German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher
Dr. Wael Abouelsaadat
Dr. Mohammed Abdel Megeed

Introduction to Computer Programming, Spring Term 2017
Practice Assignment 7
Discussion: 29.4.2017 - 4.5.2017

Exercise 7-1 Point
To be discussed in the tutorial

A point in the Cartesian plane can be defined using its X and Y coordinates.

a) Implement a Java class `Point`.

b) Augment your class with a constructor that initializes a `Point` with both its X and Y coordinates in addition to the parameter-less constructor.

c) Augment your class with the following methods:

- `static add(Point p1, Point p2)` returns a new point as a result of the summation of points p1 and p2 x-coordinates and y-coordinates.
- `add(Point p)` add the values of the X and Y co-ordinates of p to the `Point` object on which the method is invoked.
- `static swap(Point p1, Point p2)` swaps the values of the X and Y co-ordinates of both p1 and p2.
- `swap(Point p)` swaps the values of the X and Y co-ordinates with that of the `Point` object on which the method is invoked.
- `toString()` override the implementation of the `toString` method to display the values of the X and Y coordinates of the point.

d) Augment your class with a class variable to keep track of the number of Point objects created.

e) Augment your class with a main method to test your implementation.

Solution:

```
class Point{
        double x;
        double y;
        static int count = 0;

        Point() {
                this(0,0);
        }

        Point(double x, double y) {
                this.x = x;
                this.y = y;
                count++;
        }
```

```java
        public void add(Point p)
        {
                x+=p.x;
                y+=p.y;
        }


        static Point add (Point p1, Point p2) {
                return new Point(p1.x + p2.x, p1.y + p2.y);
        }

        public void swap(Point p) {
                double tempX = p.x ;
                double tempY = p.y;
                p.x = this.x;
                p.y = this.y;
                this.x = tempX;
                this.y = tempY;
        }

        public static void swap(Point p1, Point p2) {
                double tempX = p1.x ;
                double tempY = p1.y;
                p1.x = p2.x;
                p1.y = p2.y;
                p2.x = tempX;
                p2.y = tempY;
        }

        public String toString() {
                return "x␣=␣" + x + "␣y␣=␣" + y;
        }

        public static void main(String []args){
                Point p1 = new Point();
                Point p2 = new Point(30,50);

                p1.add(p2);

                System.out.println(p1);
                Point p3 = add(p1,p2);
                System.out.println(p3);
                p1.swap(p3);
                System.out.println(p1);
                System.out.println(p3);

                swap(p2,p1);
                System.out.println(p1);
                System.out.println(p2);

                System.out.println(count);


        }
}
```

Exercise 7-2 Triangle
To be discussed in the lab

A Triangle can be described using the 3 points that represent the vertices.

a) Using your implementation of Class `Point`, implement a Java class `Triangle`.

b) Augment your implementations with two constructors. The first is the parameter-less constructor (takes no arguments) that sets coordinates of the three points to 0. The overloaded constructor takes the three points.

c) Augment your class with the following methods:

- `copy()` that returns the a new instance of the `Triangle` object with the same values as the object on which the method was invoked.
- `rotate()` that swaps each point with the point next to it.
- `toString()` that returns the 3 points that construct a triangle.

d) Augment your class with a class variable to keep track of the number of Triangle objects created.

e) Augment your class with a main method that constructs at least two `Triangle` objects and tests all methods defined above. You should also print the number of `Triangle` as well as `Point` objects created.

Solution:

```
class Triangle {
        Point x;
        Point y;
        Point z;
        static int count = 0;

        Triangle() {
                x = new Point(0,0);
                y = new Point(0,0);
                z = new Point(0,0);
                count++;
        }

        Triangle(Point x, Point y, Point z) {
                this.x = x;
                this.y = y;
                this.z = z;
                count++;
        }

        Triangle copy() {
                return new Triangle(this.x, this.y, this.z);
        }

        void rotate(){
                Point pTemp = x;
                x = y;
                y = z;
                z = pTemp;
        }

        public String toString() {
                return "Point 1 " + x + "\n" +
                                "Point␣2␣" + y + "\n" +
                                "Point␣3␣" + z ;
        }

        public static void main (String arggs [] ) {
                Point x = new Point(1,2);
                Point y = new Point(2,4);
                Point z = new Point(4,6);
                Triangle t = new Triangle(x,y,z);
                System.out.println(t);
                System.out.println("X␣coordinates␣of␣Point␣x␣" + t.x.x);
                System.out.println("Y␣coordinates␣of␣Point␣x␣" + t.x.y);
                t.rotate();
                System.out.println(t);
                Triangle t2 = t.copy();
                System.out.println("Triangles␣created␣" + count);
                //to call a static variable of a different class use
                //ClassName.StaticVarirableName
                System.out.println("Points␣created␣" + Point.count);

        }
}
```

Exercise 7-3 Student

A student is defined by his/her first name, GPA and whether they are a senior `isSenior` (a student is a senior if they are at the last year of study) or not. Assuming you have a skeleton for class `Student` (available on the course website). Download the file and do the following:

a) The file contains no constructors, Can it run (using the available main method)?

Solution:

The file will compile, Since the Java by default creates the "default constructor" in any java class that does not have a constructor. The "default constructor" initializes all the instance variables to their default values (`int` to 0, `double` to 0.0, `boolean` to `false` and `Object` to `null`).

b) Augment the class with another constructor that initializes the `name`, `GPA` and `isSenior` to specific values. Try compiling the class again.

Solution:

```
Student (String fN, double g, boolean s) {
  firstName = n;
  gpa       = g;
  isSenior  = s;
}
```

The file will not compile with the error :

```
Student.java:13: cannot find symbol
symbol : constructor Student()
location: class Student
Student s1 = new Student();
^
1 error
```

Since Java does not include the default constructor, unless there are no constructors in the java file, once you define a constructor, the Java default constructor will be removed. Thus, to solve the problem, we can define our own parameter-less constructor that initializes the variables to their default values.

```
Student () {
        firstName = null;
        gpa       = 0.0;
        isSenior  = false;
}
```

Exercise 7-4 Pair Of Dice
To be discussed in the tutorial

Design a class `PairofDice` with two instance variables to represent the numbers showing on each dice.

a) Implement a method `roll` that sets the value of each dice to a random number between 1 and 6.

b) Implement a constructor that rolls the dice, so that they initially show some random values.

c) Implement a method `getFirstDice` that returns the number showing on the first dice.

d) Implement a method `getSecondDice` that returns the number showing on the second dice.

e) Implement a method `getTotal` that returns the total showing on the both the dices.

f) Implement a main method that rolls a pair of dice until the dice come up *snake eyes* (with a total value of 2). The method should count and reports the number of rolls.

Solution:

```
public class PairOfDice{

        int dice1;
        int dice2;


        public void roll()
        {
                dice1 = (int) (6*Math.random()) +1;
                dice2 = (int) (6*Math.random()) +1;
        }

        public PairOfDice(){
                roll();
        }

        public int getFirstDice()
        {
                return dice1;

        }
        public int getSecondDice()
        {
                return dice2;

        }

        public int getTotal()
        {
```

```
        return dice1+dice2;

    }


    public static void main(String[] args)
    {
        PairOfDice p = new PairOfDice();
        int counter=1; //already rolled once when constructing the object

        while(p.getTotal()!=2)
        {
            p.roll();
            counter++;

            System.out.println(p.getFirstDice()
                + "␣,␣" + p.getSecondDice());
        }


        System.out.println("count␣is␣:␣" + counter);


    }
}
```

Exercise 7-5 Lego bricks- Final Spring 2014
To be discussed in tutorial

The following classes manage a pile of colored Lego-bricks. Please consider the class `Color` and the class `Brick`.

Further consider the tester class `Bricklayer`, which instantiates some colors and bricks.

```
public class Color {
String name;

public Color (String name) {
    this.name = name;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
}


public class Brick {
static int nextSerial = 0;
int serial;
Color color;

public Brick () {
    this.serial = nextSerial++;
}

public Brick (Color color) {
    this();
    this.color = color;
}

public Color getColor() {
    return color;
}

public void setColor(Color color) {
    this.color = color;
}

public void display() {
    System.out.println("Brick " + serial + " (" + color.getName() + ")");
}
}

public class Bricklayer {

public static void main(String[] args) {
  Color red = new Color ("Red");
  Brick redBrick = new Brick (red);

  Color blue = new Color ("Blue");
  Brick blueBrick = new Brick (blue);

  Brick yellowBrick = new Brick (new Color ("Yellow"));

  Color green = red;
  green.setName("Green");
  Brick greenBrick = new Brick (green);

  Brick orangeBrick = greenBrick;
  orangeBrick.getColor().setName("Orange");

  yellowBrick.setColor(blue);

  Brick blackBrick = new Brick ();

}

public static boolean compare1(Brick a, Brick b) {
    return a == b;
}

public static boolean compare2(Brick a, Brick b) {
  return a.getColor() == b.getColor();
}

public static boolean compare3(Brick a, Brick b) {
  return a.getColor().getName() == b.getColor().getName();
}
}
```

a) Give the exact output once we include the following lines to the `main` method:

```
redBrick.display();
blueBrick.display();
yellowBrick.display();
greenBrick.display();
orangeBrick.display();
```

Solution:

```
Brick 0 (Orange)
Brick 1 (Blue)
Brick 2 (Blue)
Brick 3 (Orange)
Brick 3 (Orange)
```

b) Give the exact output once we include the following lines to the `main` method:

```
 System.out.println("The blocks are "
        + (compare1(orangeBrick, greenBrick) ? "equal" : "distinct"));
 System.out.println("The blocks are "
    + (compare2(orangeBrick, greenBrick) ? "equal" : "distinct"));
 System.out.println("The blocks are "
    + (compare3(orangeBrick, greenBrick) ? "equal" : "distinct"));
```

Solution:

```
The blocks are equal
The blocks are equal
The blocks are equal
```

c) Give the exact output once we include the following lines to the `main` method:

```
System.out.println("The blocks are "
    + (compare1(blueBrick, yellowBrick) ? "equal" : "distinct"));
System.out.println("The blocks are "
    + (compare2(blueBrick, yellowBrick) ? "equal" : "distinct"));
System.out.println("The blocks are "
    + (compare3(blueBrick, yellowBrick) ? "equal" : "distinct"));
```

Solution:

```
The blocks are distinct
The blocks are equal
The blocks are equal
```

d) Give the output once we include the following line to the `main` method:

```
blackBrick.display();
```

Solution:

`NullPointerException` since the object `blackBrick` was created using the default constructor that initializes `blackBrick.color` with `null`.