Multithreading

↳ Language Level concept.

1. Introduction
2. Ways to define a thread.
   - → By extending Thread class (C)
   - → By implementing Runnable (I)
3. Getting & Setting Name of Thread
4. Thread priorities
5. Method to prevent Thread Execution
   1. yield()
   2. Join()
   3. Sleep()
6. Synchronization
7. Inter thread Communication
8. Deadlock
9. Daemon thread
10. Multithreading Enhancement

→ Introduction.

Multitasking - Doing multiple task simultaneously.

→ Process based multitasking - Executing several task simultaneously where each task separate independent program.

→ only at OS level

→ Thread based multitasking - Separate independent task of the same program.

→ only at programmer level

→ Same address space will be share.

Thread - flow of execution, Independent job

→ Defining a thread. By 2 ways.

① Extending thread class

② By Implementing Runnable Interface

① By Extending thread class.

Defining a thread

```
class MyThread extends Thread
{
    Public void run()
    {
        Sop("child thread");
    }
}
```

→ Job of a Thread

```
class ThreadDemo
{
    PSVM(String[] args)
    {
        MyThread t = new MyThread();
        t.start();
    }
}
```

Main thread

→ Thread Instantiation

→ Starting of a Thread.

Main thread / child thread

Case-1 Thread Scheduler (part of JVM)

↳ If multiple threads are there, schedule the execution of threads in some order.

→ Algorithm varies from JVM to JVM

→ No possible order can be predict

Thread class → start method is responsible to t.start()

Case-2

t.start()

A separate ~~method~~ thread will created, and that thread is responsible to call child thread.

1. Register this Thread with Thread Scheduler
2. Perform all other mandatory activites
3. Invoke run(); only

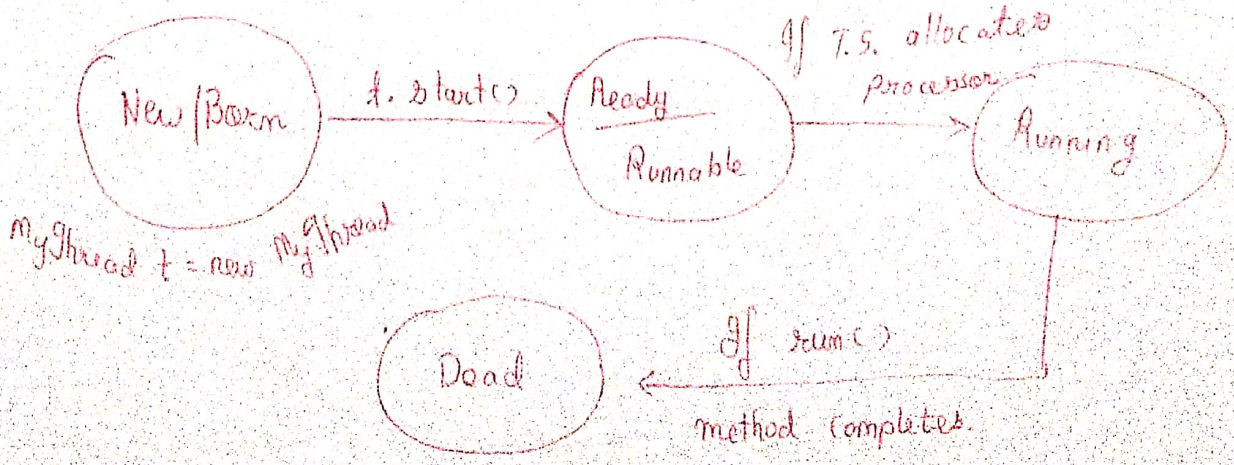
Not overloaded method.

Not to overide t.start

Normal execution call

t.run()

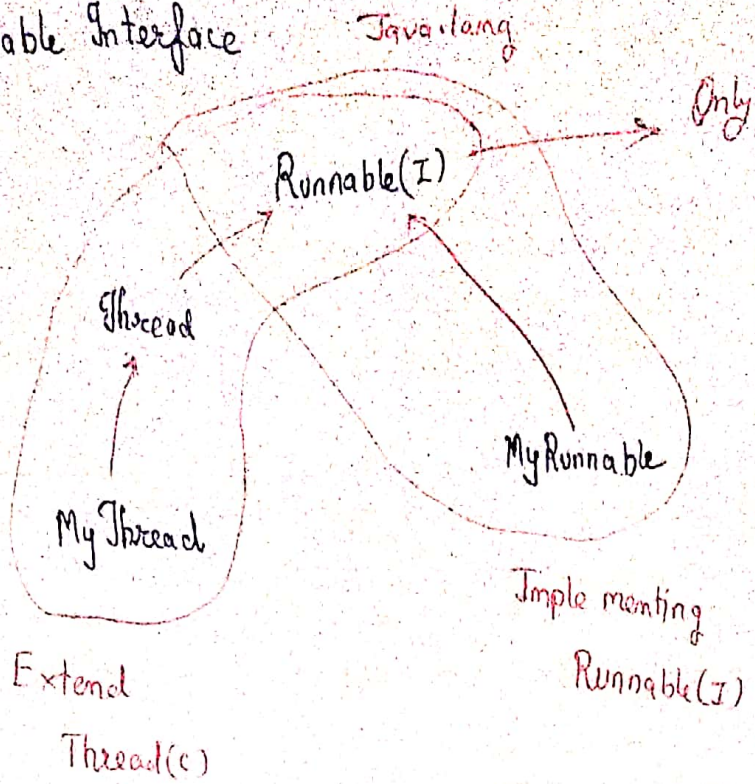
Normal flow of execution will take place.



If once thread t is start / then it is not possible to restart the same thread

↳ Cause : IllegalThreadStateException

② By Runnable Interface




```
class MyRunnable implements Runnable
{
    Public void run()
    {
        Sop("child Thread");
    }
}

class ThreadDemo
{
  PSV main(String[] args)
  {
     MyRunnable r = new MyRunnable();
     Thread t = new Thread(r);
     t.start();
  }
}
```

Defining a thread

Job of a thread.

Target Runnable

```
MyRunnable r = new MyRunnable();
Thread t1 = new Thread();
Thread t2 = new Thread(r);
```

Case 1 `t1.start();` — thread will create thread class empty run() will call.

Case 2 `t1.run();` — No thread will create

Case 3 `t2.start();` — thread will create with target(r) runnable class.

Case 4 `t2.run()` — No thread, normal main thread execution

Case 5 `r.start();` Cannot find Symbol : method start location: class MyRunnable

Case 6 `r.run();` Normal execution

Thread class Constructor

① `Thread t = new Thread();`

②
```
Thread(Runnable r);
Thread(String name);
Thread(Runnable r, String name);
Thread(ThreadGroup g, String name);
Thread(ThreadGroup g, Runnable r);
Thread(ThreadGroup g, Runnable r, String name);
Thread(ThreadGroup g, Runnable r, String name,
       StackSize);
```

→ System.out.println(Thread.currentThread().getName());

→ MyThread t = new MyThread();
t.getName();

→ Thread.currentThread().setName("Vishal");

Thread - priority

Range - [1 - 10]

Min priority

Max priority.

Thread.MIN_priority = 1

Thread.NORM_priority = 5

Thread.MAX_priority = 10

→ Thread Scheduler → It gives priority while allocating resource

→ Highest priority will get chance first

→ If two threads having same priority we can't expect execution order. Depends on thread scheduler.

(1) Public final int getPriority()

(2) public final void setPriority(int p) [1 - 10]

↳ IllegalArgumentException

→ Default-priority ⟶ ———

↳ Only for main thread is 5, But for all remaining thread default priority will be inherited.

That is whatever priority parent has thread has the same priority will be there for child thread.

Prevent thread Execution

① yield()
② join()
③ sleep()

→ Yield method

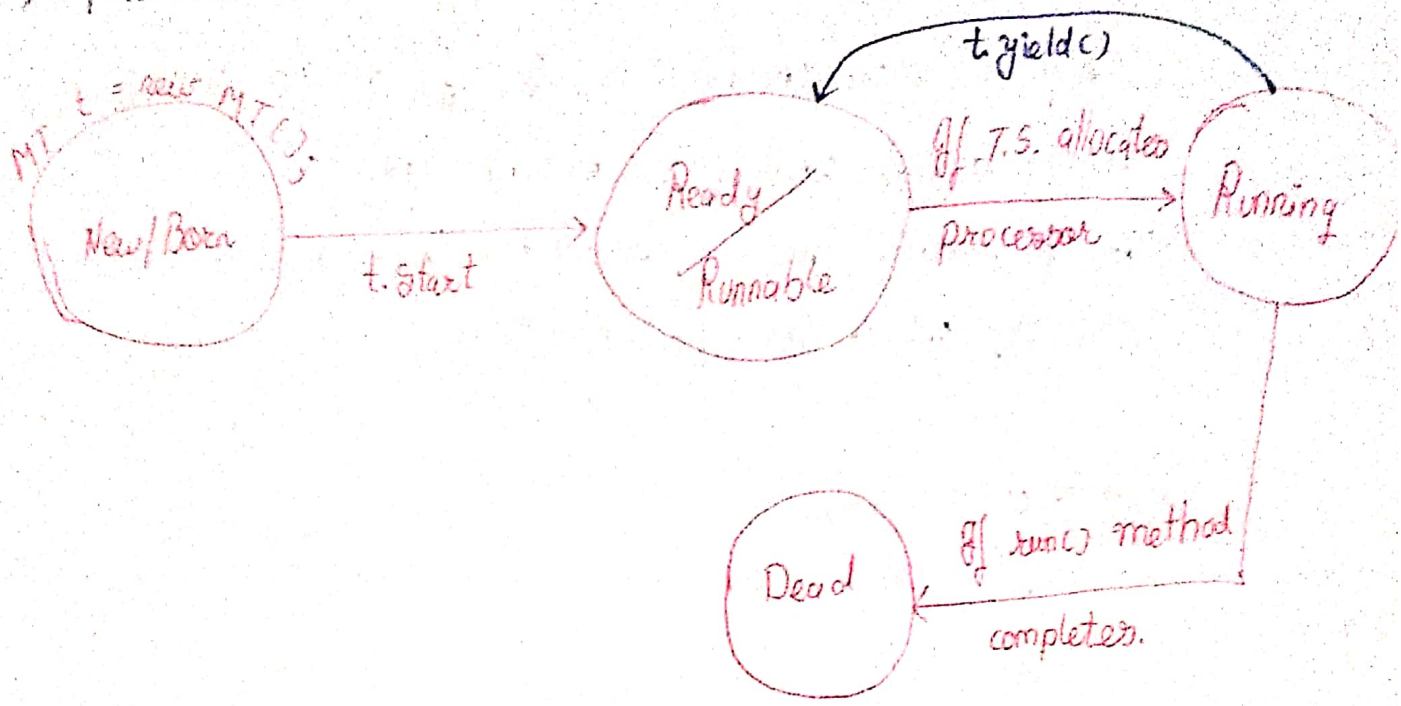
↳ To give the chance to waiting thread with same priority

→ To pause current executing thread, to give the chance for waiting thread of same priority.
→ If there is not waiting thread or all waiting thread have low priority.
→ Then same thread continues execution.

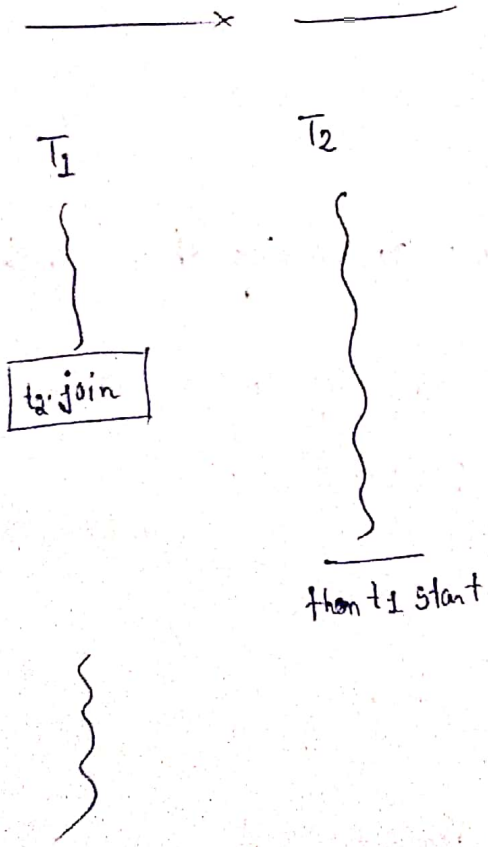
→ Multiple thread → waiting → Same priority
→ Execution order depend on Thread Scheduler.

→ When a thread yield / leave a processor, we can't expect when we get next time with processor.

→ public static native void yield(); ← Prototype.



Join()



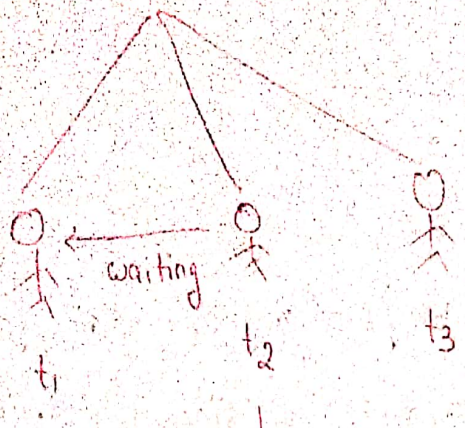
→ If t_1 thread want to wait for thread t_2 execution complete.

→ The waiting thread t_1 has to call t_2.join

→ Untill completes of t_2 thread, thread t_1 has to waiting state.

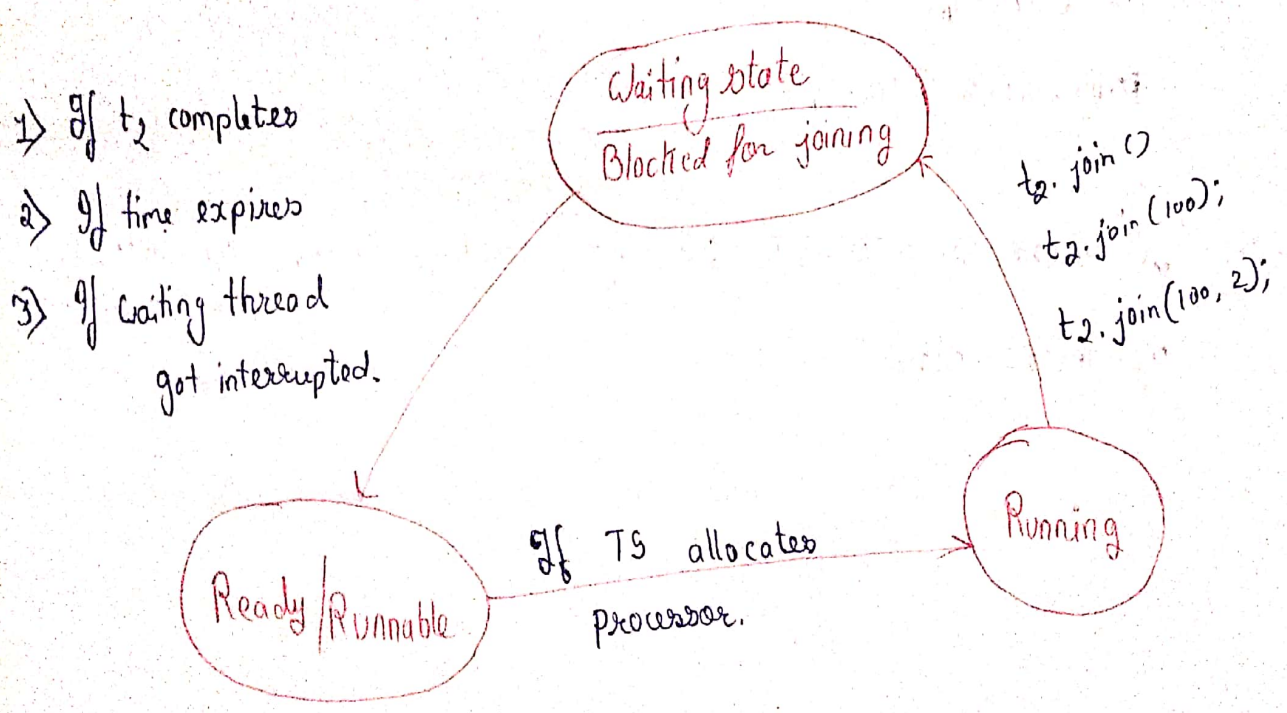
Method
1. public final void join();
2. public final void join(long ms);
3. public final void join(long ms, int ns)

Join method :-



t_1.join → If some other thread t_3 interrupted t_2.

Then InterruptedException throws.



1) If t_2 completes
2) If time expires
3) If waiting thread got interrupted.

If a thread call itself join - Deadlock

↳ Thread.currentThread().join();

⟹ Sleep();

If a thread don't want to perform any operations for a particular amount of time, then we should go for sleep() method.

→ public static native void sleep (long ms)
→ Public static void sleep (long ms, int ns);

} throws InterruptedException

⟹ Thread.interrupt()

public void interrupt() → Sleeping thread
→ Waiting thread

→ A thread can interrupt sleeping, waiting thread by interrupt of thread class.

Note → A thread can interrupt only when thread is sleeping or waiting stage.

Property	yield()	join()	sleep()
Purpose	-	-	-
overloaded	✗	✓	✓
final	✗	✓	✗
throw IE	✗	✓	✓
native	✓	✗	sleep(long ms) → native sleep(long ms, int ns) → non native
static	✓	✗	✓

Synchronization

→ Synchronized is an access modifier applicable for method and block.

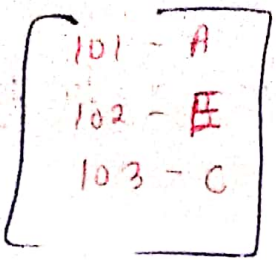
Note: If multiple user will try to use same java object then there is a chance of data-inconsistency problem. To overcome this problem Synchronized will require.

→ We can resolve data-inconsitency problem

→ Cons - It increase waiting time of thread and creates performance Problem, if there is no specific requirement then it is not recommended to use Synchronized keyword.

Ex-1

```
psvm( ){
    CHM m = new CHM();
    m.put(101, 'A');
    m.put(102, 'B');
    m.putIfAbsent(103, C);
    m.putIfAbsent(101, D);
    m.remove(101, D);
    m.replace(102, B, E);
    Sout(m)
```



Ex-2

```
class MyThread Extends threads{
    run()
    {
      Thread.sleep(2000);
    }
    catch ( ){ }
    Sout(child updating map);
    m.put(103, c);
}
```

Concurrent HashMap

1. Underlying Data structure is Hash table.
2. C-HM allows concurrences read & thread-safe operations.
3. To perform operations (read) thread can't require any lock.
   For update operations thread require lock.
   But it is lock in particular part of Map (Segment lock / Bucket level lock)
4. Concurrent update achieved by internally dividing map into smaller portions, which is defined by [Concurrency level]
5. The default concurrency level is 16
6. CHM allows any no. of read operations but 16 update operations at a time by default
7. Null is not allowed for both keys & values.
8. While one thread iterating, the other thread can perform update operation & never throws ConcurrentModification Exception

Constructor

```
CHM m = new CHM();
        (int initialCapacity)
        (int initialCapacity, float fillRatio)
        (initalCapacity, fillRatio, concurrencyLevel)

CHM m = new CHM(m);
```

3) boolean replace (object key, object oldvalue, object newvalue)

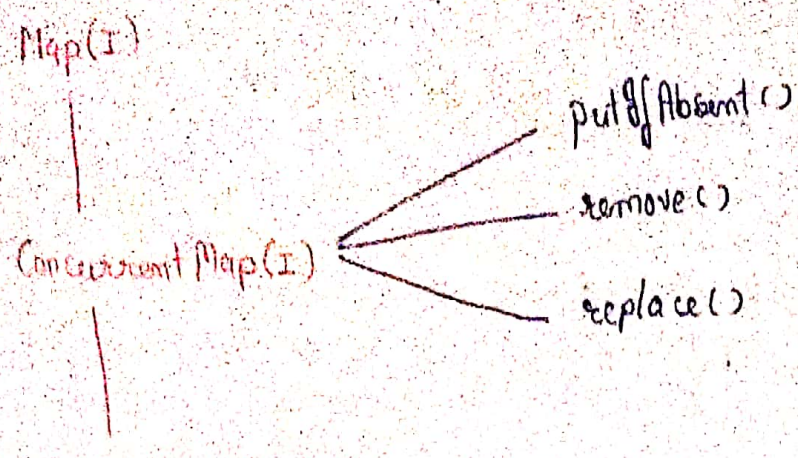
iff key & old value matched
then only new value will be replaced.

```
m.put(101, 'D');
m.replace(101, 'D', 'R');
Sout(m) = 101 = R
```

Concurrent Hashmap (C)

HashMap	Hashtable	ConcurrentHashMap
↓	only 1 thread	16 thread
Data Inconsistency		Thread safe
Not thread safe	Thread safe	
	Lock on total Hashtable	For read - no lock For write - lock on particular bucket So total 16 thread can work simultaneously Concurrency level = 16

Concurrent Collections ← Java.util.concurrent



1) object putIfAbsent (key, value);

```
m.put(101,'D');
m.put(101,'S');
Sout(m);
```

101 = S

```
m.put(101,'D');
m.putIfAbsent(101,'S');
Sout(m);
```

101 = D

If key is available then don't do anything.

2) boolean remove (Object key, Object value);

```
m.put(101,'D');
m.remove(101);
Sout(m) = 101, D
```

```
m.put(101,'D');
m.remove(101,'S');
Sout(m) = 101, D
```

If key, value is matched then only it will remove.

CopyOnWriteArrayList(C)

→ Thread Safe version of ArrayList