Creating a thread in Java

To implement multithreading, Java defines two ways by which a thread can be created.

- By implementing the **Runnable** interface.
- By extending the **Thread** class.

Implementing the Runnable Interface

The easiest way to create a thread is to create a class that implements the runnable interface. After implementing runnable interface, the class needs to implement the `run()` method.

Run Method Syntax:

```
public void run()
```

- It introduces a concurrent thread into your program. This thread will end when `run()` method terminates.
- You must specify the code that your thread will execute inside `run()` method.
- `run()` method can call other methods, can use other classes and declare variables just like any other normal method.

```
class MyThread implements Runnable
{
    public void run()
    {
        System.out.println("concurrent thread started running..");
    }
}

class MyThreadDemo
{
    public static void main(String args[])
    {
        MyThread mt = new MyThread();
        Thread t = new Thread(mt);
        t.start();
    }
}
```

Output:

```
concurrent thread started running..
```

To call the `run()` method, `start()` method is used. On calling start(), a new stack is provided to the thread and run() method is called to introduce the new thread into the program.

Note: If you are implementing Runnable interface in your class, then you need to explicitly create a Thread class object and need to pass the Runnable interface implemented class object as a parameter in its constructor.

Extending Thread class

This is another way to create a thread by a new class that extends **Thread** class and create an instance of that class. The extending class must override `run()` method which is the entry point of new thread.

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("concurrent thread started running..");
    }
}

classMyThreadDemo
{
    public static void main(String args[])
    {
        MyThread mt = new  MyThread();
        mt.start();
    }
}
```

Output:

```
concurrent thread started running..
```

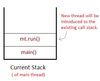

In this case also, we must override the `run()` and then use the `start()` method to run the thread. Also, when you create MyThread class object, Thread class constructor will also be invoked, as it is the super class, hence MyThread class object acts as Thread class object.

What if we call `run()` method directly without using `start()` method?

In above program if we directly call `run()` method, without using `start()` method,

```
public static void main(String args[])
{
    MyThread mt = new MyThread();
    mt.run();
}
```

Doing so, the thread won't be allocated a new call stack, and it will start running in the current call stack, that is the call stack of the **main** thread. Hence Multithreading won't be there.



Can we Start a thread twice?

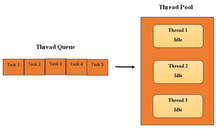
No, a thread cannot be started twice. If you try to do so, **IllegalThreadStateException** will be thrown.

```
public static void main(String args[])
{
    MyThread mt = new MyThread();
    mt.start();
    mt.start();     //Exception thrown
}
```

When a thread is in running state, and you try to start it again, or any method try to invoke that thread again using **start()** method, exception is thrown.

Thread Pool

In Java, is used for reusing the threads which were created previously for executing the current task. It also provides the solution if any problem occurs in the thread cycle or in resource thrashing. In Java Thread pool a group of threads are created, one thread is selected and assigned job and after completion of job, it is sent back in the group.



There are three methods of a Thread pool. They are as following:

1. newFixedThreadPool(int)
2. newCachedThreadPool()
3. newSingleThreadExecutor()

Following are the steps for creating a program of the thread pool

1. create a runnable object to execute.
2. using executors create an executor pool
3. Now Pass the object to the executor pool
4. At last shutdown the executor pool.

Example:

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class WorkerThread implements Runnable
    {
        private String message;
        public WorkerThread(String a)
        {
            this.message=a;
        }
    public void run()
    {
        System.out.println(Thread.currentThread().getName()+" (Start) message = "+message);
        processmessage();
        System.out.println(Thread.currentThread().getName()+" (End)");
    }
    private void processmessage()
    {
```

Output:

```
D:\Studytonight3>Java ThreadPoolDemo1.java

D:\Studytonight3>Java ThreadPoolDemo1
pool-1-thread-4 (Start) message = 3
pool-1-thread-5 (Start) message = 4
pool-1-thread-6 (Start) message = 1
pool-1-thread-1 (Start) message = 0
pool-1-thread-3 (Start) message = 2
pool-1-thread-2 (End)
pool-1-thread-5 (End)
pool-1-thread-6 (Start) message = 6
pool-1-thread-4 (End)
pool-1-thread-6 (Start) message = 7
pool-1-thread-3 (Start) message = 5
pool-1-thread-1 (End)
pool-1-thread-7 (Start) message = 8
pool-1-thread-1 (End)
pool-1-thread-8 (Start) message = 9
pool-1-thread-5 (End)
pool-1-thread-6 (End)
pool-1-thread-8 (End)
pool-1-thread-7 (End)
pool-1-thread-3 (End)
********All threads are finished********
```

← Prev Next →