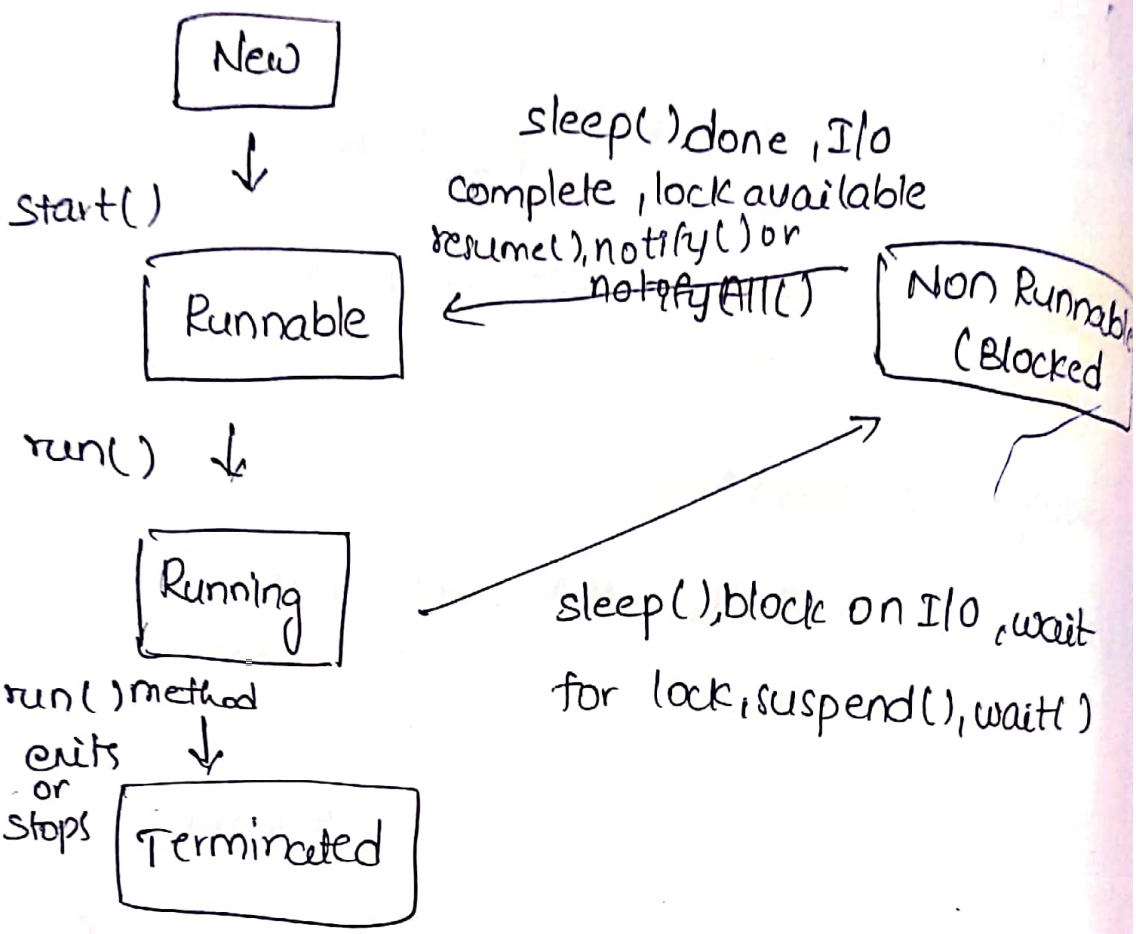
# Multi-threaded programing

Thread:
Life cycle.



Process: program under Execution. is called a process.

A process is simply a task that can be done by executing a program in our operating system.

Thread: It is a light wait process.

The thread is a smallest unit of work (or) task in a process.

Differences Between the Thread and process

* time
* context
* memory
*

Multi Tasking:

performing multiple tasks at a time (or) simultaneously is called Multitask.

It is of two ways:

1. Multi process
2. Multi threaded

Multi process:

performing multiple tasks at a time (or) simultaneously is called multiprocess

Multi Threaded:

Performing (or) executing multiple thread in a single process is called Multi thread

Advantages:

1. ~~cpu utilizations~~
2. efficient use of CPU utilization
3. Time saving
4. Throughput is high.

New: It creates a new object (or) instance for Thread class. then the Thread in new state.

Runnable: The Thread is ~~runnable (or)~~ ready to execute (or) Runnable and it wait for cpu time (or) process time

once the cpu is available (or) proces is available it go to the running state.

Running: it execuits the different Threads

∴ The compiler execuits the run() by calling start()

waiting / Blocked Non-runnable

when the Thread is in active that may be one of the following reason

Blocked - The Thread may be a locked.

waiting - ~~waiting for process~~ (or) sleep the Threads untill some amount of time

Terminate: The Thread is go to the ~~terminate state~~ ~~waiting state~~ terminate state.

1. when the thread is executed successfully
2. some unusual errors

abnormal terminates, timeouts etc...

Main Thread:

→ When a java program started up one thread being executed is called main Thread.

→ The main is implement because of two reason,

1. The main Thread is a thread from which other child threads will be ~~spawned~~ handled the all

2. Main thread must be a last thread to finish the execution.

Exam Eplem:

Write a java program to demonstrate the main Thread:

source code:

```
class currentThreadDemo
{
  public static void main (String a[])
  {
    Thread t= Thread currentThread();
    System.out.println ("current Thread:"+t);
    t.setName ("Apple");
    System.out.println ("Current thread:+t")
    try
    {
      for (i=1; i<=10; i++)
```

```
        {
            System.out.println(i);
            Thread.sleep(5000);
        }
    }
    catch (InterruptedException ie)
        {
            System.out.println(ie);
        }
    }
}
```

Syntax:

→ // public static Thread currentThread();

→ // public static void sleep(long millisec) throws InterruptedException;

→ Purposes of currentThread:-

It can be represented current thread present in our program.

→ purpose of sleep():-

~~What are the~~

We can create the Thread ~~class~~ in Java

~~class~~ 1. Runnable Interface
2. Thread class

## 1. Runnable Interface

By implementing the runnable interface the class can become a Thread. The runnable interface having only one extract method run.

Syntax:

```
public void run();
```

We must and should override the run method in your Thread class.

```
class NewThread implements Runnable
{
    Thread t;
    NewThread()
    {
        t = new Thead(this, "child thread");
        System.out.println("child thread: "+t);
        t.start(); //start the thread.
    }
    public void run()
    {
        try
        {
            for(int i=1; i<=15; i++)
            {
                System.out.println("Child Thread: "+i);
                Thread.sleep(1000);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("child interrupted");
        }
        System.out.println("exiting child thread");
    }
}
```

```
class ThreadDemo
{
    public static void main (String a[])
    {
        NewThread nt = new NewThread();
                    //create a new thread.
        try
        {
            for(char i='A'; i<='Z'; i++)
            {
                System.out.println("Main Thread:"+i);
                Thread.sleep(5000);
            }
        }
        catch(InterruptedException e)
        {
            System.out.println("Main Thread interupte
        }
        System.out.println("Main thread exiting");
    }
```

```
class childThread implement Runnable
{
   Thread t;
   public void new()
   {
      ChildThread ch=new childthread()
          t=new Thread();
         ch.Runnable();
   }
   public void Runnable()
   {
      t.start();
   }
```

```
public static void sleep(long milse)
                throws Interrepted Exception
```

```
Public void run()
{
   try
   {
      for (int i=1; i<=10; i++)
      {
         S.o.p(i);
      Thread.sleep(2000);
   }
}
```

```
catch (InterruptedException ie)
    {
        so.p(ie);
    }
}
class ThreadDemo
{ public static void main (String s[])
    {
        childThread ch=new childThread();
        ch.New();
    }
}
```

## Multiple Thread Creation:

```
class ChildThread implements Runnable
{
    Thread t;
    String threadname;
    public ChildThread (String name)
    {
        threadname=name;
        t=new Thread (this,name);
        System.out.println("current child thread
                                    "+t);
        t.start();
    }
    public void run()
    {
        try
        {
            for(int i=1; i<=10;i++)
            System.out.println("threadname+t":"+i);
            Thread.sleep(1000);
        }
    }
    catch (Interrupted Exception ie)
    {System.out.println("child was interrupted");
}
```

```
        System.out.println ("child thread completed.");
    }
}

class Newthread Demo
{
    public static void main(String a[])
    {
        childthread ch1=new childthread("Apple");
        childthread ch2=new childthread("Orange");
        childthread ch3=new childthread("mango");
    }
}
```

Syncronization:- A syncranisation is a process, when two or more threads need to access to a shared resource, They need to ensure that the resource will be used by only one thread at a time.

Syncranization is acheived by using the conceptof monitors.

-A monitor is an object that used for

The monitor can enecuit single thread at a time.

Syncranization means acheived in 2 ways

1. Syncranization method
2. Syncranization Block

1. Syncranization Method:- In Synchronization method an object ented in to a monitor then that method has been modified by synchronized keyword

While a thread in syncranization method that all other threads try to call it on same time have to wait

To make the method or a synchronized simply use the synchronized keyword.

Ex:

```
class Message
{
  public Synchronized void Resource(String
                                    threadname)
  {
    try
    {
      for(int i=1;i<=10;i++)
      {
        System.out.println(threadname+":"+i);
        Thread.sleep(1000);
      }
    }
    catch(InterruptedException ie)
    {
      System.out.println(threadname+"child was
                                    interrupted");
    }
    System.out.println(threadname+"child
                        Thread completed:");
  }
}
```

```
class childThread implements Runnable.
{ Thread t;
  String threadname;
  Message ms;
  public ChildThread (String name, Message m)
  {
    t = new Thread (this, name);
    System.out.println ("child thread completed:" +t);
    threadname = name;
    ms = m;
    t.start();
  }
  public void run()
  {
    ms.Resource (threadname);
  }
}
class SynchronizeDemo
{
  public static void main(String args[])
  {
    Message ms = new Message();
    ChildThread ch1 = new ChildThread("Apple", ms);
    ChildThread ch2 = new ChildThread("Orange", ms);
    ChildThread ch3 = new ChildThread("Mango", ms);
  }
}
```

~~Program: (Thread class~~

~~We can create the 1),~~

## Synchronization Thread:- (block)

In Synchronization Block (or) statement an object enter in to the monitor. The class blocked when other classes are try to access they have to wait

Syntax:

```
synchronize (Runnable objRef)
{
    body;
}
```

Thread class:-

It is used to create a new Thread.

The Thread class have different constructors some of them are as follows.

Thread(String type name)

Thread(Runnable object, String name)

The methods in Thread class:

1. getName
2. getPriority
3. isAlive
4. join → wait for a thread to terminate
5. run
6. sleep
7. start
8. setName
9. setPriority.

wait()

notify() notifyall();

This method tells to the calling Thread to give up the monitor and go to sleep untill some other threads enter in to the same monitor and calls the notify() (or)

notifyall()

notify() :-

wakeup a thread that call wait() on the same object

notifyall() :-

wakeup the all the thread ~~that call~~ call wait on the same object

## creating a Thread By using Thread class:

```
class NewThread extends Thread
 {
   NewThread()
    {
      super ("Child Thread");
      System.out.println("childthread:"+this);
      start(); //start the thread
    }
 }
 catch (InterruptedExceptione)
  {
   System.out.println("child interrupted.");
 }
  System.out.println("Exiting child Thread");
 }
}
```

```
class ThreadDemo
{
    public static void main(String args[])
    {
        NewThread nt = new NewThread();
                    //create a new thread
    }
}
```

InterThread Communication :- In interthread communication we are providing interaction between two (or) more threads.

This interthread communication is used in technique polling.

polling is a loop that is used to check some conditions repeatedly

once the condition is true the appropriate ~~while loop is~~ action is taken.

it wasting the cpu time. TO avoid the polling and providing the interthread communication via. wait(), notify() (or) notifyall(),

All the three methods must be placed in Synchronization method only

Ex: Demonstrate the inter thread communication by using

source code:

```
class showRoom
{
  int n;
  boolean flag=false;
  synchronized void put(int n)
  {
    while (flag==true)
    {
      try{
        wait();
      }
      catch(InterruptedException ie){
        System.out.println(ie);
      }
    }
    this.n=n;
    System.out.println("produced model No:"+n);
    flag=true;
    notify();
  }
  synchronized int get()
  {
    while(flag==false)
    {
```

```
        try {
            wait();
        } catch (InterruptedException ie)
        {
            System.out.println(ie);
        }
    }
    System.out.println("consumed modelNo:"+n);
    flag = false;
    notify();
    return n;
  }
}

class Producer implements Runnable
{
    ShowRoom m;
    Thread t;
    Producer(ShowRoom m)
    {
        this.m = m;
        t = new Thread(this, "producer");
        t.start();
    }
    public void run()
    {
        while (true)
        {
            m.get();
        }
```

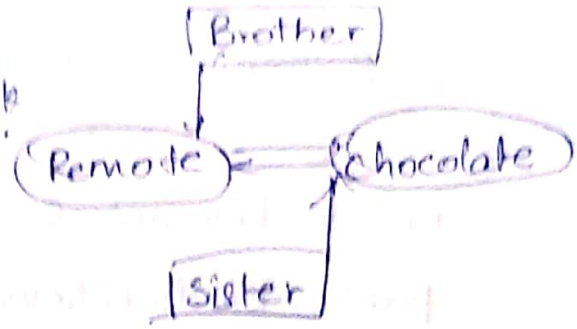
```
Class consumer implements Runnable
{
    Show Room m;
    Threat t;
    consumer(ShowRoom m)
    {
        this.m=m;
        t=new Thread(this, "consumer");
        t.start();
    }
    Public void run()
    {
        while(true)
        {
            m.get();
        }
    }
}

class InterThreadDemo
{
    public static void main(String a[])
    {
        ShowRoom m=new ShowRoom();
        new producer(m);
        new consumer(m);
        System.out.println("press Cntrl-C to stop
                            the execution");
    }
}
```

Deadlock-

manually creating a deadlock

Example Program - Situation




```
class Resource
{
  Synchronized void Remote(Resource s2)
  {
    System.out.println("Remote is used");
     try { Thread.sleep(1000);}
     catch(InterruptedException ie){
        System.out.println(ie);
     }
     s2.chocolate(this);
     System.out.println("Remote is end");
  }
  Synchronized void chocolate(Resource s1)
  {
    System.out.println("chocolate is used");
    try{Thread.sleep(1000);}
    catch(InterruptedException ie){
        system.out.println(ie);
    }
    s1.Remote(this);
  system.out.println("chocolate is end");
}}
```

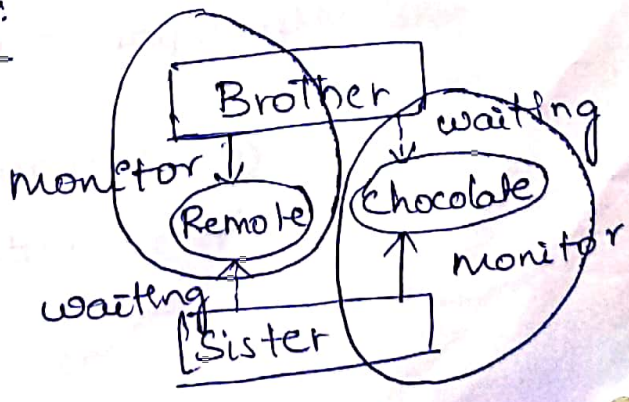
```
Class Brother extends Thread
{
  private Resource S1,S2;
  public Brother(Resource S1, Resource S2)
  {
     this.S1 =S1;
     this.S2 = S2;
  }
   public void run( ){
     S1.Remote(S2);
   }
}
Class Sister extends Thread
{
   private Resource S1, S2;
   public Sister (Resource S1, Resource S2){
     this.S1 =S1;
     this.S2 =S2;
   }
   public void run( ){
     S2. Chocolate (S1);
   }
}
```

```
public class DeadlockDemo5{
  public static void main (String[] args)
  {
    Resource s1 = new Resource();
    Resource s2 = new Resource();
    Brother t1 = new Brother(s1, s2);
    t1.start();
    Sister t2 = new Sister(s1, s2);
    t2.start();
  }
}
```

## Deadlock:

def: Deadlock is a situation where a Thread is waiting for Resource but the Resource is locked (or) allocated for another thread.

## diagram:



## Daemon Thread:-

It is a low priority Thread That execute the background Threads like Garbage collection, finilize method

Imp Topics: in multiThreading-

- Inter Thread
- Thread lifecycle
- Thread class(or) creation
- Synhronization.

# Input/Output

Sequence of data

Stream is an extraction that may produce (or) consume the information.

We have two types of streams:

1. Bytestream
2. Character stream

It read and write the data in form of bytes.

Bytestream having 2 abstract classes-

Inputstream / Outputstream. for reading and writting.

These 2 abstractclasses having different types of Bytestreams. some of them

1. BufferInputStream
2. BufferOutputStream
3. ByteArrayInputStream
4. FileInputStream
5. FileOutputstream.

BufferInputStream

Character Stream:

TO Read and write the character type of Information.

Character Stream having 2 abstract classes

1. reader
2. Writer.

it provides the different types of classes one of them.

BufferReader
BufferWriter
FileReader
FileWriter
ByteArrayReader
ByteArrayWriter.

InputStream: The methods in Inputstream for reading.

int available ( )

int read ( )

int read (byte buffer( ))

int read (byte buffer, int offset, int numbytes)

Output stream : The methods in output stream

```
void Write (byte n)
void Write (byte buffer[])
abstract
    void write(byte buffer[], int offset,
                int numbytes);
```

Reader : The method in Reader Stream are:

```
int  read ( )
String  readLine( )
int read (char buffer[])

int read (char buff[], int offset,
            int numchar)
```

Writer: The methods in Writer Stream are:

```
void write (char ch)
void Write (char buffer[])

abstract
    void Write(char buffer[], int off,
                int numchar)
```

~~The methods in InputStream~~

Example: Java program to demonstrate the read a single character for a buffer read a string.

```
import java.io.*;
class BRRead
{
    public static void main(String args[]) throws
                                                IOExcep
    {
        InputStreamReader is=new InputStream
                                    Reader
                                    (System.in);
        BufferedReader br=new BufferedReader(is
        System.out.println("Enter a Character");
        int ch=br.read();
        System.out.println("character "+(char)ch);
        System.out.println("Enter a String");
        String str=br.readLine();
        System.out.println("String "+str);
    }
}
```

# File Operations:

Java program to ~~demonstrate~~ copy the content from one file to another file by using byte string.

//reading

classes: FileInputStream fin=new FileInputStream (String filepath);

//writing

FileOutputStream fout=new FileOutputStream ( );

## Sourcecode:

```
import java.io.*;
class FileDemo
{
  public static void main(String args[]) throws IOException
  {
    int ch;
    FileInputStream fin=null;
    FileOutputstream fout=null;
    try
    {
      fin=new FileInputStream
      ("D:/java/alphabets.txt");
```

```
fout=new FileOutputStream("D:/java/newalpha.txt");

ch=fin.read();
while(ch!=-1)
{
    fout.write((char)ch);
    ch=fin.read();
}
System.out.println("copy completed");
}catch(Exception e)
{
System.out.println("File is not found");
}
finally
{
   fin.close();
   fout.close();
}
}
}
```

Write a java program

```
import java.io.*;
class fileDemo
{ public static void main (String args[ ] throws
                                    IOException
  {
    char ch;
    FileReader  fin = Null;
    FileWriter  fout = null;
    try
    {
     fin = new  FileReader ("D:/java/alphabets.
                                    txt");
    fout = new  FileWriter ("D:/java/new
                                    alphabets.txt");

    ch = fin.read( );
    while (ch != -1)
    {
      fout.writer(ch);
      ch = fin.read( );
    }
  System.out.println ("copy completed");
}
```

```
catch (Exception e)
{
  System.out.println("File is not found");
}
finally
{
      fin = close();
      fout = close();
  }
}
}
```