Multithreading is a multitasking operation where the different independe task of a single program gets executed by a processor.
Here the thread execution is simultaneous and there is no certain pattern of output.

Thread Scheduler takes the responsibility of thread execution.
Here always we get mixed output.

How to create a thread

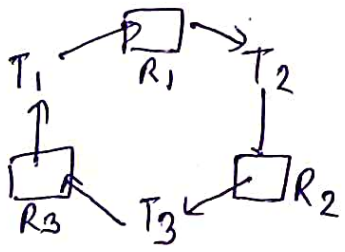
1. By extending Thread class
2. By implementing Runnable Interface

* public void run() :- It is inside thread class.
  - It must be overridden during the thread creation.
  - The ~~portions~~ independent portions of the program must be written inside a run method.

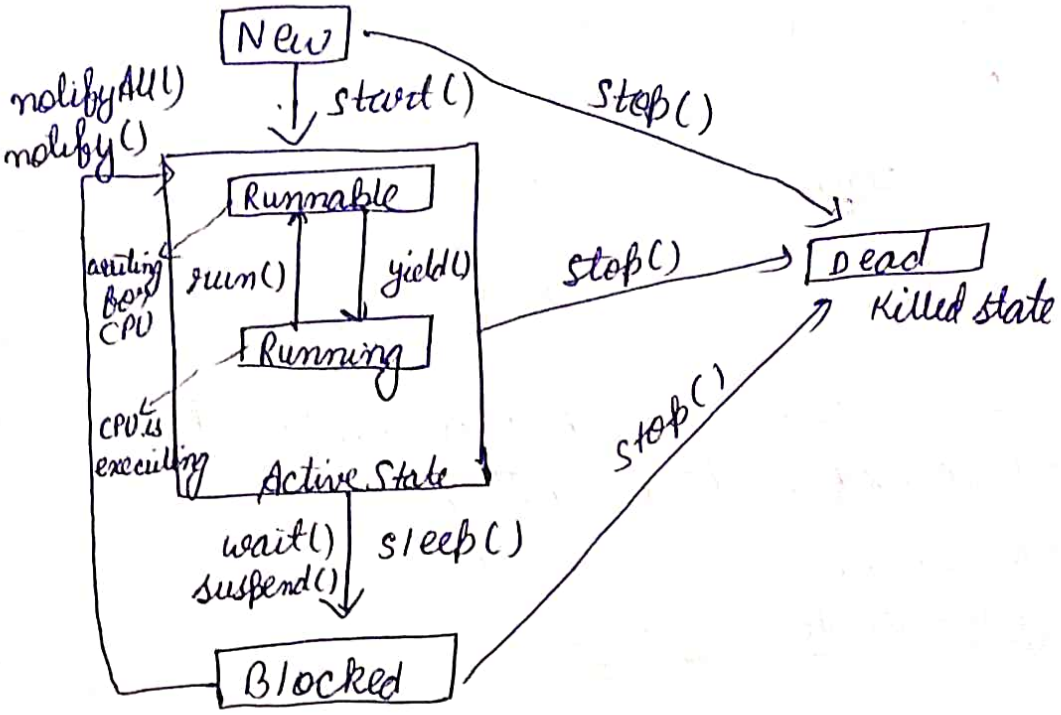
* start() :- It must be called by the thread object to create one active thread.

The run method is never called in the program, it is called by default inside the start method.

* Thread Life Cycle



DeadLock
Starvation

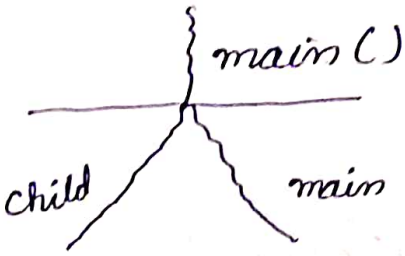


join() :- Interthread communication

* class ChildThread extends Thread
{
    public void run()
    {
        for(int i=0; i<3; i++)
        {
            SOPln("ChildThread");
        }
    }

class MainThread {
    PSVM(String args[])
    {
        for(int i=0; i<3; i++)
        {
            SOPln("MainThread");
        }
    }
}

ChildThread ct = new ChildThread(
ct.start();



* when we call run(), then fix output is displayed & multithreading will be not multithreading.

Create Thread by implementing Runnable Interface:-

- Thread class also implements Runnable interface

```
class MyThread implements Runnable
 {
     public void run()
     {
         for(int i=0;i<3;i++)
           SOPln("MyThread");
     }
 }
 class ThreadDemo
 {
     Public static void main(String[]args)
     {
         MyThread mt = new MyThread();
         Thread t = new Thread(mt);
         t.start();
     }
 }
```

Constructors of thread class :-

① Thread t = new Thread ();

② Thread t = new Thread (Runnable r);

③ Thread t = new Thread (String name); // To give name of Thread

④ Thread t = new Thread (Runnable r, String name);

⑤ Thread t = new Thread (ThreadGroup g);

⑥ Thread t = new Thread (Runnable r, ThreadGroup g);

⑦ Thread t = new Thread (Runnable r, ThreadGroup g, String name);

⑧ Thread t = new Thread (Runnable r, ThreadGroup g, String name, int stackSize);

Getting & setting name of Thread :-

```
class Demo
{
    psvm (String args[])
    {
        MyThread mt = new MyThread();
        MyThread mt1 = new MyThread();
        mt.start();
        S.O.Pln(Thread.currentThread.
        mt1.start();
        S.O.Pln(Thread.currentThread().getName()); // Main Thread
        S.O.Pln(mt.getName()); // Thread 0
        S.O.Pln(mt1.getName()); // Thread 1
        Thread.currentThread().setName("BIT Sindri");
        S.O.Pln(Thread.currentThread().getName()); // BIT Sindri
    }
}
```

* getName();

Public final String getName();

* setName();

public final void setName(String name);

* Thread Priority :-

Thread Scheduler is a device which assigns priority.

Value :- 1 to 10

Thread.Max

Thread.MAX_PRIORITY = 10
Thread.NORM_PRIORITY = 5
Thread.MIN_PRIORITY = 1

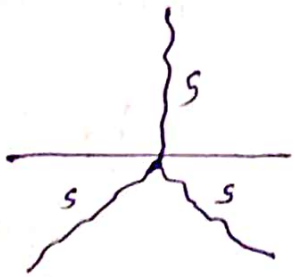
- setPriority (int val) :- public final void setPriority (int val);
- getPriority () :- public final int getPriority ();

(1)

```
class Demo
{
    psvm(String args[])
    {
        MyThread mt = new MyThread();
        mt.start();
        SOP(Thread.currentThread().getPriority());
        for(int i=0; i<3; i++)
        {
            SOP("Main Thread");
        }
    }
```

(2)

```
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0; i<3; i++)
        {
            SOP("Child Thread");
        }
    }
}
```



- By default, Main thread priority is 5
  Child thread priority is inherited from parent thread

→ mt.setPriority(10);
→ Thread.currentThread().setPriority(20); // RE : IllegalArgumentException

Note :- Even if you use change priority we use will get mixed output.

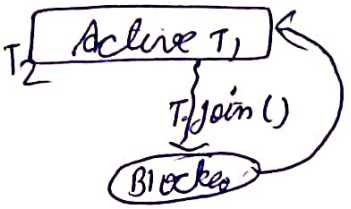
* Thread Execution get affected by
a) yield()
b) join()
c) sleep()

a) yield():- The thread invoking the yield method temporarily join the runnable state but never get blocked. Here, the execution of thread either in active or runnable state are unpredictable. The Thread that are in active or runnable state are having same priority.
In this case, we will get the mixed output.

```
class MyThread extends Thread {
    public void run()
    {
        for(int i=0;i<3;i++)
        {
            SOPln("My Thread");
        }
    }
}
class Demo
{
    PSVm(String args[])
    {
        MyThread mt = new MyThread();
        mt.start();
        Thread.currentThread().yield();
        for(i=0;i<3;i++)
        {
            SOPln("Demo");
        }
    }
}
```

Signature:- public static native void yield();

b) join():-



after completion of T_2 thread, T_1 will invoke
after certain time/ with interrupti

Signature:- public final void join(); // T_1 will wait until T_2 complete
public final void join(long ms);
public final void join (long long ms, int ns); // T_1 will wait for certain time

* It throws interrupted exception

public final void join () throws InterruptedException;

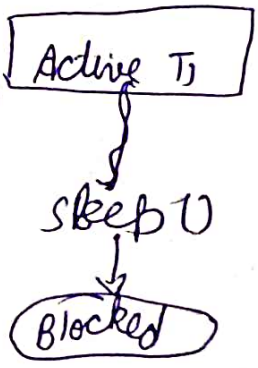
The caller that is using join method must declare InterruptedException or handle it.

```
class Demo
{ psvm( String args[] ) throws InterruptedException
  {
    MyThread mt = new MyThread();
    mt.start();                    Thread-0
    mt.join();     // Thread-0
    for (int i=0; i<3; i++)
      { s.o.println("Demo");
      }
  }
}
```

* ~~Thread.count~~ Thread.currentThread().join(); // infinite wait

c) sleep():-

public ~~final~~ static native void sleep(long ms) throws IE;
public static void sleep(long ms, int ns) throws IE;



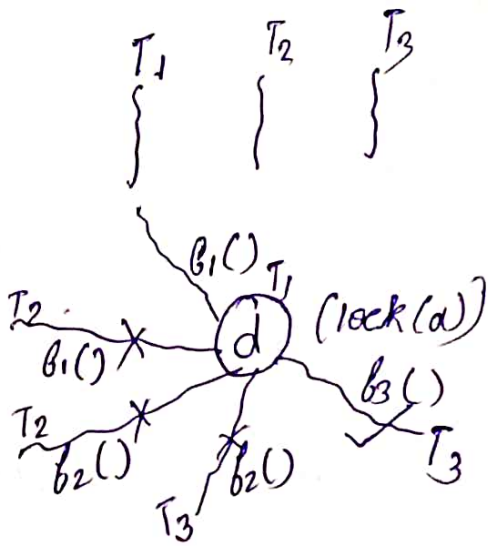
- The caller that is using sleep method must declare or handle InterruptedException

```
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<3;i++)
        {   S.O.Pln("MyThread");
        try { Thread.sleep(10000);
            }
        catch (InterruptedException e)
        {
        }
    }
}
class Demo
{ psvm(String args[]) throws InterruptedException
    {
      MyThread mt = new MyThread();
      mt.start();
     for(int i = 0;i<3;i++)
       {S.O.Pln("Demo");
       }
    }
}
```

Synchronization:- It is a concept used in java to restrict the multiple threads accessing a single resource, to avoid data inconsistency.

- Here, at a time only 1 thread can access the java object.
- In java, the keyword 'synchronized' is used to make synchronization.
- Generally a method or a block is made synchronized.
- Synchronization avoids data inconsistency while doing updation or deletion operation.

```
Class Demo
{
    synchronized void f1()
    {
    }
    synchronized void f2()
    {
    }
    synchronized void f3()
    {
    }
}
```

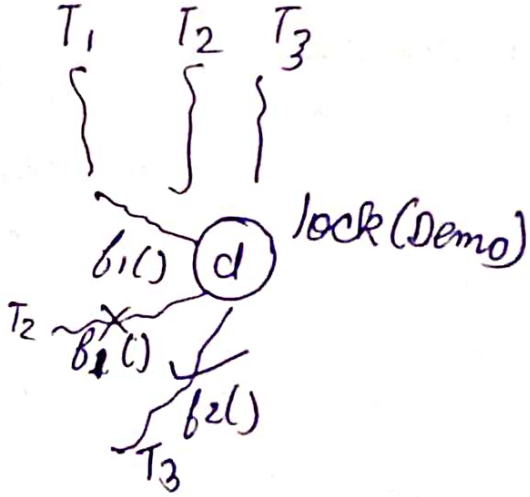


In java, two types The concept of synchronization works with lock concept In java, two types of locks are there:-

(I) Object level lock:- Each java object are having unique lock, but it comes onto effectiveness when synchronized keyword is used as a modifier.

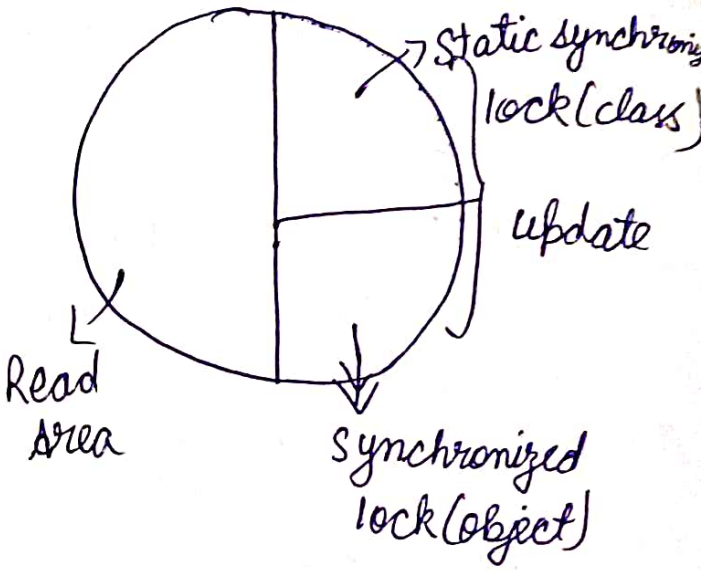
If a thread wants to access a synchronized block or method the thread has to hold the lock over the object. The other threads cannot access any of the synchronized method or block until the lock is not released by previous thread.

(II) Class level lock:-



- When the method or block of a class is not only synchronized but static, there is the need of class level lock.
- Every class is having one unique lock which comes to picture when "static synchronized" is modifier is used.
- If one thread is accessing a static synchronized block or method by holding class level lock, at that time no other thread can access the same block or method until the lock is released.

```
class Demo
{
  static synchronized void f1()
                  {
                  }
      synchronized void f2()
                  {
                  }
        void f1()
          {
          }
  }
```




```
class Resource
{
    void Name(String name)
    {
       for (int i=0;i<3;i++)
          {
             System.out.println("Resource Name is "+name);
          }
    }
}
class MyThread extends Thread
{
      String name;
      MyThread mt = new My MyThread();
```

Event Handling:-

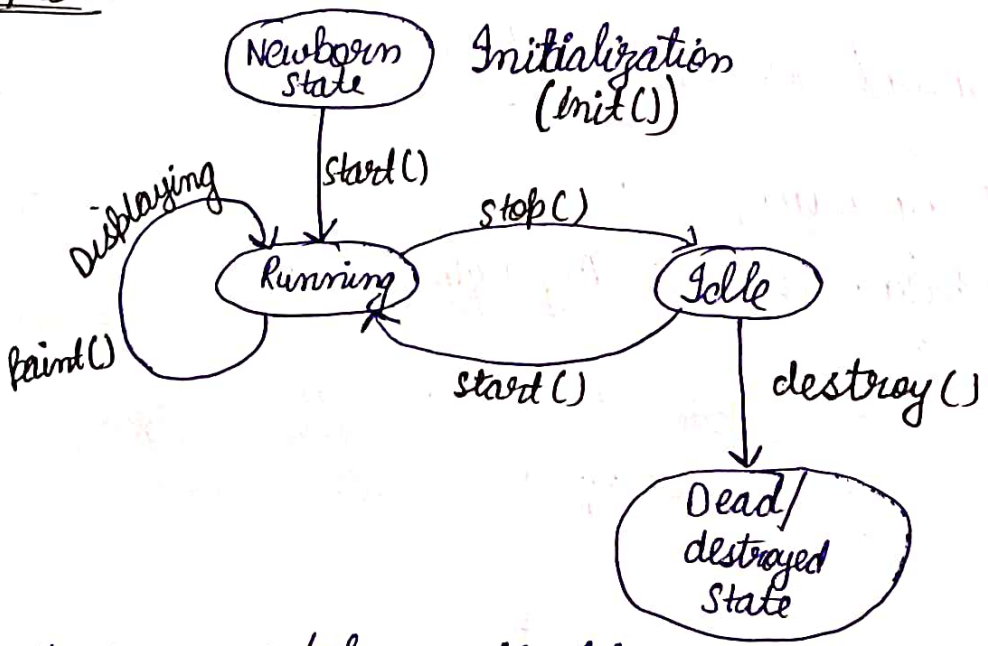
Component :- java.awt.Component
Container :- java.awt.Container { canvas, Applet, Frame, Window

Applet :- (class) It is a container ~~class~~ which holds different components used as client side application. Applets are the small applications used as client application.
Applets are not having any main method.

Application	Applet
These are having main method	No main method
Requires JRE	Required web browser
Does not require any security	Required to high level of security
having main method as call back method	having an init method as call back method
Does not incoporate with HTML tags	Incorporate with HTML tags
Constructor used for initialization	Init method is used for initialization

call back method: method which are not called by programmer it is automatically called.

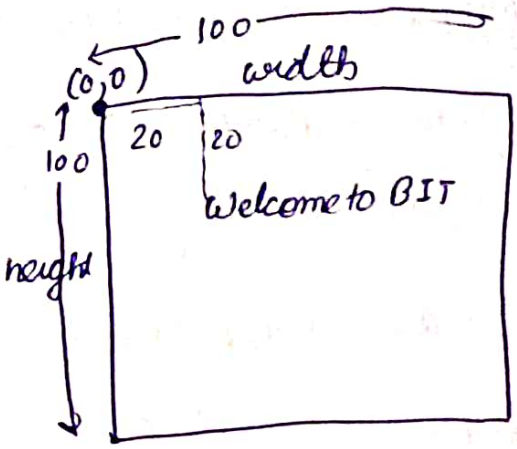
Life Cycle of Applet:-



- In Application, there is no need of any explicit package
- In Applet, two package ~~are~~ are required ~~are~~ awt, applet.

```
public void init()
{
}
public void paint(Graphics g)
{
}
```

applet package, awt package



Program (Demo.java)

```
import java.awt.*;
import java.applet.*;
class Demo extends Applet
{
    String s;
    public void init()
    {
        s = "Welcome to BIT";
    }
    public void paint(Graphics g)
    {
        g.drawString(s,20,20);
    }
    public void destroy()
    {
        s = null;
    }
}
/*<applet code="Demo" width=100 height=100> </applet>*/   // for Applet viewer
```

Running Applet

way1: Web browser

way2: Applet viewer

Demo.html

```
<html>
<body>
<applet code="Demo" width=100 height=100>
</applet>
</body>
</html>
```

Compile Demo.java → Demo.class

(open) → Demo.html

} for web browser

>javac Demo.java ↵
>appletviewer Demo.html ↵ } for applet viewer

Sending Parameters from html to to Applet.

Ogetka String getParameter(String)

Q Create one applet where get two numbers and show the addition result.

```
import java.awt.*;
import java.applet.*;
public class Demo extends Applet {
        int x, y;
        String S1, S2;
    public void init()
    {
        S1 = getParameter("First Number");
        S2 = getParameter("Second Number");
        x = Integer.parseInt(S1);
        y = Integer.parseInt(S2);
    }
    public void paint(Graphics g)
    {
        g.drawString("Enter First No: "+x, 50, 30);
        g.drawString("Enter Second No: "+y, 50, 50);
        g.drawString("Addition Result: "+(x+y), 50, 60);
        showStatus("This is Addition");
    }
/*<applet code=Demo.class width="200" height="200">
 <param name="First Number" value="20">
 <param name="Second Number" value="30">
 </applet> */
```

Banner Program Using Applet

```
import java.awt.*;
import java.applet.*;
public class Demo extends Applet implements Runnable {
        Thread t= null;
        String s =null;
    public void init ()
    {
        S= "Welcome to BIT";
        t= new Thread(this);
    }
    public void run()
    {   char c;
        while(true)
        {
            try{
              repaint();
              Thread.sleep(100);
              ch =s.charAt(0);
              S= substring(1, s.length()); // ch=s.charAt(s.length)
                                           // S=substring (0,s.length(
                                           // S=
              S+= ch;
        } catch (InterruptedExcepthone)
    public void {
                    }
    public void paint (Graphics g)
      {
          g.drawString(s,50,50);
          showstatus("Banner Program");
      }
}
```

Java.lang

→ Object :- Base class of all predefined or undefined class directly or indirectly

12 methods → (11 + 1)

- → toString()
- → hashCode()
- → equals()
- → clone()
- → finalize()
- → getClass()
- → wait()
- → wait(long ms)
- → wait(long ms, int ns)
- → notify()
- → notifyAll()

① registerNatives() used by object itself → private class

```
public String toString()
public int hashCode()
public boolean equals(Object)
protected native Object clone() throws CloneNotSupportedException
protected void finalize() throws Throwable
public final Class getClass()
public final void wait() throws InterruptedException
public native final void wait(long ms) throws InterruptedException
public final void wait(long ms, int ns) throws InterruptedException
public final native void notify()
public final native void notifyAll()
```

join()	wait()
one thread waits for another thread to complete	one threads waits for another object get updated.

The wait method is exactly same as the join method. In join method one thread waits for another thread to complete get executed to complete. But in wait() execution, one thread waits for another thread until the notification.

```
public String toString(){
    return getClass().getName() + "@" + toHexValue(hashCode());
}
```

equals()

```
class Student {
  String name;
  int roll;
  Student(String name, int roll)
    {
      this.name = name
      this.roll = roll;
    }
  public static void main(String arg[])
    {
      Student S1 = new Student ("Somesh", 101);
      Student S2 = new Student ("Rahul", 102);
      Student S3 = new Student ("Somesh", 101);
      System.out.println(S1 = S2);
      System.out.println(S1.equals(S2));
      System.out.println(S1.equals(S3));
      System.out.println(S2.equals(S3));
```

```
public boolean equals(Object obj)
{ String name1 = this.name;
  int roll1 = this.roll;
  String name2 = obj.name;
  String roll2 = obj.roll;
    if (name1.equals(name2) &&
        (roll1 == roll2)
      { return true; }
    else { return false; }
}
```

```
false
false
false  // true
false
```

String

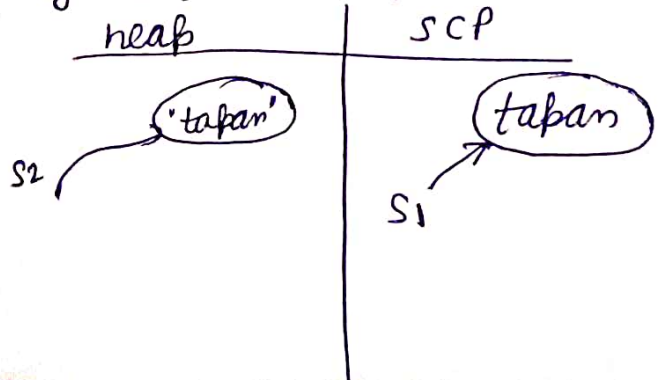
→ immutable, if we try to change the string value with the changed value new object will be created

```
String S1 = "tapan";
String S2 = new String ("tapan");
```

} → Total 2 object created one in heap one in SCP

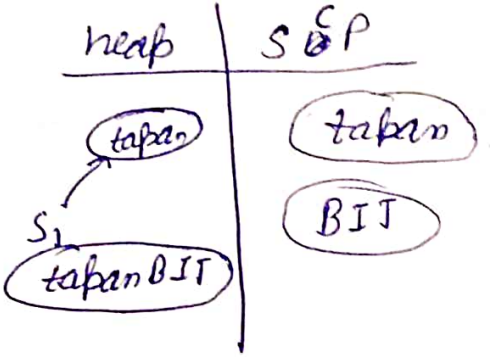
① String const (S1) ⟶ String constant pool

② String Object (S2) ⟶ heap area.



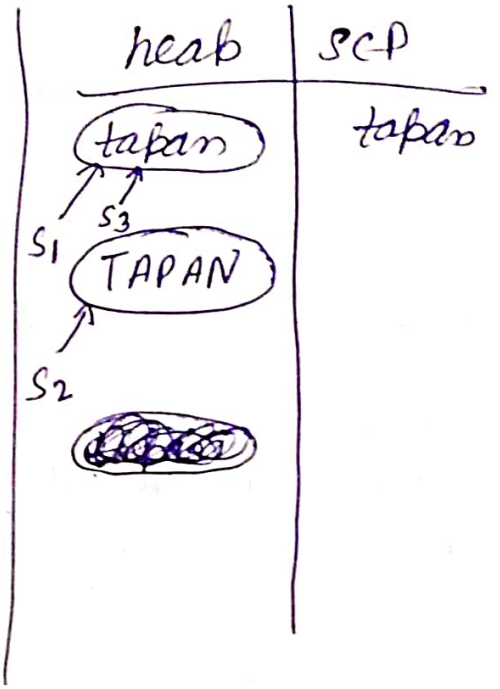
```
String S1 = new String ("tapan");
S1.concat ("BIT");
SOPln (S1);
```

4 object



tapan

```
String S1 = new String ("tapan");
String S2 = S1.toUpperCase();
String S3 = S1.toLowerCase();
SOPln (S1);
SOPln (S2);
SOPln (S3);
```

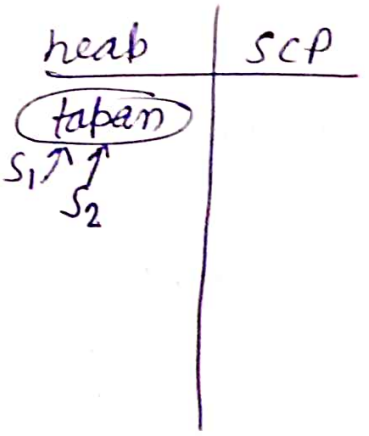


Object = ?

2 in heap
1 in SCP

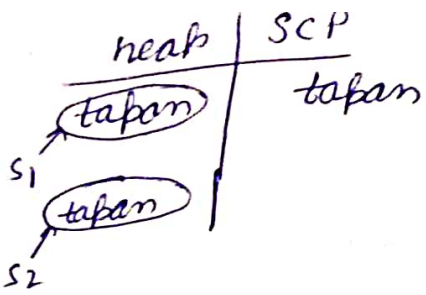
tapan TAPAN tapan

```
byte b[] = {116, 97, 112, 97, 110};
String S1 = new String (b);
String S2 = S1.replace ('c', 'k');
SOPln (S1);
SOPln (S2);
```



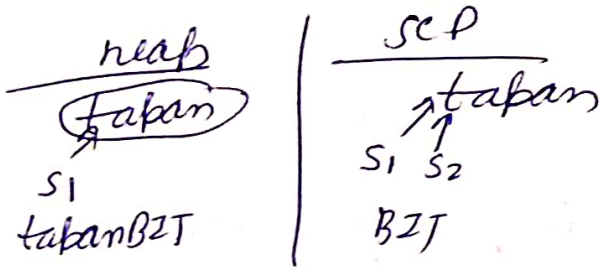
tapan tapan

String s1 = new String("tapan");
String s2 = new String("tapan");

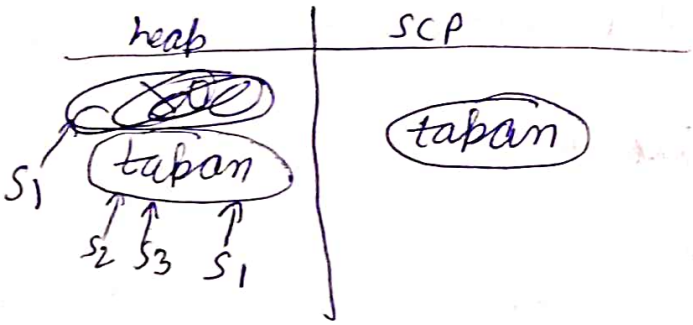


* new operator must create an object in heap area.

String s1 = new String("tapan");
String s2 = "tapan";
String s3 = s2.concat("BIT");

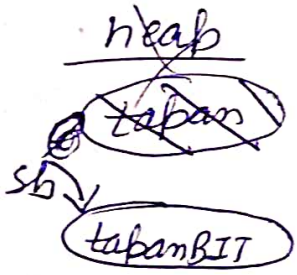


String s1 = new String();
String s2 = s1.concat("tapan");
String s3 = s2.trim();
SOPln(s3.length()); // 5



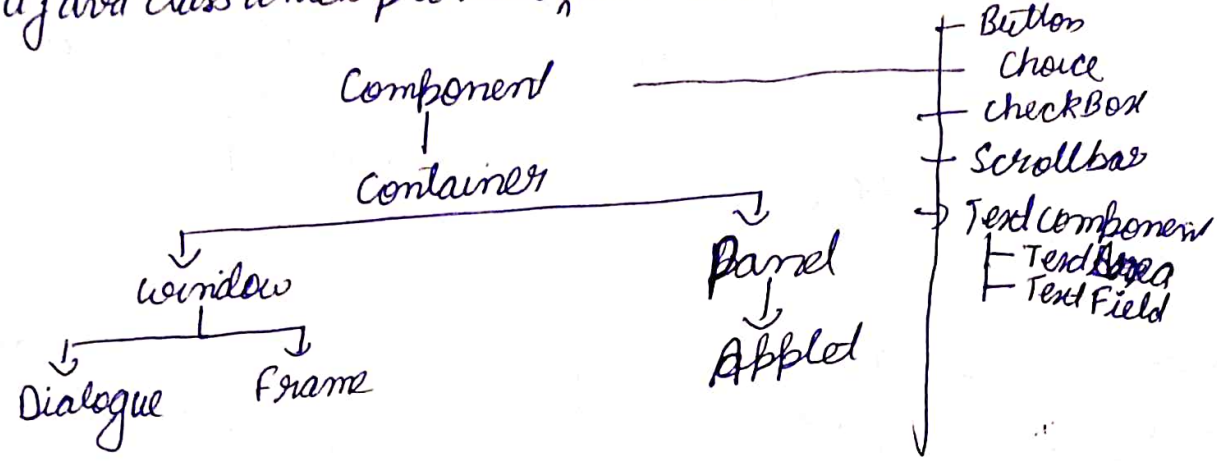
String Buffer :- mutable (~~There is no case of SCP~~)

StringBuffer sb = new StringBuffer("tapan");
~~sb.append("tapan");~~ SOPln(sb.capacity());
sb.~~abpd~~(append("BIT");



StringBuffer sb = new StringBuffer("tapan");
SOPln(sb.capacity()); 21 (16+5)
sb.append("BIT");
SOPln(sb.capacity()); 21
sb.append("Jharkhand");
SOPln(sb.capacity()); 21

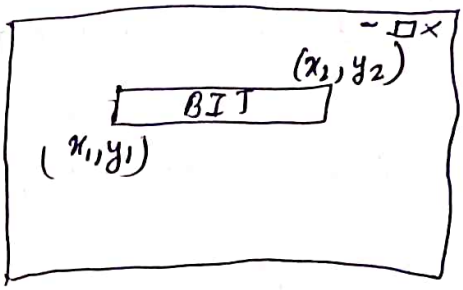
Frame:- It is a java class which provides support for GUI based application.



Layout for placing components on Container

① ~~Pixelformat~~ PixelFormat ② PositionFormat

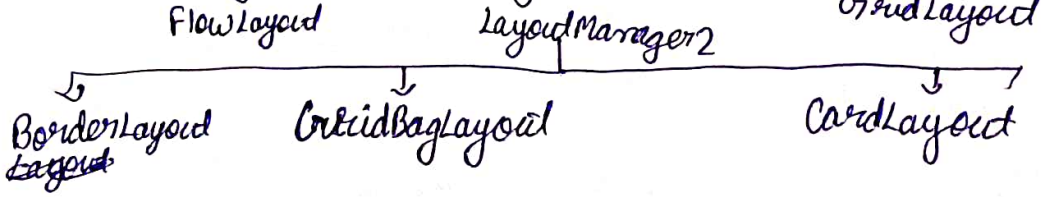
① PixelFormat



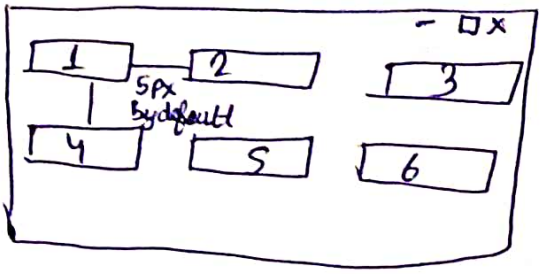

```
import java.awt.*;
Public class DemoPixelformat extends Frame{
        public DemoPixelformat(){
         setLayout(null);  → setSize(200,200);
        Button btn = new Button ("BIT");
        btn.setBounds(70, 50, 100, 120);
                      x1  y1  x2  y2
        add(btn);
        setVisible(true);
    }
public static void main(String args[]){
        new DemoPixelformat();
    }
}
```

② Position format → FlowLayout, BorderLayout, GridLayout, CardLayout, GridBagLayout

LayoutManager (class) java.awt



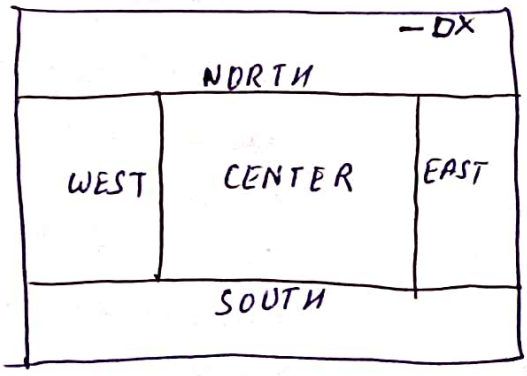
1) Flay FlowLayout :-



In this components are added from left to right on the top on the base component and by default the horizontal & vertical gaps is 5 px. It is the default layout manager for Applet, Frame.

```
import java.awt.*;
public class DemoFlowLayout extends Frame {
    public DemoFlowLayout () {
        setLayout (new FlowLayout());
        for (int i=0; i<6; i++)
        {
            add (new Button ("BIT + (i+1)));
        }
        setSize(200, 200);
        setVisible (true);
    }
    psvm(String args[]){
        new DemoFlowLayout ();
    }
}
```

2) BorderLayout:- 5 regions = 4 boundaries + 1 center



→ 5 components can be added on borderlayout
→ default gap = 0 px
→ default layout manager for window, panel, dialog dialogue.
→ For scroll layout, borderlayout is best.
→ BorderLayout.NORTH
→ BorderLayout.SOUTH
BorderLayout.EAST
BorderLayout.WEST
BorderLayout.CENTER

```
setLayout (new BorderLayout (5,5));
Button btn1= new Button ("BIT1");
Button btn2= new Button ("BIT2");
   btn1.add (BorderLayout.NORTH);
   btn2.add (BorderLayout.WEST);
   SetVisible (true);
   SetSize (200,200);
}
```

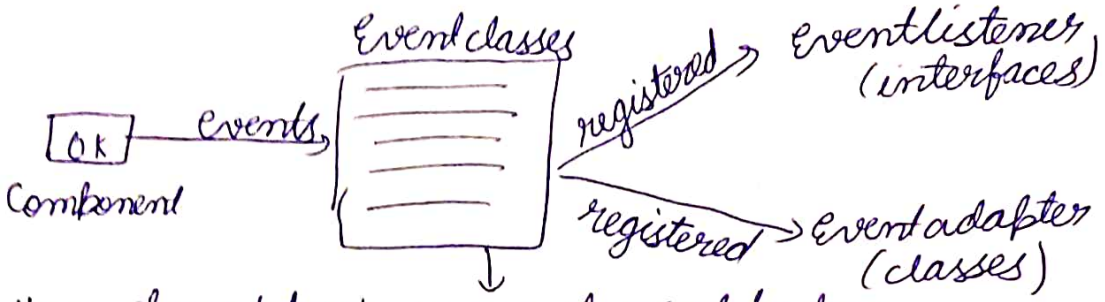
Q. Write a java program to create one GUI based application with Frame as base component; background color as red, 10 buttons each one of different colour and the name of buttons with different font size

```
import java.awt.*;
class Demo extends Frame{
      Demo ()
      {
        setLayout (null);
        for(int i=1; i<=10; i++)
        setBackground (Color.red);
        SetSize (200,200);
        setVisible ("true");
        for (int i=1; i<=10; i++)
        {
          add (new Button ("BIT"+i);
        } Button btn1= new Button ("BIT 1");
          Button btn2= new Button ("BIT 2");
          Button btn3 = new Button ("BIT 3");
          Button btn4= new Button ("BIT 4");
          Button btn5 = new Button ("BIT 5");
          Button btn 6= new Button ("BIT 6");
          Button btn7 = new Button ("BIT 7");
          Button btn8 = new Button ("BIT 8");
          Button btn9= new Button ("BIT 9");
          Button btn10= new Button ("BIT 10");
            btn1.setBackground (color.green);
            btn2.setBackground (color.orange);
            btn3.setBackground (color.yellow);
            btn4.setBackground (color.gray);
```

btn5.setBackground(Color

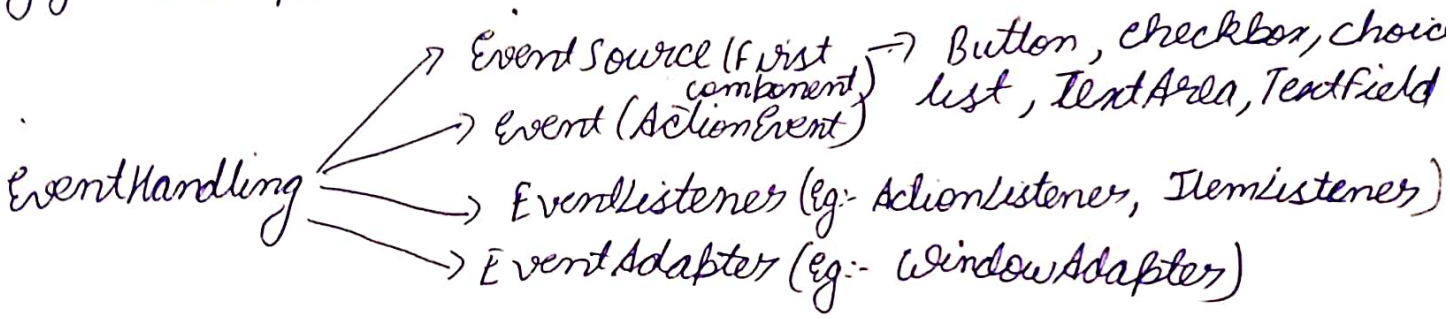
Event Handling

* Events



→ All these classes belong to java.awt.event package.

* Event Handling:- It is a process of responding the event components after they get acted upon.

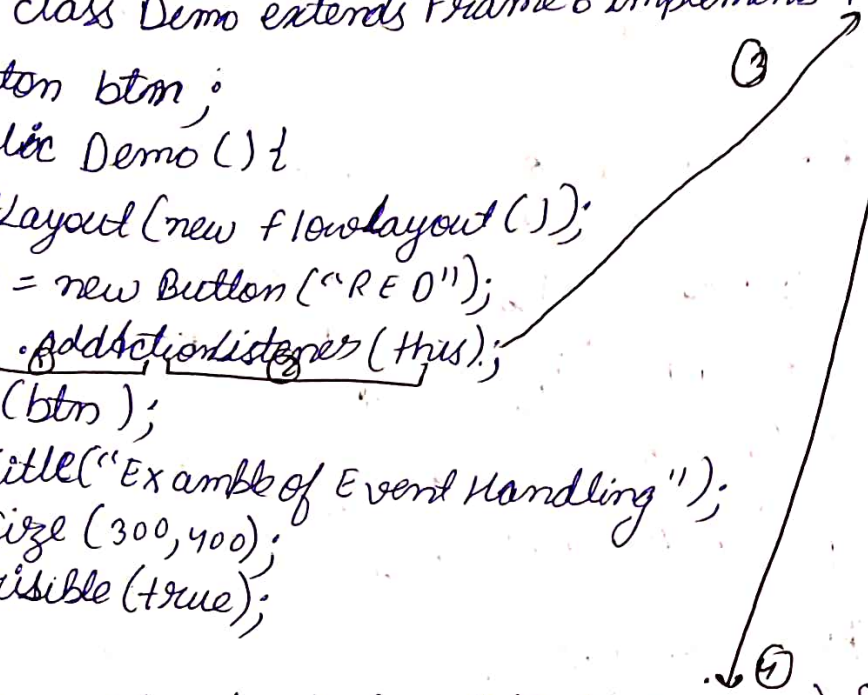


Component	Event	Listeners
Button, TextField, List	ActionEvent	ActionListener → actionPerformed()
Choice, CheckBox	ItemEvent	ItemListener
Scrollbar	AdjustmentEvent	AdjustmentListener
Frame	WindowEvent	WindowListener

Event Adapter:- Some of the event listeners are having more than one methods. All these methods are abstract methods as Listeners are interfaces. To use one of them we have to override all of them the methods. To handle this problem the corresponding classes of those listeners are available known as event adapters

Event Handling Process:- Event Delegation Process

```
import java.awt.*;
import java.awt.event.*;
public class Demo extends Frame implements ActionListener {
    Button btn;
    Public Demo(){
     setLayout(new FlowLayout());
     btn = new Button("RED");
     btn.addActionListener(this);
    add(btn);
    SetTitle("Example of Event Handling");
    setSize(300,400);
    setVisible(true);
    }
public void actionPerformed(ActionEvent e){
        String s= e.getText(); // getActionCommand();
        if(s.equals("RED")
          {
            setBackground(Color.red);
        }
    }
public static void main(String args[])
    {  new Demo();
}}
```



Button
↓ generates
Action Event
↓ Registered
ActionListener having actionPerformed(ActionEvent)

```
import java.awt.*;
import java.awt.event.*;
public class Demo extends Frame implements ActionListener{
   Label lb1, lb2, lb3;
   TextField Tf1, Tf2, Tf3;
   public Demo (){
    setLayout(new FlowLayout());
    lb1 = new Label("User id");
    lb2= new Label("Password");
    lb3= new Label("Result");
    Tf1 = new TextField(8);
    Tf2= new Textfield(5);
    Tf3= new TextField(10);
    Tf2.addActionListener(this);
    Tf2.setEchoChar('*');
   setVisible(true);
   setSize(300,400);
}
public void actionPerformed(ActionEvent e){
     String S1= e.getText();
    String S2=Tf1.getText();
   if(S1.equals("BIT")&& S2.equals("CSEIT"));
      Tf3.setText("VALID");
   else
      { Tf3.setText("NOT VALID");
      }
}
ps vm (String args[])
  {
     new Demo();
  }
```