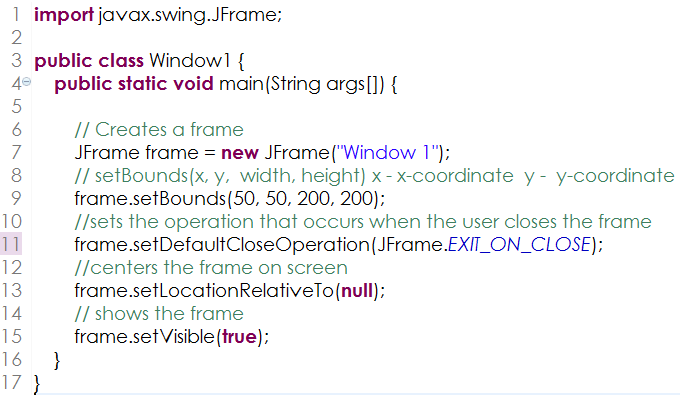
# Swing Component

**Swing** is the primary Java GUI widget toolkit. It is part of Oracle’s Java Foundation Classes (JFC) – an API for providing Graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components. Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

# Creating a Swing Window

Create a new project in Eclipse. Create **Window1.java** and key in the following codes. What do you see when you run the program?



Try the following change one by one then run the program and observe the output.

Enter the changes observed below . (Remember to restore each change before doing the next change, hence comment the old code and modify to the changes mentioned.)

1. Change Line 7 to JFrame frame = new JFrame(“”);

Observation 🡪 the title “Windows 1” disappear

1. Comment Line 11 and then try to close the window.

Observation 🡪 can’t close

1. Comment Line 13 and observe the window.

Observation 🡪 the frame stick to the side

1. Change Line 9 to frame.setBounds(50, 50, 200, 100)

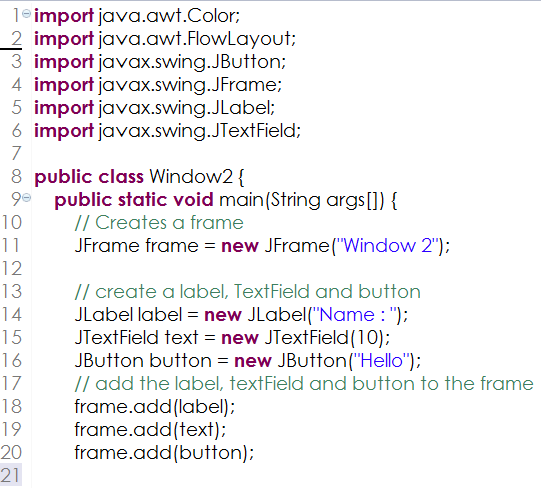
Observation 🡪 the width become smaller

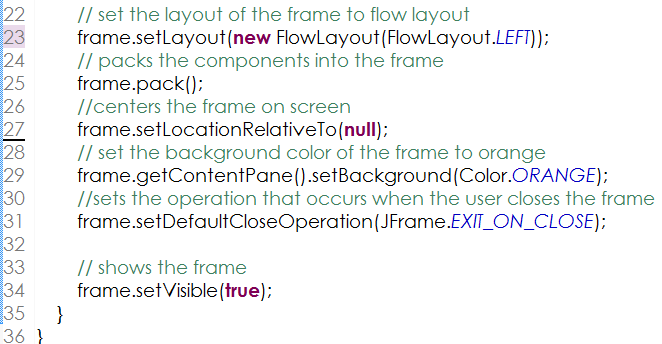
1. Comment Line 15 and observe the window.

Observation 🡪 nothing will be shown

# Adding GUI components

Create **Window2.java** that adds the button, textfield and label components to the frame. Key in the following codes to create Window2.





Based on your observation of the code, answer the following questions :

1. Write down the line number that creates a **label** component.

14

1. Write down the line number that creates a **text field** component.

15

1. Write down the line number that creates a **button** component.

16

1. Write down the line number(s) that adds the **label**, **textfield** and **button** components to the frame.

18-20

1. Comment Line 23 and write down your observation ? Elaborate on the importance of specifying Flowlayout.

arranges components in a directional flow

1. Comment Line 25 and write down your observation. What is purpose of this line based on your observation.

layout managers and components to size themselves optimally

1. Comment Line 25 and Line 27 and insert the line frame.setBounds(50, 50, 300, 100). Write down your observation.

The panel is messed up

1. Modify the value of the width in the setBounds method to 200 and Write down your observation. Compare your observation of the window before you make this change. Elaborate on how this behavior is relate to the Flow Layout setting of the Frame.

.The width become smaller

# FLOW LAYOUT

A flow layout arranges components in a **directional flow**, much like lines of text in a paragraph.



The flow direction is determined by the container's *componentOrientation property* and may be one of two values:

* + ComponentOrientation.LEFT\_TO\_RIGHT
  + ComponentOrientation.RIGHT\_TO\_LEFT

Flow layouts are typically used to arrange buttons in a panel. It arranges buttons horizontally until no more buttons fit on the same line. The line alignment is determined by the *align property*. The possible values are:

* + [LEFT](http://docs.oracle.com/javase/7/docs/api/java/awt/FlowLayout.html#LEFT)
  + [RIGHT](http://docs.oracle.com/javase/7/docs/api/java/awt/FlowLayout.html#RIGHT)
  + [CENTER](http://docs.oracle.com/javase/7/docs/api/java/awt/FlowLayout.html#CENTER)
  + [LEADING](http://docs.oracle.com/javase/7/docs/api/java/awt/FlowLayout.html#LEADING)
  + [TRAILING](http://docs.oracle.com/javase/7/docs/api/java/awt/FlowLayout.html#TRAILING)

1. Most Swing components’ names start with "J". The reference is given in the table below.

|  |  |
| --- | --- |
| Component | Reference |
| JFrame | <http://java.sun.com/javase/6/docs/api/javax/swing/JFrame.html> |
| JLabel | <http://java.sun.com/javase/6/docs/api/javax/swing/JLabel.html> |
| JTextField | <http://java.sun.com/javase/6/docs/api/javax/swing/JTextField.html> |
| JButton | <http://java.sun.com/javase/6/docs/api/javax/swing/JButton.html> |
|  |  |

1. Write down the method for each of the Components

|  |  |  |
| --- | --- | --- |
| Component | Description | Method / API |
| JFrame | Gets the title of the frame. The title that is displayed on the frame's border. | getTitle();(returns a string) |
| JLabel | The method that obtains the text string that the label displays. | getText(); |
| JTextField | The method that Sets the text of the TextComponent to the specified text | setText(value); |
| JButton | Returns the state of the button. True if the button is selected, false if it's not. | isSelected(); return boolean |
|  |  |  |

# Adding Event Handlers

<http://java.sun.com/docs/books/tutorial/uiswing/events/actionlistener.html>

An action listener is implemented to define what should be done when an user performs certain operation.

An action event occurs, whenever an action is performed by the user.

*Examples: When the user clicks a button, chooses a menu item, presses Enter in a text field*.

The result is that an actionPerformed message is sent to all action listeners that are registered on the relevant component.

To write an Action Listener, follow the steps given below:

1.Declare an event handler class and specify that the class either implements an ActionListener interface or extends a class that implements an ActionListener interface. For example:

public class MyClass implements ActionListener {

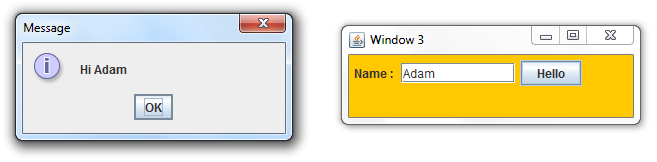
2.Register an instance of the event handler class as a listener on one or more components. For *example: someComponent.addActionListener(instanceOfMyClass);*

3.Include code that implements the methods in listener interface. For example:

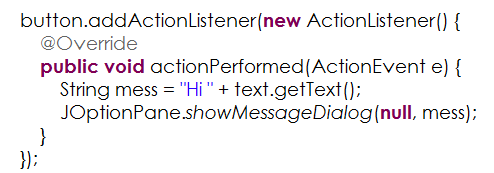
public void actionPerformed(ActionEvent e) {

...//code that reacts to the action... }

1. Create a **Window3.java** similar to the window2 so that when the user enters a text “Adam” in the text box and when the user clicks the Hello button, a dialog box as shown below appears. The text “class” is obtained from the text box in the window.



* The code to display the dialog box is **JOptionPane.showMessageDialog(null, text);**
* The action listener code is as shown below.



* You need to import the files

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

* You need to re-declare the variable text as final JTextField text, since the variable text is used inside the actionPerformed method.

**Final** is a keyword or reserved word in java and can be applied to member variables, methods, class and local variables in Java. Once you make a reference final you are not allowed to change that reference and compiler will verify this and raise compilation error if you try to re-initialized final variables in java.

1. Observe what happens if final is not added ? Explain the importance of the final keyword.

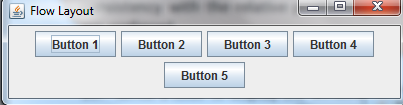
The text can’t be taken out of the class. The importance of the final keyword is to make sure that the variable can be passed out of its classes to other classes’ usages.

# Containers and Layout Components

<http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html>

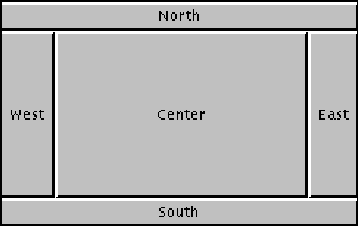
A layout manager manages the placement of components / controls in a container. Resize the windows and take note of the size and the placement of the buttons when you do so. Answer the following questions

1. Write a code to display the window with five buttons as shown below using Flow Layout



|  |
| --- |
| **import** javax.swing.JFrame;  **import** javax.swing.JButton;  **import** java.awt.Color;  **import** java.awt.FlowLayout;  **public** **class** Window2 {  **public** **static** **void** main(String args[]) {  JFrame frame = **new** JFrame("Window 1"); // class  // sets the x,y,width and height of the frame.  frame.setBounds(50, 50, 200, 200);  // create a button (class)  JButton button = **new** JButton("Button 1");  JButton button2 = **new** JButton("Button 2");  JButton button3 = **new** JButton("Button 3");  JButton button4 = **new** JButton("Button 4");  JButton button5 = **new** JButton("Button 5");  frame.add(button);  frame.add(button2);  frame.add(button3);  frame.add(button4);  frame.add(button5);  frame.setLayout(**new** FlowLayout(FlowLayout.***TRAILING***));  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  // centers the frame on the screen  frame.setLocationRelativeTo(**null**);  frame.getContentPane().setBackground(Color.***WHITE***);  //  frame.pack();  // shows the frame  frame.setVisible(**true**);  }  } |

A border layout lays out a container, arranging and resizing its components to fit in five regions: **north, south, east, west, and center**. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER.



When adding a component to a container with a border layout, use one of these five constants, for example:

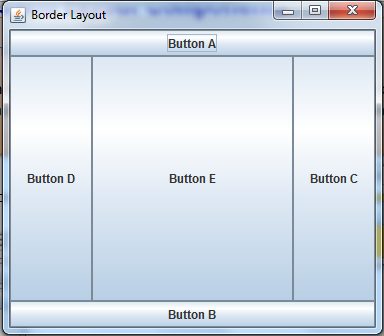
JFrame p = new Frame();

**p.setLayout(new BorderLayout());**

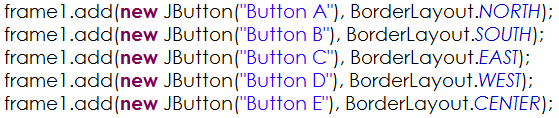
p.add(new Button("Okay"), *BorderLayout.SOUTH*);

Note: These are equivalent to PAGE\_START, PAGE\_END, LINE\_START and LINE\_END respectively. For consistency with the relative positioning constants used by other components, the latter constants are preferred.

1. Write a code to display the window with five buttons as shown below using Border Layout



The code to add the buttons to the appropriate regions is given below:



|  |
| --- |
| **import** java.awt.BorderLayout;  **import** java.awt.Button;  **import** java.awt.Color;  **import** javax.swing.JFrame;  **public** **class** Window3 {  **public** **static** **void** main(String args[]) {  JFrame p = **new** JFrame("Window 1");  p.setLayout(**new** BorderLayout());  p.add(**new** Button("Button A"), BorderLayout.***NORTH***);  p.add(**new** Button("Button B"), BorderLayout.***SOUTH***);  p.add(**new** Button("Button C"), BorderLayout.***EAST***);  p.add(**new** Button("Button D"), BorderLayout.***WEST***);  p.add(**new** Button("Button E"), BorderLayout.***CENTER***);  p.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  // centers the frame on the screen  p.setLocationRelativeTo(**null**);  p.getContentPane().setBackground(Color.***WHITE***);  p.pack();  // shows the frame  p.setVisible(**true**);  }  } |

1. Which Layout Manager does not resize its components?

The content pane of a JFrame is the main container for components. Swing allows other containers to be created using the JPanel API. These containers can be used to hold other components and themselves be added to the content pane.

The **JPanel class** provides general-purpose containers for lightweight components. By default, panels do not add colors to anything except their own background; however, you can easily add borders to them and otherwise customize them accordingly.

1. With the help of JPanel create a Combined Layout as shown below. Which Layout Manager does not resize its components?

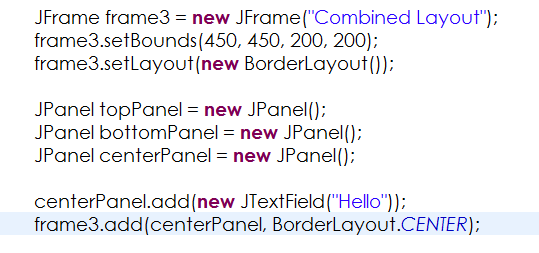
# 

Bottom panel

Center panel

Top panel

The code below shows the 3 panels and the “Hello” textfield being added to the centerPanel and the centerPanel being added to the frame3



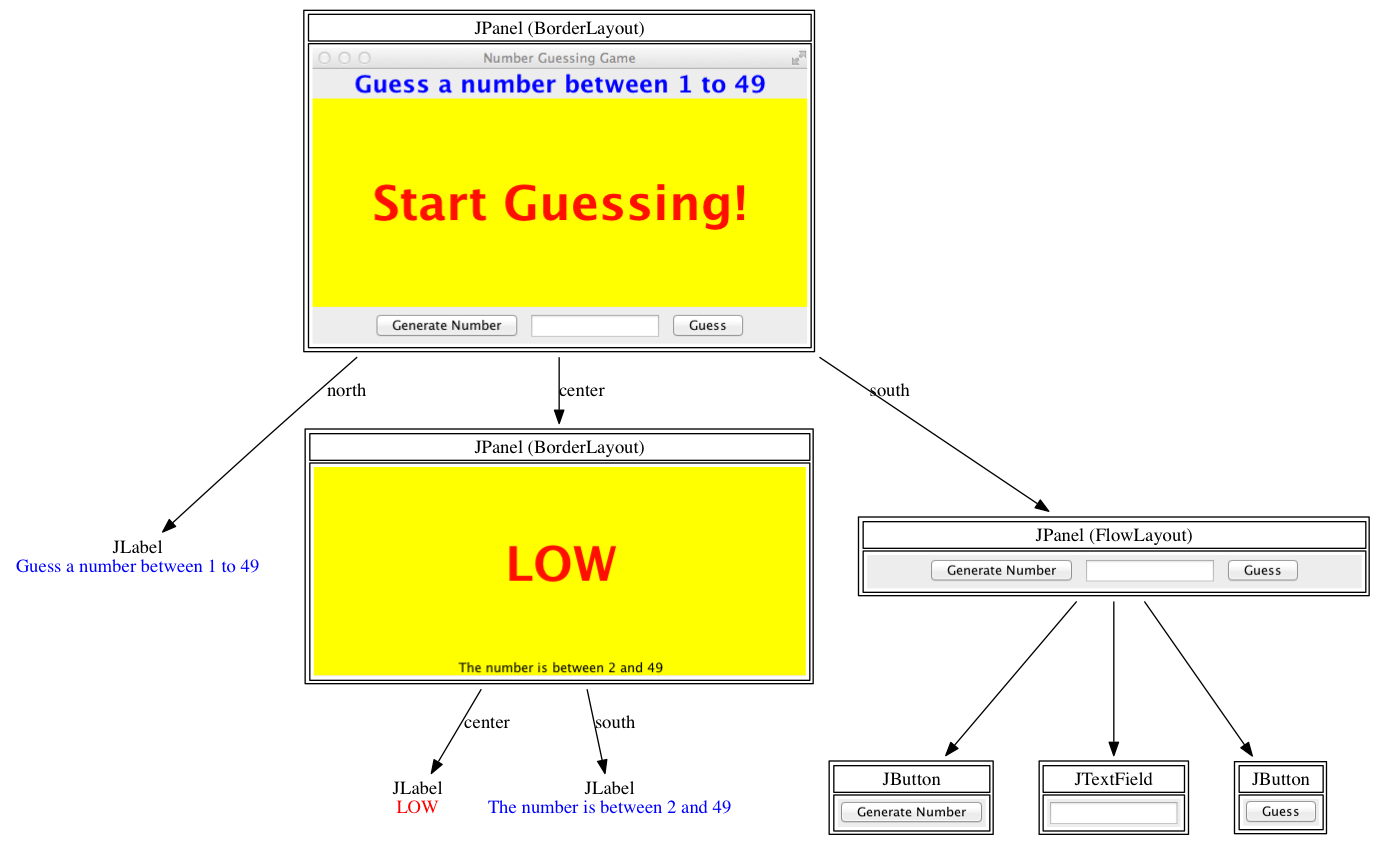
|  |
| --- |
| **import** java.awt.BorderLayout;  **import** java.awt.Color;  **import** java.awt.FlowLayout;  **import** javax.swing.JButton;  **import** javax.swing.JFrame;  **import** javax.swing.JLabel;  **import** javax.swing.JPanel;  **import** javax.swing.JTextField;  **public** **class** Window4 {  **public** **static** **void** main(String args[]) {  JFrame frame = **new** JFrame("Combined Layout");  frame.setBounds(50, 50, 300, 300);  frame.setLayout(**new** BorderLayout());  JPanel topPanel = **new** JPanel();  topPanel.setLayout(**new** FlowLayout());  JLabel label1 = **new** JLabel("Label A");  JLabel label2 = **new** JLabel("Label B");  JLabel label3 = **new** JLabel("Label C");  topPanel.add(label1);  topPanel.add(label2);  topPanel.add(label3);  frame.add(topPanel, BorderLayout.***NORTH***);  JPanel bottomPanel = **new** JPanel();  bottomPanel.setLayout(**new** FlowLayout());  JButton buttonA = **new** JButton("Button A");  JButton buttonB = **new** JButton("Button B");  bottomPanel.add(buttonA);  bottomPanel.add(buttonB);  frame.add(bottomPanel, BorderLayout.***SOUTH***);  JPanel centerPanel = **new** JPanel();  centerPanel.setLayout(**new** FlowLayout());  JTextField text = **new** JTextField("Hello");  centerPanel.add(text);  frame.add(centerPanel, BorderLayout.***CENTER***);  JButton buttonC = **new** JButton("Button C");  JButton buttonD = **new** JButton("Button D");  frame.add(buttonC, BorderLayout.***EAST***);  frame.add(buttonD, BorderLayout.***WEST***);  // centerPanel.add(new JTextField("Hello"));  // frame.add(centerPanel, BorderLayout.CENTER);  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  // centers the frame on the screen  frame.setLocationRelativeTo(**null**);  frame.getContentPane().setBackground(Color.***WHITE***);  //  frame.pack();  // shows the frame  frame.setVisible(**true**);  }  } |

# Analysing the Problem

In this section we will analyse the solution to the problem statement. We will first identify the Inputs to the program, process the inputs and then provide the result.

## Creating the GUI for the Problem Statement

The following diagram depicts the GUI structure for the Number Guessing Game application.



Create the GUI structure for the Number Guessing Game application.

## Creating the Game

It would be useful to start off by creating the game separately from the GUI.

1. Create a class and call the method that creates the GUI
2. Write the method to Create the GUI
3. Event 🡪 When Generate Number button is clicked – The action required is to allow the computer guess a number. So randomly create a secret number and store it into a variable whenever this button is pressed.

// To create a random number from 1 to 10 in java  
int secret = new java.util.Random().nextInt(10);

1. Event 🡪 When Guess button is clicked. Compare the secret number with given number in the test box .
2. Based on the comparison result, display the lable with a message “LOW” or “HIGH”
3. Continue to compare until the result is equal, each time the user guess’s a number.

*End of Worksheet*