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 - 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.

2 Ways to create Thread -

1. We can create a Thread by extending Thread class – start() method of Thread class is used to start a newly created Thread. After this, a thread moves from new state to Runnable state.

**public** **class** InterfacePrac **extends** Thread {

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

InterfacePrac i = **new** InterfacePrac();

i.start();

// Minute observation - If we extends Thread class to the current Class, then after creating an instance of the current class, we can call start() method to initiate the thread. No need to separately create an instance of Thread class. But if we didn't extend Thread class. Current class object instance won't be able to call start() method itself.

@Override

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

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

**for**(**int** i=0;i<5;i++) {

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

System.***out***.println(e);

}

System.***out***.println(i);

}

}

Note : Run() method will automatically triggers once start() method kicked out

Here, we have used keyword @override.. Its not mandatory to use this keyword. Its just a reference that says run() method belongs to Thread class.

1. We can create Thread by implementing Runnable interface as well. If we don’t define Thread class, we just implement Runnable interface, then we have to explicitly define instance object of Thread class to initiate a Thread. Also, we have to pass the current class instance as parameter of Thread class object creation.

**public** **class** InterfacePrac **implements** Runnable {

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

InterfacePrac i = **new** InterfacePrac();

Thread t = **new** Thread(i);

t.start();

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

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

**for**(**int** i=0;i<5;i++) {

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

System.***out***.println(e);

}

System.***out***.println(i);

}

}

Note : Run() method will automatically triggers once start() method kicked out

We have seen sleep method of Thread class. And we know that at a time, only 1 thread will be executed. So, if we sleep a thread for the specified time, the thread schedular picks up another thread and so on.

Note : We can’t start a thread twice. 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.

In above example, just mention 1 more sentence,

t.start()

We’ll get IllegalThreadStateException

**Imp Concept : What happen if we call run() method directly instead of start() method –**

Ans : 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.

We’ll see this concept with 2 code examples:

Ex1 : We’ll just use 1 Thread, and will not initiate start() method, we just call out run() method. So logic is that, it will invoke run() method of Thread class, but it will run like a normal method, not a thread method, point is we have to call run() method explicitly,

**public** **class** InterfacePrac **extends** Thread {

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

InterfacePrac i = **new** InterfacePrac();

**i.run();**

**}**

**public void run() {**

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

}

Output : Thread started.

But this will not run as a thread object.

Ex2: Now, we’ll use 2 Threads. And call 2 run() methods.

**public** **class** InterfacePrac **extends** Thread {

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

InterfacePrac i = **new** InterfacePrac();

InterfacePrac i1 = **new** InterfacePrac();

//i.start();

//i1.start();

i.run();

i1.run();

}

@Override

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

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

**for**(**int** i=0;i<4;i++) {

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

System.***out***.println(e);

}

System.***out***.println(i);

}

}}

Output :

Thread started

0

1

2

3

Thread started

0

1

2

3

Logic is : that there is no context-switching because here i and i1 will be treated as normal object not thread object. So, it will run one after another.

**Join() method in Thread –**

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

Ie. If we use join() method to any thread, and there are some other threads also persists in the program. Then the thread to which join() appends, that method will end its execution first. After that, other threads will starts its execution.

**public** **class** InterfacePrac **extends** Thread {

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

InterfacePrac t1 = **new** InterfacePrac();

InterfacePrac t2 = **new** InterfacePrac();

InterfacePrac t3 = **new** InterfacePrac();

t1.start();

**try** {

t1.join();

} **catch** (Exception e) {

System.***out***.println(e);

}

t2.start();

t3.start();

}

@Override

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

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

**for**(**int** i=0;i<5;i++) {

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

System.***out***.println(e);

}

System.***out***.println(i);

}

}

}

// Since we have attached join() with Thread t1. So Execution will stop for other threads before thread t1 completes its execution. After t1 completed its execution, then thread t2 and thread t3 act as a Thread object.

Few more methods of Thread class –

1. getName() – It returns the name of the thread.
2. setName() – It changes the name of the thread.
3. getId() – It returns the ID of the thread.
4. currentThread() – It returns a reference to the currently executing Thread object.

Note : By-default each thread has a name i.e. thread-0, thread-1 and so on

**public** **class** InterfacePrac **extends** Thread {

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

InterfacePrac t1 = **new** InterfacePrac();

InterfacePrac t2 = **new** InterfacePrac();

InterfacePrac t3 = **new** InterfacePrac();

System.***out***.println(t1.getName());

System.***out***.println(t2.getName());

System.***out***.println(t1.getId());

t1.start();

t2.start();

System.***out***.println(Thread.*currentThread*());

t1.setName("Tushar"); // changes the name of this thread to be equal to argument name

System.***out***.println(t1.getName());

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

t3.start();

t2.setName(“Tushar”);

System.***out***.println(Thread.*currentThread*());

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

}

@Override

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

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

}

Priority of Thread :

Each thread has a priority. We have 3 constants defined in Thread class related to setting priority:

1. public static int MIN\_PRIORITY
2. public static int NORM\_PRIORITY
3. public static int MAX\_PRIORITY

Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10.

**public** **class** InterfacePrac **extends** Thread {

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

InterfacePrac t1 = **new** InterfacePrac();

InterfacePrac t3 = **new** InterfacePrac();

t3.setPriority(Thread.***MAX\_PRIORITY***);

t1.setPriority(Thread.***MIN\_PRIORITY***);

t1.start();

t3.start();

// Since we have set the priority Max priority to Thread t3, so it will run first before Thread t1. Also, Thread t3 will be assigned 1st thread name like thread-0

}

@Override

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

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

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

System.***out***.println(Thread.*currentThread*().getId());

System.***out***.println(Thread.*currentThread*().getPriority());

}}

Output :

Thread started

Thread-0

Thread started

Thread-1

12

10

11

1

Imp Note : When we see like, we instantiated 3 or 4 Thread instances or Current class Objects in case of Runnable interface. All these instances will hit single method run(). Its like we are performing a single task by Multiple threads. Means all the threads will execute run() method. In backend but, each thread runs in a separate callstack. This is the example of Single Task by multiple Threads.

Now, we’ll see Example of Multiple Tasks by Multiple Threads :