Building A Graphical User Interface (GUI)  
Using WindowBuilder from Eclipse

Objectives
At the completion of this lab you should be able to:

- Build a basic Graphical User Interface in Java using the WindowBuilder plugin for Eclipse.
- Refactor the code for the GUI so that the components for each panel are added by get methods which implement the singleton design pattern.
- Implement the Model View Controller (MVC) design pattern.

A GUI for a Calculator
The major goal of this lab is to develop a graphical user interface for a calculator that matches the calculator GUI that is given in Figure 1.

Use the material on using the Eclipse WindowBuilder plugin presented in class and the following outline to develop the graphical user interface for the calculator. Important points to remember are:

- The complex organization of the components of the calculator is achieved by nesting JPanels.
- How components, which include nested JPanels, are arranged in a panel is determined by the layout manager of the containing panel.
- Spacing within a JPanel can be modified by the use of Struts and Rigid Areas.

Creating the Project Structure
1. Create a new Java Project. Make sure and add your cscollections project on the Projects tab.

![Figure 1: GUI for Calculator](image-url)
2. Create three packages under the source folder. Name the packages as follows:
   - **ui** – This is the package that will contain the user interface class. (i.e. the View)
   - **calculator** – This is the package that will contain the application class.
   - **calcEngine** – This is the package that will contain your Expression class and calculator control class. (i.e. both the Model and the Control classes, respectively)

3. In the **ui** package create the View class by selecting a WindowBuilder Swing Designer JFrame class. Name this class appropriately.

4. Create an application class in the **calculator** package using the Java Class from the New menu. Do not have the system generate a main method in this class. Cut the main method from the class created in step 3 and past it into the application class. Add the imports necessary to resolve any resolution errors. Compile and run the **calculator** program. You should see an window without content or a caption on the screen. The minimize, maximize, and close buttons should work.

5. Refactor the code in your user interface class using the Refactor | Extract Method command. Your refactored code should look like the code specified in Figure 2.

```java
/**
 * Create the frame.
 */
public TestUI() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    getContentPane().setContentPane(getContentPane());
}

/**
 */
private JPanel getContentPane() {
    if (contentPane == null) {
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
    }
    return contentPane;
}
```

**Figure 2: Refactored Code**

**Adding the Panels**

1. The panel structure for the project is specified in Figure three below. Add each panel as directed.
2. Enter the WindowBuilder Designer by selecting the Design tab with your UI class open in the Eclipse editor.

3. Add a title to your JFrame window by selecting the `javax.swing.JFrame` component in the Components Pane and then entering “CS Calculator” in the title line of the Properties Pane.

4. Add the displayPanel to the contentPane by selecting JPanel from the Containers section of the Palette and dragging it to the North region of the contentPane in the window displayed on the right. Use the Properties Pane to set the variable name to `displayPanel`.

5. Refactor the code of your UI class so that the displayPanel is added by a method named `getDisplayPanel`. This method should implement the singleton design pattern.

6. As you did for the displayPanel add the `keyPad` panel to the content pane. Set its variable name to `keyPad` and add an EtchedBorder using the Properties Pane.

7. Check the code for the UI class to determine if the keyPad panel was added using a `getKeyPad` method. If not, refactor your code.

Add the Display

1. The first step in adding the display is to set the height of the `displayPanel` to 30 pixels. Click the Vertical Strut in the Struts and Springs section of the Palette and drop it into the `displayPanel` by clicking on `displayPanel` in the Components Pane. Set the height of the strut to 30 pixels by expanding Factory in the Properties Pane and changing arg0 from 20 to 30.

2. Add the display to the `displayPanel` selecting JTextField in the Components section of the Palette and clicking on `displayPanel` in the Components Pane.

3. Set the following properties of the JTextField as specified:
   - Variable – display
Building a GUI

- border – BevelBorder
- columns – 0
- editable – false
- preferred size – 230,20
- text – “”

4. Check your code and refactor it if necessary.

The directions from this point on outline what needs to be done. The process used should follow that used in the steps above. Each time you add a component to the GUI, remember to check the code and refactor if necessary.

Special Step
Stop at this point and add the line pack(); as the last line of the constructor for your ui class. This command causes each of the components to be resized to their preferred size and the layout for each container to be reapplied. In most cases when creating a complex Swing design it is necessary to add this line of code in order to get a Swing container (the JFrame in this case) to display correctly.

Adding the KeyPad
1. Note from Figure 3 the keyPad contains a VerticalBox and a JPanel named commandPanel. The Vertical Box will contain the numeric keys and the commandPanel will contain the op keys. The Vertical Box contains a Rigid Area for spacing that measures 20 by 20 pixels and a JPanel named numberKeyPanel.
2. Add the Vertical Box to keyPad.
3. Add the commandPanel to keyPad.
4. Add the RigidArea to the Vertical Box and set its dimensions to 20 by 20 pixels.
5. Add the numberKeyPanel the Vertical Box. Note that if the RigidArea and the panel are not in the right order, just drag the one you want first above the other in the Component Pane. Set the Layout property for numberKeyPanel to GridLayout and set the grid to have 3 columns and 4 rows. (Note you will not see the grid until you add the buttons.)
6. Add a Vertical Box to the commandPanel. To the Vertical Box add a JPanel named operatorPanel and a JButton.
7. Set the Layout property of operatorPanel to GridLayout and set the grid to have 2 columns and 4 rows.

Adding the Buttons
1. In order to add the buttons to the numberKeyPanel and the operatorPanel select JButton in the Palette and add the button to the JPanel by clicking on the appropriate JPanel in the Component Pane. Add all the buttons to a JPanel before using the Property Pane to change their name and the text on the button. The two missing buttons in the numberKeyPanel should be filled with JLabels whose text property is empty. Name your number and operator buttons with meaningful names like oneButton or plusButton.
2. As you rename all the buttons in a panel they should resize and the containing panels should fit side by side.
3. Play with making the = button fill the bottom row of the Vertical Box in the commandPanel.
4. The C key should clear the display and the CE key should clear the last character added.

Making the Keys Do Something
1. Each number key should add the digit on the key to the display. In order to do this we have to add an Action Listener to each key. An action listener may be added by double clicking on the key.
2. Double clicking on a key should take you to the Source in the Eclipse editor and you should see the code for a blank actionPerformed method in an anonymous ActionListener class. The completed code for my sevenButton is given in Figure 4. The ActionListener code is highlighted.

```java
private JButton getSevenButton() {
    if (sevenButton == null) {
        sevenButton = new JButton("7");
        sevenButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                addDigit2Disp('7');
            }
        });
    }
    return sevenButton;
}
```

Figure 4: Action Listener

The code for addDigit2Display is given in Figure 5. Note that the text property of the display is accessed by call display.text().

```java
public void addDigit2Disp(char num) {
    StringBuffer newExp = new StringBuffer(display.getText());
    newExp.append(num);
    display.setText(newExp.toString());
}
```

Figure 5: Code to Add a Digit to the Display

Finishing Up
After you have each of the key components added, change the size of the JFrame by altering its bounds so that they are 260 x 220 pixels.