COMP132 – Computer Science II
Fall 2015

Homework #5
Graphical User Interfaces
Solutions

1. Give a snippet of Java code that performs each of the following tasks. You do not have to give a full program, just the necessary lines of code.

a. Create a `JButton` that is labeled "Panic Button".

```
JButton panic = new JButton("Panic Button");
```

b. Create a `JSider` with values that range from 32…212 inclusive, the slider should have major tick marks every 50 units and minor ticks every 10 units. The initial value of the slider should be 98.

```
JSlider slider = new JSlider(32, 212, 98);
slider.setMajorTickSpacing(50);
slider.setMinorTickSpacing(10);
slider.setPaintTicks(true);
slider.setPaintLabels(true);
```

c. Create a `JLabel` with the text "Don't Label Me" that will appear in red.

```
JLabel non = new JLabel("Don't Label Me");
non.setForeground(Color.red);
```

d. Create a `JCheckBox` that has the label "Check 1 2" and is checked.

```
JCheckBox check = new JCheckBox("Check 1 2");
check.setSelected(true);
```

e. Create a `JTextField` that is 30 columns wide and contains the text "Can't touch this". The text field should not be able to be edited.

```
JTextField hammer = new JTextField(30);
hammer.setText("Can't touch this");
hammer.setEditable(false);
```

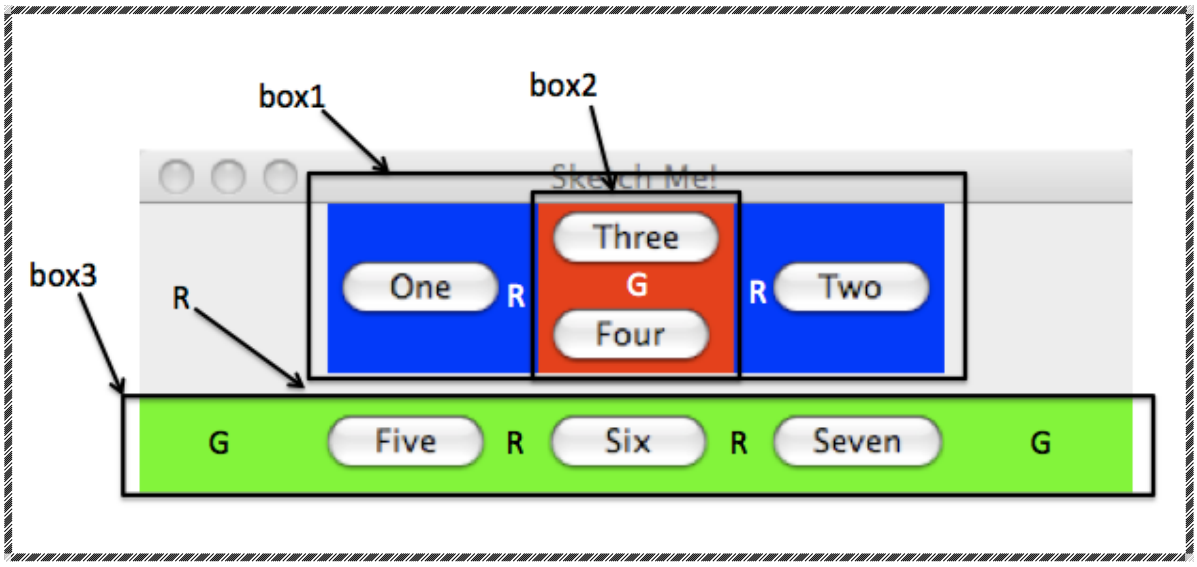
f. Create an editable `JComboBox` containing the entries Mathematics, Computer Science, Physics, Astronomy, Biology, and Chemistry.

```
String[] items = new String[] { "Mathematics", "Computer Science",
   "Physics", "Astronomy", "Biology", "Chemistry" };
JComboBox science = new JComboBox(items);
```

2. Sketch the GUI that would be created by the following code. Draw a box around each `JPanel` and label it with the associated variable name from the code. Also indicate the locations of all rigid areas with "R" and all glue with "G". Do not compile and run the code.

```
public class SketchGUI extends JFrame {

   public SketchGUI() {
      super("Sketch Me!");
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel mainPanel = new JPanel();
      this.add(mainPanel);
      mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));

      mainPanel.add(getBox1());
      mainPanel.add(Box.createRigidArea(new Dimension(0, 10)));
      mainPanel.add(getBox3());
      this.pack();
   }

   private JPanel getBox1() {
      JPanel box1 = new JPanel();
      box1.setBackground(Color.blue);
      box1.setLayout(new BoxLayout(box1, BoxLayout.X_AXIS));
      box1.add(new JButton("One"));
      box1.add(Box.createRigidArea(new Dimension(10, 0)));
      box1.add(getBox2());
      box1.add(Box.createRigidArea(new Dimension(10, 0)));
      box1.add(new JButton("Two"));
      return box1;
   }
```

```
   private JPanel getBox2() {
      JPanel box2 = new JPanel();
      box2.setLayout(new BoxLayout(box2, BoxLayout.Y_AXIS));
      box2.add(new JButton("Three"));
      box2.add(Box.createVerticalGlue());
      box2.add(new JButton("Four"));
      return box2;
   }

   private JPanel getBox3() {
      JPanel box3 = new JPanel();
      box3.setBackground(Color.green);
      box3.setLayout(new BoxLayout(box3, BoxLayout.X_AXIS));
      box3.add(Box.createHorizontalGlue());
      box3.add(new JButton("Five"));
      box3.add(Box.createRigidArea(new Dimension(15, 0)));
      box3.add(new JButton("Six"));
      box3.add(Box.createRigidArea(new Dimension(15, 0)));
      box3.add(new JButton("Seven"));
      box3.add(Box.createHorizontalGlue());
      return box3;
   }
}
```



3. Give the code for a class that creates the GUI shown below. Use the `BoxLayout` as we did in class. If the window is made taller, buttons "One" and "Two" remain at the top and bottom of the window and buttons, "Three", "Four" and "Five" remain one on top of the other, with 20 pixels between them and centered vertically. If the window is made wider the buttons should remain at the left edge of the window. You do not have to create any event handlers for the buttons.

```
public class BuildGUI extends JFrame {

   public BuildGUI() {
      super("Build Me!");
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JPanel mainPanel = new JPanel();
      mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.X_AXIS));
      this.add(mainPanel);

      mainPanel.add(getBox1());
      mainPanel.add(Box.createRigidArea(new Dimension(10, 0)));
      mainPanel.add(getBox2());
      mainPanel.add(Box.createHorizontalGlue());

      this.pack();
   }

   private JPanel getBox1() {
      JPanel box1 = new JPanel();
      box1.setLayout(new BoxLayout(box1, BoxLayout.Y_AXIS));

      box1.add(new JButton("One"));
      box1.add(Box.createVerticalGlue());
      box1.add(new JButton("Two"));

      return box1;
   }

   private JPanel getBox2() {
      JPanel box2 = new JPanel();
      box2.setLayout(new BoxLayout(box2, BoxLayout.Y_AXIS));

      box2.add(Box.createVerticalGlue());
      box2.add(new JButton("Three"));
      box2.add(Box.createRigidArea(new Dimension(0, 20)));
      box2.add(new JButton("Four"));
      box2.add(Box.createRigidArea(new Dimension(0, 20)));
      box2.add(new JButton("Five"));
      box2.add(Box.createVerticalGlue());

      return box2;
   }

   public static void main(String[] args) {
      BuildGUI bg = new BuildGUI();
      bg.setVisible(true);
   }
}
```

4. Modify the `MyFirstEventHandler` program from the `comp132.examples.gui` package in the `132SampleCode` project so that the GUI displays a `JSlider` instead of a `JButton`. Each time the slider is moved the message "Slider Changed to X" should be printed, where X is the new value of the slider. Note: The `JSlider` generates `changeEvents` that must be handled by a class that implements the `ChangeListener` interface. You can find information about using `JSliders` and creating `ChangeListeners` in the Visual Guide to Swing Components and also in the Java Doc for the `javax.swing` package.

```
public class MyFirstEventHandler extends JFrame {

   private JSlider mySlider;

   public MyFirstEventHandler() {
      super("Slider GUI");
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      Container mainPanel = new JPanel();
      mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
      this.add(mainPanel);

      mySlider = new JSlider();
      mainPanel.add(mySlider);

      MySliderListener msl = new MySliderListener();
      mySlider.addChangeListener(msl);

      this.pack();
   }

   public static void main(String[] args) {
      MyFirstEventHandler mfal = new MyFirstEventHandler();
      mfal.setVisible(true);
   }

   private class MySliderListener implements ChangeListener {
      public void stateChanged(ChangeEvent e) {
         System.out.println("New Value: " + mySlider.getValue());
      }
   }
}
```

5. Consider the following Observer/Observable classes:

```
class Sidewalk extends Observable {
   private int people;

   public Sidewalk() {
      people = 0;
   }

   public void addWalker() {
      people++;
      setChanged();
      notifyObservers();
   }

   public void removeWalker() {
      people--;
      setChanged();
      notifyObservers();
   }

   public int getWalkers() {
      return people;
   }
}

class Dog implements Observer {
   private Sidewalk mySidewalk;

   public Dog(Sidewalk sw) {
      mySidewalk = sw;
   }

   public void update(Observable o, Object arg) {
      int count=mySidewalk.getWalkers();
      for (int i=0; i<count; i++) {
         System.out.print("Bark ");
      }
      System.out.println();
   }
}
```

What output would be generated by the following snippet of code:

```
   Sidewalk sw = new Sidewalk();
   Dog d = new Dog(sw);
   sw.addObserver(d);
```

```
   sw.addWalker();     ->   Bark
   sw.addWalker();     ->   Bark Bark
   sw.removeWalker();  ->   Bark
   sw.addWalker();     ->   Bark Bark
   sw.addWalker();     ->   Bark Bark Bark
```