UNIT-3

Page No.

Multithreading :-

Here a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel.

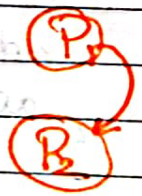
```
A
{
    m1      — thread
    m2      — thread
}
```

Multitasking :-

It is the process of executing several task simultaneously.

↳ **Process - based Multitasking :-**

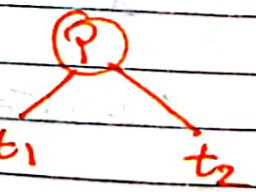
Eg :- Facebook chat, Music



- Heavy weight
- Own memory address space.
- Interprocess communication is expensive

↳ **Thread Based Multitasking :-** eg:- text editor — edit text — Print

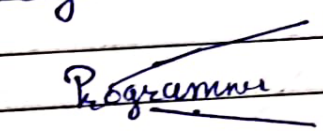
- Light weight
- Address space is shared
- Inter thread communication is cheap.



Page No.

Benefits of Multithreading :-

<i> Enable Programmer to do multiple task at one time



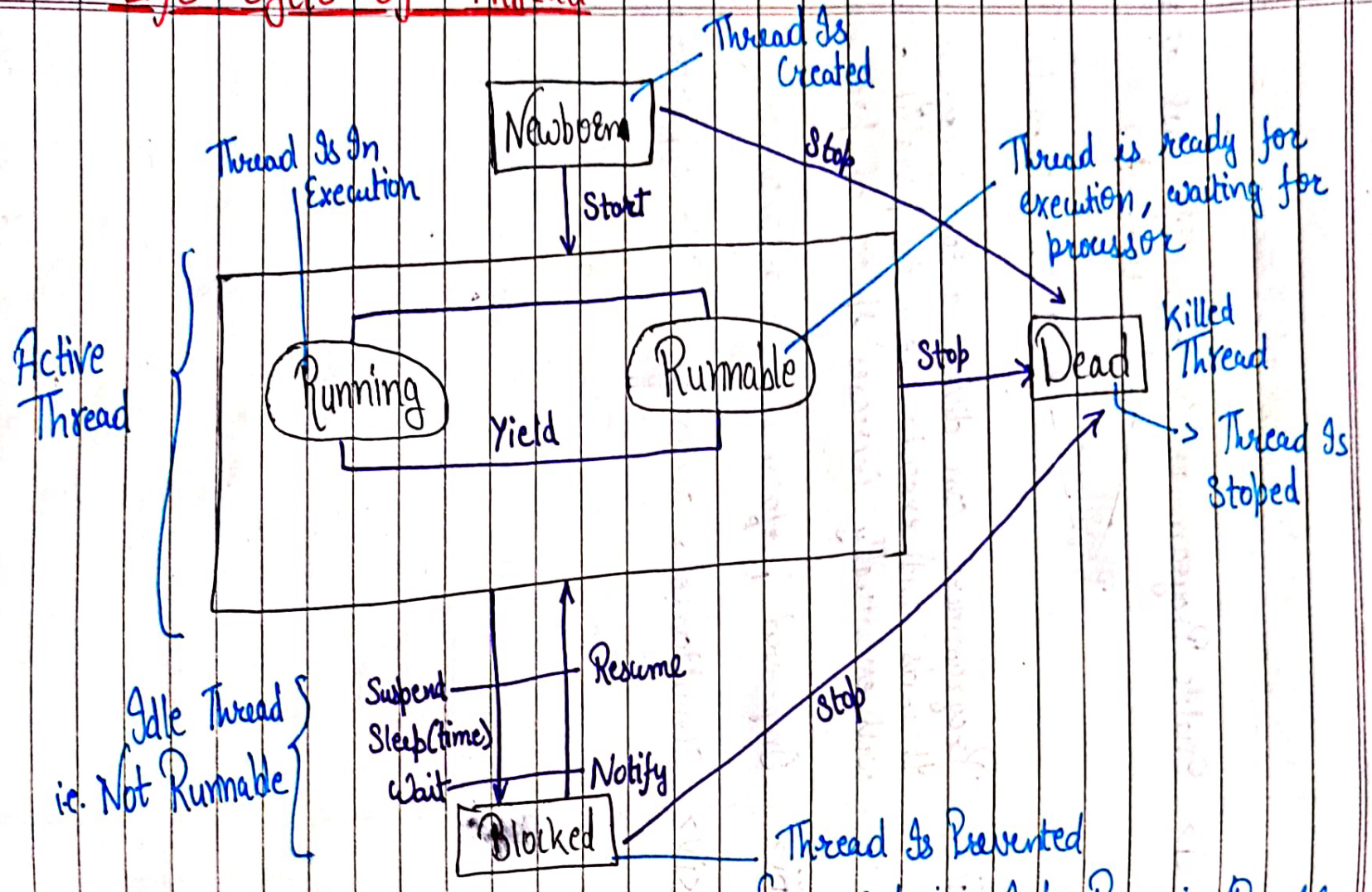
<ii> Programmers can divide a long program into threads and execute them in parallel which will eventually increases speed of program execution.

<iii> Improved performance and concurrency.

<iv> Simultaneous access to multiple applications.

Imp

Life Cycle Of Thread



Page No.

Page No.

Thread Creation

Main Thread :-

Every time a Java program starts up, one thread begins running which is called as the main thread of the program because it is the one that is executed when your program begins

You can implement Runnable Interface

Class t1 implements Runnable

You can extend the Thread class.

Class t1 extend Thread.

Thread Class

```
Class t2 extends Thread
{ public void run()
  { S.O.P("Thread t2");
  }
p.s.v.m (String a[])
  {
    t2 obj1 = new t2();
    obj1.start();
  }
}
```

Implementing Runnable Interface

```
class t1 implements Runnable
{ public void run()
  { S.O.P("Thread t1")
  }
p.s.v.m (String a[])
  {
    t1 obj1 = new t1();
  Thread t = new Thread(obj1);
    t.start();
  }
}
```

Page No.

Use Of Stop() Method :-

Stop() method kills the thread on execution

Ex :-

```
class A extends Thread
{
    public void run()
    {
        for (int i=1; i<=5; i++)
        {
            if (i==2)
                Stop();
            S.O.P("A:" +i);
        }
    }
    public Static void main (Sting a[])
    {
        A a = new A()
        a.Start();
    }
}
```

Thread Gets Terminated ←

Output

A:1

A:
← loop
A:

Page No.

Use of Sleep() Method

It causes the currently running thread to block for atleast the specified number of milliseconds.

You need to handle exception while using sleep() method

Example :-

```
class A extends Thread
{
    public void run()
    {
        for (int i=1; i<=5; i++)
        { try()
          {
            if (i==2) Sleep(1000);
          }
          catch (Exception e)
          {
          }
        }
        S.O.P("A"+i);
    }
}
p.s.v.m(String a[])
{
    A a = new A();
    a.Start();
}
```

Milliseconds

oms →

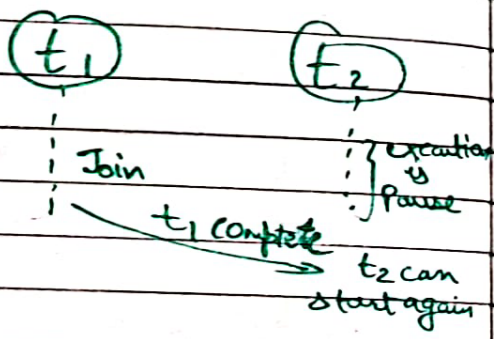
Page No.

Use of isAlive() and Joint() Method :-

isAlive() method tests if the thread is alive.

{ True / False } public final boolean isAlive()

joint() method waits for a thread to die. It causes currently running thread to stop executing until it joins complete its task.



Example :-

```
class A extends Thread
  {
    public void run()
    {
      S.O.P("Status: " + isAlive());
    }
  }
class alivetest
  {
    p.s.v.m(String[] arg)
    {
      A a = new A();
      a.start();
      S.O.P("New Status" + a.isAlive());
    }
  }
```

Output
Status : True
NewStatus : True

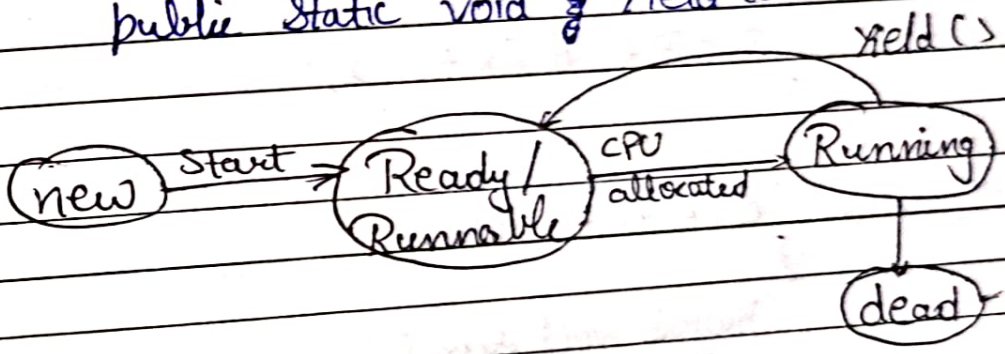
Page No.

Use Of Yield() Method :-

Yield method causes to pause current executing thread for giving the chance to remaining waiting threads of same priority

-> If there are no waiting threads or all waiting threads have low priority then the same thread will continue its execution once again.

public static void yield()



Example :-

```
class A extends Thread
{
  public void run()
  {
    for (int i=1; i<=5; i++)
    {
      if (i==2) yield();
      S.O.P("A"+i);
    }
    S.O.P("A Exit");
  }
}
class B extends Thread
{
  public void run()
  {
    for (int j=1; j<=5; j++)
    {
      S.O.P("B"+j);
    }
    S.O.P("B Exit");
  }
}
```

Page No.

```
class Yieldtest
  {
    p.s.v.m (String a[ ])
    {
      A a = new A();
      B b = new B();
      a.start();
      b.start();
    }
  }
```

Can we Start a thread twice?

No

```
class A extends Thread
  {
    public void run()
    {
      S.O.P("Thread A");
    }
    p.s.v.m (String [] ar)
    {
      A a1 = new A();
      a1.start();
      a1.start();
    }
  }
```

Output

Error :- IllegalThreadState Exception

Page No.

Synchronization :-

↳ Used to solve data inconsistency problem.

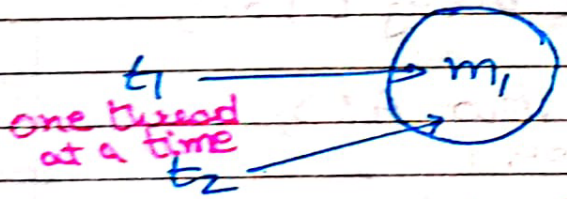
↳ Synchronization is the modifier applicable only for methods and blocks.

↳ Can't apply for classes and ~~objects~~ variables.

↳ Synchronization keyword in Java creates a block of code known as critical section.

↳ To enter a critical section, a thread needs to obtain the corresponding object's lock.

```
Synchronized (object)
{
    // Statements
}
```



Page No.

Program without Synchronization

```
class A
{
Synchronized void addnew(int i)
    {
      Thread t = Thread.CurrentThread;
       for(int n=1; n<=5; n++)
          {
            S.O.P(t.getName()+" - "(i+n);
          }
    }
class B extends Thread
    {
      A a1 = new A();
      public void run()
        {
          a1.addnew(100);
        }
    }
Class Syntest
  { p.s.v.m(String a[])
  { B b = new B();
  Thread t1 = new Thread(b);
  Thread t2 = new Thread(b);
   t1.setName("T1");
   t2.setName("T2");
   t1.start();
   t2.start();
  }
}
```

O/P without Synchronized

```
T-101
T2-101
T2-102
T1-102
 :
 :
```

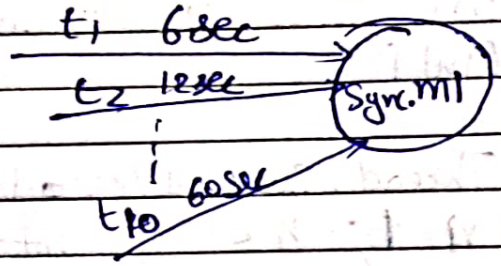
O/P with Synch.

```
T1 -101
T1 -102
T1 -103
T1 -104
T1 -105
T2 -101
T2 -102
T2 -103
T2 -104
T2 -105
```

Page No.

Problem Of Synchronization

↳ Increases waiting time



↳ Effect the performance of the system

S.No	Method with discription
1)	public void start()
	Starts the thread in a seperate path of execution, then invokes the run() method on this Thread object.
2)	public void run()
	If this Thread object instantiated using a seperate Runnable target, the run() method is invoked on that Runnable object.
3)	public final void setName(String name)
	changes the name of the Thread object. There is also a getName() method for retrieving the name.

Page No.

4> public final void setPriority (int priority)

Sets the priority of this Thread object. There is also a getName() method for retrieving the name.

5> public final ~~void~~ void join (long millisec)

The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.

6> public void interrupt()

Interrupts this thread, causing it to continue execution if it was blocked for any reason.

7> public final boolean isAlive()

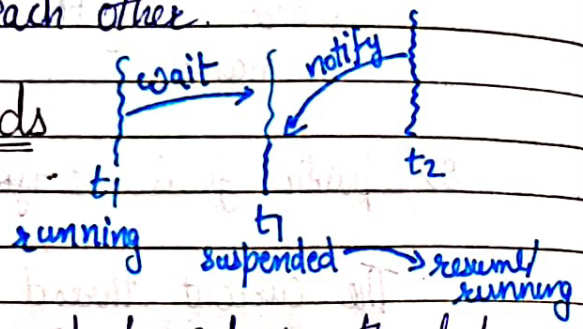
Returns true if the thread is alive.

Page No.

Interthread Communication

Used to allow synchronized thread to communicate with each other.

Methods



1) **wait :-**

Causes current thread to release the lock and wait until
- specified time elapsed
- another thread calls notify or notifyAll method.

public void wait ()

2) **notify :-**

wakes up the single thread i.e. waiting on this object's monitor

public void notify ()

3) **notifyAll :-**

Wakes up all the threads waiting.

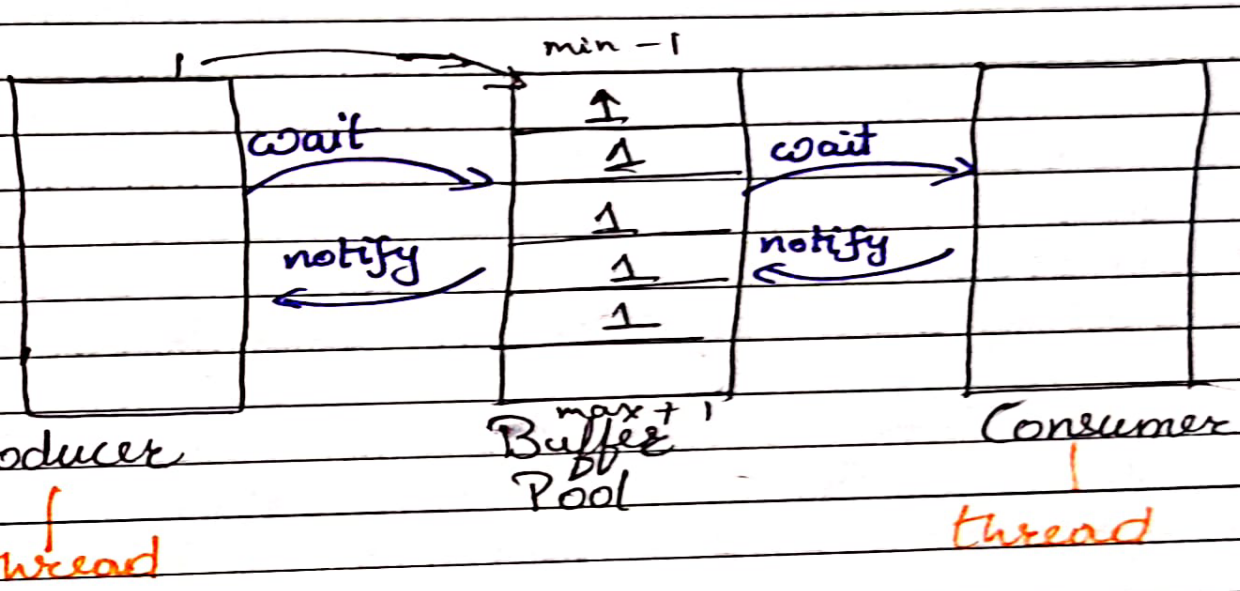
public void notifyAll ()

Page No.

Example of Inter thread communication

Producer - Consumer Problem

↳ multi process synchronization



Producer can't add data into Buffer if it is full.

Consumer can't consume data if it is empty.

Page No.

Page No.

Java AWT Programming

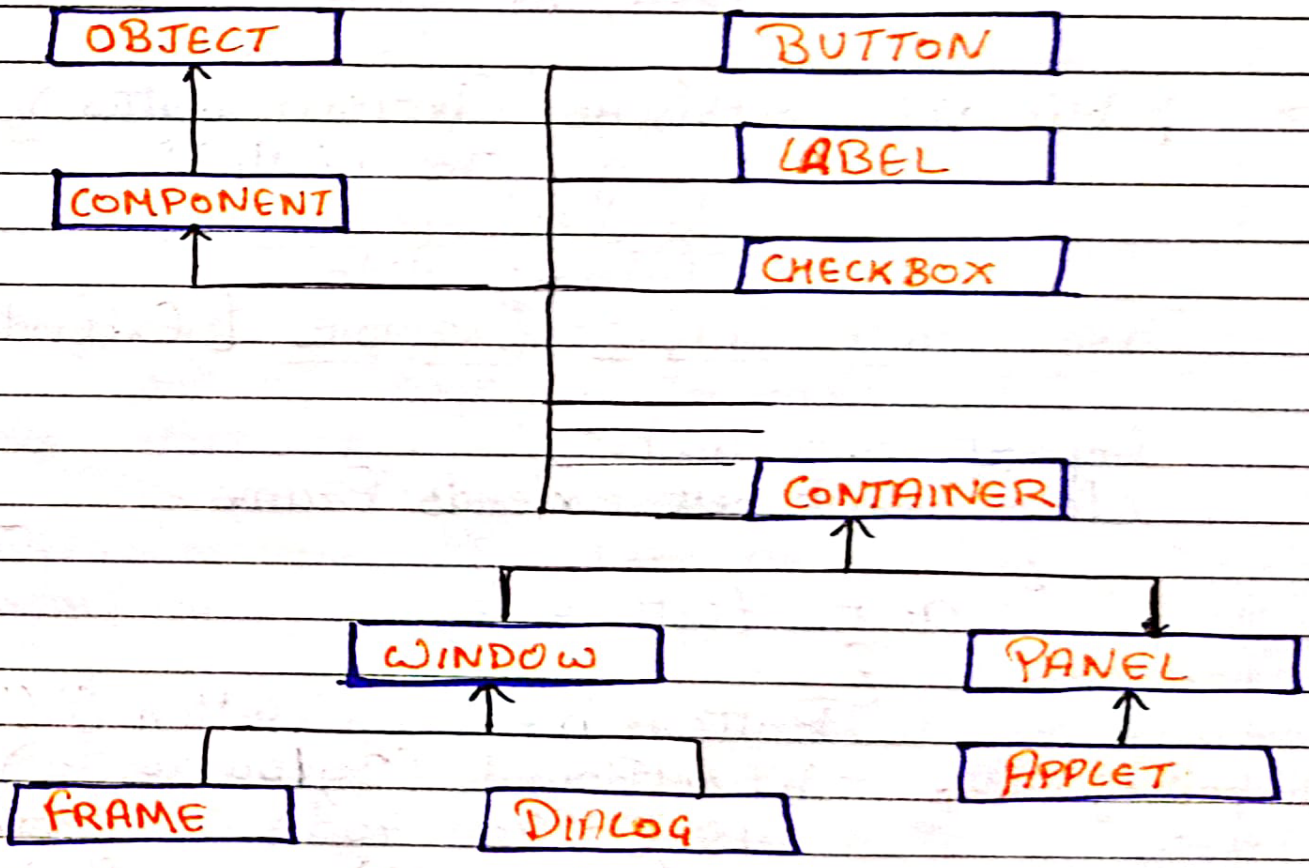
AWT - Abstract Window Toolkit

Used to develop GUI or Window based application.

AWT components are Platform dependent.

java.awt package is used.

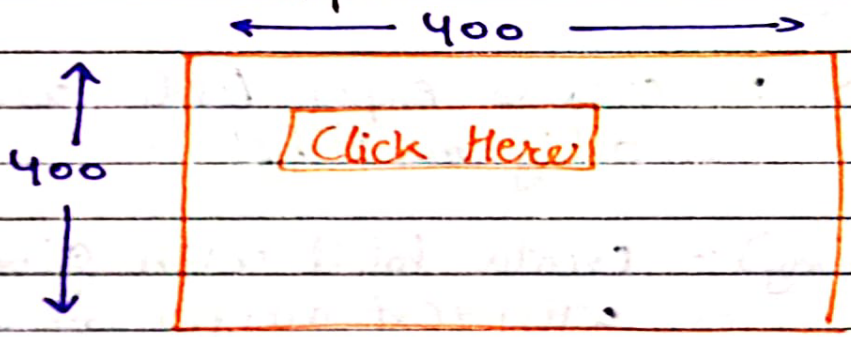
Hierarchy



Page No.

```
public static void main (String [] ar)
    {
      awt_test a1 = new awt_test ();
    }
}
```

Output



Components Of AWT :-

Every GUI based program consists of a screen with set of objects. In java, these objects are called components.

Components frequently used as Buttons, Checkboxes, RadioButton, textfield etc.

At the top of AWT hierarchy is the component class. Component is an abstract class that encapsulates all the attributes of the component.

All User Interface elements that are displayed on the screen and interact with user are the subclasses of the component.

Page No.

IMPORTANT METHODS OF COMPONENT CLASS

<i> public void add (Component c)
↳ Inserts component in Container.

<ii> public void setSize (int width, int height)
↳ set the size.

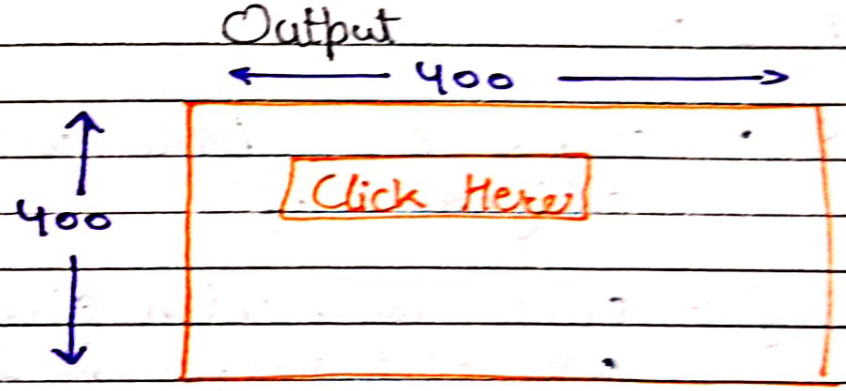
<iii> public void setLayout (LayoutManager m)
↳ defines Layout but if not mention then, by default it is Flow Layout.

<iv> public void setVisible (boolean Status)
↳ Changes the visibility
True :- Show
False :- Hide.

Java AWT Simple Example [Extend Frame]

```
import java.awt.*;
class awt_test extends Frame
  {
    awt_test ()
      {
        Button b = new Button ("Click Here");
        b.setBounds (30,100,80,30);   // set the pos^n of Button
        setSize (400,400)             // Frame Size
        setLayout (null);             // By default Flow Layout
        setVisible (true);            // to add button
```

```
public static void main (String [] ar)
    {
      awt_test a1 = new awt_test ();
    }
}
```



Components Of AWT :-

Every GUI based program consists of a screen with set of objects. In java, these objects are called components.

Components frequently used as Buttons, Checkboxes, RadioButton, textfield etc.

At the top of AWT hierarchy is the component class. Component is an abstract class that encapsulates all the attributes of the component.

All User Interface elements that are displayed on the screen and interact with user are the subclasses of the component.

Page No.

① Labels :-

↳ Object of type label
↳ Contains a string which it displays.
↳ do not support any interaction with the user.

Constructors :-

Label() :- creates empty lable with its text allignment left.

Label(String) :- creates label with given string with text allignment left.

Label(String, int align) :- creates Label with given string and with given allignment.

Available Allignment :-

Label.left
Label.right
Label.center

Page No.

Example

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

public class LabelDemo extends Applet
  {
     public void init ()   // init always use
      {
       Label one = new Lable ("One", Label.RIGHT);
        One.setBackground (Color.Red);
                                  // for Background color

       Label two = new Lable ();
       two.setBackground (Color.blue);
        two.setText ("Two");

       Lable three = new Label ("Three")
        three.setforeground (Color.orange);
                                  // For Character color
       Lable four = new Label ();
       four.setText (three.getText ());

       add (one);
       add (two);
       add (three);
       add (four);
  }
```

Output

Page No.

Page No.

② Buttons :-

Buttons are simple components that trigger some action on your interface.

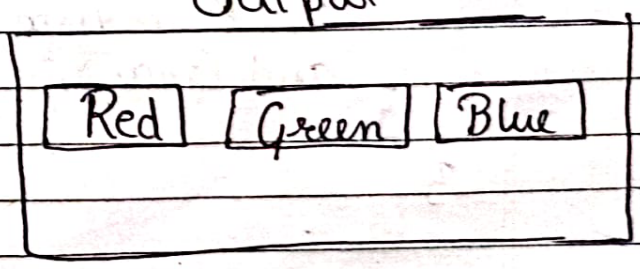
Button ()
Button (String)

Contains a label and generates an event when pressed

Example

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

public class ButtonDemo extends Applet
{
    Button rbtn, gbtn, bbtn; // initialize button name.
    public void init ()
    {
        rbtn = new Button ("Red");
        gbtn = new Button ("Green");
        bbtn = new Button ("Blue");

        add (rbtn);
        add (gbtn);
        add (bbtn);
    }
}
```

Output



Page No.

③ Checkbox :-

Control used to turn option on or off

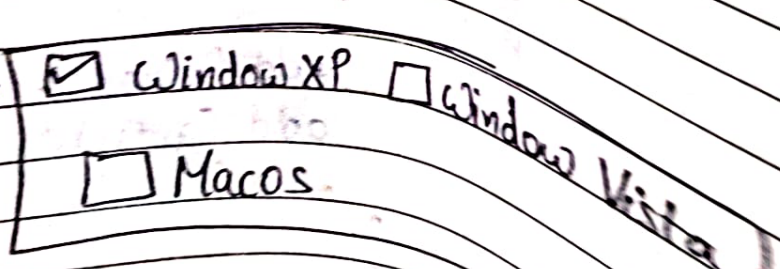
Checkbox()
Checkbox(String str)
Checkbox(String str, boolean on)

In this we can choose 1 or more options.

Example :-

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

public class checkboxDemo extends Applet
{
Checkbox winXP, winVista, mac; //initializing checkbox name it show
public void intt()
{
winXP = new Checkbox("Window XP", true);
winVista = new Checkbox("Window Vista");
mac = new Checkbox("Mac OS");

add(winXP);
add(winVista);
add(mac);
}
}
```

Output



Page No.

④ Checkbox Group :-

Helps to create a set of mutually exclusive check boxes
These check boxes are often called radio buttons
In this, we can only select 1 option.

CheckboxGroup ( )

Constructor of checkbox class used for creating radio buttons.

Checkbox (String str, boolean on, CheckboxGroup cbg)
or
Checkbox (String str, CheckboxGroup cbg, boolean on)

Example

```
public class CBGroup extends Applet
{
    Checkbox winXP, winVista;
    CheckboxGroup cbg;
    public void init ()
    {
        cbg = new CheckboxGroup ();
        winXP = new Checkbox ("Window XP", cbg, true);
        winVista = new Checkbox ("Windows Vista", cbg, false);

        add (winXP);
        add (winVista);
    }
}
```

Output

(•) Windows XP ( ) Windows Vista

Page No.

Layout Manager

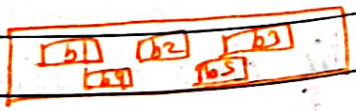
↳ Layout manager is the process that determines the size and position of components in a container.

↳ Each time we create a container java automatically creates a default layout manager i.e. Flow Layout.

↳ You can create different types of layout manager in order to control how your applet looks.

↳ Set by setLayout() method.

1) Flow Layout



↳ Flow layout is the default layout for JPanel

↳ It sets the components (button) as its preferred size in Java.

↳ It also sets the width of the button according to its content. (component)

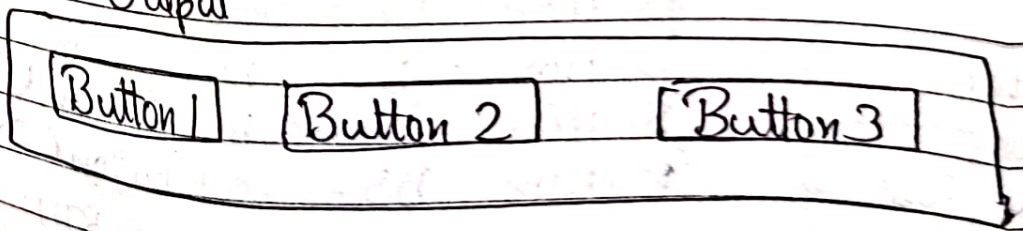
↳ If width of Jframe is less than it shifts the extra components to the next row.

↳ If width of Jframe is more than it aligns the components to the center.

Example :-

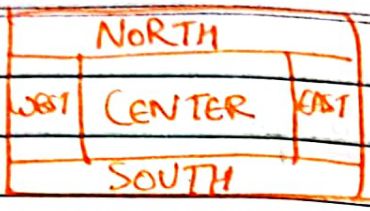
```
import java.awt.*;
import java.applet.*;

public class Flowlayout extends Frame
  { Flowlayout()
   {Button b1 = new Button ("Button 1");
   Button b2= new Button ("Button 2");
   Button b3= new Button ("Button 3");

   add(b1);
   add(b2);
   add(b3);
  }
public static void main(String arr[])
   {
   Flowlayout f = new Flowlayout();
} }
```

Output



Page No.

2) Boder Layout :-



It is used to arrange the components in five region :- north, south, east, west and centre Each region contain only 1 component.

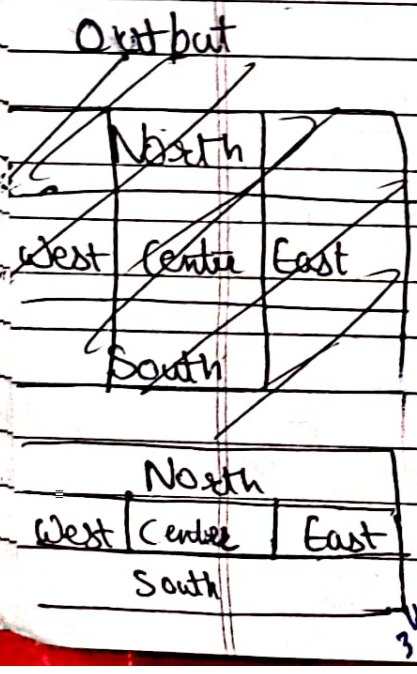
Border layout () :- Creates a border layout but with no gaps b/w the components.

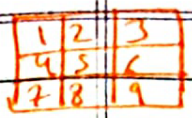
Borderlayout (int hgap, int vgap) :

Creates a border layout with the given horizontal and vertical gaps b/w the components.

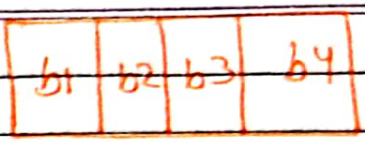
Ex :-
```
import java.awt.*;
public class Border extends Frame
{
    Border ()
    {
Button b1 = new Button ("North");
Button b2 = new Button ("South");
Button b3 = new Button ("West");
Button b4 = new Button ("East");
Button b5 = new Button ("Center");

add (b1, Borderlayout.NORTH);
add (b2, Borderlayout.SOUTH);
add (b3, Borderlayout.WEST);
add (b4, Borderlayout.EAST);
add (b5, Borderlayout.CENTER);
}
public static void main ()
{
Border b = new Border ();
```

Output





Grid Layout



↳ Just like flow layout, grid layout also sets the component left to right in a flow.

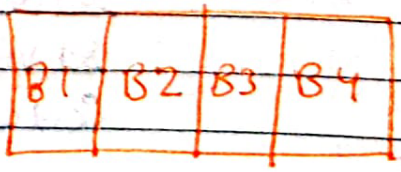
↳ In grid layout, all the available space is consumed by the components.

↳ Grid layout can be devided into the form of rows and columns.

↳ In grid layout, all the components have same size.

Ex:-

```
public class GridlayoutSample extend JFrame
{
    JButton b1= new JButton ("B1")
    JButton b2 = new JButton ("B2")
    JButton b3= new JButton ("B3")
    JButton b4 = new JButton ("B4")

    public GridlayoutSample ()
    {
        Gridlayout g = new Gridlayout ();
        C.get C.Set layout (g);
        add (b1);
        add (b2);
        add (b3);
        add (b4);
    }
public static void main ();
    { GridlayoutSample g = new Gridlayout (2,2)
    } Setlayout (g);
```



Page No.

Grid Bag Layout

↳ It is the most powerful and flexible of all the layout managers but more complicated to use.

↳ Unlike grid layout, where the components are arranged in rectangular grid and each component in the container is forced to be same size, in GridBagLayout, components are also arranged in rectangular grid but can have different sizes and can occupy multiple rows and columns.

```
import java.awt.*;
import java.awt.event.*;
public class Mylayout extends JFrame
{
    JButton btn1 = new JButton("B1");
    "      btn2 = "    "    ("B2");
    "      btn3 = "    "    ("B3");
    "      btn4 = "    "    ("B4");
    "      btn5 = "    "    ("B5");

    public Mylayout()
    {
        setTitle("Grid Baglayout Example");
        setBounds(100,200,500,300)
        setVisible(true);
        Container c = getContentPane();
```

Page No.

```
GridBagLayout g = new GridBagLayout();
c.setLayout(g);

GridBagConstraints gbc = new GridBagConstraints();

gbc.gridx = 0;
gbc.gridy = 0;
c.add(btn1, gbc);

~~gbc.grid~~
gbc.gridx = 1;
gbc.gridy = 0;
c.add(btn2, gbc);

gbc.gridx = 2;
gbc.gridy = 0;
c.add(btn3, gbc);

gbc.gridwidth = 3;
gbc.gridx = 0;
gbc.gridy = 1;
gbc.ipady = 40;
c.add(btn4, gbc);

gbc.gridwidth = 2;
gbc.ipady = 0;
gbc.gridx = 1;
gbc.gridy = 2;
gbc.anchor = GridBagConstraints.PAGE_END;
gbc.weight = 1;
c.add(btn5, gbc);
```

Grid layout Example

B1	B2	B3
B4		
	B5	

Card Layout

↳ It treats each component in a container as a card.

↳ Only one card is visible at a time and the container acts as a stack of cards.

↳ The first component added to a CardLayout object is the visible component when the container is first displayed.

↳ CardLayout has some useful methods to flip the cards i.e.

first()
last()
next()
previous()
show()

Ex:-
```
class CardLayoutExample
{
    public static void main(String arg[])
    {
        JFrame frame = new JFrame();
        frame.setVisible(true);
        frame.setBounds(100, 100, 500, 400);
        frame.setTitle("Card Layout Example");
        Container c = frame.getContentPane();
```

Page No.

```
CardLayout Card = new CardLayout();
~~c.setLayout(CardLayout~~
c.setLayout(Card);

JButton B1 = new JButton("Page 1");
"  B2 = "  "  ("Page 2");
"  B3 = "  "  ("Page 3");
"  B4 = "  "  ("Page 4");

~~Center~~
c.add(B1, "1");   <-- ID
c.add(B2, "2");
c.add(B3, "3");
c.add(B4, "4");
}
}
```