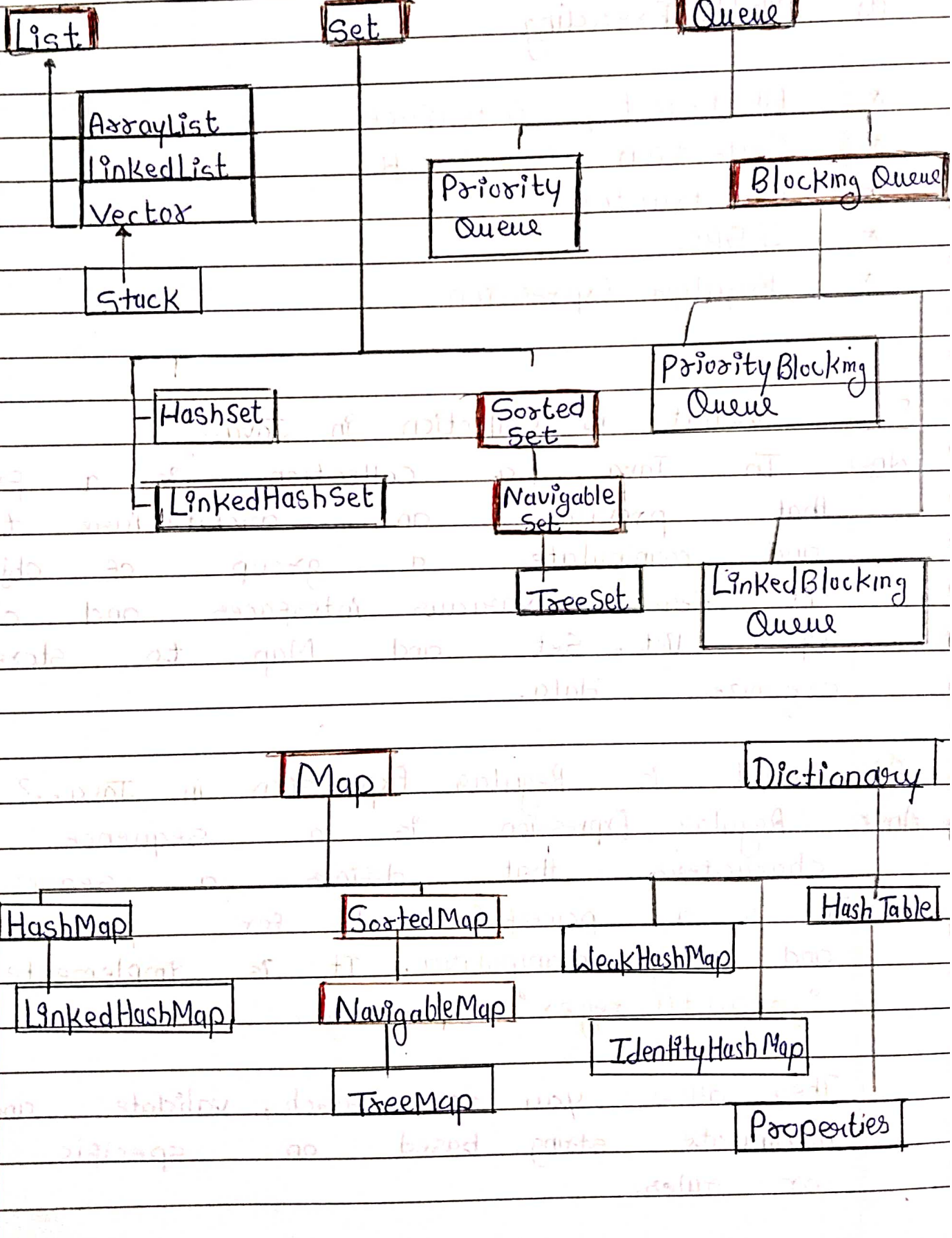
DATE
PAGE

* Black - class
* Red - Interface

Collection



:- Multi Tasking -:

The concept of multitasking is basically taken from human behaviour for example sometimes we do chatting on whatsApp while talking on phone also.

So, the same concept is use in computer for example we use VLC media player and downloading song simultaneously.

So, performing more than one task at same time is known as multitasking.

:- Types Of Multitasking -:

1) Process Based Multitasking
2) Thread Based MultiTasking

:- Process Based -:

It means the ability to execute multiple program (process) at the same time. Example - using VLC player while downlo-ading songs and creating a java program simultaneously.

:- Thread Based :-

It is a ability to execute the several parts of the program at the same time.

example :- When we type in MS-Word at the same time spelling - checker activates and modify if any mistake arise. Since typing and spelling checker and spelling modifier perform in a single program (process) itself is known as thread based multitasking (Multi-Threading)

Uses of Threads :-

(1) Server Side Program (Multiple client)
(2) Games, Animation (Simultaneously task)

:- Difference MP & MT :-

(1) Each process allocates separate memory area and Thread share same address

(2) Process is heavy weight
Thread is liteweight.

(3) Cost of communication between the process is high, Because switching from one

process to another requires some time for saving and loading registers etc.

Cost of Communication between the Thread is low.

Note :- Generally we think that the multiple process are executing simultaneously, but actually they execute one by one, Because the speed of CPU is very fast so that it appears as they are executed simultaneously.
To execute a process so many process scheduling algorithm we use such as :-

1. First - Come First Served (FCFS) scheduling
2. Shortest Job Next (SJN) scheduling
3. Priority Scheduling
4. Shortest Remaining Time
5. Round Rubin (RR) scheduling
6. Multi-Level Queue Scheduling

Q.1 What is Process scheduling ?

The act of determining which process in the ready state and should be moved to the running state is known as the process scheduling.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all program.

Q.2 Why thread is Lightweight ?

Thread is more efficient and uses the fewer resaurce (" thats why "lighter") and They also share the same address space.

Switching between thread is fast Comparatively than process.

The memary is shared that makes thread communication mare easier.

Creating Thread

1). Thread ek predefined class hai jo java.lang Package me rakhi hui hai.

2). Thread banane ke liye hame sabse phle Thread class ko inherit /extend karna hota hai.

3). Extend karne ke baad hame Thread class ki Run (run) method ko override karna hota hai.

4). Run method ko call karne ke liye hame Start() method ko call karna hota hai.

5). Thread start method ko call karne par hi create hoti hai.

```
Class A extends Thread
{
   public void run()
{
      System.out.println("ram");
}
}  Class Demo
{
   psvm (String ar[])
{  A a1 = new A();
}  a1.start();          // New Thread created
}
```

ram

```
Class A extends Thread
{
   public void run()
{
    int i;
  for(i=1; i<=10; i++)
{
    System.out.println("run="+i);
  }
}
   Class Demo
{
public static void main(String ar[])
{
   A a1 = new A();
 a1.start();          // 2nd Thread create hui yha se.
    int i;
   for(i=1; i<=10; i++)
{
    System.out.println("main="+i);
  }
 }
}
```

mixed output
run = 1
run = 2
main = 1
run = 3
20 values

yha mixed output isliye hai kyu ki dono Thread ek sath run hogi. Schedular kis thread ko priority dega ye fix nhi hai

```
class A exends Thread
{
   public void show()
{
    int i;
  for (i=1; i<=10; i++)
{
    System.out.println("run="+i);
   }
}
   class Demo
{
public static void main (String ar[])
{
   A a1. = new A();
    a1.show();
 int i;
    for (i=11; i<=20; i++)
{
    System.out.println("main="+i);
   }
 }
}
```

run = 1 to run = 10
main = 11 to main = 20

Thread class ki run method abstract nahi hai. Use override karna compulsory nahi hai.

Start method ko call karne se hi Thread create hoti hai.

```
class A extends Thread
{
  public void run()
  {
    int i;
    for (i=1; i<=10; i++)
    {
      S.o.p("run = "+i);
    }
  }
} class Demo
{
  p.s.v.m (String ar[])
  {
    A a1 = new A();
    A a2 = new A();
  a1.run();
  a2.run();
  }
```

Fixed Output
20 values
run = 1
to
run 10
run = 1
to
run = 10

Can we manually call run method..?
Yes..! we can manually call the run method but it does not create any Thread. It will treated as normal Instance method.

How Thread Created

Start method ke call hone par hi thread create hoti hai. and start method Tread class ki call hoti hai and Thread class ki start method sub class ki run method ko call karti hai.

Sirf Thread class ki start method call hone par hi thread create hoti hai.

```
Class A extends Thread
{
   String s1;
A (String s1)
{
     This. s1 = s1;
 }
public void run()
{  int i;
   for (i= 1; i<= 10; i++)
{
     S.o.p (s1+" "+i);
   }
}} Class Demo
{ psvm (String ar...)
{
     A a1= new A("A");
     A a2 = new A("B");
  a1. ~~start~~ A();
} a2. ~~start~~ ();
```

Fixed output 20 values

A = 1
to
A = 10
B = 1
to
B = 10

DATE
PAGE

— purana Code same —

```
public static void main (String ar[])
{
    A a1 = new A ("A");
    A a2 = new A ("B");

    a1.start();
    a2.start();
}
```

mixed output
20 values
A = 1
B = 1
B = 2
A = 2 ...

```
Class A extends Thread
{
    public void start()
    {
        int i;
        for (i = 1; i<=10; i++)
        {
            S.o.p ("start = "+i);
        }
    }}
Class Demo
{
    psvm (String ar[])
    {
        A a1 = new A();
        A a2 = new A();

    a1.start();
    a2.start();
```

Fixed Output
20 values
start = 1
to
start = 10
Start = 1
to
Start = 10

yha par thread create nahi hui hai. Thread sirf Thread (super) class ki start method call hone par hi create hoti hai.

Can we OverRide Start method..?

yes..!! we can override Start method, But then it will not create any thread. It will treated as normal Instance method.

```
Class A extends Thread
{
public void run ()
{
    S.o.p ("run method");
}
public void start ()
{
    S.o.p ("start method");
}
}
  Class Demo
{
   psvm (String ar[])
{
   A a1 = new A();
   A a2 = new A();
a1.start();
a2.start();
}
```

Start method
Start method

yha sirf class A ki start method call hogi.

```
class A extends Thread
{
  public void run()
  {
    S.o.p ("run method");
  }
public void start()
{
    super.start(); // Thread class ki call hogi
  S.o.p ("start method");
  }
}
   class Demo
{
    psvm (String ar[])
  {
  A a1 = new A();
  A a2 = new A();
  a1.start();
  a2.start();
  }
}
```

mixed output
4 values
run method
Start method
run method
Start method

yha start method phle A class ki call hui then super class ("Thread") ki start method call hone se thread create ho jayegi.

Isliye mixed output aayega.

```
Class A extends Thread
{
    public void run()
  {
      S.o.p ("run");
  }
}
   Class Demo
{
    psvm (String ar[])
{
     A a1 = new A();
  a1.start();
  a1.start();
  }
}
```

output → IllegalThreadStateException.

> Ham ek object ke corresponding start method ("Thread class ki") ek hi baar call kar sakte hai.

2nd Way Of creating Thread

Ham Thread Runnable interface ko implement kar ke bhi create kar sakte hai.

```
Class A implements Runnable
{
   public void run()
   {
        S.o.p ("ram");
   }
} class Demo
{
   psvm (String ar[])
   {
     A a1 = new A();
     a1. start();
   }
```

Error :- Can not Find Symbol start

* Runnable interface ke pass start() method nahi hoti hai isiliye error aa rha hai.

* Runnable interface ki run() method abstract hai means use override karna compul -sory hota hai.

* Runnable interface ke pass single method hi hai.

```
class A implements Runnable
{
    public void run()
{
    int i;
    for (i=1; i<=10; i++)
{
        S.o.p(i);
    }
}
} class Demo
{
    psvm(String ar[])
{
    A a1 = new A();
Threading t1 = new Thread(a1);
 t1.start();
}
```

output

1
2
to
10

Process of Creating Thread Through Interface

(1) Jis class ka thread banana hai use ham Runnable interface se implement krenge

(2) Run method ko override krenge.

(3) Uss class ka object create krke usse ham Thread class ke object ke constructur me pass krke thread class ke object se start method call krenge.

```
class A implements Runnable
{
 public void run ()
{
   int i;
for (i=1 ; i<=10 ; i++)
{
    S.o.p ("A = " +i);
 }
}
 } class Demo
{
    psvm (String ar[])
{
   A a1 = new A();
 Thread t1 = new Thread (a1);
t1.start();
int i;
 for (i=1; i<=10; i++)
{
    S.o.p ("B = "+i);
 }
}
 }
```

mixed output - 20 values
A = 1
A = 2
B = 1
A = 3
B = 2

```
Class A implements Runnable
{
   A()
{
      A a2 = new A();
      A a3 = new A();

  Thread t1 = new Thread (a2);
  Thread t2 = new Thread (a3);

t1.start();
t2.start();
}
public void run()
{
    for(int i=1; i<=10; i++)
  {
      S.o.p ("A = "+i);
    }
}
} Class Demo
{
    psvm (String ar[])
{
    A a1 = new A();
}
}
```

StackOverflowException

Ham Constructor ke andar object bana rhe hai jise first constructor call hoga & then fir object create hoga so first Constructor call hoga isse infinite constructor calli hoga jisse stack overflow ho jayegi

Thread Super

DATE
PAGE

Thread class ka bhi super/parent Runnable interface. Thread class runnable interface ko hi implement karti hai.

```
interface MyRunnable
{
    void run();
}
class MyThread implements MyRunnable
{
 public void run()
{
    S.o.p ("My thread Run method");
}
public void start()
{
    S.o.p ("start method");
run();
}
```

```
class Demo
{
    psvm (String ar[])
{
    A a1 = new A();
  a1.start();
}
```

```
class A extends MyThread
{
  public void run()
{
    S.o.p ("Class A run method");
}
}
```

```
class Demo
{ psvm (string ar[])
 {
     A a1 = new A();
     a1.start();
}
```

Start method
Class A run method

How Start Method Call Sub-class Method...?

Super/Thread class ki start method ke andar sirf "run()" ko call kia hua hai. Jab inheritence apply hoga to start method super class ki sub-class me inherit hokar aa jayegi and start method ke andar "run()" likha hua hai.

Ab start method sub-class me inherit hokar aane se run method sub-class ki call ho jayegi.

— Pichla example pura same —

```
class A
{public void run()
{
    super.run();
    S.o.p("class A run method");
}
}
```

Start method
Mythread run method
class A run method

Lambda Expression

Lambda expression was introduced in Java 1.8 version. It works only in Functional interface.

Function Interface :- A Interface who contains only one abstract method in it called functional Interface.

```
Interface Inter1
{
    void show();
}
class Demo
{
  psvm (String ar[])
{
  Inter1 i = () → S.o.p ("ram");
  i.show();
}
}
```

output → ram

Lambda expression me hame kuch bhi likhna nahi hota hai yha automatic sab man leta hai kuki ek hi method hoti hai and uska return type bhi likhne ki help hoti hai.

Lambda Expression

```
class Demo
{
    psvm (String ar[])
{
    Runnable i = () ->
    {
        int j;
    for (j=1; j<=10; j++)
    {
      S.o.p(i);
    }
};
    Thread t1 = new Thread (i);
    Thread t2 = new Thread (i);
t1.start();
t2.start();
}
```

20 values mixed output

Yha ham Runnable interface ki run method ko lambda expression ke through override kar rahe hai.

```
class Demo
{
    psvm(String ar[])
{
    Runnable i = () ->
{   int j;
    for (i=1; i<=10; i++)
{
```

```
        S.o.p (i);
    }
};
    Thread t1 = new Thread (i);
int j;
for (j=1; j<=10; j++)
{
        S.o.p (" main = " +j);
    }
t1.start ();
}
```

Fixed output

main = 1
to
main = 10
1
to
10

yha Second thread main thread ka code complete hone ke baad create ho rahi hai isliye Fixed autput aa rha hai.

```
class A
{
    A()
    {
        new Thread() { };
    }
}
class Demo
{ psvm (String ar[])
{
    A a = new A();
}
```

Blank Screen

yha ek anonymous inner class create hogi jo Thread class ko inherit kar legi. means thread class ki sari properties ko access or inherit kar legi.

isAlive()

isAlive() method : isAlive method ka return type boolean hai means agar thread bachi hui hai to ye "true" and thread complete hone ke baad "false" return karti hai.

```
class A
{
    String s1;
A (String s1)
{
    this.s1 = s1;
}
public void run()
{   int i;
 for (i=1; i<=10; i++)
{
    S.o.p (s1 + " " +i);
}
  try { Thread.sleep(100); } Catch (Exception e){}
 } }
class Demo
{  psvm (String ar[])
{
  A a1 = new A ("A");
  A a2 = new A ("B");
a1.start ();
a2. start ();
    S.o.p (a1.isAlive()); // true
    S.o.p (a2. isAlive()); // true
}
```

True → Thread sleep ho isliye ye phle
True
mixed output.

Join() types

1) join () → Thread complete execute hone tak join karke rakhta hai.

2) join (long) → Diye gaye mini sec tak hi join karke rakhta hai.

3) join (long, int) → Mini and Nano sec tak join karke rakhta hai.

Signature of Join ().

(1) public final void join () throws InterruptedException

(2) public final synchronized void join (long) throws Interrupted

(3) public final synchronized void join (long, int) throws InterruptedException.

20, Jan, 2024

DATE
PAGE

* What is Thread schedulaer?

Ans (i) One process contain many threads

(ii) Only one thread can execute at a time

(3) Thread schedulaer is a part af JVM

(4) Thread schedulaer decides that which thread shauld execute at a time and allocates processaer to the thread.

(5) Thread schedulaer consider varaus aspects to decide which thread to execute. The aspect are :-

(a) Thread Paiarity
(b) Time slicing
(c) Nature of Thread

(a) Thread schedulaer check faar the paiarity af the thread and allocate the processaer to the thread having high paiarity.

(b) If the thread having same paiarity then the thread schedulaer laak faar the time slicing that i.e law waiting af the thread ar any thread having law waiting time gets the processaer.

Thread Priority

1. All the thread contain same priority. The range of priority is 1 to 10 in range. where 1 is the lowest priority and 10 is considered as the highest priority.

2. By default the priority of Thread is 5

3. There are two methods to get and set the priority of thread.

(i) public final void getPriority() :- Gives the priority of Thread.

(ii) public final void setPriority() :- Sets the priority of the thread.

Thread Contains 3 static Constant.:-

* min Priority → 1
* max Priority → 10
* Norm Priority → 5

Some operating System do not support the thread properly. for example in Windows we need to get patch file from microsoft for these.

Note:- If we set the priority which is not in range of 1 to 10 it gives IllegalArgumentException.

Current Thread

```
class Demo
{
  public static void main (String ar[])
  {
    System.out.println (Thread.currentThread());
  }
```

Output → Thread [main, 5, main]

Name — priority — Group

currentThread(). method current thread ka name, priority, Group print kr dete hai

---

```
S.o.p (Thread.currentThread().getName());
S.o.p (Thread.currentThread().getPriority());
```

main
5

.getName() method Thread ka name return krti hai.
.getPriority() method current thread ki priority return kregi

---

```
S.o.p (Thread.currentThread().setName("aa"));
S.o.p (Thread.currentThread().setPriority(8));
  S.o.p (Thread.currentThread());
```

Thread [aa, 8, main]

.setName :- Used to change the name of Thread.
.setPriority :- Used to set or change the priority of thread.

```
class A extends Thread
{
   public void run()
   {
      S.o.p(Thread.currentThread());
   }
}  class Demo
   {
   psvm (String ar[])
   {
      S.o.p(Thread.currentThread());
   A a1 = new A();
   a1.start();
   }
```

```
Thread [main, 5, main]
Thread [Thred-0, 5, main]
```

Manually create ki gayi thread ki naming o se start hoti hai. & group main hi hota ha

```
   A a1 = new A();
   A a2 = new A();
   a1.start();
   a2.start();
```

```
Thread [Thread-0, 5, main]
Thread [thread-1, 5, main]
```

```
A a1 = new A();
a1.setName("aa");
a1.setPriority(8);
a1.start();
}
```

```
Thread [aa, 8, main]
```

```
class Demo
{
  public static void main (String ar[])
{
  System.out.println (Thread.MIN_PRIORITY);
  System.out.println (Thread.MAX_PRIORITY);
  System.out.println (Thread.NORM_PRIORITY);
 }
}
```

1
10
5

```
  class A extends Thread
{
  public void run()
 {
   S.o.p (Thread.currentThread());
  }
}
  class Demo
{
   psvm (String ar[])
{
    A a1 = new A();
    a1.setPriority (Thread.MAX_PRIORITY);
    a1. start();
  }
}
```

Thread [Thread-0, 10, main]

```
    A a1 = new A();
    a1.setPriority (16);
    a1.start();
}
```

IllegalArgumentException.

Note: Ham thread ki priority sirf 1 se 10 ki range me hi rakh sakte hai.

Best Way to Create Thread

(1) By extend thread class
or
By Implements Runnable interface.

Best Way → Implements Runnable Interface

Reason Why...?

Agar ham ek class extend kr lete hai to fir ham kisi or class ko inherit nahi kar sakte hai.

means agar hamne thread class ko extend kr lia hai to ham or kisi class ko inherit kar hi nahi sakte even hame need ho.

Interface ham kitne bhi implements kar sakte hai ek class ke andar. Isliye Runnable ko implement karke thread ko create karna best way hota hai.

Need for 2nd Type

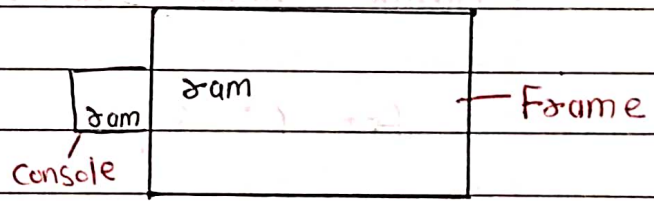
DATE
PAGE

```
class A extends Frame
{
}
```

Abb is case me ham thread kar hi nahi sakte hai. Isiliye hame 2nd type thread ko create karne ka provide kia hua hai.

```
import java.awt.*;
import java.awt.event.*;

class FDemo extends Frame implements Runnable
{
 public void paint(Graphics g)
{
   g.drawString("ram", 50, 50);
}
public void run()
{
   S.o.p("ram");
}
}} class Demo
{ psvm(String ar[])
{
  FDemo f = new FDemo();
  f.setVisible(true);
  f.setBounds(100,100,50,50);
f.addWindowListener(new WindowAdapter()
 {
public void WindowClosing(WindowEvent e)
{
   System.exit(0);
}
}; Thread t1 = new Thread(f);
   t1.start();
```



26, Jan, 2024

Synchronized

In java synchronized is a keyward that is used to control acess to shared rescurces among multiple threads. It ensures that only one thread can acess the synchronized block or method at a time, preventing data inconsistencies and conflicts. It helps in acheiving thread safety in concurrent prog -ramming.

```
Class A extends Thread
{
    static int x;
public ~~void~~ synchronized void run ()
{
    int i;
for (i=1; i<=50000; i++)
{
        x++;
}
}}
Class Demo
{
    psvm (String ar[]) throws Exception
{
    A a1 = new A();
    A a2 = new A();

a1.start();
a2. start();
```

```
a1.join();
a2.join();

S.o.p (a1.x);
S.o.p (a2.x);
}
```

random value
80442
80442

Yaha par variable x static hai means vo shared resource hai among multiple thre and x variable ki common memory alloca -te hogi kyuki vo static hai.

Ab jab alag - alag object kP thread ke liye run method ki alag memory allocate hogi kyuki dono object ke liye run method alag hi call hogi kyuki vo common nahi hai har ek object ke corresponding.

Dono thread x ko increment krenge same time par Lekin x static hai isliye dono thread ke liye common hoga.

Par dono thread ek sath increment krengi to x do baar increment na hokar ek hi baar hoga kyuki same time par increment ho raha hai.

Isliye random value aa rhi hai variable x ki.

```
class A implements Runnable
{
 public void run()
 {
   S.o.p (Thread.currentThread());
 }
} class Demo
{
   psvm (String ar[])
   {
      A a1 = new A();
      A a2 = new A();
  Thread t1 = new Thread(a1);
  Thread t2 = new Thread (a2);

t1.start();
t2.start();
}
```

```
Thread [Thread-0, 5, main]
Thread [Thread-1, 5, main]
```

Directly Setting Thread Name

~~upar vala same~~ -

```
A a1 = new A();
A a2 = new A();

Thread t1 = new Thread (a1, "A");
Thread t2 = new Thread (a2, "B");

t1.start();
t2.start();
```

```
Thread [A, 5, main]
Thread [B, 5, main]
```

Synchronized Method

DATE
PAGE

```
class A implements Runnable
{
  int x;
public synchronized void run()
{
   int i;
for (i=1; i<=10000; i++)
{
   x++;
 }
} }
class Demo
{
   psvm (String ar[]) throws Exception
 {
   A a1 = new A();
   A a2 = new A();
Thread t1 = new Thread (a1, "A");
Thread t2 = new Thread (a1, "B");
t1.start();
t2.start();
t1.join();
t2.join();
          S.o.p (a1.x);
          S.o.p (a2.x);
```

20,000
0
coz use hi nhi kia na initialize

Method ko synchronized banane ki wajah se us method ko ek time par ek hi thread access kr sakti or jab tak enter ki gyi thread method se bhr nahi aati tab tak sari thread wait krti hai. Synchronized se lock lag jata hai.

Synchronized Block

A Synchronized Block in java is a section of code that enclosed within the synchronized keyword. It allows us to control access shared resources by multiple threads. It is also known as object level lock.

```
class A implements Runnable
{
   int x;
 public void run ()
{
    int i;
  For (i= 1 ; i<= 10,000 ; i++)
  {
       Synchronized (this)
        {
              ++x;
        }
  } }
} class Demo
{
    psvm (String ar[])
   {
      A a1 = new A();
    Thread t1 = new Thread (a1);
    Thread t2 = new Thread (a1);
t1.start();
t2.start();
 S.o.p (a2.x);
}
```

20,000
Fixed value

Ek time par ek hi thread Synchronized block me enter hoti hai lock object se

	Synchronized Method	Synchronized Block
1.	Declared with the Synchronized keyward.	Enclosed within a Synchro-nized statement
2.	Acess the entire method.	Allows for mare fine grained control over synchronisation.
3.	Only one thread can execute Synchronized method at a time for an given object.	Requires specifying a lock object.
4.	Waiting time is high	Wating time is low.
5.	Performance is Low	Performance is high.
6.	Doesn't throw an NullPointer Exception	Can throw the Null Pointer Exception.

```
class A implements Runnable
{
    int x;
public void run()
{
    int i;
for (i=1; i<=10; i++)
{
S.o.p (Thread.currentThread().getName()+ " = " +i);
}
}}
```

Data Inconsistancy

Data inconistancy is a situation where there are multiple tables within a database that deals with the same data but may receive it from different inputs.

Data inconsistancy refers to a situation where the data in a system or database is not uniform or does not match up. - it can occur when there are discrepancies, conflict or error in a database leading to conflicting or incorrect information. It is like when you have different versions of documen with the conflicting information.

Jab ek hi object par mare than one thread hoti hai ya fir kaam kar rahi hoti hai par ham ek thread ka effect dusre thread par nahi show karna chahte hai toh iss condition me ham Synchronized ka use karte hai.

	Class Level Lock	Object Level Lock
1.	Applied to static method or block using Synchronized Keyword.	Applied to non-static method or block using synchronized keyward.
2.	Aquires a lock on the class itself, not on the individual object.	Aquires a lock on the individual object or specific instance.
3.	Only one thread can execute a static synchronized method or block across the all instances.	Each instance has its own lock allowing multiple thread to execute a synchronized method on different instance simultaneously
4.	Ensures synchronisation across all the instances of the class.	If one thread is executing synchronized method or block on specific object other thread will be blocked from executing synchronized method or block one the same instance until the lock is released.

Limitation of Obj. Level lock.

1. Can't applied to static method
2. DeadLock May occur.
3. Lack of flexibility

→ Pichla same Pura ←

Fixed output
4 thread

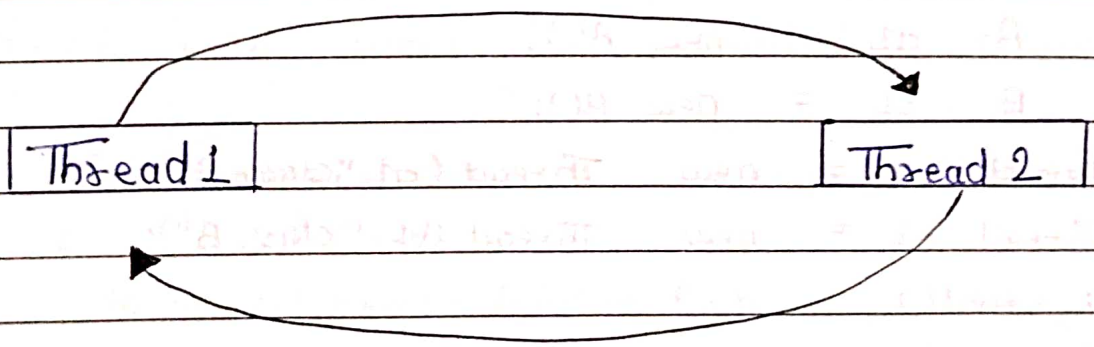
public synchronized void run()
{
→ pura same ←
}

Synchronized krne se ek hi thread enter hogi ek time par.

Dead Lock

Dead Lock in java occur when two or more threads or block forever, waiting for each other to release the resaurce they need. Its like traffic jam, where each car is waiting for the other to mo

DeadLock is a situation in multi-threaded programming where two or more threads are unable to proceed becaused each is waiting for a resaurce that the other thread hold.



wait()

DATE
PAGE

```
class A
{
   Synchronized  void  show () throws Exception
{
    S.o.p ("ram");
    S.o.p ("wait...");
try {S.o.
        wait(); }
catch (Exception e) { }
    S.o.p ("sita");
} }
class  Demo
{
  public static void main (String ar [])
{
    A  a  =  new A();
    a.show();
}
}
```

ram
wait...
(Infinite wait)

Wait() method sirf Synchronized method and synchronized block me hi work karti hai.

1) wait() :- For infinite waiting // wait ~~(100)~~

2) wait (long) :- For specific time waiting // wait (100);

3) wait (long, int) :- Also for specific time waiting //wait (10,200);

notify() method

```
class A
{
   synchronized void show1()
{
     S.o.p ("show1 call ");
      notify();
     S.o.p ("gita");
}
 synchronized void show2()
{
     S.o.p ("ram");
     S.o.p ("wait ... ");
try {
        wait(); } catch (Exception e) { }
    S.o.p ("sita");
}
}  class Demo
{
   psvm (String ar[])
{
    A a1 = new A();
    A a2 = new A();
   a1. show2();
   a1. show1();
}
```

```
ram
wait...
(Infinite waiting)
```

Ye show1() method tak Jaa hi nahi payega Kyuki flow show2() method par hi thread (main) waiting me ho Jayega isliye.

Wait()

wait() method is a part of Object class and it is used for inter-thread communication. It allows a thread to voluntarily give up the monitor (lock) on an object and enter a waiting state until another thread notifies it to wake up. The thread will remain in waiting state until :-

1) Another thread calls the notify() or notifyAll() on same object which was in waiting state.

2) A specific amount of time elapses and the thread automatically wake ups.

notify()

notify() method is a part of object class and it is used for inter-thread communication. It wake up a single thread that is waiting on the same object. When thread calls notify() method on a object, it notifies a waiting thread to wake up and continue its execution.

It can be only called within the synchronized method or synchronized block. to ensure the synchronization. otherwise it will throw the IllegalMonitorStateException