TCSS143
Fundamentals of Object-Oriented Programming-Theory and Application
Programming Assignment 7

The purpose of this programming project is to demonstrate the functionality of Linked Lists while reviewing such things as Abstract classes, inheritance, compareTo, file I/O, and various uses of the Scanner class.

REQUIREMENTS
You will submit a single file "**Assignment7.zip**" through the Programming Assignment 7 Submission link on Canvas. This zipped file will contain 5 classes (files) that make up your solution to this assignment. Also, you will create your own test input file "In7.txt" to use for your own testing. However, do **NOT** include in7.txt in the zipped file for submission. I will use my own for testing. **Make sure these files are not zipped into a folder**.

DETAILS
In this assignment, you will create 3 shape classes (Circle, Rectangle, Triangle) that all inherit from a single **abstract** class called Shape. You are also responsible for creating the driver class "Assignment7.java" (program that tests your classes and described on page 3) which does the following:

- ✓ reads input data from a file
- ✓ instantiates various objects of the three shapes based on the input data
- ✓ stores each in a LinkedList
- ✓ outputs this list to an output file
- ✓ sorts a copy of this LinkedList of objects
- ✓ outputs the sorted version of the list to the output file
- ✓ outputs the original list to the output file

This driver program also needs to ignore errors in the input file that breach the specified input format as described in the Assignment7.java details (described on page 3).

1. Shape.java

This is an abstract class that has no fields, 1 abstract method, and one complete method:

```
public abstract class Shape implements Comparable<Shape> {
        public abstract double calculateArea();    // This is the abstract method
        public int compareTo(final Shape theOther) {
        // You need to complete this method
        }
}
```

2. Circle.java

Details of each of these shape classes (circle, rectangle, triangle) are fairly straight forward based on the method names.

Be sure to use the invariant that throws an IllegalArgumentException when a method argument that is <= 0 is used to set the radius and supply an appropriate error message in the parameter list (discussed in class).

```
public class Circle extends Shape {
    Fields: myRadius: this should be double
    Methods:
    public Circle (final double theRadius){ }
    public void setRadius(final double theRadius) { }
    public double calculateArea( ) { }
    public String toString( ) { }

}
```

toString should **only** return a String that includes the name of the class, radius, and the area, e.g.

output.out.println(circle1); might produce: `Circle [Radius: 4.40] Area: 60.82`

3. Rectangle.java

Be sure to use the invariant that throws an IllegalArgumentException when method arguments that are <= 0 are used to set the length and width fields. Supply an appropriate error message in the parameter list.

public class Rectangle extends Shape {

Fields: myLength and myWidth: both should be double
Methods:
public Rectangle (final double theLength, final double theWidth) { }
public void setLength(final double theLength) { }
public void setWidth(final double theWidth) { }
public double calculateArea( ) { }
public String toString( ) { }

}

toString should **only** return a String that includes the name of the class, length, width, and area, e.g.

output.out.println(rect1); might produce: `Rectangle [Length: 2.50, Width: 3.00] Area: 7.50`

4. Triangle.java

Be sure to use the invariant that throws an IllegalArgumentException when a method argument is used to set any of the sides of the triangle to values that are <= 0 **AND** if the longest side is >= to the sum of the remaining sides. Supply an appropriate error message in the parameter list.

public class Triangle extends Shape {
Fields: mySideA, mySideB, mySideC: all should be double
Methods:
public Triangle (final double theSideA, final double theSideB, final double theSideC) { }
public void setSideA(final double theSideA) { }
public void setSideB(final double theSideB) { }
public void setSideC(final double theSideC) { }
public double calculateArea( ) { }
public String toString( ) { }

}

toString should **only** return a String that includes the name of the class, the sides, and the area, e.g.

output.out.println(tri1); might produce: Triangle [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75

If you feel the need, please implement any other methods you think the above classes may need.

5. Assignment7.java

This is the test driver class that will include main. This program MUST read a file named in7.txt and generate an output file named out7.txt. The in7.txt file must be created by you based on formatting described shortly. In7.txt and out7.txt are **NOT** to be included in the zipped file.

The input file "In7.txt" will contain multiple lines of input. The input is mostly valid input but, there may be some invalid lines interspersed throughout the file. Valid input is as follows:

Single value: input for the radius of a circle

Two values separated by a space: input for the two sides of a rectangle, 1st = length, 2nd = width

Three values separated by a space: input for the three sides of a triangle in SideA, SideB, SideC order.

Lines containing anything else (could be anything above plus other data) or nothing at all are considered invalid and should simply be ignored by your program. However, a line such as: 3.2 -5.1 should throw an IllegalArgumentException within the Rectangle class due to the negative value.

Again, valid lines will **only** contain 1, 2, or 3 values, for circles, rectangles, or triangles respectively. Lines with anything else are invalid.

Decompose main by calling methods that input data into the List and output the List. Be sure to use a try/catch block inside the input of data method. The try section should call an appropriate class to instantiate an object (Circle, Rectangle, Triangle) passing the appropriate data for the radius, or sides, respectively. If the constructor of the class discovers inappropriate data, it should throw a new IllegalArgumentException (described above in each shape class). The exception, in turn, will be caught in the catch section of the try/catch block and will print to the console an appropriate error message when an exception is thrown. The program should not terminate but instead, continue to the next line of input.

Of course, if no exception occurs, execution will simply bypass the catch block and continue as it should for valid input data.

As you input the data, you should create objects of the appropriate type (mentioned in the above paragraph) and then insert them into a List of Shape objects using a LinkedList.

The List should be declared as: List<Shape> myList = new LinkedList<Shape>( );

To further demonstrate the power and flexibility of inheritance, polymorphism, and abstract classes, instantiate your "copy" List as an ArrayList, have all methods that receive a List declare the parameter as List<Shape>. Because your LinkedList and an ArrayList both implement List, either type can be sent to your output method and used without changing any code.

After your Linked List has been filled, you can instantiate the copy list as such:

List<Shape> copyList = new ArrayList<Shape>(myList);

Passing the original list to the constructor saves the trouble of copying each element within a loop after copyList has been created.

Aside from the input/List creation process and testing the functionality of all these objects, your program should:

- Iterate through myList to display all the shapes and their area (in the output method)
- Copy myList to a new list named copyList
- Sort copyList in ascending order by each shapes area (using Collections Class static sort method)
- Display copyList in the sorted order (in the output method)
- Display myList which should be in its original order (in the output method)

All valid output (the above mentioned) should be sent to the output file out7.txt.

Exceptions thrown should report their output to the console.

Sample I/O, Next Page - - - >

Sample I/O may appear as follows:

Suppose In7.txt contains:

```
2.5
2.5 3
8.1 3.0 5.0
2.5 3 4
4.4
tuesday
-7
3 three
3 -9
3 5
```

During execution the above input file will first produce the following (error) output to the console:

```
----jGRASP exec: java Assignment7

java.lang.IllegalArgumentException: ERROR! Not a Triangle.  Longest side too long.
java.lang.IllegalArgumentException: ERROR! Negative or 0 value can't be applied to a circle radius.
java.lang.IllegalArgumentException: ERROR! Negative or 0 value(s) can't be applied to a rectangle.

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

When the program finishes, an output file ("Out7.txt") should have been created with the following contents:

Original List[unsorted]:
Circle [Radius: 2.50] Area: 19.63
Rectangle [Length: 2.50, Width: 3.00] Area: 7.50
Triangle [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75
Circle [Radius: 4.40] Area: 60.82
Rectangle [Length: 3.00, Width: 5.00] Area: 15.00

Copied List[sorted]:
Triangle [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75
Rectangle [Length: 2.50, Width: 3.00] Area: 7.50
Rectangle [Length: 3.00, Width: 5.00] Area: 15.00
Circle [Radius: 2.50] Area: 19.63
Circle [Radius: 4.40] Area: 60.82

Original List[unsorted]:
Circle [Radius: 2.50] Area: 19.63
Rectangle [Length: 2.50, Width: 3.00] Area: 7.50
Triangle [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75
Circle [Radius: 4.40] Area: 60.82
Rectangle [Length: 3.00, Width: 5.00] Area: 15.00

Remember, only zip the following files together: Shape.java, Circle.java, Rectangle.java, Triangle.java, and Assignment7.java into a zip file named Assignment7.zip and submit it to Canvas through the submission link.