Lab 04 – Using Inheritance and Polymorphism

Objectives:

- Learn to derive subclasses from an existing class
- Understand the advantage of using inheritance
- Understand the practical use of polymorphism
- Identify the difference between public, private, and protected

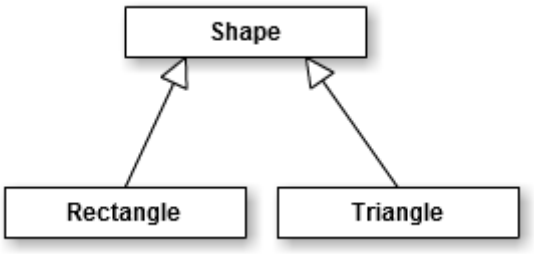
In this tutorial, you are given two class templates - Rectangle and Triangle to start with. They represent two types of shape respectively and provide certain basic methods to print themselves to screen.

1. Copy the two classes (in appendix) to your project. Each of the classes has a main method to create and display a few shapes for testing. The expected outputs are shown here for your reference. You are asked to complete their draw() method to produce similar output. Study their logic and structure before you start to write the program.

```
Rectangle (width:1, height:1, char:*)
                                        *
Rectangle (width:3, height:3, char:*)
                                       ***
                                       ***
                                       ***
Rectangle (width:10, height:5, char:#)
                                    ##########
                                    ##########
                                    ##########
                                    ##########
                                    ##########
```

```
Triangle (height: 5, char *)
                                         *
                                        ***
                                       *****
                                      *******
                                     *********
Triangle (height: 10, char @)
                                          @
                                         @@@
                                        @@@@@
                                       @@@@@@@
                                      @@@@@@@@@
                                     @@@@@@@@@@@
                                    @@@@@@@@@@@@@
                                   @@@@@@@@@@@@@@@
                                  @@@@@@@@@@@@@@@@@
                                 @@@@@@@@@@@@@@@@@@@
```

2. It is found that there is a great portion of code common to these two classes. You are therefore asked to "refactor" them, which means to merge their common parts into a parent class Shape for extending. The UML diagram of their relationship is as follow:



When refactoring the classes, carefully choose and critically justify whether certain variables or methods are appropriate to be defined in the class.

i. Write down all the common parts for the two children
ii. Write down the reason for moving a member field (method/variable) to parent

Reflection:

What is the advantage of refactoring the classes (using inheritance) in the above example? Consider the situation that there could be more varieties (more subclasses) of Shape and the implementation of the Shape class may be changed in a future release.

3. To test the classes, write a main program, ***TestShape***, to create some instances of the Rectangle or Triangle. Your program needs to ask the user to enter the number of shapes to create and the corresponding parameters for creating the shapes. You may define an array of Shape to store the instances of Rectangle and Triangle altogether. Then you can print them in a loop by calling

```
shapes[i].draw();  // shapes[] is the array of Shape
```

For the above step to work, you must also define a dummy draw() method (with empty body) in the Shape class which is then overridden by its subclasses. A sample dialogue is shown below:

```
How many shapes to create? 3
Creating 3 shapes.
1) Rectangle or Triangle ? ("R" or "T"): R
Enter the rectangle's width height (integer integer): 2 3
2) Rectangle or Triangle ? ("R" or "T"): T
Enter the triangle's height (integer): 3
3) Rectangle or Triangle ? ("R" or "T"): r
Enter the rectangle's width height (integer integer): 4 2
Enter drawing character for all shapes: *
Rectangle (width: 2, height: 3, char: *)
                                        **
                                        **
                                        **
Triangle (height: 3, char *)
                                         *
                                        ***
                                       *****
Rectangle (width: 4, height: 2, char: *)
                                        ****
                                        ****
```

Reflection:

Explain why a dummy draw() method is needed in the Shape class. How this problem is related to **static type checking** and **dynamic method binding**?

4. **[Self-checkpoint]** The size of a shape, either Rectangle or Triangle, is represented by the count of printed characters in the draw() method. Suppose you have a Shape array with an arbitrary number of Rectangle and Triangle instances, write a method to find the biggest shape in the array and return its index.

```
public static int max(Shape[] shapes)
```

The ***TestShape*** program is then modified to find and display the biggest shape from the input as follow.

```
How many shapes to create? 3
Creating 3 shapes.
1) Rectangle or Triangle ? ("R" or "T"): R
Enter the rectangle's width height (integer integer): 2 3
2) Rectangle or Triangle ? ("R" or "T"): T
Enter the triangle's height (integer): 3
3) Rectangle or Triangle ? ("R" or "T"): r
Enter the rectangle's width height (integer integer): 4 2
Enter drawing character for all shapes: @
Rectangle (width: 2, height: 3, char: @)
                                        @@
                                        @@
                                        @@
Triangle (height: 3, char @)
                                         @
                                        @@@
                                       @@@@@
Rectangle (width: 4, height: 2, char: @)
                                       @@@@
                                       @@@@
The biggest shape is at [1]:
Triangle (height: 3, char @)
```

APPENDIX - *Rectangle and Triangle Class*

```
/**
 * Rectangle.java
 *
 * Draws rectangles using character in the middle (horizontally) of the screen.
 * Object-oriented style.
 *
 * @author vanting
 */
public class Rectangle {

    //=== instance variables ===
    /** Width of this Rectangle */
    private int width;
    /** Height of this Rectangle */
    private int height;
    /** Character used to draw this Rectangle, default to '*' */
    private char drawingChar = '*';
    //=== constants ===
    /** The middle column in the screen */
    private static final int MIDDLE = 40;

    //=== constructor ===
    /**
     * Constructor with given width and height.
     */
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    //=== static methods (utility methods) ===
    /**
     * Draws a number of characters horizontally.
     * @param ch the character to draw
     * @param num the number of characters to draw
     * @remark This is a static method, as it is not specific to a particular
     * instance of Rectangle.
     */
    static private void drawChars(char ch, int num) {
        for (int i = 0; i < num; i++) {
            System.out.print(ch);
        }
    }

    //=== instance methods ===
    //=== accessor and mutator methods
    /**
     * Retrieves the width.
     * @return the width.
     */
    public int getWidth() {
        return width;
    }

    /**
     * Changes the width.
     * @param width the new width to set.
     */
    public void setWidth(int width) {
        this.width = width;
    }

    /**
     * Retrieves the height.
     * @return the height.
     */
    public int getHeight() {
        return height;
    }

    /**
     * Changes the height.
     * @param height the new height to set.
     */
    public void setHeight(int height) {
        this.height = height;
```

```
    }

    /**
     * Retrieves the drawing character.
     * @return the drawing character.
     */
    public char getDrawingChar() {
        return drawingChar;
    }

    /** Changes the drawing character.
     * @param drawingChar the new drawing character.
     */
    public void setDrawingChar(char drawingChar) {
        this.drawingChar = drawingChar;
    }

    /**
     * Draws this Rectangle to the screen.
     */
    public void draw() {

        // complete this method

    }

    /**
     * Gets a String representation of this Rectangle.
     * @return a String representation of this Rectangle.
     */
    @Override
    public String toString() {
       return "Rectangle (width:" + width + ", height:" + height + ", char:" + drawingChar + ")";
    }

    /**
     * The main method, creating a few Rectangles and testing the methods.
     */
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(1, 1);
        System.out.println(r1);
        r1.draw();

        Rectangle r2 = new Rectangle(1, 1);
        r2.setWidth(3);
        r2.setHeight(3);
        System.out.println(r2);
        r2.draw();

        Rectangle r3 = new Rectangle(10, 5);
        r3.setDrawingChar('#');
        System.out.println(r3);
        r3.draw();
    }
}
```

```java
/**
 * Triangle.java
 * Object-oriented style.
 *
 * @author vanting
 */
public class Triangle {

    private int height;
    private char drawingChar = '*';
    private static final int MIDDLE = 40;

    static private void drawChars(char ch, int num) {
        for (int i = 0; i < num; i++) {
            System.out.print(ch);
        }
    }

    public Triangle(int height) {
        this.height = height;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public char getDrawingChar() {
        return drawingChar;
    }

    public void setDrawingChar(char drawingChar) {
        this.drawingChar = drawingChar;
    }

    public void draw() {

        // complete this method

    }

    @Override
    public String toString() {
        return "Triangle(height:" + height + ", char:" + drawingChar + ")";
    }

    public static void main(String[] args) {
        Triangle t1 = new Triangle(5);
        System.out.println(t1);
        t1.draw();

        Triangle t2 = new Triangle(10);
        t2.setDrawingChar('@');
        System.out.println(t2);
        t2.draw();
    }
}
```

- END -