CS 5 (Programming in Java) Unit III Long Answer Type

Q) Explain about Java Applet(s).

Applet is a special type of Java program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantages of Applets :-

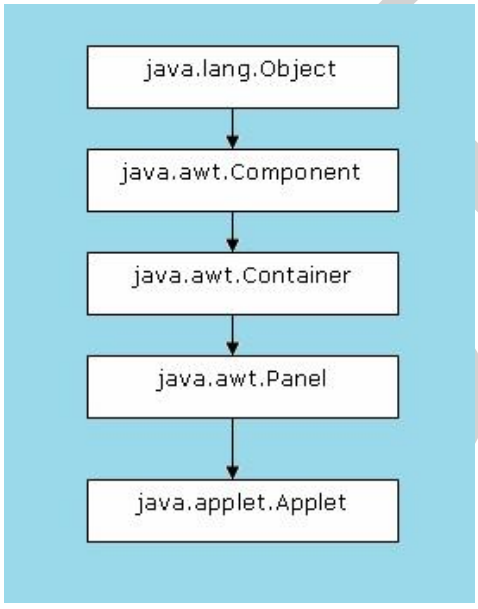
There are many advantages of applets. They are as follows:

- It works at client side, so less response time.
- Secured
- It can be executed by browsers (which has JVM running in it) running under many platforms, including Linux, Windows, Mac Os etc.

Drawback of Applets :-

- Plug-in is required at client browser to execute applet.

Hierarchy of Applet Class:-



The above classes provide support for Java's window-based GUI.

To create an user friendly graphical interface we need to place various components on GUI window. The Component class derives several classes for GUI components. These classed include Check box, Choice, List, Button, and so on.

Applet Class:-

Applet class is defined in java.applet package.

java.applet.Applet is the superclass of all the applets. Thus, all the applets, directly or indirectly, use the methods of Applet class.

It provides all the methods to start, stop, and maintain the applets.

It also has the methods to support multimedia.

Every applet program must have the following two import statements:

import java.awt.;*
import java.applet.;*

Example of Applet Program :-

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

/* <applet code="FirstApplet"    width=200    height=150>
</applet>*/

public class FirstApplet extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString("This is my First Applet", 10, 10);
    }
}
```

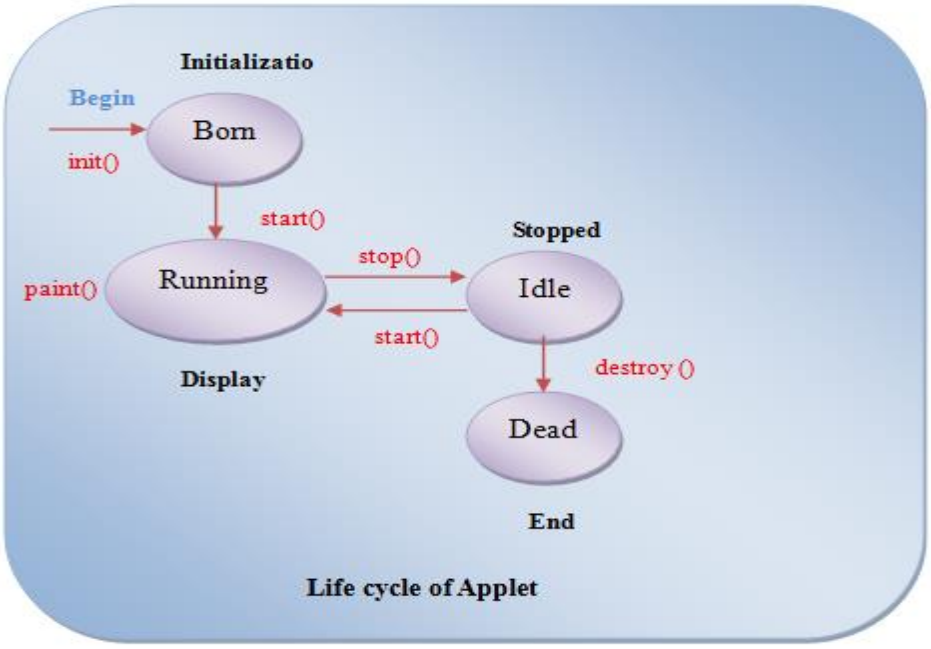
The program draws the string *This is my First Applet* at the pixel position (10, 10).

Q) Explain about Applet Life Cycle in detail.

Applet Life Cycle:-

An applet may move from one state to another during its execution. The following are the 4 states of applet life cycle:

- Born
- Running
- Idle
- Dead



Following methods are responsible for the applet's life cycle:

init( )
start( )

stop( )
destroy( )

Born State: Applet enters this state after init( ) method is called. This is the first method to execute when the applet is executed. Variable declaration and initialization operations are performed.

Running State: Applet moves to this state automatically after the born state. It enters Running state by calling start( ) method. The start() method executes immediately after the *init()* method. The start() method contains the actual code of the applet that should run. It also executes whenever the applet is restored, maximized or moving from one tab to another tab in the browser.

Idle State: Applet goes to Idle state when it is stopped from running. If we leave a web page, applet automatically goes to idle state. Applet can also be stopped by calling stop( ) method. The stop() method stops the execution of the applet. The stop() method executes when the applet is minimized or when moving from one tab to another in the browser.

Dead State: Applet goes to dead state when it is destroyed by calling destroy( ) method. It completely removes the applet from memory. The destroy() method executes when the applet window is closed. *stop()* is always called before destroy() .

Example program that demonstrates the applet life cycle :

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
public void init()
{
                System.out.println("Applet initialized");
}
public void start()
{
                System.out.println("Applet execution started");
}
public void stop()
{
```

```
                System.out.println("Applet execution stopped");
    }
    public void paint(Graphics g)
    {
                System.out.println("Painting...");
    }
    public void destroy()
    {
                System.out.println("Applet destroyed");
    }
    }
```

Q) Explain about Event Handling and types of events.

Or

Explain about Event Delegation Model.

Event Handling / Event Delegation Model:

Changing the state of an object is known as an event.

In event delegation model, a source generates events which are sent to one or more listeners. The source is called event source(event-generating component), and is generated by a component like button, mouse click, etc.

The listeners are responsible for receiving the event. Once received, the events are processed or handled in the way required.

This model has three dimensions, namely events, event sources, and event listeners.

An **event** is an object that describes a state change in the source.

An **event-source** is an object which generates the event.

Event Listeners are the objects that are notified when an event occurs on an event-source.

Types of Events and Listeners:-

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
WindowEvent	WindowListener
FocusEvent	FocusListener

Sample Program : When user clicks on the button "**Ok**", then only the text field shows "**Welcome**". Clicking the button is nothing but generating the event. When this happens, the statements in the actionPerformed( ) method are executed. These statements perform the action to take when the event occurs. Executing a particular methods is nothing but handling the event.

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

/* <applet code="Demo" width=200 height=150>
</applet>*/

public class Demo extends Applet implements ActionListener
{
        Button b;
        TextField  tf;

    public void init ()
    {
         b=new Button("Ok");
         tf=new TextField(15);

        add(tf);
        add(b);
        b.addActionListener(this);
```

```
    }
      public void actionPerformed(ActionEvent e)
      {
         tf.setText("Welcome");
       }
}
```

Q) Explain types of events and event listeners.

The event handling model has three dimensions, namely events, event sources, and event listeners.

An **event** is an object that describes a state change in the source.
An **event-source** is an object which generates the event.
Event Listeners are the objects that are notified when an event occurs on an event-source.

Types of Events and Listeners:-

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
WindowEvent	WindowListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. **Register the component with the Listener**

   Registering a component means adding a matching event listener to the container to receive events from the component.

   For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

2) Implement the method(s) to handle the event

- **Button**

  ActionListener public void actionPerformed(ActionEvent e){ }

- **TextField**

  ActionListener public void actionPerformed(ActionEvent e){ }
  TextListener public void textValueChanged(TextEvent e) { }

- **Checkbox**

  ItemListener public void itemStateChanged(ItemEvent e){ }

- **Choice**

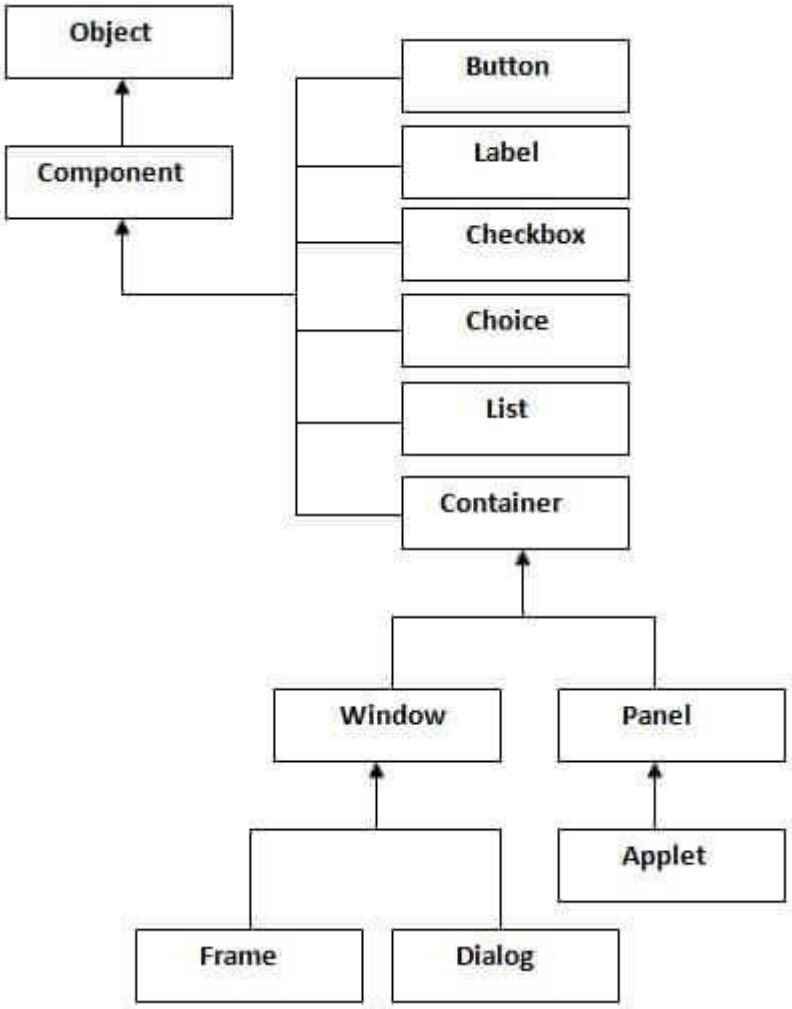
  ItemListener public void itemStateChanged(ItemEvent e){ }

- **List**

ActionListener	public void actionPerformed(ActionEvent e){ }
ItemListener	public void itemStateChanged(ItemEvent e){ }

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

/* <applet  code="Demo"   width=200   height=150>
</applet>*/

public class Demo extends Applet implements ActionListener
{
        Button b;
        TextField  tf;

    public void init ()
    {
         b=new Button("Ok");
         tf=new TextField(15);

        add(tf);
        add(b);
        b.addActionListener(this);
    }
      public void actionPerformed(ActionEvent e)
      {
         tf.setText("Welcome");
       }
}
```

Q) Explain about AWT Components and Containers.



Components:-

A graphical user interface is built of graphical elements called components. Examples are buttons, labels, and text fields. Components allow the user to interact with the program and provide the user with visual feedback about the state of the program. In the AWT, all user interface components are instances of Component class or one of its subclasses.

Containers:-

Whenever we create a graphical user interface with Java AWT or Swing functionality, we will need a container for our application. These containers are Applet, **JFrame**.

Components do not stand alone, but rather are found within containers. Containers contain and control the layout of components. Containers are themselves components, and can thus be placed inside other containers. In the AWT, all containers are instances of Container class or one of its subclasses.

Spatially, components must fit completely within the container that contains them.

Types of containers

The AWT provides four container classes. They are Window, Frame, Dialog, and Panel.

a) **Container**

The Container is a component in AWT that can contain other components like buttons, textfields, labels etc. The classes that extend Container class are Frame, Dialog and Panel.

b)Window

A top-level display surface. An instance of the Window class is not attached to nor embedded within another container. The Window class has no borders, no menu bars, and no title bar. We must use frame, dialog or another window for creating a window.

c)Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

d)Frame

A top-level display surface (a window) with borders and title. It may have menu bars. It can have other components like button, textfield etc.

e)Dialog

A top-level display surface (a window) with borders and titlebar. A Dialog class cannot exist without an instance of the Frame class.

Q) Explain about AWT Button class with an example.

AWT Button class : Button class is used to create a **push button control.** The application will do some action when the button is pushed. It generates an **ActionEvent** when it is clicked. In order to handle a button click event, **ActionListener** interface should be implemented. Buttons can be added to the container like Frame or a Panel.

Constructor	Description
public Button()	Creates a button with no text on it.
public Button(String text)	Creates a button with a text on it.

Steps to use a button:

1) Create an object of Button class using any of its constructors:

```
Button b1=new Button( );

Button b2=new Button("Ok");
```

2) After the button object is created, it needs to be added to the container using the method:

```
add(buttonobjectname);
```

3) If necessary, add Action Listener to the container to receive events from the button:

```
buttonobjectname . addActionListener(this);
```

4) If necessary, implement actionPerformed( ) method:

```
public void actionPerformed(ActionEvent e)
    {
        - - - - -
        - - - - -
    }
```

Sample Program1:

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

/* <applet   code="ButtonDemo"    width=200    height=150> </applet>*/

public class ButtonDemo extends Applet
{
      Button  b;
   public void init ()
   {
       b=new Button("Ok");
      add(b);
 }
}
```

Sample Program2:

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

/* <applet   code="ButtonDemo"    width=200    height=150> </applet>*/

public class ButtonDemo extends Applet implements ActionListener
{
        Button  b;
       TextField   tf;

    public void init ()
    {
         b=new Button("Ok");
        tf=new TextField(15);

        add(tf);
        add(b);
        b.addActionListener(this);
     }

     public void actionPerformed(ActionEvent e)

    {

        tf.setText("Welcome");

     }
}
```

Q) Explain about AWT Label class with an example.

AWT Label class : Label is used to only display text information to the user. It never calls an action method.

Constructors of Label

Constructor	Description
public Label()	Creates a Label with an empty text.
public Label(String text)	Creates a Label with specified text.
public Label(String text, int *alignment*)	Creates a Label with specified text and *alignment*.

Label's *alignment* could be any of these fields -

- **Label.RIGHT**, to align a label to the right.
- **Label.CENTER**, to align a label to the left
- **Label.LEFT**, to align a label to the center.

Sample Program1:

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

/* <applet   code="LabelDemo"    width=200    height=150> </applet>*/

public class LabelDemo extends Applet
{
        Label name, mobno;

         public void init ()
        {
              name=new Label ("Your name");
              mobno =new Label ("Your mobile number",Label.CENTER);

              add(name);
              add(mobno);
         }
}
```

Q) Explain about AWT Checkbox class with an example.

AWT Checkbox class : The Checkbox class is used to create a checkbox. It is used to turn an option **on** (true) or **off** (false). Clicking on a Checkbox changes its state from "**on**" to "**off**" or from "**off**" to "**on**".

Constructors of Checkbox

Constructor	Description
public Checkbox()	Creates a checkbox with no text, this checkbox is unchecked by default..
public Checkbox(String text)	Creates a checkbox with a text, this checkbox is unchecked by default..
public Checkbox(String text,	Creates a checkbox with a text, this checkbox is checked or

boolean b)	unchecked depending on the boolean value.

Some Methods of Checkbox class

Methods	Description
public void setLabel()	Sets a String text on button.
public String getLabel()	Gets the String text of this button.
public void setState(boolean b)	Sets a state of Checkbox.
public boolean getState()	Gets the state of Checkbox.

Sample Program1:

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

/* <applet   code="CheckboxDemo"    width=200    height=150> </applet>*/

public class CheckboxDemo extends Applet
{
        Checkbox chk;

         public void init ()
        {
            chk=new Checkbox(“Do you like Java?”);

            add(chk);
         }
 }
```

Q) Explain about AWT Radio Button or CheckboxGroup class with an example.

AWT CheckboxGroup class : Radio buttons are a special kind of checkboxes. The CheckboxGroup class is used to group together a set of Checkboxs. At a time only one check box button is allowed to be in "on" state and remaining check box buttons in "off" state.

As only a single selection is allowed among many, they are called radio buttons.

The CheckboxGroup class has only one constructor(default constructor) which creates an empty group.

Constructor of Checkbox to use:

Constructor	Description
public Checkbox(String text, CheckBoxGroup cbg, boolean b)	Creates a checkbox with a text, this checkbox is checked or unchecked depending on the boolean value. cbg specifies the radio group this checkbox belongs to.

Some Methods of Checkbox class

Methods	Description
public void setLabel()	Sets a String text on button.
public String getLabel()	Gets the String text of this button.
public void setState(boolean b)	Sets a state of Checkbox.
public boolean getState()	Gets the state of Checkbox.

Sample Program1:

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

/* <applet   code="RadiobuttonDemo"    width=200    height=150> </applet>*/

public class RadiobuttonDemo extends Applet
{

        CheckboxGroup cbg;
        Checkbox chk1;
        Checkbox chk2;

         public void init ()
        {
                cbg = new CheckboxGroup();
                chk1=new Checkbox("M",cbg,true);
                chk2=new Checkbox("F",cbg,false);

                add(chk1);
```

```
            add(chk2);
        }
    }
```

Q) Explain about layout managers and types of layout managers.

Layouts tell Java **where to put components** in containers (`JPanel`, Frame, Applet, etc).

The layout manager automatically positions all the components within the container.

When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

Steps to create a Layout Manager:

Every container has a default layout, but it's better to set the layout explicitly for clarity.

1.Create a new layout object (using one of its constructors)

2.Use the container's `setLayout` method to set the layout.

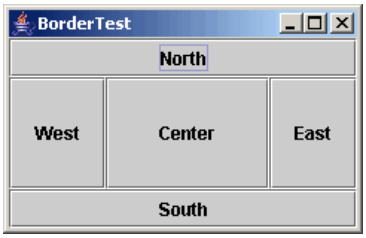
Each layout has its own way to resize components to fit the layout, and we must become familiar with these.

Types of AWT Layout Managers:

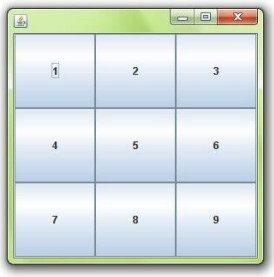
Following is the list of commonly used controls layout managers:

- **FlowLayout** – Arranges the components in a line, left to right, top to bottom. It is the default layout . Good for initial testing. FlowLayout does not respect preferred size information, so it may make variable size elements (eg, a graphics panel) extremely small.

- **BorderLayout** - north, east, south, west, and center areas. Has various rules for how components are stretched to fit these areas. Both common and useful.



- **GridLayout** - equal sized grid elements in the form of a rectangular grid(rows and columns).



- **GridBagLayout** - unequal sized grid. Can produce excellent results, but can be difficult to use.

- **CardLayout** treats each component in the container as a card. Only one card is visible at a time. Tabbed panes are something like a card layout with tabs.

Q) What is a layout manager and explain about FlowLayout with an example.

Layouts tell Java **where to put components** in containers (`JPanel`, Frame, Applet, etc).

The layout manager automatically positions all the components within the container.

When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

Steps to create a Layout Manager:

Every container has a default layout, but it's better to set the layout explicitly for clarity.

1.Create a new layout object (using one of its constructors)

2.Use the container's `setLayout` method to set the layout.

Each layout has its own way to resize components to fit the layout, and we must become familiar with these.

- **FlowLayout** – Arranges the components in a line, left to right, top to bottom. It is the default layout . Good for initial testing. FlowLayout does not respect preferred size information, so it may make variable size elements (eg, a graphics panel) extremely small.

Sample Program using FlowLayout:

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

public class MyFlowLayout
{
    JFrame f;

   MyFlowLayout()
   {
          f=new JFrame();

          JButton b1=new JButton("1");
          JButton b2=new JButton("2");
          JButton b3=new JButton("3");
          JButton b4=new JButton("4");
          JButton b5=new JButton("5");

          f.add(b1);  f.add(b2);  f.add(b3);    f.add(b4);    f.add(b5);

          //setting flow layout of right alignment
          f.setLayout(new FlowLayout(FlowLayout.RIGHT));

          f.setSize(300,300);
          f.setVisible(true);
     }
```

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

Output:-

Q) What is a layout manager and explain about BorderLayout with an example.

Layouts tell Java **where to put components** in containers (`JPanel`, Frame, Applet, etc).

The layout manager automatically positions all the components within the container.

When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

Steps to create a Layout Manager:

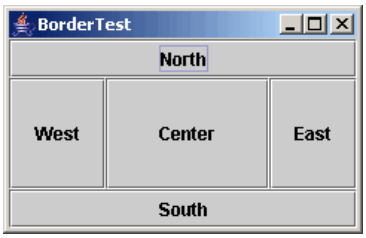
Every container has a default layout, but it's better to set the layout explicitly for clarity.

1.Create a new layout object (using one of its constructors)

2.Use the container's `setLayout` method to set the layout.

Each layout has its own way to resize components to fit the layout, and we must become familiar with these.

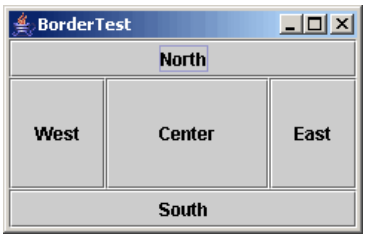
- **BorderLayout** - north, east, south, west, and center areas. Has various rules for how components are stretched to fit these areas. Both common and useful.



Sample Program using BorderLayout:

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

public class Border {
JFrame f;
Border(){
   f=new JFrame();

   JButton b1=new JButton("NORTH");;
   JButton b2=new JButton("SOUTH");;
   JButton b3=new JButton("EAST");;
   JButton b4=new JButton("WEST");;
   JButton b5=new JButton("CENTER");;

   f.add(b1,BorderLayout.NORTH);
   f.add(b2,BorderLayout.SOUTH);
   f.add(b3,BorderLayout.EAST);
   f.add(b4,BorderLayout.WEST);
   f.add(b5,BorderLayout.CENTER);

   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args) {
   new Border();
}
}
```

Output:-



Q) What is a layout manager and explain about GridLayout with an example.

Layouts tell Java **where to put components** in containers (`JPanel`, Frame, Applet, etc).

The layout manager automatically positions all the components within the container.

When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

Steps to create a Layout Manager:

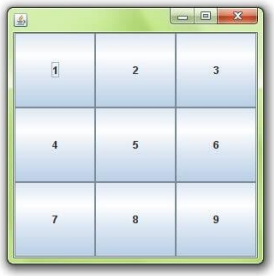
Every container has a default layout, but it's better to set the layout explicitly for clarity.

1.Create a new layout object (using one of its constructors)

2.Use the container's `setLayout` method to set the layout.

Each layout has its own way to resize components to fit the layout, and we must become familiar with these.

- **GridLayout** - equal sized grid elements in the form of a rectangular grid(rows and columns).

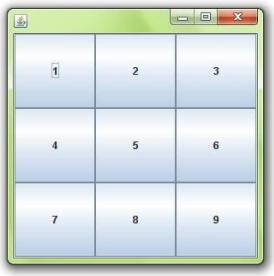


Sample Program using GridLayout:

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

public class MyGridLayout{
```

```
JFrame f;
MyGridLayout(){
   f=new JFrame();

   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");
      JButton b6=new JButton("6");
      JButton b7=new JButton("7");
   JButton b8=new JButton("8");
      JButton b9=new JButton("9");

   f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
   f.add(b6);f.add(b7);f.add(b8);f.add(b9);

   f.setLayout(new GridLayout(3,3));
   //setting grid layout of 3 rows and 3 columns

   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args) {
   new MyGridLayout();
}
}
```

Output:-



Q) Explain about (i) JApplet (ii) JFrame (iii) JPanel

Whenever you create a graphical user interface with Java Swing functionality, you will need a container for your application. In the case of Swing, this container is called a **JApplet, JFrame**, **JPanel.**

(i) JApplet

JApplet is used to create applets using swings.

JApplet is a top-level container class like JFrame.

JApplet extends Applet class, hence all the features of Applet class are available in JApplet.

Every swing-based applet program should create a subclass of JApplet class.

Components are added to JApplet.

Sample Program:-

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

/*
<applet code="JAppletDemo" width="500" height="200">
</applet>
*/

public class JAppletDemo extends JApplet
{
        JButton  b;

        public void init()
        {
                setLayout(new FlowLayout());
                b=new JButton("Ok");
                add(b);
        }
}
```

(ii) JFrame

JFrame is also a top-level container like JApplet to which components are added.

It is a subclass of java.awt.Frame. So, it inherits all the features of Frame.

Sample Program:-

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

public class JFrameDemo extends JFrame
{
        JButton b;
        void print()
        {
                b=new JButton("Ok");
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                add(b);
                setSize(300,200);
                setVisible(true);
        }
        public static void main(String args[])
        {
                JFrameDemo obj=new JFrameDemo();
                obj.print();
        }
}
```

(iii) JPanel

JPanel is a lightweight container used to hold components.

It provides space in which an application can attach any other component.

To use JPanel, simply we need to add its object to its top-level container using the add( ) method.

Sample Code:-

```
import java.awt.*;
import javax.swing.*;
public class JPanelDemo {
```

```
    JPanelDemo()
    {
      JFrame f= new JFrame("Panel Example");
      JPanel panel=new JPanel();
      panel.setBounds(40,80,200,200);
       panel.setBackground(Color.gray);
      JButton b1=new JButton("Ok");
      JButton b2=new JButton("Cancel");
      panel.add(b1);
      panel.add(b2);
      f.add(panel);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
    public static void main(String args[])
    {
            new JPanelDemo();
    }
}
```

Q) Explain about the types of JDBC drivers.

JDBC Driver is a software component that enables java application to interact with the database.

Types of JDBC Drivers:-

There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) **JDBC-ODBC bridge driver :** The JDBC-ODBC bridge driver uses ODBC driver to connect to the database.

The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

Advantages:-

It is readily available.

Disadvantages:-

- Performance is very slow.
- Not platform-independent.
- Not suitable for web applications.

2) Native-API / Partly Java driver : In this driver. JDBC calls the native API driver.

Type 2 driver converts JDBC method calls into native calls of the database API.

Advantages:-

It performance is better than Type 1 driver.

Disadvantages:-

- Libraries need to be installed on client machine.
- Not suitable for web applications.

3)Net-protocol Driver : This driver is used in a network environment. The requests are routed to a middle tier which converts JDBC calls to database-specific calls.

Advantages:-

- Clients are insulated from the database specific libraries.
- It can be used in web applications.

Disadvantages:-

Database specific libraries need to be available in middle tier.

4)Pure Java Driver : This Type 4 driver is used to connect to the database directly. It requires a different driver for every database.

Advantages:-

- It is platform independent.
- Its performance is very good.