Taming a Professional IDE for the Classroom*

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

An important question that must be addressed in a coherent computing curriculum is which programming environments to use across the curriculum. For Java, currently the most widely used language in computing education, a wide variety of professional integrated development environments (IDEs) are available—including the increasingly popular, open-source Eclipse environment. Professional IDEs for Java work well in advanced courses, but they are poorly matched to introductory courses because they deluge beginning students with a complex array of features. In addition, professional IDEs fail to shield students from distracting complications like the Java command line interface and Java console I/O. For this reason, many educators favor using a "pedagogic" IDE such as BlueJ or DrJava to provide a gentle introduction to the mechanics of Java programming.

To eliminate the gap between pedagogic and professional IDEs for Java, we have developed a plug-in for Eclipse that supports exactly the same programming interface as DrJava. It features an Interactions pane for evaluating program statements and expressions "on the fly" as in DrJava. With this plug-in, Eclipse is accessible to beginning programmers. In this configuration, Eclipse is a suitable vehicle for teaching introductory programming–enabling Eclipse to be used across the entire spectrum of the computing curriculum.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments

Keywords

DrJava, Eclipse, plug-in, Interactions pane

1. Introduction

Despite the growing popularity of Java as an instructional language [3], it has some characteristics that make teaching programming concepts unnecessarily difficult. Although Java syntax is familiar territory for experienced C/C++ programmers, it can be quite challenging for beginners to learn. In addition, Java console I/O and the Java command line interface are complex and intimidating for students with little programming experience. Students who try to learn Java using a conventional text editor and command line interface are often overwhelmed by the mechanics of writing and running a program. As a result, they 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. Use of IDEs in the Classroom

Professional IDEs are designed to help software developers write programs more quickly and produce better quality code. They help reduce the incidence of clerical errors by providing a text editor that analyzes the syntax of a program and highlights its major syntactic features. Nearly all IDEs support the automatic indenting of program text according to the program's nesting structure. Some IDEs like Eclipse perform incremental parsing, identifying any portion of the program text that is ungrammatical as each keystroke is typed. IDEs also simplify the process of compiling and executing new versions of a program as it is developed by providing "compile" and "run" buttons in place of a conventional command line interface. When a program faults during execution, most IDEs provide an integrated source level debugger that helps the programmer selectively trace and interrupt the execution of the program to track down the source of the error. With the growing emphasis on unit testing and test-driven design, some environments have integrated support for unit testing using frameworks like JUnit [4].

All of these basic features of IDEs, with the possible exception of source level debugging, can be helpful in teaching *introductory* programming. But introductory programming courses impose three special requirements on IDEs that tend to conflict with the requirements of professional software developers.

- First, the programming interface must be simple and intuitive. Professional IDEs generally fail this *simplicity* test because they are designed to provide all of the features that might be helpful to professional developers, creating a complex interface that is bewildering to novices. For beginning students, providing a simple, intuitive user interface is far more important than

*This research has been partially supported by IBM Corporation, the Texas Advanced Technology Program, the National Science Foundation, and Sun Microsystems, Inc.

offering every conceivable feature. Even well-designed professional tools presume a reasonable grasp of the language, placing their 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 console I/O operations. The `main` convention is painful to teach to beginners because it forces a discussion of access modifiers (e.g., `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 public 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. Indeed, reading input via a `BufferedReader` even requires an explanation of checked exceptions, introducing undue complexity for such simple concepts.

- Finally, a pedagogic IDE should be fairly 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 professional IDEs may be a good match for intermediate and advanced programming courses, they do not satisfy the three requirements for introductory programming courses listed above. To cite one example, the default Eclipse perspective for Java programs presents students with an interface containing no less than 10 menus, 6 visible or available panes, and 4 poorly labeled toolbars, each with unconventional context menus. Professional IDEs like Eclipse have 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, professional IDEs like Eclipse perform sluggishly, particularly on large programs such as the "case studies" used in many introductory courses [17].

3. Existing Pedagogic IDEs

Pedagogic IDEs like BlueJ [12] and DrJava [1] have been developed specifically to address the requirements for teaching introductory programming identified above. They are much smaller and simpler than professional IDEs, and they provide an interface for performing computations and producing output that bypasses the Java command line interface and the definition of a method

`public static void main(String[] args)`

for beginning program execution. BlueJ provides a visually-oriented object workbench and DrJava provides an Interactions pane for evaluating statements and expressions and printing their results.

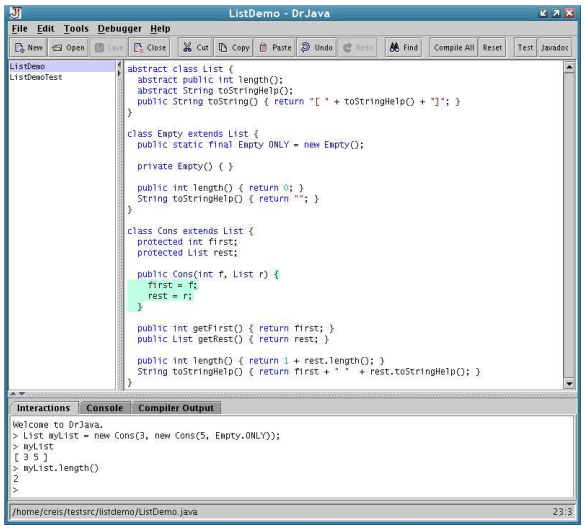

Figure 1: DrJava

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. It consists of three panes: (*i*) a Definitions pane used to enter program text; (*ii*) an OpenFiles panel listing the open files and highlighting the one selected for display in the Definitions pane, and (*iii*) an Interactions pane used to evaluate arbitrary statements and expressions in the context of the files listed in the OpenFiles pane. (See Figure 1 for a screenshot of DrJava.)

3.1 Interactions Pane

In essence, DrJava's Interactions pane transforms Java from a "batch-oriented" (command-line based) language to a reactive one comparable in interactive flexibility to functional languages like Scheme and ML. Almost all of DrJava's other features and integrated tools build upon the 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 [15] 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 for Scheme called DrScheme [6]. The DrJava Interactions pane uses DynamicJava [10] to interpret interactions, leveraging reflection and dynamic class loading for efficiency on par with command line execution.

The Interactions pane provides students with a simple interface for executing Java programs, eliminating the need for `public static void main(...)`. More significantly, this interface enables students to directly observe the behavior of individual methods in the programs that they write—reinforcing the idea that each individual unit of a program should be separately tested. The Interactions pane also provides a simple framework for exploring the behavior of the Java libraries and for conducting computational experiments, even at breakpoints during debugging.

In the classroom, the Interactions pane has proven to

be a very effective teaching tool; instructors can easily explain new concepts, language features, and even common programming pitfalls in lecture by demonstrating them in the Interactions pane. Recent versions of DrJava also support playback of recorded Interactions pane histories so that instructors can compose such classroom demonstrations in advance. At any point during a DrJava session, the user can save the interactions history to a file, which can be edited if desired to eliminate clerical mistakes. When playing back a history, each interaction stored in the file is loaded into the Interactions pane and evaluated on cue from the instructor. The Interactions pane currently plays a significant role in demonstrations in introductory courses at several universities, including Rice University, the University of Pennsylvania, and the University of Washington.

3.2 Additional Pedagogic Features

Like most Java IDEs, DrJava also helps students learn the syntax of the language through various forms of syntax highlighting. To keep the implementation lightweight, DrJava does not perform complete incremental parsing of program text; it uses a very simple incremental parsing scheme sufficient to recognize the major syntactic features of a program: keywords, strings, comments, and various forms of brackets. In contrast to some other lightweight IDEs, the DrJava editor *always* displays accurate syntax highlighting.

DrJava integrates all of the tools that are essential to Java software development: Java compilers (which are plug-ins) including Generic Java [5, 18], the widely used JUnit testing framework [4], a source-level debugger, and the standard Javadoc documentation tool. The DrJava debugger is closely integrated with the Interactions pane [14], so that users can set breakpoints in source code to suspend the evaluation of any method call from the Interactions pane. Hence, individual program units (methods) can be debugged in isolation from the rest of the program. During a breakpoint pause in program execution, users can interact with the state of the program directly in the Interactions pane, calling any methods and accessing or modifying any fields or local variables that are in scope. In short, DrJava supports a very powerful but intuitively familiar interface for understanding and debugging Java code.

Unlike other pedagogic IDEs, which are typically limited to small programs, DrJava scales to developing large production programs, including itself. For the past year, DrJava has been maintained and extended by a team of students using DrJava almost exclusively. On such a code base, consisting of over 40,000 lines of source code (excluding comments) plus the DynamicJava interpreter, DrJava performs very responsively on machines of modest capability, such as a 500 MHz G3 Apple iBook, that are too slow to run professional IDEs like Eclipse. In addition, its code base is sufficiently compact and accessible that junior and senior undergraduates can easily extend and modify it. We use DrJava as the principal source of student projects in the undergraduate course that we teach on production programming [2].

3.3 Potential Drawbacks

In contrast to professional IDEs, 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 comprehensive plug-in architecture, making it more difficult for other educators to add new pedagogic features to the environment, such as an automated project turn-in mechanism or lecture content dissemination. Despite the effectiveness of DrJava on sizable projects, it lacks several important features commonly supported in professional IDEs, including code completion, navigation tools, and program refactoring tools.

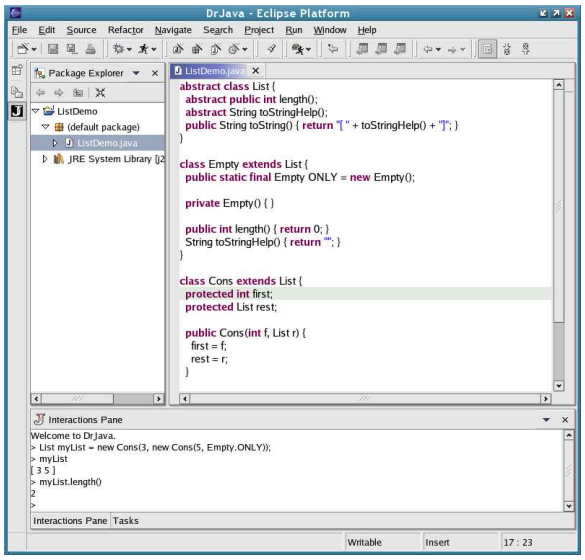

Figure 2: The DrJava Plug-in for Eclipse

4. A DrJava Plug-in For Eclipse

In principle, a professional IDE could be configured to provide a simple and interactive interface like that of DrJava. As such, it would be suitable for use in introductory programming classes.[1] Moreover, as students grew more sophisticated, the course could progressively expose more of the features of the environment. Fortunately, the recent development of the open-source Eclipse environment has placed this scenario within practical reach. Eclipse supports a plug-in and configuration architecture that makes it possible to add new tools like DrJava's Interactions pane and reconfigure the environment to present a simpler interface.

During the past year, we have written a "DrJava plug-in" for Eclipse supporting a simplified user interface that closely resembles the interface provided by DrJava. The DrJava plug-in for Eclipse creates an Interactions pane supporting exactly the same form of incremental program interaction as DrJava. After installing the plug-in, users can configure Eclipse to use the "DrJava Perspective," producing a graphical layout and user interface very similar to DrJava, consisting of a Definitions pane (the Eclipse editor) and an Interactions pane. Figure 2 shows a screenshot of the DrJava perspective that looks quite similar to the corresponding screenshot for DrJava in Figure 1.

Just as DrJava's Interactions pane is integrated with the source files open in DrJava, the plug-in's Interactions pane is closely integrated with the projects that are open in Eclipse. Specifically, the classpath entries for each project are automatically in scope within the Interactions pane.

[1]Assuming the availability of potent personal computers to produce acceptable performance.

However, there is one notable discrepancy between the models of compilation in DrJava and Eclipse which affects the Interactions pane. In DrJava, the Interactions pane is reset after each compilation, in order to load the modified class files to make the Interactions pane's behavior consistent with the current program text. In Eclipse, source files are compiled incrementally on a frequent basis. Resetting the Interactions pane after each incremental compilation would result in too many unexpected interruptions, so instead, these compilations simply produce a single warning message to the user, requesting a manual reset at the user's convenience.

By facilitating a simple and interactive interface for development with most of the pedagogic strengths of DrJava, 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, instructional programs can use the Eclipse environment for writing Java programs at all levels of the curriculum without adversely affecting the atmosphere and content of introductory courses. With the DrJava plug-in, Eclipse is transformed from an intimidating professional environment to a pedagogic IDE with a reasonably simple interface. In addition, the DrJava plug-in can be combined with other pedagogic plug-ins, enabling an instructor to create an environment with the specific capabilities required for a particular course. If the computing resources available to students in introductory courses can run Eclipse with good response times, then Eclipse, equipped with the DrJava plug-in, is well-suited to teaching the rudiments of Java programming.

- Second, the DrJava plug-in supports a smooth, painless transition from native DrJava to Eclipse. 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 provide a suitable interactive interface. With the DrJava plug-in, Eclipse supports precisely the same interactive behavior as DrJava.

- 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 a potent tool for professional software developers. As a result, we anticipate that some professional developers will use the DrJava plug-in for Eclipse.

4.2 Implementation Architecture

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 development time and effort, to minimize the introduction of new errors, and to simplify program maintenance. Indeed, this technique ensures that any new features or bug fixes for the core code of the Interactions pane in DrJava will be immediately available in the Eclipse plug-in as well.

Our program development process emphasizes the use of design patterns [8] and follows the central tenets of Extreme Programming [11] including incremental, test-driven development; comprehensive unit testing; continuous integration; frequent refactoring; and small releases. In this context, we found it easy to re-use the existing DrJava code base to support the plug-in. Specifically, once we performed two primary refactoring transformations to the DrJava code base (to support a stand-alone Interactions pane and the use of Eclipse's SWT graphical windowing toolkit), very little new code was required to implement the Eclipse plug-in. In fact, only eight extra classes comprise the current pre-release of the plug-in, which provides a fully functional Interactions pane and a simplified user interface. Approximately 80% of the code in the Eclipse plug-in is shared with the native DrJava implementation.

The DrJava plug-in for Eclipse is still in active development. The current pre-release (version 0.92) supports a simplified user interface and a fully functional Interactions pane. The full release (version 1.0) of the plug-in will also couple the Eclipse debugger with the Interactions pane, providing Eclipse with the same interactive debugging capabilities as DrJava.

5. Related Work

BlueJ [12] is a pedagogic IDE that also eliminates 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 developing small programs in introductory courses.

Eclipse itself provides a Java Scrapbook feature to promote interactive evaluation of Java code, but its interface is much less intuitive and 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 to evaluate it using a context menu or one of several toolbar buttons. 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 at a later point. This is a weakness shared by conventional "read-eval-print loops" 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 simply by compiling the defined program and executing the sequence of statements and expressions in the interactions history. Eclipse's scrapbook does not provide a convenient means for incrementally building such a history. Meanwhile, DrJava's Interactions pane provides sharper distinctions between interactions, results, and errors than the scrapbook.

6. Directions for Further Work

Our immediate plans for improving the Eclipse plug-in are twofold. First, we want to provide a simpler user-interface for beginners and offer better support for course instruction

by integrating the DrJava plug-in with plug-ins being developed by the Gild[16] and Penumbra[13] projects. These projects are developing plug-ins to make Eclipse more suitable for introductory programming by eliminating unnecessary features from the default Java perspective (reducing the number of panes, menu bars, and commands per menu) and by supporting courseware integration. We also plan to port some additional features of the native DrJava Interactions pane, notably the logging and playback of interactions histories, that are not part of the core shared with the Eclipse plug-in.

Second, we plan to extend the DrJava plug-in to support the language extensions in Java 1.5.[2] We are in the process of adding these extensions to the Interactions pane in native DrJava which already supports the experimental Java 1.5 compiler (versions 2.2 and 2.3). We are working on providing full static type checking—including generic types—in the Interactions pane. The current version of the Interactions pane only detects type errors dynamically during execution. We are also working on extending the interpreter in the Interactions pane to accept the new syntactic forms (such as enumeration types, autoboxing and unboxing, and foreach statement) that will be introduced in Java 1.5 [18]. We are writing a preprocessor that expands the new Java 1.5 constructs into existing constructs, eliminating the need to revise the DynamicJava interpreter. The new features in Java 1.5 are already accepted by the DrJava editor.

As soon as Eclipse's editor and compiler are updated to support Java 1.5, we will enable support for Java 1.5 in the DrJava plug-in. The requisite code will already be in the DrJava code base.

7. 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 variety of sophisticated features and tools suitable for production programming and advanced programming courses. We have developed an Eclipse plug-in that supports the same interactive user interface as DrJava—making Eclipse suitable for use by beginning students with sufficiently powerful machines, and providing advanced users with a better interface for performing program interactions. In the process, we have demonstrated how design patterns and Extreme Programming practices can facilitate the re-use of a large code base like DrJava.

8. References

[1] E. Allen, R. Cartwright, B. Stoler. *DrJava: A Lightweight Pedagogic Environment for Java. SIGCSE 2002*, March 2002. (URL: `drjava.org`)

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

[3] O. Astrachan *et al. Recommendations of the AP Computer Science Ad Hoc Committee*, October 2000. (URL: `apcentral.collegeboard.com/repository/ap01.pdf.ad_7908.pdf`)

[4] K. Beck, E. Gamma. "JUnit, Testing Resources for Extreme Programming." (URL: `www.junit.org`)

[5] 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.

[6] R. Findler *et al. DrScheme: A pedagogic programming environment for Scheme*. In *International Symposium on Programming Languages: Implementations, Logics, and Programs*, 1997.

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

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

[9] J. Heiss. "New Language Features for Ease of Development in the Java 2 Platform, Standard Edition 1.5: A Conversation with Joshua Bloch." (URL: `java.sun.com/features/2003/05/bloch_qa.html`).

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

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

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

[13] F. Mueller, A. Hosking. "Penumbra: An Eclipse plugin for introductory programming." In Eclipse Technology Exchange Workshop, OOPSLA 2003, October 2003.

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

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

[16] M. Storey, et al. "Improving the Usability of Eclipse for Novice Programmers." In Eclipse Technology Exchange Workshop, OOPSLA 2003.

[17] The College Board. "The AP Computer Science Course Description." (URL: `www.collegeboard.com/prod_downloads/ap/students/compsci/ap03_compsci.pdf`)

[18] Sun Microsystems, Inc. Early Access Downloads page. (URL: `developer.java.sun.com/developer/earlyAccess`)

[2] Planned for release in 2004.