DR. J VS. THE BIRD: JAVA IDE'S ONE-ON-ONE*

Michael Olan
Computer Science and Information Systems
Richard Stockton College
Pomona, NJ 08240
609-652-4587
olanm@stockton.edu

ABSTRACT

An important decision facing instructors of introductory programming courses is the choice of supporting software development tools. Usually this involves selecting an integrated development environment (IDE). BlueJ has received widespread adoption for first year courses that use the Java programming language; however, DrJava is emerging as an alternative. This paper features a comparison of the pedagogical approaches used by BlueJ and DrJava as a guideline for selecting the tool best suited to the teaching style used in the introductory course.

1. INTRODUCTION

The choice was simple when text editors and the command line were the only tools for developing programs. That changed with the introduction of integrated development environments (IDE's), and campuses nationwide adopted Borland's Turbo Pascal. Languages also have changed. Pascal was designed as a teaching language, but now academic programs use advanced languages designed for professional software engineers. Making these complex languages accessible to beginning students requires a careful selection of the development environment. This discussion will only include Java, currently the dominant language choice for introductory courses.

Let us first consider several levels of tool support in the introductory course sequence:

1) Minimal: A text editor and the command line.
2) Simple IDE: Just the basics, no bells and whistles.
3) Professional IDE: Powerful and feature filled.

---

* Copyright © 2004 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

A difficulty associated with the minimalist approach of using only a simple text editor and the command line is that the student must work directly with the underlying operating system and file system, adding an extra dimension to the learning process. They must also deal with the complexities of the language system, such as compilation dependencies, environment variables, debugging, and execution.

Professional IDE's effectively isolate the developer from much of this complexity; however, their increased sophistication adds a complexity of its own. Beginners faced with the steep learning curve and large feature sets of such IDE's tend to be distracted from the already difficult task of mastering object-oriented programming concepts and Java.

The developers of the BlueJ IDE recognized the gap between these extremes, and designed a tool specifically for introductory level software courses using Java [7, 8]. Their small, simple, and free IDE has become immensely popular, but is not the only IDE developed with the beginning student in mind. DrJava offers another simple pedagogical tool with an alternative approach. A number of other tools, while not specifically targeted for the classroom, are nonetheless viable candidates for instructional use. The focus of this paper is a comparison of the pedagogical approaches taken by BlueJ and DrJava, and their appropriateness for different styles of teaching introductory software design.

2. KEEPING IT SIMPLE

Instructors in beginning software design courses are faced with a number of pedagogical choices when selecting topics to cover and tools to support instruction. Should knowledge of the underlying operating system and file system be required? How much knowledge of the language system will be required? Should the course use an "objects first", "objects early", or "objects last" (traditional structured) approach? The pressure to cram more and more material into introductory courses is strong.

If maintaining a focus on object-oriented design concepts is the goal, then an IDE that isolates the student from underlying complexities will be a benefit. Ease of use is critically important to allow students to concentrate first and foremost on problem solving and programming tasks. Both BlueJ and DrJava were designed as tools that would require a minimal effort to learn.

The greatest strength of both tools is an emphasis on encouraging interactive experimentation with Java constructs. Beginners do not have to write complete programs to experiment with simple expressions and statements, and get immediate feedback. Once they do start writing their own classes and methods, they can continue to use the interactive features of the IDE to test their results. It is here that the two systems take significantly different approaches. BlueJ uses a primarily visual interface, while DrJava's interactions are based on a read-eval-print interpreter similar to Scheme. The choice of IDE should be determined by the teaching style being used in the course. The remainder of this paper outlines the capabilities of BlueJ and DrJava to assist in making such a choice.

3. BLUEJ

BlueJ was developed at Monash University, Australia specifically as an environment for teaching introductory object-oriented programming [3].

3.1 Objects First

A hallmark of BlueJ is the ability to interactively create and manipulate objects. The rationale behind this is to introduce objects by allowing students to experiment with them without having to write any code [9]. An instructor can provide prewritten classes, or have students create objects from classes in the Java standard library. The user creates an object by selecting a constructor for a class. Then a dialog box shows the signature and documentation for the selected constructor, and presents a template for entering the values of arguments (see Figure 1). The layout of the template gives visual suggestions to the actual Java syntax of the method call, and is annotated to emphasize the required type for each argument.

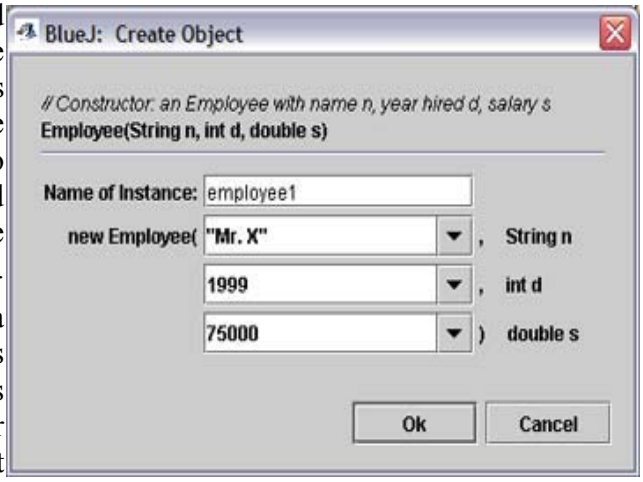

Figure 1. BlueJ Object Creation Dialog.

Creating an object displays a UML object diagram in BlueJ's "object bench". Selecting this diagram gives access to an inspector for viewing the state of the object (see Figure 2).

The object's methods are accessible from a menu (see Figure 3) and can be executed by supplying arguments in a dialog box similar to the one shown for the constructor in Figure 1. However, objects with a large number of inherited methods can present an overwhelming list that is difficult to navigate.

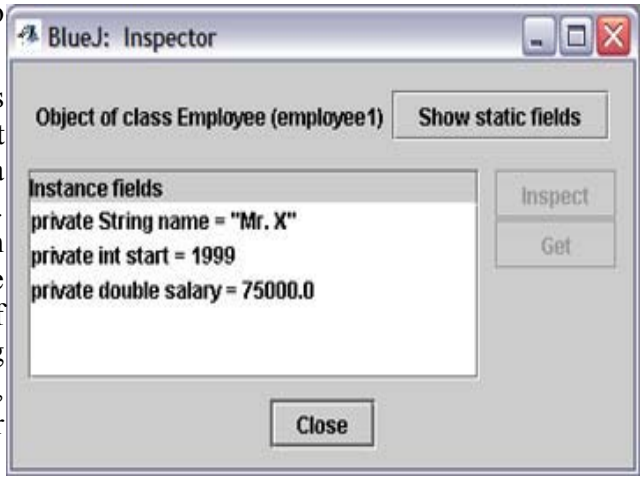

Figure 2. Object Inspector.

These interactive features provide a simple way to experiment with and test the functionality of a class without requiring a test driver. By creating and experimenting with objects, students experience the fundamental concepts of object-oriented programming (classes, objects, methods, parameters) without seeing or writing a single line of code [8].

3.2 BlueJ's User Interface

The organizational unit in BlueJ is the project, which is implemented as a directory in the file system. All Java source code and class files for the project reside in this directory. A BlueJ project is similar to a package, and although supported, BlueJ does not require beginning programmers to deal with this advanced concept.

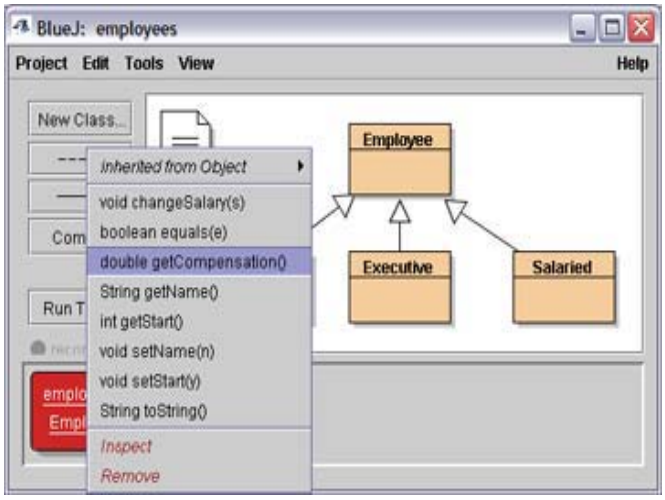

Figure 3. Method Access.

Once a project is defined, classes are easily added by clicking a **New Class** button. BlueJ's main window displays classes using simplified UML class diagrams (see Figure 3). Arrows connect class icons to show dependency and inheritance relationships. Adding an "inheritance" arrow automatically inserts the appropriate extends clause to the source code of the subclass. Adding a "uses" arrow does not affect the source code. Likewise, writing an extends clause or declaring an instance variable of a class type in the source code causes the appropriate arrow to connect the class icons.

BlueJ decorates a class icon with diagonal stripes, as a visual cue indicating that (re)compilation is required. Clicking a **Compile** button in the main window will compile all modified files, and classes that depend on them. BlueJ automatically saves modified files before compiling. The BlueJ compiler stops when the first syntax error is found and opens an editor window on the line where the error was detected.

Classes with a main method can be executed by right-clicking the class icon and selecting `void main(a)` from a popup menu. Standard output is displayed in a terminal window.

Double-clicking a class icon in the main window opens the source file in an editor window. A template provides a sample skeleton with the class header, a constructor, a sample instance variable, and a sample method stub. These templates can be edited so that programmers can use their own preferred form of template. Editor features include auto-indentation, keyword highlighting, and parenthesis/braces matching. It does not have any code completion features. Individual classes can be compiled from their editor window.

BlueJ has a simple to use debugger with a graphical interface. Breakpoints are toggled in an editor window with a mouse click. The debugger window shows the call stack, instance variables, local variables and static variables. The usual step commands are available by clicking buttons.

JUnit [7] test cases can be added to a project, and a skeleton template is provided. BlueJ also supports building test cases by recording actions from the object bench. This encourages interactive experimentation with objects by executing their methods and

recording the experiment as a test method, and minimizes required knowledge of the JUnit API.

Additional features include exporting to executable jar files; complete javadoc generation for a project; browsing javadocs of Java API classes and project classes; and a limited form of expression evaluation.

Since version 1.3.0, BlueJ supports extensions, the first being a mechanism for online project submissions.

4. DRJAVA

DrJava is an ongoing project at Rice University, Texas, developed and maintain by students [4]. DrJava shares the goal of providing a pedagogic environment that minimizes the intimidation factor experienced by beginning students [1]. A priority of this project is providing an interface that is simple, interactive, and with a focus on the language [10].

4.1 Read-Eval-Print

The distinguishing feature of DrJava is the "interactions pane," a "read-eval-print loop" (REPL) for evaluating Java expressions and statements interactively. Functional languages like Scheme have long used REPL to facilitate incremental program development. Users experiment with Java constructs by typing an expression or statement and having it evaluated immediately, without having to write a full Java program. REPL makes it possible to completely avoid the difficulties of Java I/O at the introductory level by using parameters and function return values. The interface is text based, and does require using Java syntax (see Figure 4). This design was intentionally chosen so that only a single medium is required for dealing with program development, in contrast to the combination of UML and Java source code required by BlueJ [1].

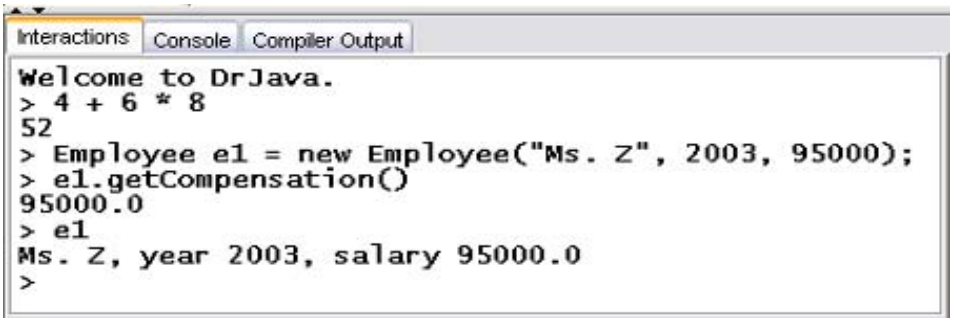

Figure 4. DrJava Interactions Pane

Note that the last expression (`e1`) in Figure 4 implicitly invokes the `toString` method for class `Employee`. Currently, this is the only way to inspect the instance variables of an object outside of the debugger.

Interactions can easily process graphic objects as shown in Figure 5, where a JFrame is created, and then manipulated by resizing, changing color, adding a title, and moving

it to a new location on the screen. This example shows that a text based interaction environment is not a limiting factor in providing visual feedback with graphical objects.

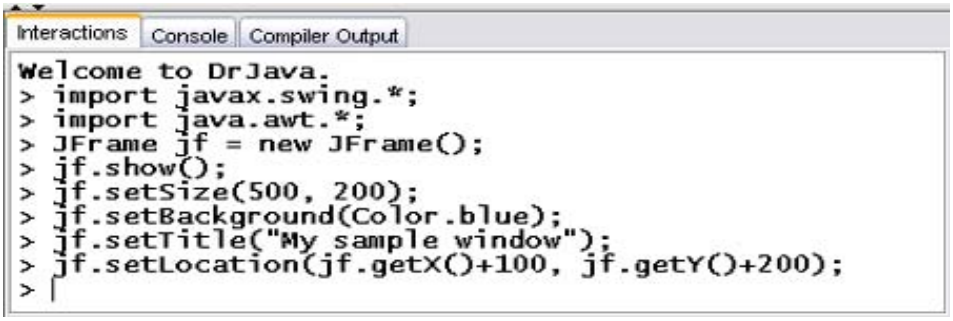

Figure 5. Interaction with a GUI object.

4.2 DrJava's User Interface

DrJava maintains a focus on the Java language, and provides a single window environment (see Figure 6) with two components: a "definitions pane" (editor) and an "interactions pane" (described above). When a class is compiled in the definitions pane, it is immediately available for use in the interactions pane. The REPL features of the interactions pane encourage incorporating functional programming concepts in conjunction with the object-oriented model.

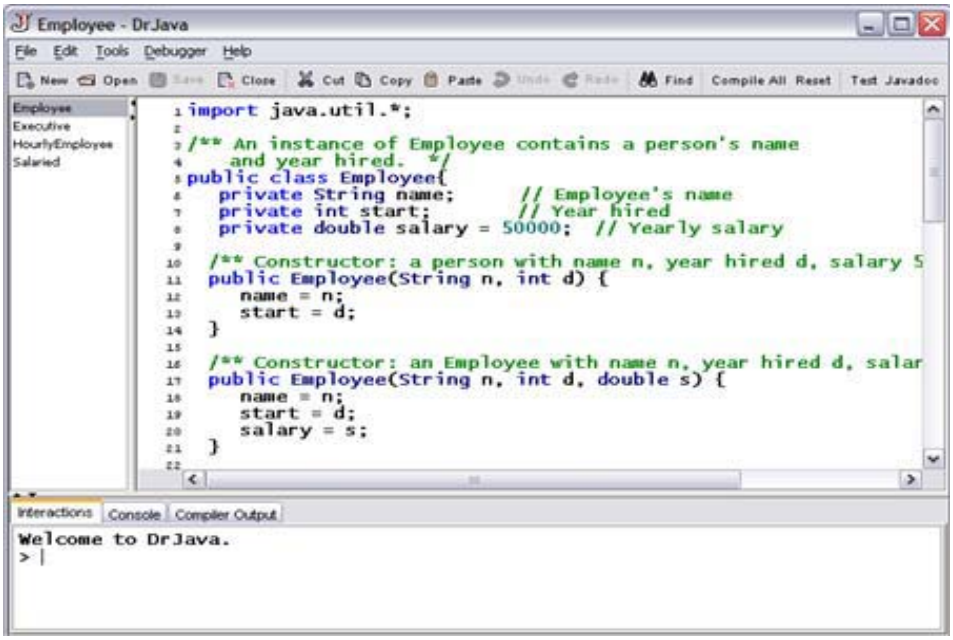

Figure 6. DrJava Definitions Pane and Interactions Pane

The interactions pane maintains a history list for easily recalling previously entered commands. This significantly reduces the typing required when performing experimental evaluations.Interactions can be saved as a script and reloaded in later sessions. A **Lift Current Interaction to Definitions** command copies an interaction into the editor. This

provides a convenient way to move experimental tests into a JUnit test to make them repeatable.

Since each class method can be executed independently, the interactions pane is an effective tool for simple testing and debugging. DrJava also includes a debugger that supports setting breakpoints and defining watches. When execution is suspended at a breakpoint in debug mode, the interactions pane can be used to inspect or modify state variables using Java expressions and statements. Again, the design approach was to maintain a simple and consistent interface throughout.

The DrJava compiler parses the entire file and reports all syntax errors. Clicking an error message highlights the error in the source code.

The editor provides automatic indentation, parenthesis/braces matching and quotation/comment highlighting that is updated with every keystroke. An **Indent lines** command will properly format a complete Java class. The editor supports multiple documents but does not organize files into projects. It is necessary to use the operating system's file management facilities to organize files.

Additional features of DrJava include built-in support for JUnit test cases; generation of javadoc documentation; and javadoc preview for the current document.

5. COMPARISONS

BlueJ is the more feature rich of the introductory IDE's, but it could be argued that the leaner interface of DrJava is an advantage. The more significant distinguishing features of BlueJ include:

- Files organized into projects
- Object inspector for observing the state of an object
- Access to Java API documentation
- Jar file export

Significant features integrated into DrJava include:

- History list of statements/expressions in the interactions pane
- Single medium for program development - text
- Simpler interface - a single window divided into two panels

Elaborating on these fundamental differences, BlueJ provides a more visual and graphical interface, but can result in a number of open windows (main, editors, object inspectors, debugger, terminal) that must be navigated. DrJava has a single window with two panels. BlueJ's interactions with objects does not require writing any Java code, but the user does need to select menu items repeatedly, enter arguments, and duplicate this process for each new object. DrJava's command history makes it simple to recall and edit commands so that repeating an experiment can be done with minimal duplication of effort.

Both BlueJ and DrJava are free and open source. Both have integrated debuggers and support unit testing with JUnit. Both have tools for generating javadoc documentation. Both are available for multiple platforms, including Windows, Linux, and Macintosh.

The BlueJ developers say that students may have a difficult time making the transition to a more sophisticated IDE, but recommend such a switch after completing introductory programming courses [9]. DrJava's developers suggest that the IDE is useful beyond the beginner level, and cite advanced courses at Rice University that feature projects for extending the tool. One such project produced a DrJava plugin available for the Eclipse IDE that should ease the transition to the more powerful features of a professional IDE. In fact, this plugin is a significant improvement over Eclipse's built in support for interactive expression evaluation.

6. OTHER TOOLS AND IDE'S

The following are several tools and IDE's that the author considers worth mentioning for their potential usefulness for instructional purposes. The IDE's in particular are good options when students outgrow BlueJ or DrJava. This sample does not include any commercial products.

6.1 BeanShell

BeanShell is an interpreter with read-eval-print functionality similar to the interactions pane of Dr.Java. BeanShell is a standalone tool with no additional IDE features [2].

6.2 GEL

GEL is a lightweight but full featured native Windows freeware IDE that is relatively simple to use [6]. GEL is highly customizable and includes many of the features found in professional IDE's, such as code completion, parameter hints, syntax highlighting for a number of languages, Ant support, and CVS support.

6.3 Eclipse

Eclipse, a sophisticated professional IDE, is an IBM donation to the open source community [5]. Eclipse has a large and active developer community contributing to the evolution of the project both directly and through plugin development. Eclipse provides a wealth of advanced features and potential for student oriented projects. Ironically, this powerful professional tool provides significantly better compiler error messages, complete with suggested corrections, than either BlueJ or DrJava. In fact, the incremental compilation feature of Eclipse detects most syntax errors as the user types.

7. CONCLUSIONS

IDE's are valuable tools for easing the entry of beginners into software development courses. Both BlueJ and DrJava provide simple interfaces specifically designed for teaching. Both encourage interactive experimentation and incremental development, which is perhaps the single most important pedagogical feature of these IDE's. DrJava has a cleaner and simpler interface that maintains a focus on the Java language. BlueJ

provides a more elaborate and graphical interface and emphasizes objects first. Both provide excellent options for simple entry level software development in Java.

8. REFERENCES

[1] Allen, Eric and Robert Cartwright, Brian Stoler "DrJava: A Lightweight Pedagogic Environment for Java," *SIGCSE 2002*, March 2002.

[2] BeanShell. http://www.beanshell.org/ (Web site)

[3] BlueJ. http://www.bluej.org/ (Web site)

[4] DrJava. http://drjava.sourceforge.net/ (Web site)

[5] Eclipse. http://www.eclipse.org (Web site)

[6] GEL. http://www.gexperts.com/ (Web site)

[7] JUnit. http://www.junit.org (Web site)

[8] Kölling, Michael. "The Problem of Teaching Object-Oriented Programming, Part 2: Environments," *Journal of Object-Oriented Programming*, 11(9): 6-12, 1999.

[9] Kölling, Michael and Bruce Quig, Andrew Patterson, John Rosenberg. "The BlueJ System and Its Pedagogy," *Journal of Computer Science Education*, Vol 13, No 4, December 2003.

[10] Stoler, Brian. "A Framework for Building Pedagogic Java Programming Environments," Master's Thesis, Rice University, April 2002.