22.1.20

UNIT-IV
MULTI THREADING

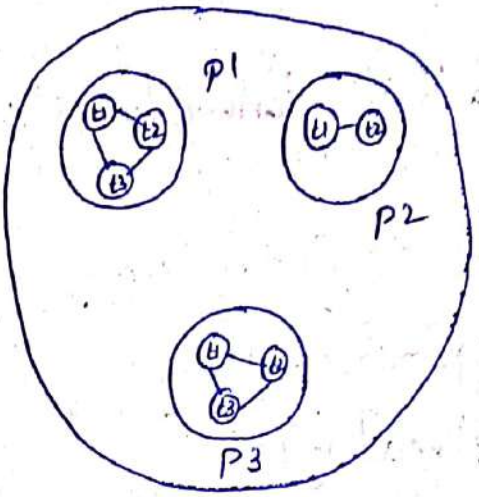
Multi Tasking:

* Performing multiple tasks simultaneously is known as multi tasking.
* Multi-tasking can be achieved in two ways
  1. process based (Multi processing)
  2. thread based (Multi threading)
* Java supports Multi threading
* A thread is a light weight sub process
* A process is a program in execution (or) excutional Instance (or) Time instance of a program

Differences between Multiprocessing and Multithreading

Multiprocessing	Multithreading
1. process of performing Multiple processes at a time	1. process of performing multiple threads at a time
2. process is a heavy weight	2. Thread is a light weight
3. Context switching is complex	3. Context switching is simple
4. Multiprocessing requires more resources	4. Multi threading requires less resources.
5. Each and every process requires seperate memory blocks.	5. All the threads of a process shares a common memory area.

The relationship between processors and threads.



OS

* A thread can be created in a java program in two ways

1. Extending a thread class
2. Implementing a Runnable interface

Thread is a built in class or predefined class

Example for extending a thread class:

```
class Sam extends Thread
{
  - - -
  - - -
}
```

Example for implementing a Runnable interface

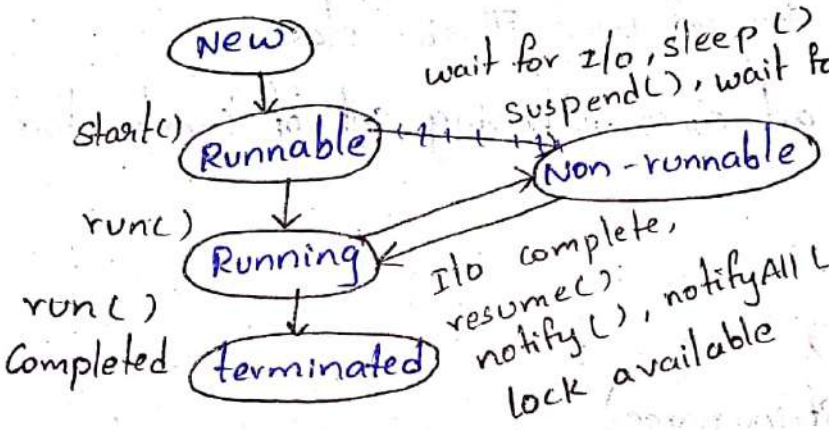
```
class Sam implements Runnable
{  - - - -
   - - -
}
```

Thread life cycle:

* The states of a thread is known as life cycle of a thread.

* There are 5 states

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



New:

A thread enters into a new state whenever an instance of thread class is created

Runnable:

A thread enters into a runnable state if it calls a start() method

Running:

A thread enters into a running state if it and only if it calls the run() method and the thread scheduling of JVM selects the thread to run

Non-runnable (or) Blocked:

If the thread instance requires as I/o or suspending for some time (or) sleeping (or) waiting for a lock on

particular resource then it enters into a non-runnable (or) blocked state.

Whenever I/O is completed (or) resume (or) notify (or) notifyAll is called (or) not to lock for a particular resource is available then it enters into running state

Terminated:

If the run() method execution is completed then thread is terminated (or) dead

Creating a thread using Thread class

The Thread class is having set of constructors and methods.

Constructors

1. Thread()
2. Thread(String name)
3. Thread(Runnable r)
4. Thread(Runnable r, string name)

Example:

```
class SamThread extends Thread
{
    public void run()
    {
        S.O.P("Hello");
    }
}
class ThreadDemo
{   public static void main(String args[])
    {
```

```
        SamThread s1 = new SamThread();
        SamThread s2 = new SamThread();
        s1.start();
        s2.start();
    }
}
```

o/p: Hello
Hello

creating a thread by implementing a runinable

```
class SamThread implements Runnable
{   public void run()
    {   S.O.P("Helo");
    }
}
class RunnableDemo
{
    P.S.V.m(String args[])
    {
        SamThread s1 = new SamThread();
        SamThread s2 = new SamThread();
        Thread t1 = new Thread(s1);
        Thread t2 = new Thread(s2);
        t1.start();
        t2.start();
    }
}
```

o/p: Hello
Hello

Illustration of sleep method:

The sleep() method is a static method and it is for making the thread to sleep.

The sleep() method takes milliseconds as input

```
class SamThread implements Runnable
{   public void run()
    {  for(int i=1 ; i<=5 ;i++)
       {  try
          { Thread.sleep(500);
          }
          catch(Exception e)
          {
             S.O.P(e);
          }
          S.O.P(" "+i);
       }
    }
}
class sleepDemo
  {   P.S.V.M(String args[])
      {
          SamThread s1 = new SamThread();
          SamThread s2 = new SamThread();
          Thread t1 = new Thread(s1);
          Thread t2 = new Thread(s2);
          t1.start();
          t2.start();
      }
  }
```

o/p:

```
1
1
2
2
3
3
4
4
5
5
```

Illustration of join() method:

The join() method waits for a thread to die

```
class SamThread extends Thread
{   public void run()
    {  for(int i=1; i<=5; i++)
       {  try:
          {  Thread.sleep();
          }
          catch (Exception e)
          {  S.O.P(e);
          }
          S.O.P(i);
       }
    }
}
class JoinDemo
 {   P.S.V.m (String args[])
     {   SamThread s1 = new SamThread();
         SamThread s2 = new SamThread();
         SamThread s3 = new SamThread();
         s1.start();
         try
         {   s1.join();
         }
         catch (Exception e)
         {   System.out.println(e);
         }
         s2.join() s2.start();
         s3.join()  s3.start();
     }
}
```

O/P:

```
1
2
3
2
4
5
1
1
2
2
3
3
4
4
5
5
```

All the above examples are single task by multiple threads

Multiple Tasks by multiple threads:

```
class Test1 extends Thread
{
    public void run()
    {  for(int i=1; i<=5; i++)
       {  try
          {  Thread.sleep(500);
          }
          catch(Exception e)
          {  sop(e);
          }
          sop("Hello");
    }  }
}
class Test2 extends Thread
{  public void run()
   {  for(int i=1; i<=5; i++)
      {  try
         {  Thread.sleep(500);
         }
         catch(Exception e)
         {  sop(e);
         }
         sop("SVEC");
   }  }
}
```

```
class MultipleTaskThread
{   P.S.V.m (String args[])
    {   Test1 t1=new Test1();
        Test2 t2=new Test2();
        t1.start();
        t2.start();
    }
}
```

Illustration of ~~current thread~~ currentThread(), getName(), setName(), getId(), setPriority(), getPriority():

```
class Thread1 extends Thread
{   public void run()
    {   S.O.P(Thread.currentThread());
        S.O.P(Thread.currentThread().getName());
        S.O.P(Thread.currentThread().getId());
        S.O.P(Thread.currentThread().getpriority());
    }
}
class Sam
{   public static void main(String args[])
    {   Thread1 t1 = new Thread1();
        t1.start();
        Thread1 t2 = new Thread2();
        t1.setName("aaa");
        t1.start(); t2.setName("bbb");
        t2.start();
    }
}
```

o/p:

```
Thread[aaa,5,main]
aaa
5
Thread[bbb,5,main]
bbb
5
```

```
class Thread1 extends Thread
{   public void run()
    {   S.O.P(Thread.currentThread().getName()+"
            +Thread.currentThread().getId()+"
            +Thread.currentThread().getPriority());
}   }
class Sam
{   public static void main(String args[])
    {   Thread1 t1=new Thread1();
        Thread1 t2=new Thread1();
        Thread1 t3=new Thread1();
        t1.setName("aaa");
        t2.setName("bbb");
        t3.setName("ccc");
        t1.setPriority(Thread.MIN_PRIORITY);
        t2.setPriority(Thread.MAX_PRIORITY);
        t1.start();
        t2.start();
    }   t3.start();
}
```

O/P:

~~bbb~~ aaa 8 1
ccc 10 5
bbb 9 10

getName()

public String getName() returns the name of the current thread. If no name is set by the user then it returns the thread name as "Thread-0, Thread-1 ...

setName()

public void setName(String s) sets the name for a Thread.

getId()

public int getId() returns the id of a thread

getPriority()

public int getPriority() returns the priority ~~of~~ value of a thread. If we donot set the priority of thread then it returns 5.

setPriority()

public void setPriority(int i) sets the priority of a thread. There are three priority constants in a Thread class which are represented by names:

1. Thread.MAX_PRIORITY (10)
2. Thread.MIN_PRIORITY (1)
3. Thread.NORM_PRIORITY (5)

CurrentThread():

public static Thread currentThread() returns currently running thread object

Thread Synchronization:

Synchronization is the process to control multiple threads (Processes) to access shared resource.

There are two types of thread synchronization

1. Mutual exclusion
   - (a) Synchronized methods
   - (b) synchronized blocks
2. Inter process Communication

Mutual exclusion:

Synchronized methods:

By adding the keyword synchronized to the prototype of a method then it is known as synchronized method

Illustration of multiple threads accessing common resource

without synchronized methods

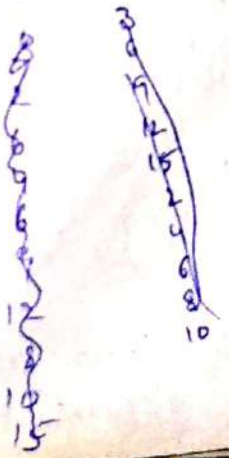
```
class Table
{   void PrintTab(int n)
    {   for(i=1 ; i<=5; i++)
        {    s.o.p(n*i);
        }
        try
        {   Thread.sleep(300);
        }
        catch (Exception e)
        {
        }
    } } }
}
```

```
class Thread1 extends Thread
{   public void run
    Table t;
    Thread1 (Table t)
    {   this.t = t;
    }
    public void run()
    {   t.printTab(3);
    }
}
class Thread2 extends Thread
{   Table t;
    Thread2 (Table t)
    {   this.t = t
    }
    public void run()
    {   t.printTab(2)
    }
}
class Sam
{   Table t = new Table();
    Thread1 t1 = new Thread1(t);
    Thread2 t2 = new Thread2(t);
    t1.setName
    t1.start();
    t2.start();
}
```

o/p:

3
2
4
6
6
9
12
8
10
15

o/p:



with synchronized:

```
class Table
{ synchronized
    void printTab(int n)
    {   for(int i=1; i<=5; i++)
        {
            S.O.P(n*i);
        }
        try
        {
            Thread.sleep(300);
        }
        catch (Exception e) { }
    }
}
class Thread1 extends Thread
{   Table t;
    Thread1(Table t)
    {   this.t = t;
    }
    public void run()
    {   t.printTab(3);
    }
}
class Thread2 extends Thread
{   Table t;
    Thread2 (Table t)
    {   this.t = t;
    }
    public void run()
    {   t.printTab(2);
    }
}
```

Synchronized block

```
class Table
{ void printTab(int n)
  { Synchronized(this)
    { for(int i=1;i<=5;i++)
      { S.O.P(n*i);
      }
      try
      { Thread.sleep(300);
      }
      catch(Exception e)
      {
      }
    }
  S.O.P(Thread.currentThread
  ().getName()
```

o/p:

```
3
6
9
12
15
2
aaa
4
6
8
10
bbb
```

```
class Sam
{   Thread pvsm(String args[])
   { Table t=new Table();
     Thread1 t1=new Thread1(t);
     Thread2 t2=new Thread2(t);
                    t1.setName("aaa");
       t1.start(); t2.setName("bbb");
       t2.start();
   }
}
```

o/p:

```
2
4
6
8
10
3
6
9
12
15
```

Synchronization using Anonymous classes

```
class Table
{  void printTab(int n)
     {  for(i=1; i<=5;i++)
        {  s.o.p(n*i);
        }
        try
         {  Thread.sleep(300);
         }
         catch(Exception e){}
     }
}
class Sam
{ static final Table t=new Table();
   p.s.v.m(String args[])
   {   new Thread()
       {
```

```
        public void run()
          {   t.printTab(3);
          }
    }.start();
    new Thread()
    {   public void run()
        {   t.printTab(2);
        }
    }.start();
}   }
```

o/p:

```
2                3
4                6
6                9
8                12
10      (or)     15
3                2
6                4
9                6
12               8
15               10
```

Dead lock:

Lock (or) Monitor:

Lock (or) monitor is a concept in JVM (or) entity in JVM. Every object has a lock associated with it If a thread needs a consistent access on a particular objects fields (or) methods before accessing them and then release the lock when it is done.

Deadlock:

Dead lock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object

Deadlock:

lock that is acquired by first thread

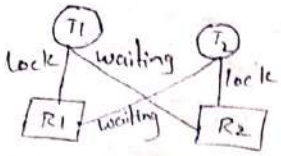


Illustration of deadlock scenario

```
class DeadLockDemo
{ public static void main(String args[])
  {
     String
     final String R1 = "aaa"; s2 = "bbb";
     final String R2 = "bbbb";
     new Thread()
     { public void run()
       { synchronized (R1)
         { S.O.P("Thread1 lock resource1");
         }
         try
         { Thread.sleep(300);
         }
         catch (Exception e) { }
         Synchronized (R2)
         { S.O.P("Thread1 lock resource2");
         }
       }
     }.start();
     new Thread()
     { public void run()
```

```
{   Synchronized (R2)
    {   S.O.P("Thread2 locks resource2");
        try
        {  Thread.sleep(300);
        }
        catch (Exception e) {}
        Synchronized (R1)
        {   S.O.P("Thread2 locks resource1");
        }
    }
}.start();
}}
```

o/p: Thread1 locks resource1
Thread2 locks resource2
(we will not get prompt to terminate it press control c)

* Deadlock Avoidance:

A deadlock can be avoided in the above program by changing the code as

Thread1 and Thread2 should acquire the lock on R1 and then on R2

Program

```
class DeadlockDemo
{   P.S.V.m (String args[])
    {   final String r1 = "aaa";
        final String r2 = "bbb";
        new Thread().
        {   synchronized(r1)
```

```
public void run()
 {    synchronized(r1)
      {  S.O.P("Thread1 locks r1");
         try
          {  Thread.sleep(300);
          }
          catch(Exception e){}
          synchronized(r2)
          {    S.O.P("Thread1  locks r2");
          }
      }
    }
}.start();
new Thread()
 {  public void run()
    {  synchronized(r1)
       {  S.O.P("Thread2 locks r1");
          try
           {  Thread.sloop(300);
           }
           catch(Exception e){}
          synchronized(r2)
          {  S.O.P("Thread2 locks r2");
          }
       }
    }
}.start();
} }
```

o/p:

```
Thread1 locks r1
Thread1 locks r2
Thread2 locks r1
Thread2 locks r2
```

Interprocess communication (IPC) (or) Process-Communication
Interprocess coperation

Inter thread communication:

Inter thread communication is all about allowing synchronized thread to communicate with each other.

cooperation is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter into the same critical section to be executed.

It is implemented by the following methods of Object class.

Method 1 - wait()
Method 2 - notify()
Method 3 - notifyAll()

wait():

pauses current thread to release the lock and wait until another thread invokes notify() or notifyAll() method for this object

Prototype1:

public final void wait() throws InterruptedException

* wait until the object is notified

Prototype 2:

public final void wait(long milliseconds) throws InterruptedExceptio

* waits for the specified amount of time.

notify():

wakes up a single thread that is waiting on this objects monitor (or) lock.

If any threads are waiting on this object one of them is choosen to be awakend.

The choice is arbitrary (or) random

prototype

public final void notify();

notifyAll():

wakes up all threads that are waiting on this objects monitor (or) lock

Prototype:

public final void notifyAll()

Example of inter thread communication

```
class Customer
{   int bal = 10,000;
    synchronized void withdraw(int amount)
    {   if(amount > bal)
        {   s.o.p("less ballance and waiting
            for deposit");
            try
            {   wait();
            }
            catch(InterruptedException e)
            {
            }
        }
    else
    {   bal = bal - amount;
```

```
            s.o.p ("withdraw completed");
        }
    }
    synchronized void deposit(int amount)
    {
        bal = bal + amount;
        s.o.p ("Deposit completed");
        notify ();
    }
}
class ThreadCommunication
{
    Customer
    public static void main (String args[])
    {
        final Customer c = new Customer();
        new Thread()
        {
            public void run()
            {
                c.withdraw(15000);
            }
        }.start();
        new Thread()
        {
            public void run()
            {
                c.deposit(10000);
            }
        }.start();
    }
}
```

O/P: less balance and waiting for deposit
Deposit completed

```
            s.o.p ("withdraw completed");
        }
    }
    synchronized void deposit(int amount)
    {
        bal = bal + amount;
        s.o.p ("Deposit completed");
        notify();
    }
}
class ThreadCommunication
{
    Customer
    public static void main(String args[])
    {   final Customer c = new Customer();
        new Thread()
        {   public void run()
            {   c.withdraw(15000);
            }
        }.start();
        new Thread()
        {   public void run()
            {   c.deposit(20000);
            }
        }.start();
    }
}
```

O/P: less balance and waiting for deposit
Deposit Completed

Daemon T
* Daemo
its a servic
* The th
* The th
1. setDa
Prototype
Public
It sets
upon the st
Prototype o
public
Returns
user Threa
Illustratio
class
{

}
O/P.

Daemon Threads :

* Daemon Thread provides services to user thread that means its a service provider
* The life of Daemon thread depends upon user threads
* The thread class is having two methods
  1. setDaemon() 2. isDaemon()

Prototype:

Public void setDaemon(boolean status)

It sets a currentThread as a Daemon Thread or user Thread depending upon the status that is either True or false.

Prototype of isDaemon:

public boolean isDaemon()

Returns True, if it is a Daemon Thread. Returns false if it is a user Thread.

Illustration of Daemon Thread:

```
class Sam1 extends Thread
{   public void run()
    {   if(Thread.currentThread().isDaemon())
        S.O.P(Thread.currentThread().getName()+" Daemon Thread");
        else
        S.O.P(Thread.currentThread().getName()+" user Thread");
    }
    public static void main(String args[])
    {   Sam1 t1 = new Sam1(), t2 = new Sam1(), t3 = new Sam1()
        t1.setDaemon(true);
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Op: Thread 0 Daemon Thread
Thread 2 user Thread
Thread 1 user Thread

Files sequential access files (Byte oriented classes):
Illustration of writing into a file and reading from a file using fileOutPutStream and FileInputStream.

```
import java.io.*;
class Sam
{
    psvm(String args[])
    { String s="Srivasavi";
      byte[] b=s.getBytes();
      try
      {
      FileOutputStream fout=new FileOutputStream
                                ("abc.txt");
       fout.write(b);
       s.o.p("success----");
       fout.close();
      }
      catch(Exception e){}
      try
      { FileInputstream fin=new FileInputStream
                                ("abc.txt");
        s.o.p((char)fin.read());
        fin.close();
      }
      catch(Exception e){}
    }
}
```

o/p:

```
Success....
s
```

31/1/20

Reading all the data from a file

we can check the end of file by using a special literal '-1'.

The end of file can be represent as '-1'

```
import java.io.*;
class Sam
{   public static void main(String args[]) throws Exception
    {  FileInputStream fin = new FileInputStream("abc.txt");
       int i;
       while((i=fin.read())!=-1)
       {
           S.O.P((char)i);                    (P = print)
       }
       fin.close();
    }
}
```

o/p:

Sri Vasavi

sequential access files using character oriented classes:

Illustration of FileWriter and FileReader

```
import java:io.*;
class Sam
 {   psvm(String args[]) throws Exception.
     {  FileWriter fout = new FileWriter("MNO:txt");
        fout.write('A');
        fout.close();
        FileReader fin = new FileReader("MNO.txt");
```

```
        char c = (char) fin.read();
        s.o.p(c)
        fin.close();
    }
}
```

o/p: A

Illustration of random access file

```
import java.io.*;
class Sam
{   psvm(String args[]) throws Exception
    {
        RandomAccessFile fout = new RandomAccessFile
                                ("abc.txt","rw");

        fout.seek(3);
        String s = "****";
        byte b[] = s.getBytes();
        fout.write(b);
        fout.close();
        RandomAccessFile fin = new RandomAccessFile
                               ("abc.txt", "r");

        fin.seek(6);
        byte bi[] = new byte[15];
        fin.read(bi);
        S.O.P(new String(bi));
    }
}
```

o/p: Vi

There are two types of files

1. sequential access file
2. Random access file

Sequential Access Files:

The read and write operations on sequential access files can be performed serially or sequentially That is write from ~~starting position~~ beginning of the file.

There are two ways to perform read and op-write operations on sequential access files

1. byte oriented
2. character oriented.

There are two classes under byte oriented

1. FileOutputStream
2. FileInputStream

There are two classes under character oriented

1. FileWriter
2. FileReader

FileOutputStream

Opening a file for writing.

```
FileOutputStream fout = new FileOutputStream("abc.txt");
```

path of a file (or) name of a file

methods:

1. void write(int b)
   writes the corresponding equivalent ASCII character into a FileOutputStream
2. void write(byte[] b) (or) void write(byte b[])
   It is used to write bytes from the byte array into the FileOutputStream
3. void close()
   Used to close the FileOutputStream.
4. FileDescriptor getFD()
   returns the FileDescriptor associated with the stream.

FileInputStream:

Reads or obtains input bytes from a file.
It is used for reading byte oriented data.
Opening a file for reading

```
FileInputStream fin = new FileInputStream("abc.txt");
```

methods:

1. int read()
   returns a single byte from the file.
2. int read(byte b[])
   reads an array of bytes from the file
3. void close()
   closes the FileInputStream

character oriented classes:

1. FileReader
2. FileWriter

These classes read or write single character at a time.

openning a file to write a character

```
FileWriter fout = new FileWriter("abc.txt");
```

openning a file for reading a character.

```
FileReader fin = new FileReader("abc.txt");
```

Methods of FileWriter:

1. void write(char c)
   write a single character into a file
2. ~~void write(new String~~
2. void write(new String(byte []))
   writes a string

3. void close()
   closes the FileWriter stream (output stream.

Methods of FileReader:

1. int read().
   reads a byte from the file and returns ASCII integer

2. int read()

3. void close()
closes the FileReader / input stream.

RandomAccessFile
opening a file for reading
RandomAAccessFile fin = new RandomAccessFile("abc.txt","r")

opening a file for writing
RandomAccessFile fout = new RandomAccessFile("abc.txt","rw")
Methods of RandomAccessFile
1. void seek(int position)
Moves the file pointer to a specified position

2. void write(int b)
Writes a single byte

3. void write(byte[] b)
writes a byte array into a file

4. int read()
Reads a single byte

5. int read(byte[] b)
Reads a set of bytes into a byte array b

UNIT-5

Applet Programming

Applet is a special kind of program runs on a web browser and displays a dynamic content on a web.

An Applet can run on anytype of OS like windows, Linux, mac and any web browser like google chrome, firefox, opera etc.

In order to create an applet the class should extend a built in class which is Applet

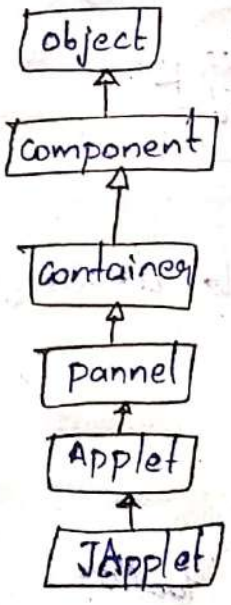
The Applet class is available in the package

```
java.applet.Applet;
```

```
class SamApplet extends Applet.
{

}
```

The class hirarchy of Applet class



The super class / parent class of Applet is pannel

The super class of pannel is Container

The super class of Container is Component

The super class of Component is object

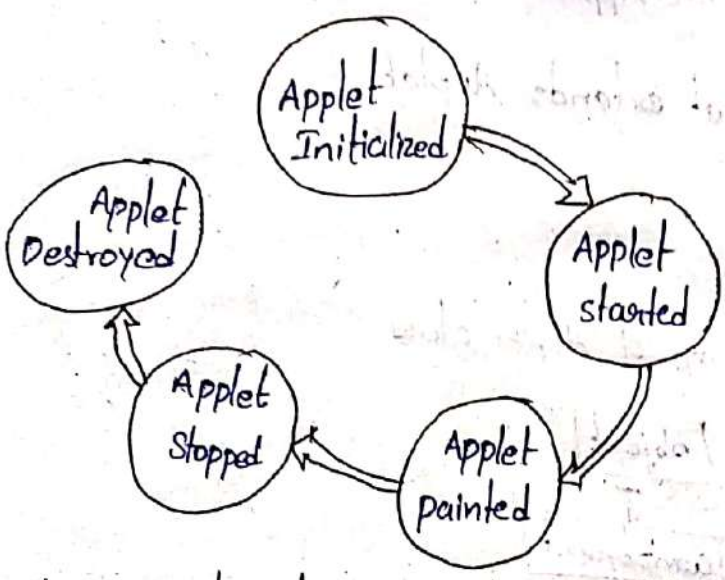
The child class of Applet is JApplet.

* The class Applet can use all the properties and methods of all the parent classes.

Applet Life cycle:

There are 5 stages in Applet life cycle

1. Applet is Initialized
2. Applet is started
3. Applet is Painted
4. Applet is Stopped
5. Applet is Destroyed.



Applet Life cycle classes:

There are 2 Applet life cycle classes

1. java.applet.Applet
2. java.awt.Component

awt – abstract window toolkit

• The above two classes consists of set of methods regarding Applet life cycle.

ds of

java.applet.Applet:

1. public void init()
2. public void start()
3. public void stop()
4. public void destroy()

- The init() method initializes the Applet which means initializes the parameters of an applet. The init() method is called only once during the life cycle
- The start() method starts the applet and it is called whenever the browser is maximised or the window is maximised.
- The stop() method stops the applet and it is called whenever the browser is minised.
- The destroy() method is called only once during the life cycle whenever we close the browser window.

java.awt.Component

public void paint(Graphics g)

This method should be defined in order to generate a dynamic content on the web.

This method takes a parameter which is an object of Graphics class.

The Graphics class provides several methods like to draw strings, to draw the shapes and to display images

Running an applet:

There are two ways to run

1. by html file (by browser)
2. by using appletviewer

Running an applet by html file:

```
import java.applet.Applet;
import java.awt.Graphics;
public class FirstApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Hello world!",100,150);
    }
}
```

html file:

FirstApplet.html

```
<html>
<body>
<applet code = "FirstApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

The Applet can only run on browser if and only if the browser having applet plugging software.

Running an applet by using Appletviewer

```
import java.applet.Applet;
import java.awt.Graphics;
public class FirstApplet extends Applet
{  public void point(Graphics g)
   {  g.drawString("Hello world!",100,150);
   }
}
/*
 <applet code="FirstApplet.class" width="300" height="300">
 </applet>
*/
```

compilation and running of an applet program using appletviewer:

```
javac FirstApplet.java
appletviewer FirstApplet.java
```

Illustration of sample applet with all the life cycle methods.

```
import java.applet.Applet;
import java.awt.Graphics;
public class FirstApplet extends Applet
   {  public void init()
      {  s.o.p("Applet Initialized");
      }
      public void start()
      {  s.o.p("Applet started");
      }
      public void stop()
      {  s.o.p("Applet stopped");
      }
```

```
public void destroy()
{   S.o.p("Applet Destroyed");
}
public void paint(Graphics g)
{   Font f = new Font("Serif", Font.BOLD, 32);
    g.setFont(f);
    g.drawString("Hello world!", 100, 150);
}
}
/*
<applet code = "FirstApplet.class" width = "300" height = "300">
</applet>
*/
```

o/p:

javac FirstApplet.java

appletviewer FirstApplet.java

Console

Applet Initialized
Applet started
Applet stopped
Applet started
Applet Destroyed

Applet window

Methods of Graphics class:

The Graphics class is available in java.awt.Graphics

1. public abstract void drawString(String s, int x, int y)
   displays a string at specified xy coordinates
2. public abstract void drawRect(int x, int y, int w, int h)
   draws a rectangle with specified width (w) and height (h) at specified xy coordinates.
3. public abstract void drawLine(int x1, int y1, int x2, int y2)
   draws a line from x1 y1 to x2 y2
4. public abstract void drawArc(int x, int y, int w, int h, int startAngle, int ArcAngle)
   Displays an arc either circular or eliptical
5. public abstract void drawOval(int x, int y, int w, int h)
   Displays a oval with specified width and height
6. public abstract void setColour(Colour c)
   Sets the graphics current colour to specified colour

* The Colour is a built in class of java.awt package, this class is having several constants represents different colours. For example Colour.GREEN (or) Colour.green

7. public abstract void setFont(Font f)
   sets the graphics current font to specified font

* Font is a built in class of java.awt package, the Font class is having several constants. For example.

Font.Serif

8. public abstract void fillRect(int x, int y, int w, int h)
draws a rectangle and fills with the default colour or specified colour.

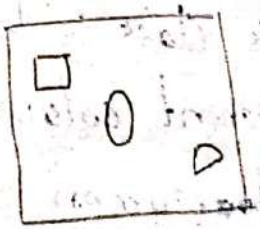
9. public abstract void fillOval(int x, int y, int w, int h)

10. public abstract void fillArc(int x, int y, int w, int h, int startAngle, int arcAngle)
draws a arc and fills with the default color or specified color

11. public abstract boolean drawImage(Image i, int x, int y, ImageObserver observer)
displays an image at particular xy coordinates.

Write a java applet program to display different shapes filled with green

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Colour;
class Applet1 extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Colour.green);
        g.fillRect(50, 150, 20, 40);
        g.fillOval(150, 100, 25, 50);
        g.fillArc(200, 250, 50, 55, 30, 60)
    }
}
```



/*
<applet code
</applet>
*/

import java.
import java
public clas
{
Imag
publ
{
}
p
{
}
}
/* <ap
</
*/

```
/*
<applet code = "Applet1.class" width = "350" height = "350"
</applet>
*/


/*
import java.applet.Applet;
import java.awt.*;
public class Sam extends Applet.
{
    Image i;
    public void init()
    {   i = getImage(getDocumentBase(), "f1.jpg");
    }
    public void paint(Graphics g)
    {   g.drawImage(i, 50, 50, this);
    }
}
/*<applet code = "Sam.class" width = "50" height = "50">
</applet>
*/
```

* getting parameter values from the html file or getting parameter values

getParameter() is a method of Applet class to retrive the value of a parameter of a html file

program

```
import java.applet.Applet;
// import java.awt.*;
public class Sam extends Applet
{
    String s;
    public void init()
    {
        s1=getParameter("msg1");
        s2=getParameter("msg2");
    }
    public void paint(Graphics g)
    {
        g.drawString(s1,50,50);
        g.drawString(s2,150,150);
    }
}
```

```
<html>
<body>
<applet code=Sam1.class width=300 height=300>
<param name="msg1" value="svec">
<param name="msg2" value="CSE">
</applet>
</body>
</html>
```

```
/*
<applet code=Sam1.class
width=300 height=300
<param name="msg1"
value="svec">
<param name="msg2"
value="CSE">
</applet>
*/
```

svec
CSE

getting

retrive

//displaying digital clock using applet.

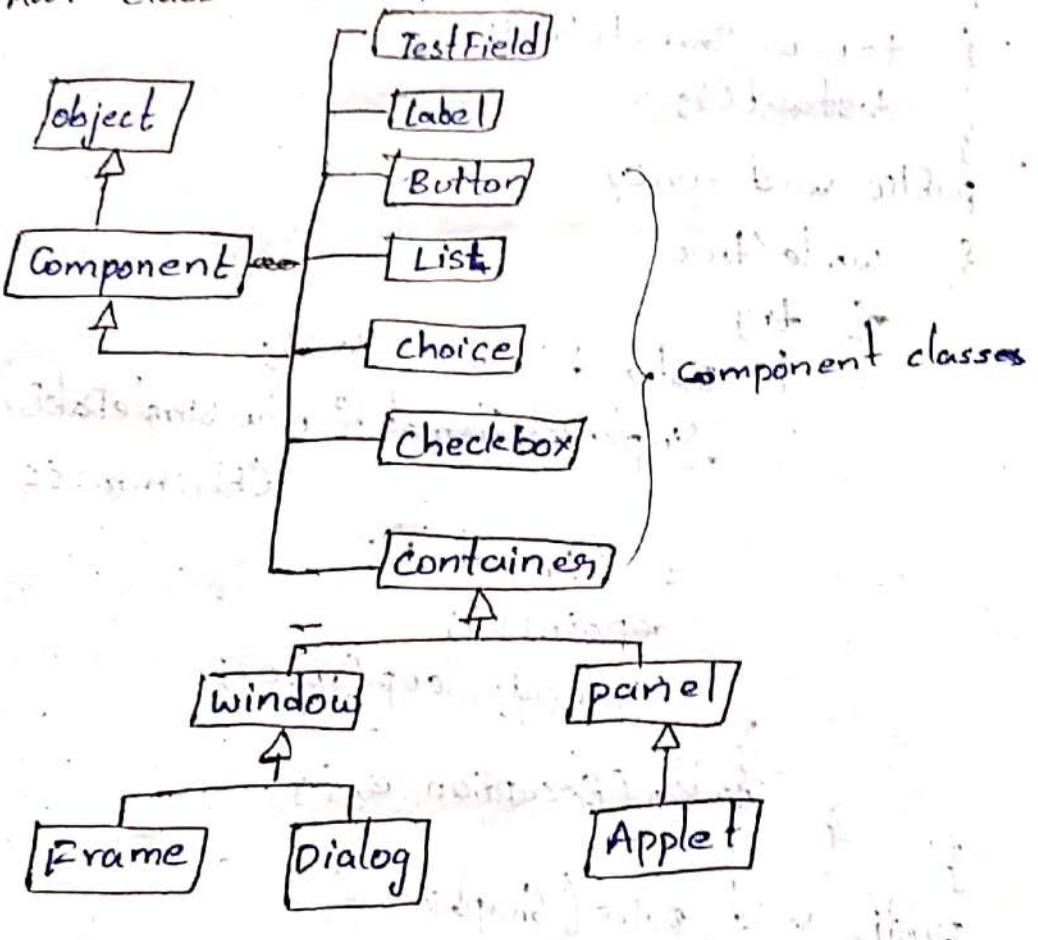
```
import java.applet.Applet;
import java.awt.*;
import java.util.*;
import java.text.*;
public class Sam1 extends Applet implements Runnable
{   String s;
    Thread t;
    public void init()
    {   t=new Thread(this);
        t.start();
    }
    public void run()
    {   while(true)
        {   try
            {   Date d=new Date();
                SimpleDateFormat f=new SimpleDateFormat
                                   ("hh:mm:ss");
                s=f.format(d);
                repaint();
                Thread.sleep(1000);
            }
            catch(Exception e){}
        }
    }
    public void paint(Graphics g)
    {   g.drawString(s,50,150);
    }
}
/*
<applet code=Sam1.class width=300 height=300>
</applet>
*/
```

AWT (Abstract Window Toolkit)

The AWT API (or) Components are used for developing Graphical user Interphase (or) window based applications. All the AWT components are available in java.awt package.

The AWT Components are heavy weight, because it uses underlined OS resources

AWT class Hierarchy:



Container

container is a Components which holds other Components like Buttons, Labels, textfields, etc.

Window:

Window is a container that have no boarders and menu bars, so we must use frame

Panel:
Panel is a container that have no title bar and menu bar.

Frame:
Frame is container that can have boarders and title bar.

Basic methods of Component class:

1. public void add(Component c)
   Adds a component to the container
2. public void setSize(int width, int height)
   Sets the size of a component
3. public void setVisible(boolean status)
   This method takes a boolean value either true (or) false. If it is true then only the componant is visible the default is false.
4. public void setLayout(LayoutManager l)
   Sets the layout for a component if we do not want to set any layout then we can pass null.

Creating a frame

There are two ways to create a Frame

1. using inheritance. ("is-a" kind of relation ship)
   extending a Frame class
2. Using association ("has-a" relationship)
   Creating object of Frame class

Illustration of extending a Frame class

```
import java.awt.*;
class SampleFrame extends Frame
{
    SampleFrame()
    {
        Button b1 = new Button("Click");
        add(b1);
        b1.setBounds(30,30,10,20);
        setSize(300,300);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SampleFrame s = new SampleFrame();
        // new SampleFrame();
    }
}
```

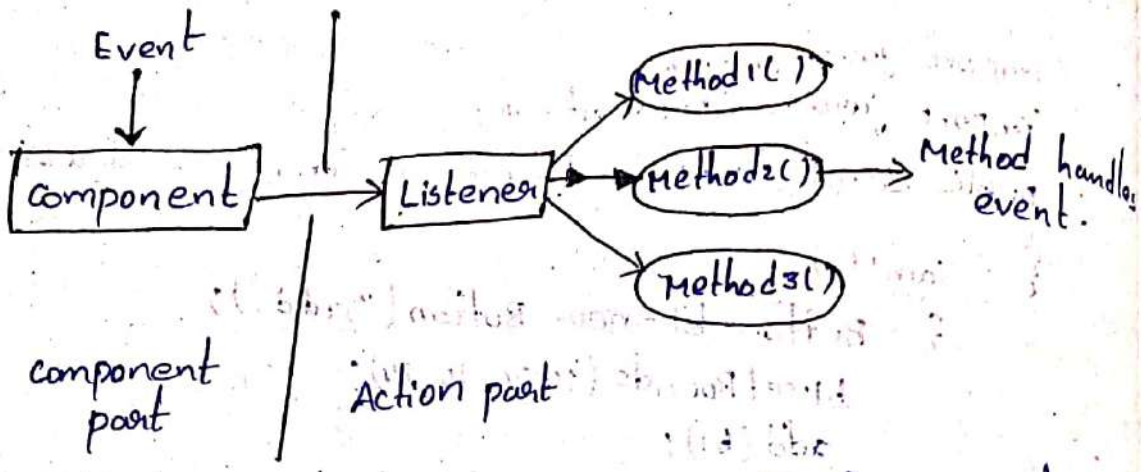
Illustration of creating a frame by using association

```
import java.awt.*;
class Sam
{
    Sam()
    {
        Frame f = new Frame();
        Button b1 = new Button("Click");
        f.add(b1);
        b1.setBounds(30,30,10,20);
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        Sam s = new Sam();
    }
}
```

Event handling:

```
import java.awt.*;
import java.awt.event.*;
public class Sam1 extends Frame implements WindowListener
{   Sam1()
    {   Button b1=new Button("add");
        b1.setBounds(10,20,10,10);
        add(b1);
        setSize(500,500);
        setVisible(true);
        setLayout(null);
        addWindowListener(this);
    }
    public void windowClosing(WindowEvent e)
    {   dispose();
    }
    public void windowClosed(WindowEvent e)
    {
    }
    public void windowOpened(WindowEvent e)
    {
    }
    public void windowActivated(WindowEvent e)
    {
    }
    public void windowDeactivated(WindowEvent e)
    {
    }
    public void windowIconified(WindowEvent e)
    {
    }
    public void windowDeiconified(WindowEvent e)
    {
    }
    public static void main(String args[])
    { new Sam1();
    }
}
```

Event Delegation Model:



Event Delegation Model is the process; if an event is performed on a component, then that event is delegated to the listener, the corresponding handler i.e the method handles that event.

Event is the change of state of a component.

For every event, for listening those events there are corresponding classes and interfaces in java.awt.event package.

Java event classes and listener interfaces.

Event classes	Listener Interfaces
1. ActionEvent	1. ActionListener
2. MouseEvent	2. MouseListener, MouseMotionListener
3. KeyEvent	3. KeyListener
4. ItemEvent	4. ItemListener
5. TextEvent	5. TextListener
6. WindowEvent	6. WindowListener
7. AdjustmentEvent	7. Adjustment Listener

8. ComponentEvent
9. ContainerEvent
10. FocusEvent
11. MouseWheelEvent

8. ComponentListener
9. ContainerListener
10. FocusListener
11. MouseWheelListener

steps to perform event handling:

Step1: Register the component with the listener.

Registration Methods:

For registering the component with the listener many classes provides the registration methods.

* Button

public void addActionListener(ActionListener l)

* TextField

public void addActionListener(ActionListener l)

public void addTextListener(TextListener l)

* TextArea

public void addTextListener(TextListener l)

* Checkbox

public void addItemListener(ItemListener l)

* Choice

public void addItemListener(ItemListener l)

* MenuItem

public void addActionListener(ActionListener l)

* List

public void addActionListener(ActionListener l)

public void addItemListener(ItemListener l)

Event handling by implementing ActionListener (ListenerInterface)

Event handling by anonymous class

Event handling by implementing ActionListener

The corresponding handler for handling ActionEvent is

public void actionPerformed(ActionEvent e)

import javaai.

E.

class Saml—e1

```
import java.awt.*;
import java.awt.Event;
class Saml extends Frame implements WindowListener,
                                    ActionListener
{
    Saml()
    { Button * b1 = new Button();
      b1.setBounds(10,10,10,10);
      add(b1);
      b1.addActionListener(this);
      setSize(500,500);
      setVisible(true);
      setLayout(null);
      addWindowListener(this);
    }
    public void windowClosing(WindowEvent e)
    {
        dispose(); // system.exit(0);
    }
public void windowClosed(WindowEvent e)
{
```

```
public void windowOpened(WindowEvent e)
{
}
public void windowActivated(WindowEvent e)
{
}
public void windowDeactivated(WindowEvent e)
{
}
public void windowIconified(WindowEvent e)
{
}
public void windowDeiconified(WindowEvent e)
{
}
public void actionPerformed(ActionEvent e)
{  S.O.P("Hello world!");
}
public static void main(String args[])
{   new Sam1();
}
}
```

Listeners and their methods

12/2/20

EventListeners and their methods (or) Handlers

Component	Listener	Methods (or) Handlers
1. Button	ActionListener	public void actionPerformed (ActionEvent e)
2. Checkbox	ItemListener	public void itemStateChanged (ItemEvent e)
3. CheckboxGroup	ItemListener	public void itemStateChanged (ItemEvent e)
4. TextField	i. ActionListener	public void actionPerformed (ActionEvent e)
	ii. FocusListener	i. public void focusGained (FocusEvent e)
		ii. Public void focusLost. (FocusEvent e)
5. TextArea	i. ActionListener	public void actionPerformed (ActionEvent e)
	ii. FocusListener	i. public void focusGained (FocusEvent e)
		ii. public void focusLost (FocusEvent e)
6. Choice	i. ActionListener	public void actionPerformed (ActionEvent e)
	ii. ItemListener	public void itemStateChanged (ItemEvent e)
7. List	i. ActionListener	pubic void actionPerformed (ActionEvent e)

8. Frame

9. Scro

10. key (not

	ii, ItemListener	public void ItemStateChanged (ItemEvent e).	
8. Frame	WindowListener	i public void windowClosing (WindowEvent e)	
		ii, public void windowClosed (WindowEvent e)	
		iii, public void WindowOpened (WindowEvent e)	
		iv public void windowActivated (WindowEvent e)	
		v public void windowDeactivated (WindowEvent e)	
		vi public void windowIconified (WindowEvent e)	
		vii, public void windowDeiconified (WindowEvent e)	
9. Scrollbar	i AdjustmentListener	public void adjustmentValue Changed (AdjustmentEvent e)	
10.	ii, MouseMotionListener	i public void mouseMoved (MouseEvent e)	
		ii, public void mouseDragged (MouseEvent e)	
10. keyboard (not a component)	keyListener	public void keyPressed (KeyEvent e)	
		public void keyReleased (KeyEvent e)	
		public void keyTyped (KeyEvent e)	

11. Mouse (Not a component)

i. MouseListener
- i. public void mouseClicked (MouseEvent e)
- ii. public void mouseEntered (MouseEvent e)
- iii. public void mouseExited (MouseEvent e)
- iv. public void mousePressed (MouseEvent e)
- v. public void mouseReleased (MouseEvent e)

ii. MouseMotionListener
- i. public void mouseDragged (MouseEvent e)
- ii. public void mouseMoved (MouseEvent e)

12. Label

—

—

// Illustration of a Button component with ActionListener

```
import java.awt.*;
import java.awt.event.*;
class Sami implements ActionListener
{   Frame f;
    Button b1, b2;
    Sami()
    {   f=new Frame("Button and ActionListener");
        b1=new Button("click1");
        b2=new Button("click2");
        f.setSize(500,500);
        f.setVisible(true);
```

}

```
f.setLayout(null);
b1.setBounds(50,80,80,30);
b2.setBounds(50,100,80,30);
b1.addActionListener(this);
b2.addActionListener(this);
f.add(b1);
f.add(b2);
}
public void actionPerformed(ActionEvent e)
{   if(e.getSource()==b1)
        S.O.P("Hello World!");
    else
        s.o.p("Hello SVEC");
/* if(e.getActionCommand()=="Click")
    S.O.P("Helloworld!");
    else
    s.o.p("Hello SVEC");
*/
}
public static void main(String args[])
{   new Sam1();
}
}
```

Illustration of ActionListener with Anonymous inner class

```
import java.awt.*;
import java.awt.event.*;
class Sam1
{
    Frame f;
    Button b1;
    Sam1()
    {
        f=new Frame("Button and ActionListener");
        b1=new Button("click");
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(null);
        b1.setBounds(50,80,80,30)
        b1.addActionListener(
                        new ActionListener()
                        { public void actionPerformed
                            (ActionEvent e)
                        { if(e.getSource()==b1)
                            s.o.p("Hello world");
                        //else
                        // s.o.p("Hello sree");
                        } }
                    );
        f.add(b1);
}
public static void main(String args[])
{
    new Sam1();
```

Adapter classes:

The adapter classes provides the default implementations for listener interface methods.

Listeners	Adapters
1. WindowListener	1. WindowAdapter
2. MouseListener	2. MouseAdapter
3. MousMotionListener	3. MouseMotionAdapter
4. KeyListener	4. KeyAdapter
5. FocusListener	5. FocusAdapter
6. ComponentListener	6. ComponentAdapter
7. ContainerListener	7. ContainerAdapter

All these Adapter classes are available in java.awt.event package.

Illustration of WindowAdapter for closing the Frame

```
import java.awt.*;
import java.awt.event.*;
class Sam1 extends WindowAdapter
{   Frame f;
    Sam1()
    {
        f=new Frame("WindowAdapter");
        f.setSize(500, 500);
        f.setVisible(true);
        f.setLayout(null);
        f.addWindowListener(this);
    }
    public void windowClosing (WindowEvent e)
    {   f.dispose();
    }
    public static void main(String args[])
    {   new Sam1();
    }
}
```

Illustration of WindowAdapter with Annonymous inner class

```
import java.awt.*;
import java.awt.event.*;
class Sam1
{   Frame f;
    Sam1()
    {   f=new Frame("Window Adapter");
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(null);
        f.addWindowListener(
                            new WindowAdapter()
                            {
                            public void windowClo
                                (WindowEvent e)
                            {   f.dispose();
                            }
                            }
                            );
    }
    public static void main(String args[])
    {   new Sam1();
    }
}
```

Illustration of checkbox with ItemListener

```
import java.awt.*;
import java.awt.event.*;
class Sam1 implements ItemListener
{   Frame f;
    Label l;
    Checkbox c1,c2;
    Sam1()
    {   f=new Frame("Checkboxes");
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(null);
        f.addWindowListener(new WindowAdapter()
                            {public WindowClosing
                                (WindowEvent e)
                             {f.dispose();
                             }
                            }
                           );
        c1=new Checkbox("Java",true);
        c2=new Checkbox("Python");
        L=new Label("subjects");
        l.setBounds(50,50,200,30);
        c1.setBounds(50,100,100,30);
        c2.setBounds(50,150,100,30);
        c1.addItemListener(this);
        c2.addItemListener(this);
        f.add(L);
        f.add(c1);
        f.add(c2);
    }
    public void itemStateChanged(ItemEvent e)
    {   if(e.getStateChanged()==1)
```

```
            l.setText(e.getItem() + "checked");
        else
            l.setText(e.getItem() + "Unchecked");
    }
    public static void main(String args[])
    {   new Sami();
    }
}
```

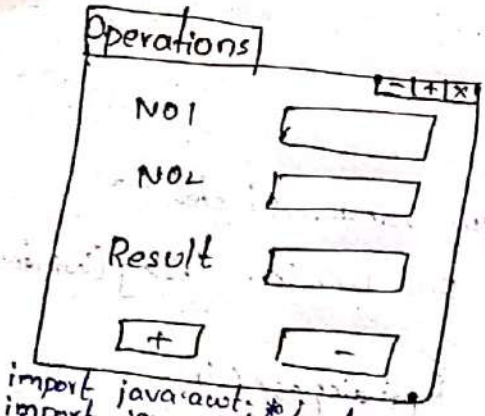
Illustration of CheckboxGroup with ItemListener.

Constructors of Checkbox

1. Checkbox(String s),
2. checkbox(String s, boolean b)
3. Checkbox()
4. Checkbox(String s, CheckboxGroup cbg, boolean b)

```
import java.awt.*;
import java.awt.event.*;
class Sami implement ItemListener
{   Frame f;
    Label l;
    Checkbox c1, c2;
    CheckboxGroup cbg;
    Sami()
    {   cbg = new CheckboxGroup();
        f = new Frame("Checkbox Group");
        l = new Label("Subject");
        c1 = new checkbox("java", cbg, true);
        c2 = new checkbox("python", cbg, false);
```

```
f.setSize(500,500);
f.setVisible(true);
f.setLayout(null);
f.addWindowListener(
                    new WindowAdapter()
                    { public void windowClosing(WindowEvent
                                                e)
                      { f.dispose();
                      }
                    }
                   );
else
 l.setBounds(50,50,100,30);
 c1.setBounds(50,100,100,30);
 c2.setBounds(50,150,100,30);
 c1.addItemListener(this);
 c2.addItemListener(this);
 f.add(l);
 f.add(c1);
 f.add(c2)
}
public void itemStateChanged(ItemEvent e)
 {  if(e.getStateChange()==1)
        l.setText(e.getItem()+"Checked");
    else
        l.setText(e.getItem()+"Unchecked");
 }
public static void main(String args[]);
{   new Sam1();
}
}
```




```
import java.awt.*;
import java.awt.event.*;
class Sam implements ActionListener
{
    Frame f;
    Label l1, l2, l3;
    Button b1, b2;
    TextField t1, t2, t3;
    Sam1()
    {
        f = new Frame("Operations");
        l1 = new Label("NO1");
        l2 = new Label("NO2");
        l3 = new Label("result");
        b1 = new Button("+");
        b2 = new Button("-");
        t1 = new TestField();
        t2 = new TestField();
        t3 = new TestField();
        f.setSize(1000, 1000);
        f.setVisible(true);
        f.setLayout(null);
        f.addWindowListener(new WindowAdapter()
                { public void windowClosing(Window
                    Event e)
```

```
        {
            f.dispose();
        }
    }
);

L1.setBounds(50,50,50,30);
L2.setBounds(50,100,50,30);
L3.setBounds(50,150,50,30);
b1.setBounds(150,100,50,30);
b2.setBounds(200,200,50,30);
t1.setBounds(100,50,100,20);
t2.setBounds(100,100,200,30);
t3.setBounds(100,150,200,30);
f.add(L1);
f.add(L2);
f.add(L3);
f.add(b1);
f.add(b2);
f.add(t1);
f.add(t2);
f.add(t3);
b1.addActionListener(this);
b2.AddActionListener(this);
}
public void ActionPerformed(ActionEvent e)
{
    if(e.getSource()==b1)
    int i=t1.getText
    int i = Integer.ParseInt(t1.getText);
    int j = Integer.parseInt(t2.getText);
    if(e.getsource()==b1)
    { t3.setText(String.valueOf(i+j))
    }
```

```
        else
            t3.setText(String.valueOf(i-j));
    }
    public static void main(String args[])
    {   new Sam1();
    }
}
```

// Illustration of MouseListener

```
import java.awt.*;
import java.awt.event.*;
class Sam1 implements MouseListener
{   Frame f;
    Label l;
    Sam1()
    {   f=new Frame("MouseListener");
        L=new Label("subject");
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(null);
        L.setBounds(50,50,200,30);
        f.add(l);
        f.addMouseListener(this);
        f.addWindowListener(new WindowAdapter()
                            {  pv WindowClosing()
                               { f.dispose();
                               }
                            }
                           );
}
```

```
public void mouseClicked(MouseEvent e)
{   L.setText("Mouse clicked");
}
public void mouseEntered(MouseEvent e)
{   L.setText("Mouse Entered");
}
public void mousePressed(MouseEvent e)
{   L.setText("Mouse Pressed");
}
public void mouseReleased(MouseEvent e)
{   L.setText("Mouse Released");
}
public void mouseExited(MouseEvent e)
{   L.setText("Mouse Exited");
}
public static void main(String args[])
{   new Sam1();
}
}
```

Illustration of MouseAdapter to know the click counts and to know the coordinates of each click

```
import java.awt.*;
import java.awt.event.*;
class Sam extends MouseAdapter
{   Frame f;
    Label l1,l2;
    int i=0;
    Sam()
    {   f=new Frame("Operations");
        f.setSize(800,800);
        f.setVisible(true);
```

```
f.setLayout(null);
f.addWindowListener(
                    new WindowAdapter()
                    { public void windowClosing(WindowEvent
                      { f.dispose();
                      }
                    }
                   );
l1=new Label("Mouse coordinates");
l2=new Label("Mouse clicks");
l1.setBounds(50, 50, 200, 30);
l2.setBounds(50, 50, 200, 30);
f.add(l1);
f.add(l2);
f.addMouseListener(this);
}
public void mouseClicked(MouseEvent e)
{   l1.setText("x: "+e.getX()+"  "+"y: "+e.getY());
    i+=e.getClickCount();
    l2.setText(String.valueOf(i));
}
public static void main(String args[])
{   new Sum();
}
}
```

```
// Illustration of mouseMotionListener
import java.awt.*;
import java.awt.event.*;
class Sam implements MouseMotionListener
{   Frame f;
    Label L;
    Sam()
    {
        f=new Frame("Mouse MotionListener");
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(null);
        l=new Label("Mouse Events");
        f.add(l);
        f.setBounds(100,100,100,30);
        f.addWindowListener( new WindowAdaptor().
                             {.public void windowClosing(WindowEvent
                                                         e)
                              { f.dispose(); }
                             }
                           );
        f.addMouseMotionListener(this);
    }
    public void mouseMoved(MouseEvent e)
    { l.setText("Mouse Moved");
    }
    public void mouseDragged(MouseEvent e)
    { l.setText("Mouse Dragged");
    }
    public static void main(String args[])
    { new Sam();
    }
}
```

```
// Illustration of mouseMotionListener
import java.awt.*;
import java.awt.event.*;
class Sam implements MouseMotionListener
{   Frame f;
    Label l;
    Sam()
    {
        f=new Frame("Mouse MotionListener");
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(null);
        l=new Label("Mouse Events");
        f.add(l);
        f.setBounds(100,100,100,30);
        f.addWindowListener( new WindowAdapter()
                             {public void windowClosing(WindowEvent e)
                              { f.dispose(); }
                             }
                           );
        f.addMouseMotionListener(this);
    }
    public void mouseMoved(MouseEvent e)
    { l.setText("Mouse Moved");
    }
    public void mouseDragged(MouseEvent e)
    { l.setText("Mouse Dragged");
    }
    public static void main(String args[])
    { new Sam();
    }
}
```

Illustration of KeyListener

```
import java.awt.*;
import java.awt.event.*;
class Sam implement keyListener
{
    Frame f;
    Label l;
    TextField t;
    Sam()
    {
        f=new Frame("key Listener");
        t= new TextField("");
        l = new Label( );
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(null);
        l.setBounds(250,250, 100, 30);
        t.setBounds(250,250, 100,30);
        f.add(t);
        f.add(l);
        t.addKeyListener(this);
        f.addWindowListener( new WindowAdaptor()
                { public void windowClosing(WindowEvent
    }
    public void keyPressed(keyEvent e)
    {   l.setText("key pressed");
    }
    public void keyReleased(keyEvent e)
    {   l.setText("key Released");
    }
```

```
{ f.dispose(); }
});
```

```
public void keyTyped(KeyEvent e)
{   l.setText("key Typed");
}
public void static void main(String args[]);
{
    new Sam();
}
}
```

```
import java.awt.*;
import java.awt.event.*;
class Sam extends KeyAdaptor
{   Frame f;
    TextField t;
    Label l;
    String s="";
    Sam()
    {   f=new Frame("Key Adapter");
        t=new TextField();
        l=new Label();
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(null);
        f.addWindowListener(new WindowAdapter()
                { public void windowClosing
                    (WindowEvent e)
                { f.dispose();
                }
                }
        );
        t.setBounds(50,50,200,30);
        t.setBounds(50,150,200,30);
        f.add(t)
        f.add(l);
```

```
        t.addKeyListener(this);
    }
    public void keyTyped(KeyEvent e)
    {   s+= e.getChar();
        t.setText("key-Typed:"+e.getkeyChar());
        l.setText(s);
    }
    public static void main(String args[]);
    {
        new Sam();
    }
}
```

// Illustration of scrollbar with AdjustmentListener

```
import java.awt.*;
import java.awt.event.*;
class Sam implements AdjustmentListener
{   Frame f;
    Label l;
    Button b;
    Scrollbar s;
    Sam()
    {   f= new Frame("Adjustment listener");
        f.setSize(500,500);
        f.setLayout(null);
        f.setVisible(true);
        l= new Label("Adjustment scroll bar");
        l.setBounds(100,100,100,30);
        f.add(l);
        f.addWindowListener( new windowAdapter(WindowEvent e)
                                {   public void windowClosing
                                    (WindowEvent
```

```
            {  f.dispose();
            }
        }
    );
     s = new Scrollbar();
     s.setBounds(100, 150, 100, 30);
    s.addAdjustmentListener(this);
     f.add(s);
}
public void adjustmentValueChanged(AdjustmentEvent e)
{
     l.setText("Scrollbar adjusted value: " + e.getValue());
}
public static void main(String args[])
{
     new Sam();
}
```

o/p:

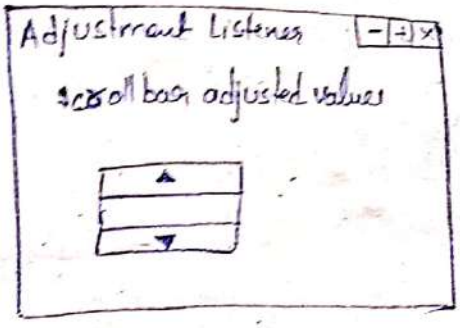


Illustration of TextArea with

```
import java.awt.*;
import java.awt.event.*;
class Sam implements ActionListener
{  Frame f;
   Label l;
   Button b;
   TextArea t;
   Sam()
   {  f=new Frame("Text Area");
      f.setSize(500,500);
      f.setVisible(true);
      f.setLayout(null);
      L= new Lable("cha ");
      l.setBounds(50,50,200,30);
      f.add(l);
      b = new Count Button("Count");
      b.setBounds(50,150 400, 50,50);
      f.add(b);
      t = new TextArea();
      t.setBounds(50,150,200,200)
      f.add(t);
      b.addActionListener(this);
      f.addWindowListener( new WindowAdapter()
                           { public void windowClosing(WindowEvent e)
                             { f.dispose();
                             }
                           }
                         );
}
```

```
public void actionPerformed(ActionEvent e)
{
        String s = t.getText();
        String si[] = s.split(" ");
        int i
        int
        l.setText("character count:"+s.length()+"  "+
                "word count : " + si.length);
}
public static void main(String args[])
{   new Sam();
}
}
```

Illustration of Choice component

```
import java.awt.*;
import java.awt.event.*;
class Sam implements ActionListener
 {   Frame f;
     Button b;
     Label l;
     Choice c;
     Sam()                                   Example
     {   f=new Frame("Choice component");
         f.setSize(500,500);
         f.setVisible(true);
         f.setLayout(null);
         b = new Button("select");
         b.setBounds(50,400,50,50);
         f.add(b);
```

```
c=new Choice();
c.setBounds(50,100,200,200);
c.add("Java");
c.add("C");
c.add("C++");
c.add("Python");
f.add(c);
l=new Label("subjects");
l.setBounds(50,50,200,30);
f.add(l);
f.addWindowListener(new WindowAdapter()
                    { public void windowClosing
                                  (WindowEvent e)
                      {  f.dispose());
                      }
                    }
                    );
b.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{ l.setText("subject choice:"+c.getItem(c.getSelectedIndex()));
}
public static void main(String args[])
{   new Sam();
}
```

Illustration of list to select the multiple items

```
import java.awt.*;
import java.awt.event.*;
public class sam implements ActionListener
{ Frame f;
  Label l;
  List li;
  Button b;
  sam()
  { f=new Frame("List illustration");
    f.setSize(800,800);
    f.setVisible(true);
    f.setLayout(null);
    l=new Label("subject list:");
    l.setBounds(50,50,300,20);
    li=new List(4,true);
    li.setBounds(50,250,300,150);
    li.add("java");
    li.add("python");
    li.add("C++");
    li.add("C");
    b=new Button("select");
    b.setBounds(50,150,300,20);
    f.add(b);
    f.add(l);
    f.add(li);
    b.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{ String items="";
```

```
        for(String s: lg.getSelectedItems())
        {   items += s;
        }
        l.setText("Subjects choice:" + items);
    }
    public static void main(String args[])
    {   new Sam();
    }
}
```

Illustration of menu, MenuBar and MenuItem

```
import java.awt.*;
import java.awt.event.*;
public class Sam
{   Frame f;
    MenuBar mb;
    Menu m;
    MenuItems i1,i2,i3;
    Sam()
    {   f=new Frame("Menu events");
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(null);
        mb=new MenuBar();
        m = new Menu("File");
        i1 =new MenuItem("new");
        i2 =new MenuItem("Save");
        i3 =new MenuItem("Open");
        m.add(i1);
        m.add(i2);
        m.add(i3);
        mb.add(m);
```

```
        f.setMenuBar(mb);
    }
    public static void main(String args[])
    {   new sam();
    }
}
```

Illustration of PopupMenu

```
import java.awt.*;
import java.awt.event.*;
public save class Sam extends MenuAdapter implements ActionListener
{   Label l;
    Frame f;
    PopupMenu pm;
    MenuItem i1, i2, i3;
    Sam()
    {   f=new Frame("Popup Menu");
        f.setSize(800, 800);
        f.setVisible(true);
        f.setLayout(null);
        l=new Label();
        l.setBounds(50, 50, 100, 100);
        pm=new PopupMenu();
        i1=new MenuItem("new");
        i2=new MenuItem("Save");
        i3=new MenuItem("Open");
        i1.setActionCommand("new");
        i2.setActionCommand("save");
        i3.setActionCommand("Open");
        i1.addActionListener(this);
        i2.addActionListener(this);
        i3.addActionListener(this);
```

```
        pm.add(i1);
        pm.add(i2);
        pm.add(i3);
        f.add(pm);
        f.addMouseListener(this);
    }
    public void mouseClicked(MouseEvent e)
    {
        pm.show(f, e.getX(), e.getY());
    }
    public void actionPerformed(ActionEvent e)
    {
        if(e.getActionCommand()=="new")
            l.setText("New selected");
        else if(e.getActionCommand()=="save")
            l.setText("Save selected");
        else
            l.setText("open selected");
    }
    public static void main(String args[])
    {
        new Sam();
    }
}
```

Illustration of Dialog:-

```
import java.awt.*;
import java.awt.event.*;
class Sam implements ActionListener
{  Frame f;
   Button b;
   Dialog d;
   Label L;
   Sam()
   {  f = new Frame("Dialog");
      b=new Button("Display Dialog");
      d = new Dialog(f, "dialog", false);
      l=new Label("label");
      f.setSize(500,500);
      b.setBounds(50,50,100,30);
      f.setVisible(true);
      f.setLayout(null);
      f.addWin
      d.setBounds(50,50,200,150);
      f.add(b);
      d.add(l);
      b.addActionListener(this);
   }
   public void ActionPerformed(ActionEvent e)
   {   d.setVisible(true);
   }
   public static void main(String args[])
   {   new Sam();
   }
}
```

Illustration of panel

```
import java.awt.*;
import java.awt.event.*;
class Sam
{  Frame f;
   Button b1,b2;
   Panel P,P1;
   Sam()
   {  f=new Frame("Panel Example");
      P=new panel();
      P1=new Panel();
      b1=new Button("add");
      P.setBounds(50,800,200,200); P1.setBounds(200,80,200,
      b1.setBounds(50,100,80,30);
      b2=new Button("sub");
      b2.setBounds(100,100,80,30);
      f.setSize(800,800);
      f.setLayout(null);
      f.setVisible(true);
      P.setBackground(Colour.gray);
      P1.setBackground(colour.green);
      p.add(b1);
      p.add(b2);
      f.add(P);
      f.add(P1);
      f.setVisible(t
      f.setSize(800,800);
      f.setLayout(null);
      f.setVisible(true);
   }
   P
      f.addWindowListener(new WindowAdapter()
                  { public void windowClosing
                        (WindowEvent e)
                     { f.dispose();
                     }
                  }
      );
```

```
    Public static void main (String args[])
    {
        new Sam();
    }
}
```

Illustrate canvas class

```
import java.awt.*;
import java.awt.event.*;
class Sam extends Canvas
{
    Sam()
    { setBackground(Color.blue);
    }
    public void Paint(Graphics g)
    { g.setColor(Color.green);
      g.drawString("Hello", 50, 100);
      g.drawRect(50, 150, 50, 50);
    }
}
class Sam1
{ Frame f;
  Sam1()
  { f = new Frame("CanVas");
    Sam s = new Sam();
    s.setBounds(100, 100, 300, 300);
    f.add(s);
    f.setSize(800, 800);
    f.setLayout(null);
    f.setVisible(true);
    f.addWindowListener
  }
```

```
public static void main(String args[])
{ new Sam1();
}
}
```

Layout Managers

Layout Managers are used to arrange the components in particular maner.

LayoutManager is an interface that is implemented by all the classes of LayoutManagers

classes that represents layoutManagers

1. java.awt.BoarderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout

BoarderLayout:

BoarderLayout is used to arrange the components in five regions north, south, east, west and centes.

Each region may contain one component only.

It is default Layout for Frame or Window.

The BoarderLayout class provides 5 constants.

1. public static final int NORTH
2. public static final int SOUTH

3. public static final int EAST
4. public static final int WEST
5. public static final int CENTER

Constructors of BoarderLayout class:

1. BoarderLayout() - Creates a BoarderLayout with no gapes between the component.

2. BoarderLayout(int hgap, int vgap)
creates a BoarderLayout with given horizontal and vertical gap between the component.

default * f.setLayout(new BoarderLayout());

```
import java.awt.*;
import java.awt.event.*;
class Sam1
{  Frame f;
   Button b1, b2, b3, b4, b5;
   Sam1()
    {  f=new Frame("Panel Example");
       b1=new Button("AAA");
       b2=new Button("BBB");
       b3=new Button("CCC");
       b4=new Button("DDD");
       b5=new Button("EEE");
       f.add(b1, BoarderLayout.SOUTH);
       f.add(b2, BoarderLayout.NORTH);
       f.add(b3, BoarderLayout.CENTER);
       f.add(b4, BoarderLayout.EAST);
```

```
        f.add (b5, BoaadeaLayout. WEST);
        f.setsize (200,200);
        f.setVisible (true);
    }
    public static void main (String args)
    {   new Sam1();
    }
}
```

FlowLayout:

* FlowLayout is used to arrange the components in a line one after another in a flow.
* It is a default Layout for Applet or Panel
* Constants or fields of FlowLayout are
  1. public static final int LEFT
  2. public static final int RIGHT
  3. public static final int CENTER
  4. public static firial int LEADING
  5. public static final int TRAILING

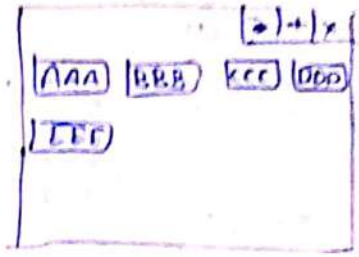
Constructor of FlowLayout class:

1. FlowLayout()
   Creates a new FlowLayout with centered allignment and a default 5 unit horizontal and vertical gap
2. FlowLayout (int allign)
   creates a new FlowLayout with the given allignment

and a default 5 unit horizontal and vertical gap

3. FlowLayout (int align, int hgap, int vgap)

o/p:




```
import java.awt.*;
import java.awt.event.*;
class Sam1
{   Sam1()
    f  f = new F
        Frame f;
        Button b1,b2,b3,b4,b5;
        Sam1()
        {  f=new Frame("panel Example");
           b1=new Button("AAA");
           b2=new Button("BBB");
           b3=new Button("CCC");
           b4=new Button("DDD");
           b5=new Button("EEE");
           f.add(b1);
           f.add(b2);
           f.add(b3);
           f.add(b4);
           f.add(b5);
           f.setLayout(new FlowLayout(FlowLayout.LEFT,
                                      50,50));
           f.setSize(500,500);
           f.setVisible(true);
        }
        public static void main(String args[]).
        {  new Sam1();
        }
}
```

GridLayout:

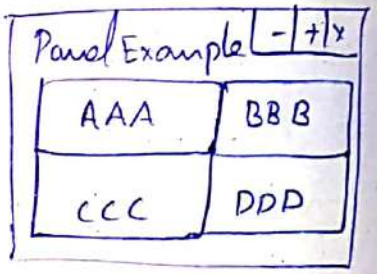
GridLayout is used to arrange the components in a rectangular grid. One component is displayed in each rectangle and

Constructors of GridLayout

1. GridLayout ( )
   Creates a GridLayout with one column and one row.
2. GridLayout (int rows, int column)
   Creates a GridLayout with number of rows and colu but no gap between the components.
3. GridLayout (int rows, int colums, int hgap, int vgap)
   Creates a GridLayout with given rows and colums along with horizontal and vertical gap.

Example for GridLayout

```
import java.awt.*;
import java.awt.event.*;
class Sam1
{   Frame f;
    Button b1,b2,b3,b4;
    Sam1()
    {   f=new Frame("Panel Example");
        b1=new Button("AAA");
        b2=new Button("BBB");
        b3=new Button("CCC");
        b4=new Button("DDD");
```



```
        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.setLayout(new GridLayout(2,2));
        f.setSize(200,200);
        f.setVisible(true);
    }
    public static void main(String args[])
    {   new sam1();
    }
}
```

BoxLayout:

Used to arrange the components either horizontally or vertically.

Fields or Constants:

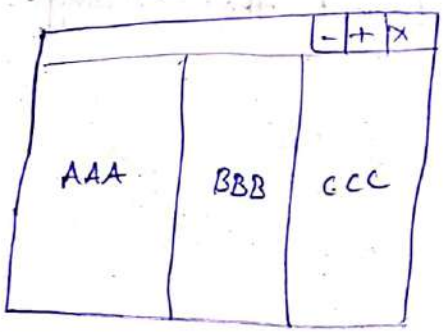
1. public static final int X_AXIS
2. public static final int Y_AXIS
3. public static final int LINE_AXIS
4. public static final int PAGE_AXIS

Constructors:

1. BoxLayout(Container c, int axis)
   Creates a BoxLayout that arranges the components in a given axis

```
import java.awt.*;
import javax.swing.*;
class Sam
{   Frame f;
    Button b1, b2, b3;
    Sam()
    {   f=new Frame("Box Layout example");
        b1 =new Button ("AAA");
        b2 =new Button ("BBB");
        b3 =new Button("CCC");
        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.setLayout(new BoxLayout(f, BoxLayout.X_AXIS));
        f.setSize(200,200);
        f.setVisible(true);
    }
    public static void main(String args[])
    {   new Sam();
    }
}
```

o/p:



CardLayout:

CardLayout mar
that only one co
It treats each
Constructors:
CardLayout
Creates
Vertical gaps.
CardLayout
Creates
Vertical gaps
Methods of C
1. public voi
used
Container
2. public v
used
Containe
3. public
Use
Contain
4. publi
Use
5. publi
Us
give

CardLayout:

CardLayout manages the component in such a manner that only one component is visible at a time.

It treats each component as a card.

Constructors:

CardLayout()

Creates a cardLayout without horizontal and vertical gapes.

CardLayout(int hgap, int vgap)

. creates a CardLayout with given horizontal and Vertical gapes.

Methods of CardLayout class:

1. public void next(Container parent)

Used to flip to the next card of the given Container

2. public void previous(Container parent)

Used to flip to the previous card of the Container

3. public void first(Container parent)

Used to flip to the first card of the given Container

4. public void last(Container parent)

Used to flip to the last card of the given Container

5. public void show(Container parent, String name)

Used to flip to the specified card with the given name.

Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Sam implement ActionListener extends JFrame
{ //Frame f;
    Button b1,b2,b3;
    CardLayout cl;
    Container c;
    Sam()
    { //f=new Frame("Card Layout example");
        b1=new Button("AAA");
        b2=new Button("BBB");
        b3=new Button("CCC");
        c=getContentPane(&getContent);
        b1.addActionListener(this); b1.setBounds(50,50,100,50
        b2.addActionListener(this); b2.setBounds(50,50,100,50
        b3.addActionListener(this); b3.setBounds(50,50,100,50
        c.add(b1); "card1", b1);
        c.add(b2); "card2", b2);
        c.add("card3", b3);
        cl=new CardLayout();
        c.setLayout(cl);
        setSize(300,300); setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
    cl.next(c);
}
```

public sta
{ new
}
}

Swings:

Swings are
used to build
to create gr

Differences bet
AW

1. AWT Compo
platform dep
2. AWT The look
AWT Componer
platform to p
3. AWT Compon
weight beco
Undealined,
4. AWT compo

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

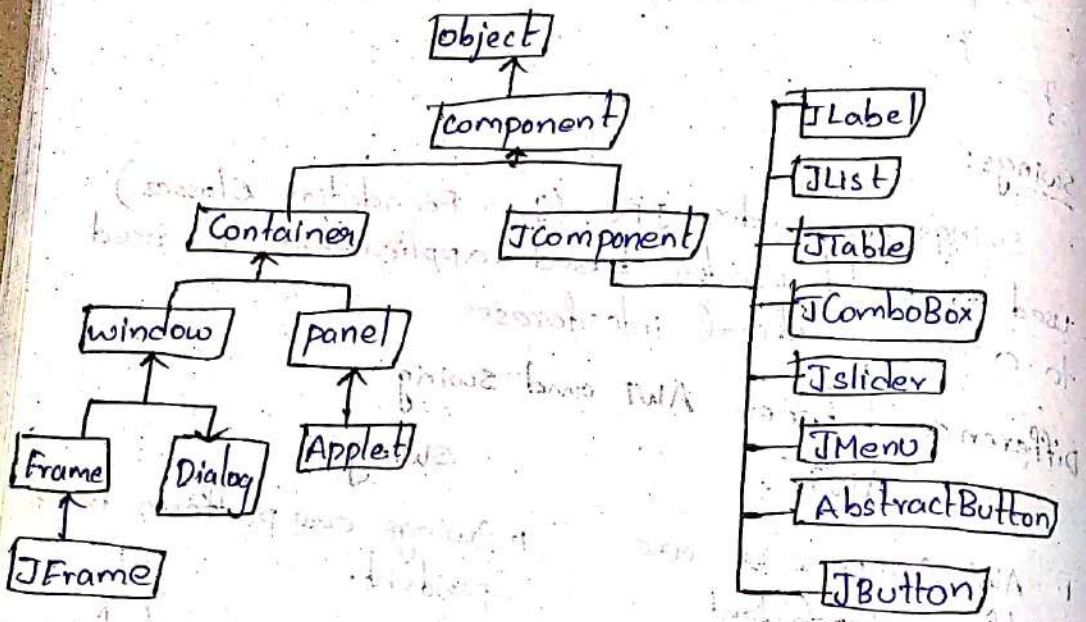
Swings:

Swings are under JFC (Java Foundation Classes) used to build window based applications. Or used to create graphical interfaceses

Differences between AWT and Swing

AWT	Swing
1. AWT components are platform dependent	1. Swings are platform independent.
2. The look and feel of AWT component differs from platform to platform	2. The look and feel of swings are same on every platform
3. AWT components are heavy weight because they uses underlined resources	3. Swings are light weight because they do not use underlined resources.
4. AWT components are less	4. Swing components are more.

Hierarchy of Java Swing classes



JFrame:

There are two ways to create a Frame.

All the swing components available in Javax.swing package. Javax is nothing but extended java.

Javax.Swing.JFrame is a subclass of Java.awt.Frame

1. By creating the object of JFrame class (association)
2. By extending the JFrame class (Inheritance)

Constructors of JFrame class:

1. JFrame()

   Creates a Frame without any title

2. JFrame(String title)

   Creates a Frame with title

To close the J
public voi

public

It is a
closing

Creating a
import j
class But
{ Butt
{

}
pu
{
}
}

To close the JFrame

public void setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)

public static final EXIT_ON_CLOSE

It is a constant to represent the close window closing

Creating a JFrame using association

```
import javax.swing.*;
class ButtonExample
{  ButtonExample()
   {  JFrame f=new JFrame("Button Exple");
      JButton b=new JButton("click");
      b.setBounds(100,100,100,40);
      f.setVisible(true);
      f.setLayout(null);
      f.add(b);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   psvm(String s)
   {  new ButtonExample();
   }
}
```

Creating JFrame using inheritance:

```
import javax.swings;
class Sam1 extends JFrame
{   Sam(String s)
    {   // super(s);
        JButton b=new JButton("click");
        b.setBounds(100,100,100,40);
        add(b);
        setTitle(s);
        setSize(300,400);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_clos
    }
    psvm(String args[])
    {   new Sam("Button Example");
    }
}
```

JButton: class declaration
public class JButton extends AbstractButton

constructors:

JButton() - Creates a button with no text and icon specified

JButton(String s) - creates a button with text specified

JButton(Icon i) - creates a button with icon object specified

Methods of JButton:

1. void setText(String s) :- It is used to set text to button
2. ~~void~~ String ~~getText~~ getText() :- It returns the text of the button

3. void setEnabled(boolean b) :- It is used to enable or disable the button (Default value is true)

4. void setIcon(Icon b) :- It is used to set icon to the button

5 Icon getIcon() :- It returns the icon of the button

6. void setMnemonic(int a) :- It is used to set a mnemonic to the Button (i.e., keyBoard shortcut.

```
import javax.swing.*;
class ButtonExample extends JFrame
{   ButtonExample(String s)
    {
        //super(s);
        JButton b= new JButton(new ImageIcon
                ("c:/users/public/pictures/Sample
                 pictures/chrysanthemum.jpg"));
        b.setBounds(100,100,100,40);
        b.setText("Click");
        b.setMnemonic('c');
        //b.setEnabled(false);
        add(b);
        setTitle(s);
        setSize(300,400);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_
                                 CLOSE);
    }
    public static void main(String args[])
    { new ButtonExample("Button Example");
    }
}
```

class declaration

JLabel: public class JLabel extends JComponent

Constructors:

1. JLabel()
2. JLabel(String s)
3. JLabel(Icon i)
4. JLabel(String s, Icon i, int horizontalAlignment)

Methods:

1. String getText()
2. void setText(String text)
3. void setHorizontalAlignment(int allignment)
4. Icon getIcon()
5. int getHorizontalAlignment()

JTextField:

class declaration

public class JTextField extends JTextComponent

constructors:

1. JTextField() :- Creates textfield with no text and no. number of colums
2. JTextField(String text) :- Creates text field with text
3. JTextField(String text, int columns) :-
4. JTextField(int columns)

JCheckbox

JRadioButton

JPasswordField:

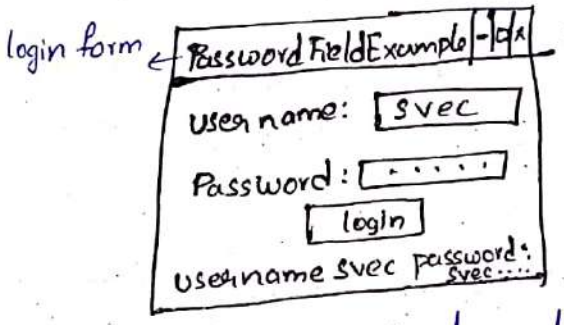
class declaration:

public class JPasswordField extends JTextField

Constructors.

1. JPasswordField()
2. JPasswordField(String text)
3. Jpassword Field(String s, int i)
4. JPasswordField(int i)

Illustration of password field or Login form.



The getPassword() method returns only few characters along with dots (...) of password field

JTable

Constructors:

JTabel()

JTable (object[][] rows, object[] columns)

Panes:

1. JOptionPane
2. JLayeredPane
3. JTappedPane
4. JSplitPane
5.

JOptionPane:

JOptionPane used to provide standard dialog boxes such as message boxes, Confirmation boxes and input boxes.

These Dialog boxes are used to display information or confirmation or get input from the user.

JOptionPane inherits from JComponent

class declaration:

public class JOptionPane extends JComponent

Constructors:

1. JOptionPane() - used to create a JOptionPane with a text message
2. JOptionPane(String message) - used to create an Instance of JOptionPane with a message

Methods:

Public static void showMessageDialog(Component parentComponent, String message)

public static int showConfirmDialog(Component parentComponent, String message)

public static String showInputDialog(Component parentComponent, String message)

Constants:

[import javax.swing.*;]

public static final int YES_OPTION
public static final int NO_OPTION
public static final int CANCEL_OPTION

Example: Illustration of JOptionPane with Confirmation box

```
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample extends WindowAdapter
{
    JFrame f;
    OptionPaneExample()
    {
        f=new JFrame();
        f.addWindowListener(this);
        f.setSize(300,300);
        f.setLayout(null);
        f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        f.setVisible(true);
    }
    public void windowClosing(WindowEvent e)
    {
        int a=JOp
        if(a==JOp
        {   f.setDef
        }
    }
    public static voi
    {   new Option
    }
}
```

JTabbedPane

Used to switch
clicking on a t
class declaration

Public class J

Constructors:

~~Empty Cons~~

1. JTabbedPane
   default

2. JTabbedPa
   pane

```
        int a = JOptionPane.showConfirmDialog(f,"Are you sure?");
        if (a==JOptionPane.YES_OPTION)
        {   f.setDefaultCloseOperation(JFrame.EXIT_ON_clos
        }
    }
    public static void main(String args[])
    {   new OptionPaneExample();
    }
}
```

JTabbedPane

Used to switch between a group of components by clicking on a tab

class declaration:

Public class JTabbedPane extends JComponent

Constructors:

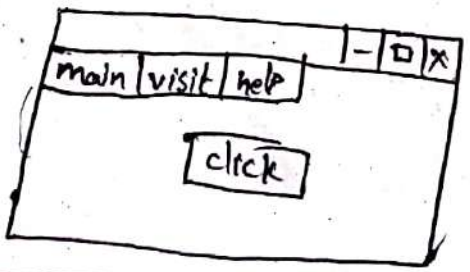
~~Empty Const~~

1. JTabbedPane() - Creates an empty TabbedPane with default Tab placement
2. JTabbedPane(int tab_Placement) - Creates a tabbed pane with a specified tab placement

Illustration of JTabbedPane

```
import javax.swing.*;
public class TabbedExample
{   JFrame f;
    TabbedPaneExample()
    {   f=new JFrame();
        JButton ta=new JButton("click");
        JPanel p1=new JPanel();
        ta.setBounds(50,50,80,30);
        p1.add(ta);
        JPanel p2 = new JPanel();
        JPanel p3 = new JPanel();
        JTabbedPane tp = new JTabbedPane();
        tp.setBounds(50, 50,200,200);
        tp.add("main", p1);
        tp.add("visit", p2);
        tp.add("Help", p3);
        f.add(tp);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
{   new TabbedExample();
}
}
```

O/P:



JLayeredPana:

It is used

It is used

a component

different layers

class declara

public clas

Constructor

JLayere

Methods:

1. Public

us

Specified

2. public

Use

Componen

JSplitPane

used

Compone

by using

(or) top

JS

class de

P

Construct

JSplit

arrange

JLayeredPana:

It is used to add a depth to swing container.

It is used to provide a third dimension for positioning a component and divide the depth range into several different layers

class declaration

public class JLayeredPane extends JComponent

Constructor

JLayeredPane() creates a new JLayeredPane

Methods:

1. Public int getLayer(Component c)
   used to return the Layer attribute of the specified componet.
2. public int getPosition(Component c)
   used to return the relative vertical position of the Component with in its layer

JSplitPane:

used to split or divide two components. The two components can be alligned either from left to right by using the constant JSplitPane.HORIZONTAL_SPLIT (or) top to bottom by using the constant

JSplitPane.VERTICAL_SPLIT

class declaration:

public class JSplitPane extends JComponent

Constructor:

JSplitPane() Creates a split pane configure to arrange the components side by side horizontally

JSplitPane(int Orientation, Components c1, Component

Constant field:

public static final int HORIZONTAL_SPLIT
public static final int VERTICAL_SPLIT

JTable:

JTable is used to display a data in a tabular format.

class declaration:

public class JTable extends JComponent

Constructors:

1. JTable() - creates a table with empty cells
2. JTable(String rows[][], string columns[])

creates a table with specified number of rows and specified number of columns.

JTree:

JTree is used to create a tree structure to display the data in a hierarchal fashion (like file explorer). All the nodes in a tree is created by using a special class known as DefaultMutableTreeNode

Constructors:

1. JTree() creates a empty tree structure
2. JTree(TreeNode root) creates a JTree with a specified tree node as root node.

DefaultMutableTreeNode:

used to create a node for a tree structure

Constructor

DefaultMutableTreeNode(String Node-name)

Creates a node with specified name)

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample
{   ~~f=new JFrame~~
    JFrame f;
    TreeExample()
    {   f=new JFrame();
        DefaultMutableTreeNode style = new DefaultMu
                                tableTreeNode("Style");
        DefaultMutableTreeNode color = new DefaultMutable
                                TreeNode("color");
        DefaultMutableTreeNode font = new DefaultMutable
                                TreeNode("Font");

        style.add(color);
        style.add(font);
        DefaultMutableTreeNode red = new DefaultMutable
                                TreeNode("red");
        DefaultMutableTreeNode blue = new DefaultMutable
                                TreeNode("blue");
        DefaultMutableTreeNode black = new DefaultMutable
                                TreeNode("black");
        DefaultMutableTreeNode green = new DefaultMutable
                                TreeNode("green");

        Color.add(red);
        Color.add(blue);
        color.add(black);
        color.add(green);
```

```
    JTree jt = new JTree(style);
    f.add(jt);
    f.setSize(200,200);
    f.setVisible(true);
  }
  public static void main(String args[])
  {   new TreeExample();
  }
}
```

JRadioButton:

Used to create radio buttons and we can choose only one out of several options.

class declaration:

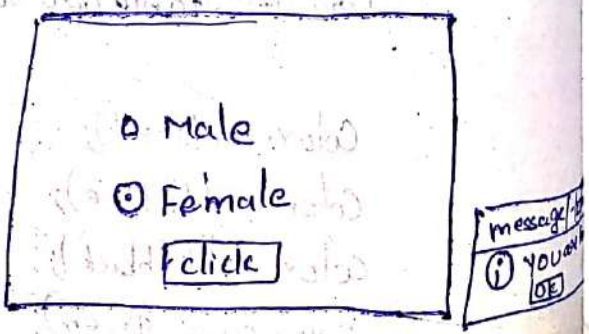
public class JRadioButton extends JComponent

Constructors:

JRadioButton() - creates a radio button without an option name.

JRadioButton(String s) - creates a RadioButton along with an option name

JRadioButton(String s, boolean status) - creates a RadioButton with an option and status is either true or false



17. JLable
1. JFrame
2. JButton
3. JTextField
4. JPasswordField
5. JComboBox
6. JCheckbox
7. JList
8. JTextArea
9. JRadioButton
10. JTable
11. JTree
12. JOptionPane
13. JTabbedPane
14. JScrollPane
15. JSplitPane
16. JLayeredPane
18. JPanel
19. JMenu
20. JMenuBar
21. JPopUpMenu
22. JApplet
23. JScrollbar