Multitasking

doing more than multiple task simultanesly.

To understand activity of a student sitting in a class room

--listen the notes

--listen the class

--while listen he may take running notes

--while taking running notes check mobile, sleeping

--if anybody like principle enters into class he runs his head from 60 to 90 so observing surroundings.

these many things a student is doing simultaneously.

Types of Multitasking

**1) Process based Multitasking**

Executing several tasks simultanesly where each task is a separate independent process.

**Example:**

-----Typing java program in the editor.

-----Listening audio music from same system.

-----Same time download a file from internet.

All these are independent process so it is a Process based Multitasking.

These type of tasking we come under OS level concept, not at programmer level.

Like if some client come ask ask to develop any app , he ask in which Lang u develop it, I says I'm comfortable with java,

If he ask features of java and if i told all above features like listen music in background.

he strikes his mind meaning what is the need to listen music , it is not a part of my aap.

so we cannot highlight these advantages on programmatic level.  
best suitable programming at programmatic level Thread based multi tasking.

**2) Thread based Multitasking**

Suppose there is program that contains 10000 lines of code.

and executing these 10000 lines of code it will take 10 hrs

now as i analyze the code, and identified that first 5k lines ,second 5k lines there is no dependency.

if there is no any dependency why second 5k lines wait until completing first 5k lines.

waste of time, performance is going to be down.

so instead of executing one by one we can execute these two parts simultaneously

these type of tasking is called thread based multi tasking.

Executing several independent parts simultaneously, where each part is the part of same program.

each independent part is called a thread.

ifi execute each tread one by one is called single thread program.

ifi execute each tread simultaneously is called multi thread program.

5K

5K

Fisrt 5k line code

Total 10000 line of code

of a program

Second 5k line code

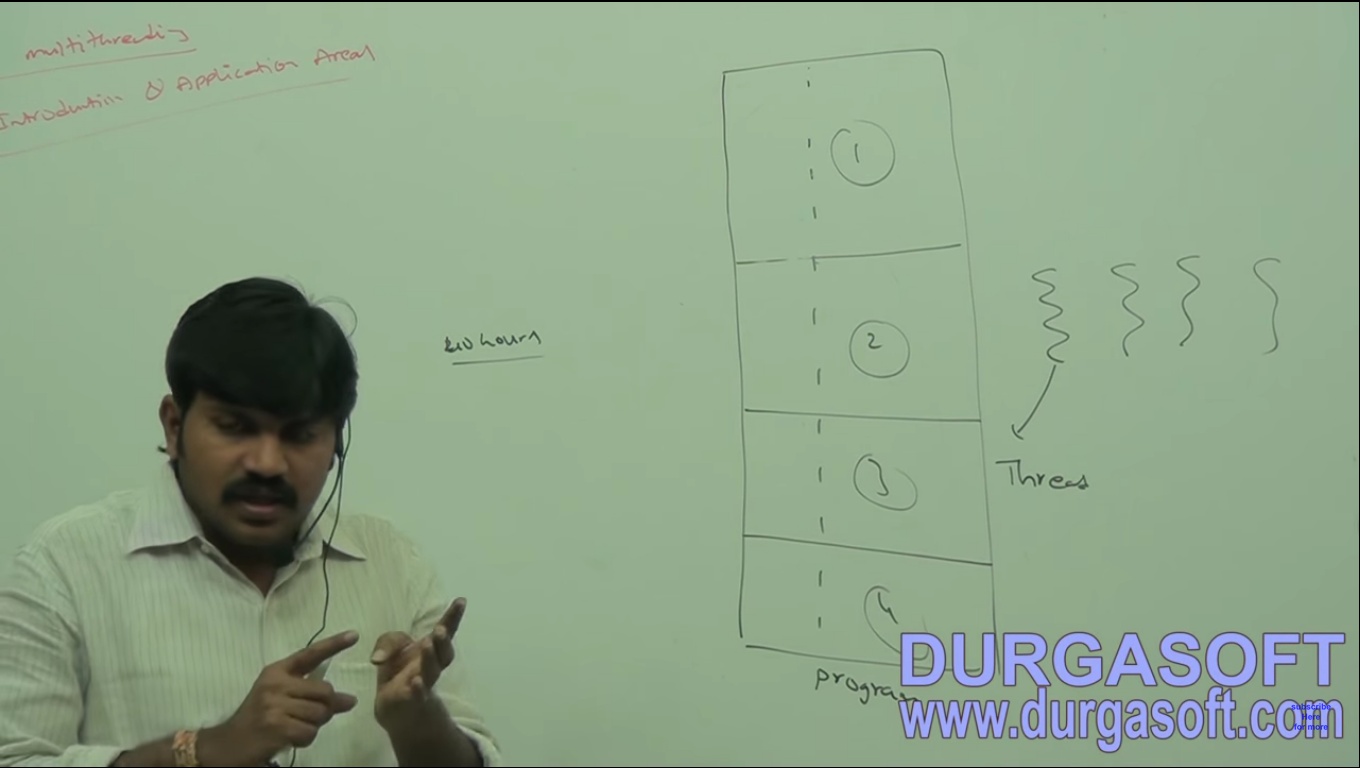
two independent parts of a program.

**A thread is a flow of execution or independent part of a program.**

if there is a program that contains 1000 line of code but there are parts in program, and these parts are independent of each other.

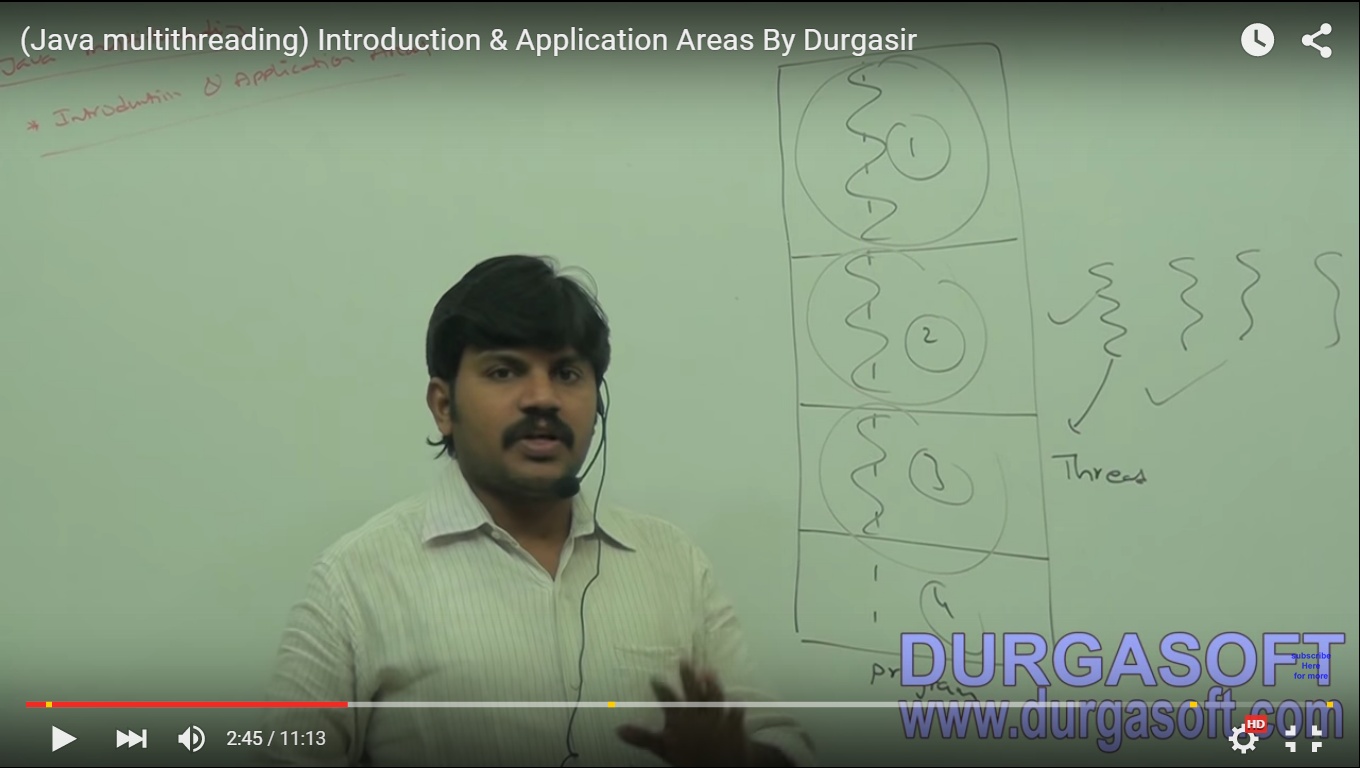
last part have to wait for 1 above parts,

meaning low performance.



if only main function then only one thread.

for a thread there is job executed in its block same for all

JObs of thread

For T1

For T2

For T3

use of multithreading is to improve system performance, by executing all threads simultaneously.

so always makes processor busy with the some activity.[reduce ideal time of cpu]

**Where we use it:**

---Video games

---Animation Movie

---Multi Media Graphics

---Server' application[best example]

Person from india request

Person from UK request

Person from US request

So three requests are processed

simultaneously.

example:

ifi ask my system to search a particular type of file

my com have c: d: E: F: drives

it will check inmy all drives one by one

very large time required so performance is poor

ifi create four threads which search for files in all drives simultanesly

performance is improved.

multithreading in java is easiest because

90% work is done by API

10% work programmer have to do

I just use methods given by java API

=================

**How to Define A Thread**

1) By Extending thread class

2) By implementing runable interface [best way]

**Example:**

class Multithread extends Thread{

public void run()

{ //Child thread

for(inti=0;i<10;i++)

System.out.println("i= "+(i+1));

}

}

classMDemo{

public static void main(String args[]){//Main thread

Multithread t=new Multithread();

t.start();

for(inti=0;i<10;i++)

System.out.println("J= "+(i+1));

} } main thread

Main thread child thread

**Thread Scheduler:**

Responsible to schedule the threads, it is a part of JVM

If multiple threads are there, but only one processor, in which order they will be executed is decided by thread scheduler.

Behavior of thread scheduler depends on JVM, it changes from JVM to JVM.

some thread May follow First come first serve,

some thread May follow shortest job first,

some thread May follow round robinal curriculum.

No need to worry in which order they will execute no need to worry.

**Life cycle of a Thread (Thread States)**

A thread can be in one of the five states. According to sun, there is only 4 states in thread life cycle in java new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

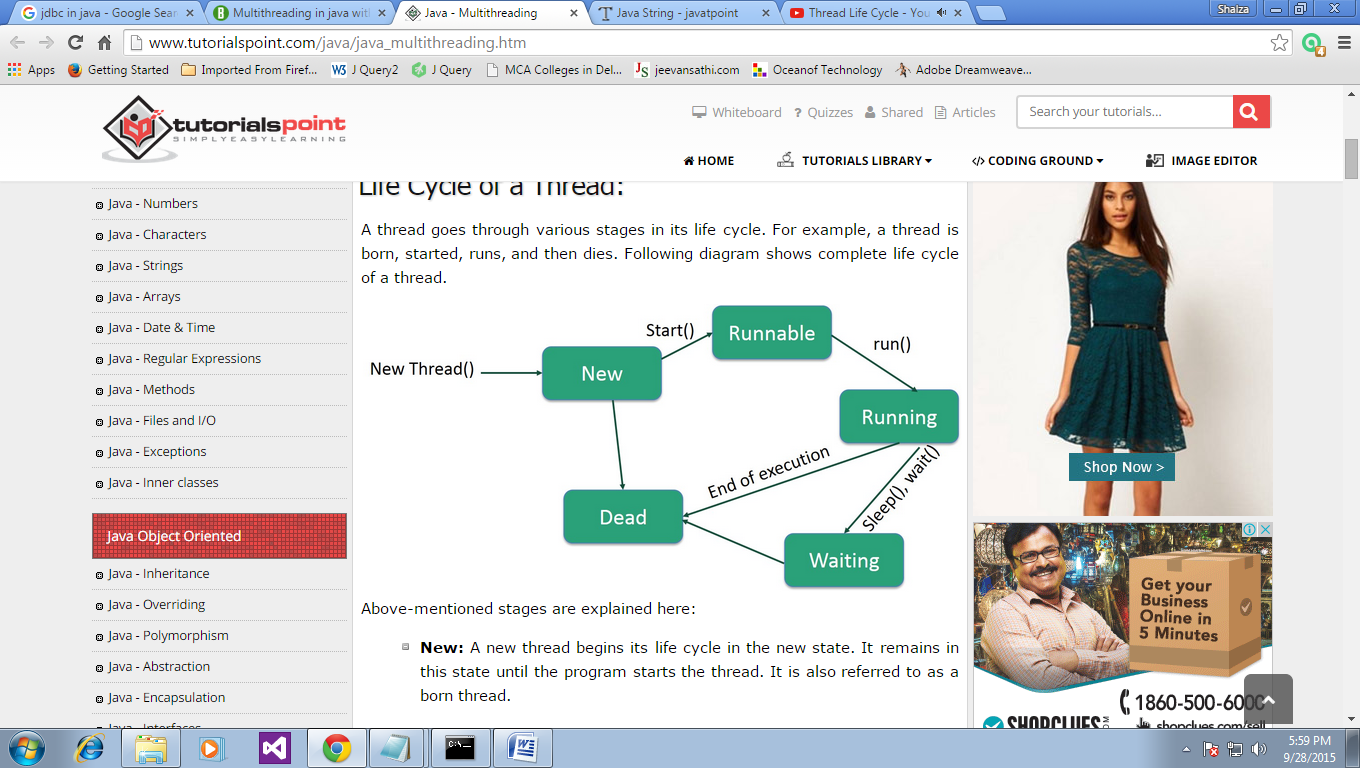
1) New

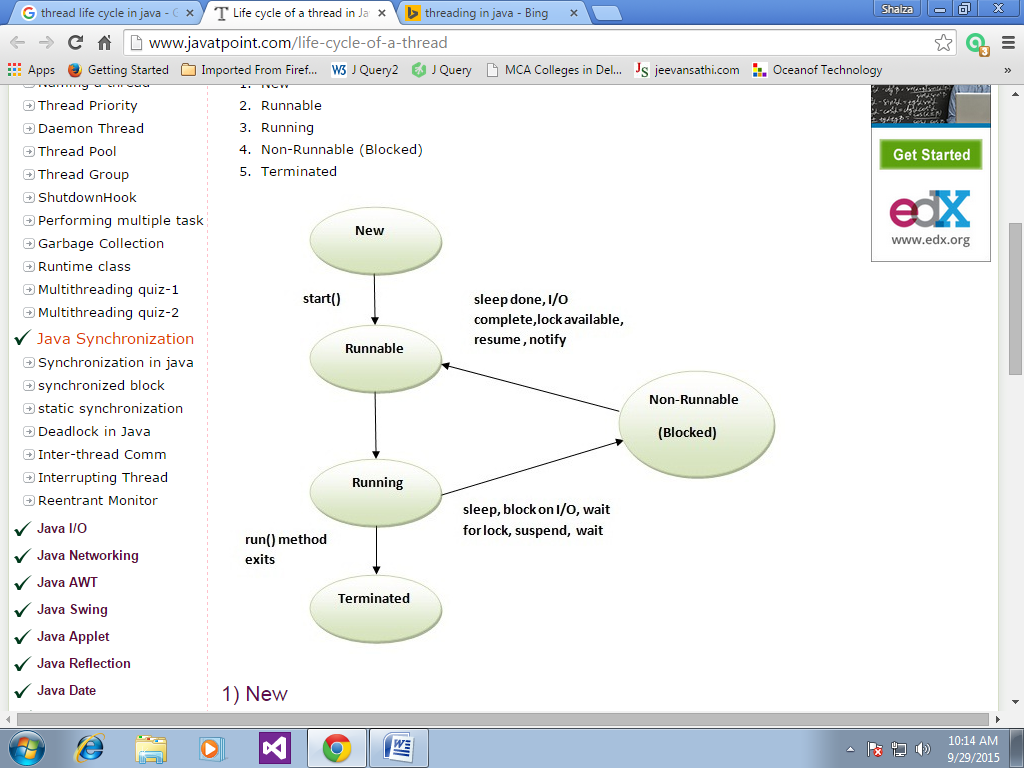
2) Runnable

3) Running

4) Non-Runnable (Blocked)

5) Terminated





**1) New**

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

**2) Runnable**

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

**3) Running**

The thread is in running state if the thread scheduler has selected it.

**4) Non-Runnable (Blocked)**

This is the state when the thread is still alive, but is currently not eligible to run.

**5) Terminated**

A thread is in terminated or dead state when its run() method exits.

**How to create thread**

There are two ways to create a thread:

1) By extending Thread class

2) By implementing Runnable interface.

**Thread class:**

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

**Commonly used Constructors of Thread class:**

Thread()

Thread(String name)

Thread(Runnable r)

Thread(Runnable r,String name)

**Commonly used methods of Thread class:**

public void run(): is used to perform action for a thread.

public void start(): starts the execution of the thread.JVM calls the run() method on the thread.

public void sleep(long miliseconds): Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

public void join(): waits for a thread to die.

public void join(long miliseconds): waits for a thread to die for the specified miliseconds.

publicintgetPriority(): returns the priority of the thread.

publicintsetPriority(int priority): changes the priority of the thread.

public String getName(): returns the name of the thread.

public void setName(String name): changes the name of the thread.

public Thread currentThread(): returns the reference of currently executing thread.

publicintgetId(): returns the id of the thread.

publicThread.StategetState(): returns the state of the thread.

publicbooleanisAlive(): tests if the thread is alive.

public void yield(): causes the currently executing thread object to temporarily pause and allow other threads to execute.

public void suspend(): is used to suspend the thread(deprecated).

public void resume(): is used to resume the suspended thread(deprecated).

public void stop(): is used to stop the thread(deprecated).

publicbooleanisDaemon(): tests if the thread is a daemon thread.

public void setDaemon(boolean b): marks the thread as daemon or user thread.

public void interrupt(): interrupts the thread.

publicbooleanisInterrupted(): tests if the thread has been interrupted.

public static boolean interrupted(): tests if the current thread has been interrupted.

==================================================

**Runnable interface:**

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

public void run(): is used to perform action for a thread.

**Starting a thread:**

start() method of Thread class is used to start a newly created thread. It performs following tasks:

--A new thread starts(with new callstack).

--The thread moves from New state to the Runnable state.

--When the thread gets a chance to execute, its target run() method will run.

**1)By extending Thread class:**

class Multi extends Thread{

public void run(){

System.out.println("thread ...");

}

public static void main(String args[]){

Multi t1=new Multi();

t1.start();

}

}

**Output**:thread is running...

**Who makes your class object as thread object?**

Thread class constructor allocates a new thread object.When you create object of Multi class,your class constructor is invoked(provided by Compiler) fromwhere Thread class constructor is invoked(by super() as first statement).So your Multi class object is thread object now.

**2)By implementing the Runnable interface:**

class Multi3 implements Runnable{

public void run(){

System.out.println("thread is running...");

}

public static void main(String args[]){

Multi3 m1=new Multi3();

Thread t1 =new Thread(m1);

t1.start();

} }

**Output**:thread is running...

If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitely create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

**Sleep method in java**

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

**Syntax of sleep() method in java**

The Thread class provides two methods for sleeping a thread:

public static void sleep(long miliseconds)throws InterruptedException

public static void sleep(long miliseconds, intnanos)throws InterruptedException

**Example of sleep method in java**

class TestSleepMethod1 extends Thread{

public void run(){

for(inti=1;i<5;i++){

try

{

Thread.sleep(500);

}

catch(InterruptedException e)

{System.out.println(e);

}

System.out.println(i);

}

}

public static void main(String args[]){

TestSleepMethod1 t1=new TestSleepMethod1();

TestSleepMethod1 t2=new TestSleepMethod1();

t1.start();

t2.start();

} }

**Output:**

1

1

2

2

3

3

4

4

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time,the thread shedular picks up another thread and so on.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Can we start a thread twice**

No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.

Let's understand it by the example given below:

public class TestThreadTwice1 extends Thread{

public void run()

{

System.out.println("running...");

}

public static void main(String args[]){

TestThreadTwice1 t1=new TestThreadTwice1();

t1.start();

t1.start();

} }

running

Exception in thread "main" java.lang.IllegalThreadStateException

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**What if we call run() method directly instead start() method?**

Each thread starts in a separate call stack.

Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

class TestCallRun1 extends Thread{

public void run(){

System.out.println("running...");

}

public static void main(String args[]){

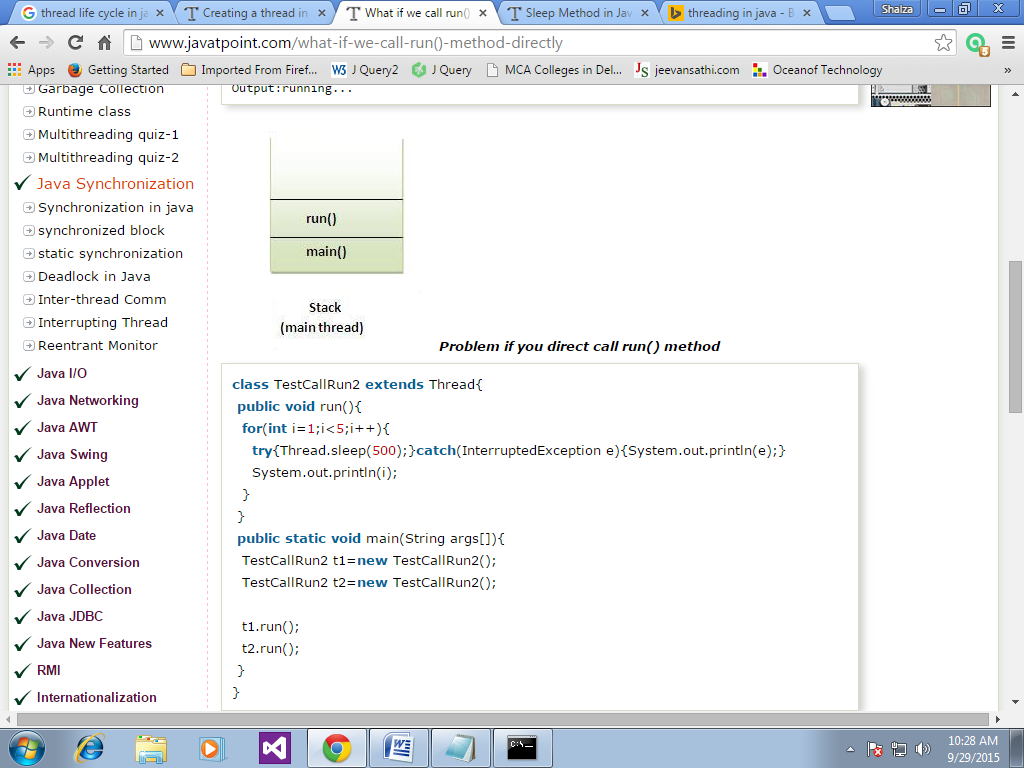
TestCallRun1 t1=new TestCallRun1();

t1.run();

//fine, but does not start a separate call stack

} }

**Output**:running...



**Problem if you direct call run() method**

class TestCallRun2 extends Thread{

public void run(){

for(inti=1;i<5;i++){

try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}

System.out.println(i);

} }

public static void main(String args[]){

TestCallRun2 t1=new TestCallRun2();

TestCallRun2 t2=new TestCallRun2();

t1.run();

t2.run();

} }

**Output**:

1

2

3

4

5

1

2

3

4

5

As you can see in the above program that there is no context-switching because here t1 and t2 will be treated as normal object not thread object.