A Friendly Face for Eclipse

Charles Reis
Rice University
6100 S. Main St.
Houston TX 77005
creis@alumni.rice.edu

Robert Cartwright
Rice University
6100 S. Main St.
Houston TX 77005
cork@cs.rice.edu

Abstract

Eclipse is a powerful integrated development environment (IDE) for Java[1] targeted at professional software developers. However, Eclipse is poorly suited for use in introductory computing education because the complexity of its interface and the associated computing environment can overwhelm beginners. In contrast, DrJava is a friendly, highly interactive IDE targeted at teaching Java to beginners. DrJava has a simple interface consisting of a Definitions pane for entering and editing program text and an Interactions pane for evaluating arbitrary Java statements and expressions given the program in the Definitions pane. This interface frees students from the complication of defining `main` methods for their programs and encourages them to explore the Java language by conducting simple experiments.

We have developed a plug-in for Eclipse, based largely on the existing DrJava code base, that provides an Interactions pane to Eclipse with precisely the same capabilities as the Interactions pane in DrJava, along with a simplified user interface. With this plug-in, Eclipse becomes a suitable vehicle for teaching introductory programming, enabling instructional programs to use the same IDE for all levels of the programming curriculum. In addition, it provides professional developers with a convenient mechanism for interactively evaluating arbitrary program text during program development—a common feature of programming interfaces for functional languages like Scheme and ML.

1 Introduction

Despite Java's growing popularity as a language for teaching programming [3], it has some characteristics that make teaching programming concepts unnecessarily difficult. Although Java syntax is familiar territory for experienced C/C++ programmers, it is quite challenging for beginners to learn. In addition, Java I/O and the compile/execute command line interface are complex and intimidating for students with little programming experience. Students who are forced to learn Java using a conventional text editor and command line execution interface are often overwhelmed by the mechanics of writing and running a program and have difficulty focusing on learning how to design object-oriented programs. For these reasons, many instructors elect to use an integrated development environment (IDE) for Java in an effort to relieve students of some of the clerical burdens involved in writing Java programs.

2 IDEs in the Classroom

In the context of computing education, there are several fundamental requirements that an IDE must satisfy. First, the IDE must assist students in writing correct syntax in a language in which they may not be very proficient. Many IDEs provide this support by giving visual cues that highlight a program's structure. Some, like Eclipse, perform full incremental parsing to provide these cues. Second, the IDE must provide a simple interface to the language compiler (such as a "compile" button) and flag any syntax errors by highlighting the offending lines of source code. Third, the IDE must enable students to execute programs easily without using a command line interface. Most IDEs provide a mechanism for running a program provided that it contains a proper `main` method.

While most Java IDEs meet these basic requirements, there are three additional requirements for IDEs used in *introductory* programming courses.

- First, the user interface should be simple and unintimidating. Professional IDEs like Eclipse generally fare poorly in this regard because they are designed to provide all of the features that professional developers expect. This array of features is bewilderingly complex for novices. For these students, the importance of a simple, intuitive user interface cannot be overstated. Even well-designed professional tools presume a reasonable grasp of the language, placing their

[1] Java™ is a trademark of Sun Microsystems, Inc.

target audience well above the introductory level.

- Second, the IDE should provide simple mechanisms for working around complications in the Java language that are pedagogic distractions. The two most prominent such complications are the `public static void main(String[] args)` convention for starting the execution of a Java program and the syntax required for I/O operations. The `main` convention is painful to teach to beginners because it forces a discussion of access modifiers (`public`), `static` methods, and arrays before students can execute even the most trivial program, e.g. "Hello World". Similarly, console input is a painful mechanism for specifying the input values for a computation. The Java language does not provide a simple external interface for creating objects and invoking methods on them. Processing console input to extract argument values for a method is far more difficult than simply writing a Java method invocation with constant arguments.

- Finally, a pedagogic IDE should be reasonably lightweight so that it executes responsively on older, less capable hardware. In contrast to professional software developers, students in introductory programming courses often do not have access to state-of-art personal computers.

While the Eclipse platform is a good match for intermediate and advanced programming courses, it does not satisfy any of the three requirements for use in introductory programming courses listed above. The default Eclipse perspective for Java programs presents students with a very complex interface containing no fewer than 10 menus, 6 visible or available panes, and 4 poorly labeled toolbars, each with unconventional context menus. Eclipse has a steep learning curve, due not only to the abundance of panes and cryptic toolbar buttons, but also to the potentially unfamiliar concepts of perspectives, plug-ins, and projects. Moreover, on many student machines, Eclipse performs sluggishly, particularly on large programs such as "case studies" in many introductory courses.

3 A Simple, Interactive IDE

DrJava [1] is a small IDE designed specifically to address the difficulties of teaching Java to students at the introductory level. The DrJava interface primarily consists of two panes: a Definitions pane used to enter program text and an Interactions pane used to evaluate arbitrary statements and expressions in the context of the classes defined in the Definitions pane. (See Figure 1 for a screenshot of DrJava.) In essence, DrJava converts Java from a "batch-oriented" (command line based) language to a reactive one comparable in interactive flexibility to (mostly) functional languages like Scheme and ML.

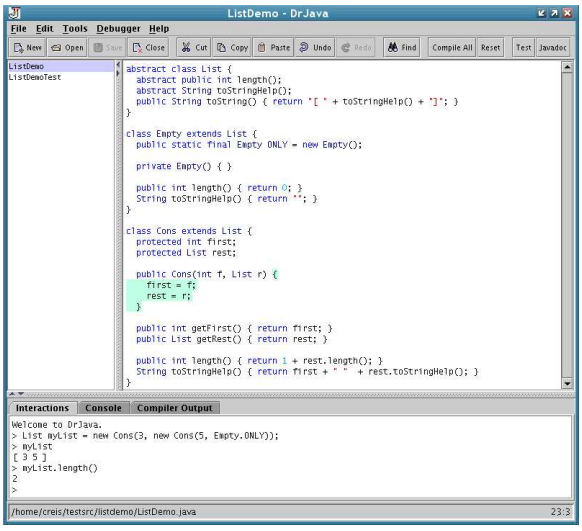

Figure 1: DrJava

As a pedagogic IDE, DrJava's most important benefits are its simplicity and its interactive interface. The user interface is designed to be accessible to beginners, with clearly labeled buttons and few distractions in a simple graphical layout. Almost all of DrJava's features and integrated tools revolve around its Interactions pane, which is presented as a console-style "read-eval-print loop" (REPL). This form of interface dates back at least to early Lisp implementations [12] and has been incorporated in many "interactive" programming languages (e.g., Lisp, Scheme, ML). The particular form of REPL used in DrJava, where the current "source program" is maintained in a separate pane, was pioneered in an earlier pedagogic environment called DrScheme [5]. The DrJava Interactions pane uses DynamicJava [8] to interpret interactions, leveraging reflection and dynamic class loading for efficiency on par with command line execution.

The Interactions pane provides students with a very simple interface for executing Java programs—eliminating the need for `public static void main(...)`—and for experimenting with the behavior of various parts of the programs that they write. Just as unit testing focuses on testing individual methods, the Interactions pane enables students to focus on executing individual methods in their programs to see how they behave. The Interactions pane also provides a simple framework for exploring the behavior of Java libraries and for conducting computational experiments at breakpoints during debugging. This tool can greatly benefit instructors as well, who can easily demonstrate new concepts and language features

in the Interactions pane in the classroom.

DrJava integrates all of the tools that are essential to Java software development: Java compilers (which are plug-ins), a unit testing framework known as JUnit, a source-level debugger, and the Javadoc documentation tool. The debugger is tightly integrated with the Interactions pane [11]. Users can set breakpoints in source code to suspend the evaluation of any method call from the Interactions pane. During such a pause in program execution, users can interact with the state of the program directly in the Interactions pane, calling any methods and querying or modifying any values that are in scope. This is an intuitive and familiar interface to understand and debug existing code.

Unlike other pedagogic IDEs, which are often limited in scope to small programs, DrJava also scales to developing large production programs, including itself. For the past year, DrJava has been developed by a team of students using DrJava almost exclusively. On such a code base, consisting of 40,000 lines of source code plus the DynamicJava interpreter, DrJava still performs very responsively on machines that are too slow to run Eclipse, such as a 500 MHz G3 Apple iBook.

In contrast to Eclipse, DrJava is restricted in scope to keep its interface simple and uniform and to ensure that it runs on slower machines. It does not support a plug-in architecture, but the code base is sufficiently compact that it is easy for advanced undergraduates to extend and modify the code base [2]. Despite DrJava's effectiveness on sizable projects, the environment lacks built-in refactoring tools, which can be useful in many software courses.

4 DrJava Plug-in For Eclipse

To support the use of Eclipse in introductory programming classes, as well as to facilitate an easier transition from DrJava to a full-featured professional IDE, it is desirable to support DrJava's simple and interactive interface within Eclipse. Fortunately, Eclipse's plug-in architecture makes these modifications to Eclipse easy to achieve, while DrJava's modular design enables significant code re-use in the implementation of such a plug-in.

The DrJava plug-in for Eclipse simplifies the Eclipse user interface for beginning programmers, while also adding an Interactions pane to Eclipse to facilitate interactive program development via DrJava's own REPL. After installing the plug-in, users can select a "DrJava Perspective" in Eclipse to obtain a graphical layout that is visually similar to DrJava, complete with an Interactions pane (see Figure 2 for a screenshot). This Interactions pane is closely integrated with the user's open projects, as the classpath entries for each

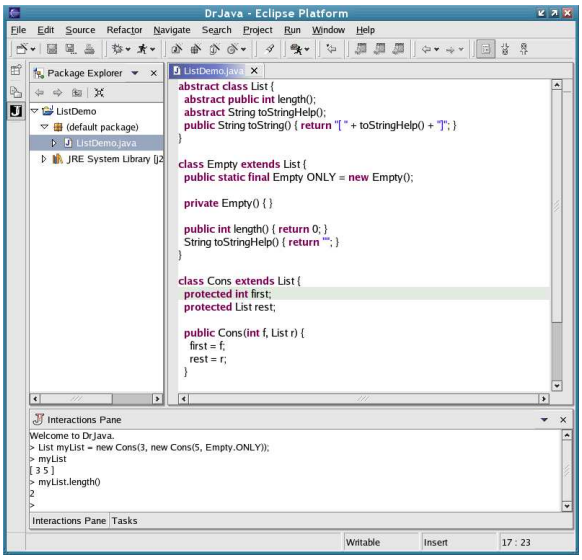

Figure 2: The DrJava Plug-in for Eclipse

project are automatically in scope, and changes to open class files result in a prompt warning the user to reset his interactions session to load updated class files.

This plug-in is still in active development. The current release supports a simplified user interface and a fully functional Interactions pane. The full release of the plug-in will also integrate the Eclipse debugger with the Interactions pane, providing Eclipse with the same interactive debugging capabilities as DrJava. By facilitating a simple and interactive interface for development and debugging, the DrJava plug-in makes Eclipse a very attractive and capable environment for teaching Java programming skills at any level, provided that students have access to machines capable of running Eclipse responsively.

4.1 Target Audience

We expect the DrJava plug-in to serve three different constituencies.

- First, many students who learn to program in Java using DrJava grow accustomed to the convenience of an Interactions pane and resist using professional IDEs in more advanced courses because they do not support program interactions. With the DrJava plug-in, Eclipse supports precisely the same interactive behavior as DrJava.
- Second, some Computer Science programs designate a single IDE as the supported programming platform for all courses using Java. This policy creates a potential conflict between the interests of beginning students and advanced students since it is difficult to find an IDE that works well in both contexts. With

the DrJava plug-in, we believe that Eclipse is well-suited to both groups.

- Third, like the advocates of functional programming environments, we believe that the ability to evaluate arbitrary program text in a "read-eval-print loop" (REPL) is very helpful during program development by professional software developers. As a result, we anticipate that some professional developers will use the DrJava plug-in for Eclipse.

4.2 Plug-in Implementation

Code re-use was a major goal in the design and implementation of the DrJava plug-in for Eclipse. We wanted the plug-in to use as much code as possible directly from the DrJava code base to reduce the work required to develop the plug-in and to minimize the introduction of new errors. Our development team emphasizes the use of design patterns [7] and follows the Extreme Programming practices [9] in software development, facilitating the refactoring and re-use of the DrJava code base.

DrJava is designed according to the *Model-View-Controller* design pattern, separating the logic for the core application (the model) from the components of the user interface. Nevertheless, prior to our writing the plug-in for Eclipse, the logic for DrJava's Interactions pane was still tightly coupled with the rest of the DrJava model. Fortunately, Extreme Programming methodology facilitates and encourages program *refactoring* to improve code and adapt to new requirements as they arise [6]. This tight coupling between the Interactions pane and the rest of the DrJava model interfered with re-using the code for the Interactions pane for the Eclipse plug-in. Thus, we refactored the design of Interactions pane and the supporting code that enables it to run safely in its own Java Virtual Machine, allowing these components to be re-used in the Eclipse plug-in independently of the rest of DrJava.

The use of the *Model-View-Controller* pattern in DrJava also helped us overcome a potential incompatibility in windowing toolkits between DrJava and Eclipse. Like most Java GUI applications, DrJava uses the Swing toolkit to implement its graphical components. Eclipse, on the other hand, uses a new SWT toolkit to provide an interface more closely integrated with individual platforms. Since DrJava's model is independent of its Swing view components, the plug-in could theoretically just provide alternative SWT view components that could be displayed in Eclipse. Unfortunately, there was a catch—the Swing toolkit provides some model-oriented classes, such as the `Document` class used by graphical text components such as the Interactions pane. DrJava's model maintained the contents of the Interactions pane using a Swing `Document`, which was incompatible with the SWT view components used in the plug-in. Rather than have duplicate representations of the document in the plug-in, we employed the *Adapter* design pattern to create a toolkit-independent document. In this way, DrJava's model uses a `DocumentAdapter` interface, which abstracts the differences between Swing and SWT and can be implemented using either Swing or SWT components. Thus, the DrJava IDE continues to use a Swing `Document` for its Interactions pane, while the Eclipse plug-in can use an SWT `StyledText` widget.

After we performed these refactoring transformations to the DrJava code base, very little new code was required to implement the Eclipse plug-in. In fact, only eight extra classes comprise the current incremental release of the plug-in, which provides a fully functional Interactions pane and a simplified user interface. We are currently extending the plug-in to integrate Eclipse's debugging features with the Interactions pane. To support this integration, we are refactoring the implementation of the DrJava debugger and its interface to the Interactions pane to support further code re-use. This technique ensures that any new features or bug fixes related to the Interactions pane in DrJava are immediately available in the Eclipse plug-in as well.

5 Related Work

BlueJ [10] is a pedagogic IDE that also strives to eliminate the dependence on Java's `main` method and console I/O. BlueJ uses class diagrams and a graphical "workbench" to allow students to visually interact with their programs. While this interface is effective for graphically representing some object-oriented program designs, it does not scale to larger projects or computations. As a result, BlueJ is limited to use in introductory courses and its interface is sufficiently unconventional that it does not prepare students to use environments suitable for more advanced coursework.

Eclipse itself provides a Java Scrapbook feature to promote interactive evaluation of Java code, but its interface is less intuitive and less flexible than the REPL in DrJava's Interactions pane. The Java Scrapbook interface relies on using the mouse to select a region of text in a document and evaluate it using a context menu. Any results or corresponding errors are inserted into the document in editable form, adjacent to the statements or expressions themselves. Because the same block of text can be repeatedly modified and evaluated, the current program state may be difficult to reconstruct.[2] The

[2]This is a weakness shared by a conventional "read-eval-print-loop" where the current program state is assembled using "load" statements that are executed by the REPL. DrJava maintains a separate Definitions pane to eliminate this problem. The current program state is generated sim-

scrapbook does not maintain a source language description of that state. In contrast, DrJava's Interactions pane maintains a history of the interactions which are used to build up state, with clear distinctions between interactions, results, and errors.

6 Conclusion

DrJava provides a simple, interactive user interface that addresses the challenges involved in using an integrated development environment for Java in introductory programming courses. Eclipse, on the other hand, provides a wide collection of sophisticated features and tools suitable for production programming and advanced programming courses. We have developed an Eclipse plug-in supporting essentially the same interactive user interface as DrJava, making Eclipse suitable for use by beginning students and providing advanced users with a more convenient interface for performing program interactions. In the process, we have shown how design patterns and Extreme Programming practices facilitate the re-use of the DrJava code base.

7 About the Authors

Charles Reis received his Masters Degree from Rice University in May 2003, working with the Java Programming Languages and Technology research group. He has worked on DrJava for two years, serving as its second lead developer.

Robert "Corky" Cartwright is a Professor of Computer Science at Rice University, where he has been a member of the faculty since 1980 including a brief stint as department chair. His principal research interests are programming languages and software engineering, with a focus in recent years on object-oriented programming in Java. He is the director of the Java Programming Languages and Technology research group at Rice University which is developing DrJava and other tools to support Java education and research.

References

[1] E. Allen, R. Cartwright, B. Stoler. *DrJava: A Lightweight Pedagogic Environment for Java. SIGCSE 2002*, March 2002.

[2] E. Allen, R. Cartwright, C. Reis. *Production Programming in the Classroom. SIGCSE 2003*, February 2003.

[3] O. Astrachan, R. Cartwright, G. Chapman, D. Gries, C. Horstmann, R. Kick, F. Trees, H. Walker, U. Wolz. *Recommendations of the AP Computer Science Ad Hoc Committee*, October 2000. (URL: `http://apcentral.collegeboard.com/repository/ap01.pdf.ad_7908.pdf`)

[4] G. Bracha, M. Odersky, D. Stoutamire, P. Wadler. *Making the future safe for the past: adding genericity to the Java programming language. OOPSLA '98*, October 1998.

[5] R. Findler, C. Flanagan, M. Flatt, S. Krishnamurthi, M. Felleisen. *DrScheme: A pedagogic programming environment for Scheme.* In *International Symposium on Programming Languages: Implementations, Logics, and Programs*, 1997, 369-388.

[6] M. Fowler, K. Beck, J. Brant. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, Mass. 1995.

[8] S. Hillion. "DynamicJava." (URL: `http://koala.ilog.fr/djava/`)

[9] R. Jefferies, A. Anderson, C. Hendrickson. *Extreme Programming Installed.* Addison-Wesley, 2001.

[10] M. Kölling, A. Patterson, B. Quig, J. Rosenberg. "BlueJ, The Interactive Java Environment." (URL: `http://www.bluej.org`)

[11] C. Reis. *A Pedagogic Programming Environment for Java that Scales to Production Programming.* Master's thesis, Rice University, April 2003.

[12] E. Sandewall. *Programming in an interactive environment: the "Lisp" experience.* In *Computing Surveys*, 10(1), March 1978, 35-71.

ply by compiling the defined program and executing the sequence of statements and expressions in the interactions history.