Penumbra: An Eclipse plugin for introductory programming

Frank Mueller* Antony L. Hosking

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

{mueller,hosking}@cs.purdue.edu
http://www.cs.purdue.edu/s3/projects/eclipse

Abstract

Eclipse is a full-featured and easily extensible integrated development environment. As such, it has grown to include a large degree of functionality that may be overwhelming to the novice programmer. Nevertheless, we believe Eclipse is an environment that students of programming will come to find useful and empowering once they become familiar with it. The trick is easing them into using Eclipse without them feeling overwhelmed at the outset. Penumbra is an Eclipse plug-in developed at Purdue University for use in our introductory programming classes. It is intended to ease the transition to use of the full-featured functionality of Eclipse. Penumbra presents an Eclipse perspective that hides all but the basic actions of Eclipse's existing Java perspective, while packaging elements of other perspectives (e.g., the CVS perspective) into simpler actions that ease the downloading and turn-in of programming assignments, and adding new code views inspired by other environments for introductory programmers. Our experiences using Eclipse with a small group of introductory programming students in the Spring of 2003 have guided the development of Penumbra, which is now being rolled out for general use by the full class of 230 students in the Fall of 2003.

1 Introduction

In the Fall of 2002, we became interested in Eclipse as a possible development environment for use by students of our introductory Java programming course for computer science majors. Eclipse, with its open-source development model, seemed to provide many of the benefits of proprietary development environments, without the cost of dependence on what has seemed to be fickle support for Java in proprietary environments (an earlier incarnation of our introductory programming course had come to rely on Microsoft's Visual/J++ environment which is no longer supported). Moreover, we saw that the extensibility of Eclipse would make it feasible to mold the environment more easily to fit the needs of our teaching and grading processes.

Thus, in the Spring of 2003, and with funding from IBM, we were able to devote time to trials of the Eclipse IDE in the classroom. Our experiences, detailed below, permitted us to spend the Summer of 2003 extending Eclipse with a plug-in tailored for use in our introductory courses. Our goals included:

1. Simplifying use of Eclipse functionality for novices to enable them to quickly start using the environment
2. Focusing their attention on Java programming rather than the IDE
3. Integrating our teaching and grading processes with Eclipse
4. Not dumbing down the interface: once the fundamentals of the development process are understood students are able to transition to the full-blown Eclipse interface

2 Experiences with Eclipse

In the Spring 2003 semester we created a special section of Purdue's introductory Java programming course for computer science majors, in which students were explicitly assisted in using the Eclipse IDE for their programming assignments. The rest of the class used a combination of Emacs and the Unix command-line driven Java SDK, which had been the default environment for several years. Our experiences working with the Eclipse-enabled students, along with the results of an exit survey of these students conducted at the end of the semester, convinced us that the benefits of Eclipse justified the investment of time and effort to train students in its use. Student responses were overwhelmingly positive, with their comments indicating that, once they were familiar with the IDE it enabled them more quickly to identify and solve their programming problems. Some students even saw Eclipse as relieving them of boring trivialities of programming by resolving questions of simple Java syntax and program structure.

The problems that we did see arose more from lack of experience working with such a complex IDE. Simply starting a project, set-

*Supported by an Eclipse Innovation Grant from IBM

ting up class-paths, and running an application appeared more complicated in Eclipse than with Emacs and the Java SDK. However, once students became familiar with these basics of the development process, they quickly appreciated the benefits of the Eclipse IDE. Consequently, we believe that the chief difficulty faced by novice programmers in using Eclipse is the learning curve necessary to just get started. Our subsequent efforts have thus focused on developing an Eclipse plug-in that eases the transition from novice user to basic development, by capturing simple development processes in a small Eclipse perspective. From that point, we expect that users can begin to acquire experience with the more complex features of the IDE as they need them. The perspective we have developed is called Purdue Penumbra ("the essence of Eclipse").

We also were interested in integrating Eclipse with our teaching and grading processes, which rely on electronic access to skeleton project source code, and electronic submission of student solutions for grading. We wanted to take advantage of existing Eclipse functionality where it was useful (e.g., the CVS plug-in permits centralized management of student project files), but to roll that into the Penumbra perspective and simplify the interactions of students for electronic check-out and turn-in of project source code without leaving the perspective.

In addition, we are interested in incorporating tools that can assist novice programmers in understanding and exploring the programming concepts that are being taught to them in class. With Java, there is the difficulty of understanding and using fundamental object-oriented concepts such as inheritance and sub-typing. It has been widely recognized that students can benefit from extra support in visualizing the structure of programs that incorporate such concepts [2, 1]. Indeed, BlueJ is an example of an IDE developed explicitly for novice programmers. However, our experiences with Eclipse lead us to believe that students can grow with the environment they first start out with, so rather than forcing abrupt transition from a teaching IDE to a production IDE (say for later classes) we would prefer simply to ease their initial experience with the production IDE. Nevertheless, we plan to integrate useful functionality, such as that developed by other projects focused on pedagogical issues of programming environments (e.g., BlueJ and GILD [3]) as it becomes available.

3 Perspective simplification

Simplifying the way in which students see Eclipse was the most important task. The default Java perspective has some features that are extremely useful to new students, but it also has features (e.g., support for refactoring, build management, and testing) that are more relevant to experienced programmers working on larger projects than the simple assignments of an introductory programming course. For this reason, we chose to repackage relevant functionality of the Java perspective in our new Penumbra perspective.

There are essentially two elements that constitute an Eclipse perspective: *action sets* and *views*. An *action set* adds menu or tool bars to the perspective while a *view* presents the user an interface (e.g., the package explorer, file editor) for working directly on resources (e.g., source code). The Penumbra perspective is intended to be as simple as necessary for students to work on the basics of their assignments. Note that nothing prevents students from switching to an alternate perspective (e.g., the full-function Java perspective) where the same resources can be worked on using that perspective's functionality. This means students can begin to transition from the simplified view of the Penumbra perspective to the full view of the Java perspective.

The Penumbra perspective (see Figure 1) contains only four views: a simplified package explorer, an object-oriented hierarchy view, the original Java editor and a console to provide I/O. Furthermore we omitted three menus (Source, Refactor, Run) and and three tool bars (run, create new elements, search). The resulting perspective is much cleaner, but still contains the elements needed by the students. The removal of some menu functions (e.g., Run, Debug, Format Code) has been compensated by providing a Penumbra menu and tool bar with shortcuts to these important functions.

Another simplification is in the steps required to run an application inside Eclipse. Usually doing so requires setting up a run configuration for each project. We added functionality to set up the configuration automatically and execute the program immediately, once a the student selects the Run shortcut. Hence, running the program requires only the click on one mouse button.

4 Object-oriented hierarchy view

We agree with Kölling [1] that development environments usually do not integrate the fact that Java is an object-oriented programming language. For example, programming in text-based editors such as Emacs does not highlight this characteristic. Experienced programmers know enough to picture the object hierarchies they define and program with, but for most new students it is a new concept that they must absorb and understand. The Eclipse Java perspective already provides a hierarchical view of classes, but this view is complex and modifications to the hierarchy can be made only in the Java editor. Inspired by BlueJ, we extended the implementation of the default hierarchy view from the Java perspective to show the full hierarchy structure of the selected project, and permit drag-and-drop modification of the hierarchy (with corresponding changes to source code). Programmers use drag-and-drop gestures within the Penumbra hierarchy view to reorganize the hierarchy. We believe this will enhance students' understanding of object-oriented concepts, by seeing the effect of hierarchy changes on the corresponding source code. The hierarchy view can be seen in the bottom left corner of Figure 1.

5 Process integration

Programming classes often require students to download skeleton source code files, to augment those files to achieve a specified solution, and to turn those files in for grading. Automation of this process often requires students to use tools other than those they use for coding. Rather than downloading files separately, importing and configuring them as a project in Eclipse, working on the assignment, and then extracting the resulting solution for turning in, we wanted to provide a seamless Eclipse-based mechanism by which students could work on a project, independently of their lo-

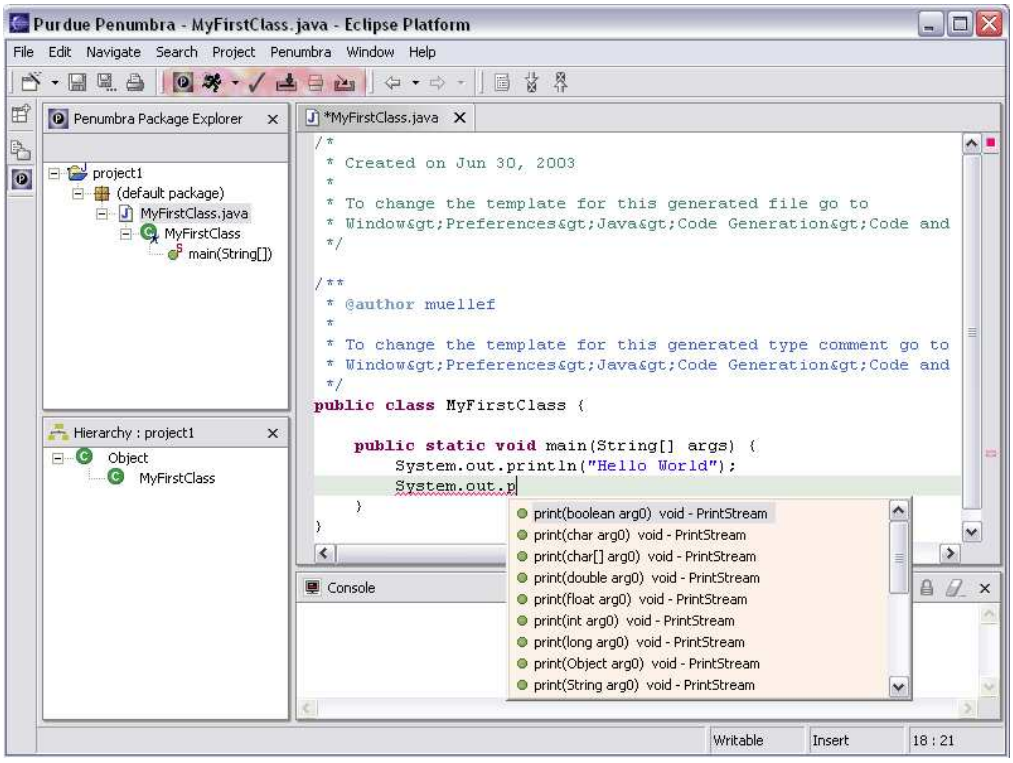

Figure 1: The Penumbra perspective

cation (i.e, home, lab, or friend's house!), and easily make their solution available for grading.

We decided that the existing CVS plug-in supported by Eclipse provides all that is needed, but that its interface needed simplifying for use by novices. Configuring a central CVS server accessible by students (while ensuring the necessary security of their files) also yields a central record of their progress which can be accessed by course instructors and graders. The server is used both to distribute projects to the students, by reading from a public repository, and to store their project files as they work on them in a private repository owned by the student but readable by instructors and graders. Moreover, students have the luxury of tracking the different versions of their projects as they work on them.

Penumbra bundles all of this functionality, so that the complexities of CVS access are hidden. Students need only enter their user name and password in the Penumbra preference settings (see Figure 2 and the CVS settings are automatically initialized for them. A mouse click on the checkout button will show them all projects available and another click will download the selected project to their computer. The first check-out will populate their project with files from the public skeleton repository, while subsequent check-outs update files from their private repository.

A few mouse clicks also allow students to commit their project changes back to the CVS repository. Students experience this via a grading submission dialog (see Figure 3), though the effect is to commit their changes to the repository. Thus, multiple submissions are permitted. Transitioning students to the full functionality of CVS (i.e., seeing the version history of their project files) can

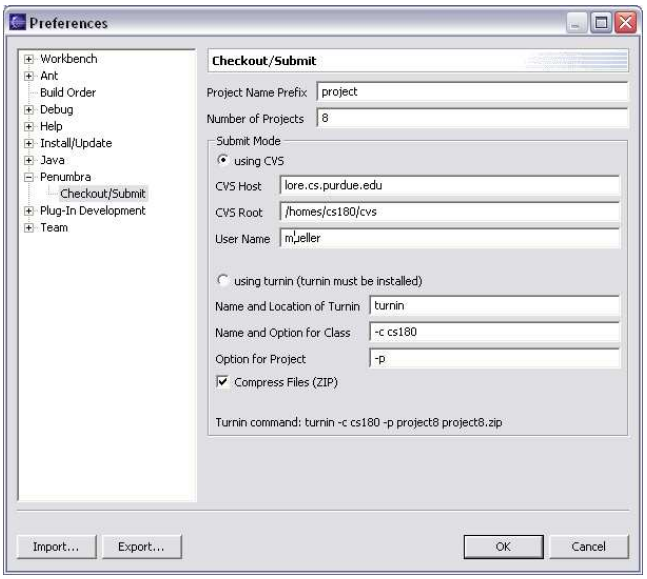

Figure 2: Penumbra preferences page

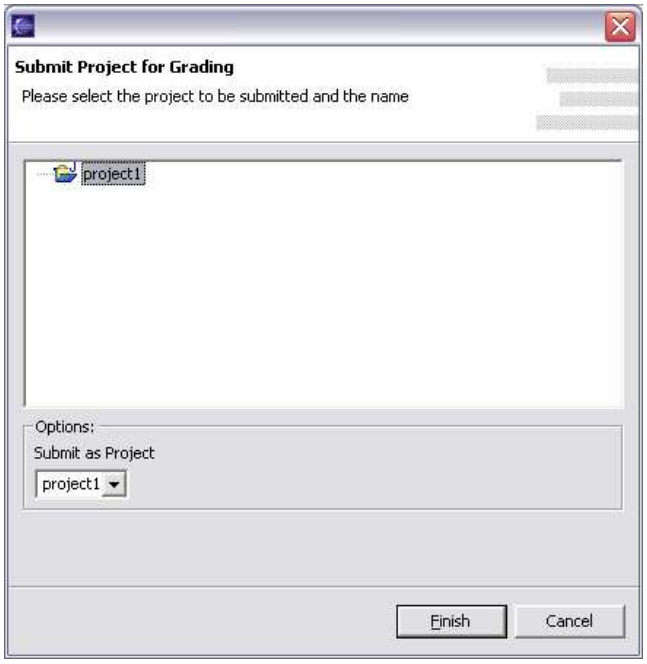

Figure 3: Project submission dialog

be achieved simply by switching to the Eclipse CVS perspective. Instructors can thereby use the versioning provided by the Eclipse CVS perspective to assist students with their projects.

6 Tailored tool bars

There are three main reasons for designing our own tool bar. Useful features (e.g., organize imports, format code) are often hidden in sub-menus. This causes new students not to use them, because they are overwhelmed by the complexity of menus. Second, we wanted to the students to be able to quickly access the features we added (e.g., checkout, submit, switch to Penumbra perspective). Third, since we decided to hide some menus (e.g., run) and reimplemented our own versions of these features we need to provide corresponding shortcuts to them. Rather than having multiple tool bars that contain one or two features the students will want to use, we merged the chosen shortcuts into a single Penumbra tool bar, highlighted in red at the top of Figure 1.

7 Help tutorial

Most help tutorials are designed for the person that knows what to look for. A programmer might ask: "What class must I import to use a certain feature?" On the other hand a new student will usually ask: "What is the problem? Why can I not use this feature?" The difference is that beginning students often do not understand the problem. Consequently, they do not know what question to ask of the help manuals. In addition, help manuals are complex and contain many details that distract and discourage students from looking for the right answer. To address this problem we decided to write our own help tutorial to guide the students through steps like setting up Eclipse/Penumbra, explaining the components, and how to use them. We also explain how to use existing functionality, such as debugging programs.

8 Conclusions and future plans

Penumbra is still a work in progress. We have implemented most of our initial ideas, and are only now beginning to experience their impact as Penumbra is "used in anger" by our students. As noted initially, our goals are relatively modest, in that we don't want to radically change the flavor and character of the Eclipse environment – ultimately, our students will be experienced programmers who will find full-featured Eclipse useful to them. Rather, we are guided by our instructional needs, and the perceived needs of our students. Nevertheless, we are particularly interested in experimenting with pedagogically-driven extensions of Eclipse for instructional use, and are keen to exchange ideas, experiences, and implementations with other groups using Eclipse for teaching programming.

References

[1] KÖLLING, M., QUIG, B., PATTERSON, A., AND ROSENBERG, J. The BlueJ system and its pedagogy. *Journal of Computer Science Education 13*, 4 (Dec. 2003).

[2] KÖLLING, M., AND ROSENBERG, J. Guidelines for teaching object orientation with Java. In *Proceedings of the 6th conference on Information Technology in Computer Science Education* (2001).

[3] STOREY, M.-A., SANSEVERINO, M., GERMAN, D., DAMIAN, D., DAMIAN, A., MICHAUD, J., MURRAY, A., LINTERN, R., CHISAN, J., LITOIU, M., AND RAYSIDE, D. Adopting GILD: An integrated learning and development environment for programming. In *Workshop on Adoption Centric Software Engineering* (May 2003).