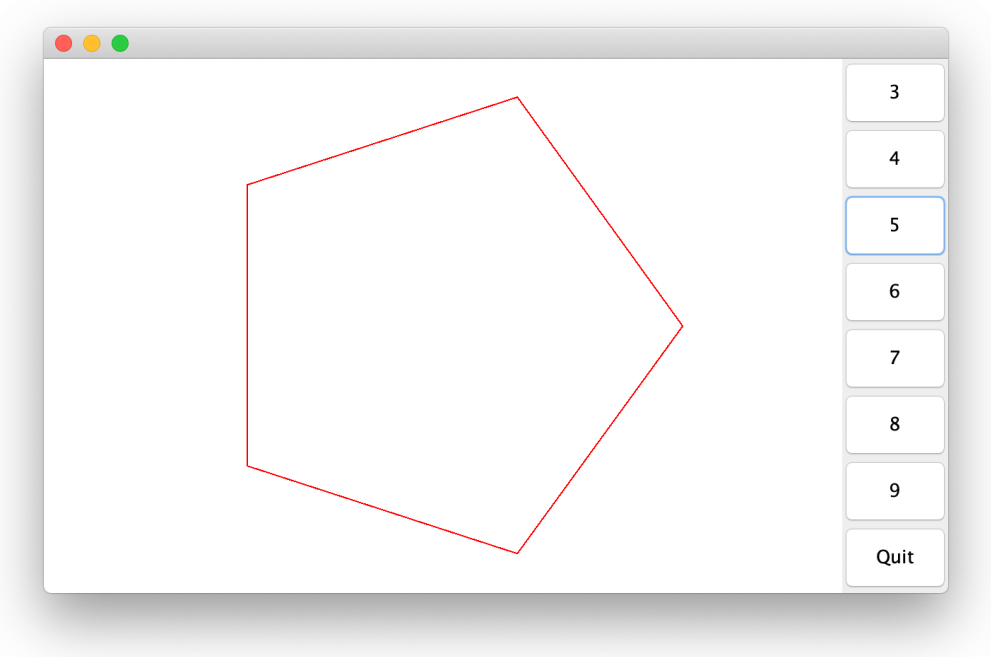
COM6516: Object oriented programming and software design: Practical session 7

The aim of this exercise is to give you experience of working with GUIs and event handling. The task builds on the 2D graphics material covered in last lab class and lecture. This practical sheet guides you through the process of building a GUI with buttons and a drawing panel. It would be a good idea to read through the entire practical sheet before you start programming.

Task 1 – buttons

The task is to write a program that displays a window with buttons labelled *3*, *4*, *5*, *6*, *7*, *8*, *9*, and *Quit*. When a numbered button is clicked a polygon with that number of sides should be drawn with a red outline in the centre of the main panel; e.g., a red triangle when the *3* button is pressed, a red square when the *4* button is pressed, etc. The figure shows an example with a pentagon drawn in response to button *5*. The *Quit* button exits the program.



How to get started

For this programming task, we need to set up a frame that has a panel to contain the column of buttons, and a panel within which the polygons are drawn. You will find `MyFrame` and `MyPanel` classes in the lab material. This works like the `SimpleFrameWithButtons` class from the previous lab class, except that layout managers control the locations of the buttons.

Creating a `MyFrame` object (in the `main` method, for example) also creates a `MyPanel` object in the centre of the frame and a column of `JButton`'s on the right (east) side.

You need to link these buttons with specific actions, using the `ActionListener` interface. When each button is drawn, the `MyFrame` object is designated as the `ActionListener` object that the button is linked to. This sets up the code to respond to user events (in this case, button presses). Refer to the Oracle tutorial and the introduction to event listeners as needed –
http://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html
http://docs.oracle.com/javase/tutorial/uiswing/events/intro.html

`MyFrame` has to implement the `ActionListener` interface's `actionPerformed` method. To see how `actionPerformed` can respond to a specific button press, write code in this method to print out messages using `System.out.println`. `ActionEvent`'s `getActionCommand` method returns the `String` name of the button pressed.

The next stage is to fill in the `MyPanel` class with methods that construct and draw `Polygon` objects as required. One of the panel's instance fields can be a reference to a `Polygon` object, initialized to be `null`. You also need to override the superclass `JPanel`'s `paintComponent` method. Note that it needs to test the polygon field for `null` before drawing. To draw `Polygon` objects on a `JPanel`, you will need to cast your `Graphics` object to a `Graphics2D` object (see last lecture and lab). `Graphics2D` has methods to draw any object that implements `Shape`. Experiment with drawing different polygons on a `JPanel`, and see the Java API for information about the `Polygon` class –
http://download.oracle.com/javase/8/docs/api/index.html?java/awt/Polygon.html

Now write a `setPolygon(int sides)` method, which sets the instance field `polygon` to refer to a polygon with the specified number of sides. This method should be called by `actionPerformed` when a button is pressed. These formulas will help:

```
x = xCentre + radius * Math.cos(rads);
y = yCentre + radius * Math.sin(rads);
```

where `rads` is an angle in radians and `radius` is the radius of the circle the polygon sits in. A `JPanel` has methods for finding its own height and width, which you can use to calculate the centre coordinates. You can calculate the angle increment by dividing 2*∏ radians (i.e., 360°) by the number of sides, so something like this:

```
double increment = 2 * Math.PI / sides;
```

You will need to turn `double` values into arrays of `int`.

You may find that a polygon is not always displayed unless you resize the window, because `Swing` buffers the contents of the window; `paintComponent` is called initially when the window becomes visible and afterwards when you resize it. You can include `this.repaint` in your `setPolygon` method to force a call of `paintComponent` and update the display of the `MyPanel`.

Task 2 – radio buttons
Modify your code to replace all the buttons with radio buttons –
http://docs.oracle.com/javase/tutorial/uiswing/components/button.html
(Note that `ButtonGroup` is not a `Swing` component.)

Add code to handle `ComponentEvent` by adding a `ComponentListener` so that your code can resize polygons in response to changes to window size. In this way, you can keep resizing the window and polygons always fit inside it.