A Design-First Curriculum for Teaching Java in a CS1 Course

Sally H. Moritz and **Glenn D. Blank**
Computer Science and Engineering Department
Lehigh University
Bethlehem, PA 18015 USA
sgh2@lehigh.edu, gdb0@lehigh.edu

Abstract

Pedagogies for teaching object-oriented programming in an introductory course are still under much debate. We present a design-first approach, which teaches problem-solving techniques using elements of UML. Objects are still introduced early in the curriculum. We also present two tools to support our curriculum: multimedia courseware to help students learn the basic concepts of objects and classes, and an IDE that includes both a UML interface and interactive tools to allow easy experimentation.

Keywords: CS 1, Java, object-oriented, programming, pedagogy, UML, objects-first

1. Introduction

Even though object-oriented programming has been with us for more than a dozen years, there is still much debate on how and when to teach it, as discussed on the SIGCSE mailing list and summarized by Kim Bruce in the December 2004 *inroads*[4]. Bruce suggested three basic approaches for instructors to try in a CS1 course: teach objects later, after students learn procedural constructs including loops and arrays; teach objects first, using pedagogical tools such as BlueJ, DrJava, or microworlds; or use a procedural or functional language for the first course and save object orientation for CS2.

We believe that a solid understanding of object orientation is crucial to a student's continued success in computer science, and that students learn best what they learn first. The curricula and tools that have been developed to support an objects-first approach offer many ways to help students visualize the abstract concepts of objects and classes. But they place little emphasis on teaching problem solving skills. Students who learn procedures for identifying a problem's requirements, breaking it into manageable pieces, and designing a solution are better equipped to apply the concepts they learn, and may better understand those concepts through their application.

We present a *design-first* curriculum, which adds to an objects first approach instruction in problem solving techniques using elements of UML. We support our curriculum with two tools. The first is CIMEL[3], multimedia courseware which presents object-oriented concepts through text, audio, graphics, and interactive exercises. The second is the Eclipse IDE for Java augmented with two plug-ins: DrJava, which provides an interactive environment for demonstrating Java code, and Omondo UML, which supports entry of class diagrams and generates Java code from designs. The first author developed and taught the curriculum at a local high school (in connection with the Lehigh Valley Partnership for Teaching Fellows, an NSF GK-12 project; see www.lehigh.edu/STEM), while the second author incorporated parts of it in a first semester CS course at Lehigh University.

2. The Curriculum

The curriculum is comprised of six units covering: use cases and class diagrams from UML; the concepts of objects, classes, and instances; identifying attributes and methods and designing a class; procedural concepts including if statements, loops, and basic character-based I/O; and graphical user interfaces using Swing. Students are also introduced to event handling and using Sun's online Java documentation. Many lessons are project-based; that is, new concepts are introduced in the context of a larger project in which they are immediately applied. Much of the class time is devoted to hands-on learning, with students working on an assignment or following a handout to carry out an exercise in Eclipse.

Pair programming is also used throughout the course. Each student is assigned a partner with whom she works for the semester. They alternate daily who types and who observes. Because all work in the high school class was done in the lab with the instructor present, the instructor was able to observe that students collaborated on problem solving, and both partners contributed equally.

A detailed description of each unit follows:

Unit 1 introduces the idea of software engineering. The process of building a software system is compared to building a house. Students list the steps in building a house, from a customer's first conception, through gathering requirements, looking at existing plans, creating a blueprint, and so on. Analogous steps in software development are discussed.

This unit also introduces students to objects as an abstract concept, and to classes as descriptions of

categories of objects. These lessons are supplemented by the CIMEL multimedia, and by several hands-on exercises in Eclipse. The exercises use the Shapes classes delivered with BlueJ and referenced in *Objects First with Java*[2]. Whereas BlueJ gives students a birds-eye view with a class diagram of the Shapes classes in an idiosyncratic notation, our setup immediately introduces students to a class diagram in UML. Interacting with this class diagram and DrJava, students create instances of Square and call methods to change their locations or colors, and see the results immediately. Students also examine the attributes of each Shape and learn about datatypes. A final exercise invites students to add their own code to an existing method and view the results.

Unit 2 teaches use cases and class diagrams in the context of developing a small application: a movie ticket machine (adapted from the idea of a train TicketMachine class in *Objects First with Java*). The instructor plays the role of the theater manager, whom the students interview to determine the system requirements. Working in pairs, they develop use cases, including a detailed description of the actions performed and the actors involved. They go on to design a Ticket Machine class. The class, including attributes and method signatures, is entered into the Eclipse UML interface. Students also develop a test plan to be executed after the code for each method is written.

In Unit 3, students learn procedural language concepts, starting with variables, assignment and arithmetic operations, followed by character-based printing and if statements. They complete exercises that give them practice in each new concept, then immediately apply what they've learned in coding the Ticket Machine's methods. They execute their test plans by creating a Ticket Machine instance and calling methods from DrJava's Interactions Pane.

Students learn how to build a character-based interface for the Ticket Machine in Unit 4. A class called "EasyReader" is provided to simplify reading character input. (We plan to replace EasyReader with the Scanner class when Eclipse supports JDK version 1.5.) Students also learn how to convert String data to integer or double datatypes using the parseInt and parseDouble methods from the Integer and Double classes. They code a simple interface that presents a menu of options to the user, and based on the user's input, performs the requested function, such as displaying the movie title and price, accepting money for tickets, or printing tickets.

Unit 5 introduces loops. Students apply while-loop logic to allow the user to enter a series of requests to the Ticket Machine. Students also learn about scope of variables and how the concept applies to a program with nested blocks, such as theirs. All students end the unit by completing a fully-tested character-based version of the Ticket Machine.

At this point, students are ready to create a similar project on their own, following the same steps they used to build the Ticket Machine. The high school students created a calculator (also with a character-based interface). Deliverables turned in and graded at each step of the process include a set of use cases, a UML class diagram, a fully coded and tested Calculator class, and a complete working program.

Unit 6 teaches students how to create a graphical interface using Swing. Students first learn how to display simple dialog boxes with JOptionPane. Next, they create a frame with simple graphical elements: text boxes, labels, and buttons. Through analyzing the code for a simple application, they learn about multi-threaded processes and how a listener works. They apply their knowledge in creating a graphical interface for their Calculator. The benefits of well-designed classes are stressed in showing the students how they can reuse the same Calculator class used in the character-based version for the graphical version of the program. They also learn how to add features to their calculator while learning how to use the Sun online Java documentation through an exercise that adds a square root function.

Unit 7 introduces applets by stepping through the process of converting the graphical Calculator application to an applet.

The complete curriculum outline, with worksheets, the Eclipse/UML/DrJava IDE, and other supporting materials, is available at www.lehigh.edu/stem/teams/dieruff. The curriculum does not refer to a textbook or recommend any particular texts. The worksheets include explanations of concepts and practical examples. In the high school course, they were used in lieu of a textbook.

3. CIMEL multimedia

CIMEL, *C*onstructive and *c*ollaborative, *I*nquiry-based *M*ultimedia *E-L*earning, is courseware designed to supplement a CS0 or CS1 course. By offering content on a breadth of topics in computer science, it presents a balanced view of the field to students who may think that computer science is only about programming. (Most of the multimedia complements *The Universal Computer: Introducing Computer Science with Multimedia.* See www.cse.lehigh.edu/~glennb/um for details.) A screen from the Objects and Classes chapter is shown in Figure 1. Documents and a demo are available at www.cse.lehigh.edu/~cimel.

CIMEL reaches out to students from a variety of backgrounds and with many different learning styles through these features:

- Multimedia personae model a diverse community of teachers and learners. The personae include two professors (one shown on the lower left), a teaching assistant, a reference librarian, and three students. In addition to graphical images, they speak in audio and/or text boxes. Personae model students and instructors studying the material together by working through interactive exercises

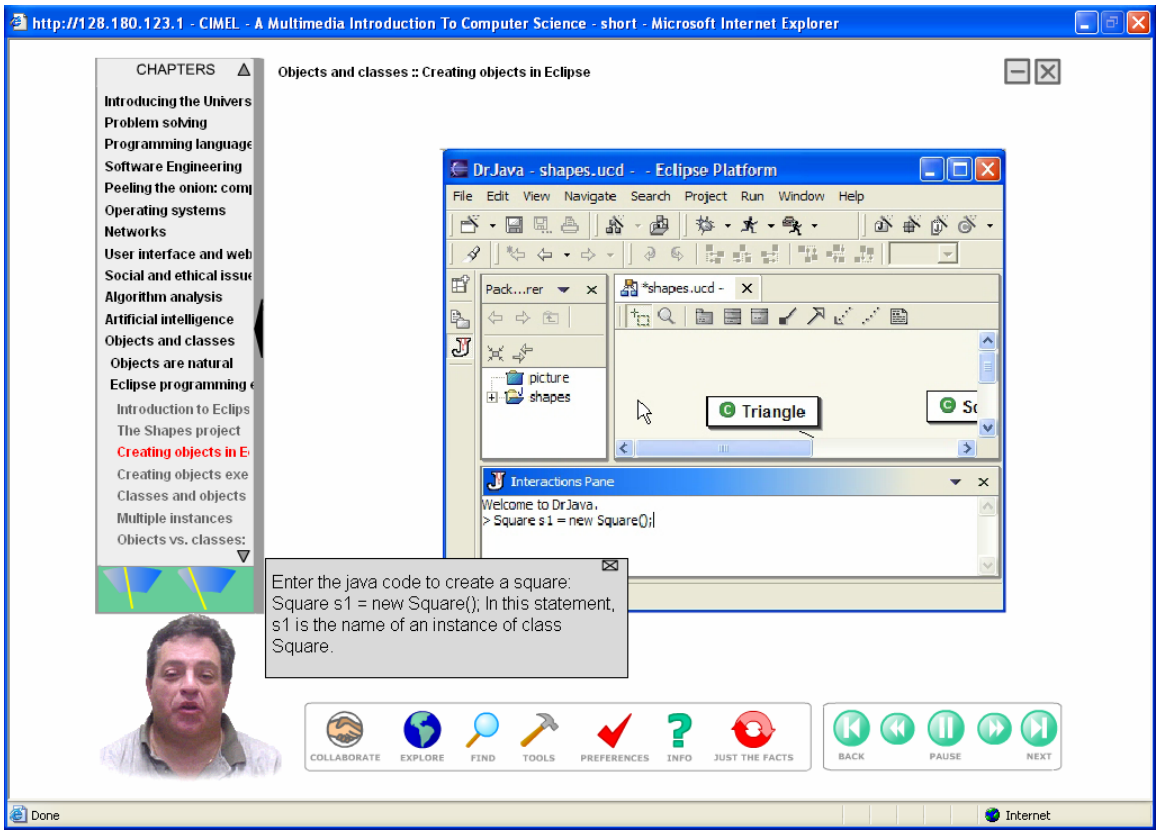

Figure 1: Screen Capture from CIMEL multimedia

and suggesting exploratory research on relevant topics using online information.

- The icons at the bottom give learners access to various tools, including EXPLORE (which links to resources on the internet and a tool which helps students identify emerging trends in the sources they find) and JUST THE FACTS (which lets students review just the text and graphics of the corresponding Flash content, i.e., without animation or interactive exercises). The PREFERENCES icon lets the user adapt the environment according to his or her personal learning style, including turning text boxes or audio on/off, toggling auto-advance or wait for next page, setting the timing rate where there is no audio narration, volume control, etc..
- Interactive exercises and quizzes enable the student to test her own knowledge, and identify areas for review. All results are stored in a database, which the instructor can use to determine which topics the class as a whole should review, or to identify students who are struggling and proactively offer help.

Experimental results indicate that first year students can learn Java "objects-first" using BlueJ, especially with the help of interactive multimedia [3]. We have modified CIMEL's chapter on Objects and Classes to introduce the Eclipse environment instead of BlueJ. The chapter uses real-world examples like car, house, and person to describe the basic concepts of object orientation. Attributes, methods, and primitive data types are also covered. Students are quizzed on these concepts through interactive exercises, with additional explanation or hints given when an incorrect response is entered. The multimedia then introduces the student to Eclipse and steps through some simple experiments using the Shapes classes. These exercises lead up to the student's assignment to add functionality to an existing method.

4. The Eclipse/UML/DrJava IDE

An integrated development environment that supports the aims of the curriculum is an important tool for successful learning. We needed these three capabilities in an IDE: 1. Inspection and entry of basic UML components (at minimum class diagrams). 2. Generation of basic code structures so students are shielded from some of the complexities of syntax. 3. Interactive capabilities, such as allowing entry and execution of a single line of code, to help students visualize what each statement does. Finally, many students would prefer to use a real world, practical IDE from the start, which they can continue using as they progress to more advanced courses and beyond academia.

No one IDE we considered had all these features, but we were able put one together based on Eclipse.

Eclipse (www.eclipse.org) is an open-source development environment created and supported by a consortium of technology companies, including IBM, HP, and Oracle. It is freely available and widely used by professional developers in both industry and academia. Its capabilities can easily be extended through the creation of plug-ins, allowing us to add functionality by selecting from many existing plug-ins or by creating our own. We found an implementation of our first and second requirements in Omondo UML, and our third requirement in DrJava.

Omondo UML (www.omondo.com) provides a window in which to create and save class diagrams. Panels to add attributes and methods to a class allow entry of all essential elements of each: datatype for attributes and parameters and return value for methods. Options such as public, private or protected, and static are given default values so students don't have to fully understand them at first, let alone remember to code them. Method panels also let developers enter method-specific body code as well as Javadoc-style comments. The interface is easy enough for a beginner to learn, and also offers features useful to experienced developers. Thus students are learning an environment they can continue to use in subsequent courses.

Students learn how to use the UML window before they are shown any code. The first exercise in Unit 1 introduces the Shapes classes via a class diagram. In the diagram, students can view the attributes and methods of a class, determine the parameters and return value of a method, and look at the documentation for each, all without reading a line of code. This allows students to concentrate on the concepts before learning the syntax of Java.

Every project assigned in the course enters Eclipse through the UML window. Because students develop use cases and a class diagram before writing any code, the class diagram is the first information they enter into Eclipse. From the class diagram, the code for the class header, attributes, and method stubs (including return value and parameters) is generated. Students may then enter the code for each method in Eclipse's standard code window (opened automatically when the student double-clicks on a class in the UML diagram), or through the UML interface (in the Implementation tab for each method). Using the code window is preferred because compiler messages and help are not yet available in the UML interface.

DrJava is a popular standalone IDE for beginners. Its goals include providing a simple, easy to use environment which nevertheless scales up to handle larger programming projects typically assigned in advanced courses. It also allows for quick and easy testing of portions of code[1]. Its Interactions Pane, separate from the code window, allows students to enter code snippets that create instances or call methods, interactively. Thus, a student can quickly test a method she just wrote, view the contents of a variable at a given point in time, or simply see the results of creating an instance of a class. The Eclipse plug-in version of DrJava provides the Interactions Pane within Eclipse.

Our curriculum makes use of the Interactions Pane in Unit 1, using the Shapes project provided with BlueJ. The four classes let students create and manipulate simple shapes drawn on a panel. The student enters one Java statement at a time in the Interactions Pane and observes the results. She creates an instance of Square and sees a square immediately appear in a window; she enters a call to the changeColor method for that instance, and watches the square change color. Creating and manipulating multiple instances at the same time reinforces student understanding of the ties between the Java statement, the concepts it represents, and the actual actions it performs.

5. Evaluation and Future Work

Six students recently completed the curriculum in a half-year inner-city high school course, and their performance is encouraging. Lecture comprised only about one-quarter of the class time; the rest was spent working on project assignments or completing scripted exercises or small standalone assignments within Eclipse. Students worked in pairs, at their own pace. All students indicated that they had no programming experience prior to the course; all were able to complete the assigned work. One student completed all assignments early, with almost no assistance from the instructor. That student had worked independently for half the semester, due to the frequent absence of his partner. The other students worked at a roughly uniform pace, with assistance from the instructor when they got stuck.

Students were graded on each step of the Calculator project, from use cases and class design, to the completed graphical interface. As the students worked, the instructor recorded the type and frequency of assistance requested, and verbally quizzed them on concepts and terminology. The students needed a refresher on terminology late in the semester. They seemed to forget some of the basic concepts of attributes and methods after spending several weeks focused on learning procedural code. Two students in particular needed reminders on creating instances and calling methods several times while working on the project.

No tests or quizzes were given during the semester, but five students took a final exam at the end of the course. Three scored 90% or higher. One student ran out of time, but did well on all but the last problem. Interestingly, for all students, most of the errors were on procedural code. They all did very well on the portion of the exam that covered object-oriented concepts.

The students also completed course evaluations. They said they liked that most of the class time was spent working in Eclipse. They liked using the handouts for reference. Although they said they didn't miss having a textbook, they did recommend adding material to some handouts. They also requested more class time be spent reviewing some of the more difficult concepts.

The course will be taught at the high school again with a few changes based on these observations. More emphasis will be placed on critical terminology, especially variables, methods, attributes, and instances. Review exercises will be added throughout the semester, and graded quizzes will be administered to give students a greater incentive to retain all the material. Specific material for new handouts and additional content for existing handouts have been identified.

We are also developing additional content for the CIMEL multimedia to provide alternative explanations to which students can refer. A new chapter on requirements gathering and design using the Movie Ticket Machine example is currently being scripted. It will step the student through the process of documenting use cases, identifying objects and designing classes. Like the Objects and Classes chapter, the new chapter will include the activities incorporated in the existing worksheets and exercises.

The Eclipse environment offers much assistance to students as they develop code within the IDE. Clear, accurate error messages replace many of the cryptic messages generated by the Java interpreter. As a student types in code, Eclipse finds matches on method names and makes suggestions in a pop-up window. And code stubs are generated by the UML interface. But these improvements are superficial. Error messages do not explain the concepts a student might be violating in a particular error, and code that is generated is not commented with explanations of why it was created or what it does. No help is available to the student creating his initial design in the UML window.

To address this shortcoming, three PhD students at Lehigh are developing an intelligent tutoring system that will interface with both Eclipse and CIMEL[5]. CIMEL ITS will have three functional components: an Expert Evaluator that compares a student's solution to one or more recommended designs for a problem and identifies errors within the student's design; a Student Model that represents the student's current knowledge state based on information from the student's performance on exercises and quizzes in the CIMEL multimedia, and on the student's work within Eclipse; and one or more Pedagogical Agents that choose a tutoring strategy tailored to the student's own learning style and present feedback and explanations of the concepts needing reinforcement.

Our goal in both refining the curriculum and providing additional online resources is to make computer science more accessible and appealing to a wider range of students. The new multimedia content and individualized tutoring will both supplement an instructor's presentation of the curriculum, and provide additional resources to students who need extra help but have limited access to a human tutor.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grants No. EIA-0087977 and 0231768 and PITA (Pennsylvania Infrastructure Technology Association).

References

[1] Allen, Eric, Cartwright, Robert, and Stoler, Brian. DrJava: A Lightweight Pedagogic Environment for Java. In *Proceedings of the SIGSCE Conference on Computer Science Education*, March, 2002.

[2] Barnes, D. & Kölling M. *Objects First With Java: A Practical Introduction Using BlueJ*, Englewood Cliffs, NJ: Prentice Hall, 2002.

[3] Blank, G. D., Pottenger, W. M., Sahasrabudhe, S. A., Li, S., Wei, F., and Odi, H. Multimedia for computer science: from CS0 to grades 7-12, *EdMedia*, Honolulu, HI, June 2003. Online at www.cse.lehigh.edu/~cimel/papers/EdMedia03.pdf.

[4] Bruce, Kim. Controversy on How to Teach CS1: A Discussion on the SIGCSE-members Mailing List. In *inroads – The SIGCSE Bulletin*, December, 2004.

[5] Wei, F., Moritz, S., Parvez, S., and Blank, G. D., A Student Model for Object-Oriented Design and Programming, *CCSCNE*, Providence, RI, April 2005. Online at www.cse.lehigh.edu/~cimel/papers/CCSCNE05.pdf.