Experiences with IDEs and Java Teaching: What Works and What Doesn't

Keitha Murray, Organizer
Department of Computer Science
Iona College
New Rochelle, NY 10801
(914)633-2536
kmurray@iona.edu

Jesse M. Heines
Department of Computer Science
University of Massachusetts Lowell
Lowell, Massachusetts
heines@cs.uml.edu

Tom Moore
Department of Computer Science
University of Wisconsin – Eau Claire
Eau Claire, Wisconsin
tommoore@uwec.edu

John Trono
Department of Computer Science
Saint Michael's College
Colchester, Vermont
jtrono@smcvt.edu

Michael Kölling
Mærsk Institute
University of Southern Denmark
mik@mip.sdu.dk

Nan C. Schaller
Department of Computer Science
Rochester Institute of Technology
Rochester, New York
ncs@cs.rit.edu

Paul J. Wagner
Department of Computer Science
University of Wisconsin – Eau Claire
Eau Claire, Wisconsin
wagnerpj@uwec.edu

Panel Description

The environment chosen to teach Java can have a profound effect on students' abilities to learn the language. Panelists will report on their experiences using different Java Interactive Development Environments (IDEs) to teach Java and what they identify as the strengthens and weaknesses of each IDE. Each panelist will discuss the most important features of the IDEs and related teaching pedagogies to address "what works and what doesn't" when teaching Java.

Panelists Statements

Jesse M. Heines

Forte (Sun ONE Studio 4, www.sun.com/software/sundev) has been the least satisfactory of the IDEs I have used. On some systems it simply wouldn't install. On others it would work for a while, then inexplicably lose its ability to "see" files in project directories. The user interface is overly complex, requiring one to change literally dozens of settings just to configure fonts and get the IDE to insert spaces instead of tabs, a critical feature where professors may view students' files with a different editor than the one students used to create them. We quickly abandoned this IDE.

JPad (ModelWorks Software, www.modelworks.com) is much more stable, but while inexpensive (US$29 for the Basic version, US$59 for the Pro version), it is not free. I currently use the Pro version myself, but it has two major drawbacks: the project implementation is particularly awkward and the help system is not industry standard. Running programs annoyingly involves two steps rather than one, as the IDE always brings up a dialog box asking you to confirm which file to run and its command line parameters. Thus while this IDE at least works, it is not optimum for student use.

Michael Kölling

Object orientation has increased the overhead of accidental complexities in first year courses. A good environment can do a lot to avoid resulting problems. The benefit of BlueJ, however, is not mainly in making the tasks smoother that students otherwise do from the command line, but much more in the provision of tools to enable activities that would not otherwise be possible. Much of the benefit comes from the pure object-orientation of the environment itself, which goes beyond what most other environments provide: IDE/OO ≠ OO/IDE ("an IDE for object orientation is not the same as an object-oriented IDE"). Using custom-designed tools for object-oriented teaching, we can employ a different pedagogy, teach in a different order, and thus shift the emphasis of topics in the curriculum.

We have used BlueJ for several years with good success. In this panel session, we will report on our approach to using the BlueJ tools as well as benefits and problems with using this approach.

Tom Moore

Eclipse is an open source, extensible IDE, as well as an open source software development project, that has the support of major software vendors such as IBM, Oracle, Rational, and Borland.

Eclipse is interesting as a pedagogical tool in at least three dimensions:

1. for introductory Java programming,
2. for upper-division courses in which Java extensions are taught, and
3. as an environment for developing special purpose tools.

Specifically, several incarnations of Eclipse are particularly valuable. Rational XDE allows young developers to see and

learn the correspondence between visual modeling in UML and Java coding. WebSphere Application Developer can be used in the development of Web-based and Enterprise Java applications, and also provides support for a number of design patterns. Finally, since Eclipse is open source, it can be used to build plug-ins that extend the basic interface and provide opportunities for learning new Java technologies such as XML, Struts, or even Google searches.

Tom Moore has been programming in Java since 1998 and working with Eclipse since Spring 2002. He has used Eclipse extensively in both lower- and upper-division classes at the University of Wisconsin – Eau Claire, where he reports that students are very excited and productive in using Eclipse in its various manifestations.

Nan Schaller

The Computer Science Department at Rochester Institute of Technology has several laboratories that are equipped with Sun Microsystem workstations running Unix. Our introductory courses require a formal laboratory component. Some students are unable to complete laboratory exercises during the formal lab period. Many students have PCs in their rooms, although some have Macs. Therefore, as it is free and runs on all three platforms, GNU Emacs is used with its JDE (Java Development Environment) to aid in teaching Java to introductory students. The JDE provides a color-coded automatic formatter and does allow for code compilation and code execution from within Emacs as long as only a single Java class is involved. It also facilitates locating compilation errors. The Emacs editor, while not the easiest to use, does provide automatic formatting that is compliant with the courses' style standard, such as indenting and lining up code properly. With larger, multiple component programs, students must invoke the Java compiler via the command line to make sure that all components are compiled. Command line invocation is also necessary for execution, if the class being edited does not contain the *main* method.

What works? Students can work on their assignments on their own computer without additional cost, regardless of the type of the computer. Using Emacs in this way encourages students to think about style standards. Students learn how to invoke the java compiler and to execute java code both through the IDE using the command line.

What doesn't work? Emacs takes a while to learn as its user interface has some non-intuitive ways of handling tasks. Emacs provides no debugging capability. Getting Emacs set up with JDE can be tricky, but the department does provide a local download site along with step-by-step instructions.

JohnTrono

Our CS1 course has a required weekly lab session that meets for 100 minutes. After two guided labs that familiarize our students with the necessary desktop software, i.e. how to access shared departmental datafiles, use network printers, etc., and the basic Integrated Development Environment (our IDE is JBuilder 6.0, running on a PC with Windows NT), they begin to develop their own Java classes during lab with possible guidance/assistance from the instructor or lab assistant.

JBuilder is a Borland product that: displays keywords in bold; facilitates easy transitions to each specific line where a syntax error is, or to where a run-time error has occurred; and makes it relatively straightforward to create a Java application or an applet. In our second course, we show students how to set (and advance to) breakpoints, how to step through the program after reaching the breakpoint, and how to examine the program's state via the current values in the variables in an effort to help them develop a reasonable set of strategies for quickly removing their software defects. We have found that the inclusion of these latter tools in the JBuilder IDE can reduce the students' frustration, and dramatically lower the time required during the testing and debugging phase of the software development cycle, both of which can increase the chances that each student learns the necessary concepts. The JBuilder IDE also makes it very easy to include packages, and other Java predefined classes, during the compilation stage.

Paul Wagner

I and several others in our department have been using a combination of IBM's Visual Age (currently the Entry Professional Edition 4.0, freely available at www7b.software.ibm.com/wsdd/zones/vajava) and TogetherSoft's Together Control Center (currently version 6.0, commercially available, see www.togethersoft.com) for the past several years in our Introduction to Object-Oriented Programming course. Students start by learning Visual Age as an environment for developing basic object-oriented Java programs. They are also introduced to Together (as well as UML in general) about halfway through the semester. From that point on the students have their choice of environments for laboratory assignments and programming exercises.

We see Visual Age as having its strengths in several areas. First, it has a fairly easy interface and is thus easy to learn, even for those with little programming experience. Second, it has a strong collection of available built-in projects/packages in areas from XML to database access using JSPs and JavaBeans. Third, it has a good set of tools for debugging and program development. Its major weakness is not having an associated class model associated with the code as the BlueJ and Together environments do.

Together's strength is in its mapping between UML class diagrams and Java code. We are able to use it first for giving the students small software system outlines and having them implement the methods, then later having them use it to implement their own designs. As we stress object-oriented design early, and begin talking about design patterns in the second programming course, Together becomes a useful tool for the students. However, Together's environment seems somewhat less intuitive than Visual Age, and there is some reluctance on the students' part to switch from Visual Age to Together. We have also had some problems with stability of the system in the laboratory environment.

Overall, students in our program adapt fairly well to the use of both products, and gain flexibility through their experience with the two environments.