CMT205 Object Oriented Development with Java

Lab Exercises Week 4

Section 1: Text and Binary Files

1. Create a text file containing the following data:

```
6
0.5 1.0
1.0 1.5
1.5 2.0
2.0 2.5
2.5 3.0
1.0 3.5
```

The data contains the number of points in a graph and the coordinates of each point. Write a Java program to read this data, store the coordinates in two arrays, named **x** and **y** and, finally, display the coordinates of each point.

2. Records of books in the library have the following fields:
   book_id: int
   title: 100 Unicode characters
   authors: 50 Unicode characters
   year: int

We assume that **book_id** is a unique positive integer number which is more or less sequentially allocated within the library. An example of a record is given as follows:-

30
Java in Two Semesters (Third Edition)
Quentin Charatan & Aaron Kans
2009

Write a Java command-line program **Books.java** that allows maintaining the library using a collection of commands, given in the command line:

- java Books add <book_id> <title> <authors> <year>

Successful execution of this command adds a record to the library. Output appropriate error message if the book with <book_id> already exists, or if the input is not in the correct form. Use double quotes if strings contain any blank space.

- java Books query <book_id>

This command makes a query of the current books in the library. If a book with <book_id> exists, the record information (title, authors and year) is printed; otherwise an error message is displayed.

- java Books delete <book_id>

Successful execution of this command deletes the record with <book_id> from the library. If the record does not exist, an error message is displayed.

Section 2: Inheritance

1. The following abstract class **Shape** is intended to represent some arbitrary shape in the 2D plane:

```
public abstract class Shape
{
    public abstract double getArea();
    public abstract double getCircumference();
    public abstract void move(double disp_x, double disp_y);
    public String toString()
    {
            return "Shape[]";
    }
}
```

Assume the standard Java coordinate system with *x* axis pointing from left to right and *y* axis pointing from top to bottom is used. The meanings of these methods are quite intuitive:

- getArea() returns the area of the shape
- getCircumference() returns the circumference of the shape
- move() changes the position of the shape, by moving along *x* direction by *disp_x*, and moving along *y* direction by *disp_y*
- toString() returns a string representation of the object.

Implement three concrete subclasses, namely **Rectangle, Circle** and **Combined**, providing each of these methods, and an appropriate constructor for each of these classes.

The **Rectangle** class represents a rectangle and its constructor should take four parameters (left, top, width and height). The **Circle** class represents a circle and its constructor should take three parameters (centre_x, centre_y and radius). A **Combined** object is used to represent a shape which is composed by putting together two other shapes. You may assume that the area and circumference of the combined shape are simply the sum of its constituent shapes. The constructor for **Combined** should take two **Shape** objects (**shape1** and **shape2**). **Combined** can be used in a nested manner (i.e. to build **Combined** objects from shapes which themselves are possibly **Combined** objects), allowing us to represent overall combined objects containing multiple rectangles and circles.

Remember to override **toString()**, so that the appropriate output like **Rectangle[(x1, y1) – (x2, y2)] or Circle[(x, y): radius]** is produced. For **Combined**, output **Combined [shape1, shape2]** where **shape1** and **shape2** are the corresponding information described above for the constituent shapes.

`ShapeTest.java` is provided for your testing. Put it in the same folder as your program.

Hint: You may test your program with partial functionalities implemented. However, all the abstract methods defined in the **Shape** class should be provided with a concrete implementation, even if with a trivial implementation that does not do anything.