25.1 Introduction

9

As educators, it’s a joy to write this chapter on what may be the most important pedagogic improvement in Java since its inception more than two decades ago. The Java community—by far the largest programming language community in the world—has grown to more than 10 million developers. But along the way, not much has been done to improve the learning process for novice programmers. That changes dramatically in Java 9 with the introduction of **JShell**—Java’s **REPL (read-evaluate-print loop)**.1

1. We’d like to thank Robert Field at Oracle—the head of the JShell/REPL effort. We interacted with Mr. Field extensively as we developed Chapter 25. He answered our many questions. We reported JShell bugs and made suggestions for improvement.

Instructors have indicated a preference in introductory programming courses for languages with REPLs—and now Java has a rich REPL implementation. And with the new JShell APIs, third parties will build JShell and related interactive-development tools into the major IDEs like Eclipse, IntelliJ, NetBeans and others. Java 9 and JShell are evolving rapidly, so we’ve placed all our Java 9 content online—we’ll keep it up-to-date as Java 9 evolves.

What is JShell?

What's the magic? It's simple. JShell provides a fast and friendly environment that enables you to quickly explore, discover and experiment with Java language features and its extensive libraries. REPLs like the one in JShell have been around for decades. In the 1960s, one of the earliest REPLs made convenient interactive development possible in the LISP programming language. Students of that era, like one of your authors, Harvey Deitel, found it fast and fun to use.

JShell replaces the tedious cycle of editing, compiling and executing with its read-evaluate-print loop. Rather than complete programs, you write **JShell commands** and Java code snippets. When you enter a snippet, JShell *immediately* **reads** it, **evaluates** it and **prints** the results that help you see the effects of your code. Then it **loops** to perform this process again for the next snippet. As you work through Chapter 25's scores of examples and exercises, you'll see how JShell and its instant feedback keep your attention, enhance your performance and speed the learning and software development processes.

Code Comes Alive

As you know, we emphasize the value of the live-code teaching approach in our books, focusing on *complete*, working programs. JShell brings this right down to the individual snippet level. Your code literally comes alive as you enter each line. Of course, you'll still make occasional errors as you enter your snippets. JShell reports compilation errors to

you on a snippet-by-snippet basis. You can use this capability, for example, to test the items in our Common Programming Error tips and see the errors as they occur.

Kinds of Snippets

Snippets can be expressions, individual statements, multi-line statements and larger entities, like methods and classes. JShell supports all but a few Java features, but there are some differences designed to facilitate JShell's explore–discover–experiment capabilities. In JShell, methods do not need to be in classes, expressions and statements do not need to be in methods, and you do not need a `main` method (other differences are in Section 25.14). Eliminating this infrastructure saves you considerable time, especially compared to the lengthy repeated edit, compile and execute cycles of complete programs. And because JShell automatically displays the results of evaluating your expressions and statements, you do not need as many print statements as we use throughout this book's traditional Java code examples.

Discovery with Auto-Completion

We include a detailed treatment of **auto-completion**—a key discovery feature that speeds the coding process. After you

type a portion of a name (class, method, variable, etc.) and press the *Tab* key, JShell completes the name for you or provides a list of all possible names that begin with what you've typed so far. You can then easily display method parameters and even the documentation that describes those methods.

Rapid Prototyping

Professional developers will commonly use JShell for rapid prototyping but not for full-out software development. Once you develop and test a small chunk of code, you can then paste it in to your larger project.

How This Chapter Is Organized

Chapter 25 is optional. For those who want to use JShell, the chapter has been designed as a series of units, paced to certain earlier chapters of the print book. Each unit begins with a statement like: “This section may be read after Chapter 2.” So you’d begin by reading through Chapter 2, then read the corresponding section of this chapter—and similarly for subsequent chapters.

The Chapter 2 JShell Exercises

As you work your way through this chapter, execute each snippet and command in JShell to confirm that the features work as advertised. Sections 25.3–25.4 are designed to be read after Chapter 2. Once you read these sections, we recommend that you do Chapter 25's dozens of self-review exercises. JShell encourages you to "learn by doing," so the exercises have you write and test code snippets that exercise many of Chapter 2's Java features.

The self-review exercises are small and to the point, and the answers are provided to help you quickly get comfortable with JShell's capabilities. When you're done you'll have a great sense of what JShell is all about. Please tell us what you think of this new Java tool. Thanks!

Instead of rambling on about the advantages of JShell, we're going to let JShell itself convince you. If you have any questions as you work through the following examples and exercises, just write to us at `deitel@deitel.com` and we'll always respond promptly.