the bad number
694             }
695             try
696             {
697                 return createLong(str);
698             }
699             catch (final NumberFormatException nfe)
700             {
701                 // NOPMD
702                 // ignore the bad number
703             }
704             return createBigInteger(str);
705         }
706 
707         //Must be a Float, Double, BigDecimal
708         final boolean allZeros = isAllZeros(mant) &&
                isAllZeros(exp);
709         try
710         {
711             if(numDecimals <= 7)
712             {
713                 // If number has 7 or fewer digits past the
                        decimal point then make it a float
714                 final Float f = createFloat(str);
715                 if (!(f.isInfinite() || (f.floatValue() ==
                        0.0F && !allZeros)))
716                 {
717                     return f;
718                 }
719             }
720         }
721         catch (final NumberFormatException nfe)
722         {
723             // NOPMD
724             // ignore the bad number
725         }
726         try
727         {
728             if(numDecimals <= 16)
729             {
730                 // If number has between 8 and 16 digits
                        past the decimal point
731                 // then make it a double
732                 final Double d = createDouble(str);
733                 if (!(d.isInfinite() || (d.doubleValue() ==
                        0.0D && !allZeros)))
```

```
734                 {
735                     return d;
736                 }
737             }
738         }
739         catch (final NumberFormatException nfe)
740         {
741             // NOPMD
742             // ignore the bad number
743         }
744 
745         return createBigDecimal(str);
746     }
747 
748     /**
749      * <p>Utility method for {@link #createNumber(java.lang
         .String)}.</p>
750      *
751      * <p>Returns <code>true</code> if s is <code>null</
         code>.</p>
752      *
753      * @param str  the String to check
754      * @return if it is all zeros or <code>null</code>
755      */
756     private static boolean isAllZeros(final String str)
757     {
758         if (str == null)
759         {
760             return true;
761         }
762         for (int i = str.length() - 1; i >= 0; i--)
763         {
764             if (str.charAt(i) != '0')
765             {
766                 return false;
767             }
768         }
769         return str.length() > 0;
770     }
771 
772     //
        -----------------------------------------------------------------------
773     /**
774      * <p>Convert a <code>String</code> to a <code>Float</
         code>.</p>
```

```
775         *
776         * <p>Returns <code>null</code> if the string is <code>
                null</code>.</p>
777         *
778         * @param str  a <code>String</code> to convert, may be
                null
779         * @return converted <code>Float</code> (or null if the
                input is null)
780         * @throws NumberFormatException if the value cannot be
                converted
781         */
782        public static Float createFloat(final String str)
783        {
784            if (str == null)
785            {
786                return null;
787            }
788            return Float.valueOf(str);
789        }
790
791        /**
792         * <p>Convert a <code>String</code> to a <code>Double</
                code>.</p>
793         *
794         * <p>Returns <code>null</code> if the string is <code>
                null</code>.</p>
795         *
796         * @param str  a <code>String</code> to convert, may be
                null
797         * @return converted <code>Double</code> (or null if
                the input is null)
798         * @throws NumberFormatException if the value cannot be
                converted
799         */
800        public static Double createDouble(final String str)
801        {
802            if (str == null)
803            {
804                return null;
805            }
806            return Double.valueOf(str);
807        }
808
809        /**
810         * <p>Convert a <code>String</code> to a <code>Integer
                </code>, handling
```

```
811         * hex and octal notations.</p>
812         *
813         * <p>Returns <code>null</code> if the string is <code>
                null</code>.</p>
814         *
815         * @param str  a <code>String</code> to convert, may be
                null
816         * @return converted <code>Integer</code> (or null if
                the input is null)
817         * @throws NumberFormatException if the value cannot be
                converted
818         */
819        public static Integer createInteger(final String str)
820        {
821            if (str == null)
822            {
823                return null;
824            }
825            // decode() handles 0xAABD and 0777 (hex and octal)
                as well.
826            return Integer.decode(str);
827        }
828
829        /**
830         * <p>Convert a <code>String</code> to a <code>Long</
                code>;
831         * since 3.1 it handles hex and octal notations.</p>
832         *
833         * <p>Returns <code>null</code> if the string is <code>
                null</code>.</p>
834         *
835         * @param str  a <code>String</code> to convert, may be
                null
836         * @return converted <code>Long</code> (or null if the
                input is null)
837         * @throws NumberFormatException if the value cannot be
                converted
838         */
839        public static Long createLong(final String str)
840        {
841            if (str == null)
842            {
843                return null;
844            }
845            return Long.decode(str);
846        }
```

```
847
848     /**
849      * <p>Convert a <code>String</code> to a <code>
            BigInteger</code>;
850      * since 3.2 it handles hex (0x or #) and octal (0)
            notations.</p>
851      *
852      * <p>Returns <code>null</code> if the string is <code>
            null</code>.</p>
853      *
854      * @param str  a <code>String</code> to convert, may be
             null
855      * @return converted <code>BigInteger</code> (or null
            if the input is null)
856      * @throws NumberFormatException if the value cannot be
             converted
857      */
858     public static BigInteger createBigInteger(final String
           str)
859     {
860         if (str == null)
861         {
862             return null;
863         }
864         int pos = 0; // offset within string
865         int radix = 10;
866         boolean negate = false; // need to negate later?
867         if (str.startsWith("-"))
868         {
869             negate = true;
870             pos = 1;
871         }
872         if (str.startsWith("0x", pos) || str.startsWith("0x
               ", pos))
873         {
874             // hex
875             radix = 16;
876             pos += 2;
877         }
878         else if (str.startsWith("#", pos))
879         {
880             // alternative hex (allowed by Long/Integer)
881             radix = 16;
882             pos ++;
883         }
884         else if (str.startsWith("0", pos) && str.length() >
```

```
            pos + 1)
885         {
886             // octal; so long as there are additional
                digits
887             radix = 8;
888             pos ++;
889         } // default is to treat as decimal
890
891         final BigInteger value = new BigInteger(str.
               substring(pos), radix);
892         return negate ? value.negate() : value;
893     }
894
895     /**
896      * <p>Convert a <code>String</code> to a <code>
            BigDecimal</code>.</p>
897      *
898      * <p>Returns <code>null</code> if the string is <code>
            null</code>.</p>
899      *
900      * @param str  a <code>String</code> to convert, may be
             null
901      * @return converted <code>BigDecimal</code> (or null
            if the input is null)
902      * @throws NumberFormatException if the value cannot be
             converted
903      */
904     public static BigDecimal createBigDecimal(final String
           str)
905     {
906         if (str == null)
907         {
908             return null;
909         }
910         // handle JDK1.3.1 bug where "" throws
               IndexOutOfBoundsException
911         if (StringUtils.isBlank(str))
912         {
913             throw new NumberFormatException("A blank string
                    is not a valid number");
914         }
915         if (str.trim().startsWith("--"))
916         {
917             // this is protection for poorness in java.lang
                   .BigDecimal.
918             // it accepts this as a legal value, but it
```

```
                does not appear
919             // to be in specification of class. OS X Java
                parses it to
920             // a wrong value.
921             throw new NumberFormatException(str + " is not
                a valid number.");
922         }
923         return new BigDecimal(str);
924     }
925
926     // Min in array
927     //
        ----------------------------------------------------------------

928     /**
929      * <p>Returns the minimum value in an array.</p>
930      *
931      * @param array  an array, must not be null or empty
932      * @return the minimum value in the array
933      * @throws IllegalArgumentException if <code>array</
         code> is <code>null</code>
934      * @throws IllegalArgumentException if <code>array</
         code> is empty
935      */
936     public static long min(final long[] array)
937     {
938         // Validates input
939         validateArray(array);
940
941         // Finds and returns min
942         long min = array[0];
943         for (int i = 1; i < array.length; i++)
944         {
945             if (array[i] < min)
946             {
947                 min = array[i];
948             }
949         }
950
951         return min;
952     }
953
954     /**
955      * <p>Returns the minimum value in an array.</p>
956      *
957      * @param array  an array, must not be null or empty
```

```
958         * @return the minimum value in the array
959         * @throws IllegalArgumentException if <code>array</
                code> is <code>null</code>
960         * @throws IllegalArgumentException if <code>array</
                code> is empty
961         */
962        public static int min(final int[] array)
963        {
964            // Validates input
965            validateArray(array);
966
967            // Finds and returns min
968            int min = array[0];
969            for (int j = 1; j < array.length; j++)
970            {
971                if (array[j] < min)
972                {
973                    min = array[j];
974                }
975            }
976
977            return min;
978        }
979
980        /**
981         * <p>Returns the minimum value in an array.</p>
982         *
983         * @param array  an array, must not be null or empty
984         * @return the minimum value in the array
985         * @throws IllegalArgumentException if <code>array</
                code> is <code>null</code>
986         * @throws IllegalArgumentException if <code>array</
                code> is empty
987         */
988        public static short min(final short[] array)
989        {
990            // Validates input
991            validateArray(array);
992
993            // Finds and returns min
994            short min = array[0];
995            for (int i = 1; i < array.length; i++)
996            {
997                if (array[i] < min)
998                {
999                    min = array[i];
```

```
1000            }
1001        }
1002
1003        return min;
1004    }
1005
1006    /**
1007     * <p>Returns the minimum value in an array.</p>
1008     *
1009     * @param array  an array, must not be null or empty
1010     * @return the minimum value in the array
1011     * @throws IllegalArgumentException if <code>array</
             code> is <code>null</code>
1012     * @throws IllegalArgumentException if <code>array</
             code> is empty
1013     */
1014    public static byte min(final byte[] array)
1015    {
1016        // Validates input
1017        validateArray(array);
1018
1019        // Finds and returns min
1020        byte min = array[0];
1021        for (int i = 1; i < array.length; i++)
1022        {
1023            if (array[i] < min)
1024            {
1025                min = array[i];
1026            }
1027        }
1028
1029        return min;
1030    }
1031
1032     /**
1033     * <p>Returns the minimum value in an array.</p>
1034     *
1035     * @param array  an array, must not be null or empty
1036     * @return the minimum value in the array
1037     * @throws IllegalArgumentException if <code>array</
             code> is <code>null</code>
1038     * @throws IllegalArgumentException if <code>array</
             code> is empty
1039     * @see IEEE754rUtils#min(double[]) IEEE754rUtils for a
              version of this method that handles NaN
1040     *  differently
```

```
1041        */
1042       public static double min(final double[] array)
1043       {
1044           // Validates input
1045           validateArray(array);
1046
1047           // Finds and returns min
1048           double min = array[0];
1049           for (int i = 1; i < array.length; i++)
1050           {
1051               if (Double.isNaN(array[i]))
1052               {
1053                   return Double.NaN;
1054               }
1055               if (array[i] < min)
1056               {
1057                   min = array[i];
1058               }
1059           }
1060
1061           return min;
1062       }
1063
1064       /**
1065        * <p>Returns the minimum value in an array.</p>
1066        *
1067        * @param array  an array, must not be null or empty
1068        * @return the minimum value in the array
1069        * @throws IllegalArgumentException if <code>array</
              code> is <code>null</code>
1070        * @throws IllegalArgumentException if <code>array</
              code> is empty
1071        * @see IEEE754rUtils#min(float[]) IEEE754rUtils for a
              version of this method that handles NaN
1072        *  differently
1073        */
1074       public static float min(final float[] array)
1075       {
1076           // Validates input
1077           validateArray(array);
1078
1079           // Finds and returns min
1080           float min = array[0];
1081           for (int i = 1; i < array.length; i++)
1082           {
1083               if (Float.isNaN(array[i]))
```

```
1084            {
1085                return Float.NaN;
1086            }
1087            if (array[i] < min)
1088            {
1089                min = array[i];
1090            }
1091        }
1092
1093        return min;
1094    }
1095
1096    // Max in array
1097    //
        ----------------------------------------------------------------------

1098    /**
1099     * <p>Returns the maximum value in an array.</p>
1100     *
1101     * @param array  an array, must not be null or empty
1102     * @return the minimum value in the array
1103     * @throws IllegalArgumentException if <code>array</
            code> is <code>null</code>
1104     * @throws IllegalArgumentException if <code>array</
            code> is empty
1105     */
1106    public static long max(final long[] array)
1107    {
1108        // Validates input
1109        validateArray(array);
1110
1111        // Finds and returns max
1112        long max = array[0];
1113        for (int j = 1; j < array.length; j++)
1114        {
1115            if (array[j] > max)
1116            {
1117                max = array[j];
1118            }
1119        }
1120
1121        return max;
1122    }
1123
1124    /**
1125     * <p>Returns the maximum value in an array.</p>
```

```
1126        *
1127        * @param array  an array, must not be null or empty
1128        * @return the minimum value in the array
1129        * @throws IllegalArgumentException if <code>array</
               code> is <code>null</code>
1130        * @throws IllegalArgumentException if <code>array</
               code> is empty
1131        */
1132       public static int max(final int[] array)
1133       {
1134           // Validates input
1135           validateArray(array);
1136
1137           // Finds and returns max
1138           int max = array[0];
1139           for (int j = 1; j < array.length; j++)
1140           {
1141               if (array[j] > max)
1142               {
1143                   max = array[j];
1144               }
1145           }
1146
1147           return max;
1148       }
1149
1150       /**
1151        * <p>Returns the maximum value in an array.</p>
1152        *
1153        * @param array  an array, must not be null or empty
1154        * @return the minimum value in the array
1155        * @throws IllegalArgumentException if <code>array</
               code> is <code>null</code>
1156        * @throws IllegalArgumentException if <code>array</
               code> is empty
1157        */
1158       public static short max(final short[] array)
1159       {
1160           // Validates input
1161           validateArray(array);
1162
1163           // Finds and returns max
1164           short max = array[0];
1165           for (int i = 1; i < array.length; i++)
1166           {
1167               if (array[i] > max)
```

```
1168            {
1169                max = array[i];
1170            }
1171        }
1172
1173        return max;
1174    }
1175
1176    /**
1177     * <p>Returns the maximum value in an array.</p>
1178     *
1179     * @param array  an array, must not be null or empty
1180     * @return the minimum value in the array
1181     * @throws IllegalArgumentException if <code>array</
          code> is <code>null</code>
1182     * @throws IllegalArgumentException if <code>array</
          code> is empty
1183     */
1184    public static byte max(final byte[] array)
1185    {
1186        // Validates input
1187        validateArray(array);
1188
1189        // Finds and returns max
1190        byte max = array[0];
1191        for (int i = 1; i < array.length; i++)
1192        {
1193            if (array[i] > max)
1194            {
1195                max = array[i];
1196            }
1197        }
1198
1199        return max;
1200    }
1201
1202    /**
1203     * <p>Returns the maximum value in an array.</p>
1204     *
1205     * @param array  an array, must not be null or empty
1206     * @return the minimum value in the array
1207     * @throws IllegalArgumentException if <code>array</
          code> is <code>null</code>
1208     * @throws IllegalArgumentException if <code>array</
          code> is empty
1209     * @see IEEE754rUtils#max(double[]) IEEE754rUtils for a
```

```
         version of this method that handles NaN
1210      *  differently
1211      */
1212     public static double max(final double[] array)
1213     {
1214         // Validates input
1215         validateArray(array);
1216 
1217         // Finds and returns max
1218         double max = array[0];
1219         for (int j = 1; j < array.length; j++)
1220         {
1221             if (Double.isNaN(array[j]))
1222             {
1223                 return Double.NaN;
1224             }
1225             if (array[j] > max)
1226             {
1227                 max = array[j];
1228             }
1229         }
1230 
1231         return max;
1232     }
1233 
1234     /**
1235      * <p>Returns the maximum value in an array.</p>
1236      *
1237      * @param array  an array, must not be null or empty
1238      * @return the minimum value in the array
1239      * @throws IllegalArgumentException if <code>array</
         code> is <code>null</code>
1240      * @throws IllegalArgumentException if <code>array</
         code> is empty
1241      * @see IEEE754rUtils#max(float[]) IEEE754rUtils for a
         version of this method that handles NaN
1242      *  differently
1243      */
1244     public static float max(final float[] array)
1245     {
1246         // Validates input
1247         validateArray(array);
1248 
1249         // Finds and returns max
1250         float max = array[0];
1251         for (int j = 1; j < array.length; j++)
```

```
1252            {
1253                if (Float.isNaN(array[j]))
1254                {
1255                    return Float.NaN;
1256                }
1257                if (array[j] > max)
1258                {
1259                    max = array[j];
1260                }
1261            }
1262
1263            return max;
1264        }
1265
1266        /**
1267         * Checks if the specified array is neither null nor
                empty.
1268         *
1269         * @param array  the array to check
1270         * @throws IllegalArgumentException if {@code array} is
                either {@code null} or empty
1271         */
1272        private static void validateArray(final Object array)
1273        {
1274            if (array == null)
1275            {
1276                throw new IllegalArgumentException("The Array
                       must not be null");
1277            }
1278            else if (Array.getLength(array) == 0)
1279            {
1280                throw new IllegalArgumentException("Array
                       cannot be empty.");
1281            }
1282        }
1283
1284        // 3 param min
1285        //
               ----------------------------------------------------------------

1286        /**
1287         * <p>Gets the minimum of three <code>long</code>
                values.</p>
1288         *
1289         * @param a  value 1
1290         * @param b  value 2
```

```
1291        * @param c  value 3
1292        * @return  the smallest of the values
1293        */
1294       public static long min(long a, final long b, final long
               c)
1295       {
1296           if (b < a)
1297           {
1298               a = b;
1299           }
1300           if (c < a)
1301           {
1302               a = c;
1303           }
1304           return a;
1305       }
1306
1307       /**
1308        * <p>Gets the minimum of three <code>int</code> values
               .</p>
1309        *
1310        * @param a  value 1
1311        * @param b  value 2
1312        * @param c  value 3
1313        * @return  the smallest of the values
1314        */
1315       public static int min(int a, final int b, final int c)
1316       {
1317           if (b < a)
1318           {
1319               a = b;
1320           }
1321           if (c < a)
1322           {
1323               a = c;
1324           }
1325           return a;
1326       }
1327
1328       /**
1329        * <p>Gets the minimum of three <code>short</code>
               values.</p>
1330        *
1331        * @param a  value 1
1332        * @param b  value 2
1333        * @param c  value 3
```

```
1334        * @return  the smallest of the values
1335        */
1336       public static short min(short a, final short b, final
              short c)
1337       {
1338           if (b < a)
1339           {
1340               a = b;
1341           }
1342           if (c < a)
1343           {
1344               a = c;
1345           }
1346           return a;
1347       }
1348
1349       /**
1350        * <p>Gets the minimum of three <code>byte</code>
              values.</p>
1351        *
1352        * @param a  value 1
1353        * @param b  value 2
1354        * @param c  value 3
1355        * @return  the smallest of the values
1356        */
1357       public static byte min(byte a, final byte b, final byte
              c)
1358       {
1359           if (b < a)
1360           {
1361               a = b;
1362           }
1363           if (c < a)
1364           {
1365               a = c;
1366           }
1367           return a;
1368       }
1369
1370       /**
1371        * <p>Gets the minimum of three <code>double</code>
              values.</p>
1372        *
1373        * <p>If any value is <code>NaN</code>, <code>NaN</code
              > is
1374        * returned. Infinity is handled.</p>
```

```
1375        *
1376        * @param a  value 1
1377        * @param b  value 2
1378        * @param c  value 3
1379        * @return  the smallest of the values
1380        * @see IEEE754rUtils#min(double, double, double) for a
                version of this method that handles NaN
1381        *  differently
1382        */
1383       public static double min(final double a, final double b
               , final double c)
1384       {
1385           return Math.min(Math.min(a, b), c);
1386       }
1387
1388       /**
1389        * <p>Gets the minimum of three <code>float</code>
               values.</p>
1390        *
1391        * <p>If any value is <code>NaN</code>, <code>NaN</code
               > is
1392        * returned. Infinity is handled.</p>
1393        *
1394        * @param a  value 1
1395        * @param b  value 2
1396        * @param c  value 3
1397        * @return  the smallest of the values
1398        * @see IEEE754rUtils#min(float, float, float) for a
               version of this method that handles NaN
1399        *  differently
1400        */
1401       public static float min(final float a, final float b,
              final float c)
1402       {
1403           return Math.min(Math.min(a, b), c);
1404       }
1405
1406       // 3 param max
1407       //
              -----------------------------------------------------------------

1408       /**
1409        * <p>Gets the maximum of three <code>long</code>
               values.</p>
1410        *
1411        * @param a  value 1
```

```
1412        * @param b  value 2
1413        * @param c  value 3
1414        * @return  the largest of the values
1415        */
1416       public static long max(long a, final long b, final long
               c)
1417       {
1418           if (b > a)
1419           {
1420               a = b;
1421           }
1422           if (c > a)
1423           {
1424               a = c;
1425           }
1426           return a;
1427       }
1428
1429       /**
1430        * <p>Gets the maximum of three <code>int</code> values
               .</p>
1431        *
1432        * @param a  value 1
1433        * @param b  value 2
1434        * @param c  value 3
1435        * @return  the largest of the values
1436        */
1437       public static int max(int a, final int b, final int c)
1438       {
1439           if (b > a)
1440           {
1441               a = b;
1442           }
1443           if (c > a)
1444           {
1445               a = c;
1446           }
1447           return a;
1448       }
1449
1450       /**
1451        * <p>Gets the maximum of three <code>short</code>
               values.</p>
1452        *
1453        * @param a  value 1
1454        * @param b  value 2
```

```
1455        * @param c  value 3
1456        * @return  the largest of the values
1457        */
1458       public static short max(short a, final short b, final
              short c)
1459       {
1460           if (b > a)
1461           {
1462               a = b;
1463           }
1464           if (c > a)
1465           {
1466               a = c;
1467           }
1468           return a;
1469       }
1470
1471       /**
1472        * <p>Gets the maximum of three <code>byte</code>
              values.</p>
1473        *
1474        * @param a  value 1
1475        * @param b  value 2
1476        * @param c  value 3
1477        * @return  the largest of the values
1478        */
1479       public static byte max(byte a, final byte b, final byte
              c)
1480       {
1481           if (b > a)
1482           {
1483               a = b;
1484           }
1485           if (c > a)
1486           {
1487               a = c;
1488           }
1489           return a;
1490       }
1491
1492       /**
1493        * <p>Gets the maximum of three <code>double</code>
              values.</p>
1494        *
1495        * <p>If any value is <code>NaN</code>, <code>NaN</code
              > is
```

```
1496      * returned. Infinity is handled.</p>
1497      *
1498      * @param a  value 1
1499      * @param b  value 2
1500      * @param c  value 3
1501      * @return  the largest of the values
1502      * @see IEEE754rUtils#max(double, double, double) for a
              version of this method that handles NaN
1503      *  differently
1504      */
1505     public static double max(final double a, final double b
          , final double c)
1506     {
1507         return Math.max(Math.max(a, b), c);
1508     }
1509
1510     /**
1511      * <p>Gets the maximum of three <code>float</code>
              values.</p>
1512      *
1513      * <p>If any value is <code>NaN</code>, <code>NaN</code
              > is
1514      * returned. Infinity is handled.</p>
1515      *
1516      * @param a  value 1
1517      * @param b  value 2
1518      * @param c  value 3
1519      * @return  the largest of the values
1520      * @see IEEE754rUtils#max(float, float, float) for a
              version of this method that handles NaN
1521      *  differently
1522      */
1523     public static float max(final float a, final float b,
          final float c)
1524     {
1525         return Math.max(Math.max(a, b), c);
1526     }
1527
1528     //
          -----------------------------------------------------------------------

1529     /**
1530      * <p>Checks whether the <code>String</code> contains
              only
1531      * digit characters.</p>
1532      *
```

```
1533        * <p><code>Null</code> and empty String will return
1534        * <code>false</code>.</p>
1535        *
1536        * @param str  the <code>String</code> to check
1537        * @return <code>true</code> if str contains only
               Unicode numeric
1538        */
1539       public static boolean isDigits(final String str)
1540       {
1541           if (StringUtils.isEmpty(str))
1542           {
1543               return false;
1544           }
1545           for (int i = 0; i < str.length(); i++)
1546           {
1547               if (!Character.isDigit(str.charAt(i)))
1548               {
1549                   return false;
1550               }
1551           }
1552           return true;
1553       }
1554
1555       /**
1556        * <p>Checks whether the String a valid Java number.</p
               >
1557        *
1558        * <p>Valid numbers include hexadecimal marked with the
                 <code>0x</code>
1559        * qualifier, scientific notation and numbers marked
               with a type
1560        * qualifier (e.g. 123L).</p>
1561        *
1562        * <p><code>Null</code> and empty String will return
1563        * <code>false</code>.</p>
1564        *
1565        * @param str  the <code>String</code> to check
1566        * @return <code>true</code> if the string is a
               correctly formatted number
1567        */
1568       public static boolean isNumber(final String str)
1569       {
1570           if (StringUtils.isEmpty(str))
1571           {
1572               return false;
1573           }
```

```
1574            final char[] chars = str.toCharArray();
1575            int sz = chars.length;
1576            boolean hasExp = false;
1577            boolean hasDecPoint = false;
1578            boolean allowSigns = false;
1579            boolean foundDigit = false;
1580            // deal with any possible sign up front
1581            final int start = (chars[0] == '-') ? 1 : 0;
1582            if (sz > start + 1 && chars[start] == '0' && chars[
                   start + 1] == 'x')
1583            {
1584                int i = start + 2;
1585                if (i == sz)
1586                {
1587                    return false; // str == "0x"
1588                }
1589                // checking hex (it can't be anything else)
1590                for (; i < chars.length; i++)
1591                {
1592                    if ((chars[i] < '0' || chars[i] > '9')
1593                        && (chars[i] < 'a' || chars[i] > 'f')
1594                        && (chars[i] < 'A' || chars[i] > 'F'))
1595                        {
1596                        return false;
1597                    }
1598                }
1599                return true;
1600            }
1601            sz--; // don't want to loop to the last char, check
                   it afterwords
1602                  // for type qualifiers
1603            int i = start;
1604            // loop to the next to last char or to the last
                   char if we need another digit to
1605            // make a valid number (e.g. chars[0..5] = "1234E")
1606            while (i < sz || (i < sz + 1 && allowSigns && !
                   foundDigit))
1607            {
1608                if (chars[i] >= '0' && chars[i] <= '9')
1609                {
1610                    foundDigit = true;
1611                    allowSigns = false;
1612
1613                }
1614                else if (chars[i] == '.')
1615                {
```

```
1616                if (hasDecPoint || hasExp)
1617                {
1618                    // two decimal points or dec in
                           exponent
1619                    return false;
1620                }
1621                hasDecPoint = true;
1622            }
1623            else if (chars[i] == 'e' || chars[i] == 'E')
1624            {
1625                // we've already taken care of hex.
1626                if (hasExp)
1627                {
1628                    // two E's
1629                    return false;
1630                }
1631                if (!foundDigit)
1632                {
1633                    return false;
1634                }
1635                hasExp = true;
1636                allowSigns = true;
1637            }
1638            else if (chars[i] == '+' || chars[i] == '-')
1639            {
1640                if (!allowSigns)
1641                {
1642                    return false;
1643                }
1644                allowSigns = false;
1645                foundDigit = false; // we need a digit
                       after the E
1646            }
1647            else
1648            {
1649                return false;
1650            }
1651            i++;
1652        }
1653        if (i < chars.length)
1654        {
1655            if (chars[i] >= '0' && chars[i] <= '9')
1656            {
1657                // no type qualifier, OK
1658                return true;
1659            }
```

```
1660            if (chars[i] == 'e' || chars[i] == 'E')
1661            {
1662                // can't have an E at the last byte
1663                return false;
1664            }
1665            if (chars[i] == '.')
1666            {
1667                if (hasDecPoint || hasExp)
1668                {
1669                    // two decimal points or dec in
                           exponent
1670                    return false;
1671                }
1672                // single trailing decimal point after non-
                       exponent is ok
1673                return foundDigit;
1674            }
1675            if (!allowSigns
1676                && (chars[i] == 'd'
1677                    || chars[i] == 'D'
1678                    || chars[i] == 'f'
1679                    || chars[i] == 'F'))
1680                    {
1681                return foundDigit;
1682            }
1683            if (chars[i] == 'l'
1684                || chars[i] == 'L')
1685                {
1686                // not allowing L with an exponent or
                       decimal point
1687                return foundDigit && !hasExp && !
                       hasDecPoint;
1688            }
1689            // last character is illegal
1690            return false;
1691        }
1692        // allowSigns is true iff the val ends in 'E'
1693        // found digit it to make sure weird stuff like '.'
               and '1E-' doesn't pass
1694        return !allowSigns && foundDigit;
1695    }
1696
1697 }
```

E.4 Manually Transformed Sources for Task Math-26

E.4.1 Defects-Sources/Math-26/Fraction-bugged-A.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under
       one or more
3   * contributor license agreements.  See the NOTICE file
       distributed
4   * with this work for additional information regarding
       copyright
5   * ownership. The ASF licenses this file to You under the
       Apache
6   * License, Version 2.0 (the "License"); you may not use
       this file
7   * except in compliance with the License.  You may obtain a
        copy of
8   * the License at
9   *
10  *      http://www.apache.org/licenses/LICENSE-2.0
11  *
12  * Unless required by applicable law or agreed to in
       writing, software
13  * distributed under the License is distributed on an "AS
       IS" BASIS,
14  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
       express or
15  * implied. See the License for the specific language
       governing
16  * permissions and limitations under the License.
17  */
18
19
20 package org.apache.commons.math3.fraction;
21
22
23 import java.io.Serializable;
24 import java.math.BigInteger;
25 import org.apache.commons.math3.FieldElement;
26 import org.apache.commons.math3.exception.util.
      LocalizedFormats;
27 import org.apache.commons.math3.exception.
      MathArithmeticException;
28 import org.apache.commons.math3.exception.
      NullArgumentException;
```

```
29 import org.apache.commons.math3.util.ArithmeticUtils;
30 import org.apache.commons.math3.util.FastMath;
31
32
33 /**
34  * Representation of a rational number.
35  *
36  * implements Serializable since 2.0
37  *
38  * @since 1.1
39  * @version $Id$
40  */
41 public class Fraction
42     extends Number
43     implements FieldElement<Fraction>, Comparable<Fraction
       >, Serializable {
44
45     /** A fraction representing "2 / 1". */
46     public static final Fraction TWO = new Fraction(2, 1);
47
48     /** A fraction representing "1". */
49     public static final Fraction ONE = new Fraction(1, 1);
50
51     /** A fraction representing "0". */
52     public static final Fraction ZERO = new Fraction(0, 1);
53
54     /** A fraction representing "4/5". */
55     public static final Fraction FOUR_FIFTHS = new Fraction
       (4, 5);
56
57     /** A fraction representing "1/5". */
58     public static final Fraction ONE_FIFTH = new Fraction
       (1, 5);
59
60     /** A fraction representing "1/2". */
61     public static final Fraction ONE_HALF = new Fraction(1,
        2);
62
63     /** A fraction representing "1/4". */
64     public static final Fraction ONE_QUARTER = new Fraction
       (1, 4);
65
66     /** A fraction representing "1/3". */
67     public static final Fraction ONE_THIRD = new Fraction
       (1, 3);
68
```

```
69      /** A fraction representing "3/5". */
70      public static final Fraction THREE_FIFTHS = new
           Fraction(3, 5);
71
72      /** A fraction representing "3/4". */
73      public static final Fraction THREE_QUARTERS = new
           Fraction(3, 4);
74
75      /** A fraction representing "2/5". */
76      public static final Fraction TWO_FIFTHS = new Fraction
           (2, 5);
77
78      /** A fraction representing "2/4". */
79      public static final Fraction TWO_QUARTERS = new
           Fraction(2, 4);
80
81      /** A fraction representing "2/3". */
82      public static final Fraction TWO_THIRDS = new Fraction
           (2, 3);
83
84      /** A fraction representing "-1 / 1". */
85      public static final Fraction MINUS_ONE = new Fraction
           (-1, 1);
86
87      /** Serializable version identifier */
88      private static final long serialVersionUID =
           3698073679419233275L;
89
90      /** The denominator. */
91      private final int denominator;
92
93      /** The numerator. */
94      private final int numerator;
95
96
97      /**
98       * Create a fraction given the double value.
99       * @param value the double value to convert to a
             fraction.
100      * @throws FractionConversionException if the continued
             fraction
101      *         failed to converge.
102      */
103     public Fraction(double value) throws
           FractionConversionException {
104        this(value, 1.0e-5, 100);
```

```
105     }
106
107     /**
108      * Create a fraction given the double value and maximum
             error
109      * allowed.
110      * <p>
111      * References:
112      * <ul>
113      * <li>
114      * <a href="http://mathworld.wolfram.com/
            ContinuedFraction.html">
115      * Continued Fraction</a> equations (11) and (22)-(26)
            </li>
116      * </ul>
117      * </p>
118      * @param value the double value to convert to a
            fraction.
119      * @param epsilon maximum error allowed.  The resulting
             fraction
120      *        is within {@code epsilon} of {@code value},
            in absolute
121      *        terms.
122      * @param maxIterations maximum number of convergents
123      * @throws FractionConversionException if the continued
             fraction
124      *         failed to converge.
125      */
126     public Fraction(double value, double epsilon, int
           maxIterations)
127         throws FractionConversionException {
128         this(value, epsilon, Integer.MAX_VALUE,
               maxIterations);
129     }
130
131     /**
132      * Create a fraction given the double value and maximum
133      * denominator.
134      * <p>
135      * References:
136      * <ul>
137      * <li>
138      * <a href="http://mathworld.wolfram.com/
            ContinuedFraction.html">
139      * Continued Fraction</a> equations (11) and (22)-(26)
            </li>
```

```
140         * </ul>
141         * </p>
142         * @param value the double value to convert to a
                fraction.
143         * @param maxDenominator The maximum allowed value for
                denominator
144         * @throws FractionConversionException if the continued
                fraction
145         *         failed to converge
146         */
147        public Fraction(double value, int maxDenominator)
148            throws FractionConversionException
149        {
150           this(value, 0, maxDenominator, 100);
151        }
152
153        /**
154         * Create a fraction given the double value and either
                the maximum
155         * error allowed or the maximum number of denominator
                digits.
156         * <p>
157         *
158         * NOTE: This constructor is called with EITHER
159         *   - a valid epsilon value and the maxDenominator set
                to
160         *     Integer.MAX_VALUE (that way the maxDenominator
                has no
161         *     effect).
162         * OR
163         *   - a valid maxDenominator value and the epsilon
                value set to
164         *     zero (that way epsilon only has effect if there
                is an exact
165         *     match before the maxDenominator value is reached
                ).
166         * </p><p>
167         *
168         * It has been done this way so that the same code can
                be (re)used
169         * for both scenarios. However this could be confusing
                to users if
170         * it were part of the public API and this constructor
                should
171         * therefore remain PRIVATE.
172         * </p>
```

```
173         *
174         * See JIRA issue ticket MATH-181 for more details:
175         *
176         *     https://issues.apache.org/jira/browse/MATH-181
177         *
178         * @param value the double value to convert to a
                fraction.
179         * @param epsilon maximum error allowed.  The resulting
                fraction
180         *        is within {@code epsilon} of {@code value},
                in absolute
181         *        terms.
182         * @param maxDenominator maximum denominator value
                allowed.
183         * @param maxIterations maximum number of convergents
184         * @throws FractionConversionException if the continued
                fraction
185         *         failed to converge.
186         */
187        private Fraction(double value, double epsilon, int
               maxDenominator,
188            int maxIterations) throws
                  FractionConversionException {
189            long overflow = Integer.MAX_VALUE;
190            double r0 = value;
191            long a0 = (long)FastMath.floor(r0);
192
193            if (a0 > overflow) {
194                throw new FractionConversionException(value, a0
                       , 1l);
195            }
196
197            // check for (almost) integer arguments, which
                   should not go
198            // to iterations.
199            if (FastMath.abs(a0 - value) < epsilon) {
200                this.numerator = (int) a0;
201                this.denominator = 1;
202                return;
203            }
204
205            long p0 = 1;
206            long q0 = 0;
207            long p1 = a0;
208            long q1 = 1;
209
```

```
210         long p2 = 0;
211         long q2 = 1;
212 
213         int n = 0;
214         boolean stop = false;
215         do {
216             ++n;
217             double r1 = 1.0 / (r0 - a0);
218             long a1 = (long)FastMath.floor(r1);
219             p2 = (a1 * p1) + p0;
220             q2 = (a1 * q1) + q0;
221             if ((p2 > overflow) || (q2 > overflow)) {
222                 throw new FractionConversionException(value
                        , p2, q2);
223             }
224 
225             double convergent = (double)p2 / (double)q2;
226             if (n < maxIterations
227                 && FastMath.abs(convergent - value) >
                        epsilon
228                 && q2 < maxDenominator) {
229 
230                 p0 = p1;
231                 p1 = p2;
232                 q0 = q1;
233                 q1 = q2;
234                 a0 = a1;
235                 r0 = r1;
236             } else {
237                 stop = true;
238             }
239         } while (!stop);
240 
241         if (n >= maxIterations) {
242             throw new FractionConversionException(value,
                    maxIterations);
243         }
244 
245         if (q2 < maxDenominator) {
246             this.numerator = (int) p2;
247             this.denominator = (int) q2;
248         } else {
249             this.numerator = (int) p1;
250             this.denominator = (int) q1;
251         }
252     }
```

```
253
254     /**
255      * Create a fraction from an int.
256      * The fraction is num / 1.
257      * @param num the numerator.
258      */
259     public Fraction(int num) {
260         this(num, 1);
261     }
262
263     /**
264      * Create a fraction given the numerator and
            denominator.  The
265      * fraction is reduced to lowest terms.
266      * @param num the numerator.
267      * @param den the denominator.
268      * @throws MathArithmeticException if the denominator
            is
269      *         {@code zero}
270      */
271     public Fraction(int num, int den) {
272         if (den == 0) {
273             throw new MathArithmeticException(
274                 LocalizedFormats.
                       ZERO_DENOMINATOR_IN_FRACTION, num, den);
275         }
276         if (den < 0) {
277             if (num == Integer.MIN_VALUE ||
278                 den == Integer.MIN_VALUE) {
279                 throw new MathArithmeticException(
280                     LocalizedFormats.OVERFLOW_IN_FRACTION,
                           num, den);
281             }
282             num = -num;
283             den = -den;
284         }
285
286         // reduce numerator and denominator by greatest
               common denominator.
287         final int d = ArithmeticUtils.gcd(num, den);
288         if (d > 1) {
289             num /= d;
290             den /= d;
291         }
292
293         // move sign to numerator.
```

```
294         if (den < 0) {
295             num = -num;
296             den = -den;
297         }
298         this.numerator   = num;
299         this.denominator = den;
300     }
301 
302     /**
303      * Returns the absolute value of this fraction.
304      * @return the absolute value.
305      */
306     public Fraction abs() {
307         Fraction ret;
308         if (numerator >= 0) {
309             ret = this;
310         } else {
311             ret = negate();
312         }
313         return ret;
314     }
315 
316     /**
317      * Compares this object to another based on size.
318      * @param object the object to compare to
319      * @return -1 if this is less than <tt>object</tt>, +1
         if this is
320      *         greater than <tt>object</tt>, 0 if they are
         equal.
321      */
322     public int compareTo(Fraction object) {
323         long nOd = ((long) numerator) * object.denominator;
324         long dOn = ((long) denominator) * object.numerator;
325         return (nOd < dOn) ? -1 : ((nOd > dOn) ? +1 : 0);
326     }
327 
328     /**
329      * Gets the fraction as a <tt>double</tt>. This
         calculates the
330      * fraction as the numerator divided by denominator.
331      * @return the fraction as a <tt>double</tt>
332      */
333     @Override
334     public double doubleValue() {
335         return (double)numerator / (double)denominator;
336     }
```

```
337
338     /**
339      * Test for the equality of two fractions.  If the
            lowest term
340      * numerator and denominators are the same for both
            fractions, the
341      * two fractions are considered to be equal.
342      * @param other fraction to test for equality to this
            fraction
343      * @return true if two fractions are equal, false if
            object is
344      *         <tt>null</tt>, not an instance of {@link
            Fraction}, or
345      *         not equal to this fraction instance.
346      */
347     @Override
348     public boolean equals(Object other) {
349         if (this == other) {
350             return true;
351         }
352         if (other instanceof Fraction) {
353
354             // since fractions are always in lowest terms,
                numerators and
355             // denominators can be compared directly for
                equality.
356             Fraction rhs = (Fraction)other;
357             return (numerator == rhs.numerator) &&
358                 (denominator == rhs.denominator);
359         }
360         return false;
361     }
362
363     /**
364      * Gets the fraction as a <tt>float</tt>. This
            calculates the
365      * fraction as the numerator divided by denominator.
366      * @return the fraction as a <tt>float</tt>
367      */
368     @Override
369     public float floatValue() {
370         return (float)doubleValue();
371     }
372
373     /**
374      * Access the denominator.
```

```
375         * @return the denominator.
376         */
377        public int getDenominator() {
378            return denominator;
379        }
380
381        /**
382         * Access the numerator.
383         * @return the numerator.
384         */
385        public int getNumerator() {
386            return numerator;
387        }
388
389        /**
390         * Gets a hashCode for the fraction.
391         * @return a hash code value for this object
392         */
393        @Override
394        public int hashCode() {
395            return 37 * (37 * 17 + numerator) + denominator;
396        }
397
398        /**
399         * Gets the fraction as an <tt>int</tt>. This returns
                the whole
400         * number part of the fraction.
401         * @return the whole number fraction part
402         */
403        @Override
404        public int intValue() {
405            return (int)doubleValue();
406        }
407
408        /**
409         * Gets the fraction as a <tt>long</tt>. This returns
                the whole
410         * number part of the fraction.
411         * @return the whole number fraction part
412         */
413        @Override
414        public long longValue() {
415            return (long)doubleValue();
416        }
417
418        /**
```

```
419         * Return the additive inverse of this fraction.
420         * @return the negation of this fraction.
421         */
422        public Fraction negate() {
423            if (numerator==Integer.MIN_VALUE) {
424                throw new MathArithmeticException(
425                        LocalizedFormats.OVERFLOW_IN_FRACTION,
426                        numerator, denominator);
427            }
428            return new Fraction(-numerator, denominator);
429        }
430
431        /**
432         * Return the multiplicative inverse of this fraction.
433         * @return the reciprocal fraction
434         */
435        public Fraction reciprocal() {
436            return new Fraction(denominator, numerator);
437        }
438
439        /**
440         * <p>Adds the value of this fraction to another,
              returning the
441         * result in reduced form. The algorithm follows Knuth,
               4.5.1.</p>
442         *
443         * @param fraction  the fraction to add, must not be {
              @code null}
444         * @return a {@code Fraction} instance with the
              resulting values
445         * @throws NullArgumentException if the fraction is {
              @code null}
446         * @throws MathArithmeticException if the resulting
              numerator or
447         *  denominator exceeds {@code Integer.MAX_VALUE}
448         */
449        public Fraction add(Fraction fraction) {
450            return addSub(fraction, true /* add */);
451        }
452
453        /**
454         * Add an integer to the fraction.
455         * @param i the <tt>integer</tt> to add.
456         * @return this + i
457         */
458        public Fraction add(final int i) {
```

```
459         return new Fraction(numerator + i * denominator,
                denominator);
460     }
461 
462     /**
463      * <p>Subtracts the value of another fraction from the
                value of
464      * this one, returning the result in reduced form.</p>
465      *
466      * @param fraction  the fraction to subtract, must not
                be
467      *        {@code null}
468      * @return a {@code Fraction} instance with the
                resulting values
469      * @throws NullArgumentException if the fraction is {
                @code null}
470      * @throws MathArithmeticException if the resulting
                numerator or
471      *   denominator cannot be represented in an {@code int
                }.
472      */
473     public Fraction subtract(Fraction fraction) {
474         return addSub(fraction, false /* subtract */);
475     }
476 
477     /**
478      * Subtract an integer from the fraction.
479      * @param i the <tt>integer</tt> to subtract.
480      * @return this - i
481      */
482     public Fraction subtract(final int i) {
483         return new Fraction(numerator - i * denominator,
                denominator);
484     }
485 
486     /**
487      * Implement add and subtract using algorithm described
                in
488      * Knuth 4.5.1.
489      *
490      * @param fraction the fraction to subtract, must not
                be
491      *        {@code null}
492      * @param isAdd true to add, false to subtract
493      * @return a {@code Fraction} instance with the
                resulting values
```

```
494         * @throws NullArgumentException if the fraction is {
                @code null}
495         * @throws MathArithmeticException if the resulting
                numerator or
496         *   denominator cannot be represented in an {@code int
                }.
497         */
498        private Fraction addSub(Fraction fraction, boolean
               isAdd) {
499            if (fraction == null) {
500                throw new NullArgumentException(
                       LocalizedFormats.FRACTION);
501            }
502
503            // zero is identity for addition.
504            if (numerator == 0) {
505                return isAdd ? fraction : fraction.negate();
506            }
507            if (fraction.numerator == 0) {
508                return this;
509            }
510
511            // if denominators are randomly distributed, d1
                   will be 1 about 61%
512            // of the time.
513            int d1 = ArithmeticUtils.gcd(denominator, fraction.
                   denominator);
514            if (d1==1) {
515
516                // result is ( (u*v' +/- u'v) / u'v')
517                int uvp = ArithmeticUtils.mulAndCheck(numerator
                       ,
518                        fraction.denominator);
519                int upv = ArithmeticUtils.mulAndCheck(fraction.
                       numerator,
520                        denominator);
521                return new Fraction
522                    (isAdd ? ArithmeticUtils.addAndCheck(uvp,
                           upv) :
523                     ArithmeticUtils.subAndCheck(uvp, upv),
524                     ArithmeticUtils.mulAndCheck(denominator,
525                             fraction.denominator));
526            }
527
528            // the quantity 't' requires 65 bits of precision;
                   see knuth 4.5.1
```

```
529         // exercise 7.  we're going to use a BigInteger.
530         // t = u(v'/d1) +/- v(u'/d1)
531         BigInteger uvp = BigInteger.valueOf(numerator)
532         .multiply(BigInteger.valueOf(fraction.denominator/
               d1));
533         BigInteger upv = BigInteger.valueOf(fraction.
               numerator)
534         .multiply(BigInteger.valueOf(denominator/d1));
535         BigInteger t = isAdd ? uvp.add(upv) : uvp.subtract(
               upv);
536
537         // but d2 doesn't need extra precision because
538         // d2 = gcd(t,d1) = gcd(t mod d1, d1)
539         int tmodd1 = t.mod(BigInteger.valueOf(d1)).intValue
               ();
540         int d2 = (tmodd1==0)?d1:ArithmeticUtils.gcd(tmodd1,
                d1);
541
542         // result is (t/d2) / (u'/d1)(v'/d2)
543         BigInteger w = t.divide(BigInteger.valueOf(d2));
544         if (w.bitLength() > 31) {
545             throw new MathArithmeticException(
546                     LocalizedFormats.
                           NUMERATOR_OVERFLOW_AFTER_MULTIPLY, w
                           );
547         }
548         return new Fraction (w.intValue(),
549                 ArithmeticUtils.mulAndCheck(denominator/d1,
550                         fraction.denominator/d2));
551     }
552
553     /**
554      * <p>Multiplies the value of this fraction by another,
                returning
555      * the result in reduced form.</p>
556      *
557      * @param fraction  the fraction to multiply by, must
               not be
558      *        {@code null}
559      * @return a {@code Fraction} instance with the
               resulting values
560      * @throws NullArgumentException if the fraction is {
               @code null}
561      * @throws MathArithmeticException if the resulting
               numerator or
562      *  denominator exceeds {@code Integer.MAX_VALUE}
```

```
563          */
564         public Fraction multiply(Fraction fraction) {
565             if (fraction == null) {
566                 throw new NullArgumentException(
                        LocalizedFormats.FRACTION);
567             }
568             if (numerator == 0 || fraction.numerator == 0) {
569                 return ZERO;
570             }
571 
572             // knuth 4.5.1
573             // make sure we don't overflow unless the result *
                    must* overflow.
574             int d1 = ArithmeticUtils.gcd(numerator, fraction.
                    denominator);
575             int d2 = ArithmeticUtils.gcd(fraction.numerator,
                    denominator);
576             return getReducedFraction
577             (ArithmeticUtils.mulAndCheck(numerator/d1, fraction
                    .numerator/d2),
578                     ArithmeticUtils.mulAndCheck(denominator/d2,
579                     fraction.denominator/d1));
580         }
581 
582         /**
583          * Multiply the fraction by an integer.
584          * @param i the <tt>integer</tt> to multiply by.
585          * @return this * i
586          */
587         public Fraction multiply(final int i) {
588             return new Fraction(numerator * i, denominator);
589         }
590 
591         /**
592          * <p>Divide the value of this fraction by another.</p>
593          *
594          * @param fraction  the fraction to divide by, must not
                    be
595          *        {@code null}
596          * @return a {@code Fraction} instance with the
                    resulting values
597          * @throws IllegalArgumentException if the fraction is
598          *         {@code null}
599          * @throws MathArithmeticException if the fraction to
                    divide by is
600          *         zero
```

```
601         * @throws MathArithmeticException if the resulting
                numerator or
602         *  denominator exceeds {@code Integer.MAX_VALUE}
603         */
604        public Fraction divide(Fraction fraction) {
605            if (fraction == null) {
606                throw new NullArgumentException(
                       LocalizedFormats.FRACTION);
607            }
608            if (fraction.numerator == 0) {
609                throw new MathArithmeticException(
610                        LocalizedFormats.
                               ZERO_FRACTION_TO_DIVIDE_BY,
611                        fraction.numerator, fraction.
                               denominator);
612            }
613            return multiply(fraction.reciprocal());
614        }
615
616        /**
617         * Divide the fraction by an integer.
618         * @param i the <tt>integer</tt> to divide by.
619         * @return this * i
620         */
621        public Fraction divide(final int i) {
622            return new Fraction(numerator, denominator * i);
623        }
624
625        /**
626         * <p>
627         * Gets the fraction percentage as a <tt>double</tt>.
                This
628         * calculates the fraction as the numerator divided by
629         * denominator multiplied by 100.
630         * </p>
631         *
632         * @return the fraction percentage as a <tt>double</tt
                >.
633         */
634        public double percentageValue() {
635            return 100 * doubleValue();
636        }
637
638        /**
639         * <p>Creates a {@code Fraction} instance with the 2
                parts
```

```
640         * of a fraction Y/Z.</p>
641         *
642         * <p>Any negative signs are resolved to be on the
                numerator.</p>
643         *
644         * @param numerator  the numerator, for example the
                three in
645         *        'three sevenths'
646         * @param denominator  the denominator, for example the
                 seven in
647         *        'three sevenths'
648         * @return a new fraction instance, with the numerator
                and
649         *         denominator reduced
650         * @throws MathArithmeticException if the denominator
                is
651         *         {@code zero}
652         */
653        public static Fraction getReducedFraction(int numerator
               , int denominator) {
654            if (denominator == 0) {
655                throw new MathArithmeticException(
656                        LocalizedFormats.
                           ZERO_DENOMINATOR_IN_FRACTION,
657                        numerator, denominator);
658            }
659            if (numerator==0) {
660                return ZERO; // normalize zero.
661            }
662
663            // allow 2^k/-2^31 as a valid fraction (where k>0)
664            if (denominator==Integer.MIN_VALUE && (numerator&1)
                   ==0) {
665                numerator/=2; denominator/=2;
666            }
667            if (denominator < 0) {
668                if (numerator==Integer.MIN_VALUE ||
669                        denominator==Integer.MIN_VALUE) {
670                    throw new MathArithmeticException(
671                            LocalizedFormats.
                               OVERFLOW_IN_FRACTION,
672                            numerator, denominator);
673                }
674                numerator = -numerator;
675                denominator = -denominator;
676            }
```

```
677 |
678 |         // simplify fraction.
679 |         int gcd = ArithmeticUtils.gcd(numerator,
            |            denominator);
680 |         numerator /= gcd;
681 |         denominator /= gcd;
682 |         return new Fraction(numerator, denominator);
683 |     }
684 |
685 |     /**
686 |      * <p>
687 |      * Returns the {@code String} representing this
            |         fraction, ie
688 |      * "num / dem" or just "num" if the denominator is one.
689 |      * </p>
690 |      *
691 |      * @return a string representation of the fraction.
692 |      * @see java.lang.Object#toString()
693 |      */
694 |     @Override
695 |     public String toString() {
696 |         String str = null;
697 |         if (denominator == 1) {
698 |             str = Integer.toString(numerator);
699 |         } else if (numerator == 0) {
700 |             str = "0";
701 |         } else {
702 |             str = numerator + " / " + denominator;
703 |         }
704 |         return str;
705 |     }
706 |
707 |     /** {@inheritDoc} */
708 |     public FractionField getField() {
709 |         return FractionField.getInstance();
710 |     }
711 | }
```

E.4.2 Defects-Sources/Math-26/Fraction-fixed-A.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under
       one or more
3   * contributor license agreements.  See the NOTICE file
       distributed
4   * with this work for additional information regarding
       copyright
5   * ownership. The ASF licenses this file to You under the
       Apache
6   * License, Version 2.0 (the "License"); you may not use
       this file
7   * except in compliance with the License.  You may obtain a
        copy of
8   * the License at
9   *
10  *      http://www.apache.org/licenses/LICENSE-2.0
11  *
12  * Unless required by applicable law or agreed to in
       writing, software
13  * distributed under the License is distributed on an "AS
       IS" BASIS,
14  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
       express or
15  * implied. See the License for the specific language
       governing
16  * permissions and limitations under the License.
17  */
18
19
20 package org.apache.commons.math3.fraction;
21
22
23 import java.io.Serializable;
24 import java.math.BigInteger;
25 import org.apache.commons.math3.FieldElement;
26 import org.apache.commons.math3.exception.util.
      LocalizedFormats;
27 import org.apache.commons.math3.exception.
      MathArithmeticException;
28 import org.apache.commons.math3.exception.
      NullArgumentException;
29 import org.apache.commons.math3.util.ArithmeticUtils;
30 import org.apache.commons.math3.util.FastMath;
31
```

```
32
33 /**
34  * Representation of a rational number.
35  *
36  * implements Serializable since 2.0
37  *
38  * @since 1.1
39  * @version $Id$
40  */
41 public class Fraction
42     extends Number
43     implements FieldElement<Fraction>, Comparable<Fraction
       >, Serializable {
44
45     /** A fraction representing "2 / 1". */
46     public static final Fraction TWO = new Fraction(2, 1);
47
48     /** A fraction representing "1". */
49     public static final Fraction ONE = new Fraction(1, 1);
50
51     /** A fraction representing "0". */
52     public static final Fraction ZERO = new Fraction(0, 1);
53
54     /** A fraction representing "4/5". */
55     public static final Fraction FOUR_FIFTHS = new Fraction
       (4, 5);
56
57     /** A fraction representing "1/5". */
58     public static final Fraction ONE_FIFTH = new Fraction
       (1, 5);
59
60     /** A fraction representing "1/2". */
61     public static final Fraction ONE_HALF = new Fraction(1,
        2);
62
63     /** A fraction representing "1/4". */
64     public static final Fraction ONE_QUARTER = new Fraction
       (1, 4);
65
66     /** A fraction representing "1/3". */
67     public static final Fraction ONE_THIRD = new Fraction
       (1, 3);
68
69     /** A fraction representing "3/5". */
70     public static final Fraction THREE_FIFTHS = new
       Fraction(3, 5);
```

```
71
72      /** A fraction representing "3/4". */
73      public static final Fraction THREE_QUARTERS = new
           Fraction(3, 4);
74
75      /** A fraction representing "2/5". */
76      public static final Fraction TWO_FIFTHS = new Fraction
           (2, 5);
77
78      /** A fraction representing "2/4". */
79      public static final Fraction TWO_QUARTERS = new
           Fraction(2, 4);
80
81      /** A fraction representing "2/3". */
82      public static final Fraction TWO_THIRDS = new Fraction
           (2, 3);
83
84      /** A fraction representing "-1 / 1". */
85      public static final Fraction MINUS_ONE = new Fraction
           (-1, 1);
86
87      /** Serializable version identifier */
88      private static final long serialVersionUID =
           369807367941923327 5L;
89
90      /** The denominator. */
91      private final int denominator;
92
93      /** The numerator. */
94      private final int numerator;
95
96
97      /**
98       * Create a fraction given the double value.
99       * @param value the double value to convert to a
            fraction.
100      * @throws FractionConversionException if the continued
            fraction
101      *         failed to converge.
102      */
103     public Fraction(double value) throws
           FractionConversionException {
104         this(value, 1.0e-5, 100);
105     }
106
107     /**
```

```
108         * Create a fraction given the double value and maximum
                error
109         * allowed.
110         * <p>
111         * References:
112         * <ul>
113         * <li>
114         * <a href="http://mathworld.wolfram.com/
                ContinuedFraction.html">
115         * Continued Fraction</a> equations (11) and (22)-(26)
                </li>
116         * </ul>
117         * </p>
118         * @param value the double value to convert to a
                fraction.
119         * @param epsilon maximum error allowed.  The resulting
                fraction
120         *        is within {@code epsilon} of {@code value},
                in absolute
121         *        terms.
122         * @param maxIterations maximum number of convergents
123         * @throws FractionConversionException if the continued
                fraction
124         *         failed to converge.
125         */
126        public Fraction(double value, double epsilon, int
               maxIterations)
127            throws FractionConversionException {
128            this(value, epsilon, Integer.MAX_VALUE,
                   maxIterations);
129        }
130
131        /**
132         * Create a fraction given the double value and maximum
133         * denominator.
134         * <p>
135         * References:
136         * <ul>
137         * <li>
138         * <a href="http://mathworld.wolfram.com/
                ContinuedFraction.html">
139         * Continued Fraction</a> equations (11) and (22)-(26)
                </li>
140         * </ul>
141         * </p>
142         * @param value the double value to convert to a
```

```
         fraction.
143      * @param maxDenominator The maximum allowed value for
             denominator
144      * @throws FractionConversionException if the continued
             fraction
145      *         failed to converge
146      */
147     public Fraction(double value, int maxDenominator)
148         throws FractionConversionException
149     {
150        this(value, 0, maxDenominator, 100);
151     }
152
153     /**
154      * Create a fraction given the double value and either
             the maximum
155      * error allowed or the maximum number of denominator
             digits.
156      * <p>
157      *
158      * NOTE: This constructor is called with EITHER
159      *   - a valid epsilon value and the maxDenominator set
             to
160      *     Integer.MAX_VALUE (that way the maxDenominator
             has no
161      *     effect).
162      * OR
163      *   - a valid maxDenominator value and the epsilon
             value set to
164      *     zero (that way epsilon only has effect if there
             is an exact
165      *     match before the maxDenominator value is reached
             ).
166      * </p><p>
167      *
168      * It has been done this way so that the same code can
             be (re)used
169      * for both scenarios. However this could be confusing
             to users if
170      * it were part of the public API and this constructor
             should
171      * therefore remain PRIVATE.
172      * </p>
173      *
174      * See JIRA issue ticket MATH-181 for more details:
175      *
```

```
176         *     https://issues.apache.org/jira/browse/MATH-181
177         *
178         * @param value the double value to convert to a
                fraction.
179         * @param epsilon maximum error allowed.  The resulting
                fraction
180         *        is within {@code epsilon} of {@code value},
                in absolute
181         *        terms.
182         * @param maxDenominator maximum denominator value
                allowed.
183         * @param maxIterations maximum number of convergents
184         * @throws FractionConversionException if the continued
                fraction
185         *         failed to converge.
186         */
187        private Fraction(double value, double epsilon, int
               maxDenominator,
188            int maxIterations) throws
                  FractionConversionException {
189            long overflow = Integer.MAX_VALUE;
190            double r0 = value;
191            long a0 = (long)FastMath.floor(r0);
192
193            if (FastMath.abs(a0) > overflow) {
194                throw new FractionConversionException(value, a0
                       , 1l);
195            }
196
197            // check for (almost) integer arguments, which
                  should not go
198            // to iterations.
199            if (FastMath.abs(a0 - value) < epsilon) {
200                this.numerator = (int) a0;
201                this.denominator = 1;
202                return;
203            }
204
205            long p0 = 1;
206            long q0 = 0;
207            long p1 = a0;
208            long q1 = 1;
209
210            long p2 = 0;
211            long q2 = 1;
212
```

```
213         int n = 0;
214         boolean stop = false;
215         do {
216             ++n;
217             double r1 = 1.0 / (r0 - a0);
218             long a1 = (long)FastMath.floor(r1);
219             p2 = (a1 * p1) + p0;
220             q2 = (a1 * q1) + q0;
221             if ((FastMath.abs(p2) > overflow)
222                 || (FastMath.abs(q2) > overflow)) {
223                 throw new FractionConversionException(value
                        , p2, q2);
224             }
225 
226             double convergent = (double)p2 / (double)q2;
227             if (n < maxIterations
228                 && FastMath.abs(convergent - value) >
                        epsilon
229                 && q2 < maxDenominator) {
230 
231                 p0 = p1;
232                 p1 = p2;
233                 q0 = q1;
234                 q1 = q2;
235                 a0 = a1;
236                 r0 = r1;
237             } else {
238                 stop = true;
239             }
240         } while (!stop);
241 
242         if (n >= maxIterations) {
243             throw new FractionConversionException(value,
                    maxIterations);
244         }
245 
246         if (q2 < maxDenominator) {
247             this.numerator = (int) p2;
248             this.denominator = (int) q2;
249         } else {
250             this.numerator = (int) p1;
251             this.denominator = (int) q1;
252         }
253     }
254 
255     /**
```

```
256         * Create a fraction from an int.
257         * The fraction is num / 1.
258         * @param num the numerator.
259         */
260        public Fraction(int num) {
261            this(num, 1);
262        }
263
264        /**
265         * Create a fraction given the numerator and
               denominator.  The
266         * fraction is reduced to lowest terms.
267         * @param num the numerator.
268         * @param den the denominator.
269         * @throws MathArithmeticException if the denominator
               is
270         *         {@code zero}
271         */
272        public Fraction(int num, int den) {
273            if (den == 0) {
274                throw new MathArithmeticException(
275                    LocalizedFormats.
                           ZERO_DENOMINATOR_IN_FRACTION, num, den);
276            }
277            if (den < 0) {
278                if (num == Integer.MIN_VALUE ||
279                    den == Integer.MIN_VALUE) {
280                    throw new MathArithmeticException(
281                        LocalizedFormats.OVERFLOW_IN_FRACTION,
                               num, den);
282                }
283                num = -num;
284                den = -den;
285            }
286
287            // reduce numerator and denominator by greatest
                   common denominator.
288            final int d = ArithmeticUtils.gcd(num, den);
289            if (d > 1) {
290                num /= d;
291                den /= d;
292            }
293
294            // move sign to numerator.
295            if (den < 0) {
296                num = -num;
```

```
297             den = -den;
298         }
299         this.numerator   = num;
300         this.denominator = den;
301     }
302 
303     /**
304      * Returns the absolute value of this fraction.
305      * @return the absolute value.
306      */
307     public Fraction abs() {
308         Fraction ret;
309         if (numerator >= 0) {
310             ret = this;
311         } else {
312             ret = negate();
313         }
314         return ret;
315     }
316 
317     /**
318      * Compares this object to another based on size.
319      * @param object the object to compare to
320      * @return -1 if this is less than <tt>object</tt>, +1
         if this is
321      *          greater than <tt>object</tt>, 0 if they are
         equal.
322      */
323     public int compareTo(Fraction object) {
324         long nOd = ((long) numerator) * object.denominator;
325         long dOn = ((long) denominator) * object.numerator;
326         return (nOd < dOn) ? -1 : ((nOd > dOn) ? +1 : 0);
327     }
328 
329     /**
330      * Gets the fraction as a <tt>double</tt>. This
         calculates the
331      * fraction as the numerator divided by denominator.
332      * @return the fraction as a <tt>double</tt>
333      */
334     @Override
335     public double doubleValue() {
336         return (double)numerator / (double)denominator;
337     }
338 
339     /**
```

```
340         * Test for the equality of two fractions.  If the
                lowest term
341         * numerator and denominators are the same for both
                fractions, the
342         * two fractions are considered to be equal.
343         * @param other fraction to test for equality to this
                fraction
344         * @return true if two fractions are equal, false if
                object is
345         *         <tt>null</tt>, not an instance of {@link
                Fraction}, or
346         *         not equal to this fraction instance.
347         */
348        @Override
349        public boolean equals(Object other) {
350            if (this == other) {
351                return true;
352            }
353            if (other instanceof Fraction) {
354
355                // since fractions are always in lowest terms,
                        numerators and
356                // denominators can be compared directly for
                        equality.
357                Fraction rhs = (Fraction)other;
358                return (numerator == rhs.numerator) &&
359                    (denominator == rhs.denominator);
360            }
361            return false;
362        }
363
364        /**
365         * Gets the fraction as a <tt>float</tt>. This
                calculates the
366         * fraction as the numerator divided by denominator.
367         * @return the fraction as a <tt>float</tt>
368         */
369        @Override
370        public float floatValue() {
371            return (float)doubleValue();
372        }
373
374        /**
375         * Access the denominator.
376         * @return the denominator.
377         */
```

```
378     public int getDenominator() {
379         return denominator;
380     }
381 
382     /**
383      * Access the numerator.
384      * @return the numerator.
385      */
386     public int getNumerator() {
387         return numerator;
388     }
389 
390     /**
391      * Gets a hashCode for the fraction.
392      * @return a hash code value for this object
393      */
394     @Override
395     public int hashCode() {
396         return 37 * (37 * 17 + numerator) + denominator;
397     }
398 
399     /**
400      * Gets the fraction as an <tt>int</tt>. This returns
         the whole
401      * number part of the fraction.
402      * @return the whole number fraction part
403      */
404     @Override
405     public int intValue() {
406         return (int)doubleValue();
407     }
408 
409     /**
410      * Gets the fraction as a <tt>long</tt>. This returns
         the whole
411      * number part of the fraction.
412      * @return the whole number fraction part
413      */
414     @Override
415     public long longValue() {
416         return (long)doubleValue();
417     }
418 
419     /**
420      * Return the additive inverse of this fraction.
421      * @return the negation of this fraction.
```

```
422         */
423         public Fraction negate() {
424             if (numerator==Integer.MIN_VALUE) {
425                 throw new MathArithmeticException(
426                         LocalizedFormats.OVERFLOW_IN_FRACTION,
427                         numerator, denominator);
428             }
429             return new Fraction(-numerator, denominator);
430         }
431 
432         /**
433          * Return the multiplicative inverse of this fraction.
434          * @return the reciprocal fraction
435          */
436         public Fraction reciprocal() {
437             return new Fraction(denominator, numerator);
438         }
439 
440         /**
441          * <p>Adds the value of this fraction to another,
                returning the
442          * result in reduced form. The algorithm follows Knuth,
                4.5.1.</p>
443          *
444          * @param fraction  the fraction to add, must not be {
                @code null}
445          * @return a {@code Fraction} instance with the
                resulting values
446          * @throws NullArgumentException if the fraction is {
                @code null}
447          * @throws MathArithmeticException if the resulting
                numerator or
448          *  denominator exceeds {@code Integer.MAX_VALUE}
449          */
450         public Fraction add(Fraction fraction) {
451             return addSub(fraction, true /* add */);
452         }
453 
454         /**
455          * Add an integer to the fraction.
456          * @param i the <tt>integer</tt> to add.
457          * @return this + i
458          */
459         public Fraction add(final int i) {
460             return new Fraction(numerator + i * denominator,
                    denominator);
```

```
461     }
462
463     /**
464      * <p>Subtracts the value of another fraction from the
            value of
465      * this one, returning the result in reduced form.</p>
466      *
467      * @param fraction  the fraction to subtract, must not
            be
468      *        {@code null}
469      * @return a {@code Fraction} instance with the
            resulting values
470      * @throws NullArgumentException if the fraction is {
            @code null}
471      * @throws MathArithmeticException if the resulting
            numerator or
472      *   denominator cannot be represented in an {@code int
            }.
473      */
474     public Fraction subtract(Fraction fraction) {
475         return addSub(fraction, false /* subtract */);
476     }
477
478     /**
479      * Subtract an integer from the fraction.
480      * @param i the <tt>integer</tt> to subtract.
481      * @return this - i
482      */
483     public Fraction subtract(final int i) {
484         return new Fraction(numerator - i * denominator,
               denominator);
485     }
486
487     /**
488      * Implement add and subtract using algorithm described
            in
489      * Knuth 4.5.1.
490      *
491      * @param fraction the fraction to subtract, must not
            be
492      *        {@code null}
493      * @param isAdd true to add, false to subtract
494      * @return a {@code Fraction} instance with the
            resulting values
495      * @throws NullArgumentException if the fraction is {
            @code null}
```

```
496         * @throws MathArithmeticException if the resulting
                numerator or
497         *   denominator cannot be represented in an {@code int
                }.
498         */
499        private Fraction addSub(Fraction fraction, boolean
               isAdd) {
500            if (fraction == null) {
501                throw new NullArgumentException(
                       LocalizedFormats.FRACTION);
502            }
503
504            // zero is identity for addition.
505            if (numerator == 0) {
506                return isAdd ? fraction : fraction.negate();
507            }
508            if (fraction.numerator == 0) {
509                return this;
510            }
511
512            // if denominators are randomly distributed, d1
                   will be 1 about 61%
513            // of the time.
514            int d1 = ArithmeticUtils.gcd(denominator, fraction.
                   denominator);
515            if (d1==1) {
516
517                // result is ( (u*v' +/- u'v) / u'v')
518                int uvp = ArithmeticUtils.mulAndCheck(numerator
                       ,
519                        fraction.denominator);
520                int upv = ArithmeticUtils.mulAndCheck(fraction.
                       numerator,
521                        denominator);
522                return new Fraction
523                    (isAdd ? ArithmeticUtils.addAndCheck(uvp,
                           upv) :
524                     ArithmeticUtils.subAndCheck(uvp, upv),
525                     ArithmeticUtils.mulAndCheck(denominator,
526                            fraction.denominator));
527            }
528
529            // the quantity 't' requires 65 bits of precision;
                   see knuth 4.5.1
530            // exercise 7.  we're going to use a BigInteger.
531            // t = u(v'/d1) +/- v(u'/d1)
```

```
532             BigInteger uvp = BigInteger.valueOf(numerator)
533             .multiply(BigInteger.valueOf(fraction.denominator/
                    d1));
534             BigInteger upv = BigInteger.valueOf(fraction.
                    numerator)
535             .multiply(BigInteger.valueOf(denominator/d1));
536             BigInteger t = isAdd ? uvp.add(upv) : uvp.subtract(
                    upv);
537
538             // but d2 doesn't need extra precision because
539             // d2 = gcd(t,d1) = gcd(t mod d1, d1)
540             int tmodd1 = t.mod(BigInteger.valueOf(d1)).intValue
                    ();
541             int d2 = (tmodd1==0)?d1:ArithmeticUtils.gcd(tmodd1,
                     d1);
542
543             // result is (t/d2) / (u'/d1)(v'/d2)
544             BigInteger w = t.divide(BigInteger.valueOf(d2));
545             if (w.bitLength() > 31) {
546                 throw new MathArithmeticException(
547                         LocalizedFormats.
                            NUMERATOR_OVERFLOW_AFTER_MULTIPLY, w
                            );
548             }
549             return new Fraction (w.intValue(),
550                     ArithmeticUtils.mulAndCheck(denominator/d1,
551                             fraction.denominator/d2));
552         }
553
554         /**
555          * <p>Multiplies the value of this fraction by another,
                 returning
556          * the result in reduced form.</p>
557          *
558          * @param fraction  the fraction to multiply by, must
                not be
559          *        {@code null}
560          * @return a {@code Fraction} instance with the
                resulting values
561          * @throws NullArgumentException if the fraction is {
                @code null}
562          * @throws MathArithmeticException if the resulting
                numerator or
563          *  denominator exceeds {@code Integer.MAX_VALUE}
564          */
565         public Fraction multiply(Fraction fraction) {
```

```
566         if (fraction == null) {
567             throw new NullArgumentException(
                    LocalizedFormats.FRACTION);
568         }
569         if (numerator == 0 || fraction.numerator == 0) {
570             return ZERO;
571         }
572 
573         // knuth 4.5.1
574         // make sure we don't overflow unless the result *
                must* overflow.
575         int d1 = ArithmeticUtils.gcd(numerator, fraction.
                denominator);
576         int d2 = ArithmeticUtils.gcd(fraction.numerator,
                denominator);
577         return getReducedFraction
578         (ArithmeticUtils.mulAndCheck(numerator/d1, fraction
                .numerator/d2),
579                 ArithmeticUtils.mulAndCheck(denominator/d2,
580                 fraction.denominator/d1));
581     }
582 
583     /**
584      * Multiply the fraction by an integer.
585      * @param i the <tt>integer</tt> to multiply by.
586      * @return this * i
587      */
588     public Fraction multiply(final int i) {
589         return new Fraction(numerator * i, denominator);
590     }
591 
592     /**
593      * <p>Divide the value of this fraction by another.</p>
594      *
595      * @param fraction  the fraction to divide by, must not
             be
596      *        {@code null}
597      * @return a {@code Fraction} instance with the
             resulting values
598      * @throws IllegalArgumentException if the fraction is
599      *         {@code null}
600      * @throws MathArithmeticException if the fraction to
             divide by is
601      *         zero
602      * @throws MathArithmeticException if the resulting
             numerator or
```

```
603         *  denominator exceeds {@code Integer.MAX_VALUE}
604         */
605        public Fraction divide(Fraction fraction) {
606            if (fraction == null) {
607                throw new NullArgumentException(
                       LocalizedFormats.FRACTION);
608            }
609            if (fraction.numerator == 0) {
610                throw new MathArithmeticException(
611                        LocalizedFormats.
                               ZERO_FRACTION_TO_DIVIDE_BY,
612                        fraction.numerator, fraction.
                               denominator);
613            }
614            return multiply(fraction.reciprocal());
615        }
616
617        /**
618         * Divide the fraction by an integer.
619         * @param i the <tt>integer</tt> to divide by.
620         * @return this * i
621         */
622        public Fraction divide(final int i) {
623            return new Fraction(numerator, denominator * i);
624        }
625
626        /**
627         * <p>
628         * Gets the fraction percentage as a <tt>double</tt>.
               This
629         * calculates the fraction as the numerator divided by
630         * denominator multiplied by 100.
631         * </p>
632         *
633         * @return the fraction percentage as a <tt>double</tt
               >.
634         */
635        public double percentageValue() {
636            return 100 * doubleValue();
637        }
638
639        /**
640         * <p>Creates a {@code Fraction} instance with the 2
               parts
641         * of a fraction Y/Z.</p>
642         *
```

```
643         * <p>Any negative signs are resolved to be on the
                numerator.</p>
644         *
645         * @param numerator  the numerator, for example the
                three in
646         *        'three sevenths'
647         * @param denominator  the denominator, for example the
                 seven in
648         *        'three sevenths'
649         * @return a new fraction instance, with the numerator
                and
650         *         denominator reduced
651         * @throws MathArithmeticException if the denominator
                is
652         *         {@code zero}
653         */
654        public static Fraction getReducedFraction(int numerator
               , int denominator) {
655            if (denominator == 0) {
656                throw new MathArithmeticException(
657                        LocalizedFormats.
                           ZERO_DENOMINATOR_IN_FRACTION,
658                        numerator, denominator);
659            }
660            if (numerator==0) {
661                return ZERO; // normalize zero.
662            }
663
664            // allow 2^k/-2^31 as a valid fraction (where k>0)
665            if (denominator==Integer.MIN_VALUE && (numerator&1)
                   ==0) {
666                numerator/=2; denominator/=2;
667            }
668            if (denominator < 0) {
669                if (numerator==Integer.MIN_VALUE ||
670                        denominator==Integer.MIN_VALUE) {
671                    throw new MathArithmeticException(
672                            LocalizedFormats.
                               OVERFLOW_IN_FRACTION,
673                            numerator, denominator);
674                }
675                numerator = -numerator;
676                denominator = -denominator;
677            }
678
679            // simplify fraction.
```

```
680             int gcd = ArithmeticUtils.gcd(numerator,
                    denominator);
681             numerator /= gcd;
682             denominator /= gcd;
683             return new Fraction(numerator, denominator);
684         }
685 
686         /**
687          * <p>
688          * Returns the {@code String} representing this
                    fraction, ie
689          * "num / dem" or just "num" if the denominator is one.
690          * </p>
691          *
692          * @return a string representation of the fraction.
693          * @see java.lang.Object#toString()
694          */
695         @Override
696         public String toString() {
697             String str = null;
698             if (denominator == 1) {
699                 str = Integer.toString(numerator);
700             } else if (numerator == 0) {
701                 str = "0";
702             } else {
703                 str = numerator + " / " + denominator;
704             }
705             return str;
706         }
707 
708         /** {@inheritDoc} */
709         public FractionField getField() {
710             return FractionField.getInstance();
711         }
712 }
```

E.4.3 Defects-Sources/Math-26/Fraction-bugged-B.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under
       one or more
3   * contributor license agreements.  See the NOTICE file
       distributed with
4   * this work for additional information regarding copyright
        ownership.
5   * The ASF licenses this file to You under the Apache
       License, Version 2.0
6   * (the "License"); you may not use this file except in
       compliance with
7   * the License.  You may obtain a copy of the License at
8   *
9   *      http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in
       writing, software
12  * distributed under the License is distributed on an "AS
       IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
       express or implied.
14  * See the License for the specific language governing
       permissions and
15  * limitations under the License.
16  */
17 package org.apache.commons.math3.fraction;
18
19 import java.io.Serializable;
20 import java.math.BigInteger;
21
22 import org.apache.commons.math3.FieldElement;
23 import org.apache.commons.math3.exception.util.
      LocalizedFormats;
24 import org.apache.commons.math3.exception.
      MathArithmeticException;
25 import org.apache.commons.math3.exception.
      NullArgumentException;
26 import org.apache.commons.math3.util.ArithmeticUtils;
27 import org.apache.commons.math3.util.FastMath;
28
29 /**
30  * Representation of a rational number.
31  *
32  * implements Serializable since 2.0
```

```
33  *
34  * @since 1.1
35  * @version $Id$
36  */
37 public class Fraction
38     extends Number
39     implements FieldElement<Fraction>, Comparable<Fraction
       >, Serializable
40 {
41
42     /** A fraction representing "2 / 1". */
43     public static final Fraction TWO = new Fraction(2, 1);
44
45     /** A fraction representing "1". */
46     public static final Fraction ONE = new Fraction(1, 1);
47
48     /** A fraction representing "0". */
49     public static final Fraction ZERO = new Fraction(0, 1);
50
51     /** A fraction representing "4/5". */
52     public static final Fraction FOUR_FIFTHS = new Fraction
       (4, 5);
53
54     /** A fraction representing "1/5". */
55     public static final Fraction ONE_FIFTH = new Fraction
       (1, 5);
56
57     /** A fraction representing "1/2". */
58     public static final Fraction ONE_HALF = new Fraction(1,
        2);
59
60     /** A fraction representing "1/4". */
61     public static final Fraction ONE_QUARTER = new Fraction
       (1, 4);
62
63     /** A fraction representing "1/3". */
64     public static final Fraction ONE_THIRD = new Fraction
       (1, 3);
65
66     /** A fraction representing "3/5". */
67     public static final Fraction THREE_FIFTHS = new
       Fraction(3, 5);
68
69     /** A fraction representing "3/4". */
70     public static final Fraction THREE_QUARTERS = new
       Fraction(3, 4);
```

```
71
72      /** A fraction representing "2/5". */
73      public static final Fraction TWO_FIFTHS = new Fraction
           (2, 5);
74
75      /** A fraction representing "2/4". */
76      public static final Fraction TWO_QUARTERS = new
           Fraction(2, 4);
77
78      /** A fraction representing "2/3". */
79      public static final Fraction TWO_THIRDS = new Fraction
           (2, 3);
80
81      /** A fraction representing "-1 / 1". */
82      public static final Fraction MINUS_ONE = new Fraction
           (-1, 1);
83
84      /** Serializable version identifier */
85      private static final long serialVersionUID =
           3698073679419233275L;
86
87      /** The denominator. */
88      private final int denominator;
89
90      /** The numerator. */
91      private final int numerator;
92
93      /**
94       * Create a fraction given the double value.
95       * @param value the double value to convert to a
            fraction.
96       * @throws FractionConversionException if the continued
            fraction failed to
97       *         converge.
98       */
99      public Fraction(double value) throws
           FractionConversionException
100     {
101         this(value, 1.0e-5, 100);
102     }
103
104     /**
105      * Create a fraction given the double value and maximum
            error allowed.
106      * <p>
107      * References:
```

```
108         * <ul>
109         * <li><a href="http://mathworld.wolfram.com/
                ContinuedFraction.html">
110         * Continued Fraction</a> equations (11) and (22)-(26)
                </li>
111         * </ul>
112         * </p>
113         * @param value the double value to convert to a
                fraction.
114         * @param epsilon maximum error allowed.  The resulting
                fraction is within
115         *        {@code epsilon} of {@code value}, in absolute
                terms.
116         * @param maxIterations maximum number of convergents
117         * @throws FractionConversionException if the continued
                fraction failed to
118         *         converge.
119         */
120        public Fraction(double value, double epsilon, int
               maxIterations)
121            throws FractionConversionException
122        {
123            this(value, epsilon, Integer.MAX_VALUE,
                   maxIterations);
124        }
125
126        /**
127         * Create a fraction given the double value and maximum
                denominator.
128         * <p>
129         * References:
130         * <ul>
131         * <li><a href="http://mathworld.wolfram.com/
                ContinuedFraction.html">
132         * Continued Fraction</a> equations (11) and (22)-(26)
                </li>
133         * </ul>
134         * </p>
135         * @param value the double value to convert to a
                fraction.
136         * @param maxDenominator The maximum allowed value for
                denominator
137         * @throws FractionConversionException if the continued
                fraction failed to
138         *         converge
139         */
```

```
140     public Fraction(double value, int maxDenominator)
141         throws FractionConversionException
142     {
143        this(value, 0, maxDenominator, 100);
144     }
145 
146     /**
147      * Create a fraction given the double value and either
           the maximum error
148      * allowed or the maximum number of denominator digits.
149      * <p>
150      *
151      * NOTE: This constructor is called with EITHER
152      *   - a valid epsilon value and the maxDenominator set
           to Integer.MAX_VALUE
153      *     (that way the maxDenominator has no effect).
154      * OR
155      *   - a valid maxDenominator value and the epsilon
           value set to zero
156      *     (that way epsilon only has effect if there is an
           exact match before
157      *     the maxDenominator value is reached).
158      * </p><p>
159      *
160      * It has been done this way so that the same code can
           be (re)used for both
161      * scenarios. However this could be confusing to users
           if it were part of
162      * the public API and this constructor should therefore
           remain PRIVATE.
163      * </p>
164      *
165      * See JIRA issue ticket MATH-181 for more details:
166      *
167      *     https://issues.apache.org/jira/browse/MATH-181
168      *
169      * @param value the double value to convert to a
           fraction.
170      * @param epsilon maximum error allowed.  The resulting
           fraction is within
171      *        {@code epsilon} of {@code value}, in absolute
           terms.
172      * @param maxDenominator maximum denominator value
           allowed.
173      * @param maxIterations maximum number of convergents
174      * @throws FractionConversionException if the continued
```

```
         fraction failed to
175       *          converge.
176       */
177      private Fraction(double value, double epsilon, int
            maxDenominator, int maxIterations)
178          throws FractionConversionException
179      {
180          long overflow = Integer.MAX_VALUE;
181          double r0 = value;
182          long a0 = (long)FastMath.floor(r0);
183          if (a0 > overflow)
184          {
185              throw new FractionConversionException(value, a0
                    , 1l);
186          }
187
188          // check for (almost) integer arguments, which
                should not go
189          // to iterations.
190          if (FastMath.abs(a0 - value) < epsilon)
191          {
192              this.numerator = (int) a0;
193              this.denominator = 1;
194              return;
195          }
196
197          long p0 = 1;
198          long q0 = 0;
199          long p1 = a0;
200          long q1 = 1;
201
202          long p2 = 0;
203          long q2 = 1;
204
205          int n = 0;
206          boolean stop = false;
207          do
208          {
209              ++n;
210              double r1 = 1.0 / (r0 - a0);
211              long a1 = (long)FastMath.floor(r1);
212              p2 = (a1 * p1) + p0;
213              q2 = (a1 * q1) + q0;
214              if ((p2 > overflow) || (q2 > overflow))
215              {
216                  throw new FractionConversionException(value
```

```
                    , p2, q2);
            }

            double convergent = (double)p2 / (double)q2;
            if (n < maxIterations
                && FastMath.abs(convergent - value) >
                   epsilon
                && q2 < maxDenominator)
                {
                p0 = p1;
                p1 = p2;
                q0 = q1;
                q1 = q2;
                a0 = a1;
                r0 = r1;
            }
            else
            {
                stop = true;
            }
        } while (!stop);

        if (n >= maxIterations)
        {
            throw new FractionConversionException(value,
               maxIterations);
        }

        if (q2 < maxDenominator)
        {
            this.numerator = (int) p2;
            this.denominator = (int) q2;
        }
        else
        {
            this.numerator = (int) p1;
            this.denominator = (int) q1;
        }

    }

    /**
     * Create a fraction from an int.
     * The fraction is num / 1.
     * @param num the numerator.
     */
```

```
260     public Fraction(int num)
261     {
262         this(num, 1);
263     }
264 
265     /**
266      * Create a fraction given the numerator and
           denominator.  The fraction is
267      * reduced to lowest terms.
268      * @param num the numerator.
269      * @param den the denominator.
270      * @throws MathArithmeticException if the denominator
           is {@code zero}
271      */
272     public Fraction(int num, int den)
273     {
274         if (den == 0)
275         {
276             throw new MathArithmeticException(
                   LocalizedFormats.
                   ZERO_DENOMINATOR_IN_FRACTION,
277                                               num, den);
278         }
279         if (den < 0)
280         {
281             if (num == Integer.MIN_VALUE ||
282                 den == Integer.MIN_VALUE)
283                 {
284                 throw new MathArithmeticException(
                       LocalizedFormats.OVERFLOW_IN_FRACTION,
285                                                     num, den)
                                                        ;
286             }
287             num = -num;
288             den = -den;
289         }
290         // reduce numerator and denominator by greatest
               common denominator.
291         final int d = ArithmeticUtils.gcd(num, den);
292         if (d > 1)
293         {
294             num /= d;
295             den /= d;
296         }
297 
298         // move sign to numerator.
```

```
299         if (den < 0)
300         {
301             num = -num;
302             den = -den;
303         }
304         this.numerator   = num;
305         this.denominator = den;
306     }
307 
308     /**
309      * Returns the absolute value of this fraction.
310      * @return the absolute value.
311      */
312     public Fraction abs()
313     {
314         Fraction ret;
315         if (numerator >= 0)
316         {
317             ret = this;
318         }
319         else
320         {
321             ret = negate();
322         }
323         return ret;
324     }
325 
326     /**
327      * Compares this object to another based on size.
328      * @param object the object to compare to
329      * @return -1 if this is less than <tt>object</tt>, +1
           if this is greater
330      *         than <tt>object</tt>, 0 if they are equal.
331      */
332     public int compareTo(Fraction object)
333     {
334         long nOd = ((long) numerator) * object.denominator;
335         long dOn = ((long) denominator) * object.numerator;
336         return (nOd < dOn) ? -1 : ((nOd > dOn) ? +1 : 0);
337     }
338 
339     /**
340      * Gets the fraction as a <tt>double</tt>. This
           calculates the fraction as
341      * the numerator divided by denominator.
342      * @return the fraction as a <tt>double</tt>
```

```
343         */
344        @Override
345        public double doubleValue()
346        {
347            return (double)numerator / (double)denominator;
348        }
349
350        /**
351         * Test for the equality of two fractions.  If the
               lowest term
352         * numerator and denominators are the same for both
               fractions, the two
353         * fractions are considered to be equal.
354         * @param other fraction to test for equality to this
               fraction
355         * @return true if two fractions are equal, false if
               object is
356         *         <tt>null</tt>, not an instance of {@link
               Fraction}, or not equal
357         *         to this fraction instance.
358         */
359        @Override
360        public boolean equals(Object other)
361        {
362            if (this == other)
363            {
364                return true;
365            }
366            if (other instanceof Fraction)
367            {
368                // since fractions are always in lowest terms,
                       numerators and
369                // denominators can be compared directly for
                       equality.
370                Fraction rhs = (Fraction)other;
371                return (numerator == rhs.numerator) &&
372                    (denominator == rhs.denominator);
373            }
374            return false;
375        }
376
377        /**
378         * Gets the fraction as a <tt>float</tt>. This
               calculates the fraction as
379         * the numerator divided by denominator.
380         * @return the fraction as a <tt>float</tt>
```

```
381         */
382        @Override
383        public float floatValue()
384        {
385            return (float)doubleValue();
386        }
387
388        /**
389         * Access the denominator.
390         * @return the denominator.
391         */
392        public int getDenominator()
393        {
394            return denominator;
395        }
396
397        /**
398         * Access the numerator.
399         * @return the numerator.
400         */
401        public int getNumerator()
402        {
403            return numerator;
404        }
405
406        /**
407         * Gets a hashCode for the fraction.
408         * @return a hash code value for this object
409         */
410        @Override
411        public int hashCode()
412        {
413            return 37 * (37 * 17 + numerator) + denominator;
414        }
415
416        /**
417         * Gets the fraction as an <tt>int</tt>. This returns
                the whole number part
418         * of the fraction.
419         * @return the whole number fraction part
420         */
421        @Override
422        public int intValue()
423        {
424            return (int)doubleValue();
425        }
```

```
426
427     /**
428      * Gets the fraction as a <tt>long</tt>. This returns
            the whole number part
429      * of the fraction.
430      * @return the whole number fraction part
431      */
432     @Override
433     public long longValue()
434     {
435         return (long)doubleValue();
436     }
437
438     /**
439      * Return the additive inverse of this fraction.
440      * @return the negation of this fraction.
441      */
442     public Fraction negate()
443     {
444         if (numerator==Integer.MIN_VALUE)
445         {
446             throw new MathArithmeticException(
447                     LocalizedFormats.OVERFLOW_IN_FRACTION,
                            numerator, denominator);
448         }
449         return new Fraction(-numerator, denominator);
450     }
451
452     /**
453      * Return the multiplicative inverse of this fraction.
454      * @return the reciprocal fraction
455      */
456     public Fraction reciprocal()
457     {
458         return new Fraction(denominator, numerator);
459     }
460
461     /**
462      * <p>Adds the value of this fraction to another,
            returning the result in reduced form.
463      * The algorithm follows Knuth, 4.5.1.</p>
464      *
465      * @param fraction  the fraction to add, must not be {
            @code null}
466      * @return a {@code Fraction} instance with the
            resulting values
```

```
467         * @throws NullArgumentException if the fraction is {
                @code null}
468         * @throws MathArithmeticException if the resulting
                numerator or denominator exceeds
469         *  {@code Integer.MAX_VALUE}
470         */
471        public Fraction add(Fraction fraction)
472        {
473            return addSub(fraction, true /* add */);
474        }
475
476        /**
477         * Add an integer to the fraction.
478         * @param i the <tt>integer</tt> to add.
479         * @return this + i
480         */
481        public Fraction add(final int i)
482        {
483            return new Fraction(numerator + i * denominator,
                    denominator);
484        }
485
486        /**
487         * <p>Subtracts the value of another fraction from the
                value of this one,
488         * returning the result in reduced form.</p>
489         *
490         * @param fraction  the fraction to subtract, must not
                be {@code null}
491         * @return a {@code Fraction} instance with the
                resulting values
492         * @throws NullArgumentException if the fraction is {
                @code null}
493         * @throws MathArithmeticException if the resulting
                numerator or denominator
494         *   cannot be represented in an {@code int}.
495         */
496        public Fraction subtract(Fraction fraction)
497        {
498            return addSub(fraction, false /* subtract */);
499        }
500
501        /**
502         * Subtract an integer from the fraction.
503         * @param i the <tt>integer</tt> to subtract.
504         * @return this - i
```

```
505         */
506        public Fraction subtract(final int i)
507        {
508            return new Fraction(numerator - i * denominator,
                   denominator);
509        }
510
511        /**
512         * Implement add and subtract using algorithm described
                in Knuth 4.5.1.
513         *
514         * @param fraction the fraction to subtract, must not
                be {@code null}
515         * @param isAdd true to add, false to subtract
516         * @return a {@code Fraction} instance with the
                resulting values
517         * @throws NullArgumentException if the fraction is {
                @code null}
518         * @throws MathArithmeticException if the resulting
                numerator or denominator
519         *   cannot be represented in an {@code int}.
520         */
521        private Fraction addSub(Fraction fraction, boolean
               isAdd)
522        {
523            if (fraction == null)
524            {
525                throw new NullArgumentException(
                       LocalizedFormats.FRACTION);
526            }
527            // zero is identity for addition.
528            if (numerator == 0)
529            {
530                return isAdd ? fraction : fraction.negate();
531            }
532            if (fraction.numerator == 0)
533            {
534                return this;
535            }
536            // if denominators are randomly distributed, d1
                   will be 1 about 61%
537            // of the time.
538            int d1 = ArithmeticUtils.gcd(denominator, fraction.
                   denominator);
539            if (d1==1)
540            {
```

```
541             // result is ( (u*v' +/- u'v) / u'v')
542             int uvp = ArithmeticUtils.mulAndCheck(numerator
                    , fraction.denominator);
543             int upv = ArithmeticUtils.mulAndCheck(fraction.
                    numerator, denominator);
544             return new Fraction
545                 (isAdd ? ArithmeticUtils.addAndCheck(uvp,
                        upv) :
546                  ArithmeticUtils.subAndCheck(uvp, upv),
547                  ArithmeticUtils.mulAndCheck(denominator,
                        fraction.denominator));
548         }
549         // the quantity 't' requires 65 bits of precision;
                see knuth 4.5.1
550         // exercise 7.  we're going to use a BigInteger.
551         // t = u(v'/d1) +/- v(u'/d1)
552         BigInteger uvp = BigInteger.valueOf(numerator)
553         .multiply(BigInteger.valueOf(fraction.denominator/
                d1));
554         BigInteger upv = BigInteger.valueOf(fraction.
                numerator)
555         .multiply(BigInteger.valueOf(denominator/d1));
556         BigInteger t = isAdd ? uvp.add(upv) : uvp.subtract(
                upv);
557         // but d2 doesn't need extra precision because
558         // d2 = gcd(t,d1) = gcd(t mod d1, d1)
559         int tmodd1 = t.mod(BigInteger.valueOf(d1)).intValue
                ();
560         int d2 = (tmodd1==0)?d1:ArithmeticUtils.gcd(tmodd1,
                d1);
561
562         // result is (t/d2) / (u'/d1)(v'/d2)
563         BigInteger w = t.divide(BigInteger.valueOf(d2));
564         if (w.bitLength() > 31)
565         {
566             throw new MathArithmeticException(
                        LocalizedFormats.
                        NUMERATOR_OVERFLOW_AFTER_MULTIPLY,
567                                               w);
568         }
569         return new Fraction (w.intValue(),
570                 ArithmeticUtils.mulAndCheck(denominator/d1,
571                         fraction.denominator/d2));
572     }
573
574     /**
```

```
575         * <p>Multiplies the value of this fraction by another,
                returning the
576         * result in reduced form.</p>
577         *
578         * @param fraction  the fraction to multiply by, must
                not be {@code null}
579         * @return a {@code Fraction} instance with the
                resulting values
580         * @throws NullArgumentException if the fraction is {
                @code null}
581         * @throws MathArithmeticException if the resulting
                numerator or denominator exceeds
582         *  {@code Integer.MAX_VALUE}
583         */
584        public Fraction multiply(Fraction fraction)
585        {
586            if (fraction == null)
587            {
588                throw new NullArgumentException(
                       LocalizedFormats.FRACTION);
589            }
590            if (numerator == 0 || fraction.numerator == 0)
591            {
592                return ZERO;
593            }
594            // knuth 4.5.1
595            // make sure we don't overflow unless the result *
                   must* overflow.
596            int d1 = ArithmeticUtils.gcd(numerator, fraction.
                   denominator);
597            int d2 = ArithmeticUtils.gcd(fraction.numerator,
                   denominator);
598            return getReducedFraction
599            (ArithmeticUtils.mulAndCheck(numerator/d1, fraction
                   .numerator/d2),
600                    ArithmeticUtils.mulAndCheck(denominator/d2,
                           fraction.denominator/d1));
601        }
602
603        /**
604         * Multiply the fraction by an integer.
605         * @param i the <tt>integer</tt> to multiply by.
606         * @return this * i
607         */
608        public Fraction multiply(final int i)
609        {
```

```
610         return new Fraction(numerator * i, denominator);
611     }
612 
613     /**
614      * <p>Divide the value of this fraction by another.</p>
615      *
616      * @param fraction  the fraction to divide by, must not
               be {@code null}
617      * @return a {@code Fraction} instance with the
              resulting values
618      * @throws IllegalArgumentException if the fraction is
              {@code null}
619      * @throws MathArithmeticException if the fraction to
              divide by is zero
620      * @throws MathArithmeticException if the resulting
              numerator or denominator exceeds
621      *  {@code Integer.MAX_VALUE}
622      */
623     public Fraction divide(Fraction fraction)
624     {
625         if (fraction == null)
626         {
627             throw new NullArgumentException(
                    LocalizedFormats.FRACTION);
628         }
629         if (fraction.numerator == 0)
630         {
631             throw new MathArithmeticException(
                    LocalizedFormats.ZERO_FRACTION_TO_DIVIDE_BY,
632                                               fraction.
                                                  numerator,
                                                   fraction.
                                                  denominator
                                                  );
633         }
634         return multiply(fraction.reciprocal());
635     }
636 
637     /**
638      * Divide the fraction by an integer.
639      * @param i the <tt>integer</tt> to divide by.
640      * @return this * i
641      */
642     public Fraction divide(final int i)
643     {
644         return new Fraction(numerator, denominator * i);
```

```
645     }
646
647     /**
648      * <p>
649      * Gets the fraction percentage as a <tt>double</tt>.
           This calculates the
650      * fraction as the numerator divided by denominator
           multiplied by 100.
651      * </p>
652      *
653      * @return the fraction percentage as a <tt>double</tt
           >.
654      */
655     public double percentageValue()
656     {
657         return 100 * doubleValue();
658     }
659
660     /**
661      * <p>Creates a {@code Fraction} instance with the 2
           parts
662      * of a fraction Y/Z.</p>
663      *
664      * <p>Any negative signs are resolved to be on the
           numerator.</p>
665      *
666      * @param numerator  the numerator, for example the
           three in 'three sevenths'
667      * @param denominator  the denominator, for example the
            seven in 'three sevenths'
668      * @return a new fraction instance, with the numerator
           and denominator reduced
669      * @throws MathArithmeticException if the denominator
           is {@code zero}
670      */
671     public static Fraction getReducedFraction(int numerator
           , int denominator)
672     {
673         if (denominator == 0)
674         {
675             throw new MathArithmeticException(
                       LocalizedFormats.
                       ZERO_DENOMINATOR_IN_FRACTION,
676                                                 numerator,
                                                       denominator
                                                       );
```

```
677         }
678         if (numerator==0)
679         {
680             return ZERO; // normalize zero.
681         }
682         // allow 2^k/-2^31 as a valid fraction (where k>0)
683         if (denominator==Integer.MIN_VALUE && (numerator&1)
                ==0)
684         {
685             numerator/=2; denominator/=2;
686         }
687         if (denominator < 0)
688         {
689             if (numerator==Integer.MIN_VALUE ||
690                     denominator==Integer.MIN_VALUE)
691             {
692                 throw new MathArithmeticException(
                        LocalizedFormats.OVERFLOW_IN_FRACTION,
693                                                   numerator
                                                      ,
                                                      denominator
                                                      );
694             }
695             numerator = -numerator;
696             denominator = -denominator;
697         }
698         // simplify fraction.
699         int gcd = ArithmeticUtils.gcd(numerator,
                denominator);
700         numerator /= gcd;
701         denominator /= gcd;
702         return new Fraction(numerator, denominator);
703     }
704
705     /**
706      * <p>
707      * Returns the {@code String} representing this
             fraction, ie
708      * "num / dem" or just "num" if the denominator is one.
709      * </p>
710      *
711      * @return a string representation of the fraction.
712      * @see java.lang.Object#toString()
713      */
714     @Override
715     public String toString()
```

```
716     {
717         String str = null;
718         if (denominator == 1)
719         {
720             str = Integer.toString(numerator);
721         }
722         else if (numerator == 0)
723         {
724             str = "0";
725         }
726         else
727         {
728             str = numerator + " / " + denominator;
729         }
730         return str;
731     }
732 
733     /** {@inheritDoc} */
734     public FractionField getField()
735     {
736         return FractionField.getInstance();
737     }
738 
739 }
```

E.4.4 Defects-Sources/Math-26/Fraction-fixed-B.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under
       one or more
3   * contributor license agreements.  See the NOTICE file
       distributed with
4   * this work for additional information regarding copyright
        ownership.
5   * The ASF licenses this file to You under the Apache
       License, Version 2.0
6   * (the "License"); you may not use this file except in
       compliance with
7   * the License.  You may obtain a copy of the License at
8   *
9   *      http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in
       writing, software
12  * distributed under the License is distributed on an "AS
       IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
       express or implied.
14  * See the License for the specific language governing
       permissions and
15  * limitations under the License.
16  */
17 package org.apache.commons.math3.fraction;
18
19 import java.io.Serializable;
20 import java.math.BigInteger;
21
22 import org.apache.commons.math3.FieldElement;
23 import org.apache.commons.math3.exception.util.
      LocalizedFormats;
24 import org.apache.commons.math3.exception.
      MathArithmeticException;
25 import org.apache.commons.math3.exception.
      NullArgumentException;
26 import org.apache.commons.math3.util.ArithmeticUtils;
27 import org.apache.commons.math3.util.FastMath;
28
29 /**
30  * Representation of a rational number.
31  *
32  * implements Serializable since 2.0
```

```
33  *
34  * @since 1.1
35  * @version $Id$
36  */
37 public class Fraction
38     extends Number
39     implements FieldElement<Fraction>, Comparable<Fraction
       >, Serializable
40 {
41
42     /** A fraction representing "2 / 1". */
43     public static final Fraction TWO = new Fraction(2, 1);
44
45     /** A fraction representing "1". */
46     public static final Fraction ONE = new Fraction(1, 1);
47
48     /** A fraction representing "0". */
49     public static final Fraction ZERO = new Fraction(0, 1);
50
51     /** A fraction representing "4/5". */
52     public static final Fraction FOUR_FIFTHS = new Fraction
       (4, 5);
53
54     /** A fraction representing "1/5". */
55     public static final Fraction ONE_FIFTH = new Fraction
       (1, 5);
56
57     /** A fraction representing "1/2". */
58     public static final Fraction ONE_HALF = new Fraction(1,
        2);
59
60     /** A fraction representing "1/4". */
61     public static final Fraction ONE_QUARTER = new Fraction
       (1, 4);
62
63     /** A fraction representing "1/3". */
64     public static final Fraction ONE_THIRD = new Fraction
       (1, 3);
65
66     /** A fraction representing "3/5". */
67     public static final Fraction THREE_FIFTHS = new
       Fraction(3, 5);
68
69     /** A fraction representing "3/4". */
70     public static final Fraction THREE_QUARTERS = new
       Fraction(3, 4);
```

```
71
72      /** A fraction representing "2/5". */
73      public static final Fraction TWO_FIFTHS = new Fraction
           (2, 5);
74
75      /** A fraction representing "2/4". */
76      public static final Fraction TWO_QUARTERS = new
           Fraction(2, 4);
77
78      /** A fraction representing "2/3". */
79      public static final Fraction TWO_THIRDS = new Fraction
           (2, 3);
80
81      /** A fraction representing "-1 / 1". */
82      public static final Fraction MINUS_ONE = new Fraction
           (-1, 1);
83
84      /** Serializable version identifier */
85      private static final long serialVersionUID =
           3698073679419233275L;
86
87      /** The denominator. */
88      private final int denominator;
89
90      /** The numerator. */
91      private final int numerator;
92
93      /**
94       * Create a fraction given the double value.
95       * @param value the double value to convert to a
            fraction.
96       * @throws FractionConversionException if the continued
            fraction failed to
97       *         converge.
98       */
99      public Fraction(double value) throws
           FractionConversionException
100     {
101         this(value, 1.0e-5, 100);
102     }
103
104     /**
105      * Create a fraction given the double value and maximum
            error allowed.
106      * <p>
107      * References:
```

```
108         * <ul>
109         * <li><a href="http://mathworld.wolfram.com/
                ContinuedFraction.html">
110         * Continued Fraction</a> equations (11) and (22)-(26)
                </li>
111         * </ul>
112         * </p>
113         * @param value the double value to convert to a
                fraction.
114         * @param epsilon maximum error allowed.  The resulting
                fraction is within
115         *        {@code epsilon} of {@code value}, in absolute
                terms.
116         * @param maxIterations maximum number of convergents
117         * @throws FractionConversionException if the continued
                fraction failed to
118         *         converge.
119         */
120        public Fraction(double value, double epsilon, int
               maxIterations)
121            throws FractionConversionException
122        {
123            this(value, epsilon, Integer.MAX_VALUE,
                   maxIterations);
124        }
125
126        /**
127         * Create a fraction given the double value and maximum
                denominator.
128         * <p>
129         * References:
130         * <ul>
131         * <li><a href="http://mathworld.wolfram.com/
                ContinuedFraction.html">
132         * Continued Fraction</a> equations (11) and (22)-(26)
                </li>
133         * </ul>
134         * </p>
135         * @param value the double value to convert to a
                fraction.
136         * @param maxDenominator The maximum allowed value for
                denominator
137         * @throws FractionConversionException if the continued
                fraction failed to
138         *         converge
139         */
```

```
140     public Fraction(double value, int maxDenominator)
141         throws FractionConversionException
142     {
143        this(value, 0, maxDenominator, 100);
144     }
145 
146     /**
147      * Create a fraction given the double value and either
           the maximum error
148      * allowed or the maximum number of denominator digits.
149      * <p>
150      *
151      * NOTE: This constructor is called with EITHER
152      *   - a valid epsilon value and the maxDenominator set
           to Integer.MAX_VALUE
153      *     (that way the maxDenominator has no effect).
154      * OR
155      *   - a valid maxDenominator value and the epsilon
           value set to zero
156      *     (that way epsilon only has effect if there is an
           exact match before
157      *     the maxDenominator value is reached).
158      * </p><p>
159      *
160      * It has been done this way so that the same code can
           be (re)used for both
161      * scenarios. However this could be confusing to users
           if it were part of
162      * the public API and this constructor should therefore
           remain PRIVATE.
163      * </p>
164      *
165      * See JIRA issue ticket MATH-181 for more details:
166      *
167      *     https://issues.apache.org/jira/browse/MATH-181
168      *
169      * @param value the double value to convert to a
           fraction.
170      * @param epsilon maximum error allowed.  The resulting
           fraction is within
171      *        {@code epsilon} of {@code value}, in absolute
           terms.
172      * @param maxDenominator maximum denominator value
           allowed.
173      * @param maxIterations maximum number of convergents
174      * @throws FractionConversionException if the continued
```

```
         fraction failed to
175       *          converge.
176       */
177      private Fraction(double value, double epsilon, int
            maxDenominator, int maxIterations)
178          throws FractionConversionException
179      {
180          long overflow = Integer.MAX_VALUE;
181          double r0 = value;
182          long a0 = (long)FastMath.floor(r0);
183          if (FastMath.abs(a0) > overflow)
184          {
185              throw new FractionConversionException(value, a0
                    , 1l);
186          }
187
188          // check for (almost) integer arguments, which
                should not go
189          // to iterations.
190          if (FastMath.abs(a0 - value) < epsilon)
191          {
192              this.numerator = (int) a0;
193              this.denominator = 1;
194              return;
195          }
196
197          long p0 = 1;
198          long q0 = 0;
199          long p1 = a0;
200          long q1 = 1;
201
202          long p2 = 0;
203          long q2 = 1;
204
205          int n = 0;
206          boolean stop = false;
207          do
208          {
209              ++n;
210              double r1 = 1.0 / (r0 - a0);
211              long a1 = (long)FastMath.floor(r1);
212              p2 = (a1 * p1) + p0;
213              q2 = (a1 * q1) + q0;
214              if ((FastMath.abs(p2) > overflow) || (FastMath.
                    abs(q2) > overflow))
215              {
```

```
216                 throw new FractionConversionException(value
                        , p2, q2);
217             }
218
219             double convergent = (double)p2 / (double)q2;
220             if (n < maxIterations
221                 && FastMath.abs(convergent - value) >
                        epsilon
222                 && q2 < maxDenominator)
223                 {
224                 p0 = p1;
225                 p1 = p2;
226                 q0 = q1;
227                 q1 = q2;
228                 a0 = a1;
229                 r0 = r1;
230             }
231             else
232             {
233                 stop = true;
234             }
235         } while (!stop);
236
237         if (n >= maxIterations)
238         {
239             throw new FractionConversionException(value,
                        maxIterations);
240         }
241
242         if (q2 < maxDenominator)
243         {
244             this.numerator = (int) p2;
245             this.denominator = (int) q2;
246         }
247         else
248         {
249             this.numerator = (int) p1;
250             this.denominator = (int) q1;
251         }
252
253     }
254
255     /**
256      * Create a fraction from an int.
257      * The fraction is num / 1.
258      * @param num the numerator.
```

```
259         */
260        public Fraction(int num)
261        {
262            this(num, 1);
263        }
264
265        /**
266         * Create a fraction given the numerator and
               denominator.  The fraction is
267         * reduced to lowest terms.
268         * @param num the numerator.
269         * @param den the denominator.
270         * @throws MathArithmeticException if the denominator
               is {@code zero}
271         */
272        public Fraction(int num, int den)
273        {
274            if (den == 0)
275            {
276                throw new MathArithmeticException(
                       LocalizedFormats.
                       ZERO_DENOMINATOR_IN_FRACTION,
277                                                  num, den);
278            }
279            if (den < 0)
280            {
281                if (num == Integer.MIN_VALUE ||
282                    den == Integer.MIN_VALUE)
283                    {
284                    throw new MathArithmeticException(
                           LocalizedFormats.OVERFLOW_IN_FRACTION,
285                                                      num, den)
                                                         ;
286                }
287                num = -num;
288                den = -den;
289            }
290            // reduce numerator and denominator by greatest
                   common denominator.
291            final int d = ArithmeticUtils.gcd(num, den);
292            if (d > 1)
293            {
294                num /= d;
295                den /= d;
296            }
297
```

```
298         // move sign to numerator.
299         if (den < 0)
300         {
301             num = -num;
302             den = -den;
303         }
304         this.numerator   = num;
305         this.denominator = den;
306     }
307 
308     /**
309      * Returns the absolute value of this fraction.
310      * @return the absolute value.
311      */
312     public Fraction abs()
313     {
314         Fraction ret;
315         if (numerator >= 0)
316         {
317             ret = this;
318         }
319         else
320         {
321             ret = negate();
322         }
323         return ret;
324     }
325 
326     /**
327      * Compares this object to another based on size.
328      * @param object the object to compare to
329      * @return -1 if this is less than <tt>object</tt>, +1
           if this is greater
330      *         than <tt>object</tt>, 0 if they are equal.
331      */
332     public int compareTo(Fraction object)
333     {
334         long nOd = ((long) numerator) * object.denominator;
335         long dOn = ((long) denominator) * object.numerator;
336         return (nOd < dOn) ? -1 : ((nOd > dOn) ? +1 : 0);
337     }
338 
339     /**
340      * Gets the fraction as a <tt>double</tt>. This
           calculates the fraction as
341      * the numerator divided by denominator.
```

```
342         * @return the fraction as a <tt>double</tt>
343         */
344        @Override
345        public double doubleValue()
346        {
347            return (double)numerator / (double)denominator;
348        }
349
350        /**
351         * Test for the equality of two fractions.  If the
               lowest term
352         * numerator and denominators are the same for both
               fractions, the two
353         * fractions are considered to be equal.
354         * @param other fraction to test for equality to this
               fraction
355         * @return true if two fractions are equal, false if
               object is
356         *         <tt>null</tt>, not an instance of {@link
               Fraction}, or not equal
357         *         to this fraction instance.
358         */
359        @Override
360        public boolean equals(Object other)
361        {
362            if (this == other)
363            {
364                return true;
365            }
366            if (other instanceof Fraction)
367            {
368                // since fractions are always in lowest terms,
                       numerators and
369                // denominators can be compared directly for
                       equality.
370                Fraction rhs = (Fraction)other;
371                return (numerator == rhs.numerator) &&
372                    (denominator == rhs.denominator);
373            }
374            return false;
375        }
376
377        /**
378         * Gets the fraction as a <tt>float</tt>. This
               calculates the fraction as
379         * the numerator divided by denominator.
```

```
380         * @return the fraction as a <tt>float</tt>
381         */
382        @Override
383        public float floatValue()
384        {
385            return (float)doubleValue();
386        }
387
388        /**
389         * Access the denominator.
390         * @return the denominator.
391         */
392        public int getDenominator()
393        {
394            return denominator;
395        }
396
397        /**
398         * Access the numerator.
399         * @return the numerator.
400         */
401        public int getNumerator()
402        {
403            return numerator;
404        }
405
406        /**
407         * Gets a hashCode for the fraction.
408         * @return a hash code value for this object
409         */
410        @Override
411        public int hashCode()
412        {
413            return 37 * (37 * 17 + numerator) + denominator;
414        }
415
416        /**
417         * Gets the fraction as an <tt>int</tt>. This returns
                the whole number part
418         * of the fraction.
419         * @return the whole number fraction part
420         */
421        @Override
422        public int intValue()
423        {
424            return (int)doubleValue();
```

```
425     }
426
427     /**
428      * Gets the fraction as a <tt>long</tt>. This returns
            the whole number part
429      * of the fraction.
430      * @return the whole number fraction part
431      */
432     @Override
433     public long longValue()
434     {
435         return (long)doubleValue();
436     }
437
438     /**
439      * Return the additive inverse of this fraction.
440      * @return the negation of this fraction.
441      */
442     public Fraction negate()
443     {
444         if (numerator==Integer.MIN_VALUE)
445         {
446             throw new MathArithmeticException(
447                     LocalizedFormats.OVERFLOW_IN_FRACTION,
                            numerator, denominator);
448         }
449         return new Fraction(-numerator, denominator);
450     }
451
452     /**
453      * Return the multiplicative inverse of this fraction.
454      * @return the reciprocal fraction
455      */
456     public Fraction reciprocal()
457     {
458         return new Fraction(denominator, numerator);
459     }
460
461     /**
462      * <p>Adds the value of this fraction to another,
            returning the result in reduced form.
463      * The algorithm follows Knuth, 4.5.1.</p>
464      *
465      * @param fraction  the fraction to add, must not be {
            @code null}
466      * @return a {@code Fraction} instance with the
```

```
         resulting values
467      * @throws NullArgumentException if the fraction is {
             @code null}
468      * @throws MathArithmeticException if the resulting
             numerator or denominator exceeds
469      *  {@code Integer.MAX_VALUE}
470      */
471     public Fraction add(Fraction fraction)
472     {
473         return addSub(fraction, true /* add */);
474     }
475
476     /**
477      * Add an integer to the fraction.
478      * @param i the <tt>integer</tt> to add.
479      * @return this + i
480      */
481     public Fraction add(final int i)
482     {
483         return new Fraction(numerator + i * denominator,
                denominator);
484     }
485
486     /**
487      * <p>Subtracts the value of another fraction from the
             value of this one,
488      * returning the result in reduced form.</p>
489      *
490      * @param fraction  the fraction to subtract, must not
             be {@code null}
491      * @return a {@code Fraction} instance with the
             resulting values
492      * @throws NullArgumentException if the fraction is {
             @code null}
493      * @throws MathArithmeticException if the resulting
             numerator or denominator
494      *   cannot be represented in an {@code int}.
495      */
496     public Fraction subtract(Fraction fraction)
497     {
498         return addSub(fraction, false /* subtract */);
499     }
500
501     /**
502      * Subtract an integer from the fraction.
503      * @param i the <tt>integer</tt> to subtract.
```

```
504         * @return this - i
505         */
506        public Fraction subtract(final int i)
507        {
508            return new Fraction(numerator - i * denominator,
                   denominator);
509        }
510
511        /**
512         * Implement add and subtract using algorithm described
                in Knuth 4.5.1.
513         *
514         * @param fraction the fraction to subtract, must not
                be {@code null}
515         * @param isAdd true to add, false to subtract
516         * @return a {@code Fraction} instance with the
                resulting values
517         * @throws NullArgumentException if the fraction is {
                @code null}
518         * @throws MathArithmeticException if the resulting
                numerator or denominator
519         *   cannot be represented in an {@code int}.
520         */
521        private Fraction addSub(Fraction fraction, boolean
               isAdd)
522        {
523            if (fraction == null)
524            {
525                throw new NullArgumentException(
                       LocalizedFormats.FRACTION);
526            }
527            // zero is identity for addition.
528            if (numerator == 0)
529            {
530                return isAdd ? fraction : fraction.negate();
531            }
532            if (fraction.numerator == 0)
533            {
534                return this;
535            }
536            // if denominators are randomly distributed, d1
                   will be 1 about 61%
537            // of the time.
538            int d1 = ArithmeticUtils.gcd(denominator, fraction.
                   denominator);
539            if (d1==1)
```

```
540         {
541             // result is ( (u*v' +/- u'v) / u'v')
542             int uvp = ArithmeticUtils.mulAndCheck(numerator
                    , fraction.denominator);
543             int upv = ArithmeticUtils.mulAndCheck(fraction.
                    numerator, denominator);
544             return new Fraction
545                 (isAdd ? ArithmeticUtils.addAndCheck(uvp,
                        upv) :
546                  ArithmeticUtils.subAndCheck(uvp, upv),
547                  ArithmeticUtils.mulAndCheck(denominator,
                        fraction.denominator));
548         }
549         // the quantity 't' requires 65 bits of precision;
                see knuth 4.5.1
550         // exercise 7.  we're going to use a BigInteger.
551         // t = u(v'/d1) +/- v(u'/d1)
552         BigInteger uvp = BigInteger.valueOf(numerator)
553         .multiply(BigInteger.valueOf(fraction.denominator/
                d1));
554         BigInteger upv = BigInteger.valueOf(fraction.
                numerator)
555         .multiply(BigInteger.valueOf(denominator/d1));
556         BigInteger t = isAdd ? uvp.add(upv) : uvp.subtract(
                upv);
557         // but d2 doesn't need extra precision because
558         // d2 = gcd(t,d1) = gcd(t mod d1, d1)
559         int tmodd1 = t.mod(BigInteger.valueOf(d1)).intValue
                ();
560         int d2 = (tmodd1==0)?d1:ArithmeticUtils.gcd(tmodd1,
                d1);
561
562         // result is (t/d2) / (u'/d1)(v'/d2)
563         BigInteger w = t.divide(BigInteger.valueOf(d2));
564         if (w.bitLength() > 31)
565         {
566             throw new MathArithmeticException(
                        LocalizedFormats.
                        NUMERATOR_OVERFLOW_AFTER_MULTIPLY,
567                                               w);
568         }
569         return new Fraction (w.intValue(),
570                 ArithmeticUtils.mulAndCheck(denominator/d1,
571                         fraction.denominator/d2));
572     }
573
```

```
574     /**
575      * <p>Multiplies the value of this fraction by another,
             returning the
576      * result in reduced form.</p>
577      *
578      * @param fraction  the fraction to multiply by, must
             not be {@code null}
579      * @return a {@code Fraction} instance with the
             resulting values
580      * @throws NullArgumentException if the fraction is {
             @code null}
581      * @throws MathArithmeticException if the resulting
             numerator or denominator exceeds
582      *  {@code Integer.MAX_VALUE}
583      */
584     public Fraction multiply(Fraction fraction)
585     {
586         if (fraction == null)
587         {
588             throw new NullArgumentException(
                    LocalizedFormats.FRACTION);
589         }
590         if (numerator == 0 || fraction.numerator == 0)
591         {
592             return ZERO;
593         }
594         // knuth 4.5.1
595         // make sure we don't overflow unless the result *
                must* overflow.
596         int d1 = ArithmeticUtils.gcd(numerator, fraction.
                denominator);
597         int d2 = ArithmeticUtils.gcd(fraction.numerator,
                denominator);
598         return getReducedFraction
599         (ArithmeticUtils.mulAndCheck(numerator/d1, fraction
                .numerator/d2),
600                 ArithmeticUtils.mulAndCheck(denominator/d2,
                        fraction.denominator/d1));
601     }
602
603     /**
604      * Multiply the fraction by an integer.
605      * @param i the <tt>integer</tt> to multiply by.
606      * @return this * i
607      */
608     public Fraction multiply(final int i)
```

```
609     {
610         return new Fraction(numerator * i, denominator);
611     }
612 
613     /**
614      * <p>Divide the value of this fraction by another.</p>
615      *
616      * @param fraction  the fraction to divide by, must not
           be {@code null}
617      * @return a {@code Fraction} instance with the
           resulting values
618      * @throws IllegalArgumentException if the fraction is
           {@code null}
619      * @throws MathArithmeticException if the fraction to
           divide by is zero
620      * @throws MathArithmeticException if the resulting
           numerator or denominator exceeds
621      *  {@code Integer.MAX_VALUE}
622      */
623     public Fraction divide(Fraction fraction)
624     {
625         if (fraction == null)
626         {
627             throw new NullArgumentException(
                   LocalizedFormats.FRACTION);
628         }
629         if (fraction.numerator == 0)
630         {
631             throw new MathArithmeticException(
                   LocalizedFormats.ZERO_FRACTION_TO_DIVIDE_BY,
632                                               fraction.
                                                  numerator,
                                                   fraction.
                                                  denominator
                                                  );
633         }
634         return multiply(fraction.reciprocal());
635     }
636 
637     /**
638      * Divide the fraction by an integer.
639      * @param i the <tt>integer</tt> to divide by.
640      * @return this * i
641      */
642     public Fraction divide(final int i)
643     {
```

```
644         return new Fraction(numerator, denominator * i);
645     }
646 
647     /**
648      * <p>
649      * Gets the fraction percentage as a <tt>double</tt>.
           This calculates the
650      * fraction as the numerator divided by denominator
           multiplied by 100.
651      * </p>
652      *
653      * @return the fraction percentage as a <tt>double</tt
           >.
654      */
655     public double percentageValue()
656     {
657         return 100 * doubleValue();
658     }
659 
660     /**
661      * <p>Creates a {@code Fraction} instance with the 2
           parts
662      * of a fraction Y/Z.</p>
663      *
664      * <p>Any negative signs are resolved to be on the
           numerator.</p>
665      *
666      * @param numerator  the numerator, for example the
           three in 'three sevenths'
667      * @param denominator  the denominator, for example the
            seven in 'three sevenths'
668      * @return a new fraction instance, with the numerator
           and denominator reduced
669      * @throws MathArithmeticException if the denominator
           is {@code zero}
670      */
671     public static Fraction getReducedFraction(int numerator
           , int denominator)
672     {
673         if (denominator == 0)
674         {
675             throw new MathArithmeticException(
                    LocalizedFormats.
                    ZERO_DENOMINATOR_IN_FRACTION,
676                                               numerator,
                                                  denominator
```

```
                                                    );
677         }
678         if (numerator==0)
679         {
680             return ZERO; // normalize zero.
681         }
682         // allow 2^k/-2^31 as a valid fraction (where k>0)
683         if (denominator==Integer.MIN_VALUE && (numerator&1)
               ==0)
684         {
685             numerator/=2; denominator/=2;
686         }
687         if (denominator < 0)
688         {
689             if (numerator==Integer.MIN_VALUE ||
690                     denominator==Integer.MIN_VALUE)
691             {
692                 throw new MathArithmeticException(
                       LocalizedFormats.OVERFLOW_IN_FRACTION,
693                                                 numerator
                                                    ,
                                                    denominator
                                                    );
694             }
695             numerator = -numerator;
696             denominator = -denominator;
697         }
698         // simplify fraction.
699         int gcd = ArithmeticUtils.gcd(numerator,
               denominator);
700         numerator /= gcd;
701         denominator /= gcd;
702         return new Fraction(numerator, denominator);
703     }
704
705     /**
706      * <p>
707      * Returns the {@code String} representing this
           fraction, ie
708      * "num / dem" or just "num" if the denominator is one.
709      * </p>
710      *
711      * @return a string representation of the fraction.
712      * @see java.lang.Object#toString()
713      */
714     @Override
```

```
715     public String toString()
716     {
717         String str = null;
718         if (denominator == 1)
719         {
720             str = Integer.toString(numerator);
721         }
722         else if (numerator == 0)
723         {
724             str = "0";
725         }
726         else
727         {
728             str = numerator + " / " + denominator;
729         }
730         return str;
731     }
732 
733     /** {@inheritDoc} */
734     public FractionField getField()
735     {
736         return FractionField.getInstance();
737     }
738 
739 }
```

E.5 Manually Transformed Sources for Task Mockito-18

E.5.1 Defects-Sources/Mockito-18/ReturnsEmptyValues-bugged-A.java

```
1  /*
2   * Copyright (c) 2007 Mockito contributors
3   * This program is made available under the terms of the
       MIT License.
4   */
5
6  package org.mockito.internal.stubbing.defaultanswers;
7
8  import java.io.Serializable;
9  import java.util.ArrayList;
10 import java.util.Collection;
11 import java.util.HashMap;
12 import java.util.HashSet;
13 import java.util.LinkedHashMap;
14 import java.util.LinkedHashSet;
15 import java.util.LinkedList;
16 import java.util.List;
17 import java.util.Map;
18 import java.util.Set;
19 import java.util.SortedMap;
20 import java.util.SortedSet;
21 import java.util.TreeMap;
22 import java.util.TreeSet;
23 import org.mockito.internal.util.MockUtil;
24 import org.mockito.internal.util.ObjectMethodsGuru;
25 import org.mockito.internal.util.Primitives;
26 import org.mockito.invocation.InvocationOnMock;
27 import org.mockito.mock.MockName;
28 import org.mockito.stubbing.Answer;
29
30 /**
31  * Default answer of every Mockito mock.
32  * <ul>
33  * <li>
34  *  Returns appropriate primitive for primitive-returning
       methods
35  * </li>
36  * <li>
37  *  Returns consistent values for primitive wrapper classes
```

```
     (e.g.
38  *  int-returning method returns 0 <b>and</b> Integer-
       returning method
39  *  returns 0, too)
40  * </li>
41  * <li>
42  *  Returns empty collection for collection-returning
       methods (works
43  *  for most commonly used collection types)
44  * </li>
45  * <li>
46  *  Returns description of mock for toString() method
47  * </li>
48  * <li>
49  *  Returns zero if references are equals otherwise non-
       zero for
50  *  Comparable#compareTo(T other) method (see issue 184)
51  * </li>
52  * <li>
53  *  Returns null for everything else
54  * </li>
55  * </ul>
56  */
57 public class ReturnsEmptyValues implements Answer<Object>,
       Serializable {
58
59      private static final long serialVersionUID =
           1998191268711234347L;
60      ObjectMethodsGuru methodsGuru = new ObjectMethodsGuru()
           ;
61      MockUtil mockUtil = new MockUtil();
62
63      /* (non-Javadoc)
64       * @see org.mockito.stubbing.Answer#answer(
65       *                org.mockito.invocation.
           InvocationOnMock)
66       */
67      public Object answer(InvocationOnMock invocation) {
68          if (methodsGuru.isToString(invocation.getMethod()))
               {
69              Object mock = invocation.getMock();
70              MockName name = mockUtil.getMockName(mock);
71              if (name.isDefault()) {
72                  return "Mock for " + mockUtil.
                       getMockSettings(mock)
73                      .getTypeToMock().getSimpleName()
```

```
74                      + ", hashCode: " + mock.hashCode();
75              } else {
76                  return name.toString();
77              }
78          } else if (methodsGuru.isCompareToMethod(invocation
                .getMethod())) {
79
80              //see issue 184.
81              //mocks by default should return 0 if
                    references are the same,
82              //otherwise some other value because they are
                    not the same. Hence
83              //we return 1 (anything but 0 is good).
84              //Only for compareTo() method by the Comparable
                     interface
85              return invocation.getMock() == invocation.
                    getArguments()[0] ? 0 : 1;
86          }
87
88          Class<?> returnType = invocation.getMethod().
                getReturnType();
89          return returnValueFor(returnType);
90      }
91
92      Object returnValueFor(Class<?> type) {
93          if (Primitives.isPrimitiveOrWrapper(type)) {
94              return Primitives.
                    defaultValueForPrimitiveOrWrapper(type);
95
96              //new instances are used instead of Collections
                    .emptyList(), etc.
97              //to avoid UnsupportedOperationException if
                    code under test modifies
98              //returned collection
99          } else if (type == Collection.class) {
100             return new LinkedList<Object>();
101         } else if (type == Set.class) {
102             return new HashSet<Object>();
103         } else if (type == HashSet.class) {
104             return new HashSet<Object>();
105         } else if (type == SortedSet.class) {
106             return new TreeSet<Object>();
107         } else if (type == TreeSet.class) {
108             return new TreeSet<Object>();
109         } else if (type == LinkedHashSet.class) {
110             return new LinkedHashSet<Object>();
```

```
111         } else if (type == List.class) {
112             return new LinkedList<Object>();
113         } else if (type == LinkedList.class) {
114             return new LinkedList<Object>();
115         } else if (type == ArrayList.class) {
116             return new ArrayList<Object>();
117         } else if (type == Map.class) {
118             return new HashMap<Object, Object>();
119         } else if (type == HashMap.class) {
120             return new HashMap<Object, Object>();
121         } else if (type == SortedMap.class) {
122             return new TreeMap<Object, Object>();
123         } else if (type == TreeMap.class) {
124             return new TreeMap<Object, Object>();
125         } else if (type == LinkedHashMap.class) {
126             return new LinkedHashMap<Object, Object>();
127         }
128 
129         //Let's not care about the rest of collections.
130         return null;
131     }
132 }
```

E.5.2 Defects-Sources/Mockito-18/ReturnsEmptyValues-fixed-A.java

```
1  /*
2   * Copyright (c) 2007 Mockito contributors
3   * This program is made available under the terms of the
       MIT License.
4   */
5
6  package org.mockito.internal.stubbing.defaultanswers;
7
8  import java.io.Serializable;
9  import java.util.ArrayList;
10 import java.util.Collection;
11 import java.util.HashMap;
12 import java.util.HashSet;
13 import java.util.LinkedHashMap;
14 import java.util.LinkedHashSet;
15 import java.util.LinkedList;
16 import java.util.List;
17 import java.util.Map;
18 import java.util.Set;
19 import java.util.SortedMap;
20 import java.util.SortedSet;
21 import java.util.TreeMap;
22 import java.util.TreeSet;
23 import org.mockito.internal.util.MockUtil;
24 import org.mockito.internal.util.ObjectMethodsGuru;
25 import org.mockito.internal.util.Primitives;
26 import org.mockito.invocation.InvocationOnMock;
27 import org.mockito.mock.MockName;
28 import org.mockito.stubbing.Answer;
29
30 /**
31  * Default answer of every Mockito mock.
32  * <ul>
33  * <li>
34  *  Returns appropriate primitive for primitive-returning
       methods
35  * </li>
36  * <li>
37  *  Returns consistent values for primitive wrapper classes
       (e.g.
38  *  int-returning method returns 0 <b>and</b> Integer-
       returning method
39  *  returns 0, too)
```

```
40  * </li>
41  * <li>
42  *  Returns empty collection for collection-returning
        methods (works
43  *  for most commonly used collection types)
44  * </li>
45  * <li>
46  *  Returns description of mock for toString() method
47  * </li>
48  * <li>
49  *  Returns zero if references are equals otherwise non-
        zero for
50  *  Comparable#compareTo(T other) method (see issue 184)
51  * </li>
52  * <li>
53  *  Returns null for everything else
54  * </li>
55  * </ul>
56  */
57 public class ReturnsEmptyValues implements Answer<Object>,
       Serializable {
58
59      private static final long serialVersionUID =
           1998191268711234347L;
60      ObjectMethodsGuru methodsGuru = new ObjectMethodsGuru()
           ;
61      MockUtil mockUtil = new MockUtil();
62
63      /* (non-Javadoc)
64       * @see org.mockito.stubbing.Answer#answer(
65       *                org.mockito.invocation.
            InvocationOnMock)
66       */
67      public Object answer(InvocationOnMock invocation) {
68          if (methodsGuru.isToString(invocation.getMethod()))
               {
69              Object mock = invocation.getMock();
70              MockName name = mockUtil.getMockName(mock);
71              if (name.isDefault()) {
72                  return "Mock for " + mockUtil.
                       getMockSettings(mock)
73                      .getTypeToMock().getSimpleName()
74                      + ", hashCode: " + mock.hashCode();
75              } else {
76                  return name.toString();
77              }
```

```
78          } else if (methodsGuru.isCompareToMethod(invocation
               .getMethod())) {
79
80              //see issue 184.
81              //mocks by default should return 0 if
                   references are the same,
82              //otherwise some other value because they are
                   not the same. Hence
83              //we return 1 (anything but 0 is good).
84              //Only for compareTo() method by the Comparable
                    interface
85              return invocation.getMock() == invocation.
                   getArguments()[0] ? 0 : 1;
86          }
87
88          Class<?> returnType = invocation.getMethod().
               getReturnType();
89          return returnValueFor(returnType);
90      }
91
92      Object returnValueFor(Class<?> type) {
93          if (Primitives.isPrimitiveOrWrapper(type)) {
94              return Primitives.
                   defaultValueForPrimitiveOrWrapper(type);
95
96              //new instances are used instead of Collections
                   .emptyList(), etc.
97              //to avoid UnsupportedOperationException if
                   code under test modifies
98              //returned collection
99          } else if (type == Iterable.class) {
100             return new ArrayList<Object>(0);
101         } else if (type == Collection.class) {
102             return new LinkedList<Object>();
103         } else if (type == Set.class) {
104             return new HashSet<Object>();
105         } else if (type == HashSet.class) {
106             return new HashSet<Object>();
107         } else if (type == SortedSet.class) {
108             return new TreeSet<Object>();
109         } else if (type == TreeSet.class) {
110             return new TreeSet<Object>();
111         } else if (type == LinkedHashSet.class) {
112             return new LinkedHashSet<Object>();
113         } else if (type == List.class) {
114             return new LinkedList<Object>();
```

```
115         } else if (type == LinkedList.class) {
116             return new LinkedList<Object>();
117         } else if (type == ArrayList.class) {
118             return new ArrayList<Object>();
119         } else if (type == Map.class) {
120             return new HashMap<Object, Object>();
121         } else if (type == HashMap.class) {
122             return new HashMap<Object, Object>();
123         } else if (type == SortedMap.class) {
124             return new TreeMap<Object, Object>();
125         } else if (type == TreeMap.class) {
126             return new TreeMap<Object, Object>();
127         } else if (type == LinkedHashMap.class) {
128             return new LinkedHashMap<Object, Object>();
129         }
130 
131         //Let's not care about the rest of collections.
132         return null;
133     }
134 }
```

E.5.3 Defects-Sources/Mockito-18/ReturnsEmptyValues-bugged-B.java

```
1  /*
2   * Copyright (c) 2007 Mockito contributors
3   * This program is made available under the terms of the
       MIT License.
4   */
5
6  package org.mockito.internal.stubbing.defaultanswers;
7
8  import java.io.Serializable;
9  import java.util.ArrayList;
10 import java.util.Collection;
11 import java.util.HashMap;
12 import java.util.HashSet;
13 import java.util.LinkedHashMap;
14 import java.util.LinkedHashSet;
15 import java.util.LinkedList;
16 import java.util.List;
17 import java.util.Map;
18 import java.util.Set;
19 import java.util.SortedMap;
20 import java.util.SortedSet;
21 import java.util.TreeMap;
22 import java.util.TreeSet;
23 import org.mockito.internal.util.MockUtil;
24 import org.mockito.internal.util.ObjectMethodsGuru;
25 import org.mockito.internal.util.Primitives;
26 import org.mockito.invocation.InvocationOnMock;
27 import org.mockito.mock.MockName;
28 import org.mockito.stubbing.Answer;
29
30 /**
31  * Default answer of every Mockito mock.
32  * <ul>
33  * <li>
34  *  Returns appropriate primitive for primitive-returning
       methods
35  * </li>
36  * <li>
37  *  Returns consistent values for primitive wrapper classes
       (e.g. int-returning method returns 0
38  *  <b>and</b> Integer-returning method returns 0, too)
39  * </li>
40  * <li>
```

```
41  *  Returns empty collection for collection-returning
       methods (works for most commonly used
42  *  collection types)
43  * </li>
44  * <li>
45  *  Returns description of mock for toString() method
46  * </li>
47  * <li>
48  *  Returns zero if references are equals otherwise non-
       zero for Comparable#compareTo(T other)
49  *  method (see issue 184)
50  * </li>
51  * <li>
52  *  Returns null for everything else
53  * </li>
54  * </ul>
55  */
56 public class ReturnsEmptyValues implements Answer<Object>,
       Serializable
57 {
58
59      private static final long serialVersionUID =
           1998191268711234347L;
60      ObjectMethodsGuru methodsGuru = new ObjectMethodsGuru()
           ;
61      MockUtil mockUtil = new MockUtil();
62
63      /* (non-Javadoc)
64       * @see org.mockito.stubbing.Answer#answer(org.mockito.
           invocation.InvocationOnMock)
65       */
66      public Object answer(InvocationOnMock invocation)
67      {
68          if (methodsGuru.isToString(invocation.getMethod()))
69          {
70              Object mock = invocation.getMock();
71              MockName name = mockUtil.getMockName(mock);
72              if (name.isDefault())
73              {
74                  return "Mock for " + mockUtil.
                       getMockSettings(mock).getTypeToMock().
                       getSimpleName()
75                          + ", hashCode: " + mock.hashCode();
76              }
77              else
78              {
```

```
79                  return name.toString();
80              }
81          }
82          else if (methodsGuru.isCompareToMethod(invocation.
               getMethod()))
83          {
84              //see issue 184.
85              //mocks by default should return 0 if
                   references are the same, otherwise some
                   other
86              //value because they are not the same. Hence we
                    return 1 (anything but 0 is good).
87              //Only for compareTo() method by the Comparable
                    interface
88              return invocation.getMock() == invocation.
                   getArguments()[0] ? 0 : 1;
89          }
90
91          Class<?> returnType = invocation.getMethod().
               getReturnType();
92          return returnValueFor(returnType);
93      }
94
95      Object returnValueFor(Class<?> type)
96      {
97          if (Primitives.isPrimitiveOrWrapper(type))
98          {
99              return Primitives.
                   defaultValueForPrimitiveOrWrapper(type);
100             //new instances are used instead of Collections
                   .emptyList(), etc.
101             //to avoid UnsupportedOperationException if
                   code under test modifies returned
102             //collection
103         }
104         else if (type == Collection.class)
105         {
106             return new LinkedList<Object>();
107         }
108         else if (type == Set.class)
109         {
110             return new HashSet<Object>();
111         }
112         else if (type == HashSet.class)
113         {
114             return new HashSet<Object>();
```

```
115         }
116         else if (type == SortedSet.class)
117         {
118             return new TreeSet<Object>();
119         }
120         else if (type == TreeSet.class)
121         {
122             return new TreeSet<Object>();
123         }
124         else if (type == LinkedHashSet.class)
125         {
126             return new LinkedHashSet<Object>();
127         }
128         else if (type == List.class)
129         {
130             return new LinkedList<Object>();
131         }
132         else if (type == LinkedList.class)
133         {
134             return new LinkedList<Object>();
135         }
136         else if (type == ArrayList.class)
137         {
138             return new ArrayList<Object>();
139         }
140         else if (type == Map.class)
141         {
142             return new HashMap<Object, Object>();
143         }
144         else if (type == HashMap.class)
145         {
146             return new HashMap<Object, Object>();
147         }
148         else if (type == SortedMap.class)
149         {
150             return new TreeMap<Object, Object>();
151         }
152         else if (type == TreeMap.class)
153         {
154             return new TreeMap<Object, Object>();
155         }
156         else if (type == LinkedHashMap.class)
157         {
158             return new LinkedHashMap<Object, Object>();
159         }
160         //Let's not care about the rest of collections.
```

```
161 |           return null;
162 |       }
163 |
164 | }
```

E.5.4 Defects-Sources/Mockito-18/ReturnsEmptyValues-fixed-B.java

```
1  /*
2   * Copyright (c) 2007 Mockito contributors
3   * This program is made available under the terms of the
       MIT License.
4   */
5
6  package org.mockito.internal.stubbing.defaultanswers;
7
8  import java.io.Serializable;
9  import java.util.ArrayList;
10 import java.util.Collection;
11 import java.util.HashMap;
12 import java.util.HashSet;
13 import java.util.LinkedHashMap;
14 import java.util.LinkedHashSet;
15 import java.util.LinkedList;
16 import java.util.List;
17 import java.util.Map;
18 import java.util.Set;
19 import java.util.SortedMap;
20 import java.util.SortedSet;
21 import java.util.TreeMap;
22 import java.util.TreeSet;
23 import org.mockito.internal.util.MockUtil;
24 import org.mockito.internal.util.ObjectMethodsGuru;
25 import org.mockito.internal.util.Primitives;
26 import org.mockito.invocation.InvocationOnMock;
27 import org.mockito.mock.MockName;
28 import org.mockito.stubbing.Answer;
29
30 /**
31  * Default answer of every Mockito mock.
32  * <ul>
33  * <li>
34  *  Returns appropriate primitive for primitive-returning
       methods
35  * </li>
36  * <li>
37  *  Returns consistent values for primitive wrapper classes
       (e.g. int-returning method returns 0
38  *  <b>and</b> Integer-returning method returns 0, too)
39  * </li>
40  * <li>
```

```
41  *  Returns empty collection for collection-returning
       methods (works for most commonly used
42  *  collection types)
43  * </li>
44  * <li>
45  *  Returns description of mock for toString() method
46  * </li>
47  * <li>
48  *  Returns zero if references are equals otherwise non-
       zero for Comparable#compareTo(T other)
49  *  method (see issue 184)
50  * </li>
51  * <li>
52  *  Returns null for everything else
53  * </li>
54  * </ul>
55  */
56 public class ReturnsEmptyValues implements Answer<Object>,
       Serializable
57 {
58
59      private static final long serialVersionUID =
           1998191268711234347L;
60      ObjectMethodsGuru methodsGuru = new ObjectMethodsGuru()
           ;
61      MockUtil mockUtil = new MockUtil();
62
63      /* (non-Javadoc)
64       * @see org.mockito.stubbing.Answer#answer(org.mockito.
           invocation.InvocationOnMock)
65       */
66      public Object answer(InvocationOnMock invocation)
67      {
68          if (methodsGuru.isToString(invocation.getMethod()))
69          {
70              Object mock = invocation.getMock();
71              MockName name = mockUtil.getMockName(mock);
72              if (name.isDefault())
73              {
74                  return "Mock for " + mockUtil.
                       getMockSettings(mock).getTypeToMock().
                       getSimpleName()
75                          + ", hashCode: " + mock.hashCode();
76              }
77              else
78              {
```

```
79                  return name.toString();
80              }
81          }
82          else if (methodsGuru.isCompareToMethod(invocation.
               getMethod()))
83          {
84              //see issue 184.
85              //mocks by default should return 0 if
                   references are the same, otherwise some
                   other
86              //value because they are not the same. Hence we
                    return 1 (anything but 0 is good).
87              //Only for compareTo() method by the Comparable
                    interface
88              return invocation.getMock() == invocation.
                   getArguments()[0] ? 0 : 1;
89          }
90
91          Class<?> returnType = invocation.getMethod().
               getReturnType();
92          return returnValueFor(returnType);
93      }
94
95      Object returnValueFor(Class<?> type)
96      {
97          if (Primitives.isPrimitiveOrWrapper(type))
98          {
99              return Primitives.
                   defaultValueForPrimitiveOrWrapper(type);
100             //new instances are used instead of Collections
                   .emptyList(), etc.
101             //to avoid UnsupportedOperationException if
                   code under test modifies returned
102             //collection
103         }
104         else if (type == Iterable.class)
105         {
106             return new ArrayList<Object>(0);
107         }
108         else if (type == Collection.class)
109         {
110             return new LinkedList<Object>();
111         }
112         else if (type == Set.class)
113         {
114             return new HashSet<Object>();
```

```
115         }
116         else if (type == HashSet.class)
117         {
118             return new HashSet<Object>();
119         }
120         else if (type == SortedSet.class)
121         {
122             return new TreeSet<Object>();
123         }
124         else if (type == TreeSet.class)
125         {
126             return new TreeSet<Object>();
127         }
128         else if (type == LinkedHashSet.class)
129         {
130             return new LinkedHashSet<Object>();
131         }
132         else if (type == List.class)
133         {
134             return new LinkedList<Object>();
135         }
136         else if (type == LinkedList.class)
137         {
138             return new LinkedList<Object>();
139         }
140         else if (type == ArrayList.class)
141         {
142             return new ArrayList<Object>();
143         }
144         else if (type == Map.class)
145         {
146             return new HashMap<Object, Object>();
147         }
148         else if (type == HashMap.class)
149         {
150             return new HashMap<Object, Object>();
151         }
152         else if (type == SortedMap.class)
153         {
154             return new TreeMap<Object, Object>();
155         }
156         else if (type == TreeMap.class)
157         {
158             return new TreeMap<Object, Object>();
159         }
160         else if (type == LinkedHashMap.class)
```

```
161         {
162             return new LinkedHashMap<Object, Object>();
163         }
164         //Let's not care about the rest of collections.
165         return null;
166     }
167 
168 }
```

E.6 Manually Transformed Sources for Task Time-15

E.6.1 Defects-Sources/Time-15/FieldUtils-bugged-A.java

```
1  /*
2   *  Copyright 2001-2005 Stephen Colebourne
3   *
4   *  Licensed under the Apache License, Version 2.0 (the "
       License");
5   *  you may not use this file except in compliance with the
        License.
6   *  You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  *  Unless required by applicable law or agreed to in
       writing,
11  *  software distributed under the License is distributed
       on an
12  *  "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
       KIND,
13  *  either express or implied. See the License for the
       specific
14  *  language governing permissions and limitations under
       the License.
15  */
16 package org.joda.time.field;
17 
18 
19 import org.joda.time.DateTimeField;
20 import org.joda.time.DateTimeFieldType;
21 import org.joda.time.IllegalFieldValueException;
22 
23 
24 /**
25  * General utilities that don't fit elsewhere.
26  * <p>
27  * FieldUtils is thread-safe and immutable.
28  *
29  * @author Stephen Colebourne
30  * @since 1.0
31  */
32 public class FieldUtils {
33 
34     /**
```

```
35      * Restricted constructor.
36      */
37     private FieldUtils() {
38         super();
39     }
40
41
42     //-----------------------------------------------------------------------
43     /**
44      * Negates the input throwing an exception if it can't
           negate it.
45      *
46      * @param value  the value to negate
47      * @return the negated value
48      * @throws ArithmeticException if the value is Integer.
           MIN_VALUE
49      * @since 1.1
50      */
51     public static int safeNegate(int value) {
52         if (value == Integer.MIN_VALUE) {
53             throw new ArithmeticException(
54                 "Integer.MIN_VALUE cannot be negated");
55         }
56         return -value;
57     }
58
59     /**
60      * Add two values throwing an exception if overflow
           occurs.
61      *
62      * @param val1  the first value
63      * @param val2  the second value
64      * @return the new total
65      * @throws ArithmeticException if the value is too big
           or too
66      *  small
67      */
68     public static int safeAdd(int val1, int val2) {
69         int sum = val1 + val2;
70
71         // If there is a sign change, but the two values
              have the same sign...
72         if ((val1 ^ sum) < 0 && (val1 ^ val2) >= 0) {
73             throw new ArithmeticException
```

```
74                  ("The calculation caused an overflow: " +
                       val1 + " + " + val2);
75          }
76          return sum;
77      }
78
79      /**
80       * Add two values throwing an exception if overflow
             occurs.
81       *
82       * @param val1  the first value
83       * @param val2  the second value
84       * @return the new total
85       * @throws ArithmeticException if the value is too big
             or too
86       *  small
87       */
88      public static long safeAdd(long val1, long val2) {
89          long sum = val1 + val2;
90
91          // If there is a sign change, but the two values
               have the same sign...
92          if ((val1 ^ sum) < 0 && (val1 ^ val2) >= 0) {
93              throw new ArithmeticException
94                  ("The calculation caused an overflow: " +
                       val1 + " + " + val2);
95          }
96          return sum;
97      }
98
99      /**
100      * Subtracts two values throwing an exception if
             overflow occurs.
101      *
102      * @param val1  the first value, to be taken away from
103      * @param val2  the second value, the amount to take
             away
104      * @return the new total
105      * @throws ArithmeticException if the value is too big
             or too
106      *  small
107      */
108     public static long safeSubtract(long val1, long val2) {
109         long diff = val1 - val2;
110
111         // If there is a sign change, but the two values
```

```
            have different signs...
112         if ((val1 ^ diff) < 0 && (val1 ^ val2) < 0) {
113             throw new ArithmeticException
114                 ("The calculation caused an overflow: " +
                       val1 + " - " + val2);
115         }
116         return diff;
117     }
118
119     /**
120      * Multiply two values throwing an exception if
           overflow occurs.
121      *
122      * @param val1  the first value
123      * @param val2  the second value
124      * @return the new total
125      * @throws ArithmeticException if the value is too big
           or too
126      *  small
127      * @since 1.2
128      */
129     public static int safeMultiply(int val1, int val2) {
130         long total = (long) val1 * (long) val2;
131         if (total < Integer.MIN_VALUE || total > Integer.
              MAX_VALUE) {
132           throw new ArithmeticException("Multiplication
                 overflows an int: "
133                   + val1 + " * " + val2);
134         }
135         return (int) total;
136     }
137
138     /**
139      * Multiply two values throwing an exception if
           overflow occurs.
140      *
141      * @param val1  the first value
142      * @param val2  the second value
143      * @return the new total
144      * @throws ArithmeticException if the value is too big
           or too
145      *  small
146      * @since 1.2
147      */
148     public static long safeMultiply(long val1, int val2) {
149         switch (val2) {
```

```
150             case -1:
151                 return -val1;
152             case 0:
153                 return 0L;
154             case 1:
155                 return val1;
156         }
157         long total = val1 * val2;
158         if (total / val2 != val1) {
159           throw new ArithmeticException("Multiplication
                overflows a long: "
160                 + val1 + " * " + val2);
161         }
162         return total;
163     }
164 
165     /**
166      * Multiply two values throwing an exception if
           overflow occurs.
167      *
168      * @param val1  the first value
169      * @param val2  the second value
170      * @return the new total
171      * @throws ArithmeticException if the value is too big
           or too
172      *  small
173      */
174     public static long safeMultiply(long val1, long val2) {
175         if (val2 == 1) {
176             return val1;
177         }
178         if (val1 == 1) {
179             return val2;
180         }
181         if (val1 == 0 || val2 == 0) {
182             return 0;
183         }
184         long total = val1 * val2;
185         if (total / val2 != val1 || val1 == Long.MIN_VALUE
              && val2 == -1
186             || val2 == Long.MIN_VALUE && val1 == -1) {
187             throw new ArithmeticException("Multiplication
                  overflows a long: "
188                 + val1 + " * " + val2);
189         }
190         return total;
```

```
191     }
192 
193     /**
194      * Casts to an int throwing an exception if overflow
           occurs.
195      *
196      * @param value  the value
197      * @return the value as an int
198      * @throws ArithmeticException if the value is too big
           or too
199      *  small
200      */
201     public static int safeToInt(long value) {
202         if (Integer.MIN_VALUE <= value && value <= Integer.
              MAX_VALUE) {
203             return (int) value;
204         }
205         throw new ArithmeticException("Value cannot fit in
              an int: " + value);
206     }
207 
208     /**
209      * Multiply two values to return an int throwing an
           exception if
210      * overflow occurs.
211      *
212      * @param val1  the first value
213      * @param val2  the second value
214      * @return the new total
215      * @throws ArithmeticException if the value is too big
           or too
216      *  small
217      */
218     public static int safeMultiplyToInt(long val1, long
          val2) {
219         long val = FieldUtils.safeMultiply(val1, val2);
220         return FieldUtils.safeToInt(val);
221     }
222 
223 
224     //
           -----------------------------------------------------------------
225     /**
226      * Verify that input values are within specified bounds
           .
```

```
227         *
228         * @param value  the value to check
229         * @param lowerBound  the lower bound allowed for value
230         * @param upperBound  the upper bound allowed for value
231         * @throws IllegalFieldValueException if value is not
                in the
232         *  specified bounds
233         */
234        public static void verifyValueBounds(DateTimeField
               field,
235                                             int value,
236                                             int lowerBound,
237                                             int upperBound) {
238            if ((value < lowerBound) || (value > upperBound)) {
239                throw new IllegalFieldValueException
240                    (field.getType(), Integer.valueOf(value),
241                     Integer.valueOf(lowerBound), Integer.
                            valueOf(upperBound));
242            }
243        }
244
245        /**
246         * Verify that input values are within specified bounds
                .
247         *
248         * @param value  the value to check
249         * @param lowerBound  the lower bound allowed for value
250         * @param upperBound  the upper bound allowed for value
251         * @throws IllegalFieldValueException if value is not
                in the
252         *  specified bounds
253         * @since 1.1
254         */
255        public static void verifyValueBounds(DateTimeFieldType
               fieldType,
256                                             int value,
257                                             int lowerBound,
258                                             int upperBound) {
259            if ((value < lowerBound) || (value > upperBound)) {
260                throw new IllegalFieldValueException
261                    (fieldType, Integer.valueOf(value),
262                     Integer.valueOf(lowerBound), Integer.
                            valueOf(upperBound));
263            }
264        }
265
```

```
266     /**
267      * Verify that input values are within specified bounds
             .
268      *
269      * @param value  the value to check
270      * @param lowerBound  the lower bound allowed for value
271      * @param upperBound  the upper bound allowed for value
272      * @throws IllegalFieldValueException if value is not
             in the
273      *  specified bounds
274      */
275     public static void verifyValueBounds(String fieldName,
276                                          int value,
277                                          int lowerBound,
278                                          int upperBound) {
279         if ((value < lowerBound) || (value > upperBound)) {
280             throw new IllegalFieldValueException
281                 (fieldName, Integer.valueOf(value),
282                  Integer.valueOf(lowerBound), Integer.
                        valueOf(upperBound));
283         }
284     }
285
286     /**
287      * Utility method used by addWrapField implementations
             to ensure
288      * the new value lies within the field's legal value
             range.
289      *
290      * @param currentValue the current value of the data,
             which may
291      *  lie outside
292      * the wrapped value range
293      * @param wrapValue  the value to add to current value
             before
294      *  wrapping.  This may be negative.
295      * @param minValue the wrap range minimum value.
296      * @param maxValue the wrap range maximum value.  This
             must be
297      *  greater than minValue (checked by the method).
298      * @return the wrapped value
299      * @throws IllegalArgumentException if minValue is
             greater
300      *  than or equal to maxValue
301      */
302     public static int getWrappedValue(int currentValue, int
```

```
        wrapValue,
                                        int minValue, int
                                           maxValue) {
        return getWrappedValue(currentValue + wrapValue,
           minValue, maxValue);
    }

    /**
     * Utility method that ensures the given value lies
        within the
     * field's legal value range.
     *
     * @param value  the value to fit into the wrapped
        value range
     * @param minValue the wrap range minimum value.
     * @param maxValue the wrap range maximum value.  This
        must be
     * greater than minValue (checked by the method).
     * @return the wrapped value
     * @throws IllegalArgumentException if minValue is
        greater
     *  than or equal to maxValue
     */
    public static int getWrappedValue(int value, int
       minValue, int maxValue) {
        if (minValue >= maxValue) {
            throw new IllegalArgumentException("MIN > MAX")
               ;
        }

        int wrapRange = maxValue - minValue + 1;
        value -= minValue;

        if (value >= 0) {
            return (value % wrapRange) + minValue;
        }

        int remByRange = (-value) % wrapRange;

        if (remByRange == 0) {
            return 0 + minValue;
        }
        return (wrapRange - remByRange) + minValue;
    }
```

```
340     //
           -----------------------------------------------------------------
341     /**
342      * Compares two objects as equals handling null.
343      *
344      * @param object1  the first object
345      * @param object2  the second object
346      * @return true if equal
347      * @since 1.4
348      */
349     public static boolean equals(Object object1, Object
           object2) {
350         if (object1 == object2) {
351             return true;
352         }
353         if (object1 == null || object2 == null) {
354             return false;
355         }
356         return object1.equals(object2);
357     }
358 }
```

E.6.2 Defects-Sources/Time-15/FieldUtils-fixed-A.java

```
1  /*
2   *  Copyright 2001-2005 Stephen Colebourne
3   *
4   *  Licensed under the Apache License, Version 2.0 (the "
       License");
5   *  you may not use this file except in compliance with the
        License.
6   *  You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  *  Unless required by applicable law or agreed to in
       writing,
11  *  software distributed under the License is distributed
       on an
12  *  "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
       KIND,
13  *  either express or implied. See the License for the
       specific
14  *  language governing permissions and limitations under
       the License.
15  */
16 package org.joda.time.field;
17
18
19 import org.joda.time.DateTimeField;
20 import org.joda.time.DateTimeFieldType;
21 import org.joda.time.IllegalFieldValueException;
22
23
24 /**
25  * General utilities that don't fit elsewhere.
26  * <p>
27  * FieldUtils is thread-safe and immutable.
28  *
29  * @author Stephen Colebourne
30  * @since 1.0
31  */
32 public class FieldUtils {
33
34     /**
35      * Restricted constructor.
36      */
37     private FieldUtils() {
```

```
38          super();
39      }
40
41
42      //
           -----------------------------------------------------------------
43      /**
44       * Negates the input throwing an exception if it can't
            negate it.
45       *
46       * @param value  the value to negate
47       * @return the negated value
48       * @throws ArithmeticException if the value is Integer.
            MIN_VALUE
49       * @since 1.1
50       */
51      public static int safeNegate(int value) {
52          if (value == Integer.MIN_VALUE) {
53              throw new ArithmeticException(
54                  "Integer.MIN_VALUE cannot be negated");
55          }
56          return -value;
57      }
58
59      /**
60       * Add two values throwing an exception if overflow
            occurs.
61       *
62       * @param val1  the first value
63       * @param val2  the second value
64       * @return the new total
65       * @throws ArithmeticException if the value is too big
            or too
66       *  small
67       */
68      public static int safeAdd(int val1, int val2) {
69          int sum = val1 + val2;
70
71          // If there is a sign change, but the two values
               have the same sign...
72          if ((val1 ^ sum) < 0 && (val1 ^ val2) >= 0) {
73              throw new ArithmeticException
74                  ("The calculation caused an overflow: " +
                       val1 + " + " + val2);
75          }
```

```
76          return sum;
77      }
78
79      /**
80       * Add two values throwing an exception if overflow
             occurs.
81       *
82       * @param val1  the first value
83       * @param val2  the second value
84       * @return the new total
85       * @throws ArithmeticException if the value is too big
             or too
86       *  small
87       */
88      public static long safeAdd(long val1, long val2) {
89          long sum = val1 + val2;
90
91          // If there is a sign change, but the two values
                have the same sign...
92          if ((val1 ^ sum) < 0 && (val1 ^ val2) >= 0) {
93              throw new ArithmeticException
94                  ("The calculation caused an overflow: " +
                        val1 + " + " + val2);
95          }
96          return sum;
97      }
98
99      /**
100      * Subtracts two values throwing an exception if
             overflow occurs.
101      *
102      * @param val1  the first value, to be taken away from
103      * @param val2  the second value, the amount to take
             away
104      * @return the new total
105      * @throws ArithmeticException if the value is too big
             or too
106      *  small
107      */
108     public static long safeSubtract(long val1, long val2) {
109         long diff = val1 - val2;
110
111         // If there is a sign change, but the two values
                have different signs...
112         if ((val1 ^ diff) < 0 && (val1 ^ val2) < 0) {
113             throw new ArithmeticException
```

```
114                 ("The calculation caused an overflow: " +
                       val1 + " - " + val2);
115         }
116         return diff;
117     }
118 
119     /**
120      * Multiply two values throwing an exception if
           overflow occurs.
121      *
122      * @param val1  the first value
123      * @param val2  the second value
124      * @return the new total
125      * @throws ArithmeticException if the value is too big
           or too
126      *  small
127      * @since 1.2
128      */
129     public static int safeMultiply(int val1, int val2) {
130         long total = (long) val1 * (long) val2;
131         if (total < Integer.MIN_VALUE || total > Integer.
               MAX_VALUE) {
132           throw new ArithmeticException("Multiplication
                 overflows an int: "
133                 + val1 + " * " + val2);
134         }
135         return (int) total;
136     }
137 
138     /**
139      * Multiply two values throwing an exception if
           overflow occurs.
140      *
141      * @param val1  the first value
142      * @param val2  the second value
143      * @return the new total
144      * @throws ArithmeticException if the value is too big
           or too
145      *  small
146      * @since 1.2
147      */
148     public static long safeMultiply(long val1, int val2) {
149         switch (val2) {
150             case -1:
151                 if (val1 == Long.MIN_VALUE) {
152                     throw new ArithmeticException(
```

```
153                             "Multiplication overflows a
                                   long: " + val1
154                             + " * " + val2);
155                 }
156                 return -val1;
157             case 0:
158                 return 0L;
159             case 1:
160                 return val1;
161         }
162         long total = val1 * val2;
163         if (total / val2 != val1) {
164           throw new ArithmeticException("Multiplication
                 overflows a long: "
165                 + val1 + " * " + val2);
166         }
167         return total;
168     }
169 
170     /**
171      * Multiply two values throwing an exception if
           overflow occurs.
172      *
173      * @param val1  the first value
174      * @param val2  the second value
175      * @return the new total
176      * @throws ArithmeticException if the value is too big
           or too
177      *  small
178      */
179     public static long safeMultiply(long val1, long val2) {
180         if (val2 == 1) {
181             return val1;
182         }
183         if (val1 == 1) {
184             return val2;
185         }
186         if (val1 == 0 || val2 == 0) {
187             return 0;
188         }
189         long total = val1 * val2;
190         if (total / val2 != val1 || val1 == Long.MIN_VALUE
              && val2 == -1
191             || val2 == Long.MIN_VALUE && val1 == -1) {
192             throw new ArithmeticException("Multiplication
                   overflows a long: "
```

```
193                 + val1 + " * " + val2);
194         }
195         return total;
196     }
197 
198     /**
199      * Casts to an int throwing an exception if overflow
           occurs.
200      *
201      * @param value  the value
202      * @return the value as an int
203      * @throws ArithmeticException if the value is too big
           or too
204      *  small
205      */
206     public static int safeToInt(long value) {
207         if (Integer.MIN_VALUE <= value && value <= Integer.
              MAX_VALUE) {
208             return (int) value;
209         }
210         throw new ArithmeticException("Value cannot fit in
              an int: " + value);
211     }
212 
213     /**
214      * Multiply two values to return an int throwing an
           exception if
215      * overflow occurs.
216      *
217      * @param val1  the first value
218      * @param val2  the second value
219      * @return the new total
220      * @throws ArithmeticException if the value is too big
           or too
221      *  small
222      */
223     public static int safeMultiplyToInt(long val1, long
          val2) {
224         long val = FieldUtils.safeMultiply(val1, val2);
225         return FieldUtils.safeToInt(val);
226     }
227 
228 
229     //
          -----------------------------------------------------------------
```

```
230     /**
231      * Verify that input values are within specified bounds
             .
232      *
233      * @param value  the value to check
234      * @param lowerBound  the lower bound allowed for value
235      * @param upperBound  the upper bound allowed for value
236      * @throws IllegalFieldValueException if value is not
             in the
237      *  specified bounds
238      */
239     public static void verifyValueBounds(DateTimeField
           field,
240                                          int value,
241                                          int lowerBound,
242                                          int upperBound) {
243         if ((value < lowerBound) || (value > upperBound)) {
244             throw new IllegalFieldValueException
245                 (field.getType(), Integer.valueOf(value),
246                  Integer.valueOf(lowerBound), Integer.
                       valueOf(upperBound));
247         }
248     }
249
250     /**
251      * Verify that input values are within specified bounds
             .
252      *
253      * @param value  the value to check
254      * @param lowerBound  the lower bound allowed for value
255      * @param upperBound  the upper bound allowed for value
256      * @throws IllegalFieldValueException if value is not
             in the
257      *  specified bounds
258      * @since 1.1
259      */
260     public static void verifyValueBounds(DateTimeFieldType
           fieldType,
261                                          int value,
262                                          int lowerBound,
263                                          int upperBound) {
264         if ((value < lowerBound) || (value > upperBound)) {
265             throw new IllegalFieldValueException
266                 (fieldType, Integer.valueOf(value),
267                  Integer.valueOf(lowerBound), Integer.
                       valueOf(upperBound));
```

```
268         }
269     }
270
271     /**
272      * Verify that input values are within specified bounds
             .
273      *
274      * @param value  the value to check
275      * @param lowerBound  the lower bound allowed for value
276      * @param upperBound  the upper bound allowed for value
277      * @throws IllegalFieldValueException if value is not
             in the
278      *  specified bounds
279      */
280     public static void verifyValueBounds(String fieldName,
281                                          int value,
282                                          int lowerBound,
283                                          int upperBound) {
284         if ((value < lowerBound) || (value > upperBound)) {
285             throw new IllegalFieldValueException
286                 (fieldName, Integer.valueOf(value),
287                  Integer.valueOf(lowerBound), Integer.
                        valueOf(upperBound));
288         }
289     }
290
291     /**
292      * Utility method used by addWrapField implementations
             to ensure
293      * the new value lies within the field's legal value
             range.
294      *
295      * @param currentValue the current value of the data,
             which may
296      *  lie outside
297      * the wrapped value range
298      * @param wrapValue  the value to add to current value
             before
299      *  wrapping.  This may be negative.
300      * @param minValue the wrap range minimum value.
301      * @param maxValue the wrap range maximum value.  This
             must be
302      *  greater than minValue (checked by the method).
303      * @return the wrapped value
304      * @throws IllegalArgumentException if minValue is
             greater
```

```
305         *  than or equal to maxValue
306         */
307        public static int getWrappedValue(int currentValue, int
               wrapValue,
308                                          int minValue, int
                                                maxValue) {
309            return getWrappedValue(currentValue + wrapValue,
                   minValue, maxValue);
310        }
311
312        /**
313         * Utility method that ensures the given value lies
               within the
314         * field's legal value range.
315         *
316         * @param value  the value to fit into the wrapped
               value range
317         * @param minValue the wrap range minimum value.
318         * @param maxValue the wrap range maximum value.  This
               must be
319         * greater than minValue (checked by the method).
320         * @return the wrapped value
321         * @throws IllegalArgumentException if minValue is
               greater
322         *  than or equal to maxValue
323         */
324        public static int getWrappedValue(int value, int
               minValue, int maxValue) {
325            if (minValue >= maxValue) {
326                throw new IllegalArgumentException("MIN > MAX")
                       ;
327            }
328
329            int wrapRange = maxValue - minValue + 1;
330            value -= minValue;
331
332            if (value >= 0) {
333                return (value % wrapRange) + minValue;
334            }
335
336            int remByRange = (-value) % wrapRange;
337
338            if (remByRange == 0) {
339                return 0 + minValue;
340            }
341            return (wrapRange - remByRange) + minValue;
```

```
342     }
343
344
345     //
           -----------------------------------------------------------------
346     /**
347      * Compares two objects as equals handling null.
348      *
349      * @param object1  the first object
350      * @param object2  the second object
351      * @return true if equal
352      * @since 1.4
353      */
354     public static boolean equals(Object object1, Object
           object2) {
355         if (object1 == object2) {
356             return true;
357         }
358         if (object1 == null || object2 == null) {
359             return false;
360         }
361         return object1.equals(object2);
362     }
363 }
```

E.6.3 Defects-Sources/Time-15/FieldUtils-bugged-B.java

```
1  /*
2   *  Copyright 2001-2005 Stephen Colebourne
3   *
4   *  Licensed under the Apache License, Version 2.0 (the "
       License");
5   *  you may not use this file except in compliance with the
        License.
6   *  You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  *  Unless required by applicable law or agreed to in
       writing, software
11  *  distributed under the License is distributed on an "AS
       IS" BASIS,
12  *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
       express or implied.
13  *  See the License for the specific language governing
       permissions and
14  *  limitations under the License.
15  */
16 package org.joda.time.field;
17 
18 import org.joda.time.DateTimeField;
19 import org.joda.time.DateTimeFieldType;
20 import org.joda.time.IllegalFieldValueException;
21 
22 /**
23  * General utilities that don't fit elsewhere.
24  * <p>
25  * FieldUtils is thread-safe and immutable.
26  *
27  * @author Stephen Colebourne
28  * @since 1.0
29  */
30 public class FieldUtils
31 {
32 
33     /**
34      * Restricted constructor.
35      */
36     private FieldUtils()
37     {
38         super();
```

```
39      }
40
41      //
           -----------------------------------------------------------------
42      /**
43       * Negates the input throwing an exception if it can't
            negate it.
44       *
45       * @param value  the value to negate
46       * @return the negated value
47       * @throws ArithmeticException if the value is Integer.
            MIN_VALUE
48       * @since 1.1
49       */
50      public static int safeNegate(int value)
51      {
52          if (value == Integer.MIN_VALUE)
53          {
54              throw new ArithmeticException("Integer.
                   MIN_VALUE cannot be negated");
55          }
56          return -value;
57      }
58
59      /**
60       * Add two values throwing an exception if overflow
            occurs.
61       *
62       * @param val1  the first value
63       * @param val2  the second value
64       * @return the new total
65       * @throws ArithmeticException if the value is too big
            or too small
66       */
67      public static int safeAdd(int val1, int val2)
68      {
69          int sum = val1 + val2;
70          // If there is a sign change, but the two values
               have the same sign...
71          if ((val1 ^ sum) < 0 && (val1 ^ val2) >= 0)
72          {
73              throw new ArithmeticException
74                  ("The calculation caused an overflow: " +
                       val1 + " + " + val2);
75          }
```

```
76          return sum;
77      }
78
79      /**
80       * Add two values throwing an exception if overflow
             occurs.
81       *
82       * @param val1  the first value
83       * @param val2  the second value
84       * @return the new total
85       * @throws ArithmeticException if the value is too big
             or too small
86       */
87      public static long safeAdd(long val1, long val2)
88      {
89          long sum = val1 + val2;
90          // If there is a sign change, but the two values
                have the same sign...
91          if ((val1 ^ sum) < 0 && (val1 ^ val2) >= 0)
92          {
93              throw new ArithmeticException
94                  ("The calculation caused an overflow: " +
                       val1 + " + " + val2);
95          }
96          return sum;
97      }
98
99      /**
100      * Subtracts two values throwing an exception if
             overflow occurs.
101      *
102      * @param val1  the first value, to be taken away from
103      * @param val2  the second value, the amount to take
             away
104      * @return the new total
105      * @throws ArithmeticException if the value is too big
             or too small
106      */
107     public static long safeSubtract(long val1, long val2)
108     {
109         long diff = val1 - val2;
110         // If there is a sign change, but the two values
                have different signs...
111         if ((val1 ^ diff) < 0 && (val1 ^ val2) < 0)
112         {
113             throw new ArithmeticException
```

```
114                 ("The calculation caused an overflow: " +
                       val1 + " - " + val2);
115         }
116         return diff;
117     }
118 
119     /**
120      * Multiply two values throwing an exception if
           overflow occurs.
121      *
122      * @param val1  the first value
123      * @param val2  the second value
124      * @return the new total
125      * @throws ArithmeticException if the value is too big
           or too small
126      * @since 1.2
127      */
128     public static int safeMultiply(int val1, int val2)
129     {
130         long total = (long) val1 * (long) val2;
131         if (total < Integer.MIN_VALUE || total > Integer.
              MAX_VALUE)
132         {
133           throw new ArithmeticException("Multiplication
                overflows an int: " + val1 + " * " + val2);
134         }
135         return (int) total;
136     }
137 
138     /**
139      * Multiply two values throwing an exception if
           overflow occurs.
140      *
141      * @param val1  the first value
142      * @param val2  the second value
143      * @return the new total
144      * @throws ArithmeticException if the value is too big
           or too small
145      * @since 1.2
146      */
147     public static long safeMultiply(long val1, int val2)
148     {
149         switch (val2)
150         {
151             case -1:
152                 return -val1;
```

```
153             case 0:
154                 return 0L;
155             case 1:
156                 return val1;
157         }
158         long total = val1 * val2;
159         if (total / val2 != val1)
160         {
161           throw new ArithmeticException("Multiplication
                overflows a long: " + val1 + " * " + val2);
162         }
163         return total;
164     }
165 
166     /**
167      * Multiply two values throwing an exception if
           overflow occurs.
168      *
169      * @param val1  the first value
170      * @param val2  the second value
171      * @return the new total
172      * @throws ArithmeticException if the value is too big
           or too small
173      */
174     public static long safeMultiply(long val1, long val2)
175     {
176         if (val2 == 1)
177         {
178             return val1;
179         }
180         if (val1 == 1)
181         {
182             return val2;
183         }
184         if (val1 == 0 || val2 == 0)
185         {
186             return 0;
187         }
188         long total = val1 * val2;
189         if (total / val2 != val1 || val1 == Long.MIN_VALUE
              && val2 == -1
190              || val2 == Long.MIN_VALUE && val1 == -1)
191         {
192             throw new ArithmeticException("Multiplication
                  overflows a long: "
193                  + val1 + " * " + val2);
```

```
194         }
195         return total;
196     }
197 
198     /**
199      * Casts to an int throwing an exception if overflow
            occurs.
200      *
201      * @param value  the value
202      * @return the value as an int
203      * @throws ArithmeticException if the value is too big
            or too small
204      */
205     public static int safeToInt(long value)
206     {
207         if (Integer.MIN_VALUE <= value && value <= Integer.
               MAX_VALUE)
208         {
209             return (int) value;
210         }
211         throw new ArithmeticException("Value cannot fit in
               an int: " + value);
212     }
213 
214     /**
215      * Multiply two values to return an int throwing an
            exception if overflow occurs.
216      *
217      * @param val1  the first value
218      * @param val2  the second value
219      * @return the new total
220      * @throws ArithmeticException if the value is too big
            or too small
221      */
222     public static int safeMultiplyToInt(long val1, long
           val2)
223     {
224         long val = FieldUtils.safeMultiply(val1, val2);
225         return FieldUtils.safeToInt(val);
226     }
227 
228     //
           -----------------------------------------------------------------
229     /**
230      * Verify that input values are within specified bounds
```

```
        .
231      *
232      * @param value  the value to check
233      * @param lowerBound  the lower bound allowed for value
234      * @param upperBound  the upper bound allowed for value
235      * @throws IllegalFieldValueException if value is not
             in the specified bounds
236      */
237     public static void verifyValueBounds(DateTimeField
            field,
238                                          int value, int
                                                lowerBound, int
                                                 upperBound)
239     {
240         if ((value < lowerBound) || (value > upperBound))
241         {
242             throw new IllegalFieldValueException
243                 (field.getType(), Integer.valueOf(value),
244                  Integer.valueOf(lowerBound), Integer.
                        valueOf(upperBound));
245         }
246     }
247
248     /**
249      * Verify that input values are within specified bounds
             .
250      *
251      * @param value  the value to check
252      * @param lowerBound  the lower bound allowed for value
253      * @param upperBound  the upper bound allowed for value
254      * @throws IllegalFieldValueException if value is not
             in the specified bounds
255      * @since 1.1
256      */
257     public static void verifyValueBounds(DateTimeFieldType
            fieldType,
258                                          int value, int
                                                lowerBound, int
                                                 upperBound)
259     {
260         if ((value < lowerBound) || (value > upperBound))
261         {
262             throw new IllegalFieldValueException
263                 (fieldType, Integer.valueOf(value),
264                  Integer.valueOf(lowerBound), Integer.
                        valueOf(upperBound));
```

```
265         }
266     }
267 
268     /**
269      * Verify that input values are within specified bounds
            .
270      *
271      * @param value  the value to check
272      * @param lowerBound  the lower bound allowed for value
273      * @param upperBound  the upper bound allowed for value
274      * @throws IllegalFieldValueException if value is not
            in the specified bounds
275      */
276     public static void verifyValueBounds(String fieldName,
277                                          int value, int
                                                lowerBound, int
                                                 upperBound)
278     {
279         if ((value < lowerBound) || (value > upperBound))
280         {
281             throw new IllegalFieldValueException
282                 (fieldName, Integer.valueOf(value),
283                  Integer.valueOf(lowerBound), Integer.
                        valueOf(upperBound));
284         }
285     }
286 
287     /**
288      * Utility method used by addWrapField implementations
            to ensure the new
289      * value lies within the field's legal value range.
290      *
291      * @param currentValue the current value of the data,
            which may lie outside
292      * the wrapped value range
293      * @param wrapValue  the value to add to current value
            before
294      *  wrapping.  This may be negative.
295      * @param minValue the wrap range minimum value.
296      * @param maxValue the wrap range maximum value.  This
            must be
297      *  greater than minValue (checked by the method).
298      * @return the wrapped value
299      * @throws IllegalArgumentException if minValue is
            greater
300      *  than or equal to maxValue
```

```
301         */
302         public static int getWrappedValue(int currentValue, int
                wrapValue,
303                                           int minValue, int
                                              maxValue)
304         {
305             return getWrappedValue(currentValue + wrapValue,
                   minValue, maxValue);
306         }
307
308         /**
309          * Utility method that ensures the given value lies
                within the field's
310          * legal value range.
311          *
312          * @param value  the value to fit into the wrapped
                value range
313          * @param minValue the wrap range minimum value.
314          * @param maxValue the wrap range maximum value.  This
                must be
315          *  greater than minValue (checked by the method).
316          * @return the wrapped value
317          * @throws IllegalArgumentException if minValue is
                greater
318          *  than or equal to maxValue
319          */
320         public static int getWrappedValue(int value, int
               minValue, int maxValue)
321         {
322             if (minValue >= maxValue)
323             {
324                 throw new IllegalArgumentException("MIN > MAX")
                       ;
325             }
326
327             int wrapRange = maxValue - minValue + 1;
328             value -= minValue;
329
330             if (value >= 0)
331             {
332                 return (value % wrapRange) + minValue;
333             }
334
335             int remByRange = (-value) % wrapRange;
336
337             if (remByRange == 0)
```

```
338         {
339             return 0 + minValue;
340         }
341         return (wrapRange - remByRange) + minValue;
342     }
343 
344     //-----------------------------------------------------------------------
345     /**
346      * Compares two objects as equals handling null.
347      *
348      * @param object1  the first object
349      * @param object2  the second object
350      * @return true if equal
351      * @since 1.4
352      */
353     public static boolean equals(Object object1, Object object2)
354     {
355         if (object1 == object2)
356         {
357             return true;
358         }
359         if (object1 == null || object2 == null)
360         {
361             return false;
362         }
363         return object1.equals(object2);
364     }
365 
366 }
```

E.6.4 Defects-Sources/Time-15/FieldUtils-fixed-B.java

```
1  /*
2   *  Copyright 2001-2005 Stephen Colebourne
3   *
4   *  Licensed under the Apache License, Version 2.0 (the "
       License");
5   *  you may not use this file except in compliance with the
        License.
6   *  You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  *  Unless required by applicable law or agreed to in
       writing, software
11  *  distributed under the License is distributed on an "AS
       IS" BASIS,
12  *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
       express or implied.
13  *  See the License for the specific language governing
       permissions and
14  *  limitations under the License.
15  */
16 package org.joda.time.field;
17 
18 import org.joda.time.DateTimeField;
19 import org.joda.time.DateTimeFieldType;
20 import org.joda.time.IllegalFieldValueException;
21 
22 /**
23  * General utilities that don't fit elsewhere.
24  * <p>
25  * FieldUtils is thread-safe and immutable.
26  *
27  * @author Stephen Colebourne
28  * @since 1.0
29  */
30 public class FieldUtils
31 {
32 
33     /**
34      * Restricted constructor.
35      */
36     private FieldUtils()
37     {
38         super();
```

```
39      }
40
41      //
           -----------------------------------------------------------------
42      /**
43       * Negates the input throwing an exception if it can't
            negate it.
44       *
45       * @param value  the value to negate
46       * @return the negated value
47       * @throws ArithmeticException if the value is Integer.
            MIN_VALUE
48       * @since 1.1
49       */
50      public static int safeNegate(int value)
51      {
52          if (value == Integer.MIN_VALUE)
53          {
54              throw new ArithmeticException("Integer.
                   MIN_VALUE cannot be negated");
55          }
56          return -value;
57      }
58
59      /**
60       * Add two values throwing an exception if overflow
            occurs.
61       *
62       * @param val1  the first value
63       * @param val2  the second value
64       * @return the new total
65       * @throws ArithmeticException if the value is too big
            or too small
66       */
67      public static int safeAdd(int val1, int val2)
68      {
69          int sum = val1 + val2;
70          // If there is a sign change, but the two values
               have the same sign...
71          if ((val1 ^ sum) < 0 && (val1 ^ val2) >= 0)
72          {
73              throw new ArithmeticException
74                  ("The calculation caused an overflow: " +
                       val1 + " + " + val2);
75          }
```

```
76          return sum;
77      }
78
79      /**
80       * Add two values throwing an exception if overflow
             occurs.
81       *
82       * @param val1  the first value
83       * @param val2  the second value
84       * @return the new total
85       * @throws ArithmeticException if the value is too big
             or too small
86       */
87      public static long safeAdd(long val1, long val2)
88      {
89          long sum = val1 + val2;
90          // If there is a sign change, but the two values
                have the same sign...
91          if ((val1 ^ sum) < 0 && (val1 ^ val2) >= 0)
92          {
93              throw new ArithmeticException
94                  ("The calculation caused an overflow: " +
                       val1 + " + " + val2);
95          }
96          return sum;
97      }
98
99      /**
100      * Subtracts two values throwing an exception if
             overflow occurs.
101      *
102      * @param val1  the first value, to be taken away from
103      * @param val2  the second value, the amount to take
             away
104      * @return the new total
105      * @throws ArithmeticException if the value is too big
             or too small
106      */
107     public static long safeSubtract(long val1, long val2)
108     {
109         long diff = val1 - val2;
110         // If there is a sign change, but the two values
                have different signs...
111         if ((val1 ^ diff) < 0 && (val1 ^ val2) < 0)
112         {
113             throw new ArithmeticException
```

```
114                 ("The calculation caused an overflow: " +
                       val1 + " - " + val2);
115         }
116         return diff;
117     }
118 
119     /**
120      * Multiply two values throwing an exception if
           overflow occurs.
121      *
122      * @param val1  the first value
123      * @param val2  the second value
124      * @return the new total
125      * @throws ArithmeticException if the value is too big
           or too small
126      * @since 1.2
127      */
128     public static int safeMultiply(int val1, int val2)
129     {
130         long total = (long) val1 * (long) val2;
131         if (total < Integer.MIN_VALUE || total > Integer.
              MAX_VALUE)
132         {
133           throw new ArithmeticException("Multiplication
                overflows an int: " + val1 + " * " + val2);
134         }
135         return (int) total;
136     }
137 
138     /**
139      * Multiply two values throwing an exception if
           overflow occurs.
140      *
141      * @param val1  the first value
142      * @param val2  the second value
143      * @return the new total
144      * @throws ArithmeticException if the value is too big
           or too small
145      * @since 1.2
146      */
147     public static long safeMultiply(long val1, int val2)
148     {
149         switch (val2)
150         {
151             case -1:
152                 if (val1 == Long.MIN_VALUE)
```

```
153                 {
154                     throw new ArithmeticException("
                           Multiplication overflows a long: " +
                            val1 + " * " + val2);
155                 }
156                 return -val1;
157             case 0:
158                 return 0L;
159             case 1:
160                 return val1;
161         }
162         long total = val1 * val2;
163         if (total / val2 != val1)
164         {
165           throw new ArithmeticException("Multiplication
                 overflows a long: " + val1 + " * " + val2);
166         }
167         return total;
168     }
169 
170     /**
171      * Multiply two values throwing an exception if
            overflow occurs.
172      *
173      * @param val1  the first value
174      * @param val2  the second value
175      * @return the new total
176      * @throws ArithmeticException if the value is too big
            or too small
177      */
178     public static long safeMultiply(long val1, long val2)
179     {
180         if (val2 == 1)
181         {
182             return val1;
183         }
184         if (val1 == 1)
185         {
186             return val2;
187         }
188         if (val1 == 0 || val2 == 0)
189         {
190             return 0;
191         }
192         long total = val1 * val2;
193         if (total / val2 != val1 || val1 == Long.MIN_VALUE
```

```
            && val2 == -1
194             || val2 == Long.MIN_VALUE && val1 == -1)
195         {
196             throw new ArithmeticException("Multiplication
                    overflows a long: "
197                 + val1 + " * " + val2);
198         }
199         return total;
200     }
201 
202     /**
203      * Casts to an int throwing an exception if overflow
                occurs.
204      *
205      * @param value  the value
206      * @return the value as an int
207      * @throws ArithmeticException if the value is too big
                or too small
208      */
209     public static int safeToInt(long value)
210     {
211         if (Integer.MIN_VALUE <= value && value <= Integer.
                MAX_VALUE)
212         {
213             return (int) value;
214         }
215         throw new ArithmeticException("Value cannot fit in
                an int: " + value);
216     }
217 
218     /**
219      * Multiply two values to return an int throwing an
                exception if overflow occurs.
220      *
221      * @param val1  the first value
222      * @param val2  the second value
223      * @return the new total
224      * @throws ArithmeticException if the value is too big
                or too small
225      */
226     public static int safeMultiplyToInt(long val1, long
            val2)
227     {
228         long val = FieldUtils.safeMultiply(val1, val2);
229         return FieldUtils.safeToInt(val);
230     }
```

```
231
232     //
           -----------------------------------------------------------------
233     /**
234      * Verify that input values are within specified bounds
           .
235      *
236      * @param value  the value to check
237      * @param lowerBound  the lower bound allowed for value
238      * @param upperBound  the upper bound allowed for value
239      * @throws IllegalFieldValueException if value is not
           in the specified bounds
240      */
241     public static void verifyValueBounds(DateTimeField
           field,
242                                          int value, int
                                               lowerBound, int
                                                upperBound)
243     {
244         if ((value < lowerBound) || (value > upperBound))
245         {
246             throw new IllegalFieldValueException
247                 (field.getType(), Integer.valueOf(value),
248                  Integer.valueOf(lowerBound), Integer.
                       valueOf(upperBound));
249         }
250     }
251
252     /**
253      * Verify that input values are within specified bounds
           .
254      *
255      * @param value  the value to check
256      * @param lowerBound  the lower bound allowed for value
257      * @param upperBound  the upper bound allowed for value
258      * @throws IllegalFieldValueException if value is not
           in the specified bounds
259      * @since 1.1
260      */
261     public static void verifyValueBounds(DateTimeFieldType
           fieldType,
262                                          int value, int
                                               lowerBound, int
                                                upperBound)
263     {
```

```
264         if ((value < lowerBound) || (value > upperBound))
265         {
266             throw new IllegalFieldValueException
267                 (fieldType, Integer.valueOf(value),
268                  Integer.valueOf(lowerBound), Integer.
                        valueOf(upperBound));
269         }
270     }
271
272     /**
273      * Verify that input values are within specified bounds
            .
274      *
275      * @param value  the value to check
276      * @param lowerBound  the lower bound allowed for value
277      * @param upperBound  the upper bound allowed for value
278      * @throws IllegalFieldValueException if value is not
            in the specified bounds
279      */
280     public static void verifyValueBounds(String fieldName,
281                                          int value, int
                                                lowerBound, int
                                                 upperBound)
282     {
283         if ((value < lowerBound) || (value > upperBound))
284         {
285             throw new IllegalFieldValueException
286                 (fieldName, Integer.valueOf(value),
287                  Integer.valueOf(lowerBound), Integer.
                        valueOf(upperBound));
288         }
289     }
290
291     /**
292      * Utility method used by addWrapField implementations
            to ensure the new
293      * value lies within the field's legal value range.
294      *
295      * @param currentValue the current value of the data,
            which may lie outside
296      * the wrapped value range
297      * @param wrapValue  the value to add to current value
            before
298      *  wrapping.  This may be negative.
299      * @param minValue the wrap range minimum value.
300      * @param maxValue the wrap range maximum value.  This
```

```
         must be
301      *  greater than minValue (checked by the method).
302      * @return the wrapped value
303      * @throws IllegalArgumentException if minValue is
             greater
304      *  than or equal to maxValue
305      */
306     public static int getWrappedValue(int currentValue, int
             wrapValue,
307                                       int minValue, int
                                             maxValue)
308     {
309         return getWrappedValue(currentValue + wrapValue,
                minValue, maxValue);
310     }
311
312     /**
313      * Utility method that ensures the given value lies
             within the field's
314      * legal value range.
315      *
316      * @param value  the value to fit into the wrapped
             value range
317      * @param minValue the wrap range minimum value.
318      * @param maxValue the wrap range maximum value.  This
             must be
319      *  greater than minValue (checked by the method).
320      * @return the wrapped value
321      * @throws IllegalArgumentException if minValue is
             greater
322      *  than or equal to maxValue
323      */
324     public static int getWrappedValue(int value, int
            minValue, int maxValue)
325     {
326         if (minValue >= maxValue)
327         {
328             throw new IllegalArgumentException("MIN > MAX")
                    ;
329         }
330
331         int wrapRange = maxValue - minValue + 1;
332         value -= minValue;
333
334         if (value >= 0)
335         {
```

```
336             return (value % wrapRange) + minValue;
337         }
338 
339         int remByRange = (-value) % wrapRange;
340 
341         if (remByRange == 0)
342         {
343             return 0 + minValue;
344         }
345         return (wrapRange - remByRange) + minValue;
346     }
347 
348     //-----------------------------------------------------------------------
349     /**
350      * Compares two objects as equals handling null.
351      *
352      * @param object1  the first object
353      * @param object2  the second object
354      * @return true if equal
355      * @since 1.4
356      */
357     public static boolean equals(Object object1, Object object2)
358     {
359         if (object1 == object2)
360         {
361             return true;
362         }
363         if (object1 == null || object2 == null)
364         {
365             return false;
366         }
367         return object1.equals(object2);
368     }
369 
370 }
```