**MultiThreading & Synchronization**

**Submitted By**

**Sai Akhil. S**

**Table Of Contents :**

**Topic Page No.**

Multi Tasking 3

Multi Threading in Java 4

Thread Creation 5

Java Thread Class 6

Java Thread States 9

Thread Joining 14

Thread Priority 16

Lambda Expression 17

Thread Synchronization 18

**Multi Threading**

**Multi Tasking:**

A multitasking OS is associate degree OS that provides you the perception of two or additional tasks/jobs/processes running at a similar time. It will this by dividing system resources among these tasks/jobs/processes and change between the tasks/jobs/processes whereas they're capital punishment over and once again. Usually electronic equipment processes only 1 task at a time however the change is thus quick that it's like electronic equipment is capital punishment multiple processes at a time. They can support either preventative multitasking, wherever the OS doles out time to applications or cooperative multitasking, wherever the OS waits for the program to present back management, leading to hangs and crashes. Also referred to as Time-sharing, multitasking may be a logical extension of execution.

Multitasking programming is of two types :

1. Process-based Multitasking
2. Thread-based Multitasking

Process Based Multitasking :

* In this 2 or additional processes and programs are run at the same time.
* In method primarily based multitasking a method or a processes is that the smallest unit.
* Program is a bigger unit and requires more overhead.
* It requires its own address space.
* Here, it's unable to realize access over idle time of mainframe.
* It is comparatively heavy weight.
* It has slower rate of multi-tasking.

Thread Based Multitasking :

* In thread primarily based multitasking 2 or more threads will be run at the same time.
* In thread primarily based multitasking a thread is that the smallest unit.
* Thread is a smaller unit and requires less overhead.
* Threads share same address space.
* Thread to Thread communication is not complex.
* It permits taking gain access over idle time taken by hardware.
* It is comparatively light weight.
* It has faster data rate multi-tasking.

**Multi Threading in Java :**

Multi threading is a Java feature that enables instruction execution of 2 or additional components of a program for optimum utilization of CPU. Each a part of such program is termed a thread. So, threads are light-weight processes within a process.

Thread can be created by :  
1. Extending the Thread class  
2. Implementing the Runnable Interface

Thread creation by extending the Thread class :

We create a class that extends java.lang.Thread class.

Code :

**public** **class** SampleThreading **extends** Thread {

}

Thread creation by implementing the Runnable Interface :

We create a new class which implements java.lang.Runnable interface and override the run() method.

Code :

**public** **class** SampleThreading **implements** Runnable {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

}

}

**Note** : Since Java doesn’t support multiple inheritance, Runnable Interface is created so we can extend other Class and implement interface for creating a Thread.

Code :

**public** **class** SampleThreading **extends** Sample **implements** Runnable {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

}

}

Java Thread Class :

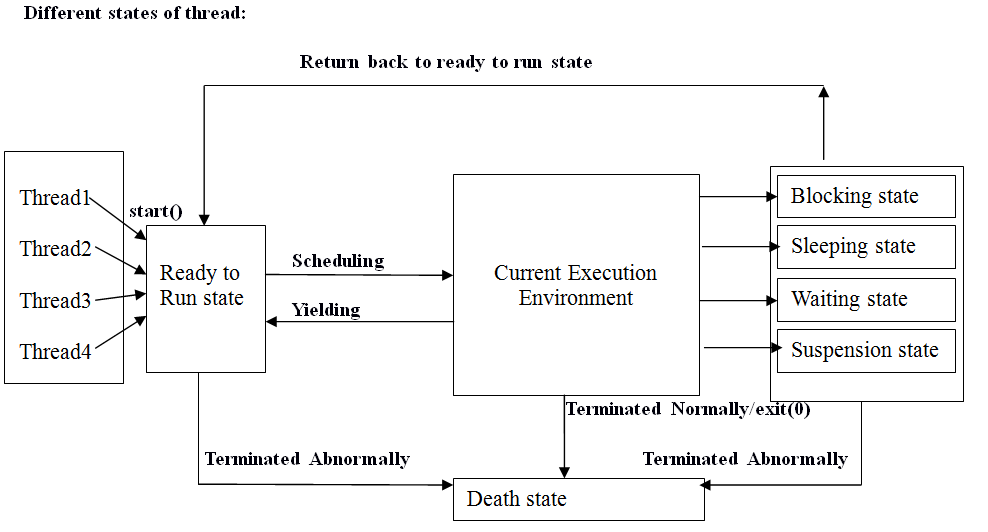
Java provides Thread class to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Java Thread Methods :

| Modifier and Type | Method | Description |
| --- | --- | --- |
| static int | [activeCount](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "activeCount())() | Returns an estimate of the number of active threads in the current thread's [thread group](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/ThreadGroup.html) and its subgroups. |
| void | [checkAccess](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "checkAccess())() | Determines if the currently running thread has permission to modify this thread. |
| protected [Object](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Object.html) | [clone](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "clone())() | Throws CloneNotSupportedException as a Thread can not be meaningfully cloned. |
| static [Thread](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html) | [currentThread](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "currentThread())() | Returns a reference to the currently executing thread object. |
| static void | [dumpStack](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "dumpStack())() | Prints a stack trace of the current thread to the standard error stream. |
| static int | [enumerate](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "enumerate(java.lang.Thread%5B%5D))([Thread](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html)[] tarray) | Copies into the specified array every active thread in the current thread's thread group and its subgroups. |
| static [Map](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/Map.html)<[Thread](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html).[StackTraceElement](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/StackTraceElement.html)[]> | [getAllStackTraces](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getAllStackTraces())() | Returns a map of stack traces for all live threads. |
| [ClassLoader](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/ClassLoader.html) | [getContextClassLoader](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getContextClassLoader())() | Returns the context ClassLoader for this thread. |
| static [Thread.UncaughtExceptionHandler](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.UncaughtExceptionHandler.html) | [getDefaultUncaughtExceptionHandler](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getDefaultUncaughtExceptionHandler())() | Returns the default handler invoked when a thread abruptly terminates due to an uncaught exception. |
| long | [getId](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getId())() | Returns the identifier of this Thread. |
| [String](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/String.html) | [getName](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getName())() | Returns this thread's name. |
| int | [getPriority](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getPriority())() | Returns this thread's priority. |
| [StackTraceElement](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/StackTraceElement.html)[] | [getStackTrace](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getStackTrace())() | Returns an array of stack trace elements representing the stack dump of this thread. |
| [Thread.State](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.State.html) | [getState](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getState())() | Returns the state of this thread. |
| [ThreadGroup](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/ThreadGroup.html) | [getThreadGroup](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getThreadGroup())() | Returns the thread group to which this thread belongs. |
| [Thread.UncaughtExceptionHandler](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.UncaughtExceptionHandler.html) | [getUncaughtExceptionHandler](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "getUncaughtExceptionHandler())() | Returns the handler invoked when this thread abruptly terminates due to an uncaught exception. |
| static boolean | [holdsLock](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "holdsLock(java.lang.Object))([Object](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Object.html) obj) | Returns true if and only if the current thread holds the monitor lock on the specified object. |
| void | [interrupt](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "interrupt())() | Interrupts this thread. |
| static boolean | [interrupted](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "interrupted())() | Tests whether the current thread has been interrupted. |
| boolean | [isAlive](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "isAlive())() | Tests if this thread is alive. |
| boolean | [isDaemon](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "isDaemon())() | Tests if this thread is a daemon thread. |
| boolean | [isInterrupted](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "isInterrupted())() | Tests whether this thread has been interrupted. |
| void | [join](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "join())() | Waits for this thread to die. |
| void | [join](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "join(long))(long millis) | Waits at most millis milliseconds for this thread to die. |
| void | [join](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "join(long,int))(long millis, int nanos) | Waits at most millis milliseconds plus nanos nanoseconds for this thread to die. |
| static void | [onSpinWait](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "onSpinWait())() | Indicates that the caller is momentarily unable to progress, until the occurrence of one or more actions on the part of other activities. |
| void | [run](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "run())() | If this thread was constructed using a separate Runnable run object, then that Runnable object's run method is called; otherwise, this method does nothing and returns. |
| void | [setContextClassLoader](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "setContextClassLoader(java.lang.ClassLoader))([ClassLoader](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/ClassLoader.html) cl) | Sets the context ClassLoader for this Thread. |
| void | [setDaemon](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "setDaemon(boolean))(boolean on) | Marks this thread as either a [daemon](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "isDaemon()) thread or a user thread. |
| static void | [SetDefaultUncaughtExceptionHandle](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "setDefaultUncaughtExceptionHandler(java.lang.Thread.UncaughtExceptionHandler))r  ([Thread.UncaughtExceptionHandler](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.UncaughtExceptionHandler.html) eh) | Set the default handler invoked when a thread abruptly terminates due to an uncaught exception, and no other handler has been defined for that thread. |
| void | [setName](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "setName(java.lang.String))([String](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/String.html) name) | Changes the name of this thread to be equal to the argument name. |
| void | [setPriority](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "setPriority(int))(int newPriority) | Changes the priority of this thread. |
| void | [SetUncaughtExceptionHandl](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "setUncaughtExceptionHandler(java.lang.Thread.UncaughtExceptionHandler))er ([Thread.UncaughtExceptionHandler](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.UncaughtExceptionHandler.html)eh) | Set the handler invoked when this thread abruptly terminates due to an uncaught exception. |
| static void | [sleep](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "sleep(long))(long millis) | Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers. |
| static void | [sleep](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "sleep(long,int))(long millis, intnanos) | Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds plus the specified number of nanoseconds, subject to the precision and accuracy of system timers and schedulers. |
| void | [start](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "start())() | Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread. |
| String | [toString](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "toString())() | Returns a string representation of this thread, including the thread's name, priority, and thread group. |
| static void | [yield](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.html" \l "yield())() | A hint to the scheduler that the current thread is willing to yield its current use of a processor. |

Java Thread States :

| Enum Constant | Description |
| --- | --- |
| [BLOCKED](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.State.html" \l "BLOCKED) | Thread state for a thread blocked waiting for a monitor lock. |
| [NEW](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.State.html" \l "NEW) | Thread state for a thread which has not yet started. |
| [RUNNABLE](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.State.html" \l "RUNNABLE) | Thread state for a runnable thread. |
| [TERMINATED](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.State.html" \l "TERMINATED) | Thread state for a terminated thread. |
| [TIMED\_WAITING](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.State.html" \l "TIMED_WAITING) | Thread state for a waiting thread with a specified waiting time. |
| [WAITING](https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Thread.State.html" \l "WAITING) | Thread state for a waiting thread. |



JVM provides Thread scheduler, which manages different states of the thread.

**1) Ready to run state:**

It is a state of waiting for CPU time. A Thread comes to ready to run state, When start() method is called on thread.

**2) Current execution state :**

It is a state of assigning CPU time to thread

**3) Blocking state :**

Thread goes to blocking state when performing I/O operations with external resources, such as keyboard, database server and other servers.

**4) Sleeping state :**

It is a state of waiting for shared resources for specific amount of time. A thread goes to sleeping state when sleep() is called on thread.

**5) waiting state :**

It is state of waiting for the shared resources until getting the resources. Thread goes to waiting state when wait() method called on thread.

**6) suspend state :**

If any child thread is going to perform any illegal transactions on the system resources, then main thread will suspend the child thread. If any thread is in suspend state it should be revoked by some other thread.

**7) Death state :**

A thread goes to death state, when terminated normally or abnormally (or) called exit(0) method on thread.

Once thread goes to death state it can not be invoked.

**8) Scheduling :**

Thread scheduler schedules the threads for execution. Thread scheduling can be done any one of the two ways.

1) priority scheduling (or) primitive scheduling

2) Round robin process.

**9) Yielding :**

If any thread enters into ready to run state (new or old thread ) thread scheduler checks priority of current execution thread with threads which are waiting for CPU time. If waiting thread is having more priority than the current execution thread, then thread scheduler forcibly brings the current execution thread into ready to run state.

**Creation of Thread :**

Any of the mentioned two ways above can be used to create a thread. Threading in Java is maintained by Java Virtual Machine (JVM).

Main Thread :

**public** **class** SampleThreading {

**public** **static** **void** main(String[] args) {

Sample s1 = **new** Sample();

}

}

Sample Class :

**public** **class** Sample **extends** Thread {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

}

}

**Running a thread :**

To run a thread first we extend the class with **java.lang.Thread** or by implementing **Runnable** Interface to a class. Then we create an object in reference with the thread class and start the thread by using the object and start() method.

Main Thread :

**public** **class** SampleThreading {

**public** **static** **void** main(String[] args) {

Sample s1=**new** Sample();

s1.start();

}

}

Sample class :

**public** **class** Sample **extends** Thread {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

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

}

}

Output :

Thread running

**Thread Joining :**

Main Thread :

**public** **class** SampleThreading {

**public** **static** **void** main(String[] args) {

Sample s1=**new** Sample();

s1.start();

System.***out***.println("End of Main");

}

}

Sample Class :

**public** **class** Sample **extends** Thread {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

System.***out***.println("Start of Thread");

}

}

Output :

End of Main

Start of Thread

In this example the main thread dies before the child processes terminates this leads to unexpected results or error during run-time as a thread should always have a parent process and the process should not die before terminating all the child processes, so join() method is used to prevent the parent process to hold until all the child processes are terminated.

Main Thread :

**public** **class** SampleThreading {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

Sample s1=**new** Sample();

s1.start();

s1.join();

System.***out***.println("End of Main");

}

}

Sample Class :

**public** **class** Sample **extends** Thread {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

System.***out***.println("Start of Thread");

}

}

Output :

Start of Thread

End of Main

**Thread Priority :**

Every thread has a priority, this is set (defined as 1) for every thread created. We can set our preferred priority by using setPriority() method. We can set the priority between any value ranging from MAX\_PRIORITY (defined as 10) to MIN\_PRIORITY (defined as 1). The child threads inherit thread priority from their parent.

Main Thread :

**public** **class** SampleThreading **extends** Thread{

**public** **static** **void** main(String[] args) {

Sample s1=**new** Sample();

System.***out***.println(s1.getPriority());

s1.setPriority(***MIN\_PRIORITY***);

System.***out***.println(s1.getPriority());

s1.setPriority(***NORM\_PRIORITY***);

System.***out***.println(s1.getPriority());

s1.setPriority(***MAX\_PRIORITY***);

System.***out***.println(s1.getPriority());

}

}

Output :

5

1

5

10

If the priority of the running thread is lower than that in the waiting state, the currently running thread will be put into wait state and the higher priority thread will be executed first.

**Lambda Expressions :**

Lambda expression is introduced to represent one method functional interface using an expression.

It can be used for iteration, filtering and extracting data in collection framework. Lambda expression is treated as function so a .class file is not created separately. A functional interface is one which has only one abstract method in it. It is provided with an annotation @FunctionalInterface to declare as an functional interface.

Main Thread without Lambda Expression :

**public** **class** SampleThreading {

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

System.***out***.println("This is a thread.");

}

});

t.start();

}

}

Output :

This is a thread.

Main Thread with Lambda Expression :

**public** **class** SampleThreading {

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread(() -> {

System.***out***.println("This is a thread.");

});

t.start();

}

}

Output :

This is a thread.

**Thread Synchronization :**

Main Thread :

**class** Counter {

**int** count;

**public** **void** increment() {

count++;

}

}

**public** **class** Syncing {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

Counter c = **new** Counter();

Thread t1 = **new** Thread(() -> {

**for** (**int** i = 1; i <= 10000; i++) {

c.increment();

}

});

Thread t2 = **new** Thread(() -> {

**for** (**int** i = 1; i <= 10000; i++) {

c.increment();

}

});

t1.start();

t2.start();

t1.join();

t2.join();

System.***out***.println("Count : " + c.count);

}

}

Output : Excepted : (Count : 20000)

Count : 15299

Count : 15972

Count : 16670

Count : 14611

Count : 15366

For the above example the expected output would be 20000 but the actual result would not be 20000 as ervery time the threads try to access the same data they crash and produce unexpected results. To overcome this issue synchronized keyword is used.

The synchronized keyword prevents multiple threads to access the data at the same time.

When a thread is already accessing the data, and if another thread is trying to access the same data it will be put to wait state.

Main Thread :

**class** Counter {

**int** count;

**public** **synchronized** **void** increment() {

count++;

}

}

**public** **class** Syncing {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

Counter c = **new** Counter();

Thread t1 = **new** Thread(() -> {

**for** (**int** i = 1; i <= 10000; i++) {

c.increment();

}

});

Thread t2 = **new** Thread(() -> {

**for** (**int** i = 1; i <= 10000; i++) {

c.increment();

}

});

t1.start();

t2.start();

t1.join();

t2.join();

System.***out***.println("Count : " + c.count);

}

}

Output :

Count : 20000

Another example when synchronized keyword is used is to prevent threads to access data even before the data is being generated by preventing this we can ensure that the threads do not raise any exceptions during runtime.

Below is an example about a producer and a consumer, where in a consumer can only produce products if and only if the producer has produced the products and are available for consumption.

Main Thread :

**class** Test {

**int** num;

**boolean** value = **false**;

**public** **synchronized** **void** put(**int** num) {

**while** (value) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

System.***out***.println("Produce : " + num);

**this**.num = num;

value = **true**;

notify();

}

**public** **synchronized** **void** get() {

**while** (!value) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

System.***out***.println("Consume : " + num);

value = **false**;

notify();

}

}

**class** Producer **implements** Runnable {

Test t;

**public** Producer(Test t) {

**this**.t = t;

Thread t1 = **new** Thread(**this**, "Producer");

t1.start();

}

**public** **void** run() {

**int** i = 0;

**while** (**true**) {

t.put(i++);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**class** Consumer **implements** Runnable {

Test t;

**public** Consumer(Test t) {

**this**.t = t;

Thread t2 = **new** Thread(**this**, "Consumer");

t2.start();

}

**public** **void** run() {

**while** (**true**) {

t.get();

**try** {

Thread.*sleep*(5000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**public** **class** PCTest {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

Test t = **new** Test();

**new** Producer(t);

**new** Consumer(t);

}

}

Output :

Produce : 0

Consume : 0

Produce : 1

Consume : 1

Produce : 2

Consume : 2

Produce : 3

Consume : 3

Produce : 4

Consume : 4

Produce : 5

Consume : 5

Here the consumer will only run when the boolean condition is met, the boolean condition is set but the producer if it has incremented the value.