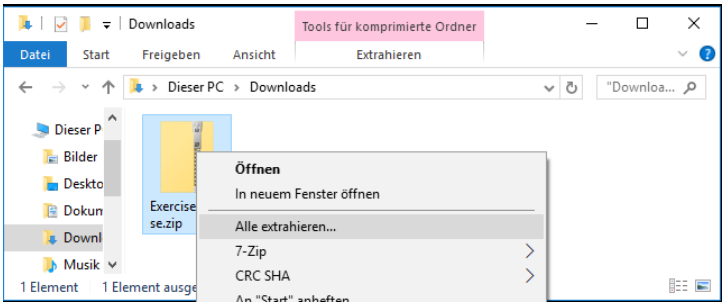
OPMS 2018 - Exercise 3

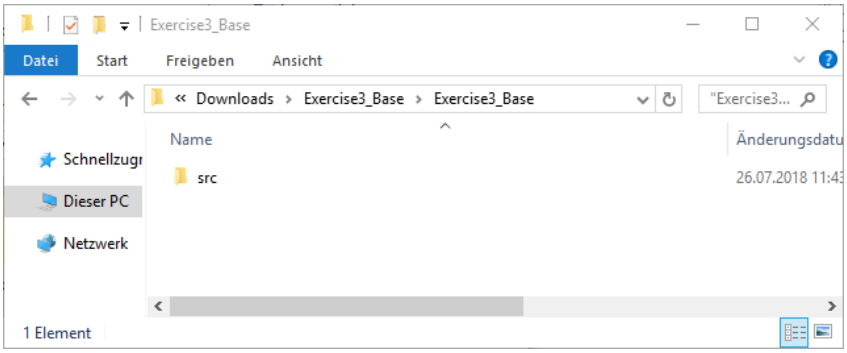
Task 1

Completing the following task requires to import some existing code first.

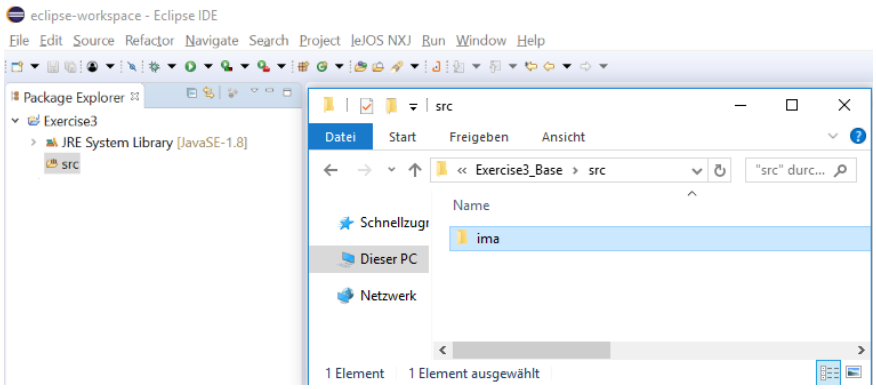
1. Download the “Exercise3_Base.zip” file from the location specified by your instructor.
2. Start your eclipse and create a new project “Exercise3”
3. Open the downloaded zip-file and copy the content into your new project. The subsequent image script shows what you need to do.



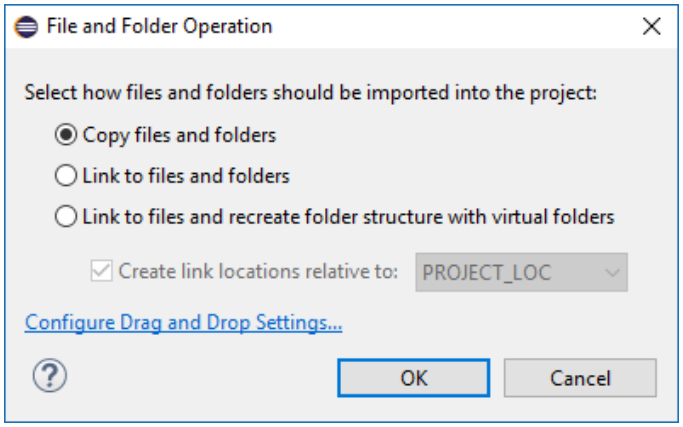
Picture 1: Extract the zip-file



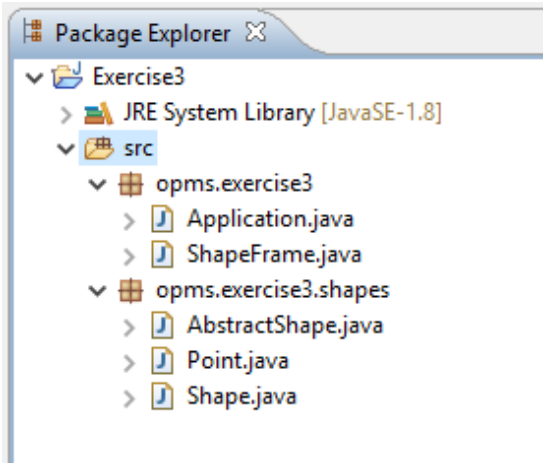
Picture 2: Content of extracted zip-file (src folder and subfolders)



Picture 3: Copy content of src folder of the zip-file into src folder of your project (you can use drag-and-drop)



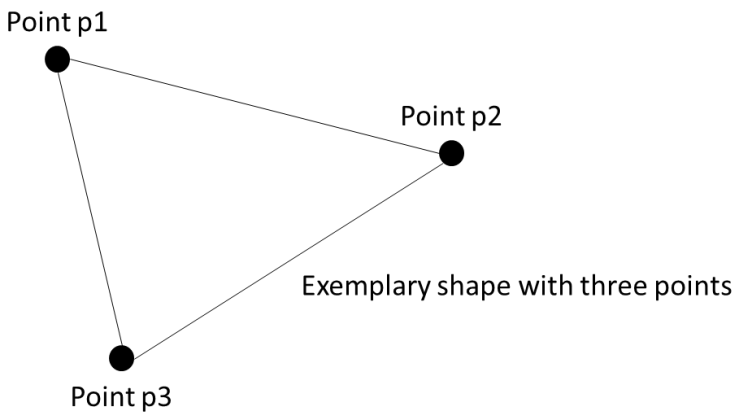
Picture 4: In the subsequent dialog select the first option



Picture 5: After copying the files, it should look like this screenshot

Look into the different classes of the package `ima.summerschool.exercise3.shapes`. Afterwards try to understand the class `Application`. The application first initializes a list of shapes and second uses the implementation of `ShapeFrame` (which you do not need to understand) to draw the different shapes in a GUI.

We define a shape as a list of points. The first point defines the starting point, so that from this point onwards a line is drawn from each point in the list. An example of a shape is depicted in the following picture.



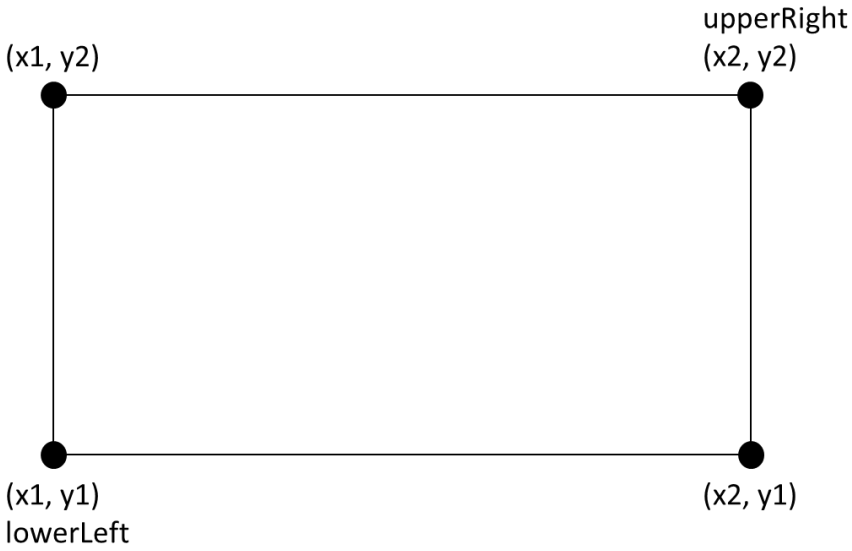
Picture 6: A shape defined by three points {p1, p2, p3}

Furthermore, a shape has a color and a fill-state.

a. Create a new class `Rectangle` in the package `opms.exercise3.shapes`. This class extends the class `AbstractShape`, which provides you with all the base implementations to define a shape. Next, implement two constructors:

```
(1) public Rectangle(Point lowerLeft, Point upperRight, Color
    color, boolean filled)
(2) public Rectangle(int x1, int y1, int x2, int y2, Color
    color, boolean filled)
```

Both has to define a rectangle. Thereby, a rectangle consists of four points (see Picture 7). The constructors therefore has to add the needed point to the inherited shape by using the corresponding `addPoint`-Method. Further, the given color and the fill-status has to be set using the corresponding inherited setters.



Picture 7: Definition of a rectangle

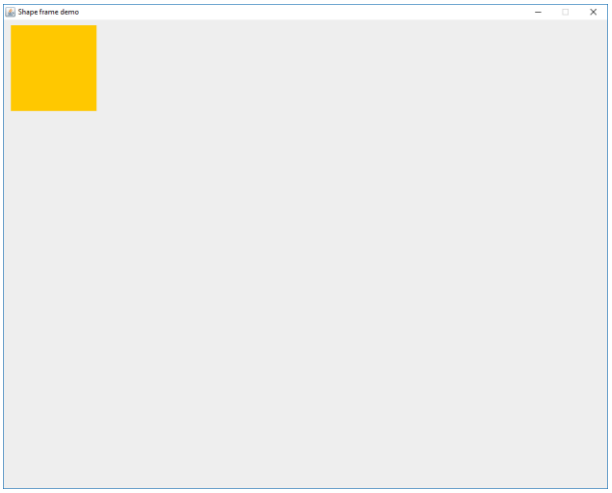
Extend the implementation of your `Application`, so that you add one Rectangle (see Picture 8).

Application.java

```
1  package opms.exercise3;
2
3  import java.awt.Color;
9
10 public class Application {
11
12     public static void main(String[] args) {
13
14         // a list of shapes that have to be drawn
15         List<Shape> shapes = new ArrayList<>();
16         shapes.add(new Rectangle(75, 100, 225, 225, Color.ORANGE, true));
17
18         // create the frame and draw the shapes
19         ShapeFrame frame = new ShapeFrame();
20         frame.setVisible(true);
21         frame.drawShapes(shapes);
22     }
23
24 }
25
```

Picture 8: Extension of the `Application` implementation

Run your code. If your implementation is correct, it should look like depicted in the following picture.

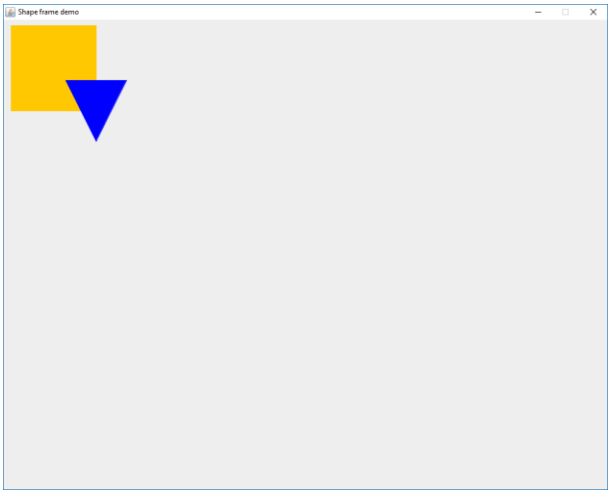


Picture 9: Generated window with rectangle

b. Create a new class `Triangle` in the package `opms.exercise3.shapes`. This class also extends the class `AbstractShape`. Next, implement one constructor:

```
(1)  public Triangle(Point p1, Point p2, Point p3, Color color,
     boolean filled)
```

Extend your application, so that besides the rectangle, a triangle is drawn as depicted in Picture 10.

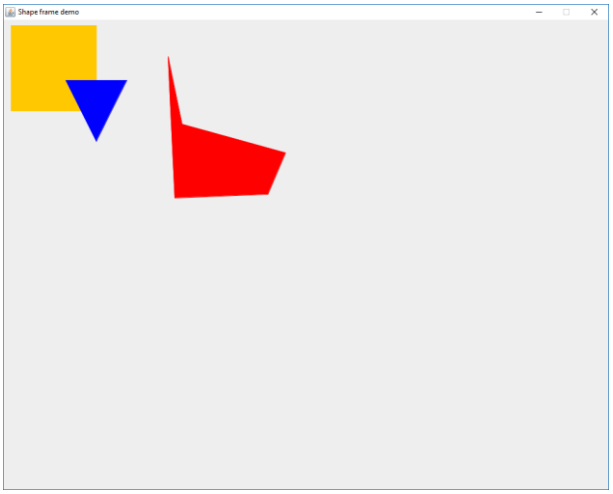


Picture 10: Generated window with rectangle and triangle

c. Create a new class `Polygon` in the package `opms.exercise3.shapes`. This class draws a polygon with any number of points. The number of points is set during construction by passing a list of points to the constructor.

```
(1)     public Polygon(List<Point> points)
```

Extend your application, so that besides the rectangle and the triangle, a polygon is drawn as depicted in Picture 11.



Picture 11: Generated window with rectangle, triangle and random polygon

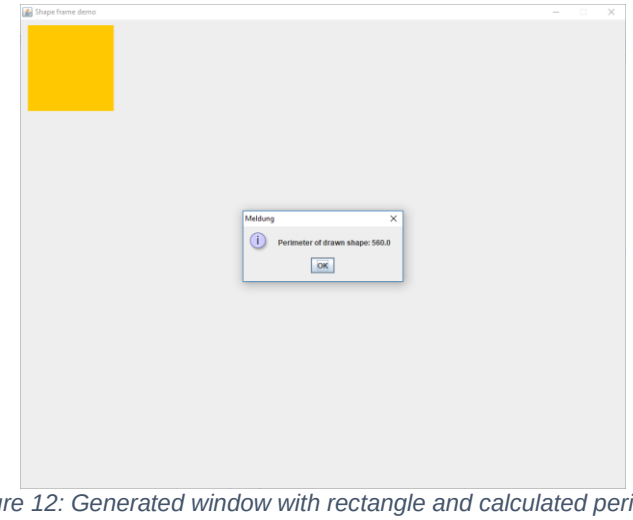
d. Extend your class `Point` by adding a method to calculate the distance to another point. The signature and return type of the method should be:

```
public double distance(Point p)
```

e. Extend your class `AbstractShape` by the following method:

```
public double calculatePerimeter()
```

Next, implement this method by summing the distances between the points. Remember to add the distance between the last and the first point. If your implementation is correct, the application should show you the calculated perimeter for each shape (after it has been drawn) – see Picture 12.



Picture 12: Generated window with rectangle and calculated perimeter