CPS109 Lab 2

Source: Big Java, Chapter 2
Preparation: read Chapter 2 and the lecture notes for this week.

Objectives:
1. **To practice using variables**
2. **To practice creating objects**
3. **To practice calling methods, using parameters and return values**
4. **To practice browsing the java API documentation**
5. **To see the difference between objects and object references**

Instructions:
The lecture notes indicate that you should do the following exercises from Big Java: Exercises P2.1, P2.2, P2.3, P2.7, P2.8, P2.9, P2.10. In this lab we lead you through those exercises and discuss them with you. The parts you must hand in are to be collected in a file called **lab2.txt** and submitted to your TA as you did with lab1. Be sure to include your **name** at the top of your submission. You can consult with your colleagues and the TA regarding how to do the lab, but what you submit must be your own **individual** work. Academic integrity is taken very seriously at Ryerson. See the course management form for more information.

1. **Exercise P2.1.** *Write an AreaTester program that constructs a Rectangle object and then computes and prints its area. Use the getWidth and getHeight methods. Also print the expected answer.*

Discussion: This question could sound intimidating if you had not already read Chapter 2. There you learn that the Rectangle class is defined in the Java class library, in particular, in **java.awt**. To use that class, you must first **import** it, as shown in the **MoveTester** program of Chapter 2, shown below. When you are learning to program, most programs that you write will closely follow some model. In Lab 1, the model was the **HelloWorld** program, which introduced the static method **main** which is used to start all stand alone applications, and the method **println** of the class **java.io.PrintStream**, of which **System.out** is an object. In this lab the model is the **MoveTester** program, which shows you how to import the Rectangle class, how to make a Rectangle object, how to invoke a method on the Rectangle object, and how to test your program by printing out what you expect the answer will be.

Step 1: Cut and paste the MoveTester program into an editor (as you did with the HelloWorld program in lab1).
Step 2: Make the name of the file match the name of the class, i.e., call the file **MoveTester.java**.
Step 3: Compile and run the program. This much should work, since it is a program prepared and tested already. So far you have just reviewed how to use the compiling environment that you have chosen. Look at the output and understand what the program has done and why the test output matches the expected output.
Step 4: Cut and paste MoveTester into **AreaTester.java**. Exactly how you do this depends on your environment, as discussed in lab 1. Try to compile the program, and as you might

expect, it does not compile, even though it did a moment ago in **MoveTester.java**. This is where you have to start changing things to make it do what you want.

```
/**
   A program which creates a Rectangle object and translates it.
   The program illustrates calling a method, printing a value and printing
   what you expect the value to be, so that you can check it.
   @author Cay Horstmann
*/
import java.awt.Rectangle;

public class MoveTester
{
  public static void main(String[] args)
  {
    Rectangle box = new Rectangle(5, 10, 20, 30);

    // Move the rectangle
    box.translate(15, 25);

    // Print information about the moved rectangle
    System.out.print("x: ");
    System.out.println(box.getX());
    System.out.println("Expected: 20");

    System.out.print("y: ");
    System.out.println(box.getY());
    System.out.println("Expected: 35");   }
}
```

Step 5: The first thing to change, if you have not done so already, is the name of the class, from **MoveTester** to **AreaTester**. Once you have done that, the program should compile and run as before. It just does not do what we want for P2.1. If you keep making small changes followed by compile and run, you will not stray into the dark woods of a program that has a million errors.
Step 6: Change the comment at the top to something simple, like **Solution to P2.1.** Change the @author line to replace **Cay Horstmann** with your name. Now make changes inside the main method so that the program becomes a solution to P2.1. For example, you could change the comment **// Move the rectangle** to something like **// Calculate the area** Then replace **box.translate(15, 25)** with a line like **double area = box.getWidth() * box.getHeight()**. Finally change the print statements so that you print out the area and what you expect the area to be.
Step 7: Compile and run the program and check that the calculated area matches the value you expected. If it matches, copy and paste your program to **lab2.txt** as your answer to question 1. If it does not match, figure out and fix your error and repeat this step.

2. **Exercise P2.2.** *Write a PerimeterTester program that constructs a Rectangle object and then computes and prints its perimeter. Use the getWidth and getHeight methods. Also print the expected answer.*

Since you solved question 1, you can repeat those steps, perhaps now using **AreaTester** as your new model for making **PerimeterTester.java**. The problem is so similar to the first problem, you might wonder what is the point – you already get the idea. The point is **practice**. Treat programming like a sport or language that you practice to improve your facility and coordination. Programming is 10% understanding and 90% practice. When you are done this question you will have pasted your solution into **lab2.txt.**

3. **Exercise P2.3.** *Write a program called FourRectanglePrinter that constructs a Rectangle object, prints its location by calling System.out.println(box), and then translates and prints it three more times, so that, if the rectangles were drawn, they would form one large rectangle.*

The model for this program could be **MoveTester**. If we keep the line **Rectangle box = new Rectangle(5, 10, 20, 30);** then picture the box as being at position x = 5, y = 10 with width 20 and height 30. You confirm that by a line **System.out.println(box)**, as suggested. Now, translate the box to the right by its width so that the original box and the translated box are side by side. The call would be **box.translate(20, 0)**, followed by another printing of the box. Do something similar two more times so that the box is translated down by its height and then to the left by its width. Compile and run your program, checking the output to confirm that it is what you explect. Paste your solution into **lab2.txt.**

4. **Exercise P2.7.** *The Random class implements a random number generator, which produces sequences of numbers that appear to be random. To generate random integers, you construct an object of the Random class, and then apply the* ***nextInt*** *method. For example, the call* ***generator.nextInt(6)*** *gives you a random number between 0 and 5. Write a program DieSimulator that uses the Random class to simulate the cast of a die, printing a random number between 1 and 6 every time that the program is run.*

```
/**
   Solution to P2.7 using the Random class.
   @author put your name here
 */
import java.util.Random ;

public class DieSimulator
{
   public static void main(String[] args)
   {
      Random generator = new Random() ;
```

```
        //todo: change the following line
        System.out.println("Hello World") ;
    }
}
```

To speed things up for you I have given you the model. You only need to change one line. Compile and run it to see that it generates a different random number each time. Paste your program into lab2.txt.

5. **Exercise 2.8.** Use your answer to P2.7 as a model to write a program that picks a combination in a lottery. The combination consists of 6 numbers from 1 to 49, such as 5, 42, 3, 1, 49, 42. Note that numbers can repeat and that 0 is not allowed. Since the generator.nextint(48) would give you a random value from 0 to 48, you would add 1 to that to make a value from 1 to 49. Print out a sentence such as "Play this combination—it'll make you rich!", followed by the 6 random numbers separated by commas. After testing it, append your solution to lab2.txt.
6. Exercise 2.9. Chapter 2 discussed the **replace** method of the String class, defined as **public String replace(String target, String replacement)**. It gave some example calls, such as **river.replace("p", "s")**, where **river** is a String variable. Suppose you want to check whether this method replaces the first "p" with "s", or does it replace all occurrences of "p" with "s". To make this check, you could try it out, or you could look it up in the Java Application Programming Interface (API). Open a browser and enter the URL (uniform resource locator) http://java.sun.com/javase/6/docs/api/index.html to go to the Java documentation for Standard Edition 6. You will see three panels in the browser window. In the top left panel, find and click on **java.lang** which is the part of the library that contains the **String** class. The **java.lang** collection of classes is always available to you without using the **import** statement. In the lower left panel find and click on the **String** class. Now in the right panel, under the **Method Summary** section, find the method String `replace(CharSequence target, CharSequence replacement).` Oh, well, it has **CharSequence** in place of **String**, but for now, assume that String behaves like a CharSequence, which it does. As your answer to this question, copy and paste the description of the method from the java API to lab2.txt. The program that we are supposed to write for this question encodes a string by replacing all letters "i" with "!" and all letters "s" with "$". The program is given below for you to compile and try. You will find that it has a bug. Fix the bug and append the program to lab2.txt.

```
/**
   Solution to P2.8 using the replace method
   @author put your name here
 */

public class Encoder
```

```
{
    public static void main(String[] args)
    {
        String river = "Mississippi" ;
        String encoding = river.replace("i", "!") ;
        encoding = river.replace("s", "$") ;
        System.out.println(encoding) ;
        System.out.println("Expect: M!$$!$$!pp!") ;
    }
}
```

---

7. **Exercise P2.10.** Use the above Encoder program as a model to make a program that switches the letters “e” and “o” in a string. Note that you do not want to undo changes that made with the first replacement. Demonstrate that the string “Hello, World!” turns into “Holle, Werld!”. Append your program to lab2.txt and send lab2.txt to your TA.
