

User Interface Components with Swing

Concept of AWT:

Java AWT (Abstract Windowing Toolkit) is an API to develop GUI or window-based application in java. Java AWT components are platform-dependent i.e., components are displayed according to the view of operating system. AWT calls Operating Systems subroutine for creating components such as textbox, checkbox, button etc. For example if we are creating a textbox in AWT that means we are actually asking OS to create a textbox for us. This is the reason why AWT components look different on different operating systems. An application build on AWT would look like a windows application when it runs on Windows, but the same application would look like a Mac application when runs on Mac OS.

Java AWT vs. Java Swing: [Imp],

Java AWT	Java Swing
i) AWT components are platform-dependent.	i) Java Swing components are platform independent.
ii) AWT components are heavyweight.	ii) Swing components are lightweight.
iii) AWT does not support pluggable look and feel.	iii) Swing supports pluggable look and feel.
iv) AWT does not follow MVC.	iv) Swing follows MVC.
v) AWT provides less components than swing.	v) Swing provides more powerful components such as tables, lists, scrollpanes etc.

Java Applets:

→ Applets are small Java applications which can be accessed on an Internet server, transported over the internet, and can be installed and run automatically as a part of a web document.

→ The applet can create a GUI and it has restricted access to resources so that complicated computations can be carried out without adding the danger of viruses and violating data integrity.

→ Any java applet is a class that extends the class of `java.applet.Applet`.

→ There is no `main()` method in an Applet class. The JVM can operate an applet application using either a web-browser plug-in or a distinct runtime environment.

Example of Simple Applet:

```
import java.awt.*;
import java.applet.*;
public class Simple extends Applet
{
    public void paint (Graphics g) {
        g.drawString("A simple Applet", 20, 20);
    }
}
```

Benefits of Applets:

→ As it operates on the client side, it requires much less response time.

→ Any browser that has JVM operating in it can operate it.

Applet Life Cycle:

These 4 methods are overridden by most applets. These 4 methods are the lifecycle of the Applet.

init(): The first technique to be called is `init()`. This is where we initialize the variable. This is called only once during applet runtime.

start(): Method `start()` is called after `init()`. This technique is called after it has been stopped to restart an applet.

stop(): Method `stop()` is called to suspend threads that do not need to operate when the applet is not noticeable.

destroy(): The `destroy()` method is called if we need to remove our applet from memory entirely.

#Swing Class Hierarchy:

- Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JMenu, JColorChooser etc.

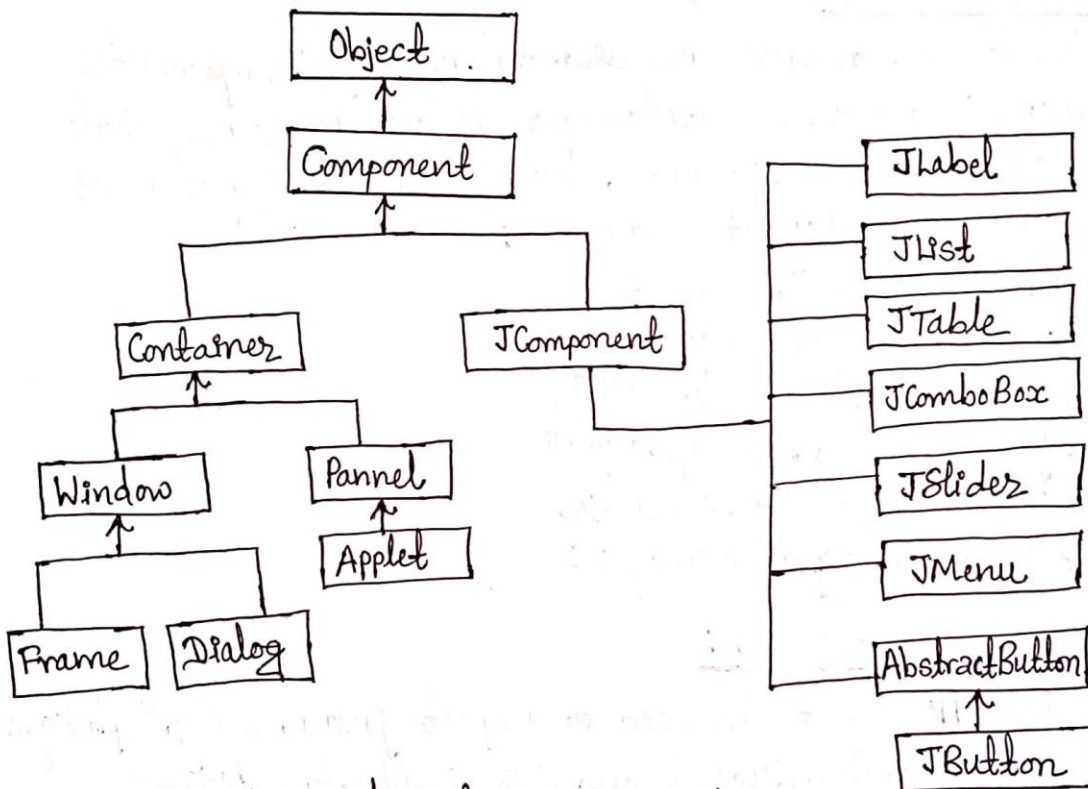


Fig: Hierarchy of Java Swing Classes.

#Components and Containers:

Components: At the top of AWT hierarchy is the Component class, Component is an abstract class that encapsulates all of the attributes of a component. All user interface elements that are displayed on screen and that interact with the user are subclasses of component. It defines over a hundreds of public methods that are responsible for managing events. Component object is responsible for remembering the current foreground and background colors, currently selected text font, and current text alignment.

Containers: The Container class is a subclass of Component. It has additional features that allow other Component objects to be placed within it. Other container objects can also be stored inside of a Container since they are themselves instances of Component. Thus, allowing multilevel containment system. A container is responsible for positioning any component placed on it. It does this through the use of various layout managers. Default layout is present in each container which can be overridden using setLayout method.

Layout Management: [Imp]

कुछ सवाल Layout को देन सके 5 marks का लागि / 10 marks का लागि सवाल describe any 2 or 3 type केर आठिन सके

The layoutManagers are used to arrange components in a particular manner. layoutManager is an interface that is implemented by all the classes. There are following classes that represents the layout managers:

- java.awt.BorderLayout
- java.awt.GridLayout
- java.awt.GridBagLayout
- javax.swing.GroupLayout
- Using No layout Managers
- Custom layout Managers

No layout (Absolute Positioning):

For No layout we need to import javax.swing.* package library to access JFrame, JLabel, and JTextField class. User Interface of having the LookAndFeel.Decorated UI is set to true as below:

```
JFrame.setDefaultLookAndFeelDecorated(true);
```

Then we need to initialize variables in our Main, variable frame as JFrame, label as JLabel, and textField as JTextField.

```
JLabel label = new JLabel("Name:");
```

```
JTextField textField = new JTextField("Hello", 15);
```

To set the layout without a layout we will use null keyword in the setLayout method of the frame.

```
frame.getContentPane().setLayout(null);
```

Now we will proceed with the absolute positioning of our components with the use of the setBounds method.

```
label.setBounds(20, 20, 200, 40);
```


And then add method is used to the components.

```

frame.getContentPane().add(label);
frame.getContentPane().add(textField);

```

lastly, set it's size, visibility to true, and having it's close operation.

```

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(400,200);
frame.setVisible(true);

```

Flow layout:

The Java Flowlayout class is used to arrange the components in a line, one after another (in a flow). Fields of Flowlayout class are left, right, center, leading, trailing etc. and specified as;

```
public static final int CENTER
```

only values are changed here
LEFT, RIGHT etc.

Flowlayout(int align, int hgap, int vgap): creates a flow layout with the given alignment, horizontal gap, and vertical gap.

Example:

```

import java.awt.*;
import java.swing.*;
public class FlowlayoutExample {
    JFrame frameObj;
    //Constructor
    FlowlayoutExample() {
        //creating a frame object
        frameObj = new JFrame();
        //creating the button
        JButton b1 = new JButton("1");
        JButton b2 = new JButton("2");
        //adding buttons to frame
        frameObj.add(b1); frameObj.add(b2);
        //parameter less constructor used so, alignment is center, and hgap and vgap
        // is 5 units.
        frameObj.setLayout(new Flowlayout());
        frameObj.setSize(300,300); frameObj.setVisible(true);
    }
    public static void main (String args[]) {
        new FlowlayoutExample();
    }
}

```

for smaller code only two buttons are created and added we can create and add multiple, like 6-7 for better view

all examples of layout are almost similar only this line of setLayout may change

BorderLayout:

A border layout places components in upto five areas: top, bottom, left, right, and center. All extra space is placed in the center area. Tool bars that are created using JToolBar must be created with a BorderLayout container, if we want to be able to drag and drop the bars away from their starting positions.

Constructor or Method

BorderLayout(int horizontalGap,
int verticalGap)

setHgap(int)

setVgap(int)

Purpose

Defines a border layout with specified gaps between components.

Sets the horizontal gap between components.
Sets the vertical gap between components.

Example:

```
import java.awt.*;  
import java.swing.*;  
public class Border {  
    JFrame f;  
    Border() { f = new JFrame();  
              JButton b1 = new JButton("Center");  
              f.add(b1, BorderLayout.CENTER);  
              f.setSize(300, 300);  
              f.setVisible(true);  
            }  
    public static void main(String args[]) {  
        new Border();  
    }  
}
```

create and add multiple buttons

for detail view notes alternative provided in drive

GridLayout:

GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns. The constructors used are:

GridLayout(int rows, int cols)

GridLayout(int rows, int cols, int hgap, int vgap)

Example: import java.awt.*;
import java.swing.*;


```

public class MyGridLayout {
    JFrame f;
    MyGridLayout() {
        f = new JFrame();
        JButton b1 = new JButton("1");
        f.add(b1);
        f.setLayout(new GridLayout(3,3));
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String args[]) {
        new MyGridLayout();
    }
}

```

→ Create multiple buttons 7-8

→ add multiple buttons to frame

→ Similar example Only this line changes setLayout line.

Gridbag layout:

Gridbag layout is a sophisticated, flexible layout manager. It aligns components to span more than one cell. The rows in the grid can have different heights, and grid columns can have different widths.

```

JPanel pane = new JPanel(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
// For each component to be added to this container:
// Create the component...
// Set instance variables in the GridBagConstraints instance...
pane.add(theComponent, c);

```

Group layout:

Group layout is a layout manager that was developed for use by GUI builder tools, but it can also be used manually. Group layout works with the horizontal and vertical layouts separately. The layout is defined for each dimension separately.

```

GroupLayout layout = new GroupLayout(pane);
pane.setLayout(layout);
// We specify automatic gap insertion:
layout.setAutoCreateGaps(true);
layout.setAutoCreateContainerGaps(true);

```


GUI Controls:

1) Text Input: Text input contains Text Fields, Password Fields, Text areas, Scroll pane, label and Labeling Components.

Text Field: A text field is a basic text control that enables the user to type a small amount of text. When the user indicates that text entry is complete (usually by pressing Enter), the text field fires an action event.

Syntax: `textField = new JTextField(20);`

TextArea: To obtain long texts like paragraph which are more than one line of input, we use text area. Text Field only takes single line of text, while TextArea takes multiple lines of text. It also allows user to edit the text.

Syntax: `textArea = new JTextArea(5, 20);`

`JScrollPane = new JScrollPane(textArea);`

`textArea.setEditable(false);`

→ for getting scroll option on textArea

→ set true if we want editable

Password Field: A password field provides specialized text fields for password entry. For security reasons, a password field does not show characters that the user types, instead displays different from typed such as an asterisk (*). A password field stores its value as an array of characters, rather than a string.

Syntax: `passwordField = new JPasswordField(10);`

`passwordField.setActionCommand("OK");`

`passwordField.addActionListener(this);`

ScrollPane: Scroll Pane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component.

Syntax: `JScrollPane scrollableTextArea = new JScrollPane(textArea);`

Label: Label is used to display a single line of read-only text. The text can be changed by a programmer but a user cannot edit it directly. To create a label, we need to create the object of Label class.

Syntax: `l1 = new Label("First");`

`l1.setBounds(50, 100, 100, 50);`

2) Choice Components:

Choice components include Check Boxes, Radio Buttons, Borders, Combo Boxes, Sliders.

Check Boxes: Check box is used to turn an option true or false. Clicking on a check box changes its state from true to false or false to true.

```
Syntax: JCheckbox checkbox = new JCheckbox("Married");
checkbox.setBounds(100, 100, 50, 50);
```

Radio Button: It is used to choose one option from multiple options. It is widely used in exam systems or quiz. It should be added in ButtonGroup to select one radio button only.

```
Syntax: JRadioButton r1 = new JRadioButton("Male");
```

Borders: Border component is used to place border in our components.

```
Syntax: Border bdr = new LineBorder(Color.ORANGE, 4, true);
```

Combo Box: The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of menu.

```
Syntax: String country[] = {"Nepal", "India", "Aus", "U.S.A"};
JComboBox cb = new JComboBox(country);
```

Sliders: The Java JSlider class is used to create the slider. By using JSlider, a user can select a value from a specific range.

```
Syntax: JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
```

#Menus:

```
// Create the menu bar.
MenuBar = new JMenuBar();
```

```
// Build the first menu.
menu = new JMenu("A Menu");
MenuBar.add(menu);
```

```
// A group of JMenuItem
menuItem = new JMenuItem("Both text and icon", new ImageIcon("images/myImg.jpg"));
menu.addSeparator();
```


// A group of JMenuItem Syntax:

```
menuItem = new JMenuItem("A text-only menu item", KeyEvent.VK_T);
```

Java JMenuItem and JMenu Example:

```
import javax.swing.*;
class MenuExample {
    JMenu menu, submenu;
    JMenuItem i1, i2, i3, i4;
    MenuExample() {
        JFrame f = new JFrame("Menu and JMenuItem Example");
        JMenuBar mb = new JMenuBar();
        menu = new JMenu("Menu");
        submenu = new JMenu("Sub Menu");
        i1 = new JMenuItem("Item 1");
        i2 = new JMenuItem("Item 2");
        i3 = new JMenuItem("Item 3");
        i4 = new JMenuItem("Item 4");
        submenu.add(i1); submenu.add(i4);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[]) {
        new MenuExample();
    }
}
```

⊗ Icons in Menu Items: Icons in menu items can be added as:

```
Icon myIcon = new ImageIcon("Resources/myIcon.png");
```

⊗ Enabling and Disabling Menu Items:

// For disabling

```
menuItem1.setEnabled(false);
```

// For enabling

```
menuItem1.setEnabled(true);
```


⊗ Tool tips:

We can create a tooltip for any JComponent with `setToolTipText()` method. For example, to add tooltip to PasswordField we do as following:

```
field.setToolTipText("Enter your password");
```

⊗ Check Box in MenuItem: & RadioButton in MenuItem:

JCheckBoxMenuItem class represents checkbox which can be included on a menu. A CheckBoxMenuItem can have text or graphic icon or both, associated with it. MenuItem can be selected or deselected. MenuItems can be configured and controlled by actions.

Example: `JMenu fileMenu = new JMenu("file");`

```
JCheckBoxMenuItem caseMenuItem = new JCheckBoxMenuItem("Option 1");
fileMenu.add(caseMenuItem);
```

Similarly JRadioButtonMenuItem class represents RadioButton which can be included on a menu similarly as we did for checkbox.

⊗ Pop-up Menu:

PopupMenu can be dynamically popped up at specific position within a component. It inherits the Menu class.

AWT PopupMenu class declaration

```
public class PopupMenu extends Menu implements MenuContainer, Accessible
```

Example:

```
import java.awt.*;
import java.awt.event.*;
class PopupMenuExample {
    PopupMenuExample() {
        final Frame f = new Frame("PopupMenu Example");
        final PopupMenu popupmenu = new PopupMenu("Edit");
        MenuItem copy = new MenuItem("Copy");
        copy.setActionCommand("Copy");
        MenuItem paste = new MenuItem("Paste");
        copy.setActionCommand("Paste");
        popupmenu.add(copy);
        popupmenu.add(paste);
    }
}
```



```

f.add (popupmenu);
f.setSize (400,400);
f.setLayout (null);
f.setVisible (true);
public static void main (String args[]) {
    new PopupMenuExample ();
}
}
}

```

* Keyboard Mnemonics and Accelerators:

A mnemonic is a key-press that opens a JMenu or selects a MenuItem when the menu is opened. An accelerator is a key-press that selects an option within the menu without it ever being open. The purpose of all this is to let people who really know the program to access functions quickly, and let people that don't use a mouse (some don't) to access the Menubar.

Example: JMenu menu = new JMenu ("Menu"); // Create Menu
 menu.setMnemonic ('M'); // Set Mnemonic
 JMenuItem menuItem = new JMenuItem ("Item");
 menuItem.setAccelerator (KeyStroke.getKeyStroke (KeyEvent.VK_I,
 KeyEvent.SHIFT_MASK));
 menu.add (menuItem);

* Toolbars:

JToolBar container allows us to group other components, usually buttons with icons in a row or column. JToolBar provides a component which is useful for displaying commonly used actions or controls.

Example: JToolBar toolbar = new JToolBar ();
 toolbar.setRollover (true);
 toolbar.add (new JButton ("Edit"));
 toolbar.add (new JComboBox (new String [] { "option1", "option2" }));
 Container contentPane = myframe.getContentPane ();
 contentPane.add (toolbar, BorderLayout.NORTH);
~~contentPane.add~~
 myframe.setSize (450, 250);
 myframe.setVisible (true);

* Option Dialogs: Creating Dialogs:

The `JOptionPane` class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The `JOptionPane` class inherits `JComponent` class.

JOptionPane class declaration:

```
public class JOptionPane extends JComponent implements Accessible
```

Example: showMessageDialog():

```
import javax.swing.*;
public class JOptionPaneExample {
    JFrame f;
    JOptionPaneExample() {
        f = new JFrame();
        JOptionPane.showMessageDialog(f, "Hello");
    }
    public static void main(String args[]) {
        new JOptionPaneExample();
    }
}
```

Similarly for showInputDialog():

```
String name = JOptionPane.showInputDialog(f, "Enter Name");
```

Similarly for showConfirmDialog():

```
f = new JFrame();
f.addWindowListener(this);
f.setSize(300, 300);
f.setLayout(null);
f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
f.setVisible(true);
public void windowClosing(WindowEvent e) {
    int a = JOptionPane.showConfirmDialog(f, "Are you sure?");
    if (a == JOptionPane.YES_OPTION) {
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```


⊗ File Choosers & Color Choosers:

The object of JFileChooser class represents a dialog window from which the user can select file. Similarly The object of JColorChooser class represents a dialog window from ~~the user~~ which the user can select color. They both inherit from JComponent class.

Example: File Choosers:

```
import javax.swing.JFileChooser;
import java.io.File;
public class FileChooserExample() {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setCurrentDirectory(new File(System.getProperty("user.home")));
    int result = fileChooser.showOpenDialog(parent);
    if (result == JFileChooser.APPROVE_OPTION) {
        //user selects a file
    }
    File selectedFile = fileChooser.getSelectedFile();
    System.out.println("Selected file:" + selectedFile.getAbsolutePath());
}
int public static void main (String args[]) {
    new FileChooserExample();
}
```

Example: Color Choosers:

```
import javax.swing.*;
public class ColorChooserExample extends JFrame implements ActionListener {
    JButton b;
    Container c;
    ColorChooserExample() {
        c.getContentPane();
        c.setLayout(new FlowLayout());
        b = new JButton("color");
        b.addActionListener(this);
        c.add(b);
    }
}
```



```

public void actionPerformed (ActionEvent e) {
    Color initialColor = Color.RED;
    Color color = JColorChooser.showDialog(this, "select a color", initialColor);
    c.setBackground (color);
}

public static void main (String args[]) {
    ColorChooserExample ch = new ColorChooserExample();
    ch.setSize (400, 400);
    ch.setVisible (true);
    ch.setDefaultCloseOperation (EXIT_ON_CLOSE);
}
}

```

* Internal Frames:

JInternalFrame is a part of Java Swing. JInternalFrame is a container that provides many features of a frame which includes displaying title, opening, closing, resizing, support for menu bars etc.

Example:

```

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

class solution extends JFrame {
    static JFrame f; // frame
    static JLabel l; // label to display text.

    public static void main (String args[]) {
        f = new JFrame ("frame"); // Create new frame
        JInternalFrame in = new JInternalFrame();
        in.setTitle ("Internal frame");

        JButton b = new JButton ("button");
        l = new JLabel ("This is a JInternal Frame");
        JPanel p = new JPanel();
        // add label and button to pannel
        p.add (l); p.add (b);
        in.setVisible (true);
        in.add (p);
        f.add (in);
        f.setSize (300, 300);
        f.show ();
    }
}

```


⊗ Advance Swing Components:

1) List: A JList presents the user with a group of items, displayed in one or more columns, to choose from. Lists can have many items, so they are often put in scroll panes.

```
list = new JList(data);  
list.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);  
list.setLayoutOrientation(JList.HORIZONTAL_WRAP);  
list.setVisibleRowCount(-1);
```

2) Trees: With the JTree class, we can display hierarchical data. A JTree object does not actually contain our data, it simply provides a view of data. Like any non-trivial Swing component, the tree gets data by querying its data model.

```
private JTree tree;  
...  
public TreeDemo() {  
    DefaultMutableTreeNode top = new DefaultMutableTreeNode("Java");  
    createNodes(top);  
    tree = new JTree(top);  
    ...  
}
```

3) Tables: The JTable class is used to display data in tabular form. It is composed of rows and columns.

```
Example: import javax.swing.*;  
public class TableExample {  
    JFrame f;  
    TableExample() {  
        f = new JFrame();  
        String data[][] = { { "101", "Amit", "67000" },  
                             { "102", "Jag", "65000" } };  
        String column[] = { "ID", "Name", "SALARY" };  
        JTable jt = new JTable(data, column);  
        jt.setBounds(30, 40, 200, 300);  
        JScrollPane sp = new JScrollPane(jt);  
        f.add(sp);  
        f.setSize(300, 400);  
        f.setVisible(true);  
    }  
    public static void main(String args[]) {  
        new TableExample();  
    }  
}
```


Q. Write a Java program to find sum of two numbers using swing components. Use text fields for input and output. Your program should display the result when the user presses a button. [10 marks] [Imp]

Solution:

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class Addition extends JFrame implements ActionListener {
    JLabel l1, l2, l3;
    JTextField t1, t2, t3;
    JButton b1;

```

public Addition() {

```

    l1 = new JLabel("First Number:");
    l1.setBounds(20, 10, 100, 20);
    t1 = new JTextField(10);
    t1.setBounds(120, 10, 100, 20);

```

→ 2nd value 2nd ke 3rd ke 4th ke 5th ke 6th ke 7th ke 8th ke 9th ke 10th ke 11th ke 12th ke 13th ke 14th ke 15th ke 16th ke 17th ke 18th ke 19th ke 20th ke 21st ke 22nd ke 23rd ke 24th ke 25th ke 26th ke 27th ke 28th ke 29th ke 30th ke 31st ke 32nd ke 33rd ke 34th ke 35th ke 36th ke 37th ke 38th ke 39th ke 40th ke 41st ke 42nd ke 43rd ke 44th ke 45th ke 46th ke 47th ke 48th ke 49th ke 50th ke 51st ke 52nd ke 53rd ke 54th ke 55th ke 56th ke 57th ke 58th ke 59th ke 60th ke 61st ke 62nd ke 63rd ke 64th ke 65th ke 66th ke 67th ke 68th ke 69th ke 70th ke 71st ke 72nd ke 73rd ke 74th ke 75th ke 76th ke 77th ke 78th ke 79th ke 80th ke 81st ke 82nd ke 83rd ke 84th ke 85th ke 86th ke 87th ke 88th ke 89th ke 90th ke 91st ke 92nd ke 93rd ke 94th ke 95th ke 96th ke 97th ke 98th ke 99th ke 100th ke 101st ke 102nd ke 103rd ke 104th ke 105th ke 106th ke 107th ke 108th ke 109th ke 110th ke 111st ke 112nd ke 113rd ke 114th ke 115th ke 116th ke 117th ke 118th ke 119th ke 120th ke 121st ke 122nd ke 123rd ke 124th ke 125th ke 126th ke 127th ke 128th ke 129th ke 130th ke 131st ke 132nd ke 133rd ke 134th ke 135th ke 136th ke 137th ke 138th ke 139th ke 140th ke 141st ke 142nd ke 143rd ke 144th ke 145th ke 146th ke 147th ke 148th ke 149th ke 150th ke 151st ke 152nd ke 153rd ke 154th ke 155th ke 156th ke 157th ke 158th ke 159th ke 160th ke 161st ke 162nd ke 163rd ke 164th ke 165th ke 166th ke 167th ke 168th ke 169th ke 170th ke 171st ke 172nd ke 173rd ke 174th ke 175th ke 176th ke 177th ke 178th ke 179th ke 180th ke 181st ke 182nd ke 183rd ke 184th ke 185th ke 186th ke 187th ke 188th ke 189th ke 190th ke 191st ke 192nd ke 193rd ke 194th ke 195th ke 196th ke 197th ke 198th ke 199th ke 200th ke 201st ke 202nd ke 203rd ke 204th ke 205th ke 206th ke 207th ke 208th ke 209th ke 210th ke 211st ke 212nd ke 213rd ke 214th ke 215th ke 216th ke 217th ke 218th ke 219th ke 220th ke 221st ke 222nd ke 223rd ke 224th ke 225th ke 226th ke 227th ke 228th ke 229th ke 230th ke 231st ke 232nd ke 233rd ke 234th ke 235th ke 236th ke 237th ke 238th ke 239th ke 240th ke 241st ke 242nd ke 243rd ke 244th ke 245th ke 246th ke 247th ke 248th ke 249th ke 250th ke 251st ke 252nd ke 253rd ke 254th ke 255th ke 256th ke 257th ke 258th ke 259th ke 260th ke 261st ke 262nd ke 263rd ke 264th ke 265th ke 266th ke 267th ke 268th ke 269th ke 270th ke 271st ke 272nd ke 273rd ke 274th ke 275th ke 276th ke 277th ke 278th ke 279th ke 280th ke 281st ke 282nd ke 283rd ke 284th ke 285th ke 286th ke 287th ke 288th ke 289th ke 290th ke 291st ke 292nd ke 293rd ke 294th ke 295th ke 296th ke 297th ke 298th ke 299th ke 300th ke 301st ke 302nd ke 303rd ke 304th ke 305th ke 306th ke 307th ke 308th ke 309th ke 310th ke 311st ke 312nd ke 313rd ke 314th ke 315th ke 316th ke 317th ke 318th ke 319th ke 320th ke 321st ke 322nd ke 323rd ke 324th ke 325th ke 326th ke 327th ke 328th ke 329th ke 330th ke 331st ke 332nd ke 333rd ke 334th ke 335th ke 336th ke 337th ke 338th ke 339th ke 340th ke 341st ke 342nd ke 343rd ke 344th ke 345th ke 346th ke 347th ke 348th ke 349th ke 350th ke 351st ke 352nd ke 353rd ke 354th ke 355th ke 356th ke 357th ke 358th ke 359th ke 360th ke 361st ke 362nd ke 363rd ke 364th ke 365th ke 366th ke 367th ke 368th ke 369th ke 370th ke 371st ke 372nd ke 373rd ke 374th ke 375th ke 376th ke 377th ke 378th ke 379th ke 380th ke 381st ke 382nd ke 383rd ke 384th ke 385th ke 386th ke 387th ke 388th ke 389th ke 390th ke 391st ke 392nd ke 393rd ke 394th ke 395th ke 396th ke 397th ke 398th ke 399th ke 400th ke 401st ke 402nd ke 403rd ke 404th ke 405th ke 406th ke 407th ke 408th ke 409th ke 410th ke 411st ke 412nd ke 413rd ke 414th ke 415th ke 416th ke 417th ke 418th ke 419th ke 420th ke 421st ke 422nd ke 423rd ke 424th ke 425th ke 426th ke 427th ke 428th ke 429th ke 430th ke 431st ke 432nd ke 433rd ke 434th ke 435th ke 436th ke 437th ke 438th ke 439th ke 440th ke 441st ke 442nd ke 443rd ke 444th ke 445th ke 446th ke 447th ke 448th ke 449th ke 450th ke 451st ke 452nd ke 453rd ke 454th ke 455th ke 456th ke 457th ke 458th ke 459th ke 460th ke 461st ke 462nd ke 463rd ke 464th ke 465th ke 466th ke 467th ke 468th ke 469th ke 470th ke 471st ke 472nd ke 473rd ke 474th ke 475th ke 476th ke 477th ke 478th ke 479th ke 480th ke 481st ke 482nd ke 483rd ke 484th ke 485th ke 486th ke 487th ke 488th ke 489th ke 490th ke 491st ke 492nd ke 493rd ke 494th ke 495th ke 496th ke 497th ke 498th ke 499th ke 500th ke 501st ke 502nd ke 503rd ke 504th ke 505th ke 506th ke 507th ke 508th ke 509th ke 510th ke 511st ke 512nd ke 513rd ke 514th ke 515th ke 516th ke 517th ke 518th ke 519th ke 520th ke 521st ke 522nd ke 523rd ke 524th ke 525th ke 526th ke 527th ke 528th ke 529th ke 530th ke 531st ke 532nd ke 533rd ke 534th ke 535th ke 536th ke 537th ke 538th ke 539th ke 540th ke 541st ke 542nd ke 543rd ke 544th ke 545th ke 546th ke 547th ke 548th ke 549th ke 550th ke 551st ke 552nd ke 553rd ke 554th ke 555th ke 556th ke 557th ke 558th ke 559th ke 560th ke 561st ke 562nd ke 563rd ke 564th ke 565th ke 566th ke 567th ke 568th ke 569th ke 570th ke 571st ke 572nd ke 573rd ke 574th ke 575th ke 576th ke 577th ke 578th ke 579th ke 580th ke 581st ke 582nd ke 583rd ke 584th ke 585th ke 586th ke 587th ke 588th ke 589th ke 590th ke 591st ke 592nd ke 593rd ke 594th ke 595th ke 596th ke 597th ke 598th ke 599th ke 600th ke 601st ke 602nd ke 603rd ke 604th ke 605th ke 606th ke 607th ke 608th ke 609th ke 610th ke 611st ke 612nd ke 613rd ke 614th ke 615th ke 616th ke 617th ke 618th ke 619th ke 620th ke 621st ke 622nd ke 623rd ke 624th ke 625th ke 626th ke 627th ke 628th ke 629th ke 630th ke 631st ke 632nd ke 633rd ke 634th ke 635th ke 636th ke 637th ke 638th ke 639th ke 640th ke 641st ke 642nd ke 643rd ke 644th ke 645th ke 646th ke 647th ke 648th ke 649th ke 650th ke 651st ke 652nd ke 653rd ke 654th ke 655th ke 656th ke 657th ke 658th ke 659th ke 660th ke 661st ke 662nd ke 663rd ke 664th ke 665th ke 666th ke 667th ke 668th ke 669th ke 670th ke 671st ke 672nd ke 673rd ke 674th ke 675th ke 676th ke 677th ke 678th ke 679th ke 680th ke 681st ke 682nd ke 683rd ke 684th ke 685th ke 686th ke 687th ke 688th ke 689th ke 690th ke 691st ke 692nd ke 693rd ke 694th ke 695th ke 696th ke 697th ke 698th ke 699th ke 700th ke 701st ke 702nd ke 703rd ke 704th ke 705th ke 706th ke 707th ke 708th ke 709th ke 710th ke 711st ke 712nd ke 713rd ke 714th ke 715th ke 716th ke 717th ke 718th ke 719th ke 720th ke 721st ke 722nd ke 723rd ke 724th ke 725th ke 726th ke 727th ke 728th ke 729th ke 730th ke 731st ke 732nd ke 733rd ke 734th ke 735th ke 736th ke 737th ke 738th ke 739th ke 740th ke 741st ke 742nd ke 743rd ke 744th ke 745th ke 746th ke 747th ke 748th ke 749th ke 750th ke 751st ke 752nd ke 753rd ke 754th ke 755th ke 756th ke 757th ke 758th ke 759th ke 760th ke 761st ke 762nd ke 763rd ke 764th ke 765th ke 766th ke 767th ke 768th ke 769th ke 770th ke 771st ke 772nd ke 773rd ke 774th ke 775th ke 776th ke 777th ke 778th ke 779th ke 780th ke 781st ke 782nd ke 783rd ke 784th ke 785th ke 786th ke 787th ke 788th ke 789th ke 790th ke 791st ke 792nd ke 793rd ke 794th ke 795th ke 796th ke 797th ke 798th ke 799th ke 800th ke 801st ke 802nd ke 803rd ke 804th ke 805th ke 806th ke 807th ke 808th ke 809th ke 810th ke 811st ke 812nd ke 813rd ke 814th ke 815th ke 816th ke 817th ke 818th ke 819th ke 820th ke 821st ke 822nd ke 823rd ke 824th ke 825th ke 826th ke 827th ke 828th ke 829th ke 830th ke 831st ke 832nd ke 833rd ke 834th ke 835th ke 836th ke 837th ke 838th ke 839th ke 840th ke 841st ke 842nd ke 843rd ke 844th ke 845th ke 846th ke 847th ke 848th ke 849th ke 850th ke 851st ke 852nd ke 853rd ke 854th ke 855th ke 856th ke 857th ke 858th ke 859th ke 860th ke 861st ke 862nd ke 863rd ke 864th ke 865th ke 866th ke 867th ke 868th ke 869th ke 870th ke 871st ke 872nd ke 873rd ke 874th ke 875th ke 876th ke 877th ke 878th ke 879th ke 880th ke 881st ke 882nd ke 883rd ke 884th ke 885th ke 886th ke 887th ke 888th ke 889th ke 890th ke 891st ke 892nd ke 893rd ke 894th ke 895th ke 896th ke 897th ke 898th ke 899th ke 900th ke 901st ke 902nd ke 903rd ke 904th ke 905th ke 906th ke 907th ke 908th ke 909th ke 910th ke 911st ke 912nd ke 913rd ke 914th ke 915th ke 916th ke 917th ke 918th ke 919th ke 920th ke 921st ke 922nd ke 923rd ke 924th ke 925th ke 926th ke 927th ke 928th ke 929th ke 930th ke 931st ke 932nd ke 933rd ke 934th ke 935th ke 936th ke 937th ke 938th ke 939th ke 940th ke 941st ke 942nd ke 943rd ke 944th ke 945th ke 946th ke 947th ke 948th ke 949th ke 950th ke 951st ke 952nd ke 953rd ke 954th ke 955th ke 956th ke 957th ke 958th ke 959th ke 960th ke 961st ke 962nd ke 963rd ke 964th ke 965th ke 966th ke 967th ke 968th ke 969th ke 970th ke 971st ke 972nd ke 973rd ke 974th ke 975th ke 976th ke 977th ke 978th ke 979th ke 980th ke 981st ke 982nd ke 983rd ke 984th ke 985th ke 986th ke 987th ke 988th ke 989th ke 990th ke 991st ke 992nd ke 993rd ke 994th ke 995th ke 996th ke 997th ke 998th ke 999th ke 1000th

→ Text Field with maximum input size 10

```

    l2 = new JLabel("Second Number:");
    l2.setBounds(20, 40, 100, 20);
    t2 = new JTextField(10);
    t2.setBounds(120, 40, 100, 20);

```

```

    l3 = new JLabel("Result:");
    l3.setBounds(20, 70, 100, 20);
    t3 = new JTextField(10);
    t3.setBounds(120, 70, 100, 20);

```

```

    add(l1); add(t1); add(l2); add(t2); add(l3); add(t3);

```

```

    b1 = new JButton("Sum");
    b1.setBounds(20, 70, 80, 20);
    add(b1);

```

```

    b1.addActionListener(this);
    setSize(400, 300);
    setLayout(null);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

```



```

@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == b1) {
        int num1 = Integer.parseInt(t1.getText());
        int num2 = Integer.parseInt(t2.getText());
        int sum = num1 + num2;
        t3.setText(String.valueOf(sum));
    }
}

public static void main(String args[]) {
    new Addition();
}
}

```

Multiply को program कर दो इसका int product = num1 * num2 अति Addition, Sum लंबेको इसका Multiplication, product लेखें इस सब same यह है।

Note: Question मा "Use key adapter to handle events" को पढि same यह program लेखे import गर्दा ये एक line को "import java.awt.event.KeyAdaptor" र @Override ~~लेखे~~ लई "class keychecker extends KeyAdaptor {" ले replace गर्ने। [For detail see unitwise question solution in collegenote website.]

Q2. Write a program using swing components to find simple interest. Use text fields for inputs and output. Program should display output if user clicks a button. [Imp]

Solution:

```

// import same all as we did for Q1 before.
class SimpleInterest extends JFrame implements ActionListener {
    // Q1 मा first 2 second number को लागि गरे जस्तै यसमा
    // चारवटा Label (l1, l2, l3, l4), JTextField (t1, t2, t3, t4), र
    // एउटा button बनाएर add गर्ने same as in Q1.
    // चारवटा textField Principal, Time, Rate, र Simple Interest (i.e, result) को
    // लागि use गरेको
}

```

```

@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == b1) {
        double P = Double.parseDouble(t1.getText());
        double T = Double.parseDouble(t2.getText());
        double R = Double.parseDouble(t3.getText());
        double SI = (P * T * R) / 100;
        t4.setText(String.valueOf(SI));
    }
}

```

कलक यसमा यहि हो int को सदा double use हुन्छ।

// Main function मा SimpleInterest() call गर्ने as we did for Addition in Q1.

JavaFX

Introduction

JavaFX is a set of Java graphics libraries for creating Java GUI applications, just like Java AWT and Swing.

JavaFX was originally targeted for Rich Interface Application (RIA, introduced in 2002), i.e., GUI webapp delivered thru a browser's plugin (competing with Adobe Flash, Microsoft Silverlight and Java Applets). However, the trend today is to use HTML5/JavaScript-based, instead of plug-in-based framework. Moreover, browsers (such as Firefox) has stopped supporting plug-ins (such as Java Plug-in for Applets).

History

Sun Microsystems created the Java Programming Language and presented JDK 1.0 in 1995/96. To support GUI programming, Java introduced AWT (Abstract Windowing Toolkit) in JDK 1.1 (1997), and Swing in JDK 1.2 (1998). But many developers felt Swing was over-complex and Java on the desktop never really took off as it did on the server.

Sun Microsystems tried several ways to make it easier to create Java GUI applications. One of these was a scripting language called JavaFX Script 1.0 (2008) which allows developers to build much more complex user Interfaces. But JavaFX Script was not Java. It is a totally new language and never really caught on with Java developers.

When Oracle acquired Sun Microsystems, they killed off JavaFX as a scripting language but added its functionality into the Java Language as JavaFX 2.0 (2011). They enhanced it as the new way to develop user interfaces, intended to replace Swing. Starting from JDK 8 (2014), JavaFX was part of JDK (as JavaFX 8).

Oracle will continue to maintain the Swing library but will not enhance it. Swing and JavaFX can be used together. But for writing new Java applications, JavaFX is recommended as it offers a much simpler way to create desktop applications, and you can write more powerful applications with much less code.

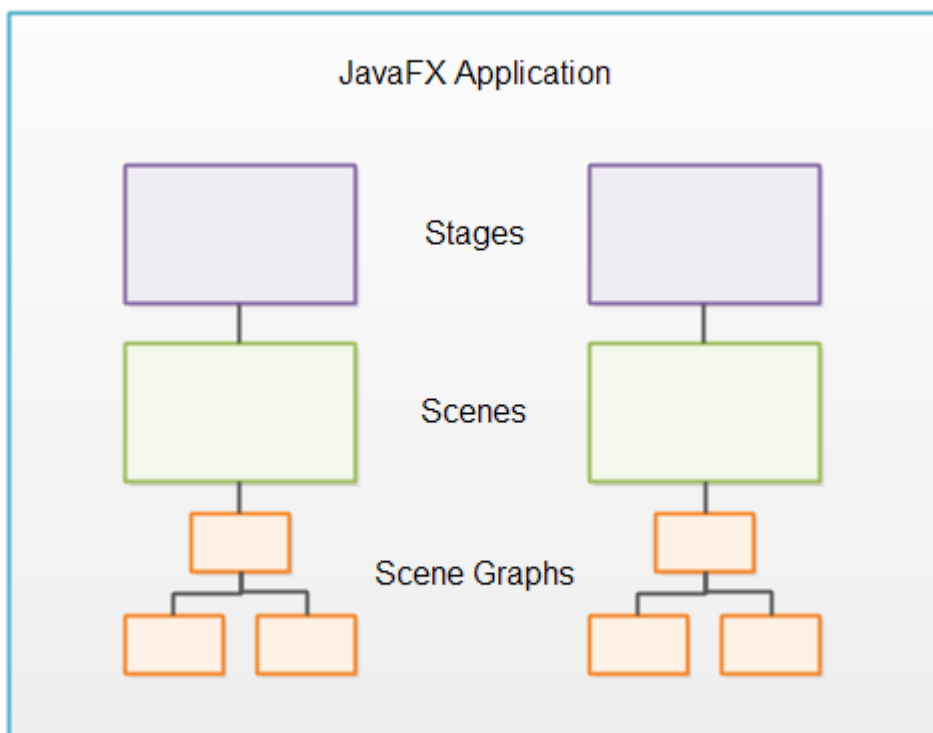
JavaFX Key Features

JavaFX's key features include:

1. From JavaFX 2.0, JavaFX is written in Java (no need to learn a new language). Starting from JDK 8, JavaFX is part of JDK.
2. Support CSS for skinning.
3. Support FXML: a XML-based declarative language to define the structure of the user interface separated from the application code.
4. Swing interoperability: You can use Swing UI in JavaFX application.
5. WebView: for embedding HTML contents.
6. 2D/3D Graphics

7. Media: audio (mp3, wav, aiff), video (flv) and image.
8. Provide a JavaScript engine.

In general, a JavaFX application contains one or more stages which corresponds to windows. Each stage has a scene attached to it. Each scene can have an object graph of controls, layouts etc. attached to it, called the scene graph. These concepts are all explained in more detail later. Here is an illustration of the general structure of a JavaFX application:



Stage

The *stage* is the outer frame for a JavaFX application. The stage typically corresponds to a window. In the early days where JavaFX could run in the browser, the stage could also refer to the area inside the web page that JavaFX had available to draw itself.

Since the deprecation of the Java browser plugin JavaFX is mostly used for desktop applications. Here, JavaFX replaces Swing as the recommended desktop GUI framework. And I must say, that JavaFX looks a whole lot more consistent and feature rich than Swing.

When used in a desktop environment, a JavaFX application can have multiple windows open. Each window has its own stage.

Each stage is represented by a Stage object inside a JavaFX application. A JavaFX application has a primary Stage object which is created for you by the JavaFX runtime. A JavaFX application can create additional Stage objects if it needs additional windows open. For instance, for dialogs, wizards etc.

Scene

To display anything on a stage in a JavaFX application, you need a *scene*. A stage can only show one scene at a time, but it is possible to exchange the scene at runtime. Just like a stage in a theater can be rearranged to show multiple scenes during a play, a stage object in JavaFX can show multiple scenes (one at a time) during the life time of a JavaFX application.

You might wonder why a JavaFX application would ever have more than one scene per stage. Imagine a computer game. A game might have multiple "screens" to show to the user. For instance, an initial menu screen, the main game screen (where the game is played), a game over screen and a high score screen. Each of these screens can be represented by a different scene. When the game needs to change from one screen to the next, it simply attaches the corresponding scene to the Stage object of the JavaFX application.

A scene is represented by a Scene object inside a JavaFX application. A JavaFX application must create all Scene objects it needs.

Scene Graph

All visual components (controls, layouts etc.) must be attached to a scene to be displayed, and that scene must be attached to a stage for the whole scene to be visible. The total object graph of all the controls, layouts etc. attached to a scene is called the *scene graph*.

Nodes

All components attached to the scene graph are called *nodes*. All nodes are subclasses of a JavaFX class called `javafx.scene.Node`.

There are two types of nodes: Branch nodes and leaf nodes. A branch node is a node that can contain other nodes (child nodes). Branch nodes are also referred to as parent nodes because they can contain child nodes. A leaf node is a node which cannot contain other nodes.

Controls

JavaFX controls are JavaFX components which provide some kind of control functionality inside a JavaFX application. For instance, a button, radio button, table, tree etc.

For a control to be visible it must be attached to the scene graph of some Scene object.

Controls are usually nested inside some JavaFX layout component that manages the layout of controls relative to each other.

JavaFX contains the following controls:

- Accordion
- **Button**
- **CheckBox**
- ChoiceBox
- ColorPicker
- ComboBox

- DatePicker
- **Label**
- ListView
- Menu
- MenuBar
- PasswordField
- ProgressBar
- **RadioButton**
- Slider
- Spinner
- SplitMenuButton
- SplitPane
- TableView
- TabPane
- TextArea
- **TextField**
- TitledPane
- **ToggleButton**
- ToolBar
- TreeTableView
- TreeView

Layouts

JavaFX layouts are components which contains other components inside them. The layout component manages the layout of the components nested inside it. JavaFX layout components are also sometimes called *parent components* because they contain child components, and because layout components are subclasses of the JavaFX class `javafx.scene.Parent`.

A layout component must be attached to the scene graph of some Scene object to be visible.

JavaFX contains the following layout components:

- Group
- Region
- Pane

- **HBox**
- **VBox**
- FlowPane
- BorderPane
- BorderPane
- StackPane
- TilePane
- GridPane
- AnchorPane
- TextFlow

Nested Layouts

It is possible to nest layout components inside other layout components. This can be useful to achieve a specific layout. For instance, to get horizontal rows of components which are not laid out in a grid, but differently for each row, you can nest multiple HBox layout components inside a VBox component.

Charts

JavaFX comes with a set of built-in ready-to-use chart components, so you don't have to code charts from scratch everytime you need a basic chart. JavaFX contains the following chart components:

- AreaChart
- BarChart
- BubbleChart
- LineChart
- PieChart
- ScatterChart
- StackedAreaChart
- StackedBarChart

2D Graphics

JavaFX contains features that makes it easy to draw 2D graphics on the screen.

3D Graphics

JavaFX contains features that makes it easy to draw 3D graphics on the screen.

Audio

JavaFX contains features that makes it easy to play audio in JavaFX applications. This is typically useful in games or educational applications.

Video

JavaFX contains features that makes it easy to play video in JavaFX applications. This is typically useful in streaming applications, games or educational applications.

WebView

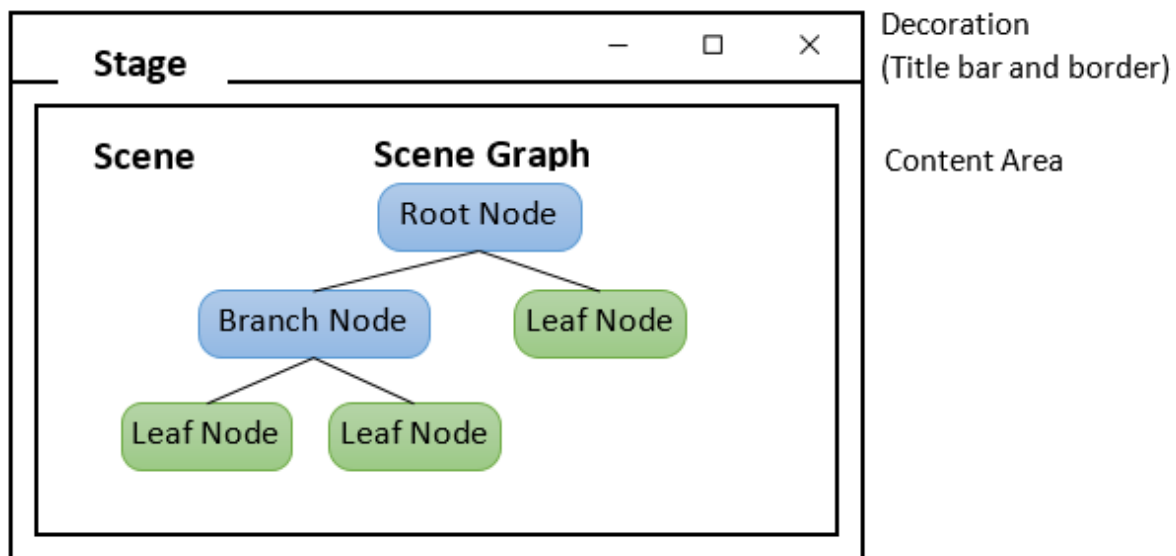
JavaFX contains a WebView component which is capable of showing web pages (HTML5, CSS etc.). The JavaFX WebView component is based on WebKit - the web page rendering engine also used in Chrome and Safari.

The WebView component makes it possible to mix a desktop application with a web application. There are times where that is useful. For instance, if you already have a decent web application, but need some features which can only be provided sensibly with a desktop application - like disk access, communication with other network protocols than HTTP (e.g UDP, IAP etc.) .

JavaFX Application Structure

A JavaFX application (`javafx.application.Application`) comprises:

1. Stage (`javafx.stage.Stage`)
2. Scene (`javafx.scene.Scene`)
3. A hierarchical scene graph of nodes (`javafx.scene.Node`)



Application and Its Life Cycle

A JavaFX application extends from `javafx.application.Application`. The JavaFX runtime maintains an Application's life cycle as follows:

1. It constructs an instance of Application.
2. It calls the Application's `init()` method.
3. It calls the Application's `start(javafx.stage.Stage)` method, and passes the primary stage as its argument.
4. It waits for the Application to complete (e.g., via `Platform.exit()`, or closing all the windows).
5. It calls the Application's `stop()` method.

[TODO] life cycle diagram

The `start()` is an abstract method, that must be overridden. The `init()` and `stop()` has default implementation that does nothing.

If you use `System.exit(int)` to terminate the application, `stop()` will not be called.