Applet:

An applet is a java program that can be embedded into a web page.
It runs inside the web browser and works at client side.
An applet is embedded in an HTML page using the <applet> & <object> tags.
Applets are used to make the website more dynamic and entertaining.

→ An applet is a java class that extends the java.applet.Applet class
→ The main() method is not invoked on an applet.
→ Applets are designed to be embedded within an HTML page.
→ A JVM is required to view an applet. The JVM can be either a plug-in of the web browser & a seperate runtime environment
→ Applets are not stand-alone programs.
→ JDK provides a standard applet viewer tool called applet viewer.
→ Output of an applet window is not performed by System.out.println().
It is handled with AWT methods, such as drawString().

Advantages :-

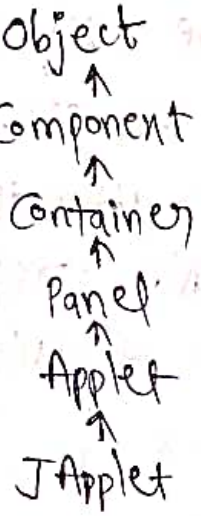
There are many advantages of applet.

→ It works at client side, so less response time

→ secured

→ It can be executed by browsers running under many platforms.

→ Drawback: plugin is required at client browser to execute applet.



* An applet is a window based programming environment.

Life Cycle :-

init() ⇒ applet is initialized
start() ⇒ applet is started
paint() ⇒ applet is painted
} applet begins

stop() ⇒ applet is stopped
destroy() ⇒ applet is destroyed
} applet is terminated.

public void init(): is used to initialize the applet. It is invoked only once.

public void start(): is invoked after the init() method. It is used to start the applet.

public void stop(): It is used to stop the applet.

public void destroy(): It is used to destroy the applet. It is invoked only once.

public void paint(Graphics g): is used to paint the applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

paint - invoked immediately after the start() method and also applet repaints itself in the browser. The paint() method is actually inherited from the java.awt.

There are 2 ways to run an applet
(i) By html file
(ii) By applet viewer tool.

Ex:-
```
/*
<applet code="First.class" width="300"
        height="300">
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
```

```
public class First extends Applet {
  public void paint(Graphics g) {
  g.drawString("Welcome to applet", 150,150);
  }
}
```

Requesting & Repainting :-

The repaint() method is defined by the AWT.
An applet writes its window only when its paint() method is called by the awt.
Calling repaint is essentially a request that your applet be repainted sometime soon.
The AWT will then execute a call to paint(), which can display the stored information.

Ex:-
```
import java.awt.*;
import java.applet.*;
/*
<applet code="StatusWindow" width=300
                height=50> </applet>
*/
public class StatusWindow extends Applet {
public void init()
    {
     setBackground(Color.cyan);
    }
```

```
public void paint (Graphics g)
{
    g.drawString ("This is applet window", 10,20);
    showStatus ("This is status");
}
```

Passing Parameters to Applets :-

Parameters are enable the user to customize the applets operation.

By defining parameters, you can increase your applets flexibility.

You can pass parameters to your applet. Use the <param> attribute in the <applet> tag, specifying the parameters name and value.

To retrieve a parameter, use the getParameter() method, defined by Applet.

Ex:
```
/*
<applet code = "Userparam.class" width = 300
                height = 300>
<param name = "msg" value = "Welcome to applet">
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
public class Userparam extends Applet
{
```

```
public void paint(Graphics g)
{
String str = getParameter("msg");
g.drawString(str, 50, 50);
}
}
```

GUI programming with Swings:-

A swing GUI consists of two key items: Components and containers. All containers are also components.
A container holds a group of components.
A container is a special type of component that is designed to hold other components.
All swing GUI will have at least one container.
Containers are components & a container can also hold other containers.

Components:-

Swing components are derived from the JComponent class.
JComponent inherits the AWT classes Container and component.
All of swing's components are represented by classes defined with in the package javax.swing.
The all swing components are given below,

JApplet	JColorChooser	JDialog
JLayeredPane	JMenuItem	JPopupMenu
JSlider	JTable	JToogleButton
JButton	JComboBox	JEditorPane
JList	JOptionPane	JProgressBar
JSpinner	JTextArea	JToolBar
JCheckbox	JComponent	JPanel

JFrame JViewport JScrollBar
JRootPane JInternalFrame JWindow

Notice that, all component classes begin with the letter 'J'.

Containers:-

swing defines 2 types of containers.

Top-level-containers: JFrame, JApplet, JWindow and JDialog

These containers do not inherit JComponent. They do, inherit the awt classes components and container.

The top-level containers are heavy weight.

A top-level container is not contained within any other container.

Lightweight containers: These inherit JComponent.

Ex:- JPanel

A Lightweight container can be contained within another container.

Java Layout Managers:-

These are used to arrange components in a particular manner. Java Layout Manager facilitates us to control the positioning and size of the components in GUI forms.

LayoutManager is an interface that can be implemented by all the classes of layout managers.

The following are the layout managers,

→ java.awt.BorderLayout,
- FlowLayout, . GridLayout, . CardLayout,
- GridBagLayout, . BoxLayout, . GroupLayout,
- ScrollPanelLayout, . SpringLayout etc.

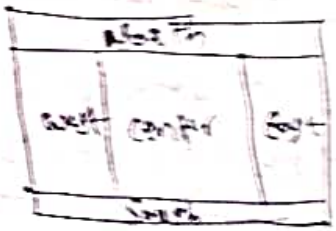
Java BorderLayout :-

It is used to arrange the components in 5 regions North, South, east, west and center.

BorderLayout() : creates a border layout but with no gaps b/w the components.

BorderLayout(int hgap, int vgap).

Eg :-
```
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");
  f.add(b1, BorderLayout.NORTH);
  f.add(b2, BorderLayout.SOUTH);
  f.setSize(300, 300);
  f.setVisible(true);
}
```



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

Java GridLayout :-

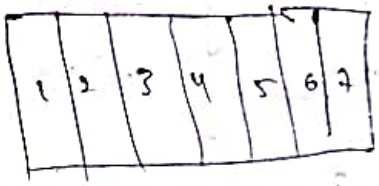
It is used to arrange the components in a rectangular grid.

GridLayout(): creates a grid layout with one column per component in a row.

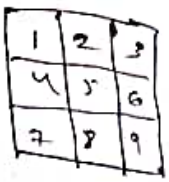
GridLayout(int rows, int columns):

GridLayout(int r, int c, int hgap, int vgap):

Ex-
```
import java.awt.*;
import javax.swing.*;
public class GridLayout {
  JFrame f;
  GridLayout()
  {
    f=new JFrame();
    JButton b1=new JButton("1");
    JButton b2=new JButton("2");
    f.add(b1); f.add(b2);
    f.setLayout(new GridLayout());
    f.setSize(300,300);
    f.setVisible(true);
  }
```



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



if fo.setLayout(new GridLayout(3,3));

Java FlowLayout:-

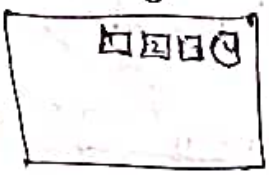
It is used to arrange the components in a line one after another. (Left, Right, Center, Leading, trailing).

FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

FlowLayout(int align);

FlowLayout(int align, int hgap, int vgap);

fo.setLayout(new FlowLayout(FlowLayout.RIGHT));



Java BoxLayout:-

It is used to arrange the components either vertically or horizontally.

X_AXIS, Y_AXIS, LINE_AXIS, PAGE_AXIS

Eg:
```
import java.awt.*;
import java.swing.*;
public class BoxLayout extends Frame{
```

```
Button buttons[];
BoxLayout(){
  {
    buttons = new Button[5];
    for(int i=0; i<5; i++)
      {
        buttons[i] = new Button("Button"+(i+1));
        add(buttons[i]);
      }
    setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
    setSize(400,400);
    setVisible(true);
  }
public static void main(String args[]){
  new BoxLayout();
  }
}
```

Event :-

Change in the state of an object is known as event.
Eg:- Clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.

Types of Event :-

There are broadly categorized into 2 types
(i) Foreground Events
(ii) Background Events

Foreground Events:-
Those events which require the direct interaction of user.
Eg:- Clicking on a button
Moving the mouse
Entering a character through keyboard.
Selecting an item from list.
Scrolling the page etc.

Background Events:-
Those events that require the interaction of end user are known as background events.
Eg:- Operating system interrupts.
Hardware or software failure.
Timer expires etc.

Event Handling:-
Event handling is the mechanism that controls the event and decides what should happen if an event occurs.
This mechanism have the code which is known as event handler.
Source:- The source is an object on which event occurs.
Source is responsible for providing information of the occurred event.
Listener:- It is also known as event handler.
Listener is responsible for generating responce to an event.

The java event classes and Listener interfaces.

Event classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent.	FocusListener.

Button:-
public void addActionListener(ActionListener a){}

MenuItem:-
public void addActionListener(ActionListener a){}

TextField:-
public void addActionListener(ActionListener a){}
public void addTextListener(TextListener a){}

TextArea:-
public void addTextListener(TextListener a){}

Checkbox, Choice:-
public void addItemListener(ItemListener a){}

List:- public void addActionListener(ActionListener a);
public void addItemListener(ItemListener a){}

Eg:-
```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame
{
 TextField tf;
 AEvent()
 {
  tf=new TextField();
  tf.setBounds(60,50,170,20);
  Button b=new Button("Pick me");
  b.setBounds(50,120,80,30);
  b.addActionListener(new ActionListener(){
   public void actionPerformed(){
    tf.setText("Hello");
   }});
  add(b); add(tf);
  setSize(300,300);
  setLayout(null);
  setVisible(true);
 }
public static void main(String args[]){
  new AEvent();
}}
```

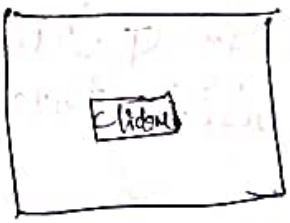
Java Button:-

It is used to create a labeled button that has platform Independent implementation.
It inherits the AbstractButton class.

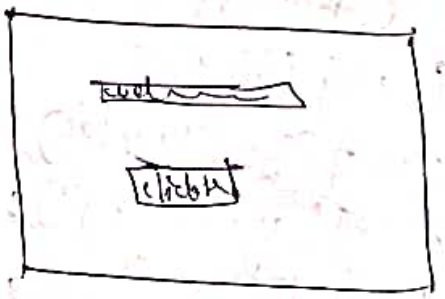
The class JButton is an implementation of a push button.

JButton()
JButton(String s)
JButton(Icon i)

```
import javax.swing.*;
public class ButtonDemo{
  public static void main(String args[]){
    JFrame f=new JFrame("ButtonDemo");
    JButton b=new JButton("Click me");
    b.setBounds(50,100,95,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
  }
}
```



Ex:-
```
import java.awt.event.*;
import javax.swing.*;
public class ButtonDemo {
public static void main(String args[]) {
JFrame f=new JFrame("Button ActionListener");
JTextField tf=new JTextField();
tf.setBounds(50,150,150,20);
JButton b=new JButton("Click Here");
b.setBounds(50,100,95,30);
b.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e){
tf.setText("Welcome to Java");
}
});
f.add(b);
f.add(tf);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```




```
JButton b=new JButton(new ImageIcon("D:\\png.png"));
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

JTextField:-
It is a part of javax.swing package.
It is a component that allows editing of a single line of text.

Ex:-
```
import javax.swing.*;
class TextFieldDemo {
public static void main(String args[])
{
  JFrame f=new JFrame("TextField");
  JTextField t1,t2;
  t1=new JTextField("Welcome");
  t1.setBounds(50,100,200,30);
  t2=new JTextField("AWT Tutorial");
  t2.setBounds(50,150,200,300);
  f.add(t1);
  f.add(t2);
  f.setSize(400,400);
  f.setLayout(null);
  f.setVisible(true);
}}
```

JLabel:-
A display area for a short text string or an image.
A label does not react to input events.
The object of JLabel is a component for placing text in a container.

It is used to display a single line of read only.

Ex:-
```
import javax.swing.*;
class LabelDemo{
public static void main(String args[])
{
JFrame f=new JFrame("LabelDemo");
JLabel l1;
l1=new JLabel("First label");
l1.setBounds(50,50,100,30);
f.add(l1);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
}
```

Image Icon:-
It is an implementation of the icon interface that paints icons from images.

JScrollPane:-
It is used to create scrollable view of a component.
When screen size is limited, we use a scroll pane to display a large component.

JList:-
It represents a list of text items. The user can choose one item or multiple items in JList.

Ex:-
```
import javax.swing.*;
public class ListDemo
{
ListDemo() {
  JFrame f=new JFrame();
  DefaultListModel<String> l=new DefaultListModel<>();
  l.addElement("Item1");
  l.addElement("Item2");
  JList<String> list=new JList<>(l);
  list.setBounds(100,100,75,75);
  f.add(list);
  f.setSize(400,400);
  f.setLayout(null);
  f.setVisible(true);
}
public static void main(String args[])
{
  new ListDemo();
}
}
```

JTable :-

It is a part of java swing package. It is used to display & edit 2-D data that having both rows and columns. This arranges data in a tabular form.

Ex- import javax.swing.*;
public class TableDemo{
JFrame f;
TableDemo(){
f=new JFrame();
String data[][] = {{"1","Girl","1742"},
{"2","Chinna","1472"}};
String column[] = {"ID","Name","Number"};
JTable t=new JTable(data,column);
t.setBounds(30,40,200,300);
JScollPane sp=new JScrollPane(t);
f.add(sp);
f.setSize(300,400);
f.setVisible(true);
}
public static void main(String args[]){
new TableDemo();
}
}

JComboBox:-
A component that combines a button or editable field and drop-down list.
The user can select a value from drop-down list.

```
import javax.swing.*;
public class ComboBoxDemo {
    JFrame f;
    ComboBoxDemo() {
        f=new JFrame("ComboBox Demo");
    String country[]={"India","Aus","USA","England"};
    JComboBox cb=new JComboBox(country);
    cb.setBounds(50,50,90,20);
    f.add(cb);
    f.setLayout(null);
    f.setSize(400,500);
    f.setVisible(true);
    }
    public static void main(String args[]) {
        new ComboBoxDemo();
    }
}
```

JTree:

It is used to display the tree structured data.
It is a complex component.
It has a root node at the top most which is a parent for all nodes in the tree.

Eg:-
```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeDemo {
 JFrame f;
 TreeDemo() {
  f=new JFrame();
  DefaultMutableTreeNode style=new DMTN("style");
  DMTN color=new DMTN("color");
  style.add(color);
  DMTN red=new DMTN("red");
  DMTN blue=new DMTN("blue");
  color.add(red);
  color.add(blue);
  JTree jt=new JTree(style);
  f.add(jt);
  f.setSize(200,200);
  f.setVisible(true);
 }
 public static void main(String args[]){
  new TreeDemo();
 }
}
```

JOptionPane:-

It is a subclass of JComponent, which includes static methods for creating and customizing modal dialogboxes using simple code.

It is used to display 4 types of dialog boxes.

Message Dialog :- It displays a message, it possible to add icons to alert the user.

Confirm Dialog :- This besides sending a message, enables the user to answer question.

Input Dialog :- It besides sending a message, allows entry of a text.

OptionDialog :- That covers the 3 previous types.

These dialog boxes are used to display information (or get input from the user.

showInputDialog : Prompt user for some input.

showMessageDialog : Shows the dialog box with a message to the user.

showConfirmDialog : Asks user a confirmation question, like yes, no, cancel.

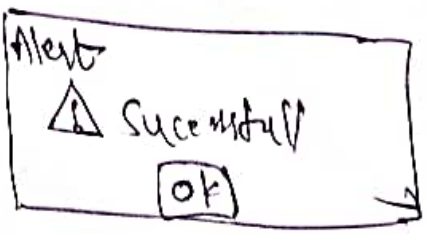
showOptionDialog : The combination of the above three.

showMessageDialog:-

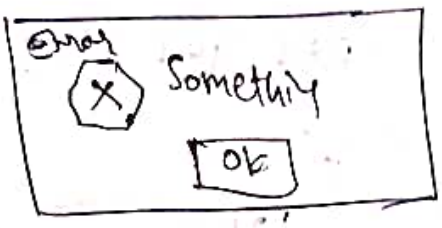
```
import javax.swing.JOptionPane;
public class Main extends JOptionPane.
{
public static void main (String args[])
{
JOptionPane.showMessageDialog (null, "Hello World");
}
}
```

Eg:-

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

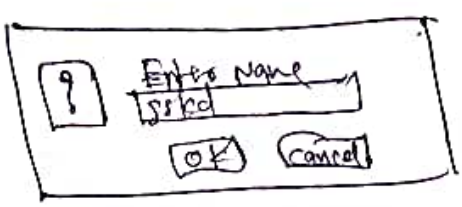


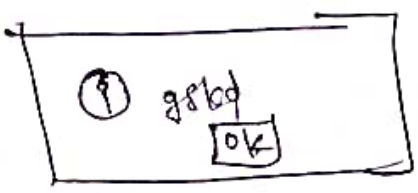
```
import javax.swing.JOptionPane;
public class OptionPane extends JOptionPane
{
public static void main(String args[])
{
JOptionPane.showMessageDialog(null,"Something",
                "Error", JOptionPane.ERROR_MESSAGE);
}
}
```



showInputDialog:-

```
import javax.swing.*;
public class OptionPane{
JFrame f;
OptionPane(){
   f=new JFrame();
   String name=JOptionPane.showInputDialog(f,
          "Enter Name");
}
public static void main(String args[]){
 new OptionPane();
}
```





showConfirmDialog :-

```
import javax.swing.JOptionPane;
public class JOptionPaneDemo extends JOptionPane
{
  public static void main(String args[])
  {
  int Input = JOptionPane.showConfirmDialog(null,
                                "Are you sure");
  SOP(input);
  }
}
```