CPSC 150L Lab 10
An Introduction to Objects

Fall 2017

1 Overview

One of the characteristics, perhaps the most important, of a good designer is the ability to deal with abstractions or generalizations. In the process of learning how to program with methods, you were asked to create software tools with reusable functionality that could be deployed in other programs and circumstances as opposed to being used in a single instance. Examples include `factorial()`, `combinations()`, `leapyear()`, etc.

For decades programmers and software designers struggled with the problem of how to effectively create reusable code, and although it looks simple for methods like those cited above, it is not easy to extend this approach to more complex scenarios.

However, OOP(/OOD) (Object Oriented Programming/Design) is an approach that has radically changed our ability to deal effectively with complex problems. At the core of OOP is the notion of creating an effective way to use abstractions to create functionality that is reusable and capitalize on previous efforts, instead of "reinventing the wheel."

In this lab we will begin to investigate some of the basic principles of OOP by examining some classes, making some simple changes to them, and finally exploring some of the many abstractions available to us through the Java libraries.

BlueJ is an IDE that helps to convey what objects are about by enhancing the environment

© Christopher Newport University, 2016

with more visual elements. The idea of OOP does not require graphics or an IDE like BlueJ to work, but it is hoped that using BlueJ will deepen your understanding.

2 Using BlueJ with Objects

Fork and clone the `cpsc150l-lab10` repository from the 150 students group on Gitlab.

2.1 BouncingBall

In this project you will find a number of classes including `BouncingBall` and `Canvas`. Leave them alone as we will be using them later in the lab. For now create another class named `MyTester`. Now open the `MyTester` class and remove all of the template code.

Before we create and draw a canvas, lets look at the `Canvas` class by looking at the data used to define characteristics of the canvas and the methods that can be used to interact with the canvas. Double click on the `Canvas` class. Next create a text file on your screen (using notepad) and then copy and paste the heading of every method from the `Canvas` class into the notepad window; JUST THE HEADINGS of the methods. Next copy the names of all variables (2 of them) that are labeled with comments as "`instance variables`". The content of this file effectively represents the programmer's interface to the `Canvas` class.

2.2 Using the Bouncing Ball Project

Now let's focus on two methods from the Canvas class.

Code:

```
public Canvas(String title)
```

and

Code:

```
public void setVisible(boolean visible)
```

We are going to create a simple main routine that will create a canvas and draw it on the screen. Write a main routine in the `MyTester` class and add the following two statements.

Code:

```
Canvas myFirstCanvas = new Canvas("first one");
myFirstCanvas.setVisible(true);
```

Compile the program and run it by invoking the main routine of your `MyTester` class.

You should find a window on your screen with the title "first one". It is reasonable to suggest that you might understand what a "constructor" function is now. In the first statement you "constructed/created" a `Canvas` and gave special instructions to name the canvas "first one".

Change the name of the canvas and rerun it.

This constructor provides you a simple interface for constructing a canvas. If you wanted to have more control over the creation of the canvas, you could use the other constructors provided. You can identify the constructors because they look like methods, but have the same name as the class.

Find the other two constructors,

Code:

```
public Canvas(String title, int width, int height, Color bg_color)
```

and

Code:

```
public Canvas(String title, int width, int height).
```

Use them to create two additional canvases: one of different size and one of a third size and a `Color.blue`. Change the title and make each canvas progressively smaller so that each canvas can be distinguished.

You will encounter compilation error(s) but this can be fixed by adding the following two statements to the top of your `MyTester.java` file

Code:

```
import javax.swing.*;
import java.awt.*;
```

Question 1: *Show your instructor your work when you have the three frames on the screen.*

Exercise 1 Complete

Run:

```
git add .
git commit -m "Completed exercise 1"
git push origin master
```

Classes are often designed with multiple constructors to match the interface to the need. Not too simple... not too complicated. As you witnessed in this example, the different constructors used parameters to customize the `Canvas` object. In the case of the `Scanner` class, an object can be created to read from various devices, but we have been choosing the keyboard as our input device of choice by supplying "`System.in`" to the constructor function. So this clears up the creation of a `Scanner` using a constructor method. In later courses you will encounter examples that create a `Scanner` which reads data from a text file in contrast with this example reading from the keyboard.

There are more features that we can explore in this `BouncingBall` project so let's keep going.

Comment out the code that created the second and third canvasses and make the following changes to the main routine. Use the

Code:

```
public BouncingBall(int xPos, int yPos, int ballDiam, Color ballColor)
```

constructor from the `BouncingBall` class to create a ball. Once you create the ball, invoke your `Canvas`'s method

Code:

```
public void add(Shape shape, Color color)
```

method and pass it

Code:

```
ball.getCircle()
```

and

Code:

```
ball.getColor()
```

so that we can have `Canvas`'s

Code:

```
public void draw()
```

method draw it.

Compile your new `MyTester` class and run it.

Question 2: *Show your instructor your work when you have the Ball drawn on your screen.*

Exercise 2 Complete

Run:

```
git add .
git commit -m "Completed exercise 2"
git push origin master
```

Amazing! With the right interface, you were able to create a canvas and draw a ball on it with just a few statements. But our objective was to do something a little more exciting so press on.

In the `Canvas` class provided that does a more effective job of entertaining you via a method called `bounce()`. Create an object of the class `Canvas` and call its "`bounce()`" method.

Two statements and it bounces a ball!

Question 3: *Show your instructor your work when you have the Ball bouncing.*

Exercise 3 Complete

Run:

```
git add .
git commit -m "Completed exercise 3"
git push origin master
```

Now for the last step, but the most difficult one. Abstraction (the discussion topic which started this lab) is the power to take a function and generalize it for more uses. Your objective

is to take the `bounce()` method and generalize it to work for any two balls, specifically balls passed into the method as parameters.

This will allow a programmer to create two balls in their own method, pass the balls to the bounce method, and the bounce method will draw the balls and bounce them. Our abstraction is to allow the method to work with any two balls, controlled by the calling routine.

1. Modify the `bounce()` method so that it does not create the balls, but uses two balls that are passed into it as parameters. Rename the `bounce()` method to,

   Code:

   ```
   public void bounce(BouncingBall ball, BouncingBall ball2)
   ```

   in `Canvas`.

2. Modify the `bounce` method to use the parameters sent.

3. Modify your `MyTester` class to create two balls (not draw them), effectively moving this code from the `bounce()` method.

4. Invoke the new `bounce()` method of the `Canvas` object you created in the previous step, passing it the two balls you just created. Make it interesting by changing the characteristics of the balls from the original colors etc.

Question 4: *Show your instructor your work when you have the two new balls bouncing on your screen.*

Exercise 4 Complete

Run:

```
git add .
git commit -m "Completed exercise 4"
git push origin master
```

---

3 Common Mistakes

Some solutions to common mistakes are as follows.

1. Be sure to read the instructions carefully. If there is an error, go back and read through the instruction thoroughly.
2. You do **not** need to edit `BouncingBall.java` and `CanvasPanel.java` and the only thing you should edit in `Canvas.java` is the `bounce` method.
3. Your `MyTester.java`'s `main` method should be *simple*. You do not need any loops or conditional statements inside of it.
4. Pay attention to the constructors for `BouncingBall` and `Canvas`.

4 Tutorial

4.1 Objects

Even though you will discover many interesting aspects to OOP, the place to begin thinking about objects is that they provide a better way to create an effective interface for the programmer... you! If you were to take your grandparents to the local cellphone store and show them some of the newest phones on the market, they would likely be overwhelmed

given that they don't even use computers frequently, much less understand how cell phones can effectively serve as both phones and computers.

They don't know which buttons to push and don't understand the purpose of the buttons once explained to them. The bottom line... the interface is wrong for them. They are forced to confront features that they don't need or understand, and it overwhelms the entire experience. This clouds the fact that there are a number of uses that they could make of the phone if presented to them in the right way that would make the use of the phone much more enjoyable and effective.

The opposite end of the spectrum is also true. Have you ever used a DVR recorder and wanted to record a show, but couldn't because you couldn't find the remote. Often the DVR players don't provide the recording functionality in the interface on the recorder and force you to use the remote. The interface misses again... here the buttons on the recorder are too few in number.

We are going to start with an example which is taken from BlueJ tutorials, using a `BouncingBall` and a `Canvas` on which we will draw the ball. If I asked you to write a program to create a ball bouncing on the screen, most of you would freak, and rightfully so. It is not a simple process to draw a figure on the screen, but do you really need to address all of the details of the process or are there some complexities that you would really rather ignore? Like the DVR and the cell phone, getting the complexity of the programming at the right level is difficult.