Exercise 6

Objectives

The aim of this exercise is to learn the concept of inheritance hierarchy. In this exercise you will learn:

- Some functions of Object class
- HAS A Relation
- IS A Relation

A: Equals [1 pt]

Please predict the output of the following program.

```
class Point{
   private int x;
   private int y;
   public Point(int x, int y){
      this.x = x;
      this.y = y;
   }
}

class EqualPointApplication{
   public static void main(String[] args){
      Point p1 = new Point(1, 2);
      Point p2 = new Point(1, 2);

      if ( p1 == p2 ) {
         System.out.println("p1 == p2");
      } else {
         System.out.println("p1 != p2");
      }

      if ( p1.equals(p2) ){
         System.out.println("p1 and p2 is equal");
      } else {
         System.out.println("p1 and p2 is not equal");
      }
   }
}
```

Explain why such an output is obtained (in `Equals.txt`).

Modify the Point class so that the following results are obtained. Of course, do not modify the EqualPointApplication class.

```
p1 != p2
p1 and p2 is equal
```

Submission Files	Types
Equals.txt	Texts in English
Point.java	Java Class

B: HAS-A Relation and IS-A Relation [1 pt]

You should create three Java Classes Shape, Quadrangle, and Point which satisfy the following conditions.

- Quadrangle IS-A Shape.
- Quadrangle HAS-A set of Points.

A Quadrangle object has four Point objects. More specifically, a quadrangle is created/printed by four points in counter-clockwise order.

You should define the Shape class as follows.

```
abstract class Shape{
    public void print(){
        System.out.print(this.getClass().getName() + ": ");
    }
    public abstract void move(int dx, int dy);
}
```

The method *print* is a method to print its type (e.g. "Quadrangle"). The method *move* is an abstract method which is implemented by the sub-class (Quadrangle) to parallel shift the shape by the specified values.

The following is the QuadrangleApplication class, whose main method instantiates a Quadrangle object, moves (parallel shift) it and prints its information.

```
class QuadrangleApplication{
    public static void main(String[] args){
        Quadrangle q =  new Quadrangle(
                                    new Point(0, 0),
                                    new Point(4, 3),
                                    new Point(1, 5),
                                    new Point(0, 2)
                                    );
        q.move(8, 5);
        q.print();
    }
}
```

The output of QuadrangleApplication should be like this:

```
Quadrangle: (8, 5)-(12, 8)-(9, 10)-(8, 7)
```

So, your task is to implement Quadrangle class. Use the following Quadrangle class as a starting point for your design and implementation.

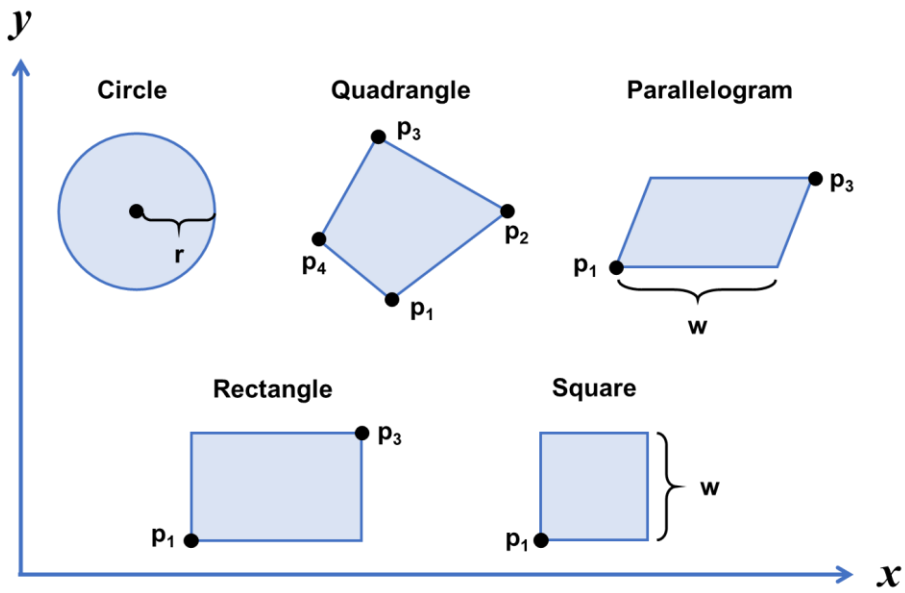
```
class Quadrangle   /* your code */   {
    protected Point[] P;

    public Quadrangle(Point p1, Point p2, Point p3, Point p4){
        /*
          your code
        */
    }

    public void print(){
        /*
          your code
        */
    }

    public void move(int dx, int dy){
         /*
          your code
        */
    }
}
```

You can use the Point class created for Problem A. Note that the Point class created at Problem A must also be extended by adding get methods to obtain the value of the x and y coordinates, respectively, and a move method to parallel shift the point by values specified for the x and y coordinates.

Submission Files	Types
Quadrangle.java	Java Class
Point.java	Java Class
Shape.java	Java Class
QuadrangleApplication.java	Java Class

C: A Hierarchy of Classes [3 pt]

You should create Java codes to implement a hierarchy of classes including Circle, Quadrangle, Parallelogram, Square, and Rectangle shown in the figure below.



The following is the ShapeApplication class, whose main method instantiates each figure, moves (parallel shift) them and prints their information.

```
class ShapeApplication{
    public static void main(String[] args){
        Shape[] shapes = {
            new Circle(new Point(0, 0), 3),
            new Quadrangle(
                        new Point(0, 0),
                        new Point(4, 3),
                        new Point(1, 5),
                        new Point(0, 2)),
            new Parallelogram(new Point(5, 4), new Point(12, 7), 5),
            new Rectangle(new Point(3, 2), new Point(7, 5)),
            new Square(new Point(5, 6), 2)
        };

        for ( int i = 0; i < shapes.length; i++)
            shapes[i].move(8, 5);

        for ( int i = 0; i < shapes.length; i++)
            shapes[i].print();
    }
}
```

The output of ShapeApplication should be like this:

```
Circle: (8, 5) radius = 3
Quadrangle: (8, 5)-(12, 8)-(9, 10)-(8, 7)
Parallelogram: (13, 9)-(18, 9)-(20, 12)-(15, 12)
Rectangle: (11, 7)-(15, 7)-(15, 10)-(11, 10)
Square: (13, 11)-(15, 11)-(15, 13)-(13, 13)
```

Your task is to implement classes for Circle, Quadrangle, Parallelogram, Rectangle and Square considering their hierarchical relations based on the following specification.

Circle	A circle is created/printed by its center (x, y) and radius.
Quadrangle	A quadrangle is created/printed by four points in counter-clockwise order.
Parallelogram	A parallelogram is created by two points on the opposite corner and the width of its base. It is printed by its four points in counter-clockwise order.
Rectangle	A rectangle is created by two points on the opposite corner and printed by its four points in counter-clockwise order.
Square	A square is created by a point and its width and printed by its for points in counter-clockwise order

As shown in the figure, the first point (p_1) of the quadrangle is the point with the smallest y-coordinate. If there is more than one such point, it should be the point with the smallest x-coordinate among them.

Let's design your class hierarchy considering the following issues:

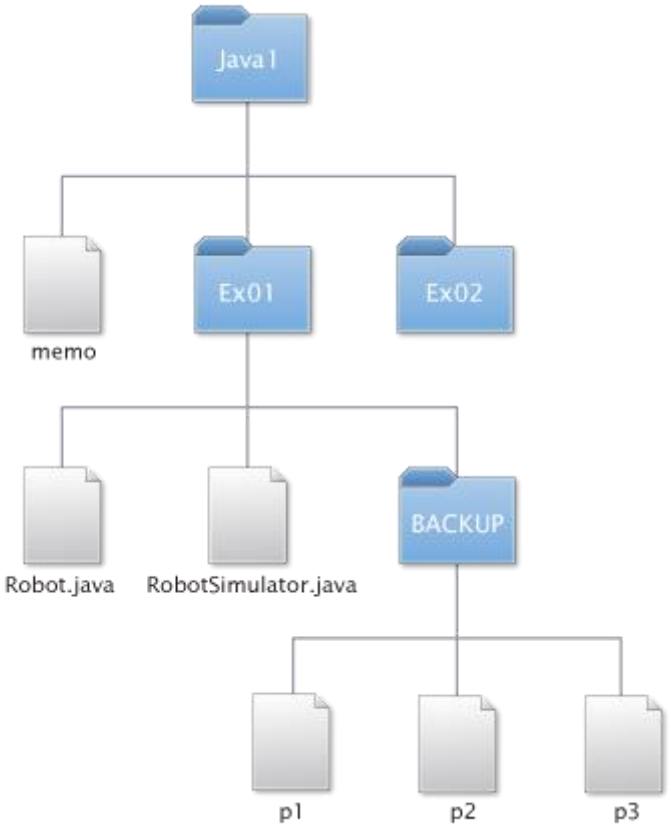
- "IS A" relationship must hold in the class hierarchy.
- "generalize" and "specialize" in the class hierarchy
- Be aware of code reuse in the class hierarchy

Note that you can use the Shape class and Quadrangle class defined for Problem B.

Submission Files	Types
`Point.java`	`Java Class`
`Circle.java`	`Java Class`
`Quadrangle.java`	`Java Class`
`Parallelogram.java`	`Java Class`
`Rectangle.java`	`Java Class`
`Square.java`	`Java Class`
`Shape.java`	`Java Class`
`ShapeApplication.java`	`Java Class`

D: File Management System [3 pt]

Your task is to develop software that simulates file management.



An instance of the class FileManagerApplication shown below creates and manipulates the file (directory) structure shown above. In the constructor, file and directory objects are created to create a hierarchical structure of files. Then, it displays the contents of a specific directory. After making further changes to a particular file and directory, it again displays the contents of the particular directory.

```
public class FileManagerApplication{

    public FileManagerApplication(){
        // create files and directory
        File file1 = new File("memo", 80);
        File file2 = new File("Robot.java", 322);
        File file3 = new File("RobotSimulator.java", 548);
        File file4 = new File("p1", 288);
        File file5 = new File("p2", 331);
        File file6 = new File("p3", 20);
        Directory dir1 = new Directory("Java1");
        Directory dir2 = new Directory("Ex01");
        Directory dir3 = new Directory("Ex02");
        Directory dir4 = new Directory("BACKUP");

        // organize the hierachy
        dir1.add(file1);
        dir1.add(dir2);
        dir1.add(dir3);
        dir2.add(file2);
        dir2.add(file3);
        dir2.add(dir4);
        dir4.add(file4);
        dir4.add(file5);
        dir4.add(file6);

        // display
        dir1.display();
        dir2.display();

        // make changes
        file2.rename("SuperRobot.java");
        dir1.add(new File("score", 100));

        // display again
        dir1.display();
        dir2.display();
    }

    public static void main(String[] args){
        new FileManagerApplication();
    }
}
```

The classes File and Directory are defined as follows.

```
public class File{
    private String name;
    private int size;

    public File(String name, int size){
        this.name = name;
        this.size = size;
    }

    public String getName(){ return name; }

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

    public int getSize(){
        return size;
    }
}
```

```
public class Directory{
    private String name;
    private File[] fileList = new File[100];
    private Directory[] directoryList = new Directory[100];
    private int numberOfFile = 0;
    private int numberOfDirectory = 0;

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

    public String getName(){ return name; }

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

    public int getSize(){
        int total = 0;
        for ( int i = 0; i < numberOfFile; i++ ){
            total += fileList[i].getSize();
        }
        for ( int i = 0; i < numberOfDirectory; i++ ){
            total += directoryList[i].getSize();
        }
        return total;
    }

    public void add(File file){
        fileList[numberOfFile++] = file;
    }

    public void add(Directory directory){
        directoryList[numberOfDirectory++] = directory;
    }

    public void display(){
        for ( int i = 0; i < numberOfFile; i++ ){
            System.out.print(fileList[i].getName() + " ");
        }
        for ( int i = 0; i < numberOfDirectory; i++ ){
            System.out.print(directoryList[i].getName() + " ");
        }
        System.out.println();
        System.out.println(this.getSize() + " bytes");
    }
}
```

The method display of class Directory outputs a list of the names of the files and directories it directly holds and the total capacity of the files it and its subdirectories contain. This capacity is obtained by a recursive process.

The output of the program is as follows. Note that the order of output of file names in a directory is arbitrary.

```
memo Ex01 Ex02
1589 bytes
Robot.java RobotSimulator.java BACKUP
1509 bytes
memo Ex01 Ex02 score
1689 bytes
SuperRobot.java RobotSimulator.java BACKUP
1509 bytes
```

The above program works correctly, but there is room for improvement in its design. Your task is to modify the program based on the following problems and hints. Note that there is no need to change the class FileManagerApplication.

- In class Directory, files and directories are defined as separate classes (objects), so it is necessary to distinguish between files and directories when adding files and directories, getting their sizes, and displaying them.
- In addition to the large amount of code required for such a design, if a new class (e.g., class Shortcut) is added due to additional specifications, dedicated code for it must be added to each process as well.
- These problems can be solved by redesigning the superclass inherited by the class File and the class Directory together as an abstract class.
- If we make this abstract class Entity, we can treat File and Directory objects (and even Shortcut objects) as objects of class Entity in a unified manner.

Submission Files	Types
`Entity.java`	`Java Class`
`File.java`	`Java Class`
`Directory.java`	`Java Class`
`FileManagerApplication.java`	`Java Class`

Summary

In this exercise, we learned some functions of Object class like equals method. Through creating a simple class hierarchy of shapes, we learned how to design classes where “generalize” and “specialize” are involved. We also saw one of useful design patterns (this is called Composite Pattern) through an application related to file management.