Crash Course Java
Shapes and Lists

louisV, mareinK, koenD

2015

We will be continuing with the code you built for the previous assignment. Load the project you created earlier; we'll transform the ColorClicker application into something more complex.
We will be creating an application that can draw lines, rectangles and ellipses in random locations by clicking a button. There will also be a 'modifying' button that changes a certain shape to a different type (i.e. rectangle to ellipse).

1 Modify the interface

We'll need to add an extra button to the window we created in the previous exercise; after all, we want an 'add' and a 'mod' button. You can simply add this button to the ButtonPanel you already made. Don't forget to add the ButtonHandler instance as an actionlistener for the new button as well; we want the ButtonHandler class to handle events from both buttons.
Now open your ButtonHandler class; we're going to implement a check to find out which button has triggered the actionPerformed method. As you can see, the actionPerformed method gets one argument of type ActionEvent. This argument contains some information about the event source, the action command associated with the event and more. We'll be using the `getActionCommand()` method to figure out which button was pressed.
Assuming you've made two buttons with texts 'add' and 'mod', use the `getActionCommand()` to find out which button was pressed; you don't need to implement any further functionality right now.
We'll now create some methods in the RectPanel class to facilitate the adding and modifying behaviour that we want to create eventually. Add two methods: `addRandomShape` and `alterNextShape`. Don't fill them yet; we'll do so later.
Now that these two methods exists, we can call them from our ButtonHandler; it doesn't really matter that they don't do anything yet. Make sure that `addRandomShape` is called when the 'add' button is clicked and `alterNextShape` is called when the 'mod' button is clicked.

2 Shape classes

We've provided you with a base structure for shapes, found in the class `MyShape`. We've also added one example implementation (`MyRectangle`). Put these two classes in your project and make sure you understand the code! Try implementing two more shapes; `MyEllipse` and `MyLine` (these should draw ellipses and lines, respectively). We won't go into detail very far about the specifics of this class design; try and work with it, you're allowed to change both provided classes if you feel the need to do so.
If you're not sure how to implement the other shapes, try googling for 'Ellipse2D' and 'Line2D'! The `MyRectangle` class should be a good guideline.

3 Functionality

3.1 Adding shapes

Once you've implemented the two other shape classes, you can implement the `addRandomShape` method in `RectPanel`. In order to be able to paint multiple shapes with each call of `paintComponent`, we need some sort of data structure to contain the shapes. You can use any implementation of the `List` interface for this; try googling to find out which implementations exist.
You could for example add an `ArrayList` instance to the `RectPanel` class:

```
    List<MyShape> shapesList = new ArrayList<MyShape>();
```

Basically, this means: create an instance of `ArrayList` that can contain objects of type `MyShape`. Since `MyRectangle/MyEllipse/MyLine` extend `MyShape` (and thus are also of type `MyShape`), we can add all these different shapes to the same list.
Now we can easily draw all our shapes in one go by writing the `paintComponent` method like this:

```
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // Cast the graphics object to type Graphics2D
        Graphics2D g2d = (Graphics2D) g;
        // Loop through the shapesList and draw every shape
        for (MyShape s : shapesList)
            s.draw(g2d);
    }
```

Now that we have a list structure to draw shapes, try implementing the `addRandomShape` method. You can (should) use the `Random` class that Java provides you with in order to determine which type of shape to add, as well as the location of the shape. The MyShape class has a `setCoords` method that you can use to set the shapes' coordinates. The size of the shapes should be the same everytime (try 30x30 for example). To add shapes to the list, use `shapesList.add(shape)`. Don't forget to call `repaint` after you add a new shape!

3.2 Modifying shapes

Now we'll implement the `alterNextShape` method. This method should change the type of the shapes in the list, starting with the first shape and wrapping around to the first shape after the last shape in the list has been modified (imagining a list (1,2,3), shapes should be altered in a (1,2,3,1,2,3) fashion). The easiest way to do this is by replacing the shape at the current index with a whole new shape. You can again use the `Random` class to determine which type the new shape will have, and you can use the `ArrayList`'s `get(int index)` and `set(int index, MyShape s)` to get and set the shape at a certain index.
Again, don't forget to call `repaint` after substituting a shape.

4 Bonus

You can do some more cool things with the modifier button; try changing the color, location or something else about the shapes you're modifying! Another option is to add a 'delete' button that deletes shapes in the same fashion as the 'mod' button. You could also try implementing extra shapes (triangles, perfect circles, etc.).