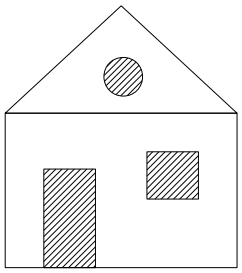

In the lecture you learned about object-oriented programming and about abstract and concrete classes (Generische und Konkrete Klassen). The following story is the motivation for this exercise:

Mr. Brush is hired to paint a house facade. To make an offer he needs to know the area that requires painting. He looks at the house facade and sees that it can be approximated by using different shapes: Triangle (width, height), Rectangle (width, height) and Ellipses (radiusx, radiusy). Holes in shapes can be accounted for by subtracting the non-paintable areas, e.g. a door or a window.



Your task is to help Mr. Brush by implementing the three required shapes: `Rectangle`,`Triangle` and `Ellipse`. Each shape is implemented with its own class that inherits from class `Shape` (Konkretisierung der Klasse `Shape`) and overrides the virtual member functions `getArea` and `draw`.

To solve this exercise you should download the skeleton project at: http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex08/ex08.zip and import it into eclipse. You can import the project by going File → Import. Then pick "Existing Project into Workspace" under *General* and click *Next*. Then instead of "Select root directory", choose "Select archive file" and navigate to the file ex08.zip you just downloaded. Select all and click *Finish*.

The Skeleton project contains 2 files: Main.java and ImageViewer.java. You can ignore the file ImageViewer.java. Also we implemented a dummy ImageViewer class in the Main.java file which is required to evaluate exercise 8.2 on the judge. You should not change the code of this class.

Main program

The main() function in the class Main is very similar to what you have seen in exercise 7. Again we have as scanner to read in the input and depending on a command `cmd` the program is executing one of the branches in the if-else block. The commands in this exercise are *"newtriangle"*, *"newrectangle"*, *"newellipse"*, *"paintarea"*, *"drawshapes"*. The first 4 commands are used in part 8.1 and the last command in 8.2

```
main(String[] args) {
    while (scanner.hasNext()) {
        String cmd = scanner.next();
        if (cmd.equals("newtriangle")) { // Insert a new triangle
            ...
        } else if (cmd.equals("newrectangle")) { // Insert a new rectangle
            ...
        } else if (cmd.equals("newellipse")) { // Insert a new ellipse
            ...
        } else if (cmd.equals("paintarea")) { //Calculate total are to be pain
            ...
        } else if (cmd.equals("drawshapes")) { // Draw all shapes
            ...
        } else {
            System.out.println("Invalid input. terminating...");
            break;
        } } }
```

As you can see in the skeleton code, depending on the command `cmd` a different branch of the if-else block is executed. The commands *"paintarea" and "drawshapes"* are already implemented in the skeleton provided. You should have a look at the code in the skeleton to understand how these commands work, since they will make use of the code you are going to implement.

8.1 Calculating Paint Area

In the first part of this exercise you have to calculate the total area of the house which has to be painted. As explained we model the house with shapes which are either painted (walls) and shapes which are not painted (doors, windows). In the Main.java we provide you with the following abstract class Shape:

```
abstract class Shape {
    // isPaintable defines if this shape will be painted or not
    Shape(boolean isPaintable) { ... }
    // This method sets the coordinates(x,y) of the shape
    public void setCoordinates(int x, int y) { ... }
    // This method returns the color of the shape, black for paintable shapes
    // and red for not-paintable shapes
    public Color getColor() { ... }
    // Returns area of the shape
    abstract public double getArea();
    // Draws the shape on the ImageViewer
    abstract public void draw(ImageViewer viewer);
    // variable to indicate if shape is paintable
    boolean     isPaintable;
    // coordinates of the shape
    int x;
    int y;
}
```

The constructor and the functions `setCoordinates and getColor` are already implemented in the skeleton provided. As you can see from the code each Shape has a variable isPaintable to indicate if this shape will be painted (wall) or not (windows, doors). Also each shape has a coordinate (x,y).

Your task is to implement the classes `Rectangle,Triangle and Ellipse` which extend this abstract shape class. Then you implement the function `getArea()` in all these classes.

After you implemented these three classes you need to implement the commands *"newtriangle", "newrectangle" and "newellipse"*. In the main function of the Main class. To add a new shape the following inputs are used:

```
newtriangle + 200 200 50 75
newrectangle - 100 100 50 50
newellipse - 100 100 25 35
```

This example input inserts a new triangle the + sign indicates that this shape is painted, the triangle has the coordinates (200,200), a width of 50 and height of 75. In the second line a new rectangle is added the − sign indicates that this shape is not painted, the rectangle has the coordinates (100, 100), width 50 and height 50. In the third line a new ellipse is added at position (100,100), with a radiusx of 25 and a radiusy of 35.

For each shape you have to implement the corresponding command in the if-else block in the main function. Here we are gonna explain the *"newrectangle"* command. Implement the functionality in

```
...
String cmd = scanner.next();
..
else if (cmd.equals("newrectangle")) { // Insert a new rectangle
    // TODO implement this
}
...
```

As you can see the command is already read in with *scanner.next()*. To read the + or − sign use *scanner.next()* which returns a string, to read in the coordinates and the width and height of the rectangle use *scanner.nextInt()*. Create a new `Rectangle` object using the height, width and a boolean which is `true` in case of a + sign and `false` in case of − sign. Set the coordinates of the newly created object by using the function `setCoordinates`. Then add the newly created Rectangle to the list `shapes` defined at the beginning of the main() function, the new rectangle can be added the following way:

```
// Add new rectangle to the list of shapes
shapes.add(...);
```

As the list `shapes` is of type `ArrayList<Shape>`, you can add any object (Rectangle, Triangle, Ellipse) which inherits from the class `Shape`.

The implementation of the the *"newtriangle" and "newellipse"* branches in the if-else block in the main function is analogous to the *"newrectangle"* command explained above.

Since the classes `Rectangle,Triangle and Ellipse` are all subclasses of `Shape` and the function `draw` is abstract in `Shape`. All its subclasses are required to implement this function. However for this part of the exercise the `draw` function is not executed, so you can simply add and empty function called draw or let it auto-generate by eclipse, like this:

```
@Override
public void draw(ImageViewer viewer) {
// TODO Auto-generated method stub
}
```

Add this stub function to all 3 subclasses.

After you implemented all this functionality, you can validate your solution here: `https://challenge.inf.ethz.ch/team/websubmit.php?problem=IB16081`.

8.2 Drawing Shapes

In Assignment 7, you learned how to extend a Polygon class with a draw function. This week you should add a draw() function all classes which extend the `Shape` class: `Rectangle,Triangle and Ellipse`. As you can see from the *"drawshapes"* branch in the main function, we want to iterate through all shapes and draw them.

Like in the last exercise, we use the ImageViewer class to draw the figures. You do not need to understand the internals of this class. Here we give again a brief overview how you can use it to draw the different shapes.

```
// Create an ImageViewer, with width: 800, and height: 600
ImageViewer viewer = new ImageViewer(800,600);
// Draw a black line on ImageViewer viewer,
// from Point (100, 100) to Point (200, 200)
viewer.line(100, 100, 200, 200, Color.black);
// Draw a red ellipse at the coordinate (150, 150)
// with radiusx (200) and radiusy (100);
viewer.ellipse(150,150, 200, 100, Color.red);
// Update the viewer, required to draw everything
viewer.update();
```

In the example above we first create a new ImageViewer. You can either draw a line on the ImageViewer with the function line() or draw an ellipse at a specific coordinate with the function `ellipse`. Use the color returned by the `getColor` function defined in the `Shape` class.

Implement the following draw function in all shape classes, `Rectangle,Triangle and Ellipse`:

```
public void draw(ImageViewer viewer)
{
    // TODO draw figure on ImageViewer viewer
    // Use the color from the getColor() function of the Shape class
}
```

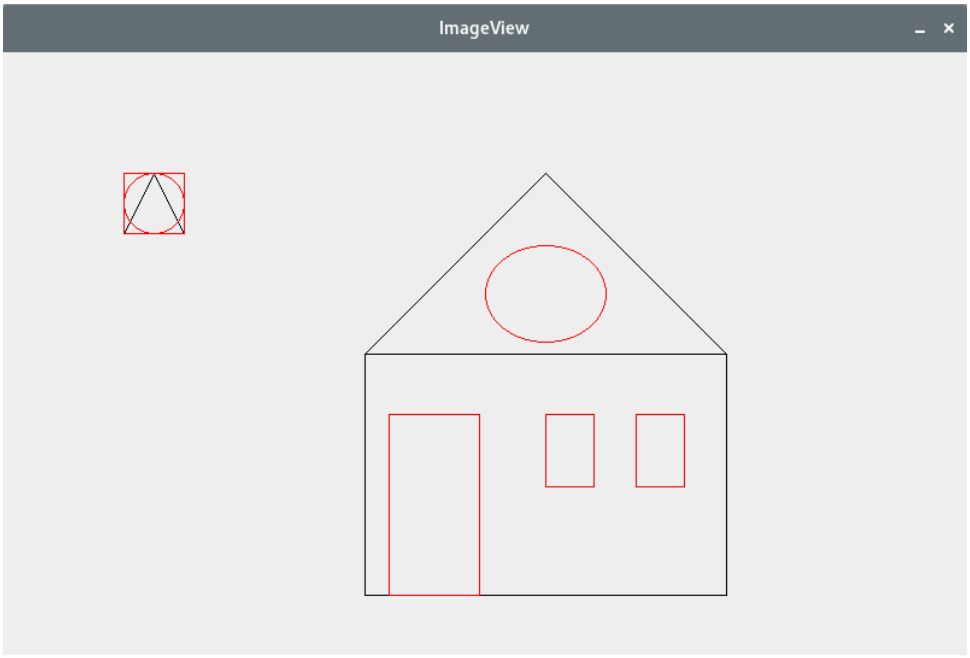
To draw the rectangle and triangle use the `line` function, for the ellipse you can use the `ellipse` function. For the Rectangle the coordinate defines the left top corner, for the Triangle the top corner and for the Ellipse the center point.

To test your implementation on the Judge we implemented a dummy ImageViewer class which instead of drawing the shapes, it prints out the drawings in text form. You can validate your solution here: https://challenge.inf.ethz.ch/team/websubmit.php?problem=IB16082.

To use the real ImageViewer and draw the shapes, comment the dummy ImageViewer class defined at the beginning of the Main.java file using `/* ... */`. And uncomment the following line:

```
//import visual.ImageViewer;
```

Now if you run your implementation again using the test input provided, you should see the following drawing:



Note: When submitting your code to the judge you have to use the dummy ImageViewer class, otherwise you will get a `COMPILER-ERROR`.