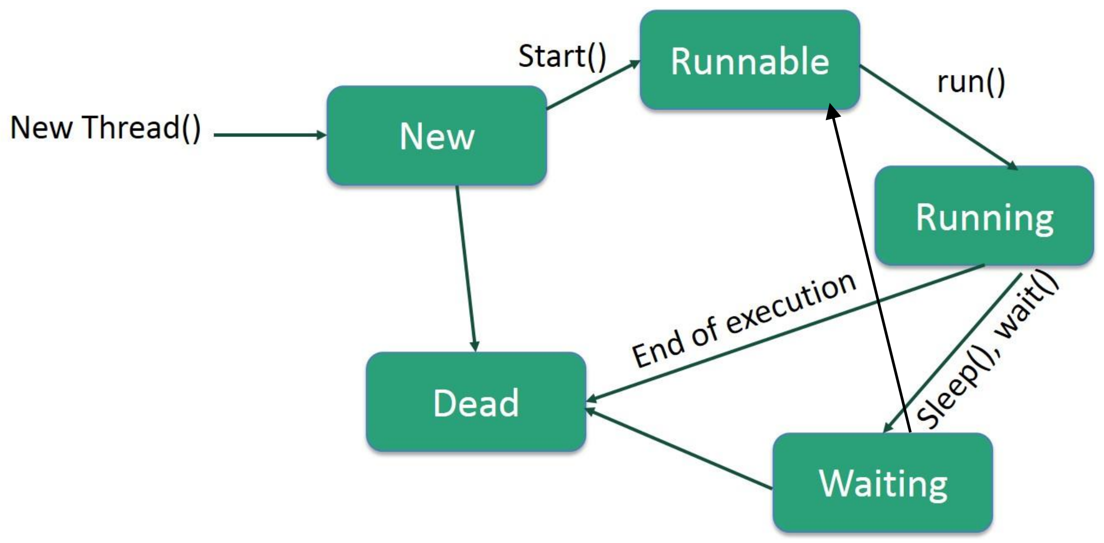
Mutlithreading

Java is a *multi-threaded programming language*. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application. Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle −

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.

- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Thread Priorities

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled. Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

Create a Thread by Implementing a Runnable Interface

If your class is intended to be executed as a thread then you can achieve this by implementing a **Runnable** interface. You will need to follow three basic steps −

Step 1

As a first step, you need to implement a run() method provided by a **Runnable** interface. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of the run() method −

```
public void run( )
```

Step 2

As a second step, you will instantiate a **Thread** object using the following constructor −

```
Thread(Runnable threadObj, String threadName);
```

Where, *threadObj* is an instance of a class that implements the **Runnable** interface and **threadName** is the name given to the new thread.

Step 3

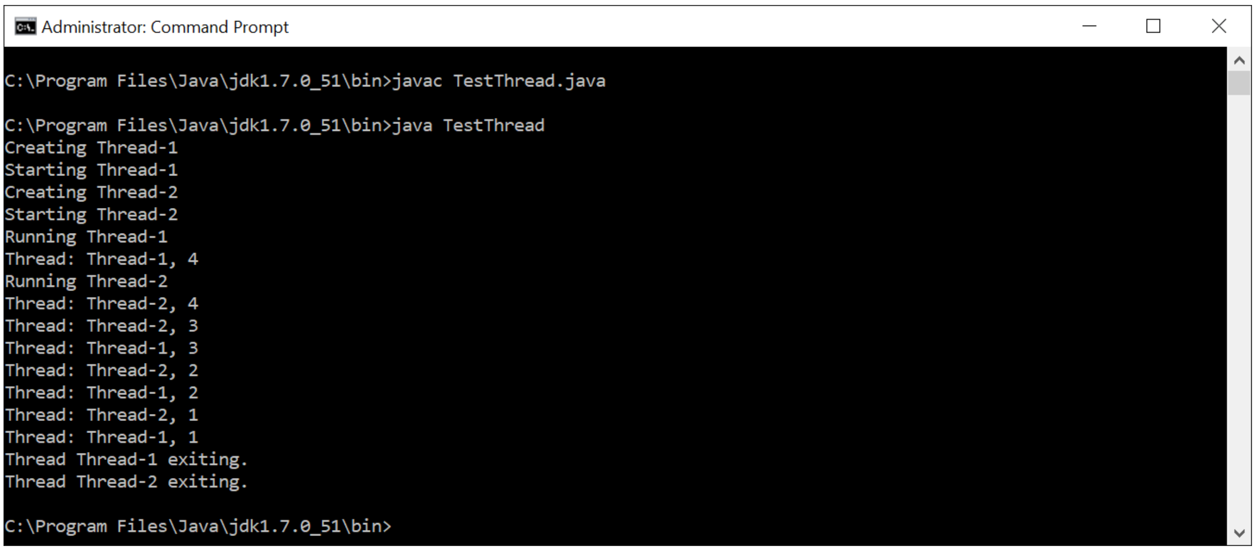
Once a Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method −

```
void start();
```

Example

Here is an example that creates a new thread and starts running it −

```
class RunnableDemo implements Runnable {
   private Thread t;
   private String threadName;

   RunnableDemo( String name) {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }

   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
      }catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
      }
      System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}

public class TestThread {
   public static void main(String args[]) {
      RunnableDemo R1 = new RunnableDemo( "Thread-1");
      R1.start();

      RunnableDemo R2 = new RunnableDemo( "Thread-2");
      R2.start();
   }
}
```



```
C:\Program Files\Java\jdk1.7.0_51\bin>javac TestThread.java

C:\Program Files\Java\jdk1.7.0_51\bin>java TestThread
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-2, 3
Thread: Thread-1, 3
Thread: Thread-2, 2
Thread: Thread-1, 2
Thread: Thread-2, 1
Thread: Thread-1, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.

C:\Program Files\Java\jdk1.7.0_51\bin>
```

OR

```
C:\Program Files\Java\jdk1.7.0_51\bin>javac TestThread.java


C:\Program Files\Java\jdk1.7.0_51\bin>java TestThread
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-2, 3
Thread: Thread-1, 3
Thread: Thread-2, 2
Thread: Thread-1, 2
Thread: Thread-2, 1
Thread: Thread-1, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Create a Thread by Extending a Thread Class

The second way to create a thread is to create a new class that extends **Thread** class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1

You will need to override **run( )** method available in Thread class. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of run() method −

```
public void run( )
```

Step 2

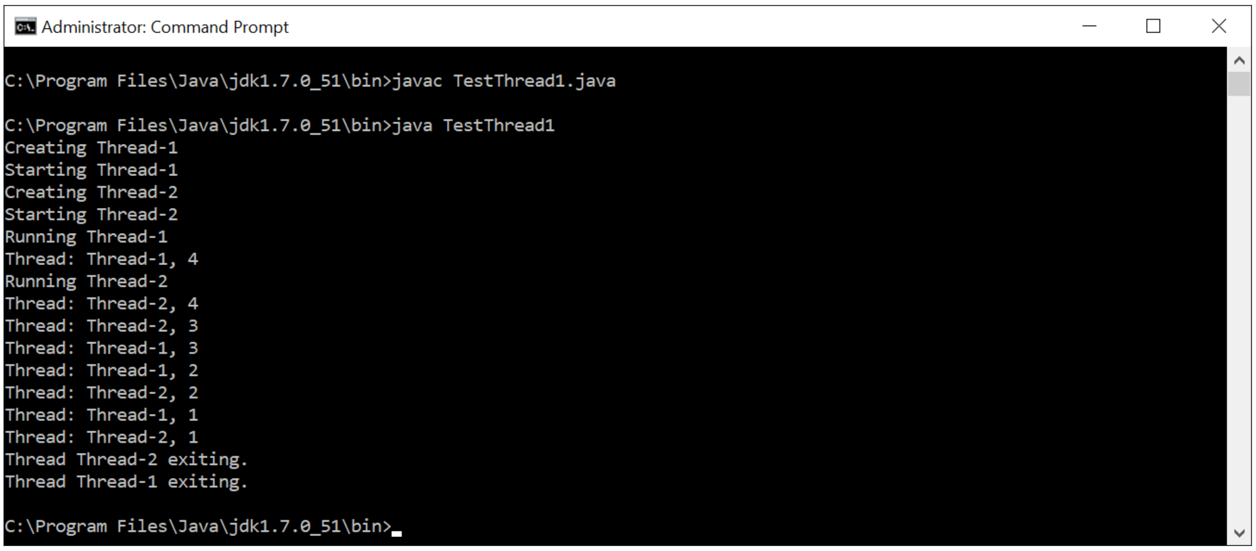
Once Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method −

```
void start( );
```

Example

Here is the preceding program rewritten to extend the Thread –

```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;

   ThreadDemo( String name) {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }

   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
      }catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
      }
      System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}

public class TestThread1 {

   public static void main(String args[]) {
      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();

      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```



```
C:\Program Files\Java\jdk1.7.0_51\bin>javac TestThread1.java

C:\Program Files\Java\jdk1.7.0_51\bin>java TestThread1
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-2, 3
Thread: Thread-1, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-2 exiting.
Thread Thread-1 exiting.

C:\Program Files\Java\jdk1.7.0_51\bin>
```

Or

```
C:\Program Files\Java\jdk1.7.0_51\bin>javac TestThread1.java


C:\Program Files\Java\jdk1.7.0_51\bin>java TestThread1
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-2, 3
Thread: Thread-1, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-2 exiting.
Thread Thread-1 exiting.
```

Thread Methods

Following is the list of important methods available in the Thread class.

Sr.No.	Method & Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

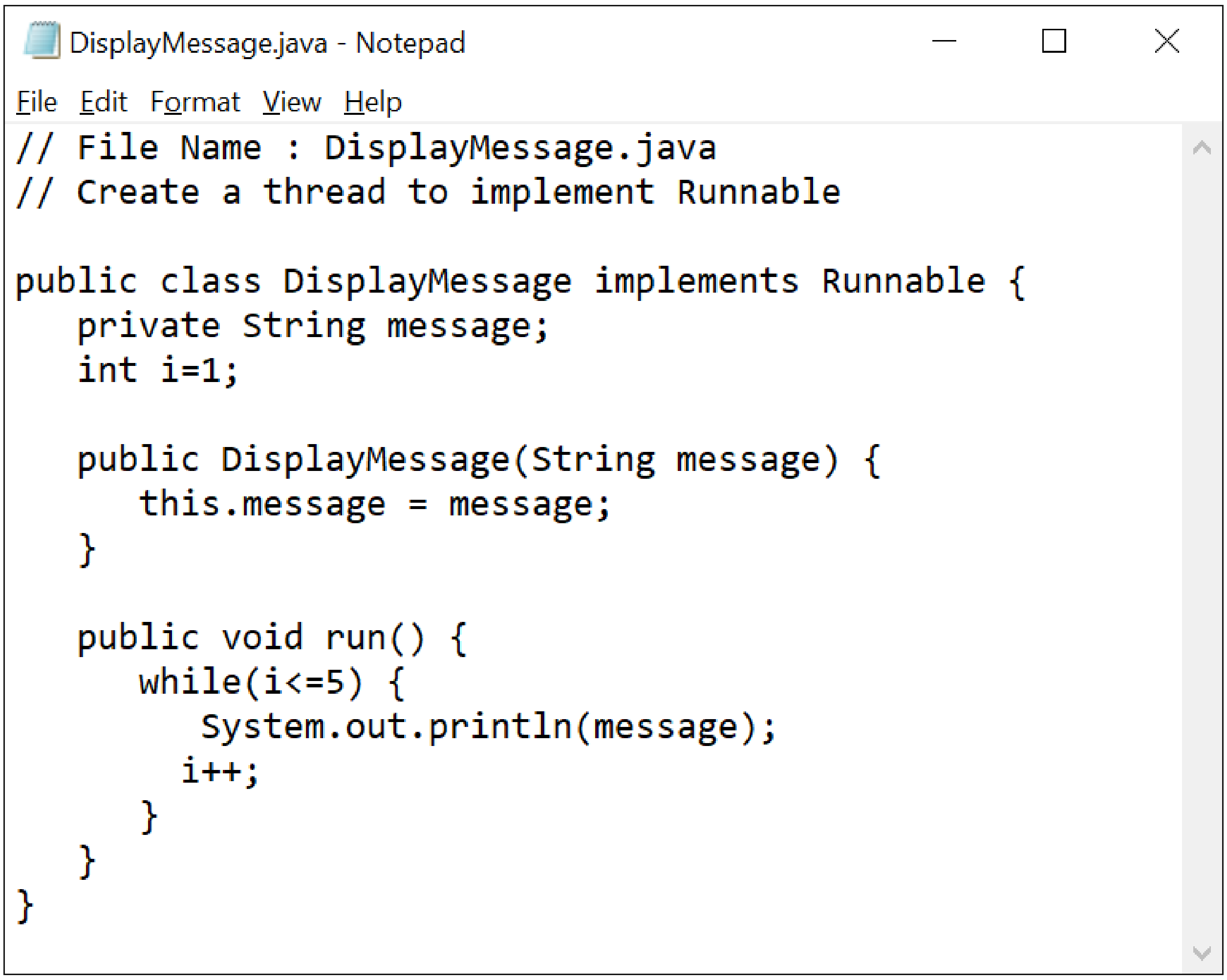
The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread.

Sr.No.	Method & Description
1	**public static void yield()** Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.
2	**public static void sleep(long millisec)** Causes the currently running thread to block for at least the specified number of milliseconds.
3	**public static boolean holdsLock(Object x)** Returns true if the current thread holds the lock on the given Object.
4	**public static Thread currentThread()** Returns a reference to the currently running thread, which is the thread that invokes this method.
5	**public static void dumpStack()** Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

Example

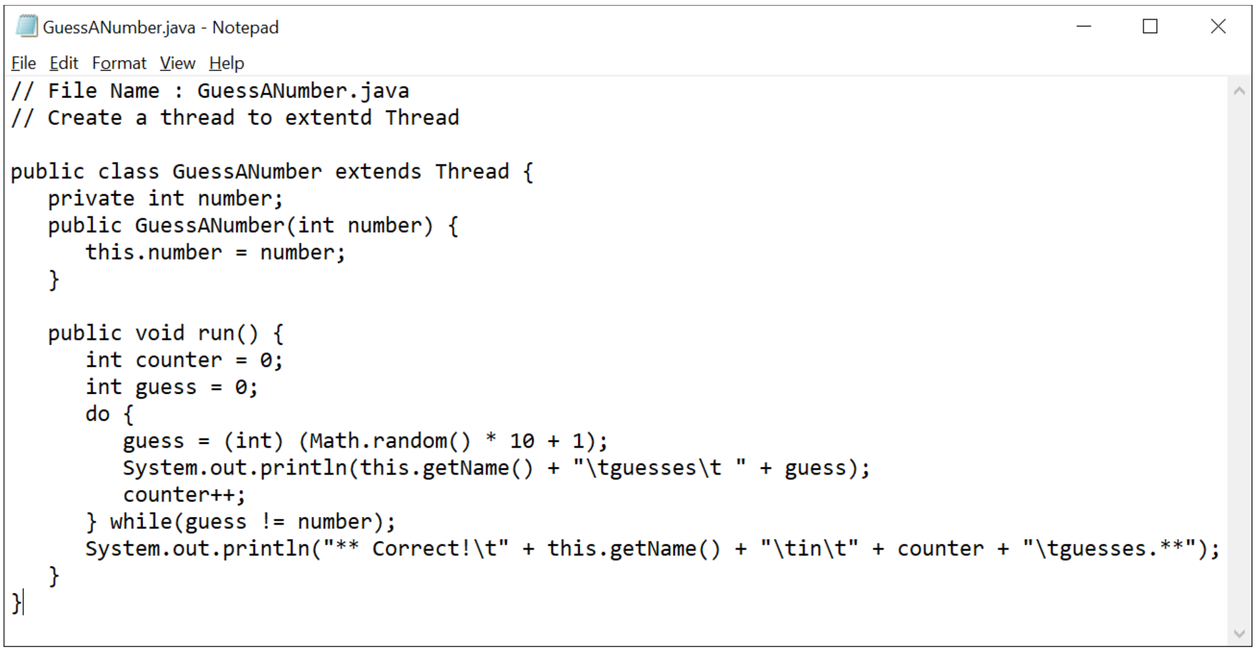
The following ThreadClassDemo program demonstrates some of these methods of the Thread class. Consider a class **DisplayMessage** which implements **Runnable** −

First file

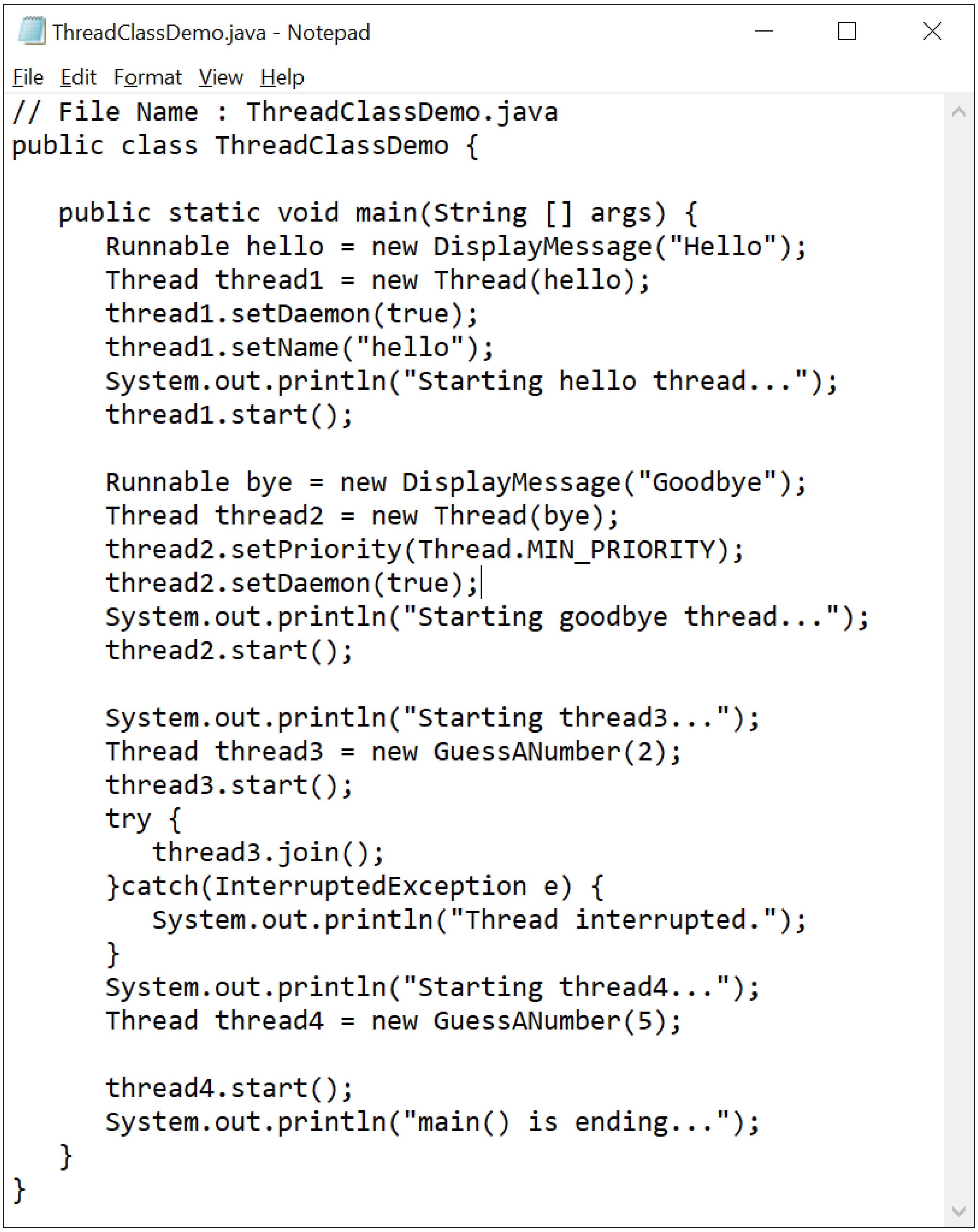


```
// File Name : DisplayMessage.java
// Create a thread to implement Runnable

public class DisplayMessage implements Runnable {
   private String message;
   int i=1;

   public DisplayMessage(String message) {
      this.message = message;
   }

   public void run() {
      while(i<=5) {
         System.out.println(message);
        i++;
      }
   }
}
```

Second file



```
// File Name : GuessANumber.java
// Create a thread to extentd Thread

public class GuessANumber extends Thread {
   private int number;
   public GuessANumber(int number) {
      this.number = number;
   }

   public void run() {
      int counter = 0;
      int guess = 0;
      do {
         guess = (int) (Math.random() * 10 + 1);
         System.out.println(this.getName() + "\tguesses\t " + guess);
         counter++;
      } while(guess != number);
      System.out.println("** Correct!\t" + this.getName() + "\tin\t" + counter + "\tguesses.**");
   }
}
```



```
// File Name : ThreadClassDemo.java
public class ThreadClassDemo {

   public static void main(String [] args) {
      Runnable hello = new DisplayMessage("Hello");
      Thread thread1 = new Thread(hello);
      thread1.setDaemon(true);
      thread1.setName("hello");
      System.out.println("Starting hello thread...");
      thread1.start();

      Runnable bye = new DisplayMessage("Goodbye");
      Thread thread2 = new Thread(bye);
      thread2.setPriority(Thread.MIN_PRIORITY);
      thread2.setDaemon(true);
      System.out.println("Starting goodbye thread...");
      thread2.start();

      System.out.println("Starting thread3...");
      Thread thread3 = new GuessANumber(2);
      thread3.start();
      try {
         thread3.join();
      }catch(InterruptedException e) {
         System.out.println("Thread interrupted.");
      }
      System.out.println("Starting thread4...");
      Thread thread4 = new GuessANumber(5);

      thread4.start();
      System.out.println("main() is ending...");
   }
}
```

Following is the sanpshop of how to run

Step by step execution no need to run first to java file. On every run we get different Output:

First run

```
C:\Program Files\Java\jdk1.7.0_51\bin>javac DisplayMessage.java
C:\Program Files\Java\jdk1.7.0_51\bin>javac GuessANumber.java
C:\Program Files\Java\jdk1.7.0_51\bin>javac ThreadClassDemo.java
C:\Program Files\Java\jdk1.7.0_51\bin>java ThreadClassDemo
Starting hello thread...
Starting goodbye thread...
Starting thread3...
Hello
Hello
Hello
Hello
Hello
Goodbye
Goodbye
Goodbye
Goodbye
Goodbye
Thread-2    guesses  9
Thread-2    guesses  7
Thread-2    guesses  1
Thread-2    guesses  5
Thread-2    guesses  4
Thread-2    guesses  9
Thread-2    guesses  1
Thread-2    guesses  6
Thread-2    guesses  4
Thread-2    guesses  4
Thread-2    guesses  2
** Correct!  Thread-2    in   11   guesses.**
Starting thread4...
```

```
main() is ending...
Thread-3        guesses  6
Thread-3        guesses  1
Thread-3        guesses  2
Thread-3        guesses  8
Thread-3        guesses  1
Thread-3        guesses  5
** Correct!    Thread-3      in     6      guesses.**
```

Second run:

```
C:\Program Files\Java\jdk1.7.0_51\bin>java ThreadClassDemo
Starting hello thread...
Starting goodbye thread...
Hello
Hello
Hello
Hello
Hello
Starting thread3...
Goodbye
Goodbye
Goodbye
Goodbye
Goodbye
Thread-2        guesses  9
Thread-2        guesses  10
Thread-2        guesses  8
Thread-2        guesses  3
Thread-2        guesses  10
Thread-2        guesses  1
Thread-2        guesses  5
Thread-2        guesses  5
```

```
Thread-2      guesses  5
Thread-2      guesses  6
Thread-2      guesses  2
** Correct!    Thread-2      in     11     guesses.**
Starting thread4...
main() is ending...
Thread-3      guesses  1
Thread-3      guesses  10
Thread-3      guesses  5
** Correct!    Thread-3      in     3      guesses.**
```

Q. Write a java program using runnable interface and with the help of thread class, create three threads. Run each thread 10 times and then stop thread execution.

```
class A implements Runnable{

        public void run(){
                int i;
                for(i=1;i<=10;i++){
                        System.out.println("Thread A : "+i);
                }

        }
}

class B implements Runnable{

        public void run(){
                int i;
                for(i=1;i<=10;i++){
                        System.out.println("Thread B : "+i);
                }

        }
}
```

```
class C implements Runnable{

        public void run(){
                int i;
                for(i=1;i<=10;i++){
                        System.out.println("Thread C : "+i);
                }

        }
}

class RunnableDemo
{
        public static void main(String hello[])throws Exception{
                System.out.println("Main starts");
                Thread t1 = new Thread(new A());
                Thread t2 = new Thread(new B());
                Thread t3 = new Thread(new C());
                t1.start();
                t2.start();
                t3.start();
                System.out.println("Main ends");
}//end main()
}//end class
```

Output:

```
C:\Program Files\Java\jdk1.7.0_51\bin>javac RunnableDemo.java
C:\Program Files\Java\jdk1.7.0_51\bin>java RunnableDemo
Main starts
Main ends
Thread B : 1
Thread B : 2
Thread B : 3
Thread B : 4
Thread B : 5
Thread B : 6
Thread B : 7
Thread B : 8
Thread B : 9
Thread B : 10
Thread C : 1
Thread A : 1
Thread C : 2
Thread A : 2
Thread C : 3
Thread A : 3
Thread C : 4
Thread A : 4
Thread C : 5
Thread A : 5
Thread C : 6
Thread A : 6
Thread C : 7
Thread A : 7
Thread C : 8
Thread A : 8
Thread C : 9
Thread C : 10
Thread A : 9
Thread A : 10
```

Q. Write a program that creates three threads. Make sure that the main thread executes last.

```
public class ThreadJoinExample{
        public static void main(String hi[]){
                Thread t1 = new Thread(new MyRunnable(),"t1");
                Thread t2 = new Thread(new MyRunnable(),"t2");
                Thread t3 = new Thread(new MyRunnable(),"t3");
                t1.start();

        //start second thread after waiting for 2 seconds or if its dead
                try
                {
                        t1.join(2000);
                }
                catch(InterruptedException e)
                {
                        System.out.println("Exception occurs:"+e);
                }
                t2.start();

        //start third thread only when first thread is dead
                try
                {
                t1.join();
                }
                catch (InterruptedException e)
                {
                        System.out.println("Exception occurs: "+e);
                }
                t3.start();

        //let all threads finish execution before finishing main thread
                try
                {
                        t1.join();
                        t2.join();
                        t3.join();
                }
                catch(InterruptedException e)
                {
                        System.out.println("Exception occurs: "+e);
                }
                        System.out.println("All threads are dead, exiting main thread");
                }
        }
        class MyRunnable implements Runnable
        {
                public void run()
                {
                System.out.println("Thread started:::"+Thread.currentThread().getName());
                try
                {
                        Thread.sleep(4000);
                    }
                    catch(InterruptedException e)
                    {
                              System.out.println("Exception occurs: "+e);
                    }
                    System.out.println("Thread ended:::"+Thread.currentThread().getName());

          }
}
```

Output:

```
C:\Program Files\Java\jdk1.7.0_51\bin>javac ThreadJoinExample.java

C:\Program Files\Java\jdk1.7.0_51\bin>java ThreadJoinExample
Thread started:::t1
Thread started:::t2
Thread ended:::t1
Thread started:::t3
Thread ended:::t2
Thread ended:::t3
All threads are dead, exiting main thread
```

Q. Write a program to create a thread which displays a message “Welcome to Java” 10 times with a delay of 5 seconds between messages. Use the Runnable interface.

```
class Display implements Runnable{
          public void run(){
                    int i;
                    try{
                                        for(i=1;i<=10;i++){
                                                  System.out.println("Welcome to Java");
                                                  Thread.sleep(5000);
                                        }
                      }
                    catch(InterruptedException e){
                              System.out.println("\n Interrupted");
                    }

          }
}
```

```
class DisplayMessage1{
        public static void main(String a[]){
                Thread t1 = new Thread(new Display());
                t1.start();
        }
}
```

```
C:\Program Files\Java\jdk1.7.0_51\bin>javac DisplayMessage1.java
C:\Program Files\Java\jdk1.7.0_51\bin>java DisplayMessage1
Welcome to Java
Welcome to Java
Welcome to Java
Welcome to Java
Welcome to Java
Welcome to Java
Welcome to Java
Welcome to Java
Welcome to Java
Welcome to Java
```

Q. Write a program to create 4 threads to perform 4 different arithmetic operations like addition, subtraction, multiplication and division. Accept two numbers from command line arguments and perform the operations using thread.

```
import java.io.*;
class Add extends Thread{
        int n1,n2;
        public Add(int x, int y){
                n1=x;
                n2=y;
        }

        public void run(){
                System.out.println("Addition is : "+(n1+n2));
        }
}

class Sub extends Thread{
        int n1,n2;
        public Sub(int x, int y){
```

```
			n1=x;
			n2=y;
		}

		public void run(){
			System.out.println("Subtraction is : "+(n1-n2));
		}
}

class Mul extends Thread{
		int n1,n2;
		public Mul(int x, int y){
			n1=x;
			n2=y;
		}

		public void run(){
			System.out.println("Multplication is : "+(n1*n2));
		}
}

class Div extends Thread{
		int n1,n2;
		public Div(int x, int y){
			n1=x;
			n2=y;
		}

		public void run(){
			System.out.println("Division is : "+(n1/n2));
		}
}

class ThreadDemo{
		public static void main(String ar[]){
			try{
			DataInputStream dis = new DataInputStream(System.in);
```

```
                        System.out.println("Enter two values");
                        int a = Integer.parseInt(dis.readLine());
                        int b = Integer.parseInt(dis.readLine());
                        new Add(a,b).start();
                        new Sub(a,b).start();
                        new Mul(a,b).start();
                        new Div(a,b).start();
                        }
                        catch(Exception e){}
            }
}
```

Output:

```
C:\Program Files\Java\jdk1.7.0_51\bin>javac ThreadDemo.java
Note: ThreadDemo.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
C:\Program Files\Java\jdk1.7.0_51\bin>java ThreadDemo
Enter two values
20
10
Subtraction is : 10
Addition is : 30
Multplication is : 200
Division is : 2
```

Use of synchronized method

```
//example of java synchronized method
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=10;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }
  }
}
```

```
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}
}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

```
C:\Program Files\Java\jdk1.7.0_51\bin>javac TestSynchronization2.java
C:\Program Files\Java\jdk1.7.0_51\bin>java TestSynchronization2
5
10
15
20
25
30
35
```

```
40
45
50
100
200
300
400
500
600
700
800
900
1000
```

Let’s see an example where multiple threads work on the same object and we use wait, notify and notifyAll methods.
These methods can be used to implement producer consumer problem where consumer threads are waiting for the objects in Queue and producer threads put object in queue and notify the waiting threads.

Message

A java bean class on which threads will work and call wait and notify methods.

```
public class WaitNotifyTest {

    public static void main(String[] args) {
        Message msg = new Message("process it");
        Waiter waiter = new Waiter(msg);
        new Thread(waiter,"waiter").start();

        Waiter waiter1 = new Waiter(msg);
        new Thread(waiter1, "waiter1").start();

        Notifier notifier = new Notifier(msg);
        new Thread(notifier, "notifier").start();
        System.out.println("All the threads are started");
    }
}
```

Waiter

A class that will wait for other threads to invoke notify methods to complete it's processing. Notice that Waiter thread is owning monitor on Message object using synchronized block.

```
public class Waiter implements Runnable{
    private Message msg;
    public Waiter(Message m){
        this.msg=m;
    }

    public void run() {
        String name = Thread.currentThread().getName();
        synchronized (msg) {
            try{
                System.out.println(name+" waiting to get notified at time:"+System.currentTimeMillis());
                msg.wait();
            }catch(InterruptedException e){
                e.printStackTrace();
            }
            System.out.println(name+" waiter thread got notified at time:"+System.currentTimeMillis());
            //process the message now
            System.out.println(name+" processed: "+msg.getMsg());
        }
    }
}
```

Notifier

A class that will process on Message object and then invoke notify method to wake up threads waiting for Message object. Notice that synchronized block is used to own the monitor of Message object.

```
public class Notifier implements Runnable {
    private Message msg;
    public Notifier(Message msg) {
        this.msg = msg;
    }

   public void run() {
        String name = Thread.currentThread().getName();
        System.out.println(name+" started");
        try {
            Thread.sleep(1000);
            synchronized (msg) {
                msg.setMsg(name+" Notifier work done");
                msg.notify();
                // msg.notifyAll();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
}
```

WaitNotifyTest

Test class that will create multiple threads of Waiter and Notifier and start them.

```
public class WaitNotifyTest {
    public static void main(String[] args) {
        Message msg = new Message("process it");
        Waiter waiter = new Waiter(msg);
        new Thread(waiter,"waiter").start();

        Waiter waiter1 = new Waiter(msg);
        new Thread(waiter1, "waiter1").start();

        Notifier notifier = new Notifier(msg);
        new Thread(notifier, "notifier").start();
        System.out.println("All the threads are started");
    }

}
```

```
C:\Program Files\Java\jdk1.7.0_51\bin>javac Message.java
C:\Program Files\Java\jdk1.7.0_51\bin>javac Waiter.java
C:\Program Files\Java\jdk1.7.0_51\bin>javac Notifier.java
C:\Program Files\Java\jdk1.7.0_51\bin>javac WaitNotifyTest.java
C:\Program Files\Java\jdk1.7.0_51\bin>java WaitNotifyTest
waiter waiting to get notified at time:1506391706818
waiter1 waiting to get notified at time:1506391706818
```

```
All the threads are started
notifier started
waiter waiter thread got notified at time:1506391707820
waiter processed: notifier Notifier work done
```

Explanation: When we will invoke the above program, we will see below output but program will not complete because there are two threads waiting on Message object and notify() method has wake up only one of them, the other thread is still waiting to get notified.