GETTING ECLIPSE INTO THE CLASSROOM

Dwight Deugo
Carleton University, 1125 Colonel By Drive, Ottawa ON, K1S 5B6, Canada

Keywords: Eclipse, Teaching, Java, Open Source.

Abstract: In this paper, I describe an approach to integrating Eclipse into the classroom for the purpose of having students develop assignments using the Java, Scheme and Prolog programming languages. The approach attempts to overcome three problems. 1) Students wanting to learn about Eclipse have no idea where to start. 2) Educators wanting to teach Eclipse do not have available the breadth of material required to introduce students to it. 3) In most institutions, there is no room in the existing curricula to dedicate courses for explicitly teaching Eclipse.

1 INTRODUCTION

In a survey conducted in November 2006 by BZ Media (BZ Media, 2006), 66.3% of SD Times subscribers surveyed reported that developers within their organizations used Eclipse (Eclipse, 2008). Their primary reasons for choosing Eclipse were its low cost (free) and that it is open source. The major uses of Eclipse reported were for its Java Development tools, followed by its J2EE Standard tools, and then its Web Standard tools. These results are particularly significant to those in computer science (CS) education. It illustrates industry's continued move towards open source software and Eclipse. There are many reasons for this move, but at zero cost and better quality, commercial offerings are having a difficult time competing in the space. Given the move, it is important for CS educators to discuss the merits of open source software with their students and provide them with education on significant open source offerings, such as Eclipse. The need is clear: students need to know how to use Eclipse to be competitive in today's corporate software market and CS educators need to teach students about Eclipse. Industry is demanding it.

While Eclipse has established a large and vibrant ecosystem of technology vendors, start-ups, universities, research institutions and individuals, the availability of focused and organized educational material on Eclipse is lacking. This situation has created at least three problems.

The first one is students wanting to learn about Eclipse have no idea where to start. The second problem is CS educators wanting to teach Eclipse do not have available the breadth of material required to introduce students to Eclipse. The third, and perhaps the largest problem, is that in most CS institutions there is no room in their existing curricula to dedicate courses for explicitly teaching Eclipse.

I believe the simple solution to the last problem is to have Eclipse used as a common tool for developing various course concepts, rather than it being the course concept itself. The benefit is that students will learn about Eclipse as a side-effect of developing their assignments. A second benefit is that they will be able to use their Eclipse knowledge to work productively in industry. A final benefit is that by helping students to learn about Eclipse, CS educators will better understand the joint needs of their students and the companies they will work for using Eclipse.

At the recent ITISCE conference, I had a few CS educators tell me Eclipse was overly complex. While this is a debatable point, my answer is that even though Eclipse can be used as an integrated development environment (IDE) suitable for commercial development, portions of it can be made easier to use and tailored towards first year students.

My implementation of the solution to the last problem noted above and to the complexity issue has been to develop plug-ins for Eclipse that enable students to create Java, Scheme, and Prolog assignments all within the same environment. The plug-ins add new functionality to Eclipse that, while not suitable for commercial development, provides CS educators and their students with a single

Deugo D. (2009).
GETTING ECLIPSE INTO THE CLASSROOM.
In *Proceedings of the First International Conference on Computer Supported Education*, pages 388-393
DOI: 10.5220/0001965303880393

development environment for teaching and applying object-oriented, functional and logic programming concepts. The plug-ins add only the functionality required to develop language concepts, not the full blown capabilities of an industrial strength IDE.

In the background section we look at implementations of different open source and commercial IDEs. The approach section discusses how we have integrated the three different languages into Eclipse and how students interact with them to develop their assignments. The final chapter provides a summary.

2 BACKGROUND

Scaled down versions of Eclipse for Java development have been done in the past. One such version was produced by the GILD (Gild, 2008) project. However, it is no longer supported, with its last release done on January 3, 2006, and intended for use with Eclipse version 3.1. Another version is Penumbra. It is plug-in developed at Purdue University for use in their introductory programming classes. It was intended to ease the transition to the 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. Although neither version seems to be currently supported, they provide direction as to what a light version of Eclipse for Java development might look like. And, while not Eclipse based, BlueJ (BlueJ 2008) can also provide another view of an IDE for teaching object-orientation to beginners.

Only one Eclipse based Scheme development environment exists: The SchemeWay (SchemeWay, 2008) project. It provides a set of Eclipse plugins for the Scheme programming language and features a powerful, fully extendible S-expression-based editor that integrates seamlessly with any Scheme interpreter. However, this environment does not come with the source code and it is not targeted at first year students. While not Eclipse based, DRScheme (DRScheme 2008) provides an environment that provides an integrated programming environment designed specifically with the needs of beginners in mind.

One free Eclipse plug-in for Prolog exists created by an undergraduate student named Juliana Barby Simão (Simão, 2004). However, it was completed in 2004 and was not continued, even though it was reported that the project would continue duing 2004 as a graduate project. Not Eclipse based, JLog (JLog 2008) is an implementation of a Prolog interpreter, written in Java. It includes a built-in source editor, query panels, online help, animation primitives, and a GUI debugger. It could be easily wrapped within an Eclipse UI, providing Prolog for Eclipse users. The only difficulty with this idea is that JLog is under the GPL license.

Whether for Java, Scheme or Prolog development, the idea is to provide first year students with Eclipse-based light IDE. By doing so, the belief is that as students become more experienced with the light versions and the language IDE, they can and will want to transition to the full versions of the IDE. Having students wanting and using the different light versions of Eclipse should also impact faculty making them more likely to include material on the Eclipse Platform in their lectures in order to help students use these environments. Where do they get the material? From the LCMS outlined in the first part of the approach.

3 APPROACH

Central to our approach is the notion of community. We wanted to develop a community of users and a community of developers providing new features and languages to our IDE for education (IDE4EDU). To meet these goals the first decision made was to make the IDE open source. The implication is that whatever software we use and whatever software we develop, it must be open source. Since Eclipse is at the heart of the implementation, the implication is that, like Eclipse, the code will ship under the Eclipse Public License (EPL 2008). Using this license ensures that users are free to download and use the IDE without needing to pay any fees to anyone and developers are free to view, modify and add to the source code. The second decision made was to create a community project for the IDE. We have created an Eclipse Summer of Code project at the Eclipse Foundation (IDE4EDU, 2008). This subproject is part of the Eclipse Technology family. Creating this project ensures that the source code will have a home location, that key members of the project are identified, and that there is a community site for others to access to become involved with the

project. This project site also ensures that users have a common place to visit to get software updates and new versions of the IDE.

Rather than having different versions of the IDE for each language, another central part of our approach is to have all languages integrated into one IDE. The reason for this is simple: there is only one installation. Educators save class time, by only needing to describe the install process once, and students save time by only going through the download and installation process once. That said, the installation is very easy: download one zip file, unzip the file's contents to a location of choice, click on the identified IDE executable, and you are done.

Taking this approach also provides a common environment for students to work with different programming languages. While there are certainly many differences between the Java, Scheme, and Prolog programming languages, students always work within the confines of the Eclipse workbench, providing common actions to the edit, compile/interpret, and execution loop.

The following sections describe how each of the three languages are integrated into Eclipse, what other open source software was used in the process and the key features that are provided for students working on assignments using the various languages.

3.1 Java Lite

While Eclipse has been used extensively for Java development, as I mentioned, I have heard people say many times, "it is too complex for first year students". While part of me believes that we should expect students to deal with a certain amount of complexity, I can agree with the argument that when students are learning a language for the first time, many things that you would want for commercial development are not required by students developing their assignments.

The Java Lite integration into Eclipse simplifies the approach for student developing Java assignments. At a minimum, students need to create a project, create a class and run their code. Optionally, they can also create a Java package. If they don't create a package, all of the source code goes into the default package. And, to get to this functionality they have to navigate to the Java-Lite Eclipse perspective. That's it. What follows are these steps in more detail, provided to illustrate that the level of complexity is something that I believe we can expect first year students to be able to manage.

After downloading and installing the IDE4EDU from its project web site, the next step is to start it up by clicking on the IDE4EDU application icon. This will bring up the Eclipse workbench.

Selecting the menu Window -> Open Perspective -> Other…-> Java Lite opens the Java-Lite Perspective, shown in Figure 1. From now on, when a student starts up IDE4EDU they will go directly to the Java-Lite Perspective. This is a feature of Eclipse, always returning to the point where it was last closed from.

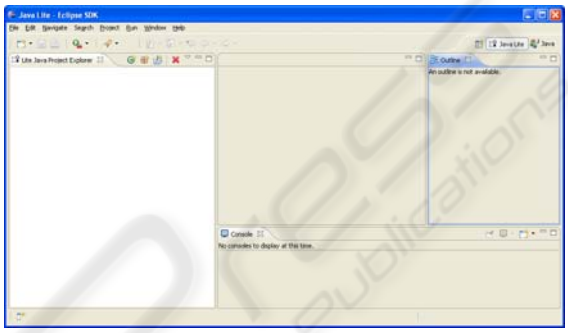

Figure 1: Java Lite Perspective.

The perspective has four distinct areas. The left area contains a Java Lite Project Explore view where students create their projects, packages, and classes. The right area provides an Outline view of the selected class's methods and variables. The bottom area is the Console view. The middle/top area is where the selected class is viewed and edited.

First students need to create a new project. In the JavaLite Package Explorer. From either the context menu or the JavaLite Package Explore toolbar menu, they select New Java Project, as shown in Figure 2. After entering the project name, as in assignment-1, students can create a new class. Using corresponding selection from the context menu or the toolbar, as shown in Figure 3, they fill in the class name as shown in Figure 4.

As a course progresses and students become more Java aware, they can select interfaces and a different superclass. However, to start with the defaults are all that is required. Closing the wizard opens the editor on the class, which has the default constructor and main method templates already written. After completing the code, selecting the class and running it as a Java application, the corresponding output will appear in the Console view, as shown in Figure 5.

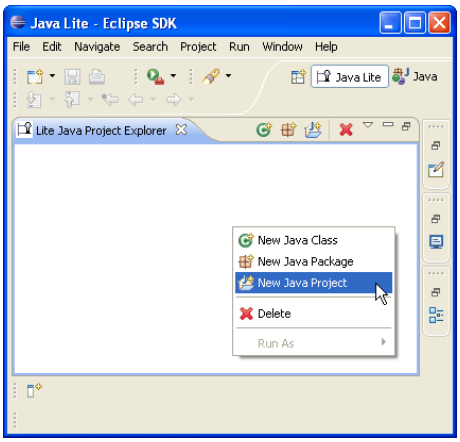

Figure 2: New Project.

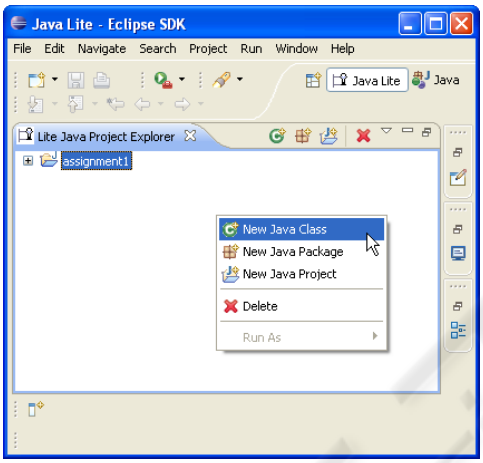

Figure 3: New Class Menu.

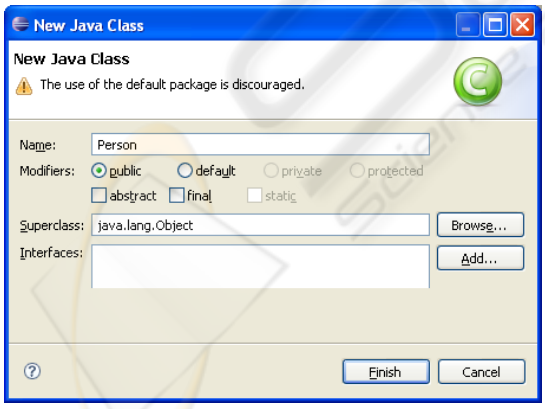

Figure 4: New Class Wizard.

Students can select their project and export it as a zip file. Given many schools have electronic submission of assignments, having an assignment in a single zip file makes assignment submission simple, and easy for teaching assistants to unzip, view, test, and mark the assignments.

As students progress in their Object-Oriented programming course, they can always switch from the Java Lite Perspective to the Eclipse's Java Perspective, giving them all of the features the Java perspective provides to developers, such as a debugger.

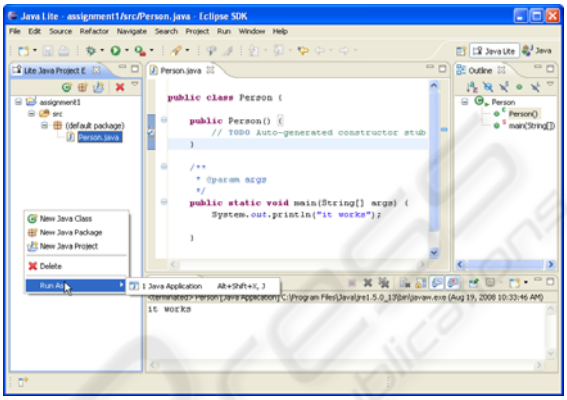

Figure 5: Editor and Consol Execution.

3.2 Scheme Lite

The integration of Scheme development into Eclipse is centered on any file created in Eclipse with the .scm extension. Since Eclipse does not have a Scheme interpreter included with it, the approach also embeds the Second Interpreter of Scheme Code (SISC, 2008) into Eclipse.

As with the Java-Lite integration, students must first create a project. In the Scheme integration, any project will do, even a Java project. However, it is suggested that students start with building a General Eclipse project, by selecting File -> New -> Other…-> General -> Project from the toolbar menu of any perspective and enter their assignment name, such as Assignment 2, as the project name. Next student select File -> New -> File to create the Scheme file, ending with the .scm extension. These actions result in what is shown in Figure 6.

The left area in Figure 6 contains the view with the project and Scheme file. The middle/top area contains a multi-tabbed editor. The editor lets students enter their Scheme code and then by switching tabs they can evaluate it. The other areas of the workbench contain views relevant to the corresponding perspective students are working in.

The editor has syntax highlighting and by clicking before or after any bracket they are shown the corresponding open or closing bracket, as shown in Figure 7. Selecting the evaluation tab, brings students to where they can evaluate the code in their

editor and other selected code, as shown in Figure 8. Hitting the Load Editor Contents button evaluates the code from the editor. Entering code in the Input and Eval area and selecting Eval Selection evaluates the corresponding code. Actions for clearing the console and resetting the Scheme environment are also available.

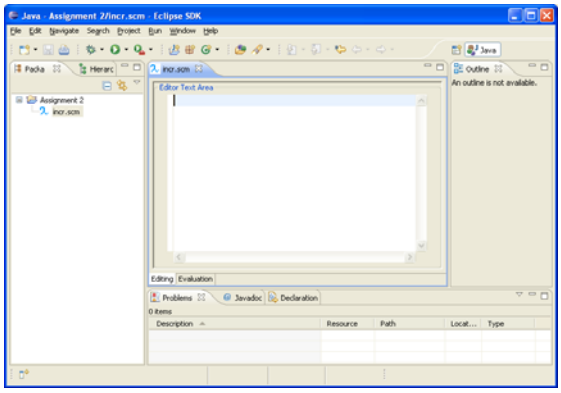

Figure 6: Scheme Multi-Tabbed Editor.

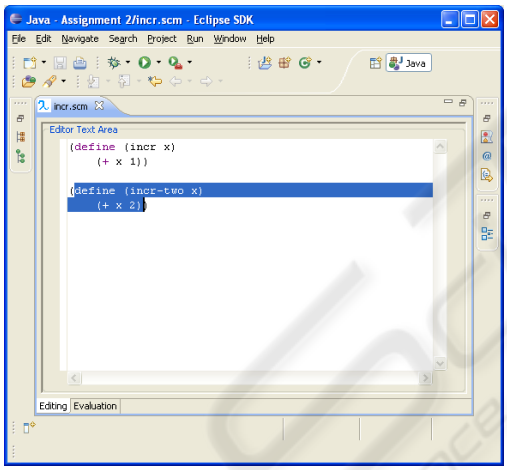

Figure 7: Scheme Edit and Evaluator.

While not full featured, students can work on the Scheme assignments along side their Java assignments using our Scheme integration into Eclipse. Exporting projects as zip files is also available.

3.3 Prolog Lite

The integration of Prolog development into Eclipse is centered on any file created in Eclipse with the .plog extension. Since Eclipse does not have a Prolog interpreter included with it, the approach also embeds JLog - Prolog in Java (JLog, 2008) into Eclipse.

As with the Java-Lite and Scheme-Lite integrations, students must first create a project. As in the Scheme integration, any type of project will do. Next, students select File -> New -> File to create the prolog file ending with the .plog extension. These actions result in the multi-tabbed editor shown in Figure 9.

The left area of Figure 9 contains the view with the project and prolog file. The middle/top area contains a multi-tabbed editor. The editor lets students enter their Prolog code, consult it, and then by switching tabs execute queries. The other areas of the workbench contain views relevant to the corresponding perspective students are working in.

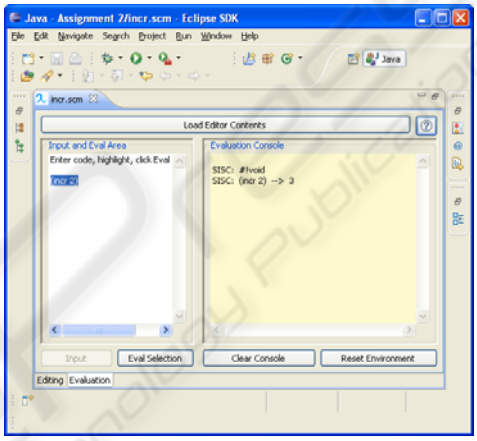

Figure 8: Evaluator.

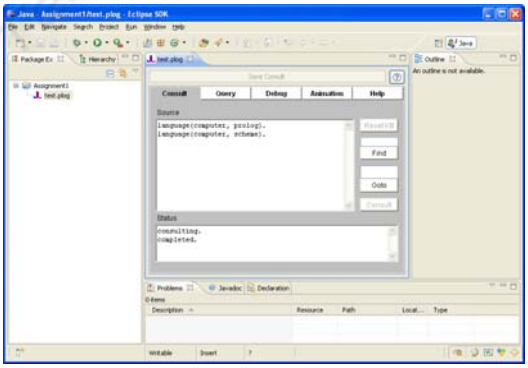

Figure 9: Prolog Multi-Tabbed Editor.

Selecting the Query tab, brings students to where they can run queries on the code they have consulted, as shown in Figure 10. Hitting the Consult button on the Consult page consults the code in the editor. Other Prolog related actions are also available.

While not full featured, students are able to perform the basic operations of consulting and querying Prolog code. Debugging is also possible and exporting projects as ZIP files is available.

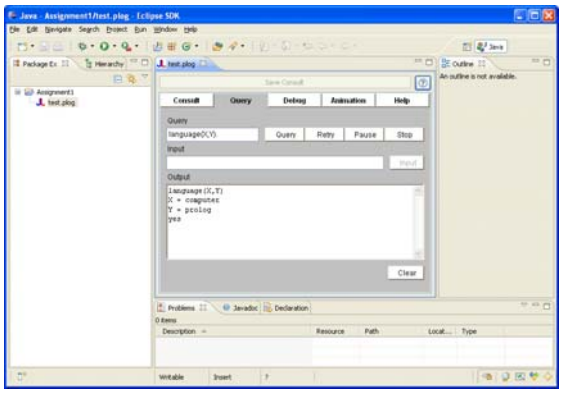

Figure 10: Query.

4 SUMMARY

While there is no room in CS curriculums to dedicate courses for explicitly teaching Eclipse, there is a need for students entering the workforce or going on co-op work terms to know how to use it. Our solution is to use Eclipse as a tool in the classroom for helping students learn how to program in Java, Scheme and Prolog. At Carleton University we use the combined IDE4EDU in our first year object-oriented programming courses and subsequent courses requiring Java. We also use the Scheme-Lite and Prolog-Lite integration in our third year Programming Paradigms course that teaches student Functional and Logic Programming concepts.

The three main features of the combined integration are important to us. First, students need only learn, and faculty need only teach how to download and install a single environment. Second, this is a community effort rather than a home-grown project. The intent is for the community to produce a plan and continue the growth of the combined integration. While there will always be project leads and committers, the goal is to encourage all interested parties to contribute to the project. Since it is important that others contribute to the project, having the project licensed under an Open Source License means everyone will have equal access source code providing them every opportunity to become involved and make improvements.

REFERENCES

BZ Media. Eclipse Adoption Study. Nov. 2006. 1 Dec. 2008. http://wiki.eclipse.org/images/d/d4/Eclipse_Adoption_Study_2006_Full_Version.pdf.

Eclipse, Eclipse.org Home Page. 1 Dec. 2008. http://www.eclipse.org/

GILD, Gild home page. 1 Dec 2008. http://gild.cs.uvic.ca/.

BlueJ. BlueJ home page, 1 Dec 2008. http://bluej.org/

SchemeWay, SchemeWay home page. 1 Dec 2008. http://sourceforge.net/projects/Schemeway

DR Scheme. DR Scheme home page. 1 Dec 2008. http://www.drScheme.org/

Juliana Barby Simão. 28 Mar. 2004. Prolog Plugin. 1 Dec 2008. http://eclipse.ime.usp.br/projetos/grad/plugin-prolog/index.html

JLog, JLog home page. 1 Dec 2008. http://jlogic.sourceforge.net/

EPL, Eclipse Public Licensee, 1 Dec. 2008. http://www.eclipse.org/legal/epl-v10.html

IDE4EDU, IDE4EDU home page. 1 Dec 2008. http://www.eclipse.org/projects/project_summary.php?projectid=technology.soc.ide4edu

SISC, SISC home page. 1 Dec 2008. http://sisc-Scheme.org/