USING BLUEJ AND BLACKJACK TO TEACH OBJECT-ORIENTED DESIGN CONCEPTS IN CS1*

Svetlana Kouznetsova, Ph.D
Department of Computing Science
Sam Houston State University
Huntsville, TX 77341
(936)294-1591
svk001@shsu.edu

ABSTRACT

This paper describes a sequence of Java programming assignments for the CS1 course which can be used to reinforce the basic concepts of object-oriented design in a logical, consistent way. By using the BlueJ environment, even programming novices are able to develop, over the course of four assignments, a working implementation of the card game Blackjack. Because the concepts are presented in the context of a fun and familiar application, the assignment increases their level of engagement. In addition, since students are allowed to discover for themselves the advantages of object-oriented design, they develop a better understanding of the material.

INTRODUCTION

The concepts of object-oriented design can be rather difficult to grasp for programming novices. Furthermore, it can be difficult for students to understand the purpose of object-oriented design. In my CS1 course, I introduce these concepts through a sequence of related programming assignments. All assignments are implemented using the BlueJ IDE [4, 6], which reduces the overhead involved in learning how to edit, compile, and execute code. BlueJ also enables students to easily create graphics to be incorporated into the project. This graphical representation is crucial, since it makes the assignment more interesting, thereby enhancing student engagement.

Introducing object-oriented concepts gradually with a fun and familiar application makes the process of learning them less daunting. In addition, students are able to discover for themselves the advantages of modularization of code and code reuse. In this paper, I describe a sequence of assignments in which students create, step by step, a rudimentary Blackjack card game application. This assignment was inspired in part by the PlayingCard example used in *An Introduction to Object-Oriented Programming* [1].

BACKGROUND

BlueJ is an integrated development environment for Java, developed by Michael Kölling and John Rosenberg [4, 6]. It has become a popular choice for introductory programming courses because of the ease with which absolute beginners can learn how to use its features [2, 3, 7]. Though admittedly the features of BlueJ are limited compared to full-fledged IDE alternatives such as jBuilder [8] and Eclipse [9], its functionality is more than sufficient for an introductory-level course. Even "lightweight" IDEs such as jGRASP [10] present a steeper learning curve for students, both in terms of installation and configuration of the software and the process of compiling and running their first Java program.

Advantages of the BlueJ environment include its graphical representation of the classes and objects within a project, and the simplicity with which students can interact with them through a sequence of pop-up menus. In particular, using the *Inspect* option of the pop-up menu associated with objects, students can directly see the values of the fields of an object. This allows them to immediately see the effect of a method invocation on that object and also simplifies the debugging process.

MOTIVATION

The introductory Java course covers fundamental concepts of object-oriented programming. The topics include data abstraction, classes and objects, state and behavior, methods, message passing, inheritance and subtyping, polymorphism and software reuse, overloading and overriding, dynamically-bound method calls, and data encapsulation.

These concepts can be rather difficult to grasp for beginning programmers. Furthermore, students often have difficulty understanding the reason for and advantages of using the object-oriented approach to software development. Hence, it is important in my opinion to introduce these concepts gradually through a fun and familiar application, so as to make the practical assignments less daunting.

At the first class meeting, I always ask the students to describe which aspects of Java or of programming in general are of interest to them. Invariably, at least a third of the class mentions a desire to write computer games. In response, I have developed a sequence of programming assignments which introduces object-oriented design concepts in a logical, coherent framework. Students are expected, over the course of four brief assignments, to develop an implementation of the card game Blackjack. Each subsequent assignment builds upon the previous one, making use of the classes already implemented, while at the same time introducing new Java features and design concepts.

ASSIGNMENT 0: Preliminaries

The BlueJ package is distributed with a library of sample code, including a project called "picture" [5]. This project contains a class Picture using an object of which a user can display onscreen a picture of a house composed of three simple shapes: square, triangle, and circle. The implementation of this project uses classes from the Java Swing and AWT APIs. Normally, using the Swing API is not feasible during the first weeks of the CS1 course. Hence, without using the BlueJ environment and the library of classes provided with it, incorporating graphics into the students' projects would be quite challenging.

In order to introduce the students to the syntax of a Java program, as well as to the features of the BlueJ IDE, I instruct them, as a guided lab exercise, to modify the existing code of class Picture. Students must write additional method calls and change parameter values in such a way that the colors of the components of the image change, and a picture of a fir tree appears next to the house. They are able to perform these tasks during the first week of the semester. Examples are shown in Figures 1 and 2.

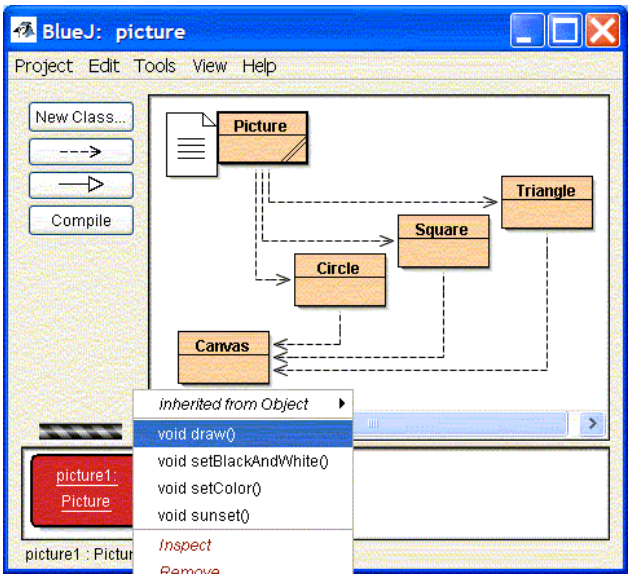

Figure 1: Classes in BlueJ project "picture" exercise

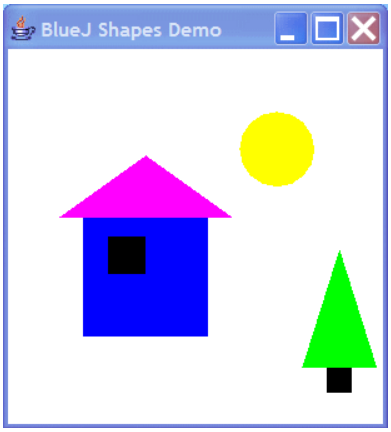

Figure 2: Results of first lab

As a preliminary "warm-up" programming assignment, based on what they learned in the lab, students are required to write a Java class that generates a picture of their own design. They are allowed to mimic the code of class Picture, but their resulting image must be significantly different. After the assignment is submitted, I show the most elaborate of the images created onscreen (with the permission of the authors). The creative component introduces an element of light-hearted competition among the students, which improves their attitude toward the assignment and increases their level of engagement. They invest extra time and effort trying to make their pictures look realistic. Some students introduce simple animation elements, and even figure out how

to modify the provided sample code for drawing simple shapes in order to best accommodate their graphical design requirements. An example of a student's "drawing" is shown in Figure 3.

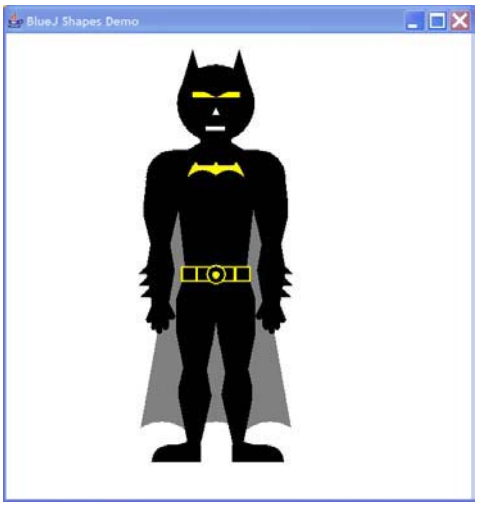

Figure 3: A student's implementation of "Assignment 0"

ASSIGNMENT 1: PlayingCard

In the first assignment of the four-assignment sequence mentioned above, students are required to implement a class PlayingCard that simulates an actual playing card. The purpose of this assignment is to introduce the concept of a Java class and its components: fields, constructors, and methods. Class PlayingCard includes fields representing the suit and rank of a card object. Students must write several (overloaded) constructors that allow creating an instance of the class using different combinations of parameters. In addition, students must provide methods for accessing the values of the fields of an object, changing its state (whether the card is facing up or down), and displaying an image of the card onscreen. The image contains a picture representing the suit icon (“hearts”, “spades”, etc.) which is composed of the same three simple shapes described above.

As a part of the same assignment, students write the method `main` in which they call constructors to create objects of class PlayingCard and invoke methods on these objects. As a result of this assignment, students learn the object-oriented concepts of encapsulation, information hiding, message passing, and even overloading. Once again, the graphical representation of a familiar object makes the assignment more engaging. An example of a student's rendering is shown in Figure 4.

ASSIGNMENT 2: CardDeck

In order to introduce the functionality of Java arrays, the next assignment requires that students implement class CardDeck, which represents a standard deck of 52 cards. Class CardDeck internally uses the Java array data structure to implement the collection of cards. Students learn how to declare an array, how to create an array object, and how to populate it with elements which are, in turn, objects of class PlayingCard. Students learn how to use a loop structure in order to create the elements of the array, and how to use the Java array (i.e., [ ]) notation to access and/or modify elements of the array. In my experience, students often have trouble grasping the concept of an array, and distinguishing between the whole array and a single element of it. By linking the concept of an array as a collection of objects to a familiar real-world “array” of cards, students understand the notion more readily.

In addition to the constructor of CardDeck, students are required to write methods that access and return PlayingCard objects stored at index 0 and at a random index in the array, and a method that “shuffles” the cards in the array by swapping pairs of elements at randomly selected indices. In doing so, students are introduced to the Java library class

java.util.Random and methods of it. The assignment also requires students to call in a loop the previously-written method of class PlayingCard that displays an image of a card onscreen.

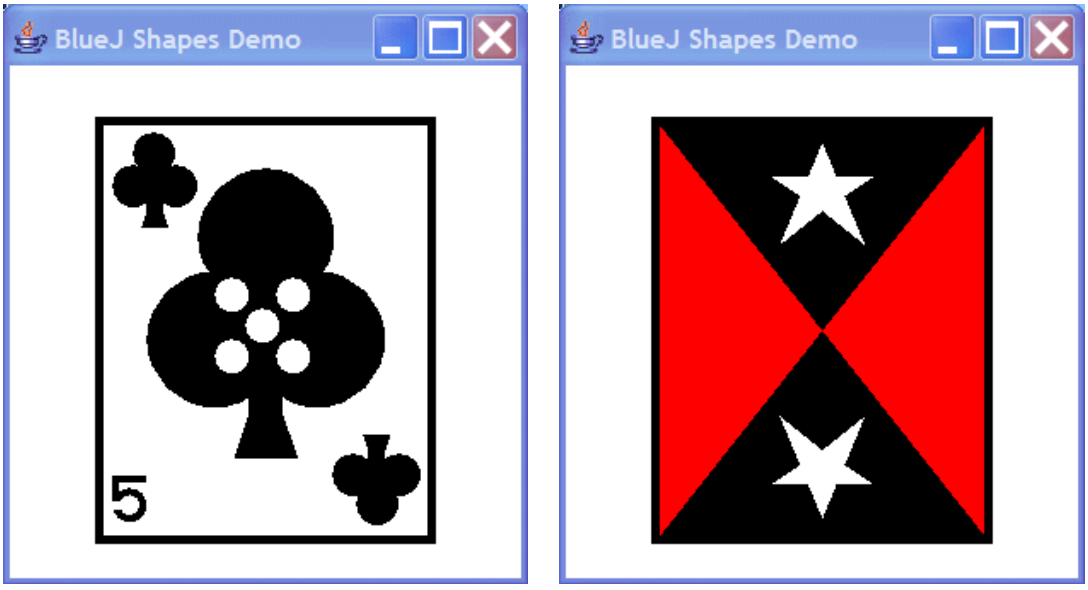

Figure 4: Onscreen images of a playing card (front and back)

ASSIGNMENT 3: GameDeck

The main purpose of this assignment is to introduce the concepts of inheritance and method overriding. Class GameDeck is implemented as a subclass of the previous class CardDeck. The difference between CardDeck and GameDeck is that the "number of cards" in GameDeck is decremented by one every time we access a card in the array. This simulates the dealing of cards in a game. In order to implement this more specialized version of a deck of cards, students must include in GameDeck a new field that represents the current number of cards in the deck, as well as a new method that "removes" a card from the given position in the array by shifting the elements of the array to the left. In addition to that, methods of class CardDeck that were used to access the first element and a random element of the array of cards must be now overridden to include a call to the new method that removes a card from the array. BlueJ allows students to see in the pop-up menu associated with an object of GameDeck the methods inherited from CardDeck alongside those explicitly written as part of class GameDeck. This helps them to better understand the mechanism of inheritance and code reuse.

ASSIGNMENT 4: Blackjack Implementation

This assignment is given towards the end of the semester, when students have gained some experience in writing Java code. It brings together many of the concepts learned throughout the course. For this assignment, students must write three new classes and link them with all of the previous classes in the project. The first class, DealerHand, implements the algorithm of playing Blackjack from the dealer's perspective. The class contains a field which keeps track of the current number of points in a hand, and a method that calls in a counter-controlled loop the method of the previous class GameDeck to deal cards one at a time from the top of the deck. As cards are being dealt, the current number of points in the hand is updated accordingly. Another method of GameDeck returns the value of the above field.

The next class, PlayerHand, is a subclass of DealerHand. It overrides the method for dealing cards: the cards are still dealt in a loop, but the loop is sentinel- controlled this time, and the method incorporates interaction with the user.

The third class, GameApp, contains the method main in which objects of DealerHand and PlayerHand are created. Methods for dealing cards are invoked on these objects. When these methods return, the winner of the game is determined according to the standard Blackjack algorithm.

The specific details of the algorithms for calculating points in each hand and for determining the winner of the game are figured out by students with practically no assistance from the instructor. By this point in the course, the students are able to write this code independently, making use of the techniques, concepts, syntax and basic structures of the Java language that they have learned during the semester.

CONCLUSION

While the application could be created using any development environment, I believe that its success in my class is dependent upon the use of BlueJ. BlueJ enables this project in two ways: (1) as a very simple-to-use tool for writing and editing code, and (2) through the provided sample code that allows users to create images onscreen without any prior knowledge of Java graphics (e.g., the Swing API). Because BlueJ minimizes the hurdles associated with graphics programming, novice students are able to create an interesting and fun application, which helps them master the basics of the object-oriented approach in the earliest stages of their CS coursework.

REFERENCES

[1] Budd, T., *An Introduction to Object-Oriented Programming*, Boston, MA: Addison-Wesley, 2002.

[2] Hagan, D., and Markham, S. Teaching Java with the BlueJ Environment. *Proceedings of Australasian Society for Computers in Learning in Tertiary Education Conference ASCILITE 2000*

[3] Kölling, M., Using BlueJ to Introduce Programming. In *The SPoP book*. Editors: J. Bennedsen, M.E. Caspersen, and M. Kölling (2006)

[4] Kölling, M., and Rosenberg, J., An object-oriented program development environment for the first programming course. *SIGSE Bulletin* 28 (1), 83-87, 1996.

[5] Kölling, M., Quig, B., Patterson, A. and Rosenberg, J., The BlueJ system and its pedagogy, *Journal of Computer Science Education*, Special issue on Learning and Teaching Object Technology, Vol 13, No 4, Dec 2003.

[6] Kölling, M. *BlueJ - Teaching Java*, 2006, http://www.bluej.org, retrieved November 2006.

[7] Van Haaster, K., and Hagan, D., *Teaching and Learning with BlueJ: an Evaluation of a Pedagogical Tool*, Information Science + Information Technology Education Joint Conference, Rockhampton, QLD, Australia, June 2004.

[8] *JBuilder*, 2006, http://www.borland.com/jbuilder , retrieved November 2006.

[9] *Eclipse*, 2006, http://www.eclipse.org , retrieved November 2006.

[10] jGRASP, 2006, http://www.jgrasp.org, retrieved November 2006