SAVITRIBAI PHULE PUNE UNIVERSITY

T.E. (Department of ENTC)

ADV. JAVA PROGRAMMING

COMPLETE NOTES

FOR INSEM EXAMINATION

DESIGNED BY

GRAD HUB TEAM

Unit 1

Unit-I
Applet

GRAD HUB

Important point :

Applet Basics, Applet architecture, HTML APPLET tag, Passing parameter to Appletget, DocumentBase() and getCodeBase() , Japplet: Icons and Labels Text Fields Buttons, Combo Boxes , Checkboxes, Tabbed Panes, Scroll Panes, Trees: Tables.

Important question :

- What are phases of project life cycle. Explain.
- Explain the need of project management in detail..
- Explain the responsibilities of a project manager in detail.
- Explain the phases of Project Management Life Cycle
- Explain various project management process groups in short with their use
- Explain project and explain significance of project management.
- Explain the causes and impacts of delays in Project Management.
- Explain the principle of project management.
- Describe essentials of project management philosophy

Unit 2

Unit-2
Event Handling using AWT/Swing components

GRAD HUB

Important point :

Events, Event sources, Event classes, Event Listeners, Delegation event model, handling mouse and keyboard events, Adapter classes, inner classes. The AWT class hierarchy, user interface components, layout manager.

Important question :

- What is event classes. Enlist its types, explain any two event classes.
- What is event handling? Explain delegation event model.
- Explain the mechanism of mouse event handling with example.
- Explain mechanism of keyboard event handling with suitable example.
- What are adapter classes? List advantages using of adapter classes.
- What is inner class and explain its types with syntax
- Explain AWT class hierarchy.
- Explain following components in AWT. (MenuBar, Text components, Lists, Dialogs, Label, buttons, canvas, scroll bar, checkbox, choices, list panels.)
- Write the short notes on – graphics programming
- Define layout manager and explain its types.

Unit 1

GRAD HUB

Project Basics :

Important question :

What is applets and limitations of AWT. Differentiate Applets and Application

Definition :

- Applet are small programs made in Java that can be sent over the internet and shown on web browsers.
- They can do various things like math, graphics, sound, and animation.

when applets are used:

- Dynamic Web Content: Applets are used for showing changing things on web pages, like moving graphics or real-time updates.
- Special Effects: They're used to add cool effects like sound, animations, or interactive visuals to websites.
- Accessible Applications: Applets are used to make applications that can be used online without needing to download anything extra. By embedded application into web pages.

Limitations of AWT (Abstract Window Toolkit):

AWT was an older system used for making graphical user interfaces in Java. However, it had some limitations:

- Platform Dependency: AWT's appearance and functionality relied on the operating system, so it looked different on various computers.

- Limited Components: It had a small set of built-in components, restricting design creativity.

- Lesser Controls: AWT had fewer control features and couldn't handle complex graphics or multimedia well.:

Unit 1

GRAD HUB

Difference between applets and application :

Applets : are small programs that work inside web browsers to do specific things online.

Applications: are programs that can do many different things on your computer or phone, not just in a web browser.

Aspect	Applets	Applications
Where they run	Inside web browsers	Installed on computers or devices
Main Method	Doesn't have a main method	Have a main method to start execution
Running Status	Runs only when embedded in a webpage	Can run independently anytime
Access to Resources	Restricted access to system resources	Full access to system resources
Deployment	Transferred over the internet	Installed or downloaded
Use Case	For interactive web content	Perform various tasks on devices

GRAD HUB

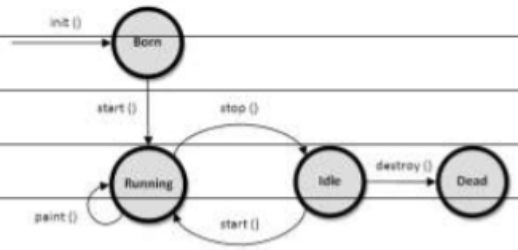
Applet Life cycle (architecture):

Important question :

Explain the life cycle of applets with suitable diagram

The following State which are typically used in applet for initialization and termination purpose :

Born state, Running state, Display state, Idle state, Dead state as shown in the below figure,



1. Born State: This is where the applet gets ready to start. The method used here is 'init()'. It sets up initial values and prepares the applet for running.

2. Running State: Once initialized, the applet enters the running state. Here, the 'start()' method gets triggered. This is where the applet begins executing and performing its tasks.

3. Display State : applet enters in the display state when it wants to display some output. This may happen when applet enters in the running state, the paint() method is for displaying or drawing the contents on the screen.

4. Idle State: The applet might move into an idle state if it's not actively being used. This happens with the 'stop()' method, pausing the applet's actions temporarily.

5. Dead or Destroyed State: When the applet is done or needs to be removed from memory, the 'destroy()' method is called. This is the termination phase where resources are released, and the applet is removed from the system.

GRAD HUB

Important question :

Explain all attributes available in <applet> tag.

HTML APPLET tag

The '<applet>' tag in HTML is used to embed Java applets into a web page. Here are the attributes available within the '<applet>' tag are as follow :

1. CODE: Specifies the filename of the applet's Java class file. It's essential to define which Java class represents the applet.

2. WIDTH: Sets the width of the applet's display area in pixels or as a percentage of the browser window width.

3. HEIGHT: Determines the height of the applet's display area in pixels or as a percentage of the browser window height.

4. ARCHIVE: Specifies a comma-separated list of archive files containing classes and resources necessary for the applet. This helps in efficient resource management.

5. ALIGN: Positions the applet within the web page. It could be set to values like "left," "right," "top," "bottom," or "middle."

6. ALT: Provides alternative text or a message that appears if the browser does not support Java or if the applet fails to load.

7. NAME: Assigns a name to the applet, allowing other HTML elements to refer to it using JavaScript or other scripts.

8. OBJECT: Represents the serialized applet file and is used instead of the 'code' attribute when the applet is stored in an external file.

9. MAYSCRIPT: Informs the browser whether the applet can interact with JavaScript

on the web page. Values can be "true" or "false.".

10. CODEBASE: Specifies the base URL for the applet's class files. It helps in locating the Java classes if they are not in the same directory as the HTML file.

11. PARAM: Allows passing parameters to the applet. These parameters can be accessed from within the applet to configure its behavior.

GRAD HUB

Important question :

Explain <PARAM> tag of applet with suitable example.

Creating an applets

- We know Applets are Java programs that run within a web browser to provide dynamic content. They use two primary classes, 'Applet' and 'Graphics'.
- The 'java.applet' package is essential for the 'Applet' class, which provides lifecycle methods like 'init()', 'start()', and 'paint()' for managing the applet.
- The 'Graphics' class is supported by the 'java.awt' package, enabling the applet to draw graphics and display content on the screen.

Applet Example:

- Below is a basic example of an applet named 'FirstApplet' that extends the 'Applet' class:

```
Import java.awt.;
Import java.applet.;

Public class FirstApplet extends Applet {
    Public void paint(Graphics g) {
        g.drawString("This is my First Applet", 50, 30);
    }
}
```

- The 'paint()' method in the 'FirstApplet' class is responsible for displaying the string "This is my First Applet" at coordinates (50, 30).

Executing the applets :

There are two methods of executing an applet program -

1. Adding Applet to HTML File

2. Using Appletviewer tool

GRAD HUB

1. Adding Applet to HTML File:

To execute an applet by adding it to an HTML file, you follow these steps:

Step 1: Create the Applet Java File

Let's say you have an applet code in a file named 'MyApplet.java'.

```
Import java.applet.Applet;
Import java.awt.Graphics;
Public class MyApplet extends Applet {
    Public void paint(Graphics g) {
        g.drawString("Hello, this is my applet!", 20, 20);
    }
}
```

Step 2: Compile the Applet Java File

Compile the 'MyApplet.java' file using the Java compiler:

```
Javac MyApplet.java
```

Step 3: Create the HTML File

Create an HTML file (let's call it 'appletExample.html') that embeds the applet:

```
<!DOCTYPE html>
<html>
<head>
    <title>Applet Example</title>
</head>
<body>
    <applet code="MyApplet.class" width="300" height="200">
        Your browser does not support Java.
    </applet>
```

```
</body>
</html>
```


Step 4: Run the HTML File

Open the HTML file ('appletExample.html') in a web browser to view the applet.

2. Using Appletviewer Tool:

Step 1. Embedding Applet Code in Java:

- Applet code can be embedded in a Java program using comment statements.
- Start by writing the Java program and enclose the applet code within comment blocks.
- For example:

```
import java.awt.;
import java.applet.;
<applet code="FirstApplet" width=300 height=100> </applet>
public class FirstApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("This is my First Applet", 50, 30);
    }
}
```

Step 2. Adding Applet Code:

- The applet code is inserted at the beginning of the program within comment tags.
- <applet code="FirstApplet" width=300 height=100> indicates that the source program is an applet named FirstApplet with dimensions 300 pixels width and 100 pixels height.

Step 3. Running the Applet using Appletviewer:

- After editing the program with the applet code, compile the Java file using javac FirstApplet.java.
- Run the applet program using the Appletviewer command.

```
D:\test>javac FirstApplet.java
D:\test>Appletviewer FirstApplet.java
```


Step 4. Result:

- The result will display the output of the applet in a separate window as specified by the width and height attributes in the applet code.
- For the given example, the applet will display the text "This is my First Applet" at the coordinates (50, 30) within the applet window.

GRAD HUB

Important question :

Explain <PARAM> tag of applet with suitable example.

Passing Parameters to an Applet

Definition:

- Parameters in applets are conveyed through <PARAM> tags within the opening and closing APPLET tags, defined as NAME=VALUE pairs.

-the important point is that Multiple <PARAM> tags can exist inside the APPLET tag.

- The 'getParameter()' method of the Applet class retrieves the values specified in the PARAM tags.

Syntax :

```
<param name="Username" value="Parth"/>
```

Code Example:

```
import java.applet.Applet;
import java.awt.Graphics;

<applet code="WelcomeParam" width=200 height=200>
    <param name="Username" value="Parth">
</applet>
public class WelcomeParam extends Applet {
    String msg = "";

    public void init() {
        msg = getParameter("Username");
        newmsg = "Welcome " + msg;
    }
    public void paint(Graphics g) {
```

```
        g.drawString(msg, 50, 50);
    }
}
```

GRAD HUB

Explanation:

- Initialization: In the 'init()' method, the 'getParameter("Username")' fetches the value specified in the PARAM tag with the name "Username".
- Message Display: The retrieved username is concatenated with the "Welcome" message.
- Painting: The 'paint()' method displays the modified message using the 'drawString()' method of the Graphics class.

This approach showcases the fundamental usage of passing parameters to an applet, dynamically customizing its behavior based on the provided parameters.

GRAD HUB

Important question :

Explain With example : getdocumentbase() and getcodebase().

1. DocumentBase():

- 'DocumentBase()' is a method in Java used to retrieve the base URL of the document where the applet is embedded.
- It returns a 'URL' object representing the location of the document containing the applet.

Example:

```
Import java.applet.;
Import java.net.;

Public class DocumentBaseExample extends Applet {
    Public void init() {
        URL documentBase = getDocumentBase();
        System.out.println("Document Base: " + documentBase);
    }
}
```

Explanation:

- In this example, the 'getDocumentBase()' method retrieves the URL of the HTML document where the applet is running.
- The 'documentBase' variable stores this URL, allowing access to resources relative to the document's location.

2. getCodeBase():

- 'getCodeBase()' is another method in Java used to retrieve the base URL of the applet itself.
- It returns a 'URL' object representing the location of the applet's code.

Example:

```
Import java.applet.;
Import java.net.;
Public class CodeBaseExample extends Applet {
    Public void init() {
        URL codeBase = getCodeBase();
        System.out.println("Code Base: " + codeBase);
    }
}
```

Explanation:

- Here, the 'getCodeBase()' method retrieves the URL of the directory containing the applet's code.
- The 'codeBase' variable stores this URL, useful for accessing resources relative to the applet's code location.

These methods are essential in Java applet programming for obtaining the base URLs of the document and the applet itself, enabling access to resources based on their relative locations

GRAD HUB

Important question :

- **Explain the various controls in applet with suitable example : (buttons, text field, combo boxes, checkboxes. Etc) (Out of these all any 3 can ask in exam)**

various GUI components in Java applets:

1. JApplet:

Definition: JApplet is a class that represents the applet in Java. It's a part of the javax.swing package and extends the Applet class.

Syntax:

```
Import javax.swing.JApplet;
Public class MyApplet extends JApplet {
    // Applet code here
}
```

Syntax Explanation: The JApplet class provides a framework for creating applets with a graphical user interface using Swing components.

Example Code with Explanation:

```
Import javax.swing.;
Public class MyApplet extends JApplet {
    Public void init() {
        // Initialize and add components to the applet
        JLabel label = new JLabel("Hello, this is a JApplet!");
        Add(label);
    }
```

```
}
```

Here, a simple JApplet named 'MyApplet' is created with a JLabel that displays "Hello, this is a JApplet!" when the applet is launched.

2. Icons

Definition: Icons in Java are graphical representations used to decorate components such as buttons, menus, etc.

Syntax:

```
Icon icon = new ImageIcon("path_to_image.png");
```

Syntax Explanation: Icons are usually created using ImageIcon class, passing the path to the image file as an argument.

Example Code with Explanation:

```
Import javax.swing.;
Public class IconExample extends JFrame {
    Public IconExample() {
        ImageIcon icon = new ImageIcon("icon.png");
        JLabel label = new JLabel(icon);
        Add(label);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(200, 200);
        setVisible(true);
    }
    Public static void main(String[] args) {
        New IconExample();
    }
}
```

In this example, an ImageIcon is created and added to a JLabel, then the JLabel is added to a JFrame to display the icon.

3. Labels

Definition:

Labels are used to display text or images in a GUI application.

Syntax:

```
JLabel label = new JLabel("Text");
```

Explanation: JLabel is used to create a display area for a string of text or an image.

Example Code with Explanation:

```
Import javax.swing.;
Public class LabelExample extends JFrame {
    Public LabelExample() {
        JLabel label = new JLabel("This is a label");
        Add(label);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(200, 100);
        setVisible(true);
    }
    Public static void main(String[] args) {
        New LabelExample();
    }
}
```

This code creates a JFrame with a JLabel displaying the text "This is a label".

4. Text Fields Buttons :

Definition: Text Fields and Buttons are components used for user input and interaction.

Syntax:

```
JTextField textField = new JTextField("Initial text", 15);
JButton button = new JButton("Click me");
```

Syntax Explanation:

JTextField creates a text input field, while JButton creates a clickable button.

Example Code with Explanation:

```
Import javax.swing.;
Public class TextFieldButtonExample extends JFrame {
    Public TextFieldButtonExample() {
        JTextField textField = new JTextField("Type here", 20);
        JButton button = new JButton("Click me");
        Add(textField);
        Add(button);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 100);
        setLayout(new FlowLayout());
        setVisible(true);
    }
    Public static void main(String[] args) {
```

```
        New TextFieldButtonExample();
    }
}
```


This code creates a JFrame with a text field and a button using JTextField and JButton.

5. Combo Boxes

Definition: Combo Boxes are components that allow users to select an item from a dropdown list.

Syntax:

```
String[] options = { "Option 1", "Option 2", "Option 3" };
JComboBox<String> comboBox = new JComboBox<>(options);
```


Explanation: JComboBox creates a dropdown list of items that users can select.

Example Code with Explanation:

```
Import javax.swing.;

Public class ComboBoxExample extends JFrame {
    Public ComboBoxExample() {
        String[] options = { "Apple", "Banana", "Orange" };
        JComboBox<String> comboBox = new JComboBox<>(options);
        Add(comboBox);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(200, 100);
        setVisible(true);
    }
```

```
    Public static void main(String[] args) {
        New ComboBoxExample();
    }
}
```


This code creates a JFrame with a JComboBox containing fruit options.

6. Checkboxes :

Definition: Checkboxes are components that allow users to make multiple selections by clicking on them.

Syntax:

```
JCheckBox checkBox = new JCheckBox("Option");
```

Explanation: JCheckBox creates a checkbox with a label indicating an option.

Example Code with Explanation:

```
Import javax.swing.;
Public class CheckBoxExample extends JFrame {
    Public CheckBoxExample() {
        JCheckBox checkBox1 = new JCheckBox("Option 1");
        JCheckBox checkBox2 = new JCheckBox("Option 2");

        Add(checkBox1);
        Add(checkBox2);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(200, 100);
        setLayout(new FlowLayout());
        setVisible(true);
    }
```

```
    Public static void main(String[] args) {
        New CheckBoxExample();
    }
}
```

This code creates a JFrame with two JCheckBox components.

7. Tabbed Panes :

Definition: Tabbed Panes are containers that allow switching between multiple panels through tabs.

Syntax:

```
JTabbedPane tabbedPane = new JTabbedPane();
```

Explanation: JTabbedPane organizes components into tabs, allowing navigation between them.

Example Code with Explanation:

```
Import javax.swing.;
Public class TabbedPaneExample extends JFrame {
    Public TabbedPaneExample() {
        JTabbedPane tabbedPane = new JTabbedPane();
        JPanel panel1 = new JPanel();
        JPanel panel2 = new JPanel();
        tabbedPane.addTab("Tab 1", panel1);
        tabbedPane.addTab("Tab 2", panel2);
        add(tabbedPane);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 200);
```

```
        setSize(300, 200);
        setVisible(true);
    }

    Public static void main(String[] args) {
        New TabbedPaneExample();
    }
}
```

This code creates a JFrame with a JTabbedPane containing two tabs.

8. Scroll Panes :

Definition: Scroll Panes are containers used to add scrolling functionality to a component that exceeds the visible area.

Syntax:

```
JScrollPane scrollPane = new JScrollPane(component);
```

Syntax Explanation: JScrollPane provides a scrollable view of another component.

Example Code with Explanation:

```
Import javax.swing.;

Public class ScrollPaneExample extends JFrame {
    Public ScrollPaneExample() {
        JTextArea textArea = new JTextArea(20, 30);
        JScrollPane scrollPane = new JScrollPane(textArea);

        Add(scrollPane);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 200);
        setVisible(true);
    }
    Public Static void main(String[] args) {
```

```
        New ScrollPaneExample();
    }
}
```

This code creates a JFrame with a JScrollPane containing a JTextArea that can be scrolled.

9. Trees and Tables :

Definition: Trees and Tables are components used to display hierarchical data and tabular data, respectively.

Syntax:

```
JTree tree = new JTree();
JTable table = new JTable(rows, columns);
```


Explanation: JTree displays hierarchical data as a tree structure, while JTable represents tabular data.

Example Code with Explanation:

```
Import javax.swing.;

Public class TreeTableExample extends JFrame {
    Public TreeTableExample() {
        DefaultMutableTreeNode root = new DefaultMutableTreeNode("Root");
        DefaultMutableTreeNode child1 = new DefaultMutableTreeNode("Child 1");
        DefaultMutableTreeNode child2 = new DefaultMutableTreeNode("Child 2");
        Root.add(child1);
```

```
        Root.add(child2);

        JTree tree = new JTree(root);
        Add(new JScrollPane(tree));

        String[][] data = { { "1", "John" }, { "2", "Alice" }, { "3", "Bob" } };
        String[] columns = { "ID", "Name" };
        JTable table = new JTable(data, columns);
        Add(new JScrollPane(table));
```



```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setLayout(new BoxLayout(getContentPane(), BoxLayout.Y_AXIS));
        setVisible(true);
    }

    Public static void main(String[] args) {
        New TreeTableExample();
    }
}
```


- This code creates a JFrame displaying both a JTree and a JTable, showcasing hierarchical and tabular data, respectively.

- These examples demonstrate the implementation of various GUI components in Java applets, including their syntax, definitions, and usage explanations.

Unit 2

Unit-2
Event Handling using AWT/Swing components

GRAD HUB

Important point :

Events, Event sources, Event classes, Event Listeners, Delegation event model, handling mouse and keyboard events, Adapter classes, inner classes. The AWT class hierarchy, user interface components, layout manager.

Important question :

- What is event classes. Enlist its types, explain any two event classes.
- What is event handling? Explain delegation event model.
- Explain the mechanism of mouse event handling with example.
- Explain mechanism of keyboard event handling with suitable example.
- What are adapter classes? List advantages using of adapter classes.
- What is inner class and explain its types with syntax
- Explain AWT class hierarchy.

✦ Explain following components in AWT. (MenuBar, Text components, Lists, Dialogs, Label, buttons, canvas, scroll bar, checkbox, choices, list panels.)

Write the short notes on – graphics programming

Define layout manager and explain its types.

Unit 2

GRAD HUB

Important question :

What is event classes. Enlist its types, explain any two classes.

Write the notes on event listeners.

Events :

Events in Java represent changes in the state of an object. For example, when a user clicks the mouse button or presses a key on the keyboard, these actions trigger events.

Event Sources :

- Definition: An event source is the object responsible for generating the event.
- Example: When a button is clicked, it generates an ActionEvent object.
- Objects: Various sources can include buttons, checkboxes, textboxes, and scrollbars.

Event Classes :

- Event classes manage the event-handling mechanism in Java.
- Top Class: EventObject class (in java.util package) is at the hierarchy's top.
- Methods:
  - 'getSource()' and 'toString()' are EventObject methods.
  - 'getId()' (java.awt.event package) identifies the nature of the event.

Various types of Event Classes:

1. ActionEvent:

- Generated when a component is activated (e.g., button press or menu item selection).

2. AdjustmentEvent:

- Generated when scrollbars are used.

3. TextEvent:

- Generated when the text of a component or a text field is changed.

4. ContainerEvent:

- Generated when components are added or removed from a container.

5. ComponentEvent:

- Generated when a component is resized, moved, hidden, or made visible.

6. ItemEvent:

- Generated when an item from a list is selected (e.g., choice or checkbox).

7. FocusEvent:

- Generated when a component receives keyboard focus for input.

8. KeyEvent:

- Generated when a key on the keyboard is pressed or released.

9. WindowEvent:

- Generated when a window is activated, maximized, or minimized.

10. MouseEvent:

- Generated when a mouse is clicked, moved, dragged, or released.

Details of important event class :

1. ActionEvent:

Description:

- Generated when a component is activated, like a button press or menu item selection.

Constructors:

- 'ActionEvent(Object source, int id, String command)'
- 'ActionEvent(Object source, int id, String command, int modifiers)'

Methods:

- 'getActionCommand()': Returns the command associated with this action.
- 'getID()': Returns the event's unique ID.

Constants:

- 'ACTION_PERFORMED': Identifies an action event.

2. TextEvent:

Description:

- Generated when the text of a component or a text field is changed.

- Constructors:
  - 'TextEvent(Object source, int id)'

Methods:

- No specific methods in TextEvent class.

Constants:

- 'TEXT_VALUE_CHANGED': Identifies a text value change event.

Event Listeners in Java :

- Event Listeners in Java are components that wait for specific actions to happen, like a button click or a key press, and then respond to those actions with predefined behavior.
- Interfaces in event listeners set of methods that typically handle different types of events, such as button clicks or mouse movements.
- note that the method in the interface are only declared and not implemented means the method is do not have body.

List of Interface in event listeners :

1. ActionListener Interface:

Purpose: Handles events when an action occurs, like a button click.

Method: 'void actionPerformed(ActionEvent act)'

Example: When you click a button, this method gets called.

2. ItemListener Interface:

Purpose: Deals with events when an item is selected, e.g., choosing from a list or checking a checkbox.

Method: 'void itemStateChanged(ItemEvent It)'

Example: When you pick an option from a dropdown, this method is triggered.

3. KeyListener Interface:

Purpose: Manages events related to key presses, releases, and typing characters.

Methods:

- 'void keyPressed(KeyEvent k)'
- 'void keyReleased(KeyEvent k)'
- 'void keyTyped(KeyEvent k)'

Example: Reacts when a key is pressed or released, or a character is typed.

4. MouseListener Interface:

Purpose: Handles various mouse activities like clicks, presses, releases, entering, and exiting.

Methods:

- 'void mouseClicked(MouseEvent m)'
- 'void mousePressed(MouseEvent m)'
- 'void mouseReleased(MouseEvent m)'
- 'void mouseEntered(MouseEvent m)'
- 'void mouseExited(MouseEvent m)'

Example: When you click the mouse, 'mouseClicked' is triggered.

5. MouseMotion Interface:

Purpose: Manages mouse drag and move events.

Methods:

- 'void mouseDragged(MouseEvent m)'

- 'void mouseMoved(MouseEvent m)'

Example: When you move the mouse, 'mouseMoved' is called.

6. TextListener Interface:

Purpose: Handles events when text in a text field or area is entered or edited.

Methods:

- 'public String paramString()'
- 'public void textValueChanged(TextEvent e)'

Example: When you type in a text field, 'textValueChanged' is invoked.

7. WindowListener Interface:

Purpose: Manages window-related events like opening, closing, activation, and deactivation.

Methods:

- 'void windowOpened(WindowEvent w)'
- 'void windowClosed(WindowEvent w)'
- 'void windowClosing(WindowEvent w)'
- 'void windowActivated(WindowEvent w)'
- 'void windowDeactivated(WindowEvent w)'

Example: When you close a window, 'windowClosing' is triggered.

GRAD HUB

Important question :

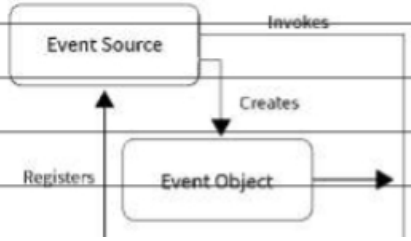
What is event handling? Explain delegation event model.

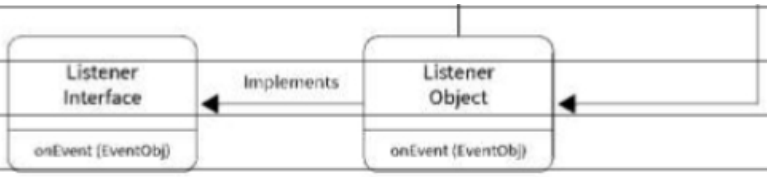
Event handling :

- Event handling in Java refers to the process of responding to user actions or system events, like mouse clicks or key presses, within a Java program.
- event handling involves registering event listeners or handlers to specific components, such as buttons or text fields, and defining methods to execute when those events occur.

Event Delegation Model in Java:

- The Event Delegation Model is a mechanism in Java used for processing events.
- It involves four main components: Event Source, Event Object, Listener Object, and Listener Interface.





1. Event Source: This is the component that generates the event. It could be a button, mouse click, or any user interaction.
2. Event Object: Represents the event itself. When an event occurs, an object encapsulating information about that event is created.
3. Listener Interface: Defines the methods that a listener class must implement to handle specific events.
4. Listener Object: An instance of a class that implements the Listener Interface. It is responsible for processing events.

Advantages of Event Delegation Model:

1. Clear Separation: The model allows a clear separation between the usage of components and their design, making the code more modular and understandable.
2. Performance Acceleration: Improves the performance of applications with multiple events.

Important question :

3. Explain the mechanism of mouse event handling with example.

Handling Mouse Events in Java :

- To manage and respond to mouse events in a Java program, we rely on Listener interfaces. - Two essential interfaces for handling mouse events are 'MouseListener' and 'MouseMotionListener'.

MouseListener Interface:

This interface includes five abstract methods:

1. 'mouseClicked(MouseEvent e)': Invoked when the mouse button is clicked (pressed and released) at the same location.
2. 'mouseEntered(MouseEvent e)': Called when the mouse enters the component's

boundary.

3. 'mouseExited(MouseEvent e)': Triggered when the mouse exits the component's boundary.

4. 'mousePressed(MouseEvent e)': Invoked when a mouse button is pressed.

5. 'mouseReleased(MouseEvent e)': Fired when a mouse button is released after being pressed.

MouseMotionListener Interface:

This interface has two abstract methods:

1. 'mouseDragged(MouseEvent e)': Called when the mouse is moved with a button pressed.

2. 'mouseMoved(MouseEvent e)': Invoked when the mouse is moved without any buttons being pressed.

Example Program:

Imagine you want to create a program that responds to mouse events. Here's a simple example:

```
import java.awt.event.;
class MyMouseListener implements MouseListener, MouseMotionListener {
    public void mouseClicked(MouseEvent e) {
        System.out.println("Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e) {
        System.out.println("Mouse Entered");
    }

    public void mouseExited(MouseEvent e) {
        System.out.println("Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
        System.out.println("Mouse Pressed");
```

```
    }
    public void mouseReleased(MouseEvent e) {
        System.out.println("Mouse Released");
    }

    public void mouseDragged(MouseEvent e) {
        System.out.println("Mouse Dragged");
}

    public void mouseMoved(MouseEvent e) {
        System.out.println("Mouse Moved");
    }
}
public class MouseEventExample {
```



```
    public static void main(String[] args) {
        MyMouseListener listener = new MyMouseListener();
        // Assuming you have a JFrame or JPanel
        yourComponent.addMouseListener(listener);
        yourComponent.addMouseMotionListener(listener);
    }
}
```

GRAD HUB

Important question :

Explain mechanism of keyboard event handling with suitable example.

Handling Keyboard Events in Java

- When a key on the keyboard is pressed, it triggers an event in Java.
- KeyListener interface provides three commonly used methods:

1. keyPressed():
   - This method is part of the KeyListener interface in Java.
   - It is triggered when a key on the keyboard is pressed down.

2. keyReleased():
   - Also part of the KeyListener interface.
   - Activated when a key, previously pressed, is released.

3. keyTyped():
   - Another method from the KeyListener interface.
   - Invoked when a key is both pressed and released, generating a character.

Java Program :

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class KeyboardDemo extends Applet implements KeyListener {

    String msg = "";

    public void init() {
        addKeyListener(this);
        requestFocus();
    }
```



```
    public void keyPressed(KeyEvent k) {
        showStatus("Key Pressed");
    }

    public void keyReleased(KeyEvent k) {
        showStatus("Key Released");
    }

    public void keyTyped(KeyEvent k) {
        Font f1 = new Font("Monotype Corsiva", Font.BOLD, 30);
        msg += k.getKeyChar();
        setFont(f1);
        repaint();
    }

    public void paint(Graphics g) {
```

```
        g.drawString(msg, 30, 70);
    }
}
```

- This Java program uses the KeyListener interface to handle keyboard events.
- keyPressed and keyReleased methods display messages when keys are pressed or released.
- keyTyped method captures the typed character, updates the message, and repaints the display with a specific font.

Important question :

What are adapter classes? List advantages using of adapter classes.

Adapter Classes in Java:

- Adapter classes in Java simplify event handling by providing empty implementations for all methods in an event listener interface.
- They're useful when we only need to handle specific events and not all of them.
- Example listener interfaces and their corresponding adapter classes include:
  - MouseListener with MouseAdapter
  - KeyListener with KeyAdapter
  - WindowListener with WindowAdapter, etc.
- Using adapter classes, we can focus on handling only the events we're interested in.
- For instance, to handle only the KeyTyped event, we can extend the KeyAdapter class and override the keyTyped method.

example:

```
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class Demo extends KeyAdapter {
    JFrame frame;
    JLabel label;

    Demo() {
        frame = new JFrame("KeyTyped Event");
        frame.setSize(300, 200);
        frame.setLayout(null);
        frame.setVisible(true);
```



```
        label = new JLabel("KeyTyped Event");
        label.setBounds(100, 50, 100, 20);
        frame.add(label);

        frame.addKeyListener(this);
    }

    public static void main(String[] args) {
        Demo demo = new Demo();
    }

    public void keyTyped(KeyEvent e) {
        char key = e.getKeyChar();
        String str = "Key Typed: " + key;
        label.setText(str);
    }
```

```
}
```

This program sets up a JFrame with a JLabel and listens for KeyTyped events, updating the label accordingly.

Important question :

- **What is inner class and explain its types with syntax**

Inner class in java :

- An inner class or nested class is a class declared inside another class or interface.
- Inner classes help in logically grouping classes and interfaces for better readability and maintainability.
- Inner classes can access all members of the outer class, including private data members and methods.

Adapter Classes in Event Handling:

- When handling events, if not all methods of a Listener interface are needed, Adapter classes are preferred.
- To use Adapter class in event handling, the implementing class extends the appropriate Adapter class and overloads only the needed methods.

Issue with Multiple Inheritance:

- Java does not support multiple inheritance, preventing the extension of two classes simultaneously.
- To overcome the multiple inheritance issue, an inner class can be defined inside the Applet subclass, extending the needed Adapter class.

Example:

```
class MyClass extends Applet {
    public void init() {
        // Register Listener to Applet and pass object
    }

    public void paint(Graphics g) {
        // Graphics Code...

        class MyAdapter extends MouseAdapter {
```



```
            public void mouseClicked(MouseEvent e) {
                // Event handler implementation goes here
            }
        }
    }
```

- This allows using the Adapter class within the Applet subclass through the inner class.

GRAD HUB

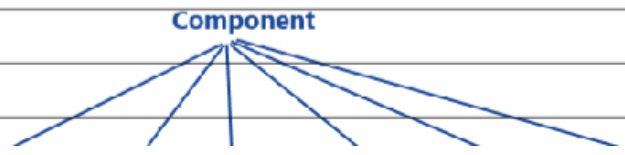
Important question :

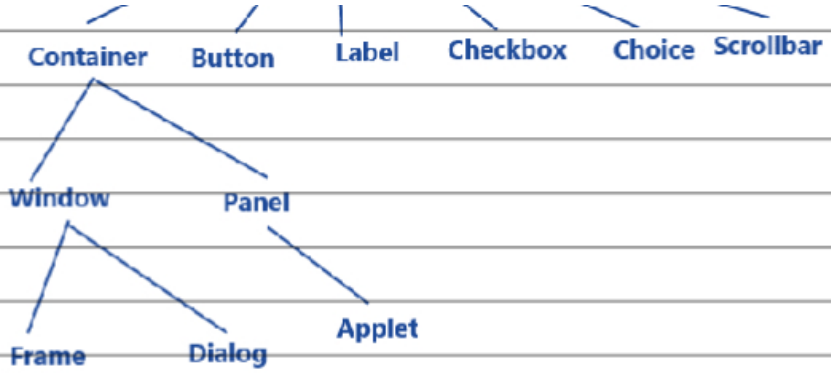
Draw and Explain AWT class hierarchy.

What is AWT?

- AWT: Stands for Abstract Window Toolkit.
- Purpose: In Java programming, AWT assists in adding graphical components like text boxes, buttons, labels, and more.

AWT Class Hierarchy





- Component: Superclass handling graphical objects, managing mouse and keyboard events.
- Container: Manages layout and placement of graphical components.
- Window: Top-level window without a border or menu bar, controlling the window's layout.
- Panel: Similar to a window but without borders, menu bars, or title bars.
- Frame: Top-level window with a border and menu bar, supporting common window events.

GRAD HUB

Limitations of AWT

1. Heaviness: AWT components are heavyweight.
2. Limited Components: Lacks support for advanced components like trees and tables.
3. Platform Dependence: AWT may not behave consistently across platforms.
4. No Picture Support: AWT buttons do not support images.

Important question :

Explain following components in AWT. (MenuBar, Text components, Lists, Dialogs, Label, buttons, canvas, scroll bar, checkbox, choices, list panels.)

Components and Classes

- Graphical components are essential for building user interfaces in Java Swing.
- These components are represented by classes, each having specific methods.

- To arrange components on a frame, the layout must be set.
- A commonly used layout is FlowLayout, placing components from left to right.

Commonly Used Components

1. Label: Displays text or an image.
2. Buttons: Trigger actions when clicked.
3. Canvas: Area for drawing graphics.
4. Scroll Bars: Enable scrolling through content.
5. Text Components: Allow text input or display.
6. Checkbox: Toggle between selected and unselected states.
7. Choices: Dropdown menus for selecting options.
8. Lists: Display a list of items.
9. Panels: Containers for organizing components.
10. Dialogs: Pop-up windows for interactions.
11. Menubar: Top-level menu structure.

Labels :

- Labels are used to display text in GUI applications.

- Label Syntax:
  - Label(String s): Creates a label with the specified text s.
  - Label(String s, int style): Creates a label with text s and a specified style (Label.LEFT, Label.RIGHT, or Label.CENTER).

- Program Example:

```
import java.awt.*;

class Use_Label {
    public static void main(String[] args) {
        // Creating a frame
        Frame fr = new Frame("Displaying Labels");
```

```
        fr.setSize(400, 200);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);

        // Creating labels
        Label L1 = new Label("OK");
        Label L2 = new Label("CANCEL");

        // Adding labels to the frame
        fr.add(L1);
        fr.add(L2);
    }
}
```


- Explanation:
  - Frame: Represents the main window.
  - fr.setSize(400, 200): Sets the size of the frame.
  - fr.setLayout(new FlowLayout()): Sets the layout manager for the frame.

  - fr.setVisible(true): Makes the frame visible.
  - Label L1 = new Label("OK"): Creates a label with the text "OK".
  - Label L2 = new Label("CANCEL"): Creates a label with the text "CANCEL".
  - fr.add(L1); fr.add(L2);: Adds labels to the frame.

Buttons :

- Definition: Buttons, often referred to as push buttons, consist of a label and generate events when pressed.
- Syntax: The syntax for creating a button in Java is Button(String label).
- Java Program Example:

```
import java.awt.*;
class ButtonExample {
```

```
    public static void main(String[] args) {
        // Creating a frame
        Frame frame = new Frame("Button Display Program");
        frame.setSize(400, 200);
        frame.setLayout(new FlowLayout());
        frame.setVisible(true);

        // Creating buttons
        Button okButton = new Button("OK");
        Button cancelButton = new Button("CANCEL");
        // Adding buttons to the frame
        frame.add(okButton);
        frame.add(cancelButton);
    }
}
```


- Explanation:
  - We import the necessary package (java.awt.) for graphical user interface components.
  - A frame (Frame) is created to host the buttons with a specified size and layout.

  - Two buttons (OK and CANCEL) are created using the Button class.
  - The buttons are added to the frame using the add method.

Canvas :

Canvas in Java is like a special drawing area on a frame where we can create graphical elements such as circles, rectangles, and lines. It's like our artistic space!

Methods to Know:

1. setSize(int width, int height): Imagine telling the canvas, "Hey, be this wide and this tall!" It sets the size.

2. setBackground(Color c): This is like picking a backdrop for your canvas. You get to choose the color you want as the background.

3. setForeground(Color c): Now, think of this as choosing the color of your pen for drawing on the canvas. You can pick any color you like!

Example Program:

```
// Let's create a canvas and do some basic settings

import java.awt.Canvas;
import java.awt.Color;
import javax.swing.JFrame;

public class CanvasExample {
    public static void main(String[] args) {
        // Creating a frame
        JFrame frame = new JFrame("My Canvas Example");

        // Creating a canvas
```



```
        Canvas canvas = new Canvas();

        // Setting the size of the canvas
        canvas.setSize(400, 300);

        // Setting the background color of the canvas
        canvas.setBackground(Color.WHITE);

        // Setting the color of the drawing pen on the canvas
        canvas.setForeground(Color.BLUE);
```

```
        // Adding the canvas to the frame
        frame.add(canvas);

        // Setting frame properties
        frame.setSize(500, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

- Here, we create a frame, add a canvas to it, and set the canvas size, background color, and drawing pen color. It's like setting up your art studio before creating your masterpiece!

Scrollbars:

- Scrollbar Representation:
  - Scrollbars are visualized using slider widgets.
  - They come in two styles: Horizontal scrollbar and Vertical scrollbar.

- Program Implementation:
  - In the Java program provided, a Frame named "This program has a scrollbars" is created.
  - Two Scrollbar instances are defined: HSelector for Horizontal and VSelector for Vertical.
  - The layout is set to FlowLayout for a clean arrangement.
  - The frame size is configured to 300x300 pixels.

- The frame is set to be visible, allowing the user to interact with the scrollbars.
- Adding Scrollbars to the Frame:
- Horizontal and Vertical scrollbars are added to the frame using Fr.add(HSelector) and Fr.add(VSelector).

Java Code :

```
import java.awt.*;
class Use_ScrollBars {
    public static void main(String[] args) {
        Frame Fr = new Frame("This program has a scrollbars");
        Scrollbar HSelector = new Scrollbar(Scrollbar.HORIZONTAL);
        Scrollbar VSelector = new Scrollbar(Scrollbar.VERTICAL);
        Fr.setLayout(new FlowLayout());
        Fr.setSize(300, 300);
        Fr.setVisible(true);
        Fr.add(HSelector);
        Fr.add(VSelector);
    }
}
```


TextField and TextArea

1. TextField:
   - Definition: TextField is a Java control for entering and manipulating single-line text.
   - Functionality: Allows input, modification, and basic text operations like copy, cut, and paste.
   - Syntax: int TextField(int n) where 'n' represents the total character limit.

2. TextArea:
   - Definition: TextArea is a Java control designed for handling multi-line text

- Definition: TextArea is a Java control designed for handling multi-line text.
- Functionality: Ideal for input and display of longer text passages.
- Syntax: TextArea(int n, int m) where 'n' is the number of lines, and 'm' is the character limit per line.

Java Program Example:

```
import java.awt.;

class TextComponentsExample {
    public static void main(String[] args) {

        // Creating a window
        Frame frame = new Frame("Text Components Example");
        frame.setSize(350, 300);
        frame.setLayout(new FlowLayout());
        frame.setVisible(true);

        // Creating labels for user guidance
        Label nameLabel = new Label("Enter your name here");
        Label addressLabel = new Label("Enter your address here");
```



```
        // Creating TextField for single-line input
        TextField nameInput = new TextField(10);

        // Creating TextArea for multi-line input
        TextArea addressInput = new TextArea(10, 20);

        // Adding components to the window
        frame.add(nameLabel);
        frame.add(nameInput);
        frame.add(addressLabel);
        frame.add(addressInput);
```

```
    }
}
```

Explanation:

- The program creates a window using Frame and sets its size.
- Labels guide the user on what to input in each component.
- TextField is used for single-line input, while TextArea is for multi-line input.
- Components are added to the frame for user interaction.

Checkbox :

- Definition: A checkbox is a small box that can be either ticked or not ticked, serving as a user interface element for selection.

- Java Usage: Checkboxes in Java are utilized to select specific items through the checkbox control.

- Appearance: The checkbox control consists of a small box accompanied by a label indicating the item's name to be selected

indicating the item's name to be selected.

- Syntax:

  Checkbox(String label)

  where the label parameter represents the associated label for each checkbox.

- State Retrieval: To obtain the state of a checkbox, the getState() method can be employed.

Java Program Example :

```
import java.awt.;

class Use_ChkBox {
    public static void main(String[] args) {
        // Setting up the frame
        Frame fr = new Frame("Checkbox Display Program");
        fr.setSize(350, 300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
```



```
        // Creating checkboxes for Candy, Ice-cream, and Juice
        Checkbox box1 = new Checkbox("Candy");
        Checkbox box2 = new Checkbox("Ice-cream");
        Checkbox box3 = new Checkbox("Juice");

        // Adding checkboxes to the frame
        fr.add(box1);
        fr.add(box2);
        fr.add(box3);
```

```
    }
}
```

Checkbox Group (Radio Buttons) :

- The Checkbox Group component allows users to select one option at a time, similar to radio buttons.
- Checkbox groups are implemented using the CheckboxGroup class in Java.
- The syntax for creating a checkbox in a group is:
  Checkbox(String label, CheckboxGroup group, boolean initialState);
- Below is a simple Java program using checkbox groups:

Java example code :

```
import java.awt.*;

class Use_CheckBoxGr {
    public static void main(String[] args) {
        // Creating a frame
        Frame frame = new Frame("Checkbox Group Example");
        frame.setLayout(new FlowLayout());
```



```
        frame.setSize(300, 300);
        frame.setVisible(true);

        // Creating a CheckboxGroup
        CheckboxGroup checkboxGroup = new CheckboxGroup();

        // Creating checkboxes with labels and adding them to the frame
        Checkbox candyCheckbox = new Checkbox("Candy", checkboxGroup, true);
        Checkbox iceCreamCheckbox = new Checkbox("Ice-cream", checkboxGroup,
```

```
false);
        Checkbox juiceCheckbox = new Checkbox("Juice", checkboxGroup, false);

        frame.add(candyCheckbox);
        frame.add(iceCreamCheckbox);
        frame.add(juiceCheckbox);
    }
}
```

- This program sets up a graphical interface with checkboxes grouped together.

Java Choice Control

- Definition: The Choice control in Java provides a popup list for selection.
- Object Creation: To use Choice, create an object using Choice obj=new Choice();

Sample Program :

```
import java.awt.;
class Use_Choice {
    public static void main(String[] args) {
        // Create a frame
        Frame fr = new Frame("Choice List Display");
```



```
        fr.setSize(350, 300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);

        // Create Choice objects
        Choice c1 = new Choice();
        Choice c2 = new Choice();
```

```
        // Add options to Choice c1
        c1.add("Mango");
        c1.add("Apple");
        c1.add("Strawberry");
        c1.add("Banana");

        // Add options to Choice c2
        c2.add("Rose");
        c2.add("Lily");
        c2.add("Lotus");

        // Add Choice objects to the frame
        fr.add(c1);
        fr.add(c2);
    }
}
```

Explanation :

- Object Creation: Choice obj=new Choice(); creates a Choice object.
- Adding Options: c1.add("Mango"); adds options to the Choice object.
- Frame Configuration: The program sets up a frame with a specified size and layout.
- Visibility: fr.setVisible(true); makes the frame visible.
- Two Choices: c1 and c2 are two Choice objects with different options.
- Adding to Frame: fr.add(c1); and fr.add(c2); add Choice objects to the frame.

Lists :

- A list is a graphical user interface (GUI) component used to display a collection of items.
- Items in a list can be selected, and one way to do this is by double-clicking on the desired item.

Java Program :

```
import java.awt.;
```

```
class Use_List {
    public static void main(String[] args) {
        int i;
        Frame fr = new Frame("List Display Program");
        fr.setSize(350, 300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        List flower = new List(4, false);
        flower.add("Rose");
        flower.add("Jasmine");
        flower.add("Lotus");
        flower.add("Lily");
        fr.add(flower);
    }
}
```

Explanation:

1. Importing Necessary Packages:

- 'import java.awt.;' - Imports the Abstract Window Toolkit (AWT) package for GUI components.

2. Setting up the Frame:

- 'Frame fr = new Frame("List Display Program");' - Creates a new frame with the title "List Display Program."
- 'fr.setSize(350, 300);' - Sets the size of the frame to 350 pixels wide and 300 pixels tall.
- 'fr.setLayout(new FlowLayout());' - Uses a flow layout manager for the frame.
- 'fr.setVisible(true);' - Sets the frame to be visible.

3. Creating a List:

- 'List flower = new List(4, false);' - Creates a list named "flower" with the

capacity to display 4 items and allows only a single item to be selected at a time.

4. Adding Items to the List:

- 'flower.add("Rose");' - Adds "Rose" to the list.
- 'flower.add("Jasmine");' - Adds "Jasmine" to the list.
- 'flower.add("Lotus");' - Adds "Lotus" to the list.
- 'flower.add("Lily");' - Adds "Lily" to the list.

5. Displaying the List:

- 'fr.add(flower);' - Adds the "flower" list to the frame for display.

Dialog Boxes :

Dialog Control:

- Dialog control is like a pop-up window that takes input from the user.
- It has a title and a border.
- Dialog boxes don't have maximize and minimize buttons.

Constructors for Dialog Box:

1. 'Dialog(Dialog owner)': Creates a window without blocking the main window.
2. 'Dialog(Dialog owner, String title)': Creates a window with a specified title.
3. 'Dialog(Dialog owner, String title, boolean modal)': Creates a window with title and option to be modal or modeless.

Modal and Modeless Dialog Boxes:

- Modal Dialog Box:
  - User must close it before interacting with other windows.
  - Temporarily halts the use of the main application.

- Modeless Dialog Box:
  - User can keep it open and interact with other parts of the application.

- Doesn't block the main application.

Java Program Example:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class DialogBoxProg extends Frame {
    public static void main(String[] args) {
        Dialog d;
        Frame frame = new Frame();
        d = new Dialog(frame, "Dialog Box Demo", true);
        d.add(new Label("This is a simple dialog box."));
        d.setSize(300, 300);
        d.setVisible(true);
        frame.setSize(330, 250);
        frame.setVisible(true);
    }
}
```

Explanation:

- Creates a dialog box with a title, "Dialog Box Demo."
- Adds a label to the dialog box.
- Sets sizes and makes both the dialog box and the main frame visible.
- The dialog is modal (blocks interaction with the main frame).

FileDialog in Java

- The 'FileDialog' class in Java is used to display a dialog window allowing users to select a file.
- When this dialog is shown, it pauses the rest of the application until the user chooses a file.

Methods Used:

- 'getDirectory()': Gets the directory of the selected file

- 'getDirectory()': Gets the directory of the selected file.
- 'getFile()': Gets the selected file.
- 'getMode()': Indicates whether the dialog is for loading a file or saving to a file.
- 'setDirectory(String)': Sets the directory for the dialog window.
- 'setFile(String)': Sets the selected file for the dialog window.
- 'setMode(int)': Sets the mode of the file dialog (loading or saving).

Variables Used:

- 'LOAD': Constant indicating the file dialog is for reading a file.
- 'SAVE': Constant indicating the file dialog is for writing a file.

Syntax for Creating FileDialog:

```
public FileDialog(Frame parent, String title, int mode)
```

- Creates a file dialog window with the specified title for loading or saving a file.
- 'mode': If 'LOAD', the dialog is for finding a file to read; if 'SAVE', it's for finding a place to write a file.

Example Java Program:

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

class FileDialogBoxProg extends Frame {
    public static void main(String[] args) {
```



```
        FileDialog fd;
        Frame frame = new Frame();

        // Creating a FileDialog for loading a file
        fd = new FileDialog(frame, "Choose a file...", FileDialog.LOAD);
```

```
        // Setting initial directory for the dialog
        fd.setDirectory("C:\\");

        // Setting size and making the dialog visible
        fd.setSize(300, 300);
        fd.setVisible(true);

        // Setting size and making the frame visible
        frame.setSize(330, 250);
        frame.setVisible(true);
    }
}
```

Menu Bar in Java GUI:

Definition:

Menus are crucial parts of Window-based Graphical User Interfaces (GUIs) that enable users to choose from various options. They are constructed using Menu items and are organized on a Menu bar.

Constructors:

1. 'public MenuBar()': Creates the Menu bar where menus can be added.
2. 'public Menu(String title)': Creates a menu with a specified title.
3. 'public MenuItem(String title)': Generates menu items with suitable titles.

Steps to Create a Menu:

1. Create Menu Bar:
   - Use 'MenuBar menuBar = new MenuBar();'

2. Create Menus:
   - Use 'Menu menu1 = new Menu("File");'

- Use 'Menu menu1 = new Menu("File");'

3. Create Menu Items:
   - Use 'MenuItem mItem1 = new MenuItem("New");'
   - Use 'MenuItem mItem2 = new MenuItem("Open");'
   - Use 'MenuItem mItem3 = new MenuItem("Save");'

4. Add Menu Items to Menus:
   - Use 'menu1.add(mItem1);'
   - Use 'menu1.add(mItem2);'
   - Use 'menu1.add(mItem3);'

5. Add Menus to Menu Bar:
   - Use 'menuBar.add(menu1);'

6. Add Menu Bar to the Frame:
   - Use 'frame.setMenuBar(menuBar);'

Java Program Example:

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

class MenuDemo extends Frame {

    public static void main(String[] args) {
        MenuBar menuBar;
```



```
        Menu menu1;
        MenuItem mItem1, mItem2, mItem3;
        Frame frame = new Frame("MenuBar and Menu Demo");

        // Creating a menu bar
        menuBar = new MenuBar();
```

```
        // Creating menu
        menu1 = new Menu("File");
        // Creating menu items
        mItem1 = new MenuItem("New");
        mItem2 = new MenuItem("Open");
        mItem3 = new MenuItem("Save");

        // Adding menu items to the menu
        menu1.add(mItem1);
        menu1.add(mItem2);
        menu1.add(mItem3);

        // Adding our menu to the menu bar
        menuBar.add(menu1);

        // Adding my menu bar to the frame
        frame.setMenuBar(menuBar);

        frame.setSize(330, 250);
        frame.setVisible(true);
    }
}
```

- The above program creates a simple GUI with a menu bar, a "File" menu, and three menu items: "New," "Open," and "Save." Students can use this as a reference for creating their GUI programs.

Important question :

Write the short notes on – graphics programming

Graphics in Java

- In Java, we can use applets to draw various shapes and add colors to them.
- The drawing happens on a special area called the canvas, which is like a blank sheet where we create our display.

Canvas Methods

To prepare our canvas for drawing 2D shapes, we use specific methods:

- 'setSize(int width, int height)' - This method sets the size of our canvas, making it like a defined piece of paper.
- 'setBackground(Color c)' - We can set the background color of our canvas using this method, just like choosing a background color for a picture.
- 'setForeground(Color c)' - This method allows us to set the color of the text or shapes we draw on the canvas. It's like picking the color of your pen before you start drawing.

Drawing Lines:

- Drawing lines in Java is a straightforward process.
- The syntax for drawing a line is as follows:

```
java
void drawLine(int x1, int y1, int x2, int y2);
```

- Here, (x1, y1) represents the starting point, and (x2, y2) represents the ending point of the line.

Example Program: LineDemo.java

- Let's explore a simple Java program to draw lines.

- The program uses the 'Canvas' class and includes a method 'drawLine' to draw lines.

```
import java.awt.;

class LineDemo extends Canvas {
```

```
    // Constructor to set size and background color
    public LineDemo() {
        setSize(200, 200);
        setBackground(Color.white);
    }

    // Main method to create and display the frame
    public static void main(String[] args) {
        LineDemo obj = new LineDemo();
        Frame fr = new Frame("Line");
        fr.setSize(300, 300);
        fr.add(obj);
        fr.setVisible(true);
    }

    // Method to paint the canvas with lines and text
    public void paint(Graphics g) {
        g.drawLine(0, 0, 200, 100); // Diagonal line from top-left
        g.drawLine(0, 100, 100, 0); // Another diagonal line
        g.drawString("It's a Line demo", 50, 80);
    }
}
```


Understanding the Program:

- The 'LineDemo' class extends 'Canvas' for drawing.
- The constructor sets the size and background color of the canvas.
- The 'main' method creates an instance of 'LineDemo', adds it to a frame, and displays the frame.
- The 'paint' method draws two diagonal lines and adds text to the canvas.

Drawing Rectangles in Java :

- In Java, we can draw two types of rectangles: a normal rectangle and a rectangle with round corners.

Drawing Rectangles:

1. 'void drawRect(int top, int left, int width, int height)' - Draws a normal rectangle.
   - Example: 'g.drawRect(10, 10, 50, 50);'

2. 'void drawRoundRect(int top, int left, int width, int height, int xDiameter, int yDiameter)' - Draws a rectangle with round corners.
   - Example: 'g.drawRoundRect(70, 30, 50, 30, 10, 10);'

Filling Rectangles:

1. 'void fillRect(int top, int left, int width, int height)' - Fills a normal rectangle.
   - Example: 'g.fillRect(40, 100, 150, 100);'

2. 'void fillRoundRect(int top, int left, int width, int height, int xDiameter, int yDiameter)' - Fills a rectangle with round corners.
   - Example: 'g.fillRoundRect(200, 10, 70, 100, 10, 10);'

Java Program Example:

```
import java.awt.;

class RectangleDemo extends Canvas {
    public RectangleDemo() {
  setSize(200, 200);
        setBackground(Color.white);
    }
```



```
    public static void main(String[] args) {
        RectangleDemo obj = new RectangleDemo();
        Frame fr = new Frame("Rectangle");
        fr.setSize(300, 300);
```

```
        fr.add(obj);
        fr.setVisible(true);
    }

    public void paint(Graphics g) {
        // Drawing rectangles
        g.drawRect(10, 10, 50, 50);
        g.drawRoundRect(70, 30, 50, 30, 10, 10);

        // Filling rectangles
        g.fillRect(40, 100, 150, 100);
        g.fillRoundRect(200, 10, 70, 100, 10, 10);

        // Adding text
        g.drawString("It's a Rectangle Demo", 30, 90);
    }
}
```

- The provided Java program example demonstrates how to use these methods to create a simple rectangle demo.

Oval Drawing in Java

Drawing Circles and Ellipses

In Java, circles and ellipses can be drawn using the 'drawOval()' method. This method takes four parameters: 'top', 'left', 'width', and 'height'.

Syntax:

```
void drawOval(int top, int left, int width, int height);
```


Filling the Oval

To fill an oval, the 'fillOval()' method is used. This method also takes four parameters: 'top', 'left', 'width', and 'height'.

Syntax:

```
java
void fillOval(int top, int left, int width, int height);
```

Setting Fill Color

To specify the color for filling the oval, the 'setColor' method is used.

Example:

```
g.drawOval(10, 10, 200, 100);
g.setColor(Color.blue);
g.fillOval(30, 50, 200, 100);
```

Java Program Example - OvalDemo.java

```
import java.awt.*;

class OvalDemo extends Canvas {
    public OvalDemo() {
        setSize(200, 200);
        setBackground(Color.white);
    }
    public static void main(String[] args) {
        OvalDemo obj = new OvalDemo();
        Frame fr = new Frame("Circle and Oval");
        fr.setSize(300, 300);
        fr.add(obj);
        fr.setVisible(true);
    }
```



```
    public void paint(Graphics g) {
        g.drawOval(10, 10, 50, 50);
        g.fillOval(200, 10, 70, 100);
```

```
        g.drawString("Its Circle and Oval Demo", 40, 40);
    }
}
```

Explanation

- The 'drawOval()' method draws the outline of an oval at coordinates (10, 10) with a width of 50 and a height of 50.
- The 'fillOval()' method fills an oval with a blue color starting from coordinates (30, 50) with a width of 200 and a height of 100.
- The program creates an instance of the 'OvalDemo' class, sets up a frame, and displays it.

This program showcases the basic usage of drawing ovals and circles in Java, providing a foundation for more advanced graphical applications.

Arc Drawing in Java:

Drawing Arcs in Java:

In Java, you can create arcs using the 'drawArc()' and 'fillArc()' functions.

- Syntax for drawArc():

  'void drawArc(int top, int left, int width, int height, int angle1, int angle2)'

- Syntax for fillArc():

  'void fillArc(int top, int left, int width, int height, int angle1, int angle2)'

- Angle Representation:

  - 'angle1': Starting angle of the arc.
  - 'angle2': Angular distance covered by the arc.

Example Program: arcDemo.java

```
import java.awt.*;

class arcDemo extends Canvas {

    public arcDemo() {
        setSize(200, 200);
        setBackground(Color.white);
    }

    public static void main(String[] args) {
        arcDemo obj = new arcDemo();
        Frame fr = new Frame("Arc");
        fr.setSize(300, 300);
        fr.add(obj);
        fr.setVisible(true);
    }

    public void paint(Graphics g) {
        // Draw a black outlined arc
        g.drawArc(100, 60, 100, 100, 0, 90);

        // Fill the arc with green color
        g.setColor(Color.green);
        g.fillArc(100, 60, 55, 70, 0, 90);

        // Draw another black outlined arc
        g.setColor(Color.black);
        g.drawArc(100, 100, 70, 90, 0, 270);
```



```
        // Display a text message
        g.drawString("It's an arc Demo", 120, 160);
    }
```

```
}
```

Explanation:

- The program creates a canvas with a white background.
- It draws a black-outlined arc at coordinates (100, 60) with dimensions 100x100, starting from angle 0 to angle 90.
- Then, it fills a part of this arc with a green color, specified by a smaller bounding rectangle (55x70).
- Another black-outlined arc is drawn at coordinates (100, 100) with dimensions 70x90, spanning from angle 0 to angle 270.
- Finally, a text message is displayed on the canvas.

Important question :

Define layout manager and explain its types.

Layout manager :

- A layout manager is interface which automatically arranges the controls on the screen by using some types of algorithm.
- each container object has layout manager associated with it.
- layout manager is an instance of any class that implements layoutManager interface.
- The layout manager is set by method :

Void setLayout(layoutManager LayoutObj)

- If no call to setLayout() is made then the default layout manger is used.

Types of layout manager :

FlowLayout

- FlowLayout is a default layout manager that embodies a straightforward layout style, akin to the flow of words in a text editor.
- The layout's direction is determined by the container's component orientation, typically left to right and top to bottom by default.
- Components are arranged line-by-line, commencing from the upper-left corner.
- When a line reaches maximum capacity, the layout proceeds to the next line.
- A small space is reserved between each component, including above, below, to the left, and to the right.

Constructors available:

- FlowLayout(): Default constructor.
- FlowLayout(int how): Constructor with alignment specification.
- FlowLayout(int how, int horz, int vert): Constructor allowing customization of horizontal and vertical spacing.

- The parameter 'how' dictates alignment on each line, with options including:

- FlowLayout.LEFT
- FlowLayout.CENTER
- FlowLayout.RIGHT
- FlowLayout.LEADING
- FlowLayout.TRAILING

- Horizontal and vertical spacing between components are determined by 'horz' and 'vert' parameters, respectively.

Syntax for implementation:

```
frame.setLayout(new FlowLayout(FlowLayout.LEFT));
```

Visual representation:

- Components are arranged in accordance with the specified alignment, with appropriate spacing between them.

BorderLayout():

- BorderLayout is a class in Java that implements a common layout style for top-level windows.
- It divides the layout into five regions: north, south, east, west, and center.
- The north, south, east, and west regions are narrow and fixed-width, while the center region is larger.

Constructors:

- BorderLayout(): Creates a new BorderLayout with default horizontal and vertical gaps between components.
- BorderLayout(int horz, int vert): Creates a new BorderLayout with specified horizontal (horz) and vertical (vert) gaps between components.

Constants:

- BorderLayout.CENTER: Specifies the center region.

- BorderLayout.SOUTH: Specifies the south region.

- BorderLayout.EAST: Specifies the east region.
- BorderLayout.WEST: Specifies the west region.
- BorderLayout.NORTH: Specifies the north region.

Example usage:

```
frame.setLayout(new BorderLayout());
frame.add(Component obj, BorderLayout.CENTER);
```


- This layout is useful for organizing components in a top-level window or container, providing a structured approach to arranging user interface elements.

GridLayout in Java :

- GridLayout organizes components in a two-dimensional grid of rows and columns.

Constructors:

- 'GridLayout()'
- 'GridLayout(int numRows, int numCols)'
- 'GridLayout(int numRows, int numCols, int horz, int vert)'

Constructor Details:

Default Constructor:

- Creates a single-column grid layout.

Second Constructor:

- Creates a grid layout with a specified number of rows and columns.

Third Constructor:

- Allows specifying horizontal and vertical space between components in 'horz' and 'vert'.

Special Cases:

- Unlimited Columns:
  - Setting 'numRows' as zero allows unlimited-length columns.

- Unlimited Rows:
  - Setting 'numCols' as zero allows unlimited-length rows.

Example Syntax:
- 'frame.setLayout(new GridLayout(5, 4));'
  - Sets a 5x4 grid layout for the frame.

Usage in Components:
- Use with 'BorderLayout.CENTER' to place a component in the center:
  - 'Component obj, BorderLayout.CENTER'

CardLayout :

- CardLayout is a layout manager in Java Swing that allows storing multiple layouts akin to different index cards in a deck.
- Each layout represents a card in the deck and can be dynamically switched to be displayed.
- Useful for UIs with optional components activated based on user input.
- Preparing layouts in advance and hiding them allows for efficient activation when needed.

- Constructors:
  - 'CardLayout()': Creates a default card layout.
  - 'CardLayout(int horz, int vert)': Specifies horizontal and vertical space between components.
- Cards are typically managed within a Panel with CardLayout selected as its layout manager.
- Cards are represented as objects of type Panel.

Methods for activating cards:
- 'void first(Container deck)': Activates the first card in the container.
- 'void last(Container deck)': Activates the last card in the container.

- `void next(Container deck)`: Activates the next card in the container.
- 'void previous(Container deck)': Activates the previous card in the container.
- 'void show(Container deck, String cardName)': Activates a specific card by its name.

Example Usage:

```
Panel cards = new Panel(); // Creating a panel to hold cards
CardLayout layout = new CardLayout(); // Creating a CardLayout instance
cards.setLayout(layout); // Setting CardLayout as the layout manager for the panel

Panel card1 = new Panel(); // Creating a panel for card 1
Panel card2 = new Panel(); // Creating a panel for card 2

cards.add(card1, "Card 1"); // Adding card 1 to the deck with name "Card 1"
cards.add(card2, "Card 2"); // Adding card 2 to the deck with name "Card 2"

layout.show(cards, "Card 1"); // Displaying card 1 on the panel
```

GridBagLayout:

- Definition: GridBagLayout specifies the relative placement of components by positioning them within cells in a grid.
- Flexibility: Each component can have different sizes, and each row can have a different number of columns.
- Terminology: Known as a "grid bag" due to its collection of small grids joined together.
- Component Placement: Determined by constraints linked to each component, including height, width, alignment, and anchor point within the cell.
- Constraints: Govern the size, placement, and spacing of components. GridBagConstraints class defines these constraints.
  - Fields include anchor, gridx, gridy, ipadx, ipady, gridheight, gridwidth, and fill.

- Location Specification:
  - 'GridBagConstraints.WEST', 'GridBagConstraints.EAST', 'GridBagConstraints.SOUTH', 'GridBagConstraints.NORTH': Specify component placement within the cell.
  - 'gridx' and 'gridy': Define X and Y coordinates for adding components.
- Size Specification:
  - 'ipadx' and 'ipady': Extra horizontal and vertical space around the component.
  - 'gridheight' and 'gridwidth': Height and width of the component.

- Resizing:
  - 'fill': Determines how a component is resized, especially when it's smaller than the cell. Values include 'GridBagConstraints.NONE', 'GridBagConstraints.HORIZONTAL', 'GridBagConstraints.VERTICAL', and 'GridBagConstraints.BOTH'.

- Methods:
  - 'setConstraints(Component comp, GridBagConstraints cons)': Sets constraints for a component.
  - 'getConstraints(Component comp)': Retrieves constraints applied to a component.

Example:

```
GridBagLayout gbag = new GridBagLayout();
GridBagConstraints gbc = new GridBagConstraints();
gbc = setLayout(gbag);
Checkbox winXP = new Checkbox("Windows XP ", null, true);
gbc.weightx = 1.0; // use a column weight of 1
gbc.ipadx = 200; // pad by 200 units
gbc.insets = new Insets(4, 4, 0, 0); // inset slightly from top left
gbc.anchor GridBagConstraints.NORTHEAST;
gbc.gridwidth GridBagConstraints.RELATIVE;
gbag.setConstraints (winXP, gbc);
```

Explanation:

- Initializes GridBagLayout and GridBagConstraints objects.
- Sets layout to the grid bag layout.
- Creates a Checkbox component for Windows XP.
- Defines weight, padding, insets, anchor, and grid width for the component.
- Sets constraints for the component using 'setConstraints' method.

– Thank You –

All The Best...