**Tutorial 3**

**The learning in this tutorial continues the Implementation section of Software Engineering to help you get more familiar with Java and object-oriented design, but also introduces user interface design important to the Project Management section that could potentially be used in your group projects either in prototyping, or in actual implementation, but which again should be immediate and fun.**

At the end of this tutorial you should be comfortable with the following:-

* Subtype polymorphism
* Writing a simple Graphical User Interface using SWING
* Creating objects of anonymous class and overriding their methods to process UI input

**Ex1 Adding a List of Shapes – subclass polymorphism**

1. Open the ShapeDraw program that you wrote in Tutorial 2.
2. Add a field that is of class ArrayList<Shape>
3. In the constructor of the Drawing class, create circles, squares and rectangles and add them to the list instead of having separate fields for each.
4. In Drawing’s paint method, use a for loop to iterate through the list’s items and call the draw method of each item.
5. Rather that use an abstract method ‘draw’ in Shape, implement it as an interface.

**Ex2 Create a Simple Window**

We are going to create a GUI program, but using a SWING JFrame object instead of an AWT Frame object as we did in the previous program. JFrame is a component of the SWING user interface library that inherits from AWT’s Frame and implements a window.

1. Create a new Gradle project and add it to Github – call it something like SimpleUI.
2. Add the application plugin to build.gradle as in the previous program, and set source compatibility 1.8.
3. Create a new class called SimpleUI.
4. Add the following code to the class and check that it creates a window when run:-

**static** GraphicsConfiguration *gc*; *// Contains this computer’s graphics configuration***public static void** main(String[] args) {  
 JFrame frame= **new** JFrame(*gc*); *// Create a new JFrame*  
 frame.setSize(500,300);   
 frame.setVisible(**true**);

*// This next line closes the program when the frame is closed*

**frame**.addWindowListener(new WindowAdapter() { *// Closes the program if close window clicked*

**public void** windowClosing(WindowEvent e) {

frame.dispose();

}

});

}

**Ex3 Add a Panel Containing a Button**

A JPanel is a window that can reside inside a JFrame and can contain controls – or other JPanels. We are going to create one and add a button to it. We are also going to create the code that is called when the button is pressed.

1. Create a new class called something like ButtonPanel that extends JPanel.
2. Create a field of class JButton in the ButtonPanel class and instantiate it in the ButtonPanel’s constructor. The constructor of JButton can take a character string which is the text on the button.
3. Call a method of the JButton called addActionListener and pass into it an object of anonymous class that inherits from ActionListener. Override its actionPerformed abstract method as follows:-

**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {

System.out.println(“Button 1 pressed”);  
 }  
}

1. ‘add’ this JButton to the ButtonPanel
2. Finally, in the SimpleUI class, before frame.setVisible(**true**); put:-

frame.getContentPane().add(new ButtonPanel()); This adds the ButtonPanel to the root client area of the JFrame

1. Build and run the program – you should see a button inside the window and pressing it should execute the action you put in actionPerformed.

**Ex4 Controlling Layout**

The layout of controls within a JFrame and a JPanel can be controlled by a layout manager. We will now use an object of class GridLayout, whose constructor takes the number of rows and columns of the grid and where each box in the grid can contain a control.

1. In the ButtonPanel constructor, call its setLayout method and pass in an object of class GridLayout that has one row and one column. (GridLayout’s constructor takes two integers – the number of rows and the number of columns.)
2. Try creating a larger grid, and add more buttons to it.

**Ex5 Adding a Canvas**

The Drawing class in your previous program extended the Canvas class. Canvases can also be added to JPanels as if they were components. Try it! Make sure that your shapes are drawing in the visible area of the canvas.

**Ex6 Controlling Drawing**

Using JPanels and Layout Managers, add a panel of controls side-by-side with ‘Drawing‘ where you can:-

1. Use three JSliders to set an RGB drawing colour
2. Select Circle, Square or Rectangle from a drop down list
3. Have a slider that can set a shape length (This length can correspond to side length or radius)

Now call addMouseListener in Drawing’s constructor and pass in an object of anonymous class that inherits from MouseListener. In the override of mouseClicked, add a new object to your shape arraylist formed from your settings above.

**Ex7 Menus**

Here is a class that creates a simple JMenuBar with one menu and one menu item. MainMenu ‘is a’ JMenuBar and also ‘has a’ ActionListener interface.

*// MainMenu is a JMenuBar that also implements the ActionListener interface by  
// having a concrete actionPerformed method***public class** MainMenu **extends** JMenuBar **implements** ActionListener{  
 JMenu **menu**; *// One menu* JMenuItem **item**; *// Which has one item* **public** MainMenu(){ *// MainMenu constructor*   
 **menu**=**new** JMenu(**"Actions"**);  
 **item**=**new** JMenuItem(**"Clear all"**);

*// This MainMenu object is also acting as the ActionListener object*  
 **item**.addActionListener(**this**);

add(**menu**); *// Add the JMenu to the JMenuBar* **menu**.add(**item**); *// Add the JMenuItem to the JMenu* }  
  
 *// If 'this' is set as the ActionListener to all menu items as above, the method*

*// below will be called whenever ANY menu item is selected, but the ActionEvent’s  
 // ’command’ will be the text of the menu item selected* **public void** actionPerformed(ActionEvent e) {  
 System.***out***.println(e.getActionCommand());  
 }  
}

1. Add this class to your project, instantiate it and add it to the frame using the setJMenuBar method of the JFrame.
2. When the ‘Clear all’ menu item is selected, clear Drawing’s list of shapes. You can force a repaint of the Drawing with Drawing’s ‘repaint’ method, inherited from Canvas.