# The University of Hong Kong

# Department of Computer Science

## COMP2396 Object-oriented Programming and Java

*Assignment 1*

**Deadline: 11:55pm, 14th Feb, 2019.**

---

## *Overview*

This assignment tests your understanding of classes and objects, and their implementations in Java. You are required to implement 4 classes, namely Shape, Square, Triangle and Circle. The Square class, Triangle class and Circle class are used to model squares, triangles, and circles, respectively. They are subclasses of the Shape class which provides an abstraction of general shapes. For each of the 4 classes mentioned above, you are required to design and implement a tester class to test the correctness of your implementation. You are also required to write Javadoc for all public classes and their public class members.

## *Specifications*

### The Shape class

The Shape class is used to model general shapes. It has instance variables for storing color, fill-type, orientation, screen coordinates of the center, and local coordinates of the vertices of a shape. It has methods for getting the screen coordinates of the vertices of a shape. Below is a detailed description for the Shape class.

Specification of the Shape class:

*instance variables*

`Color color` – a Color object specifying the color of the shape. To use the Color class, place "`import java.awt.Color;`" at the top of the source file of your Shape class. Please refer to http://docs.oracle.com/javase/8/docs/api/java/awt/Color.html for details of the Color class.

`boolean filled` – a boolean value specifying whether the shape is filled or not filled.

`double theta` – a double value specifying the orientation (in radians) of the shape in the screen coordinate system (see Figure 1).

`double xc` – a double value specifying the *x*-coordinate of the center of the shape in the screen coordinate system.

`double yc` – a double value specifying the *y*-coordinate of the center of the shape in the screen coordinate system.

`double[] xLocal` – an array of double values specifying the *x*-coordinates of the vertices (in counter clock-wise order) of the shape in its local coordinate system.

`double[] yLocal` – an array of double values specifying the *y*-coordinates of the vertices (in counter clock-wise order) of the shape in its local coordinate system.

*methods:*

`void setVertices(double d)` – a method for setting the local coordinates of the vertices of a shape. This is a dummy method and is supposed to be overridden in the subclasses.

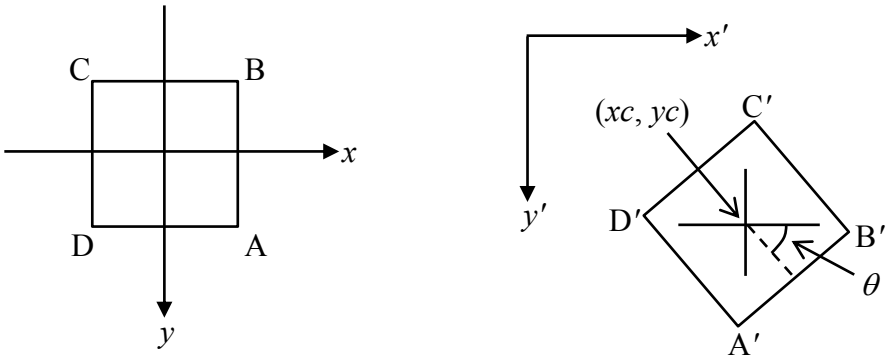
`void translate(double dx, double dy)` – a method for translating the center of the shape by *dx* and *dy*, respectively, along the *x* and *y* directions of the screen coordinate system (i.e., `dx` and `dy` should be added to `xc` and `yc` respectively).

`void rotate(double dt)` – a method for rotating the shape about its center by an angle of *dt* (in radians) (i.e., `dt` should be added to `theta`).

`int[] getX()` – a method for retrieving the *x*-coordinates of the vertices (in counter clock-wise order) of the shape in the screen coordinate system (rounded to nearest integers).

`int[] getY()` – a method for retrieving the *y*-coordinates of the vertices (in counter clock-wise order) of the shape in the screen coordinate system (rounded to nearest integers).

* *You should make all the instance variables and methods public by using the* `public` *access modifiers.*




local coordinate system screen coordinate system

**Figure 1. Relationship between local and screen coordinates of the vertices of a shape.**

The screen coordinates of the vertices can be computed from the local coordinates of the vertices based on the orientation and center of the shape using the following formula:

$$x' = x\cos\theta - y\sin\theta + xc$$
$$y' = x\sin\theta + y\cos\theta + yc$$

where (*x*, *y*) and (*x'*, *y'*) denote the local and screen coordinates of a vertex, respectively, $\theta$ the orientation (in radians) of the shape, and (*xc*, *yc*) the screen coordinates of the center of the shape.

* *You may use* `Math.sin()` *and* `Math.cos()` *to compute sine and cosine functions, and* `Math.round()` *to round a number to its nearest integer. Please refer to https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html for details of the Math class.*

**The Square class**

The Square class is used to model squares. It is a subclass of the Shape class and it inherits all the instance variables and methods of the Shape class. The Square class overrides the `setVertices()` method for setting the local coordinates of the 4 vertices of a standard square. Below is a detailed description for the Square class.

Specification of the Square class:

*instance variables inherited from Shape:*

`color, filled, theta, xc, yc, xLocal, yLocal`

*methods inherited from Shape:*

`translate, rotate, getX, getY`

*overriding method:*

`void setVertices(double d)` – a method for setting the local coordinates of the 4 vertices of a standard square. Here, a standard square has its center located at (0, 0) and its sides being parallel to the *x*- and *y*-axes of its local coordinate system. The parameter *d* specifies half-the-length of a side of the square.

* *You should make all the instance variables and methods public by using the* `public` *access modifiers.*

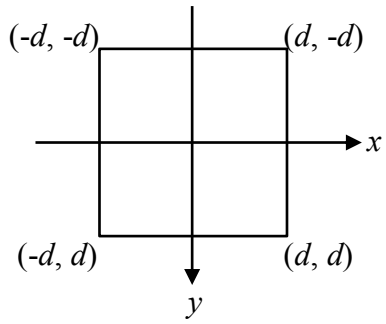



**Figure 2. The local coordinates of the 4 vertices of a standard square with a side of 2*d*.**

For a standard square with a side of 2*d*, the local coordinates of its 4 vertices in counter clockwise order starting from the lower right corner are (*d*, *d*), (*d*, -*d*), (-*d*, -*d*) and (-*d*, *d*), respectively.

**The Triangle class**

The Triangle class is used to model triangles. Like the Square class, it is a subclass of the Shape class and it inherits all the instance variables and methods of the Shape class. The Triangle class overrides the `setVertices()` method for setting the local coordinates of the 3 vertices of a standard triangle. Below is a detailed description for the Triangle class.

Specification of the Triangle class:

*instance variables inherited from Shape:*

`color, filled, theta, xc, yc, xLocal, yLocal`

*methods inherited from Shape:*

`translate, rotate, getX, getY`

*overriding method:*

`void setVertices(double d)` – a method for setting the local coordinates of the 3 vertices of a standard triangle. Here, a standard triangle is an equilateral triangle having its center located at (0, 0) and one of its vertex on the positive *x*-axis of its local coordinate system. The parameter *d* specifies the distance from the center of the triangle to any of its vertices.

* *You should make all the instance variables and methods public by using the* `public` *access modifiers.*

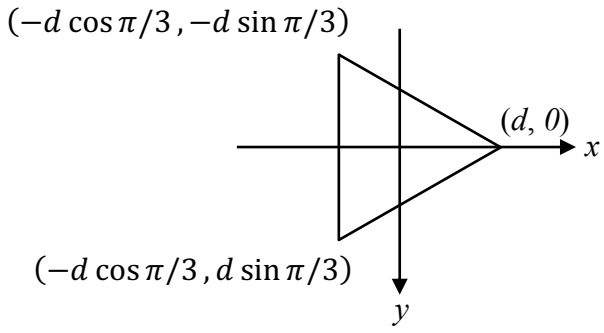



**Figure 3. The local coordinates of the 3 vertices of a standard triangle with a distance of *d* from its center to any of its vertices.**

For a standard triangle with a distance of *d* from its center to any of its vertices, the local coordinates of its 3 vertices in counter clockwise order starting from the one on the positive *x*-axis are $(d, 0)$, $(-d\cos\pi/3, -d\sin\pi/3)$ and $(-d\cos\pi/3, d\sin\pi/3)$, respectively.

**The Circle class**

The Circle class is used to model circles. Like the Square class and the Triangle class, it is a subclass of the Shape class and it inherits all the instance variables and methods of the Shape class. The Circle class overrides the `setVertices()` method for setting the local coordinates of the upper left and lower right vertices of an axis-aligned bounding box of a standard circle, as well as the `getX()` and `getY()` methods for retrieving the screen coordinates of the upper left and lower right vertices of this bounding box. Below is a detailed description for the Circle class.

Specification of the Circle class:

*instance variables inherited from Shape:*

`color, filled, theta, xc, yc, xLocal, yLocal`

*methods inherited from Shape:*

`translate, rotate`

*overriding methods:*

`void setVertices(double d)` – a method for setting the local coordinates of the upper left and lower right vertices of an axis-aligned bounding box of a standard circle. Here, a standard circle is a circle having its center located at (0, 0) of its local coordinate system. The parameter *d* specifies the radius of the circle.

`int[] getX()` – a method for retrieving the *x*-coordinates of the upper left and lower right vertices of an axis-aligned bounding box of the circle in the screen coordinate system (rounded to nearest integers).

`int[] getY()` – a method for retrieving the *y*-coordinates of the upper left and lower right vertices of an axis-aligned bounding box of the circle in the screen coordinate system (rounded to nearest integers).

* *You should make all the instance variables and methods public by using the* `public` *access modifiers.*

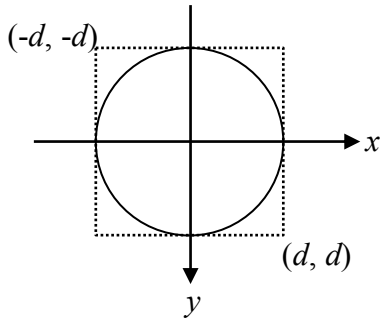



**Figure 4. The local coordinates of the upper left and lower right vertices of an axis-aligned bounding box of a standard circle with a radius of *d*.**

For a standard circle with a radius of *d*, the local coordinates of the upper left and lower right vertices of its axis-aligned bounding box are (*-d*, *-d*) and (*d*, *d*), respectively. Their corresponding screen coordinates are (*-d* + *xc*, *-d* + *yc*) and (*d* + *xc*, *d* + *yc*), respectively, where (*xc*, *yc*) denote the screen coordinates of the center of the circle.

### The tester classes

The tester classes are used to verify the correctness of your implementation of the above 4 classes. You should design your own tester classes. Generally, your tester classes should create an object of a class and access all its instance variables and methods using the dot operator, and print out debugging messages to the console. (You do not need to actually draw the shapes in your tester classes!)

## *A simple GUI for visualization*

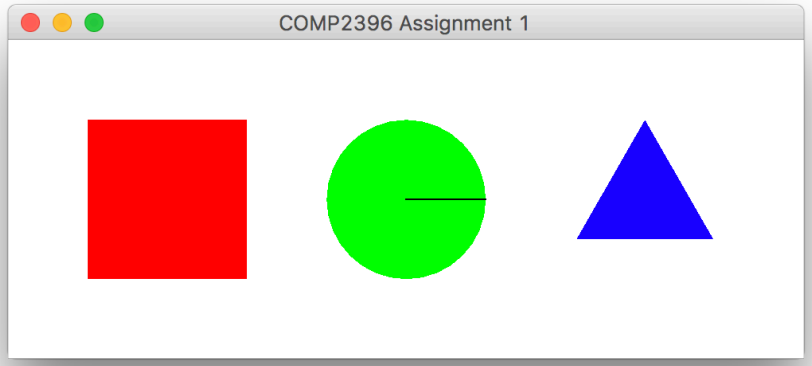



**Figure 5.** `Assign1_GUI.java` **implements a simple GUI for visualizing the shape classes.**

`Assign1_GUI.java` implements a simple GUI for you to test and visualize your shape classes. The GUI draws a square, a circle and a triangle on the canvas. When the user clicks on a shape, the shape rotates clockwise 360°.

## *Marking Scheme*

Marks are distributed as follows:

- Implementation of the Shape class and its tester class (40%)
- Implementation of the Square class and its tester class (10%)
- Implementation of the Triangle class and its tester class (10%)
- Implementation of the Circle class and its tester class (20%)
- Javadoc and comments (20%)

## *Submission*

Please pack the source code (`*.java`) of your shape classes and tester classes into a single zip file, and submit it to the course Moodle page.

A few points to note:

- Always remember to write Javadoc for all public classes and their public class members.
- Always remember to submit the source code files (`*.java`) but **NOT** the bytecode files (`*.class`).
- Always double check after your submission to ensure that you have submitted the most up-to-date source code files.
- Your assignment will not be marked if you have only submitted the bytecode files (`*.class`). You will get zero mark for the assignment.
- Please submit your assignment on time. Late submission will not be accepted.

~ End ~