# TCSS143
# Fundamentals of Object-Oriented Programming-Theory and Application
# Programming Assignment 5

The purpose of this programming project is to demonstrate the functionality of Linked Lists while reviewing such things as interface, abstract classes, inheritance, defensive copying, compareTo, file I/O, final methods, static non-final fields, and various uses of the Scanner class.

**REQUIREMENTS**
You will submit a single file "**Assignment5.zip**" through the Programming Assignment 5 Submission link on Canvas. This zipped file will contain 6 classes (files) that make up your solution to this assignment. Also, you will create your own test input file "in5.txt" to use for your own testing. Do **NOT** include in5.txt in the zipped file.

**DETAILS**
You will create 3 shape classes (Circle, Rectangle, Triangle) that all inherit from a single **abstract** class called AbstractShape which implements Shape (also created by you). You are also responsible for creating the driver class "Assignment5.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 Assignment5.java details (see page 3).

# 1. Shape.java

This is an interface that has 2 abstract methods, and passes the responsibility of implementing the compareTo method to the class that implements Shape (you may note, normally Comparable is "implemented" by a class. However, an interface cannot implement because interfaces can only contain abstract methods. That said, an interface can only extend other interfaces and the responsibility of actually "implementing" the abstract method(s) of the super class interface is passed on to the sub-classes):

```
public interface Shape extends Comparable<Shape> {
        public double calculateArea();      // This abstract method is implemented at the concrete level.
        public Shape copyShape();           // also implemented at the concrete level.
        public String getName();             // implemented in AbstractShape
}
```

# 2. AbstractShape.java

**public abstract class AbstractShape implements Shape**
This class should contain an instance field to store the name of each object. The constructor which sets this field should receive the name and a number to be concatenated to the name and then stored in the name field. Recall, when the super class has a parameterized constructor, the sub-classes will need to call it AND the sub-classes will need to also provide a constructor without parameters.

This abstract class will implement the compareTo method passed on from the Shape interface and will pass on the responsibility of implementing calculateArea to the extending sub-classes (compareTo will use the calculateArea method when comparing 2 Shape objects). Along with compareTo, one more concrete method should be included. The following will be used by the sub-classes' toString method:

**public String getName()** // Simply returns the name field data

# 3. 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 & make sure to decrement myID field before throwing the exception).

**public class Circle extends AbstractShape**

**Fields:** myRadius: this should be double
**myID:** this should be a private "static" int field shared by all Circle objects

**Methods:**

```
public Circle ( )  // calls this(1.0);
public Circle (final double theRadius)  // has to call super passing "Circle" and myID incremented.
public void setRadius(final double theRadius)
public double calculateArea( )
public final Shape copyShape()     //  Presented here as an example for the remaining concrete
   Circle newC = new Circle();      //  classes.  This is a defensive copy that returns a reference
   newC.myRadius = myRadius;        //  to a new Circle object.
   return newC;
}
public String toString( )
```

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

output.out.println(aCircle); might produce: `Circle5 [Radius: 4.40] Area: 60.82`

# 4. 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 (make sure to decrement myID field before throwing the exception).

**public class Rectangle extends AbstractShape**

**Fields:** myLength and myWidth: both should be double
**myID:** this should be a private "static" int field shared by all Rectangle objects

**Methods:**

```
public Rectangle( )  // calls this(1.0, 1.0);
public Rectangle (final double theLength, final double theWidth ) // calls super with "Rectangle" and
                                                                  // myID incremented.
public void setLength(final double theLength)
public void setWidth(final double theWidth)
public double calculateArea( )
public final Shape copyShape()     //  Returns a reference to a new Rectangle with the same field
                                   //  values as the implied parameter (defensive copy).
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(aRect); might produce: `Rectangle12 [Length: 2.50, Width: 3.00] Area: 7.50`

# 5. 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 (make sure to decrement myID field before throwing the exception).

**public class Triangle extends AbstractShape** // Continued next page…

**Fields:** mySideA, mySideB, mySideC: all should be double
**myID:** this should be a private "static" int field shared by all Triangle objects
**Methods:**
**public Triangle( )** // calls this(1.0, 1.0, 1.0);
// calls super with "Triangle" and myID incremented
**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 final Shape copyShape()** // Returns a reference to a new Triangle with the same field
// values as the implied parameter (defensive copy).
**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(aTri); might produce: Triangle1 [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75

Make sure all methods of the above 3 classes are named exactly as specified as I will use my own driver.

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

# 6. Assignment5.java

This is the test driver class that will include main. This program MUST read a file named in5.txt and generate an output file named out5.txt. The in5.txt file must be created by you based on formatting described shortly. in5.txt and out5.txt are **NOT** to be included in the zipped file.

The input file "in5.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 (program continues to the next line). 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.

Your method for input should receive an already instantiated Scanner to the input file and the already instantiated LinkedList of Shape (this will be filled inside the method and return, via the parameter, to main). This method should also **return** a List of Shape (as an ArrayList) which contains all the values of the LinkedList described above (this should be used as the List to be sorted).

The List passed to the input method should be declared as:

```
List<Shape> myList = new LinkedList<Shape>( ); // instantiate before sending it to the input method.
```

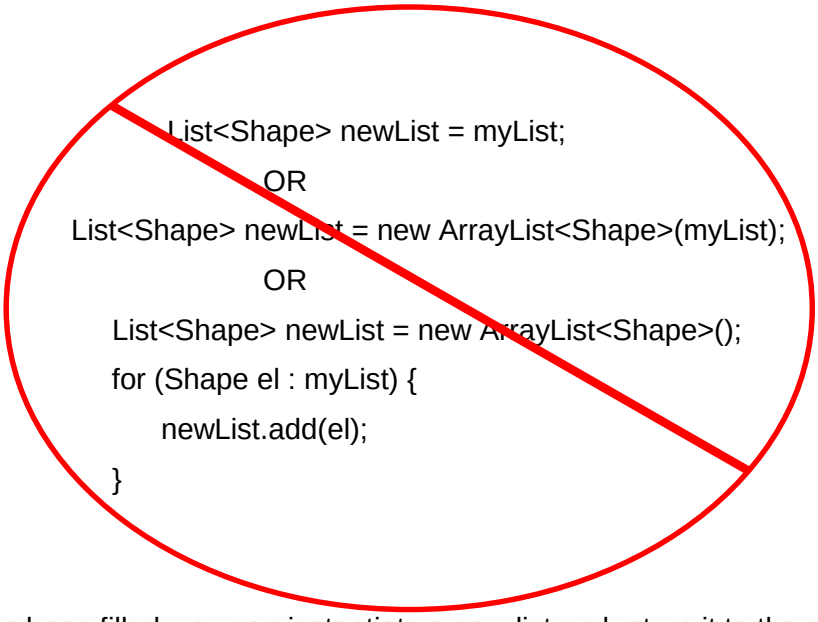
All calls to the input method should be (name the method anything you want):

```
List<Shape> copyList = getOriginalList(input, myList);  // Remember the returned list is an ArrayList.
```

Sort the copyList and print out the myList, copyList, and then myList again by calling your output method 3 times.

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.

As a defensive copy of your original list you **DO NOT** want to do any of the following or cloning:




After your Linked List has been filled, you can instantiate a new list and return it to the copyList as such:

```
List<Shape> newList = new ArrayList<Shape>();
for (Shape element : myList) {
   Shape s = element.copyShape();
   newList.add(s);
}
return newList;
```

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

- Return a copied ArrayList of the original input LinkedList to a new list in main named copyList
- Output the original list to display all the shapes and their area (pass it to the output method)
- Sort copyList in ascending order by the shapes' area (using Collections Class static sort method)
- Display copyList in the sorted order (pass it to the output method)
- Output the original list to display the original order (pass it to the output method)

All valid output (the above mentioned) should be sent to the output file out5.txt.

**Exceptions thrown should report their output to the console.**

You will submit a single file "**Assignment5.zip**" through the Programming Assignment 5 Submission link on Canvas. This zipped file will contain 6 classes (files) that make up your solution to this assignment. Also, you will create your own test input file "in5.txt" to use for your own testing. However, do **NOT** include in5.txt in the zipped file for submission. I will use my own for testing. **Make sure in5.txt is not zipped into a folder**.

## Sample I/O, Next Page - - - >

**Sample I/O may appear as follows-**Suppose in5.txt contains:

```
4.4
2.5 3
8.1 3.0 5.0
2.5 3 4
2.5
tuesday
-7
1.0
3 three
3 -9
3 5

1.0
```

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

```
----jGRASP exec: java Assignment5

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 ("out5.txt") should have been created with the following contents:

```
Original List[unsorted]:
Circle1 [Radius: 4.40] Area: 60.82
Rectangle1 [Length: 2.50, Width: 3.00] Area: 7.50
Triangle1 [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75
Circle2 [Radius: 2.50] Area: 19.63
Circle3 [Radius: 1.00] Area: 3.14
Rectangle2 [Length: 3.00, Width: 5.00] Area: 15.00
Circle4 [Radius: 1.00] Area: 3.14

Copied List[sorted]:
Circle7 [Radius: 1.00] Area: 3.14
Circle8 [Radius: 1.00] Area: 3.14
Triangle2 [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75
Rectangle3 [Length: 2.50, Width: 3.00] Area: 7.50
Rectangle4 [Length: 3.00, Width: 5.00] Area: 15.00
Circle6 [Radius: 2.50] Area: 19.63
Circle5 [Radius: 4.40] Area: 60.82

Original List[unsorted]:
Circle1 [Radius: 4.40] Area: 60.82
Rectangle1 [Length: 2.50, Width: 3.00] Area: 7.50
Triangle1 [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75
Circle2 [Radius: 2.50] Area: 19.63
Circle3 [Radius: 1.00] Area: 3.14
Rectangle2 [Length: 3.00, Width: 5.00] Area: 15.00
Circle4 [Radius: 1.00] Area: 3.14
```

**Remember, only zip the following files together: Shape.java, AbstractShape.java, Circle.java, Rectangle.java, Triangle.java, and Assignment5.java into a zip file named Assignment5.zip (no folders!) and submit it to Canvas through the submission link.**
**Extra Credit!!!!  See next page**

**For 2 extra credit points, sort the copyList descending (largest to smallest) by values.**
**For 6 extra credit points, first sort the copyList ascending by Class (not the name in the field but the actual name of the class, Circle, Rectangle, Triangle) AND for each class, sort the area values descending. (NOTE: these are 2 separate options, they cannot be combined for more points, i.e the maximum extra credit here is 6 points).**

Based on the same input as previous sample I/O both extra credits would produce the same console output:

```
----jGRASP exec: java Assignment5Test

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 myRadius.
java.lang.IllegalArgumentException: ERROR! Negative or 0 value(s) can't be applied to a rectangle.

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

With out5.txt containing the following for 2 extra credit:

```
Copied List [sorted-descending area]:
Circle5 [Radius: 4.40] Area: 60.82
Circle6 [Radius: 2.50] Area: 19.63
Rectangle4 [Length: 3.00, Width: 5.00] Area: 15.00
Rectangle3 [Length: 2.50, Width: 3.00] Area: 7.50
Triangle2 [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75
Circle7 [Radius: 1.00] Area: 3.14
Circle8 [Radius: 1.00] Area: 3.14
```

And out5.txt containing the following for 6 extra credit:

```
Copied List [sorted-Ascending Class first, descending area second]:
Circle5 [Radius: 4.40] Area: 60.82
Circle6 [Radius: 2.50] Area: 19.63
Circle7 [Radius: 1.00] Area: 3.14
Circle8 [Radius: 1.00] Area: 3.14
Rectangle4 [Length: 3.00, Width: 5.00] Area: 15.00
Rectangle3 [Length: 2.50, Width: 3.00] Area: 7.50
Triangle2 [SideA: 2.50, SideB: 3.00, SideC: 4.00] Area: 3.75
```

NOTE: The copied list created new instances of each object and performed a defensive copy of all fields. This is why the names are different than the original. However, as mentioned in class, do not depend on each object's name field for sorting. Use the Class name (Circle, Rectangle, Triangle) instead. In the example above it is just coincidence that the copied Circles appear in 5, 6, 7, 8 order. This is due to the area values but, the two Rectangles are listed as 4, 3. Again, based on the area.

Keep in mind, different input will be used for grading.