CSCI 1101 – Winter 2017
Laboratory No. 3

This lab is a continuation of the concepts of object-oriented programming, specifically the use of static variables and static methods, and object interactions. If you finish the lab early, work on your assignment.

Note:

1. *All submissions must be made on Brightspace (dal.ca/brightspace).*
2. ***Submission deadline is 11.55 p.m. (5 minutes to midnight) on Saturday, February 4th, 2017.***
3. *Put the java source code files and the text outputs for each exercise in a folder. Zip the folder into one file and submit the zip file.*

4. Marking Scheme:
Each exercise carries 10 points. Your final score will be scaled down to a value out of 10.
Working code, Outputs included, Efficient, Comments included: 10/10
No comments: subtract one point
Unnecessarily long code and inefficient program, improper use of variables: subtract one point
No outputs: subtract two points
Code not working: subtract up to six points depending upon the extent to which the program is incorrect.

***5. Error checking**: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input.*

***6. Testing your code and generating the outputs:** If the test data is included in the question, use that to test your classes. In addition, test it with two more input sets. Otherwise, create your own test data and run your program for at least 3 input sets such that they cover the range of results expected.*

Exercise 1 (a) : This is an example on static variables that we discussed in class. Run the code and trace the output so that you understand the operation of the program. **No submission required for this part.**
Warning: Cutting and pasting code may cause errors!

```
public class TurnTaker
{
	private static int turn = 0;
	private int myTurn;
	private String name;
	public TurnTaker(String n, int t)
	{
		name = n;
		myTurn = t;
	}
	public String getName()
	{
		return name;
	}
	public static int getTurn()
	{
		turn++;
		return turn;
	}
	public boolean isMyTurn()
	{
		if (turn==myTurn)
```

```
                return true;
          else
                return false;
    }

    public static void main(String[] args)
    {
          TurnTaker person1 = new TurnTaker("Romeo", 1);
          TurnTaker person2 = new TurnTaker("Juliet", 3);

          for(int i = 1; i<= 5; i++)
          {
                System.out.println("Turn = " + TurnTaker.getTurn());
                if (person1.isMyTurn())
                      System.out.println("Love from "+ person1.getName());
                if (person2.isMyTurn())
                      System.out.println("Love from " + person2.getName());
          }
    }
}
```

Exercise 1(b): Copy the class TurnTaker.java into another file TurnTaker1.java. Modify TurnTaker1.java so that you get the following output:

```
Turn = 1
Love from Romeo
Love from Juliet
Turn = 2
Love from Romeo
Love from Juliet
Turn = 3
Love from Romeo
Love from Juliet
Turn = 4
Love from Romeo
Love from Juliet
Turn = 5
Love from Romeo
Love from Juliet
```

What to submit: TurnTaker1.java and the output generated by your code in a text file.

Exercise 1(C): Make another copy of TurnTaker.java into TurnTaker2.java. Modify TurnTaker2.java to do the following: The program should prompt the user for the number of turns. Each odd turn should print "Love from Romeo" and each even turn should print "Love from Juliet", except the last turn when it prints both "Love from Romeo" and "Love from Juliet".
Run the program for at least three different inputs.
Here's a sample dialog:

```
How many turns? 7
Turn = 1
Love from Romeo
Turn = 2
```

```
Love from Juliet
Turn = 3
Love from Romeo
Turn = 4
Love from Juliet
Turn = 5
Love from Romeo
Turn = 6
Love from Juliet
Turn = 7
Love from Romeo
Love from Juliet
```

What to submit: TurnTaker2.java and the outputs generated by your code for the input sets in a text file.

Exercise 2: Your friend Chuck owns several pizza stands distributed throughout the town. You are the java expert who will help Chuck keep track of the number of pizzas sold by the stand as well as the total number of pizzas sold in all the stands. For this, define a class named PizzaStand that has an instance variable for the Pizza Stand's ID number and an instance variable for how many pizzas sold in that stand that day. Add a static variable that tracks the total number of pizzas sold by all the stands. Add another static variable that specifies the cost per pizza.

Add the following methods:
A constructor that sets the ID number to some value and the number of pizzas sold by that stand to 0.
A method named justSold that increments the number of pizzas sold by that stand by 1.
Another method to return the number of pizzas sold by that stand.
A static method to set the cost per pizza.
A static method that returns the total number of pizzas sold by all stands.
A static method to return the total sales.

Test the class for at least three different input sets (each with at least five pizza stands) that each sell a number of pizzas during the day.

A sample screen dialog is given below. The cost of the pizza is set to $5.00.

```
Pizza Sales:
1    2      (means that Pizza Stand 1 sold 2 pizzas)

2    1

3    1

4    1

5    1

Total pizzas sold: 6
Total sales: $30.00

Process completed.
```

What to submit: Pizza.java, PizzaDemo.java and the outputs generated by your code for the input sets in a text file.

Exercise 3: Design a class named MyInteger. The class contains:

- An int instance variable named value that stores the int value represented by the object.
- A constructor that creates a MyInteger object for the specified int value.
- A get method that returns the int value.
- Static methods isEven(int), isOdd(int) and isPrime(int) that return true if the specified value is odd, even, or prime, respectively, and false otherwise.
- Methods isEven(), isOdd(), and isPrime() that return true if the value in this object is even, odd or prime, respectively, and false otherwise.
- Static methods isEven(MyInteger), isOdd(MyInteger) and isPrime(MyInteger) that return true if the value of the specified object is even, odd, or prime, respectively, and false otherwise.
- Method equals(int) that returns true if the value in this object is equal to the specified integer.
- Method equals(MyInteger) that return true if the value in this object is equal to the value in another object.
- Static method parseInt(char[]) that converts an array of numeric characters to an int value.
- Static method parseInt(String) that converts a string into an int value.

Note: if c is a character from '0' to '9', then (int)c-48 converts c to an integer value.
You may assume that the user enters only digits.

Here's a demo class that tests the above methods and the output. Change the values and run the code at least three times for different input sets.

```
public class MyIntegerDemo
{
   public static void main(String[] args)
   {
      MyInteger n1 = new MyInteger(5);
      System.out.println("n1 is " + n1.getValue());
      System.out.println("n1 is even? " + n1.isEven());
      System.out.println("n1 is odd? " + n1.isOdd());
      System.out.println("n1 is prime? " + n1.isPrime());
      System.out.println("15 is prime? " + MyInteger.isPrime(15));
      char[] chars = {'3','5','3','9'};
      System.out.println(MyInteger.parseInt(chars));
      String s = "9786";
      System.out.println(MyInteger.parseInt(s));
      MyInteger n2 = new MyInteger(24);
      System.out.println("n2 is " + n2.getValue());
      System.out.println("n2 is odd? " + n2.isOdd());
      System.out.println("45 is odd? " + MyInteger.isOdd(45));
      System.out.println("n1 is equal to n2? " + n1.equals(n2));
      System.out.println("n1 is equal to 5? " + n1.equals(5));
   }
}
```

```
----jGRASP exec: java -ea MyIntegerDemo
n1 is 5
n1 is even? false
n1 is odd? true
n1 is prime? true
15 is prime? false
3539
```

```
9786
2 is 24
n2 is odd? false
45 is odd? true
n1 is equal to n2? false
n1 is equal to 5? true

----jGRASP: operation complete.
```

What to submit: MyInteger.java, MyIntegerDemo.java and the outputs generated by your code for the input sets in a text file.

Exercise 4: We discussed the following example of object interactions in the lectures.
We define a Point class and a Circle class. The Circle class has a method that checks if a Point object is enclosed inside the Circle object. Study the code, run it and understand it.

```
//Point class
//Defines a point with coordinates px and py
public class Point
{
   private double px;
   private double py;

   //constructor
   public Point(double px, double py)
   {
      this.px = px;
      this.py = py;
   }

   //get and set methods
   public void setX(double px){this.px = px;}
   public void setY(double py){this.py = py;}
   public double getX(){return px;}
  public double getY(){return py;}
   //toString method
   public String toString()
   {
      return "[px = " + px + ", py = " + py + "]";
   }
}
```

```
//Circle class
//Defines a circle object with center cx, cy and radius r
public class Circle
{
   //instance variables
   private double cx;
   private double cy;
   private double radius;

   //constructor
   public Circle(double cx, double cy, double radius){
      this.cx=cx;
      this.cy=cy;
      this.radius=radius;
   }

   //get and set methods
   public void setCX(double cx){this.cx=cx;}
   public void setCY(double cy){this.cy=cy;}
   public void setRadius(double radius){this.radius=radius;}
   public double getCX(){return cx;}
   public double getCY(){return cy;}
   public double getRadius(){return radius;}
```

```
//toString method
   public String toString()
   {
      return "Circle with center " + cx +"," + cy + " and radius " + radius;
   }

   //enclose method (illustrates object interactions)
   //checks if a point is enclosed within the circle
   //Algorithm: A point px,py is enclosed in a circle if the distance from
   //the point to the center cx,cy of the circle is less than the radius

   public boolean encloses(Point p)
   {
      double d;
      d = Math.sqrt(((p.getX()-cx)*(p.getX()-cx)) + ((p.getY()-cy)*(p.getY()- cy)));

      if (d<radius)
         return true;
      else
         return false;
   }

}
```

```
import java.util.Scanner;
public class PointCircleDemo
{
   public static void main(String[] args)
   {
       Scanner keyboard = new Scanner(System.in);
       System.out.print("Enter the x and y coordinates of the point: " );
       Point p = new Point(keyboard.nextDouble(), keyboard.nextDouble());
       System.out.print("Enter the center coordinates (x,y) of circle and
its radius: ");
       Circle c = new Circle(keyboard.nextDouble(),keyboard.nextDouble(),
keyboard.nextDouble());


       if (c.encloses(p))
          System.out.println(c + " encloses " + p);
        else
          System.out.println(c + " does not enclose " + p);
    }
}
```

a) Add a method to Circle.java that returns true if this Circle object touches another Circle object externally, and false otherwise. For example, the method should return true for the case shown in Figure 1(a).

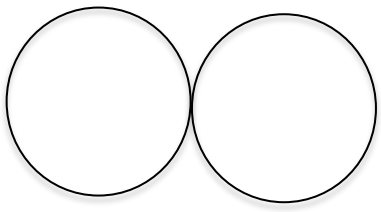

Figure 1(a): The two circles touch each other externally.

b) Add a method to Circle.java that returns true if this Circle object touches another Circle object internally, and false otherwise. For example, the method should return true for the case shown in Figure 1(b).

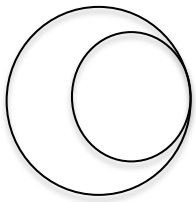

Figure 1(b): The two circles touch each other internally.

Algorithm: Find the distance d between the two centers.
Add the two radii. If d is equal to the sum of the radii, then the two circles touch each other externally.
Subtract the two radii. If d is equal to the difference between the two radii, the two circles touch each other internally.

Test your code in a demo program called CircleDemo.java for at least three different cases.

Sample runs of the program are given below:

```
----jGRASP exec: java -ea CircleDemo
Enter the center coordinates (x,y) of the first circle and its radius: 10 10 10
Enter the center coordinates (x,y) of the second circle and its radius: 30 10 10
Circle with center 10.0,10.0 and radius 10.0 touches Circle with center 30.0,10.0 and
radius 10.0 externally
Circle with center 10.0,10.0 and radius 10.0 does not touch Circle with center
30.0,10.0 and radius 10.0 internally
----jGRASP: operation complete.

----jGRASP exec: java -ea CircleDemo
Enter the center coordinates (x,y) of the first circle and its radius: 10 10 10
Enter the center coordinates (x,y) of the second circle and its radius: 15 10 5
Circle with center 10.0,10.0 and radius 10.0 does not touch Circle with center
15.0,10.0 and radius 5.0 externally
Circle with center 10.0,10.0 and radius 10.0 touches Circle with center 15.0,10.0 and
radius 5.0 internally
----jGRASP: operation complete.

----jGRASP exec: java -ea CircleDemo
Enter the center coordinates (x,y) of the first circle and its radius: 10 10 10
Enter the center coordinates (x,y) of the second circle and its radius: 30 30 5
Circle with center 10.0,10.0 and radius 10.0 does not touch Circle with center
30.0,30.0 and radius 5.0 externally
Circle with center 10.0,10.0 and radius 10.0 does not touch Circle with center
30.0,30.0 and radius 5.0 internally
----jGRASP: operation complete.
```

What to submit: Circle.java, CircleDemo.java and sample outputs in a text file. If you have used Point.java in the Circle class, submit that as well.