L52

Monday, February 7, 2022 10:59 AM

Thread class, Runnable Interface

Q1. When a Java program is launched, JVM uses the main method in a class as the starting point for code execution. Similarly, the run is the starting point for a **Thread**.

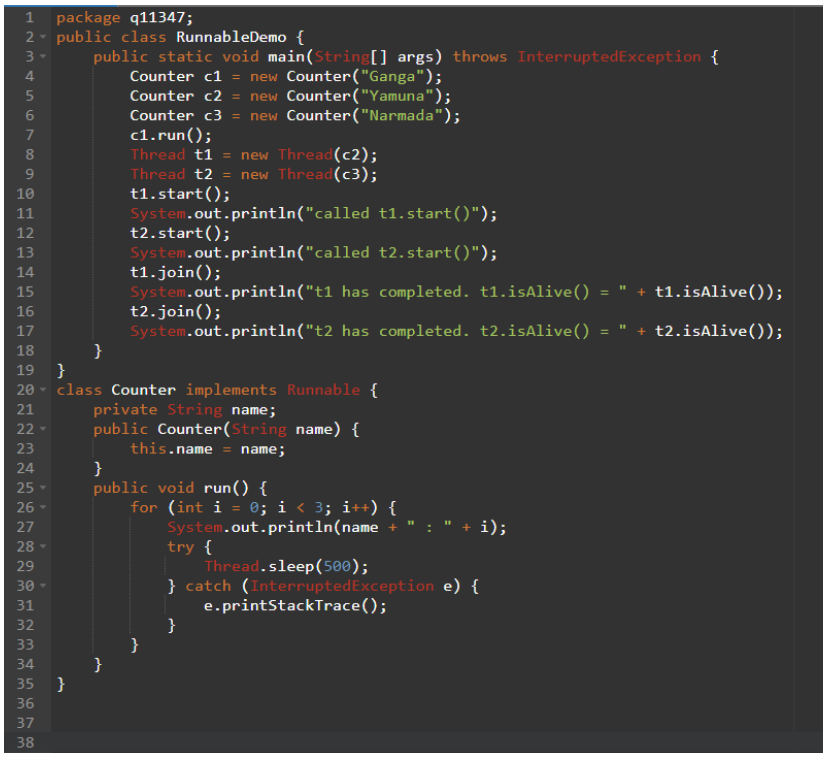
It is important to note that the run() should not be called directly. If called directly it will executed as any other normal method in the context of the current thread sequentially.

When we want the code in the run() method to be executed in a separate thread simultaneously (asynchronously), we hand over the runnable instance (which contains the run() method) to a thread object and ask the thread object to start executing. The call to the start() method on the thread object triggers the execution of the code in the run() in that thread's context.

Java also provides executors which generally hold a pool of threads to execute runnable instances. These are recommended in large-scale applications for efficient management of threads. Different types of executors can be created by calling the static methods available in the utility class Executors which is present in the java.util.concurrent package.

See and retype the below code to learn the difference between calling run() method directly and executing it in a thread.

You will notice in the output that the thread executing the main method itself is busy executing the c1.run();. And until it complete the run() on c1, main method does not even proceed to create and start threads t1 and t2. However, once the threads t1 and t2 are started, from the output you notice that the CPU time is sliced between both the threads and both of them are executing simultaneously.



```java
package q11347;
public class RunnableDemo {
    public static void main(String[] args) throws InterruptedException {
        Counter c1 = new Counter("Ganga");
        Counter c2 = new Counter("Yamuna");
        Counter c3 = new Counter("Narmada");
        c1.run();
        Thread t1 = new Thread(c2);
        Thread t2 = new Thread(c3);
        t1.start();
        System.out.println("called t1.start()");
        t2.start();
        System.out.println("called t2.start()");
        t1.join();
        System.out.println("t1 has completed. t1.isAlive() = " + t1.isAlive());
        t2.join();
        System.out.println("t2 has completed. t2.isAlive() = " + t2.isAlive());
    }
}
class Counter implements Runnable {
    private String name;
    public Counter(String name) {
        this.name = name;
    }
    public void run() {
        for (int i = 0; i < 3; i++) {
            System.out.println(name + " : " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Q2. Read the following code and select **all** the correct outputs:

```java
public class MyRunnable implements Runnable {
        public void run() {
                System.out.println("In run() method ");
                throw new RuntimeException("some problem");
        }
public static void main(String[] args) {
                Thread t1 = new Thread(new MyRunnable());
                t1.start();
                System.out.println("End of main() method");
        }
}
```

☐ In run() method

☐ End of main() method

☐ some problem

[✓ In run() method
End of main() method
java.lang.RuntimeException: some problem]

☐ In run() method
java.lang.RuntimeException: some problem

[✓ In run() method
java.lang.RuntimeException: some problem
End of main() method]

[✓ End of main() method
In run() method
java.lang.RuntimeException: some problem]

☐ java.lang.RuntimeException: some problem

Q3. Write a Java program that uses three threads to perform the below actions:

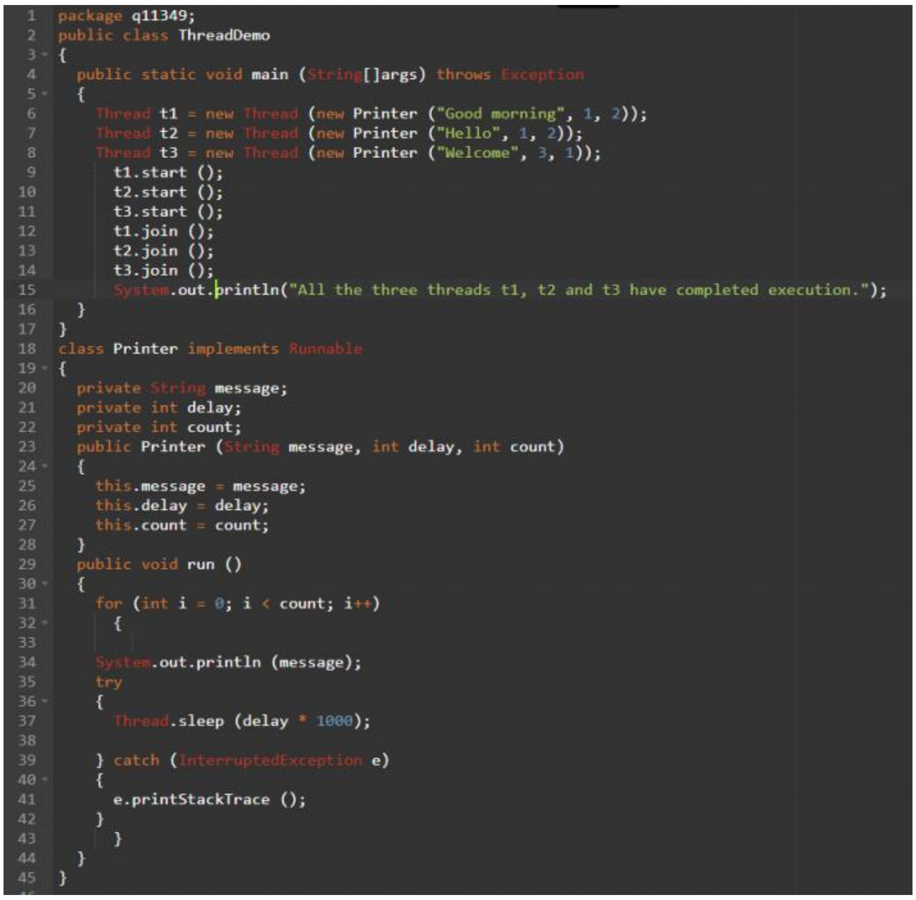
1. First thread should print "Good morning" for every 1 second for 2 times
2. Second thread should print "Hello" for every 1 seconds for 2 times
3. Third thread should print "Welcome" for every 3 seconds for 1 times

Write appropriate **constructor** in the Printer class which implements Runnable interface to take three arguments : **message, delay** and count of types **String, int** and **int** respectively.

Write code in the Printer.run() method to print the **message** with appropriate **delay** and for number of times mentioned in **count.**

Write a class called ThreadDemo with the main() method which instantiates and executes three instances of the above mentioned Printer class as threads to produce the desired output.

[**Note:** If you want to sleep for **2** seconds you should call Thread.sleep(2000); as the Thread.sleep(...) method takes milliseconds as argument.]



```java
package q11349;
public class ThreadDemo
{
  public static void main (String[]args) throws Exception
  {
    Thread t1 = new Thread (new Printer ("Good morning", 1, 2));
    Thread t2 = new Thread (new Printer ("Hello", 1, 2));
    Thread t3 = new Thread (new Printer ("Welcome", 3, 1));
      t1.start ();
      t2.start ();
      t3.start ();
      t1.join ();
      t2.join ();
      t3.join ();
      System.out.println("All the three threads t1, t2 and t3 have completed execution.");
  }
}
class Printer implements Runnable
{
  private String message;
  private int delay;
  private int count;
  public Printer (String message, int delay, int count)
  {
    this.message = message;
    this.delay = delay;
    this.count = count;
  }
  public void run ()
  {
    for (int i = 0; i < count; i++)
      {

    System.out.println (message);
    try
    {
      Thread.sleep (delay * 1000);

    } catch (InterruptedException e)
    {
      e.printStackTrace ();
    }
      }
  }
}
```

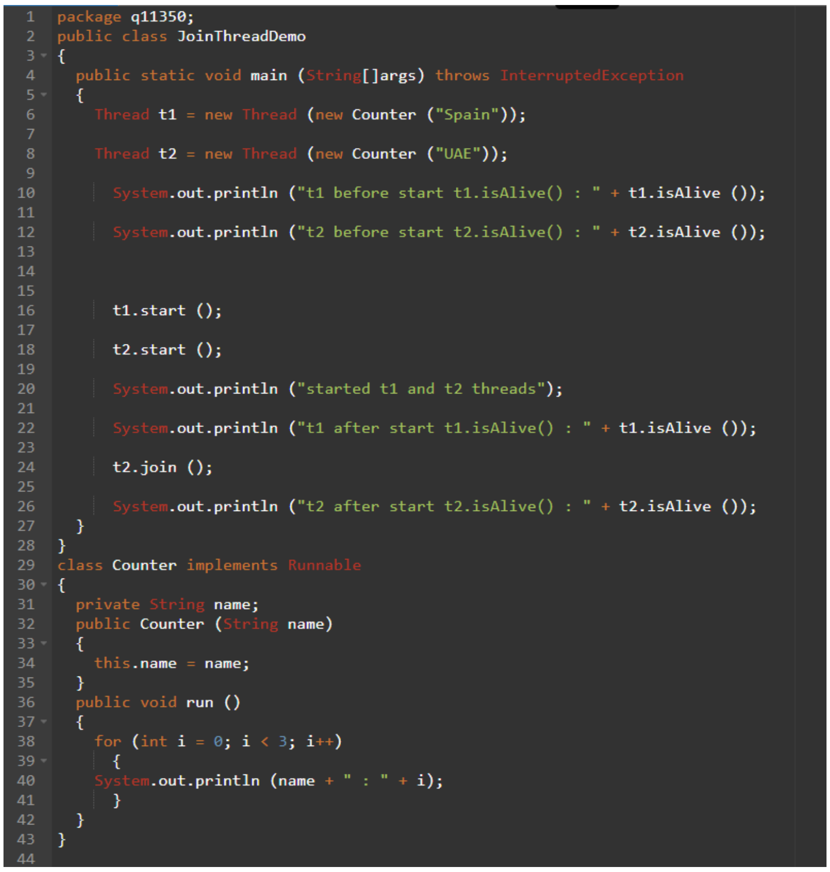
Q4. Write a Java program to demonstrate the usage of isAlive() and join() methods in threads.

Write a class JoinThreadDemo with a main() method that creates and executes two instances of Counter class which implements Runnable interface.

Let the Counter class take a **String** argument **name** and let its run() method print that message for **10** times along with the current count as given below:
System.out.println(name + " : " + i);
The JoinThreadDemo.main() method should perform the below tasks in the given order :

1. Create the first instance of thread as t1 with an instance of Counter class using **"Spain"** as the argument.

2. Create the second instance of thread as t2 with an instance of Counter class using **"UAE"** as the argument.
3. Print the isAlive() status of t1 as : **"t1 before start t1.isAlive() : " + t1.isAlive()**.
4. Print the isAlive() status of t1 as : **"t2 before start t2.isAlive() : " + t2.isAlive()**.
5. Start t1 and t2 threads respectively.
6. Print a message to the console as : **"started t1 and t2 threads"**.
7. Print the isAlive() status of t1 as : **"t1 after start t1.isAlive() : " + t1.isAlive()**.
8. Invoke the join() method on t2.
9. Print the isAlive() status of t1 as : **"t2 after start t2.isAlive() : " + t2.isAlive()**.



```
package q11350;
public class JoinThreadDemo
{
  public static void main (String[]args) throws InterruptedException
  {
    Thread t1 = new Thread (new Counter ("Spain"));

    Thread t2 = new Thread (new Counter ("UAE"));

      System.out.println ("t1 before start t1.isAlive() : " + t1.isAlive ());

      System.out.println ("t2 before start t2.isAlive() : " + t2.isAlive ());



      t1.start ();

      t2.start ();

      System.out.println ("started t1 and t2 threads");

      System.out.println ("t1 after start t1.isAlive() : " + t1.isAlive ());

      t2.join ();

      System.out.println ("t2 after start t2.isAlive() : " + t2.isAlive ());
  }
}
class Counter implements Runnable
{
  private String name;
  public Counter (String name)
  {
    this.name = name;
  }
  public void run ()
  {
    for (int i = 0; i < 3; i++)
      {
    System.out.println (name + " : " + i);
      }
  }
}
```

Q5. Daemon thread is a low priority thread (in the context of **JVM**) that runs in background to perform tasks such as garbage collection etc., they do not prevent the **JVM** from exiting when all the user threads finish their execution.

JVM terminates itself when all user threads finish their execution, even while Daemon threads are running.

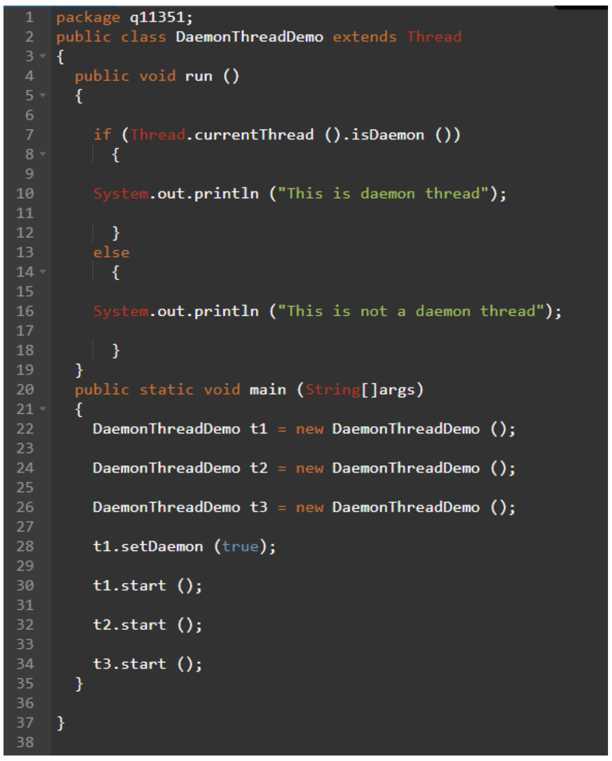
The code Thread.currentThread().isDaemon() will return true, if the current thread **is a daemon** thread and false if it **is not a daemon** thread.

Write a Java program to illustrate **daemon threads.**

Write a class DeamonThreadDemo which extends the Thread class. Override its **run()** method to check whether the current thread is either **daemon** or **user thread** and print **"This is daemon thread"** and **"This is not a daemon thread"** respectively.

Write the **main()** method in the class DeamonThreadDemo, which create three instances of class DaemonThreadDemo as t1, t2 and t3 and perform the below tasks in the given order:

1. invoke the setDaemon() method on t1 instance and pass **true** as the argument to set t1 as a daemon thread.
2. Invoke start() method on t1, t2 and t3 respectively.



```
package q11351;
public class DaemonThreadDemo extends Thread
{
  public void run ()
  {

    if (Thread.currentThread ().isDaemon ())
      {

    System.out.println ("This is daemon thread");

      }
    else
      {

    System.out.println ("This is not a daemon thread");

      }
  }
  public static void main (String[]args)
  {
    DaemonThreadDemo t1 = new DaemonThreadDemo ();

    DaemonThreadDemo t2 = new DaemonThreadDemo ();

    DaemonThreadDemo t3 = new DaemonThreadDemo ();

    t1.setDaemon (true);

    t1.start ();

    t2.start ();

    t3.start ();
  }

}

```