COSC1337 Programming Fundamentals II /ITSE2357 Advanced Object-Oriented Programming

The submission opens two days before the due day, and closes right at the due time on due day. One submission is allowed per project. It is students' responsibilities to submit timely and correctly in order to get credits. Students MUST start each project when it is first discussed in class in order to complete it on time. All projects are individual projects. Project due days may be changed to better align with lectures.

If one of the following is true, not credits will be given:

- The project is late.
- The algorithm or class design is missing.
- The project has syntax errors.
- There are no comments (Javadoc format required) at all in your code.
- Wrong files are submitted.
- **A project is a copy and modification of another student's project. (Both will receive 0.)**

Files to be submitted through Canvas:

- The UML design (UML class diagram)
- All source codes and driver programs
- All supporting files if any

Software Development Project Rubric: (Design is worth 20%; the rest is worth 80 %.)
Analysis: There will be no credit if the software doesn't meet customer specification at all.

- Does the software meet the exact customer specification?
- Does the software read the exact input data and display the exact output data as they are shown in sample runs?
- Does each class include all corresponding data and functionalities?

Design: There will be no credit if the design is missing.

- Is the design (a UML class diagram) an efficient solution?
- Is the design created correctly?

Code: There will be no credit if there are syntactic errors.

- Are there errors in the software?
- Are code conventions and name conventions followed?
- Does the software use the minimum computer resource (computer memory and processing time)?
- Is the software reusable?

Debug:

- Are there bugs in the software?

Documentation: There will be no credit if comments are not included.

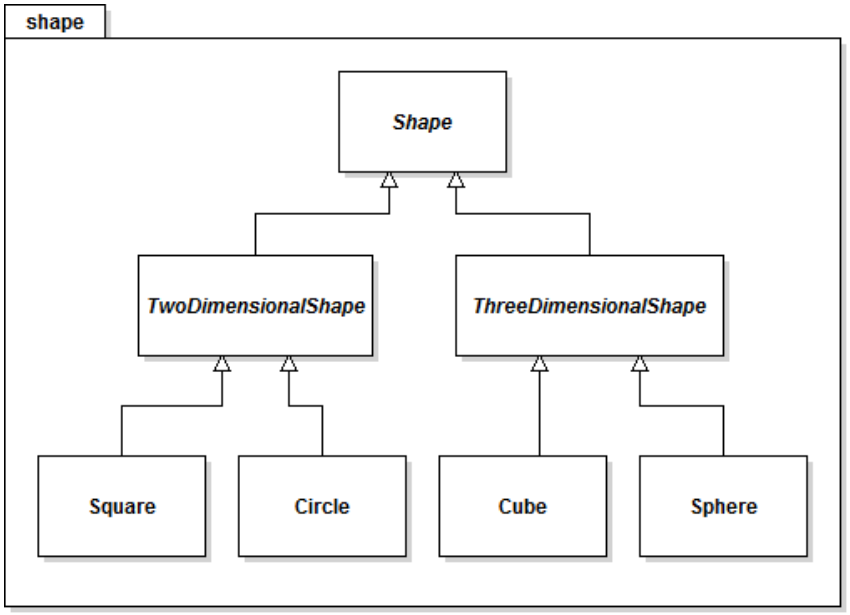
- Are there enough comments included in the software?
- Class comments must be included before a class header.
- Method comments must be included before a method header.
- More comments must be included inside each method.
- All comments must be written in Javadoc format.

Project 4 Inheritance - Shape Hierarchy

A shape is the form of an object - how it is laid out in space. A shape can be two dimensional (Circles, squares) or three dimensional (Spheres, cubes). The two dimensions are commonly called length and width. Both directions lie in the same plane. The three dimensions can be labeled by a combination of three chosen from the terms length, width, height, depth, and breadth. Any three directions can be chosen, provided that they do not all lie in the same plane.

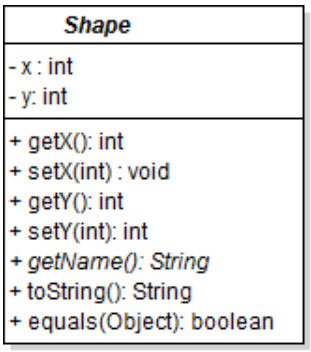
A shape has a bounding area. The upper-left corner (x, y) of the bounding area of a two-dimensional shape or the center point (x, y) of the bounding area of a three-dimensional shape can be used specify the location of this shape.

The following is a shape design hierarchy.



Implement this shape hierarchy with the exact detailed design below. This means no extra instance variables and/or methods are allowed in any classes. All instance variables must be private members.

Shape

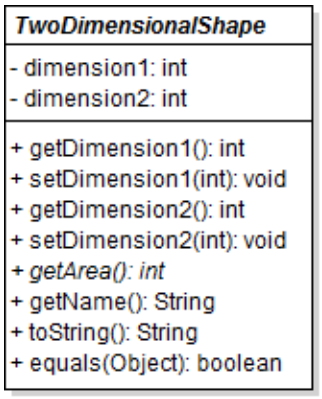


Shape is a generic concept, and should be implemented as an abstract class. A shape has an x coordinate and a y coordinate. Method *getName* should be an abstract method that is completed in subclasses. Each sub *getName* should return a name such as square, circle, cube, or sphere.

This super constructor is used to set up parent part in the subclasses.

```
public Shape(int x, int y ){
      //Set up x coordinate
      //Set up y coordinate
}
```

TwoDimensionalShape

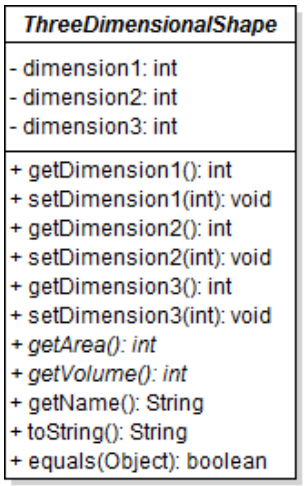


A two-dimensional shape is a generic shape with two dimensions. Each subclass of *TwoDimensionalShape* should contain method *getArea* to calculate the area of the two-dimensional shape. *TwoDimensionalShape* should be implemented as an abstract class – a subclass of *Shape*. Method getArea() is an abstract method.

This super constructor is used to set up parent part in the subclasses.

```
public TwoDimensionalShape( int x, int y,
                            int dimension1,int dimension2){
      //Use super constructor to set up parent part
      //Set up child part
}
```

ThreeDimensionalShape:



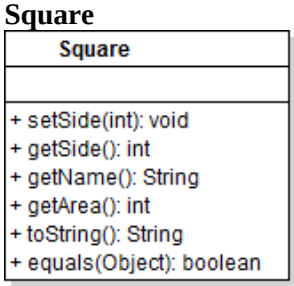
A three-dimensional shape is a generic shape with three dimensions. Each subclass of *ThreeDimensionalShape* should contain methods *getArea* and *getVolume* to calculate the surface area and volume, respectively, of the three-dimensional shape. *ThreeDimensionalShape* should be implemented as an abstract class. Methods *getArea* and *getVolume* are abstract methods.

This super constructor is used to set up parent part in the subclasses.

```
public ThreeDimensionalShape( int x, int y,
                int dimension1, int dimension2, int dimension3 ) {

     //Use super constructor to set up parent part
     //Set up child part
}
```

Square

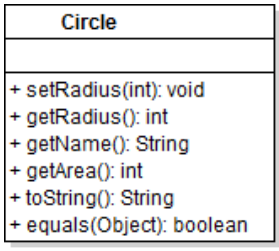


A square is a specific two-dimensional shape. Square should be implemented as a subclass of TwoDimensionalShape, a concrete class. A square has equal sides in both dimensions. This is how to create a square:

```
public Square( int x, int y, int side ){
   /* use super constructor to set up x, y, and both dimensions
      using side */
}
```

Method *setSide* sets each dimension to be `side` via dimension setters. Method *getSide* returns one of the three dimensions via one dimension getter.

Circle

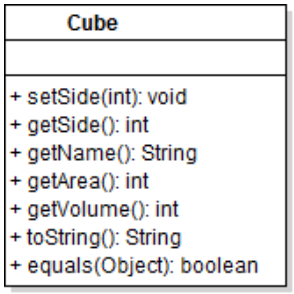


A circle is a specific two-dimensional shape. Circle should be implemented as a subclass of *TwoDimensionalShape*, a concrete class. A circle has equal radius in both dimensions. This is how to create a circle:

```
public Circle( int x, int y, int radius ){
   /* use super constructor to set up x, y, and both dimensions
      using radius */
}
```

Method *setRadius* sets each dimension to be `radius` via dimension setters. Method *getRadius* returns one of the three dimensions via one dimension getter.

Cube

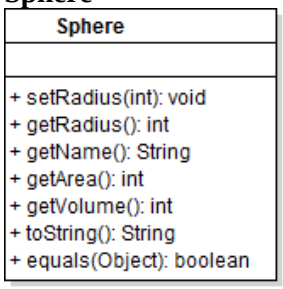


A cube is a specific three-dimensional shape. Cube should be implemented as a subclass of *ThreeDimensionalShape*, a concrete class. A cube has equal sides in all dimensions. This is how to create a cube:

```
public Cube( int x, int y, int side ){
   /* use super constructor to set up x, y, and three dimensions
      using side */
}
```

Method setSide sets each dimension to be `side` via dimension setters. Method getSide returns one of the three dimensions via one dimension getter.

Sphere



A sphere is a specific three-dimensional shape. Sphere should be implemented as a subclass of *ThreeDimensionalShape*, a concrete class. A sphere has equal radius in all three dimensions. This is how to create a sphere:

```
public Sphere( int x, int y, int radius ){
   //use super constructor to set up x, y, and three dimensions
   //using radius
}
```

Method *setRadius* sets each dimension to be `radius` via dimension setters. Method *getRadius* returns one of the three dimensions via one dimension getter.

Create a driver program that uses an array list of *Shape* references to objects of each concrete class in the hierarchy. The program should print a text description of the object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a *TwoDimensionalShape* or a *ThreeDimensionalShape*. If a shape is a *TwoDimensionalShape*, display its area. If a shape is a *ThreeDimensionalShape*, display its area and volume.