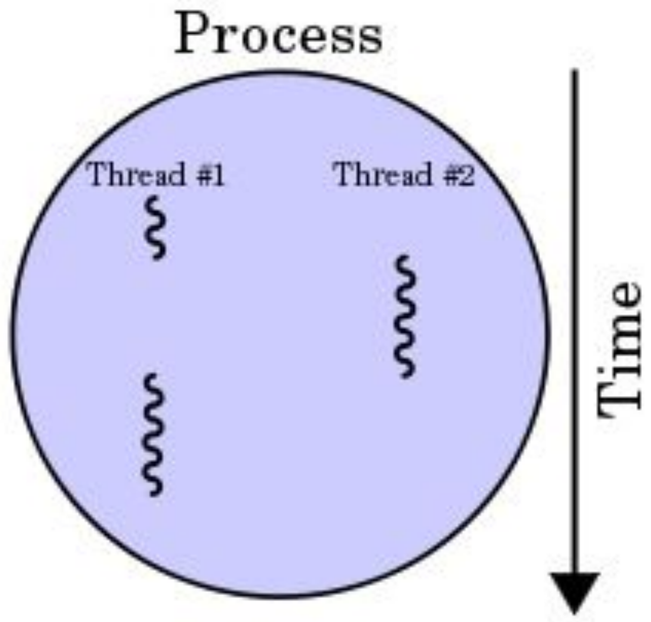
Java Multithreading
Unit-8

Rima Patel

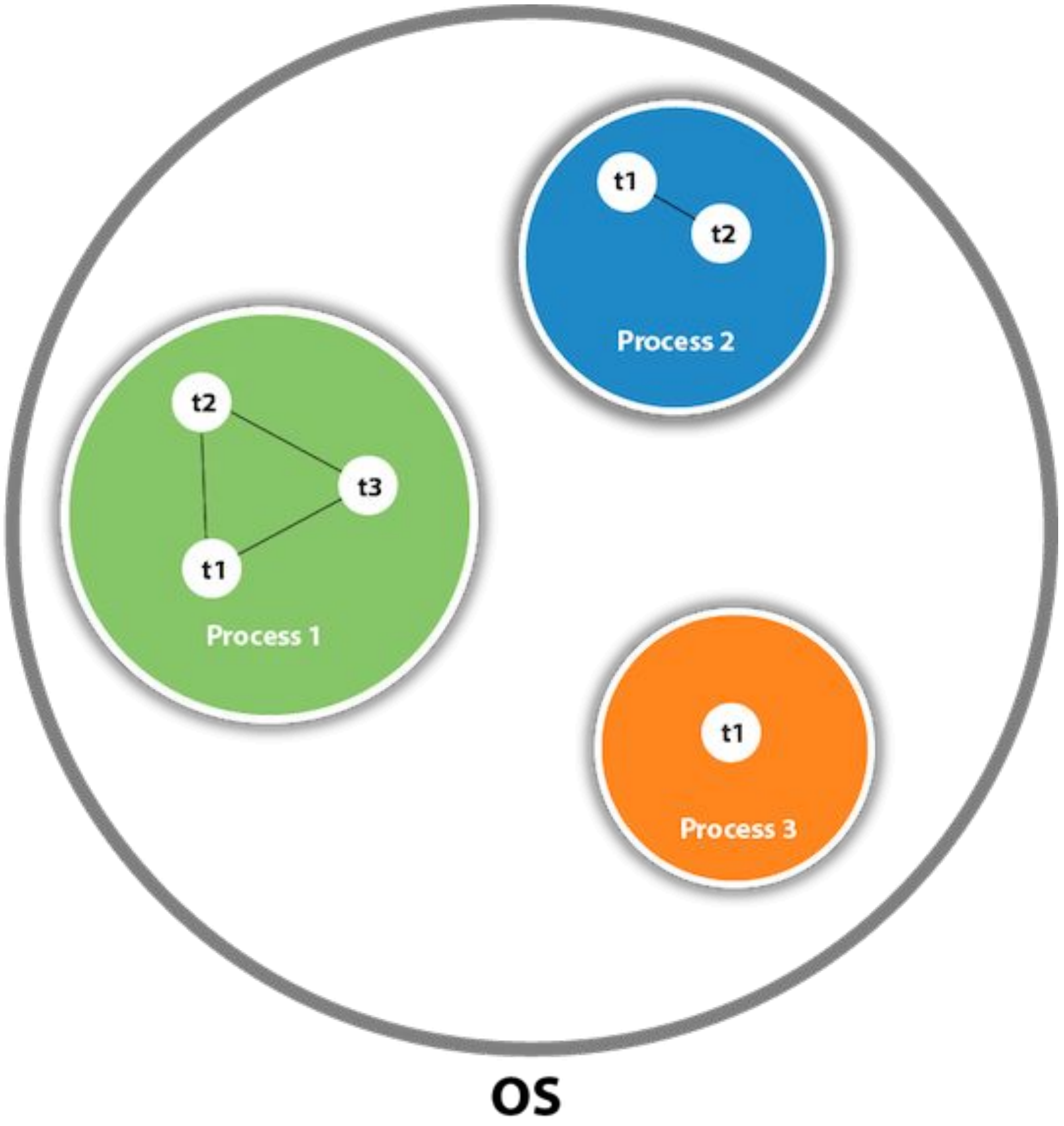
What is Thread

- **Threads** have same properties as of the process so they are called as light weight processes.
- **Threads** are executed one after another but gives the illusion as if they are executing in parallel.



- Threads are independent because they all have separate path of execution.
- All threads of a process share the common memory.
- **The process of executing multiple threads simultaneously is known as multithreading.**

- Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

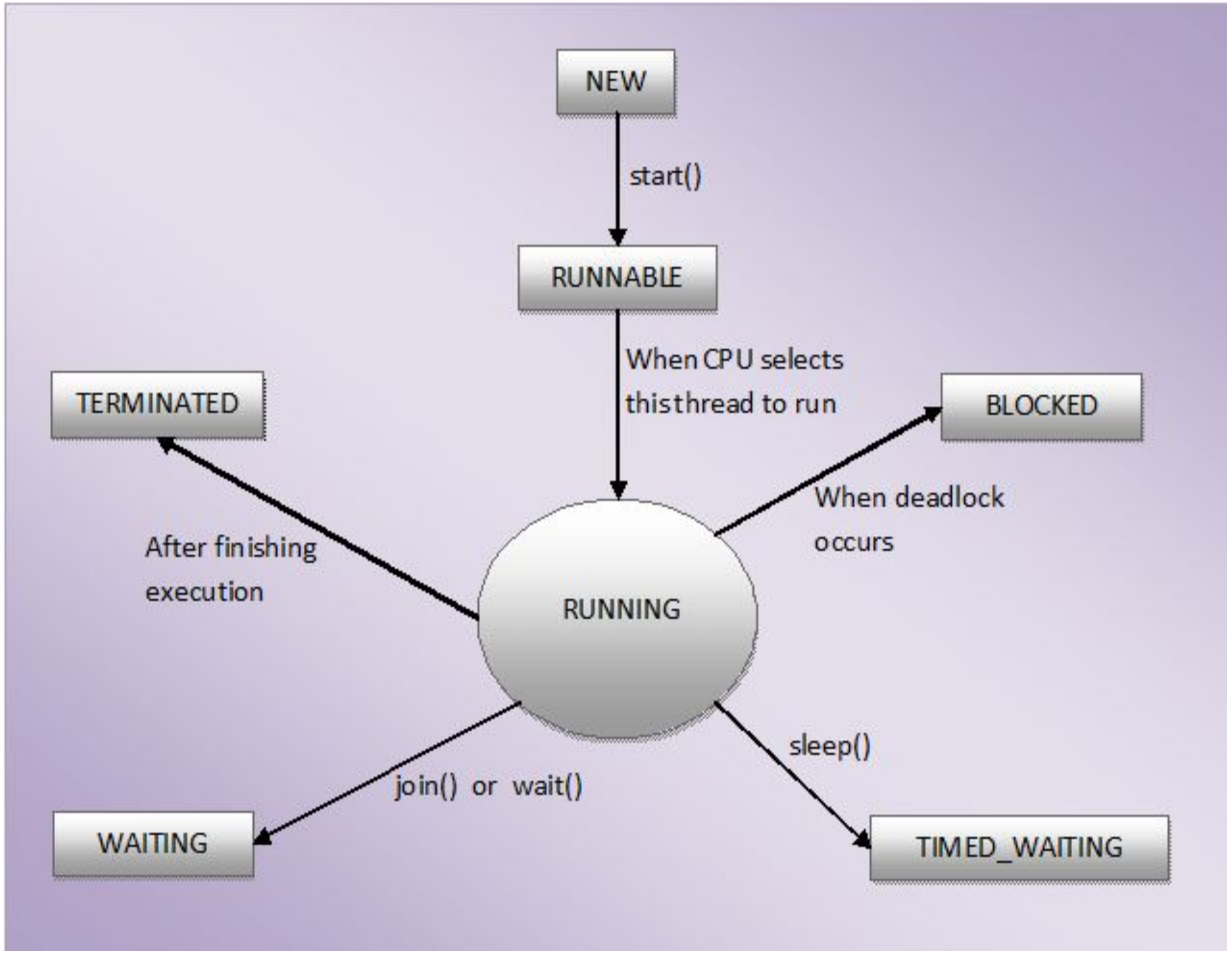

t1
t2
Process 2
t2
t3
t1
Process 1
t1
Process 3
OS

Let’s summarize the discussion in points:

1. The main purpose of multithreading is to provide simultaneous execution of two or more parts of a program to maximum utilize the CPU time.

2. Threads are lightweight sub-processes, they share the common memory space. In Multithreaded environment, programs that are benefited from multithreading, utilize the maximum CPU time so that the idle time can be kept to minimum.

Thread can be one of the following State: Life cycle of thread



1. NEW – A thread that has not yet started is in this state.

2. RUNNABLE – A thread executing in the Java virtual machine is in this state.

3. BLOCKED – A thread that is blocked waiting for a monitor lock is in this state.

4. WAITING – A thread that is waiting indefinitely for another thread to perform a particular action is in this state.

5. TIMED_WAITING – A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state

.

6. TERMINATED – A thread that has exited is in this state.
A thread can be in only one state at a given point in time.

What are Multithreading Applications?

- Web Browsers - A web browser can download any number of files and web pages (multiple tabs) at the same time and still lets you continue browsing. If a particular web page cannot be downloaded, that is not going to stop the web browser from downloading other web pages.

- Web Servers - A threaded web server handles each request with a new thread. There is a thread pool and every time a new request comes in, it is assigned to a thread from the thread pool.

- Computer Games - You have various objects like cars, humans, birds which are implemented as separate threads. Also playing the background music at the same time as playing the game is an example of multithreading.

- Text Editors - When you are typing in an editor, spell-checking, formatting of text and saving the text are done concurrently by multiple threads. The same applies for Word processors also.

Difference b/w Process & Thread

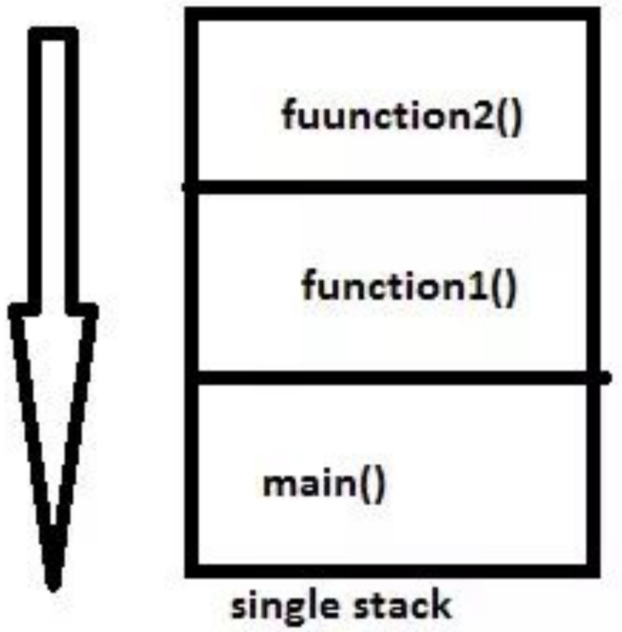
- A process consists of the memory space allocated by the operating system that can contain one or more thread.
- A thread can’t exit on its own, it must be a part of process.
- A process remains running until all of the non daemon threads are done executing.
- **Daemon** and **Non-Daemon Thread**
- **Daemons threads** are those which don't stop the JVM from exiting. Eg. A garbage collection is a daemon thread. **Non-Daemons threads** are those like the main thread , on whose exit the JVM also exits i.e the programs also finishes.

So Multithreading

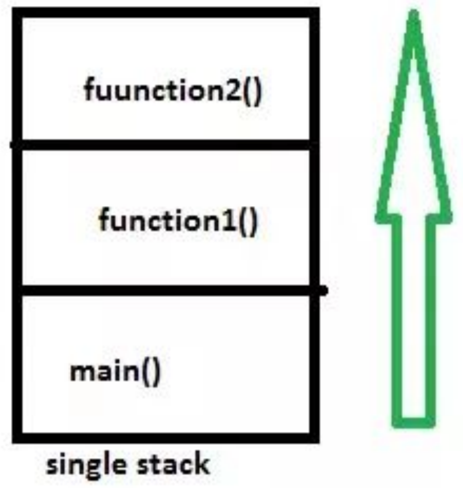
- Execute more than one thread at a time.
- Each & every thread is separate independent part of same application.
- Each thread has its own stack

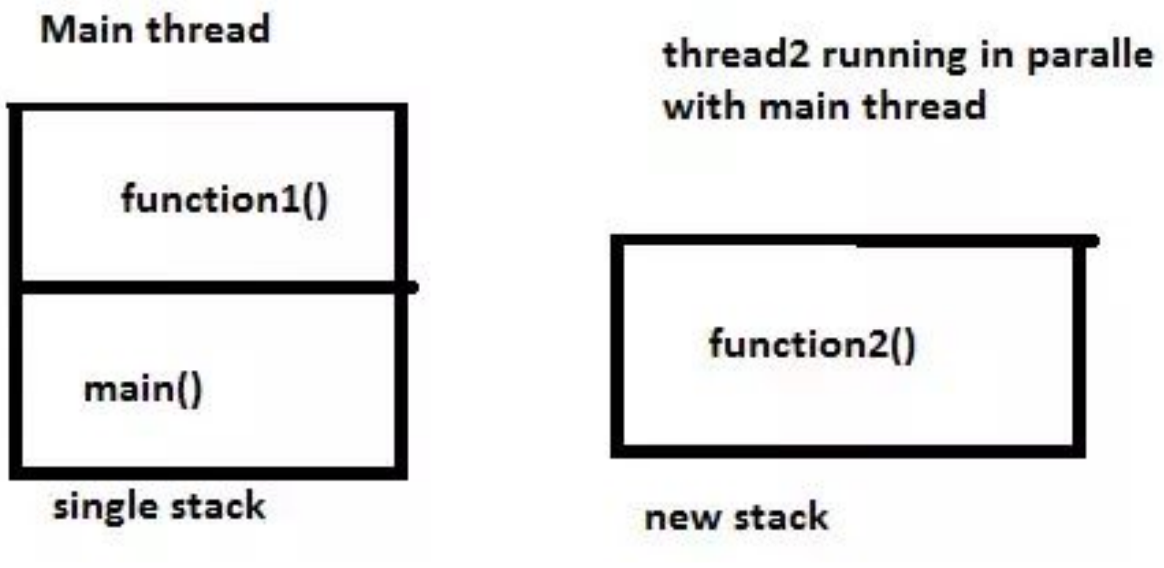
lets say we have 3 methods(normal scenario)
main() —-> funtion1() —→ function2()

```
main(
//some code
funtion1();
//some code
}
function1(){
//some code
function2();
//some code
}
```



firstly memory would be allocated to main() method where main() method variables would exist with values they are allocated during execution , then function1() and then function2() would go into stack. somewhat like this and then when function2() completes, it pops out of stack, control goes to its calling function i.e function1() , when function1() completes it pops out ,control goes to main() and finally main() ends .





invoking just run()method would produce same single stack as explained above as in normal scenario and would not achieve multithreading.
invoking start() method will trigger function2() in separate stack. achieving multithreading. where original thread and new thread would execute in parallel and independent of each other.

Creating a Thread in Java

- There are two ways in which we can create a Thread in Java

1. Creating a Thread by **extending** the **Thread class**
2. Creating a Thread by **implementing** the **Runnable interface**

Thread class & Runnable interface present in java.lang package

We will look in to each of this method with example

Thread class:

Before we begin with the programs(code) of creating threads,
let’s have a look at these methods of Thread class. We have used few of these methods in the example below.

- getName(): It is used for Obtaining a thread’s name
- getPriority(): Obtain a thread’s priority
- isAlive(): Determine if a thread is still running
- join(): Wait for a thread to terminate
- run(): Entry point for the thread
- sleep(): suspend a thread for a period of time
- start(): start a thread by calling its run() method

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Runnable interface:

- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().
- **public void run():** is used to perform action for a thread.
- Starting a thread:

- **start() method** of Thread class is used to start a newly created thread. It performs following tasks:
- A new thread starts
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

1) By Extending Thread Class

Step-1 create a new class that **extends Thread** class

Step-2 extending class must override the **run()** method, which is the entry point for the new thread.

Step-3 create an **instance of Thread** class

Step-4 call **start()** method to begin execution of the new thread.

1) Java Thread Example by extending Thread class

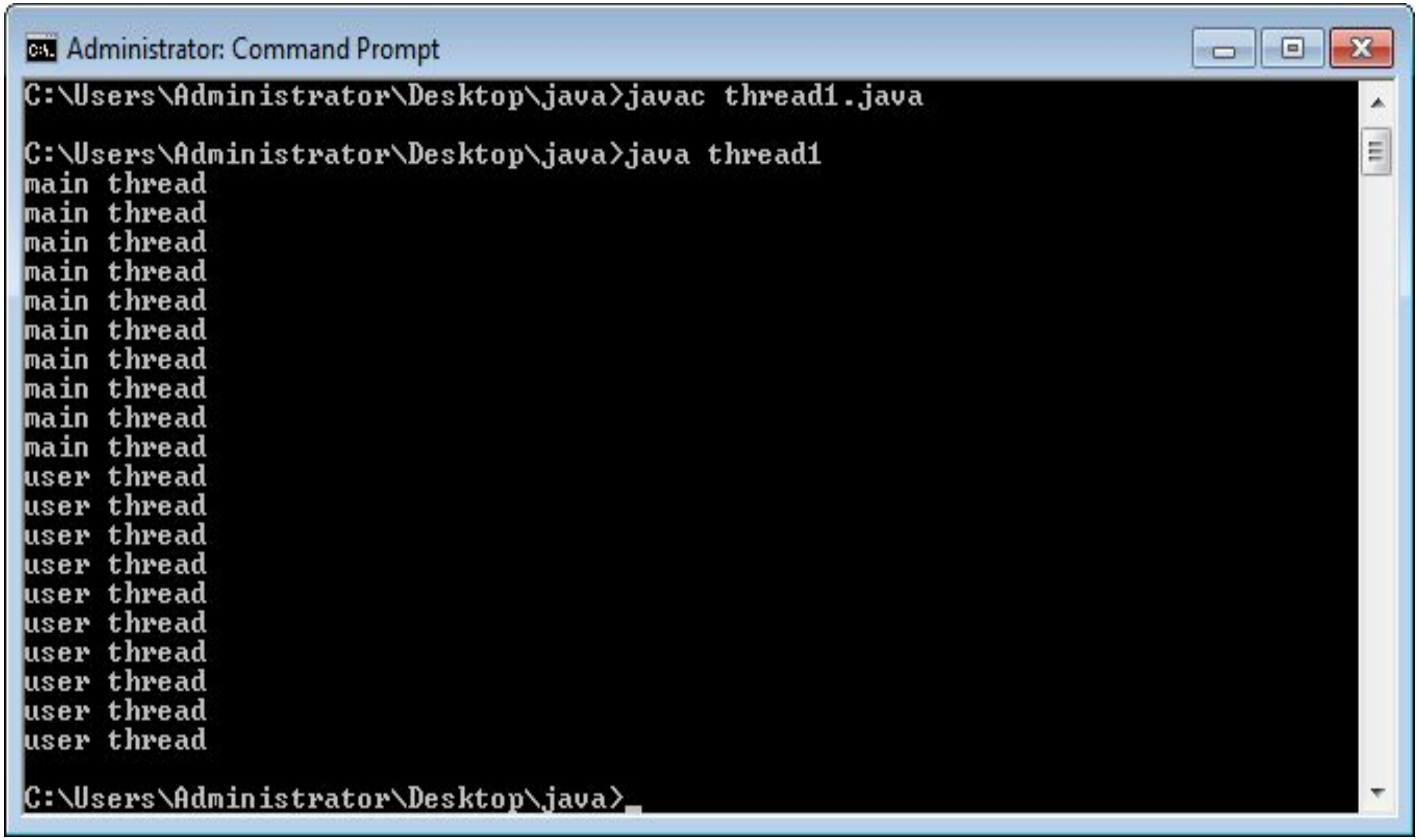
```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
}
}
```

Output:thread is running...

Example

```
class MyThread extends Thread
{
  public void run()
  {
    for(int i=0; i<10; i++)
    {
   System.out.println("user thread");
    }
  }
}
```

```
class thread1
{
      public static void main(String[] args) {
            MyThread t = new MyThread();
            t.start();
        for(int i=0; i<10; i++)
        {
         System.out.println(“main thread");
        }
      }
}
```



```
C:\Users\Administrator\Desktop\java>javac thread1.java

C:\Users\Administrator\Desktop\java>java thread1
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread

C:\Users\Administrator\Desktop\java>
```

How many thread will be created?

2 thread(As soon as the main method is called one thread starts running)

main thread created first

main thread will create another thread

Every thread have separate **stack memory**

t.start()

- At this line, application contain two thread
- main thread & user thread
- Which thread will execute first?

- Thread execution decided by **Thread scheduler**, one of the component of JVM

What happen during the execution?

- Where the start() method define?

- JVM will find start() method in **Thread** class

- Thread class start() method perform two action

1. Thread is registered to thread scheduler then thread will be created
2. Thread class start() method automatically call **run()** method

Output

```
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
user thread
```

Example-2 create another thread

```
class MyThread extends Thread
{
  public void run()
  {
    for(int i=0; i<4; i++)
    {
   System.out.println("user thread");
    }
  }
}
```

```
class ThreadDemo
{
        public static void main(String[] args) {
                MyThread t1 = new MyThread();
                t1.start();
                                MyThread t2= new MyThread();
                t2.start();

          for(int i=0; i<4; i++)
          {
           System.out.println(“main thread");
          }
        }
}
```

How many thread will be created?

3 thread

main thread created first

main thread will create **2 more thread**

Every thread have separate **stack memory**

This is the example that multiple thread performing single task

Output

```
user thread              main thread
user thread              main thread
user thread              main thread
user thread              main thread
main thread       OR     user thread
main thread              user thread
main thread              user thread
main thread              user thread
user thread              user thread
user thread              user thread
user thread              user thread
user thread              user thread
```

Can we start a thread twice

No. 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.
Let's understand it by the example given below:

```
public class TestThreadTwice1 extends Thread{
public void run(){
  System.out.println("running...");
 }
 public static void main(String args[]){
 TestThreadTwice1 t1=new TestThreadTwice1();
  t1.start();
  t1.start();
 }
}
```

Ouput:running…

```
running Exception in thread "main" java.lang.IllegalThreadStateException
```

Sleep method in java

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

Syntax of sleep() method in java

The Thread class provides two methods for sleeping a thread:

- public static void sleep(long miliseconds)throws InterruptedException
- public static void sleep(long miliseconds, int nanos)throws InterruptedException

Example of sleep method in java

```
class TestSleepMethod1 extends Thread{
public void run(){
 for(int i=1;i<5;i++){
   try{Thread.sleep(500);}
catch(InterruptedException e){System.out.println(e);}
   System.out.println(i);
  }
 }
public static void main(String args[]){
TestSleepMethod1 t1=new TestSleepMethod1();
TestSleepMethod1 t2=new TestSleepMethod1();
      t1.start();
       t2.start();
 }
}
```

Output:

```
1 1 2 2 3 3 4 4
```

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on.

Example-3

```
class MyThread extends Thread
{
  public void run()
  {
   for(int i=0; i<10; i++)
   {
  System.out.println("call No = "+i);
       try{
            sleep(1000);
       }catch(InterruptedException e){
           e.printStackTrace();
     }
   }}}
```

```
class ThreadDemo
{
     public static void main(String[] args) {
          MyThread t = new MyThread();
          t.start();
      for(int i=0; i<10; i++)
      {
       System.out.println(“main thread");
      }
    }
}
```

Output

```
call No = 0
main thread
main thread
main thread
main thread
call No = 1
call No = 2
call No = 3
call No = 4
call No = 5
call No = 6
call No = 7
call No = 8
call No = 9
```

Example-4

```
class MyThread extends Thread
{
 public void run()
 {
  for(int i=0; i<5; i++)
  {
 System.out.println("call No = "+i);
       try{
              sleep(1000);
       }catch(InterruptedException e){
            e.printStackTrace();
   }
  }}}
```

```
class ThreadDemo
{
      public static void main(String[] args) {
            MyThread t1 = new MyThread();
            t1.start();
                  MyThread t2 = new MyThread();
            t2.start();
              for(int i=0; i<4; i++)
        {
         System.out.println(“main thread");
        }
      }
}
```

Output

```
main thread
main thread
main thread
main thread
call No = 0
call No = 0
call No = 1
call No = 1
call No = 2
call No = 2
call No = 3
call No = 3
call No = 4
call No = 4
```

OR

```
call No = 0
main thread
main thread
main thread
main thread
call No = 0
call No = 1
call No = 1
call No = 2
call No = 2
call No = 3
call No = 3
call No = 4
call No = 4
```

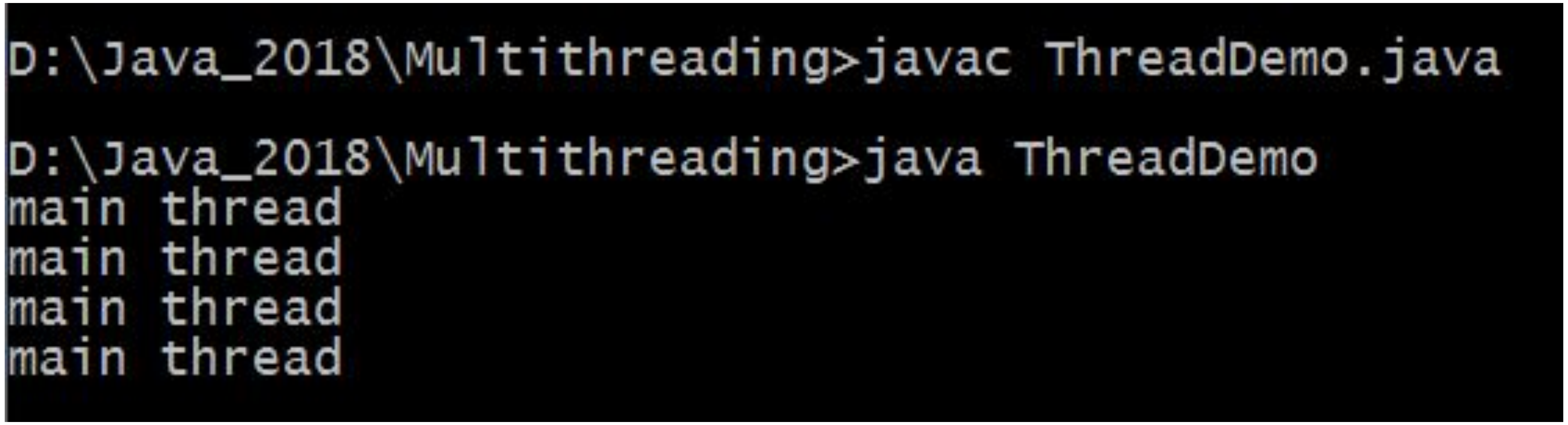
Example- MyThread class without run() method

```
class MyThread extends Thread
{
}
```

Will it compile or not?

```
class ThreadDemo
{
      public static void main(String[] args) {
            MyThread t = new MyThread();
            t.start();
       for(int i=0; i<4; i++)
        {
         System.out.println(“main thread");
        }
      }
}
```

Output-



```
D:\Java_2018\Multithreading>javac ThreadDemo.java

D:\Java_2018\Multithreading>java ThreadDemo
main thread
main thread
main thread
main thread
```

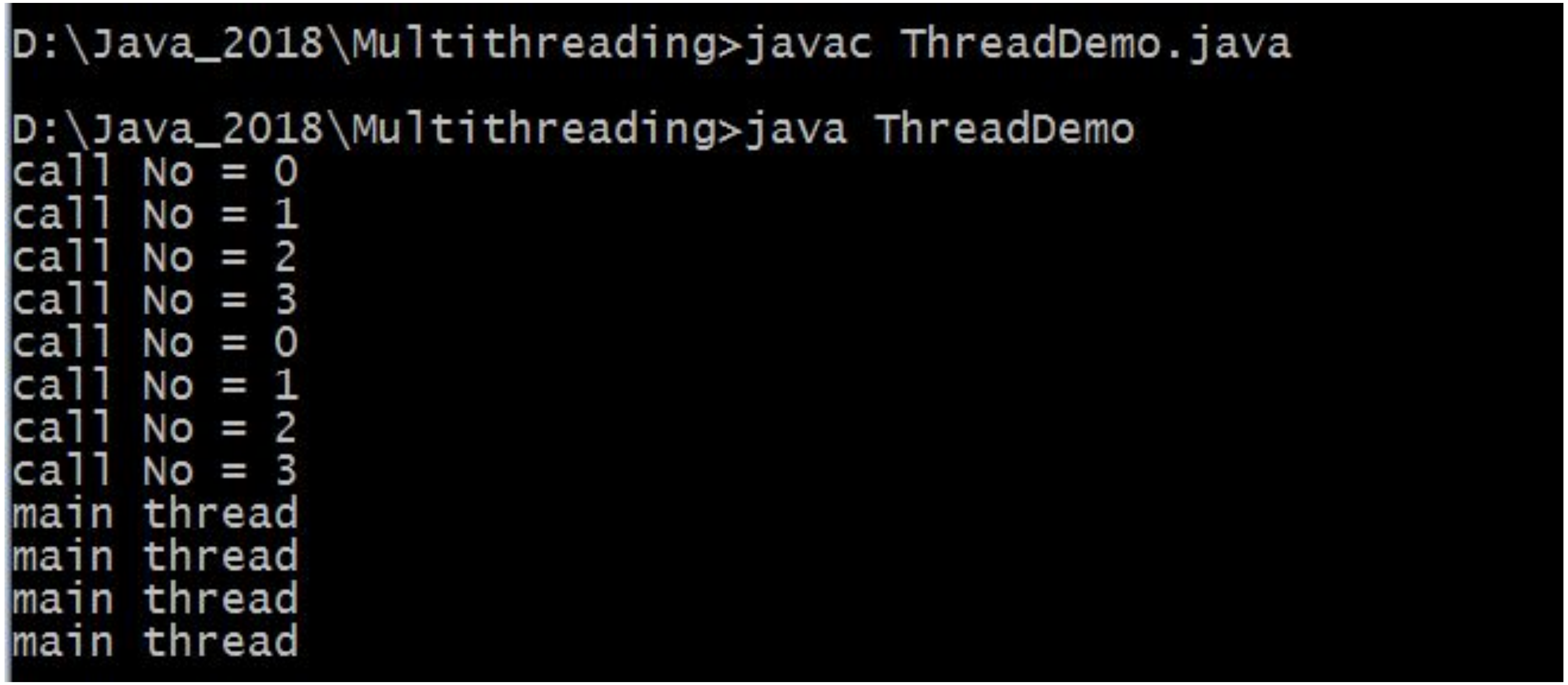
Thread class run() method will call with empty implementation- not recommended

Can we override the start() method?

```
class MyThread extends Thread
{
  public void start()
  {
    for(int i=0; i<4; i++)
    {
   System.out.println("call No = "+i);
          try{
                sleep(1000);
          }catch(InterruptedException e){
               e.printStackTrace();
          }   }
  }
}
```

```
class ThreadDemo
{
        public static void main(String[] args) {
                MyThread t1 = new MyThread();
                t1.start();
                                MyThread t2 = new MyThread();
                t2.start();

          for(int i=0; i<4; i++)
          {
           System.out.println(“main thread");
          }
        }
}
```

Output-



```
D:\Java_2018\Multithreading>javac ThreadDemo.java

D:\Java_2018\Multithreading>java ThreadDemo
call No = 0
call No = 1
call No = 2
call No = 3
call No = 0
call No = 1
call No = 2
call No = 3
main thread
main thread
main thread
main thread
```

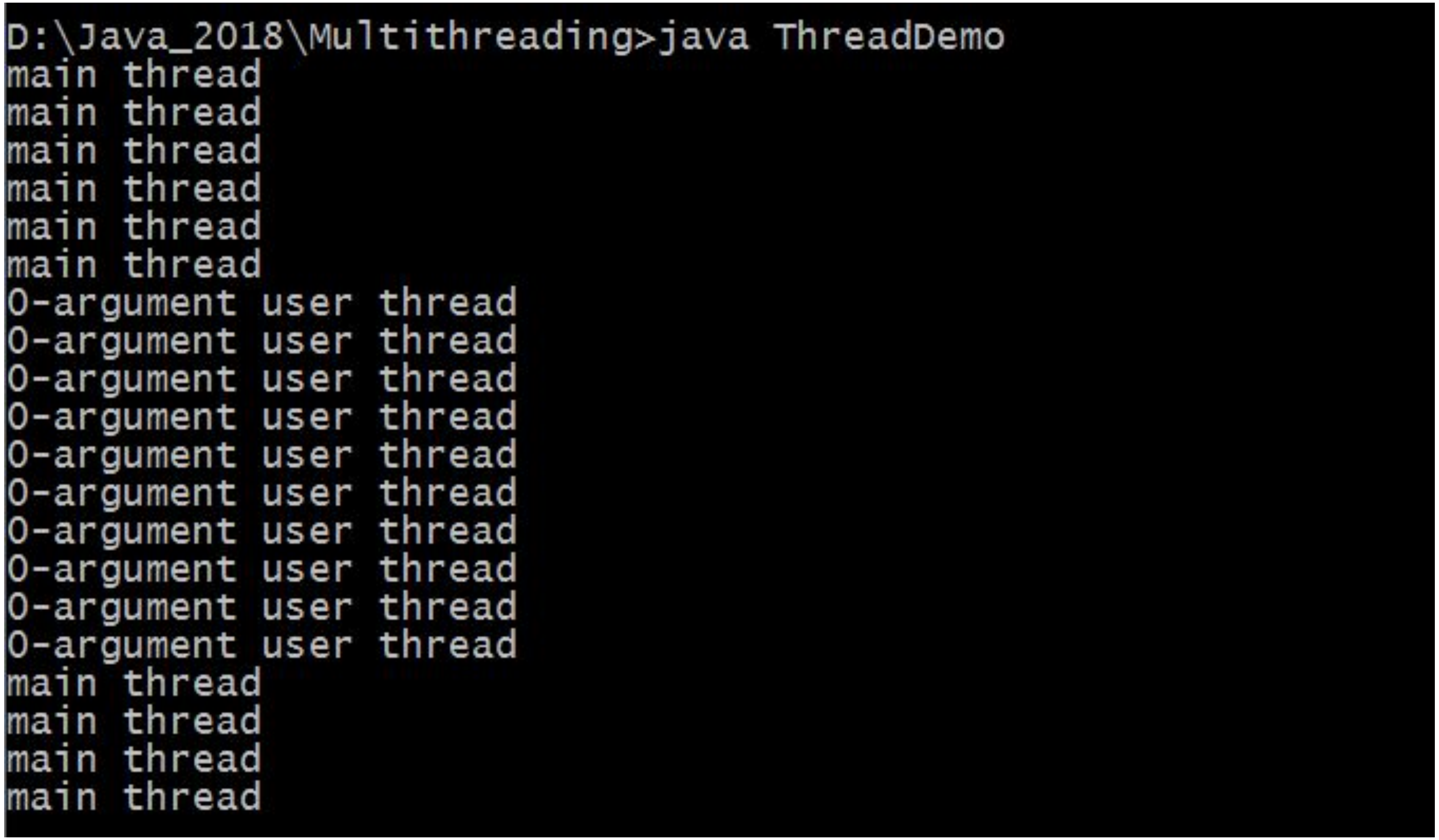
Overriding start() method but Thread will not created – means Thread class will not used

Can we overload run() method?

```
class MyThread extends Thread
{
  public void run()
  {
    for(int i=0; i<10; i++)
    {
   System.out.println("0-argument user thread");
    }
  }
public void run(int a)
  {
    for(int i=0; i<4; i++)
    {
   System.out.println("1-argument user thread");
    }  }}
```

```
class ThreadDemo
{
      public static void main(String[] args) {

                MyThread t = new MyThread();
            t.start();

          for(int i=0; i<10; i++)
          {
           System.out.println("main thread");
          }
        }
}
```

Output-



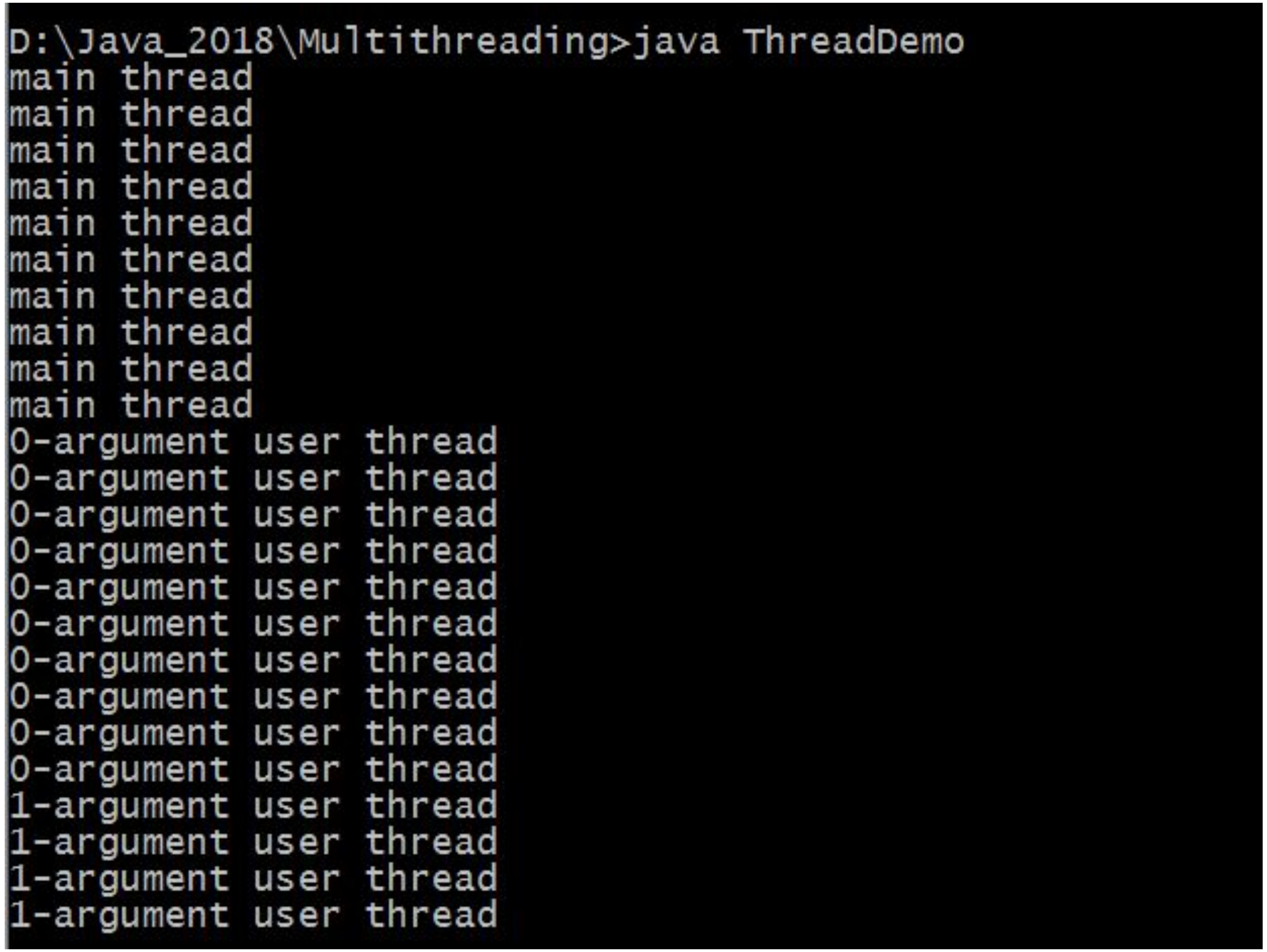
```
D:\Java_2018\Multithreading>java ThreadDemo
main thread
main thread
main thread
main thread
main thread
main thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
main thread
main thread
main thread
main thread
```

We can overload run() method but JVM always call 0-argument run() method

How to call overloaded run(int a) method?

```
public void run()
 {
   for(int i=0; i<10; i++)
   {
  System.out.println(“0-argument user thread");
   }
   run(100);
 }
```

Output-



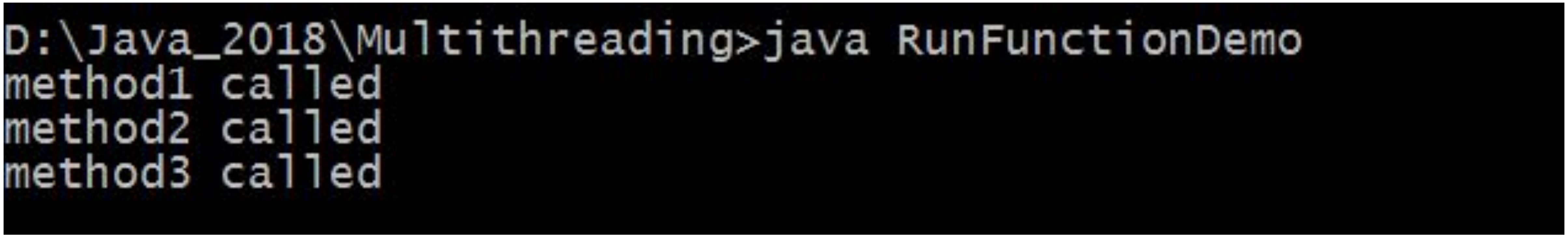
```
D:\Java_2018\Multithreading>java ThreadDemo
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
0-argument user thread
1-argument user thread
1-argument user thread
1-argument user thread
1-argument user thread
```

How to write multiple functionality in run() method?

```
class MyThread extends Thread
{
  public void run()
  {   method1();
      method2();
      method3();
  }
  void method1(){    S.O.P("method1 called");    }
   void method2(){   SO.P("method2 called");    }
   void method3(){   S.O.P.("method3 called");    }
}
```

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

Output-



```
D:\Java_2018\Multithreading>java RunFunctionDemo
method1 called
method2 called
method3 called
```

Create different thread & different task

```
class MyThreadOne extends Thread
{
  public void run()
  {    System.out.println("Thread one");   }
}
class MyThreadTwo extends Thread
{
  public void run()
  {    System.out.println("Thread two");     }
}
class MyThreadThree extends Thread
{
  public void run()
  {    System.out.println("Thread three");     }
}
```

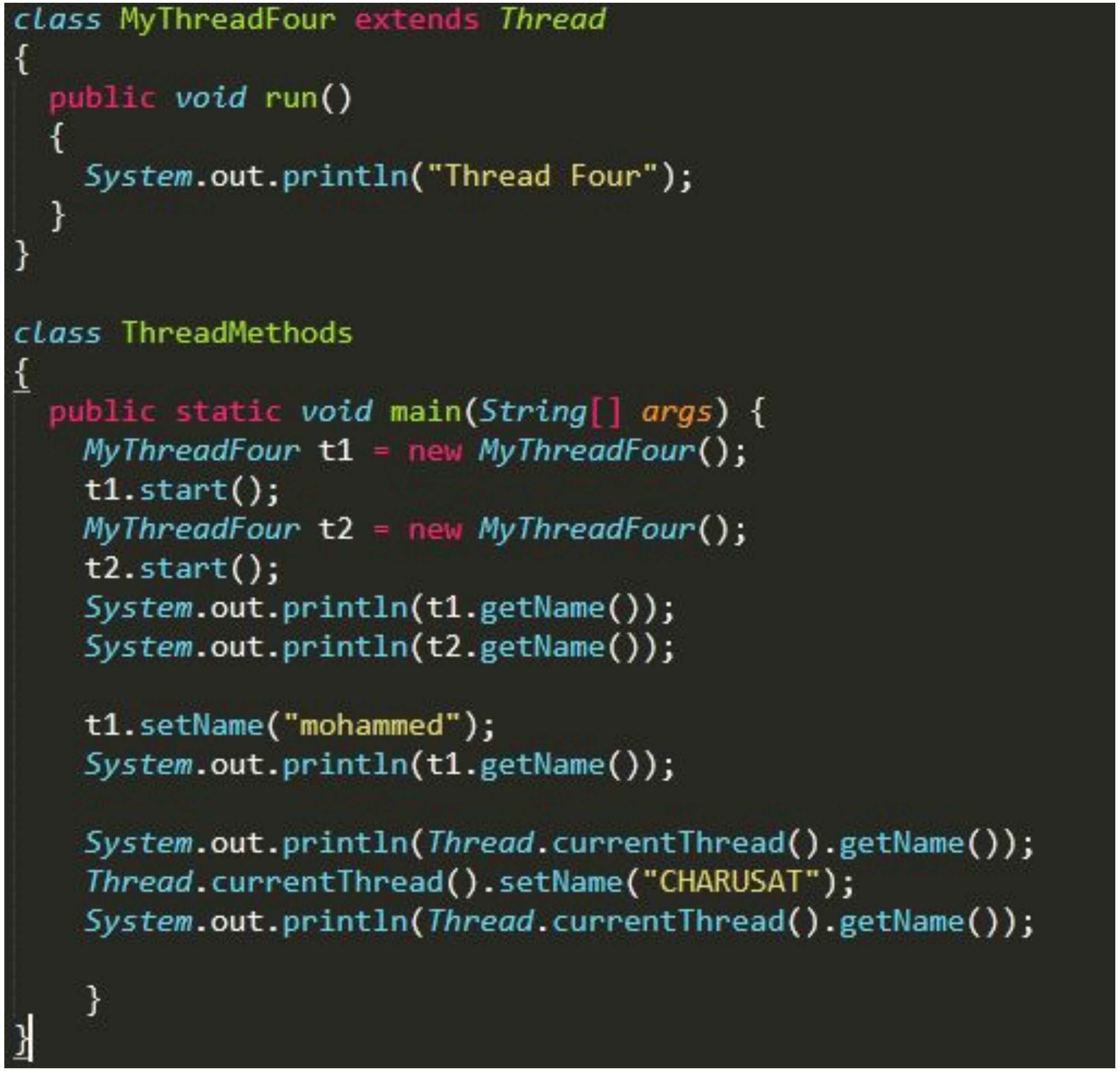
```
class MultiThreadDemo
{
  public static void main(String[] args)
{
    new MyThreadOne().start();
    new MyThreadTwo().start();
    new MyThreadThree().start();

 }
}
```

How many thread created here?
4

Thread Methods

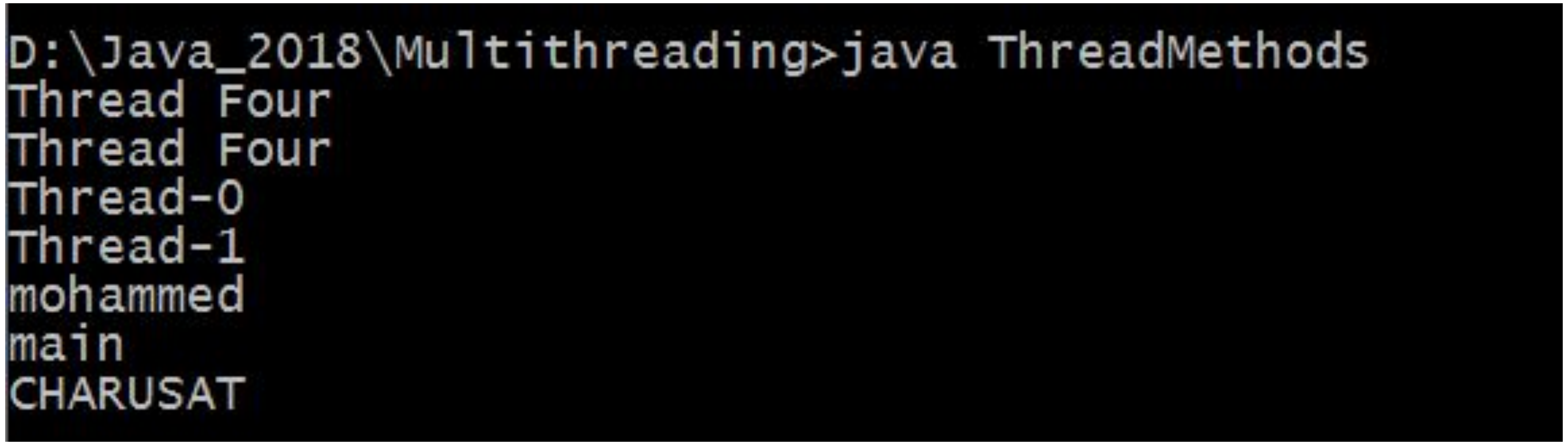
1. setName(String)
2. getName()
3. CurrentThread()
4. getId()
5. isAlive()
6. activeCount()
7. setPriority(int)
8. getPriority()
9. setDaemon(boolean)
10. isDaemon()
11. join()
12. join(long)
13. interrupt()

Method Example



```java
class MyThreadFour extends Thread
{
  public void run()
  {
    System.out.println("Thread Four");
  }
}

class ThreadMethods
{
  public static void main(String[] args) {
    MyThreadFour t1 = new MyThreadFour();
    t1.start();
    MyThreadFour t2 = new MyThreadFour();
    t2.start();
    System.out.println(t1.getName());
    System.out.println(t2.getName());

    t1.setName("mohammed");
    System.out.println(t1.getName());

    System.out.println(Thread.currentThread().getName());
    Thread.currentThread().setName("CHARUSAT");
    System.out.println(Thread.currentThread().getName());

    }
}
```

Output-



```
D:\Java_2018\Multithreading>java ThreadMethods
Thread Four
Thread Four
Thread-0
Thread-1
mohammed
main
CHARUSAT
```

Daemon Thread

- Background thread
- Thread which is working on background called daemon thread.
- Daemon thread provide support to foreground thread
- Low priority thread
- Its life depend on user thread

- The sole purpose of the daemon thread is that it provides services to user thread for background supporting task.
- If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

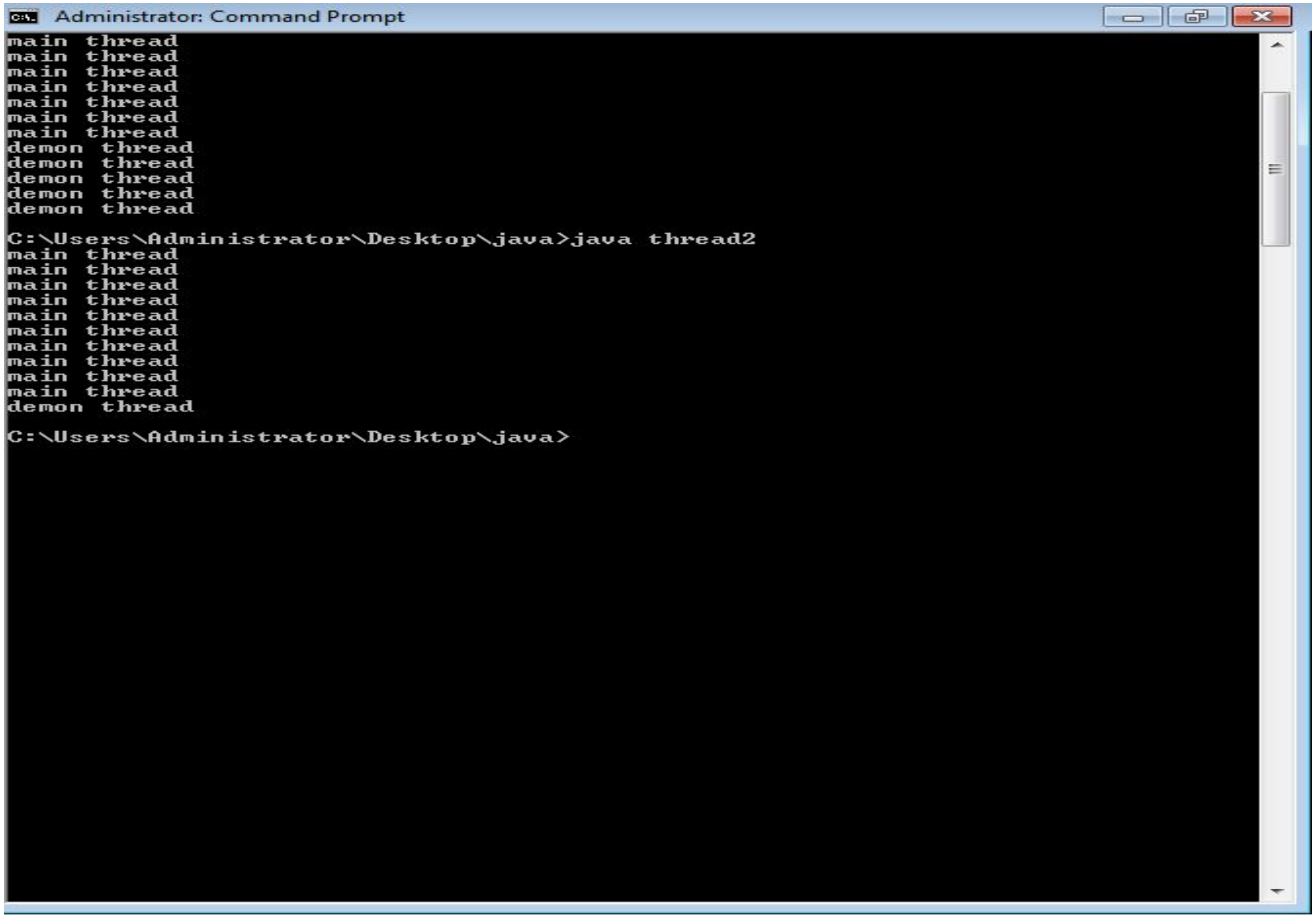
Few points about daemon thread

- Once main thread is completed, daemon thread is terminated whether its completed or not.

- main thread never wait, daemon thread

```
class MyThread extends Thread
{
  public void run()
  {
   for(int i=0; i<10; i++)
{
   System.out.println("demon
thread");
 }
}
}
```

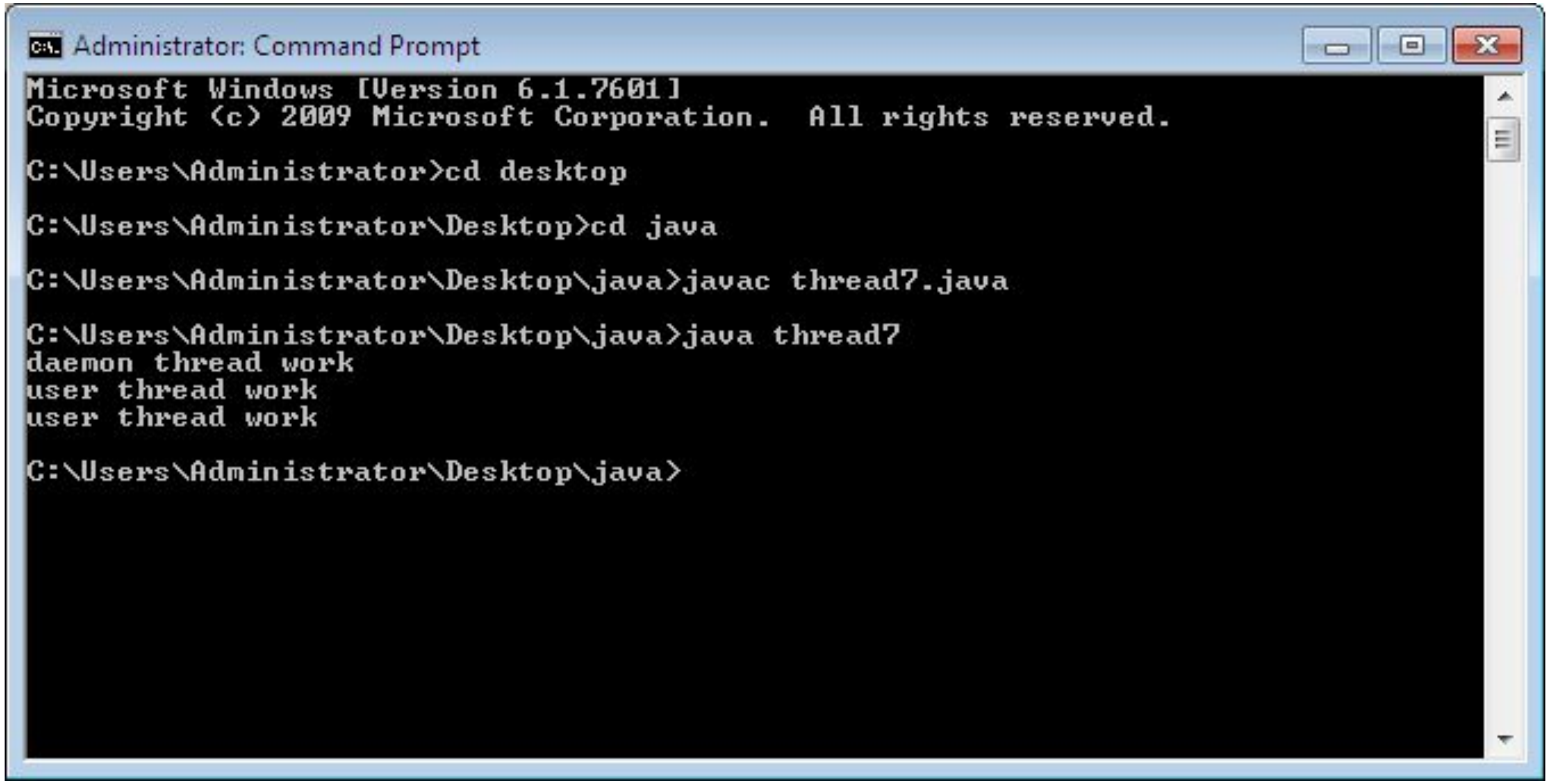
```
class thread2
{
public static void main(String[] args)
 {
                MyThread t = new MyThread();
                t.setDaemon(true);
                t.start();
          for(int i=0; i<10; i++)
          {
           System.out.println("main thread");
          }
        }
}
```


Administrator: Command Prompt

```
main thread
main thread
main thread
main thread
main thread
main thread
main thread
demon thread
demon thread
demon thread
demon thread
demon thread

C:\Users\Administrator\Desktop\java>java thread2
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
demon thread

C:\Users\Administrator\Desktop\java>
```

```
public class TestDaemonThread1 exten
ds Thread{
 public void run(){
  if(Thread.currentThread().isDaemon()){
//checking for daemon thread
   System.out.println("daemon thread w
ork");
  }
 else{
  System.out.println("user thread work")
;
 }
 }
```

```
 public static void main(String[] args)
{
  TestDaemonThread1 t1=new TestD
aemonThread1();//creating thread
  TestDaemonThread1 t2=new TestD
aemonThread1();
  TestDaemonThread1 t3=new TestD
aemonThread1();

t1.setDaemon(true);//now t1 is dae
mon thread
 t1.start();//starting threads
  t2.start();
  t3.start();
 }
}
```


Administrator: Command Prompt

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>cd desktop

C:\Users\Administrator\Desktop>cd java

C:\Users\Administrator\Desktop\java>javac thread7.java

C:\Users\Administrator\Desktop\java>java thread7
daemon thread work
user thread work
user thread work

C:\Users\Administrator\Desktop\java>
```

- Note: If you want to make a user thread as Daemon, it must not be started otherwise it will throw IllegalThreadStateException.

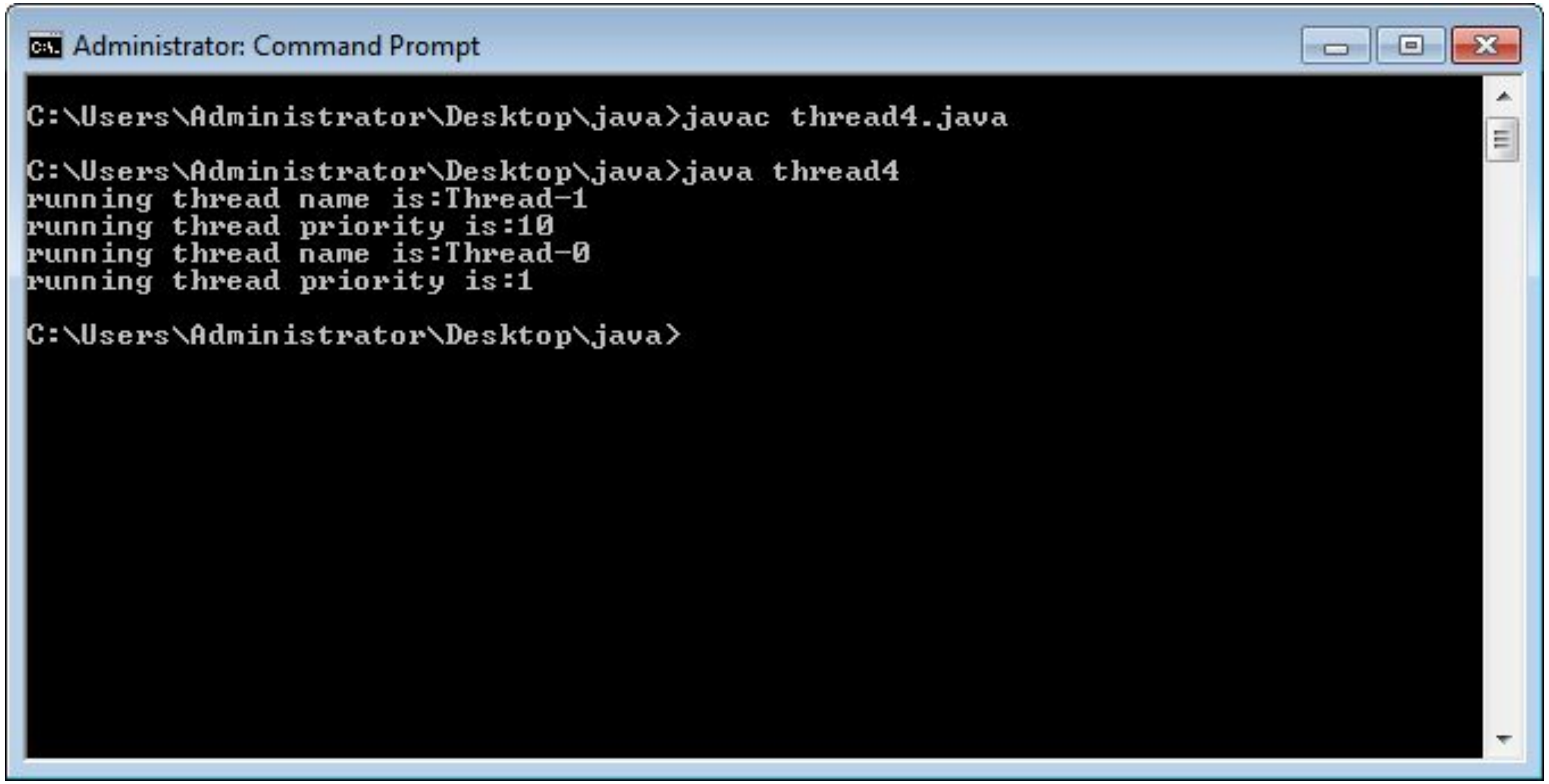
Thread Priority

- Each thread have a priority.
- Priorities are represented by number b/w 1 and 10.
- Thread scheduler will use priority to schedule the threads.
- 3 constants fields are defined in Thread class:

1. public static int MIN_PRIORITY --⍰ default is 1
2. public static int NORM_PRIORITY --⍰ default is 5
3. public static int MAX_PRIORITY --⍰ default is 10

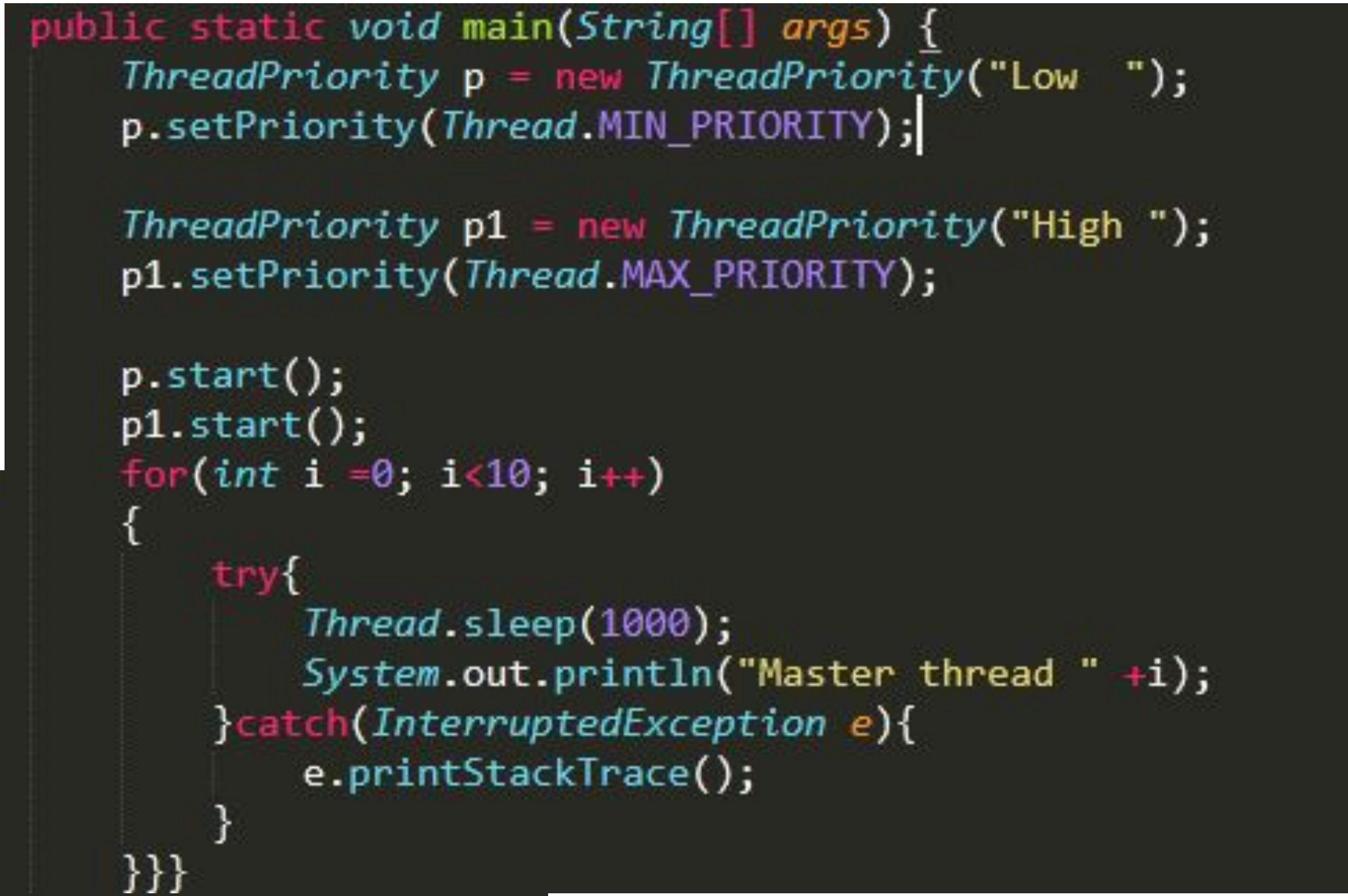
But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

```
class TestMultiPriority1 extends Thread{
 public void run(){
   System.out.println("running thread name is:"+Thread.currentThread().getName());
   System.out.println("running thread priority is:"+Thread.currentThread().getPriority());


  }

public static void main(String args[]){
  TestMultiPriority1 m1=new TestMultiPriority1();

  TestMultiPriority1 m2=new TestMultiPriority1();

  m1.setPriority(Thread.MIN_PRIORITY);
  m2.setPriority(Thread.MAX_PRIORITY);
  m1.start();
  m2.start();

 }
}
```

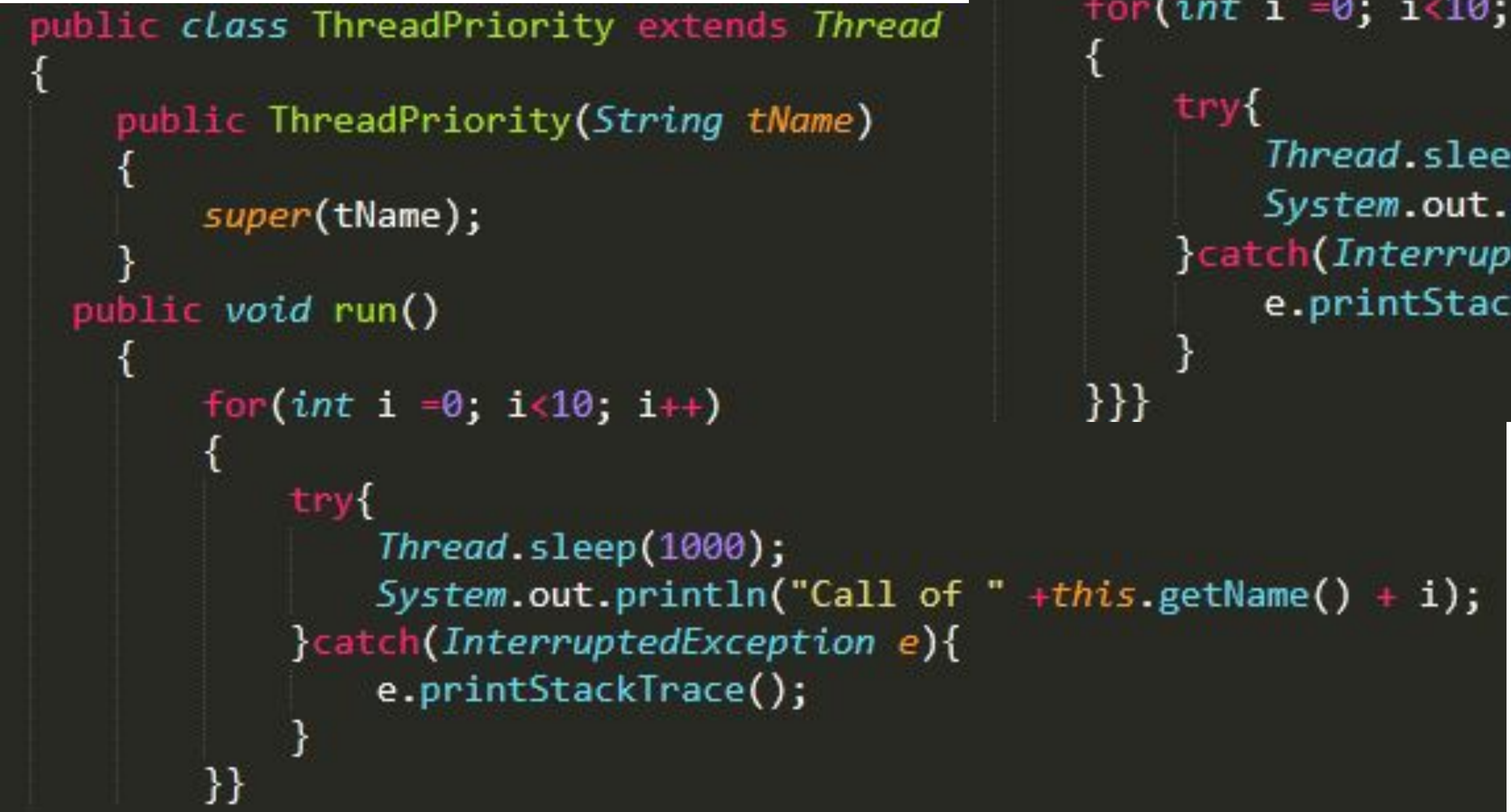

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac thread4.java

C:\Users\Administrator\Desktop\java>java thread4
running thread name is:Thread-1
running thread priority is:10
running thread name is:Thread-0
running thread priority is:1

C:\Users\Administrator\Desktop\java>
```

Example

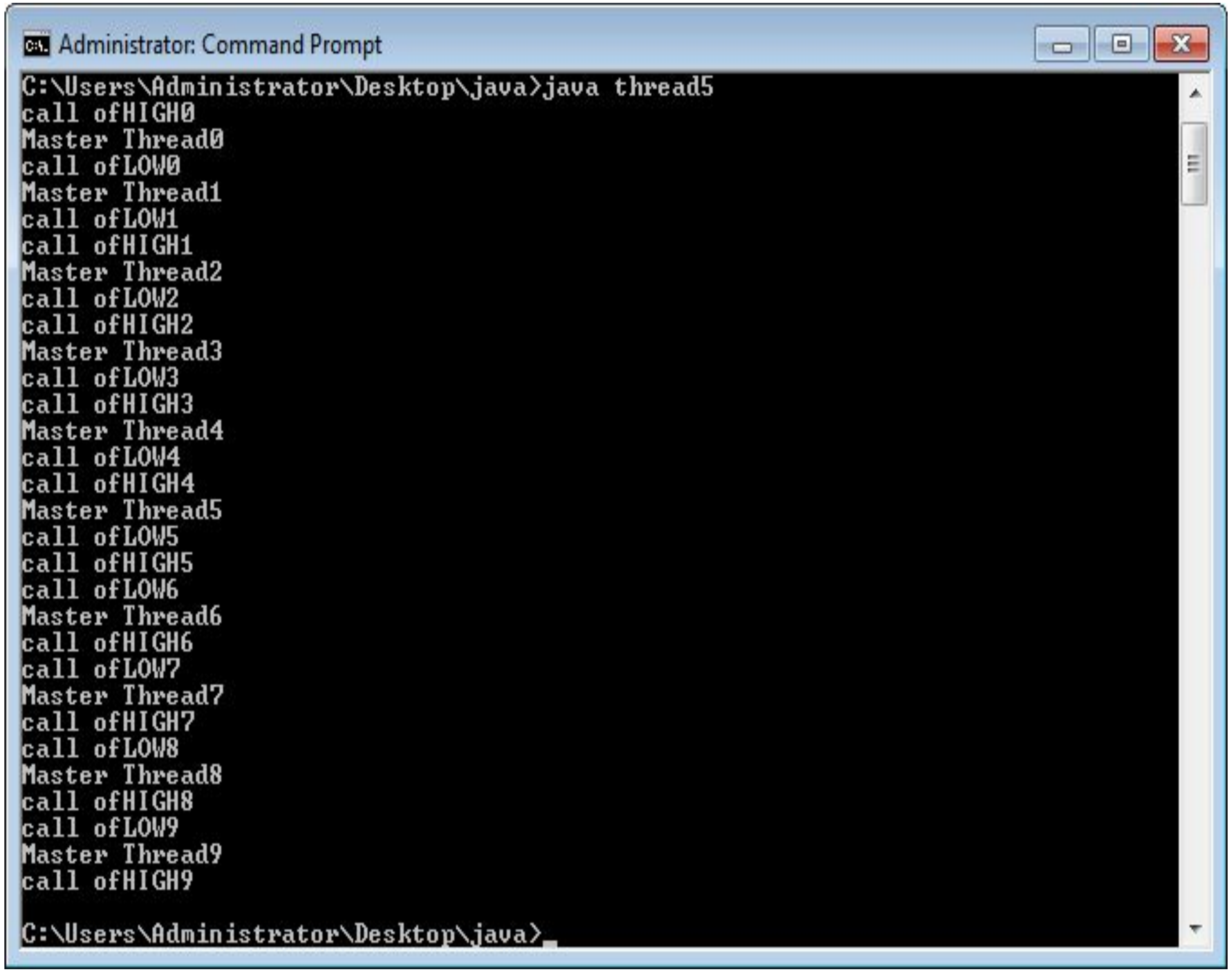


```
public static void main(String[] args) {
    ThreadPriority p = new ThreadPriority("Low  ");
    p.setPriority(Thread.MIN_PRIORITY);

    ThreadPriority p1 = new ThreadPriority("High ");
    p1.setPriority(Thread.MAX_PRIORITY);

    p.start();
    p1.start();
    for(int i =0; i<10; i++)
    {
        try{
            Thread.sleep(1000);
            System.out.println("Master thread " +i);
        }catch(InterruptedException e){
            e.printStackTrace();
        }
    }}}
```



```
public class ThreadPriority extends Thread
{
    public ThreadPriority(String tName)
    {
        super(tName);
    }
  public void run()
    {
        for(int i =0; i<10; i++)
        {
            try{
                Thread.sleep(1000);
                System.out.println("Call of " +this.getName() + i);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }}
```

Output

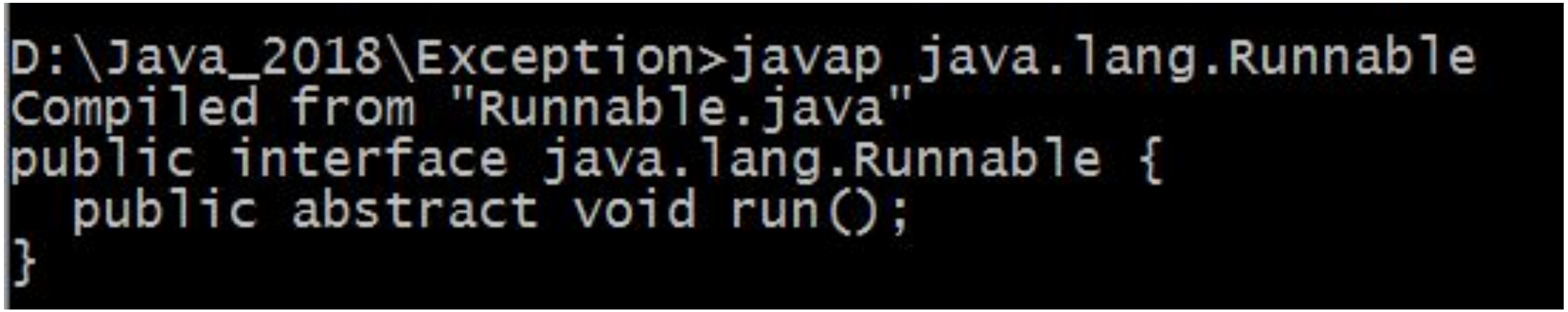


```
C:\Users\Administrator\Desktop\java>java thread5
call ofHIGH0
Master Thread0
call ofLOW0
Master Thread1
call ofLOW1
call ofHIGH1
Master Thread2
call ofLOW2
call ofHIGH2
Master Thread3
call ofLOW3
call ofHIGH3
Master Thread4
call ofLOW4
call ofHIGH4
Master Thread5
call ofLOW5
call ofHIGH5
call ofLOW6
Master Thread6
call ofHIGH6
call ofLOW7
Master Thread7
call ofHIGH7
call ofLOW8
Master Thread8
call ofHIGH8
call ofLOW9
Master Thread9
call ofHIGH9

C:\Users\Administrator\Desktop\java>_
```

2) By Implementing Runnable Interface

- The easiest way to create a thread is to create a class that implements the Runnable interface.
- To implement Runnable, a class need to implement only a single method called run().
- After implementing Runnable, create **an object of class**
- Call the Thread class constructor **Thread(Runnable r, String name)**

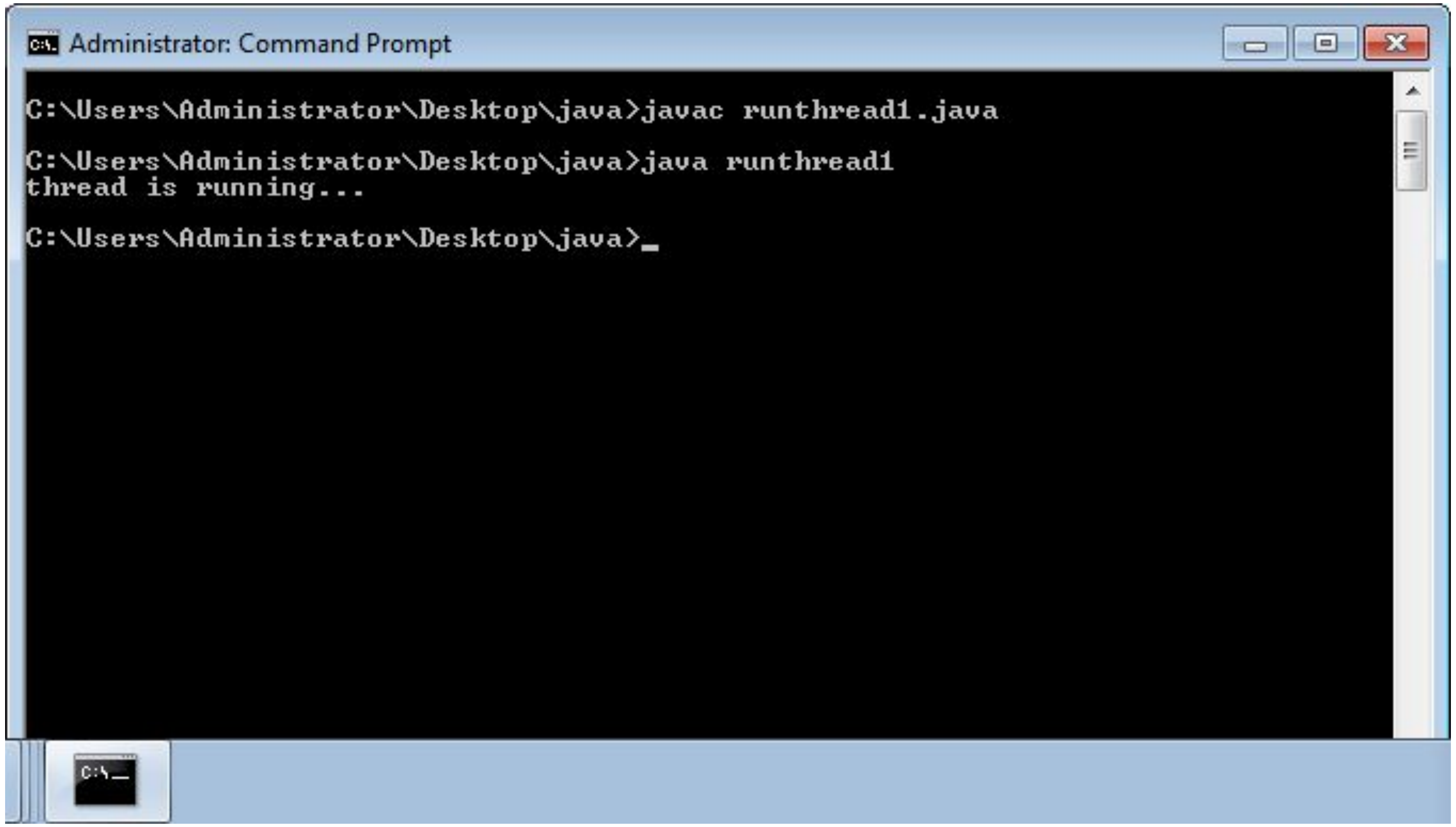
Check Runnable Interface in lang package



```
D:\Java_2018\Exception>javap java.lang.Runnable
Compiled from "Runnable.java"
public interface java.lang.Runnable {
  public abstract void run();
}
```

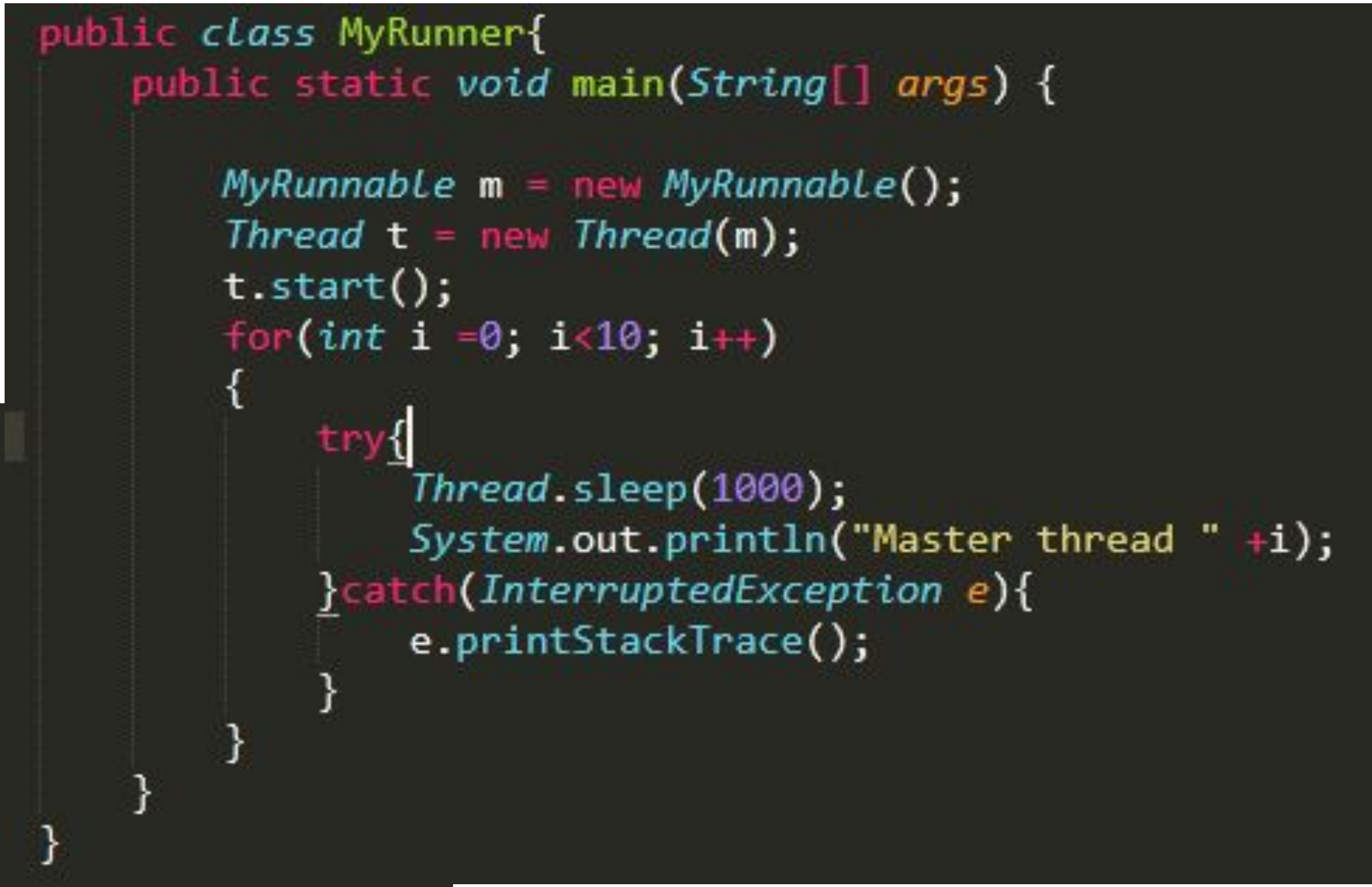
Example

```
class RunnableThread implements Runnable{
public void run(){
    System.out.println("thread is running...");
}
 public static void main(String args[]){
     RunnableThread m1=new RunnableThread();
     Thread t1 =new Thread(m1);
     t1.start();
  }
 }
```

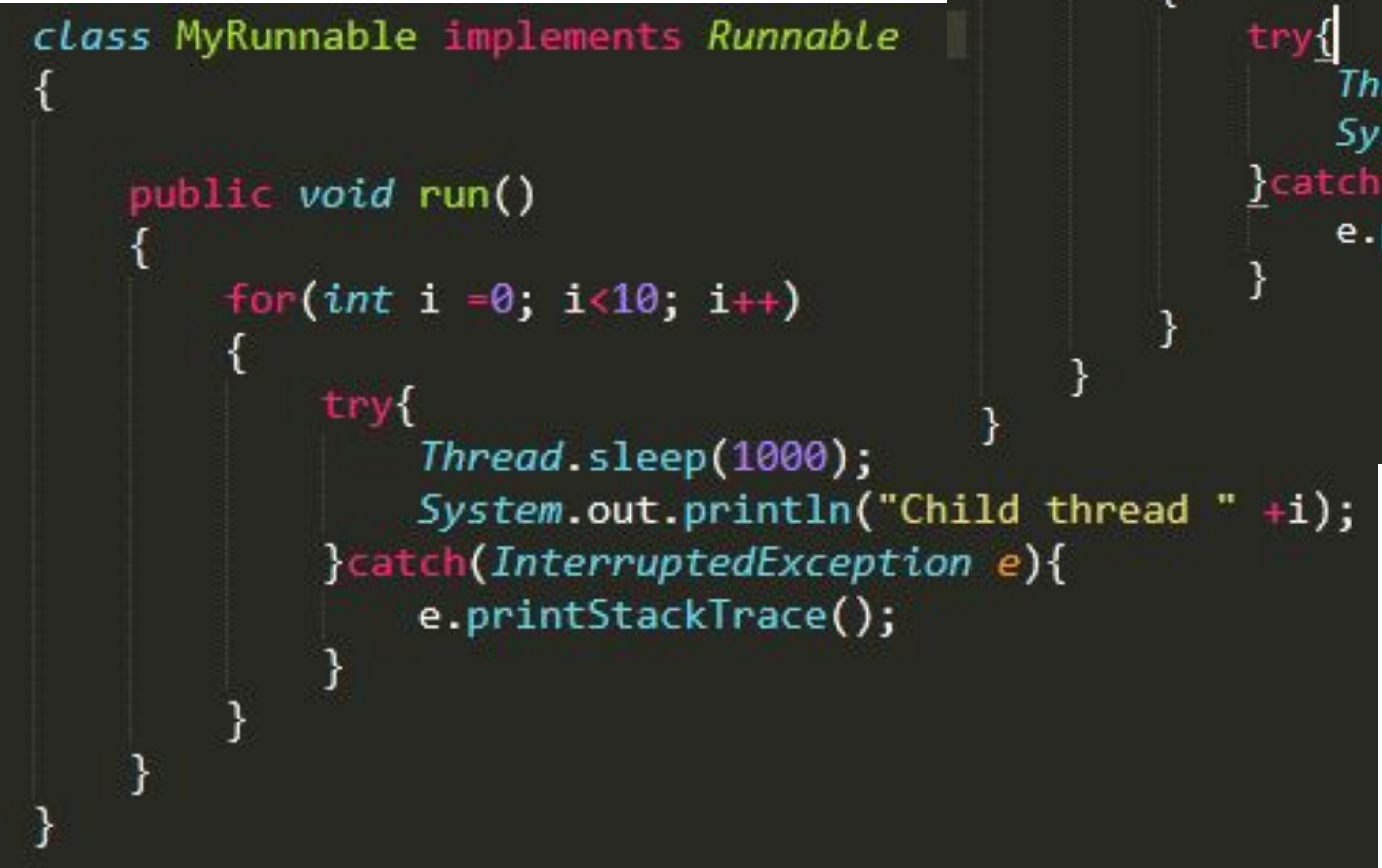


```
C:\Users\Administrator\Desktop\java>javac runthread1.java

C:\Users\Administrator\Desktop\java>java runthread1
thread is running...

C:\Users\Administrator\Desktop\java>_
```

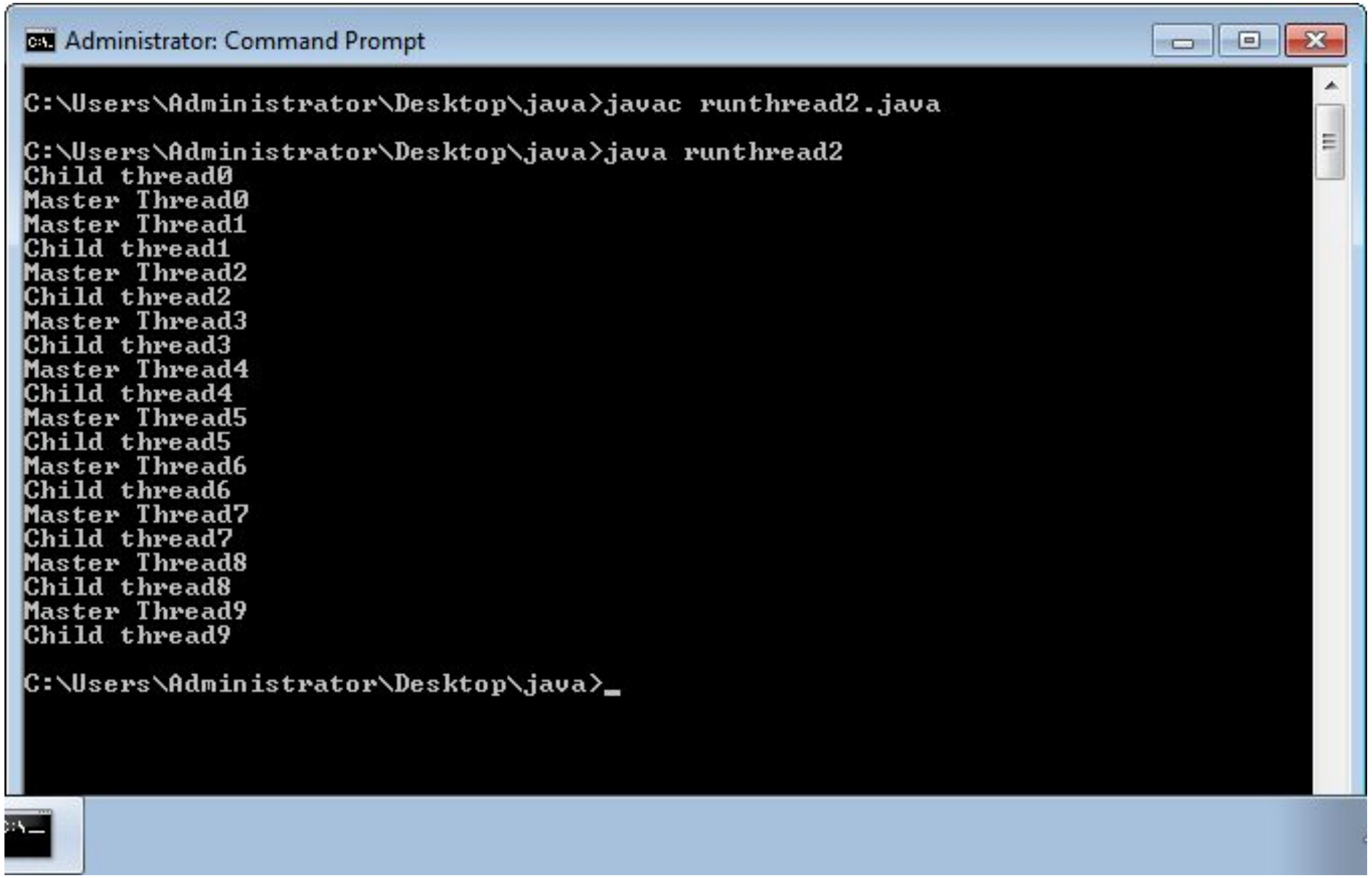
Example-2



```
public class MyRunner{
    public static void main(String[] args) {

        MyRunnable m = new MyRunnable();
        Thread t = new Thread(m);
        t.start();
        for(int i =0; i<10; i++)
        {
            try{
                Thread.sleep(1000);
                System.out.println("Master thread " +i);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```



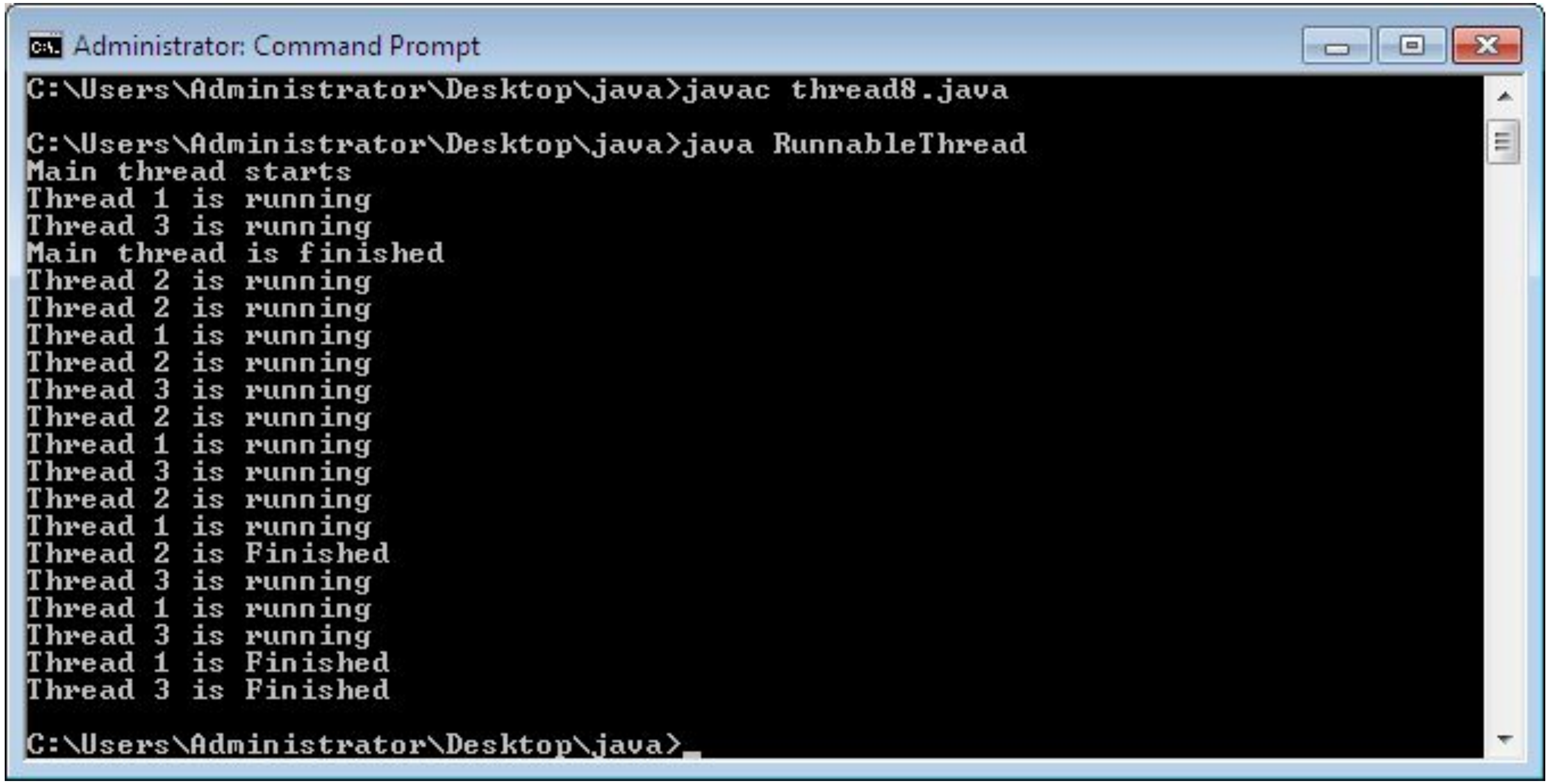
```
class MyRunnable implements Runnable
{

    public void run()
    {
        for(int i =0; i<10; i++)
        {
            try{
                Thread.sleep(1000);
                System.out.println("Child thread " +i);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```


Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac runthread2.java

C:\Users\Administrator\Desktop\java>java runthread2
Child thread0
Master Thread0
Master Thread1
Child thread1
Master Thread2
Child thread2
Master Thread3
Child thread3
Master Thread4
Child thread4
Master Thread5
Child thread5
Master Thread6
Child thread6
Master Thread7
Child thread7
Master Thread8
Child thread8
Master Thread9
Child thread9

C:\Users\Administrator\Desktop\java>_
```

```
class MyThread implements Runnable {
   private int threadId;
   private int delay;
   MyThread(int id,int d){
     threadId = id;
     delay = d;//time for which thread sleeps
   }
   public void run(){
       for(int i=0;i<5;i++){
          System.out.println("Thread "+ threadId +" is
running");
          try{
             Thread.sleep(delay);
          }
          catch(InterruptedException e){
             e.printStackTrace();
          }
       }
      System.out.println("Thread "+ threadId +" is
Finished");
    }
}
```

```
public class RunnableThread {
   public static void main(String[] args){
      System.out.println("Main thread starts");
      MyThread r1 = new MyThread(1,500);
      MyThread r2 = new MyThread(2,300);
      MyThread r3 = new MyThread(3,600);
      Thread t1 = newThread(r1);
      Thread t2 = newThread(r2);
      Thread t3 = newThread(r3);
      t1.start();
      t2.start();
      t3.start();
      System.out.println("Main thread is finished");
   }
}
```



```
C:\Users\Administrator\Desktop\java>javac thread8.java

C:\Users\Administrator\Desktop\java>java RunnableThread
Main thread starts
Thread 1 is running
Thread 3 is running
Main thread is finished
Thread 2 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 3 is running
Thread 2 is running
Thread 1 is running
Thread 3 is running
Thread 2 is running
Thread 1 is running
Thread 2 is Finished
Thread 3 is running
Thread 1 is running
Thread 3 is running
Thread 1 is Finished
Thread 3 is Finished

C:\Users\Administrator\Desktop\java>_
```

Difference between thread and runnable

Real Time Examples???

- Online bus/anything ticket booking : here many users trying to book available ticket at same time (ex : tatkal booking) , here application needs to handle different threads (diff users request to server) , if tickets sold out/not available then rest users will get correct response as not available to book.
- Joint Account holder having multiple ATM cards for same account and trying to perform operation same time . at time one operation is handled for account and then next one on updated data.

How to prevent Thread Execution?

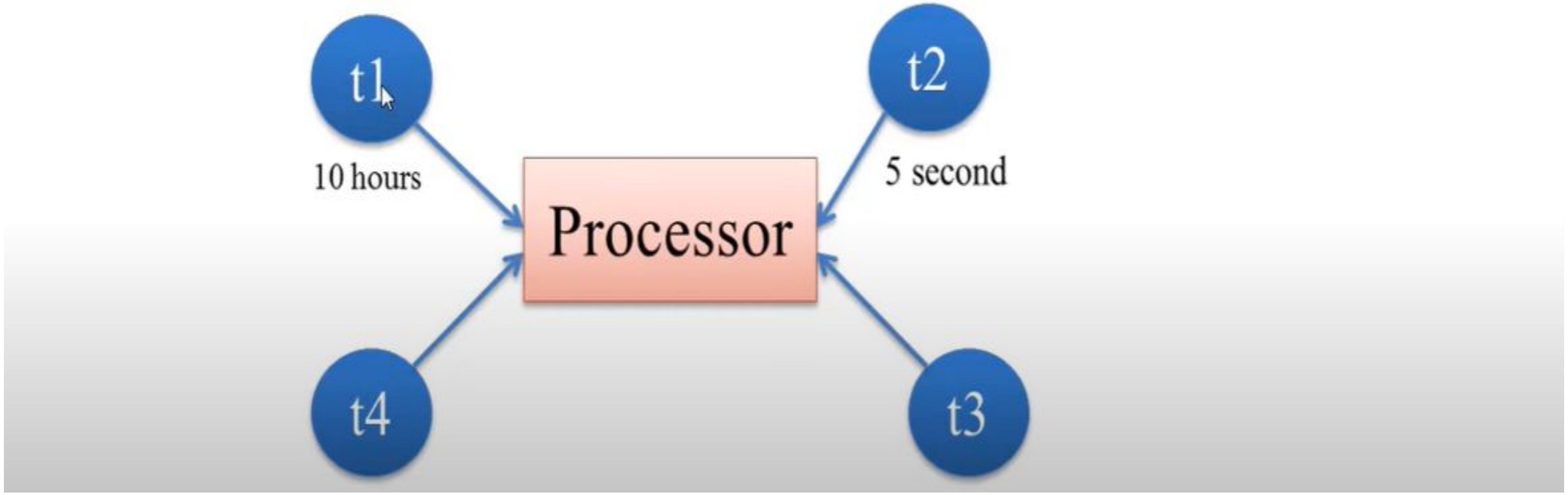
- Thread offer 3 methods to prevent Thread execution

1. yield()
2. join()
3. sleep()

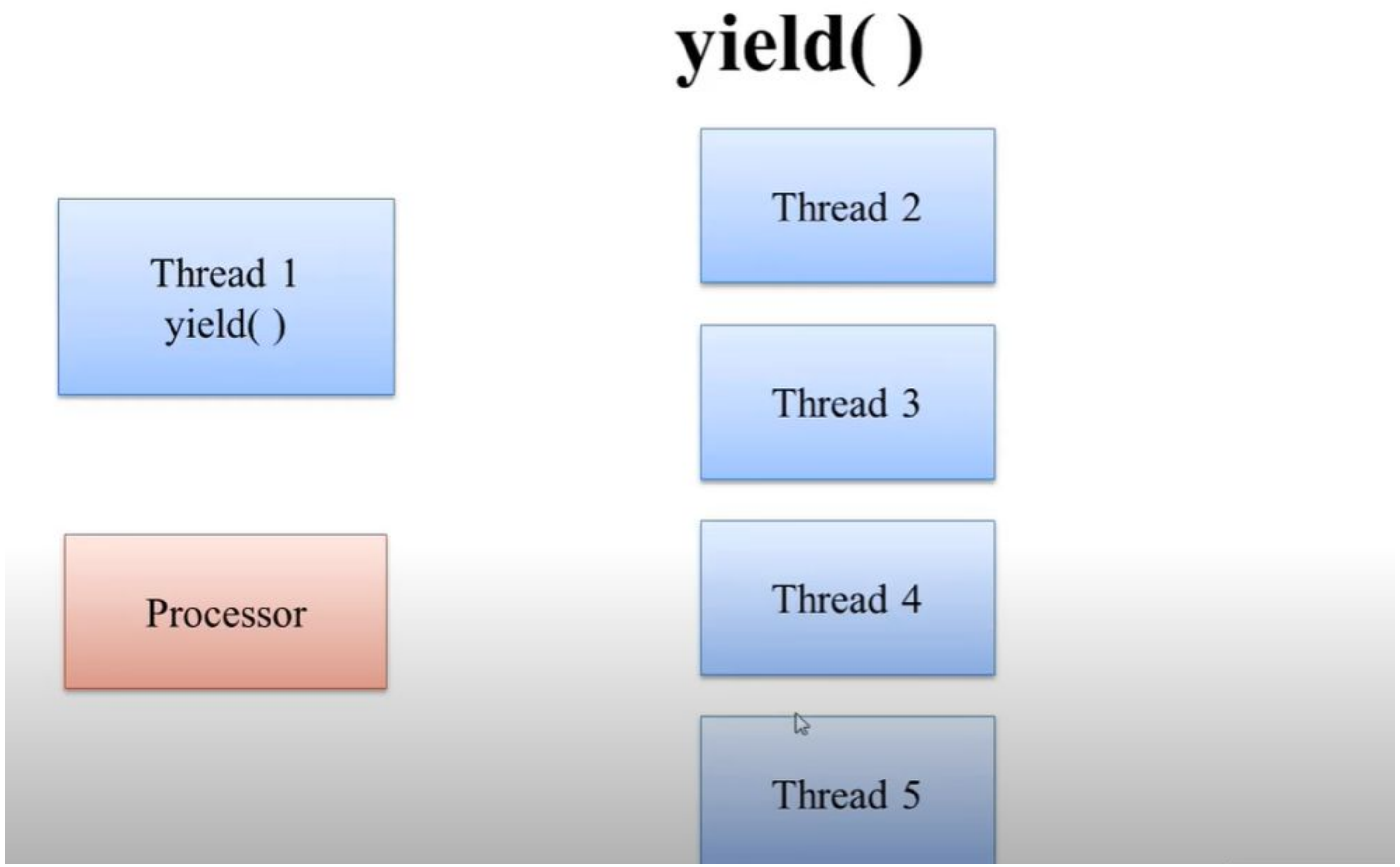
What is the yield() method?

- So yield() method pauses current execution to give the chance for remaining waiting thread of same priority.
- If no other thread is waiting or all waiting threads have low priority then same thread can continue its execution.

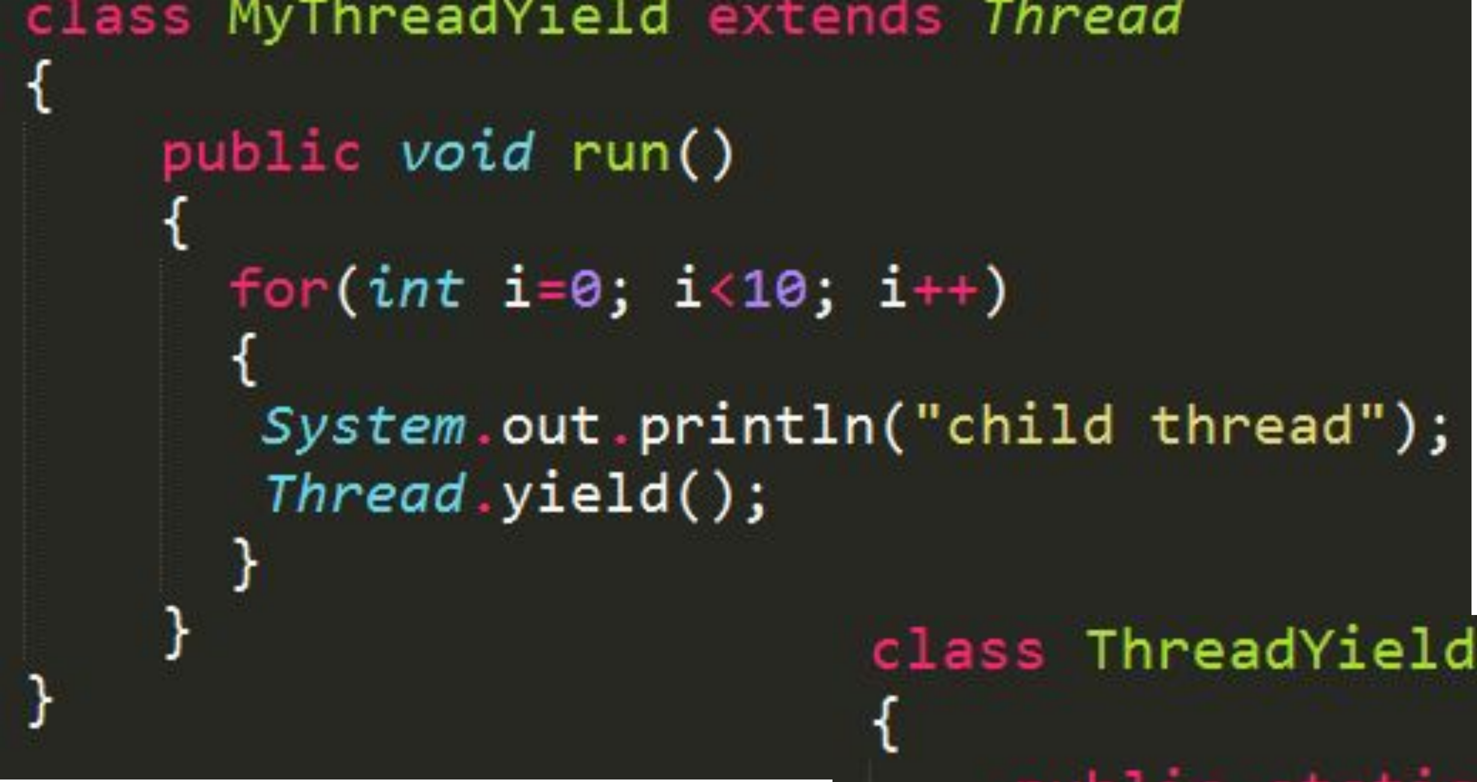
yield()



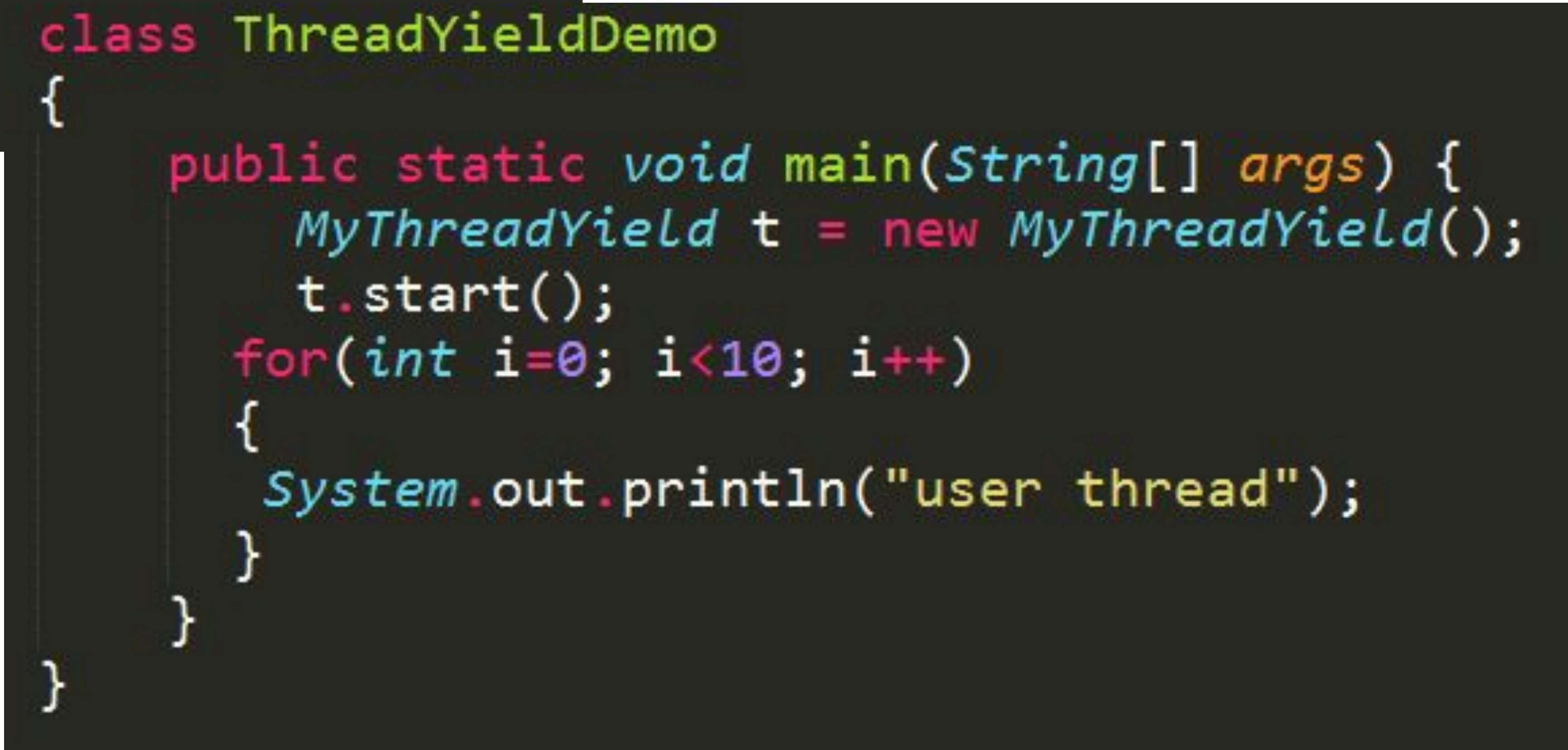
yield()


yield( )
Thread 1
yield( )
Processor
Thread 2
Thread 3
Thread 4
Thread 5

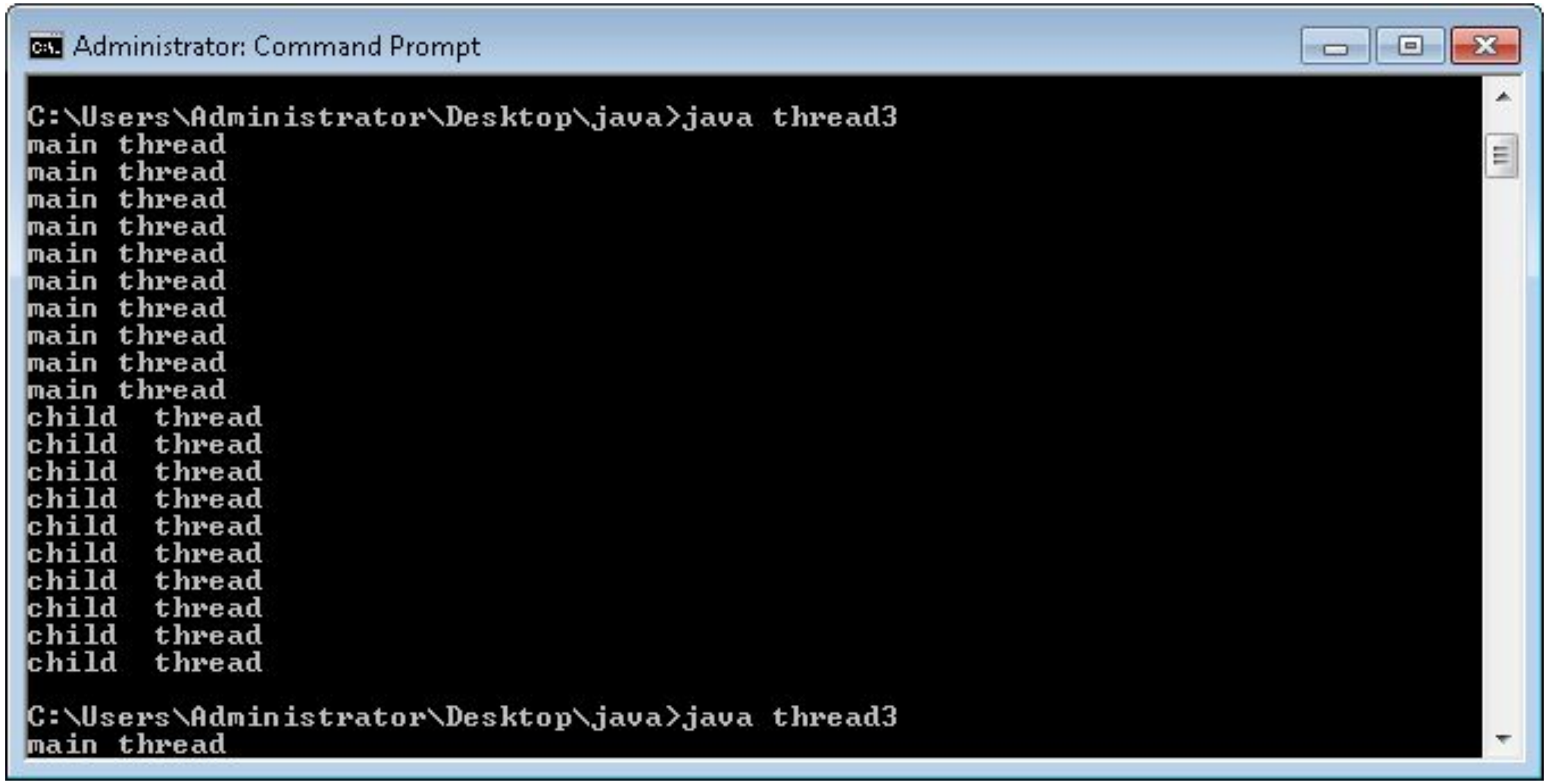
Example



```
class MyThreadYield extends Thread
{
    public void run()
    {
      for(int i=0; i<10; i++)
      {
       System.out.println("child thread");
       Thread.yield();
      }
    }
}
```



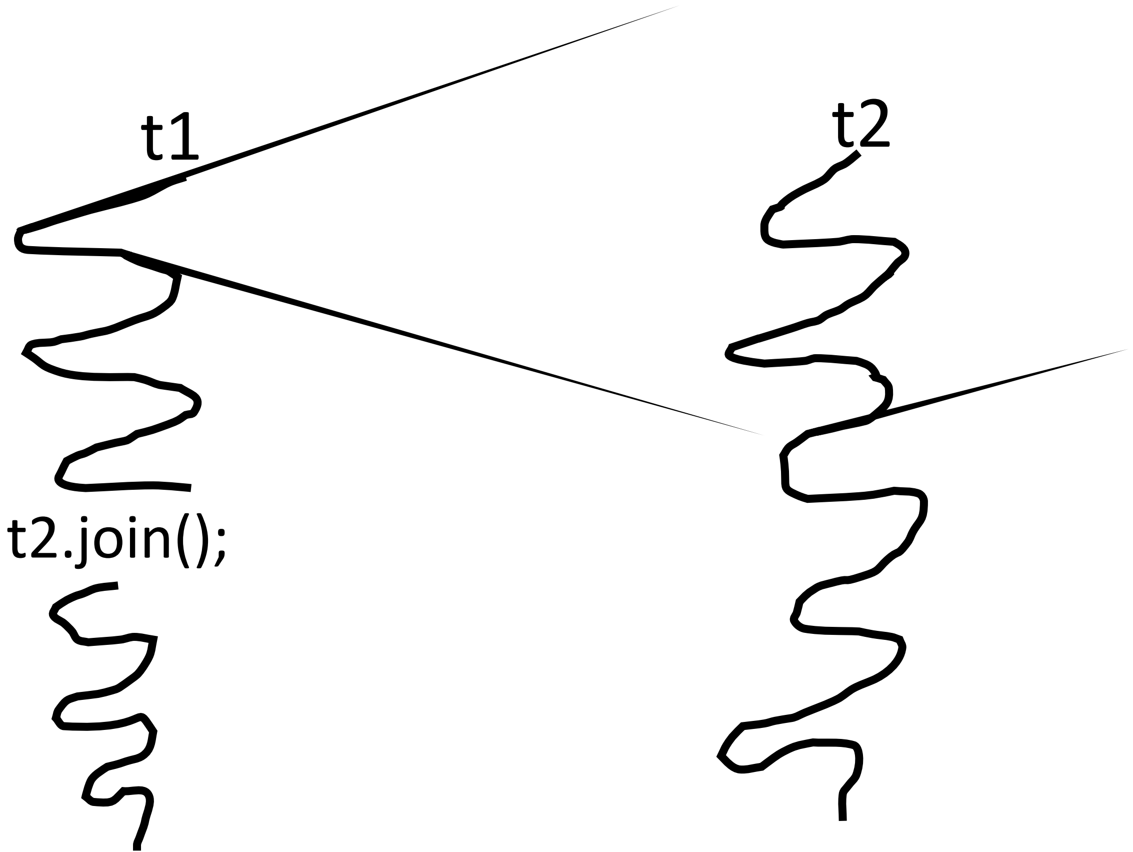
```
class ThreadYieldDemo
{
    public static void main(String[] args) {
        MyThreadYield t = new MyThreadYield();
        t.start();
      for(int i=0; i<10; i++)
      {
       System.out.println("user thread");
      }
    }
}
```


Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>java thread3
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
child  thread
child  thread
child  thread
child  thread
child  thread
child  thread
child  thread
child  thread
child  thread
child  thread

C:\Users\Administrator\Desktop\java>java thread3
main thread
```

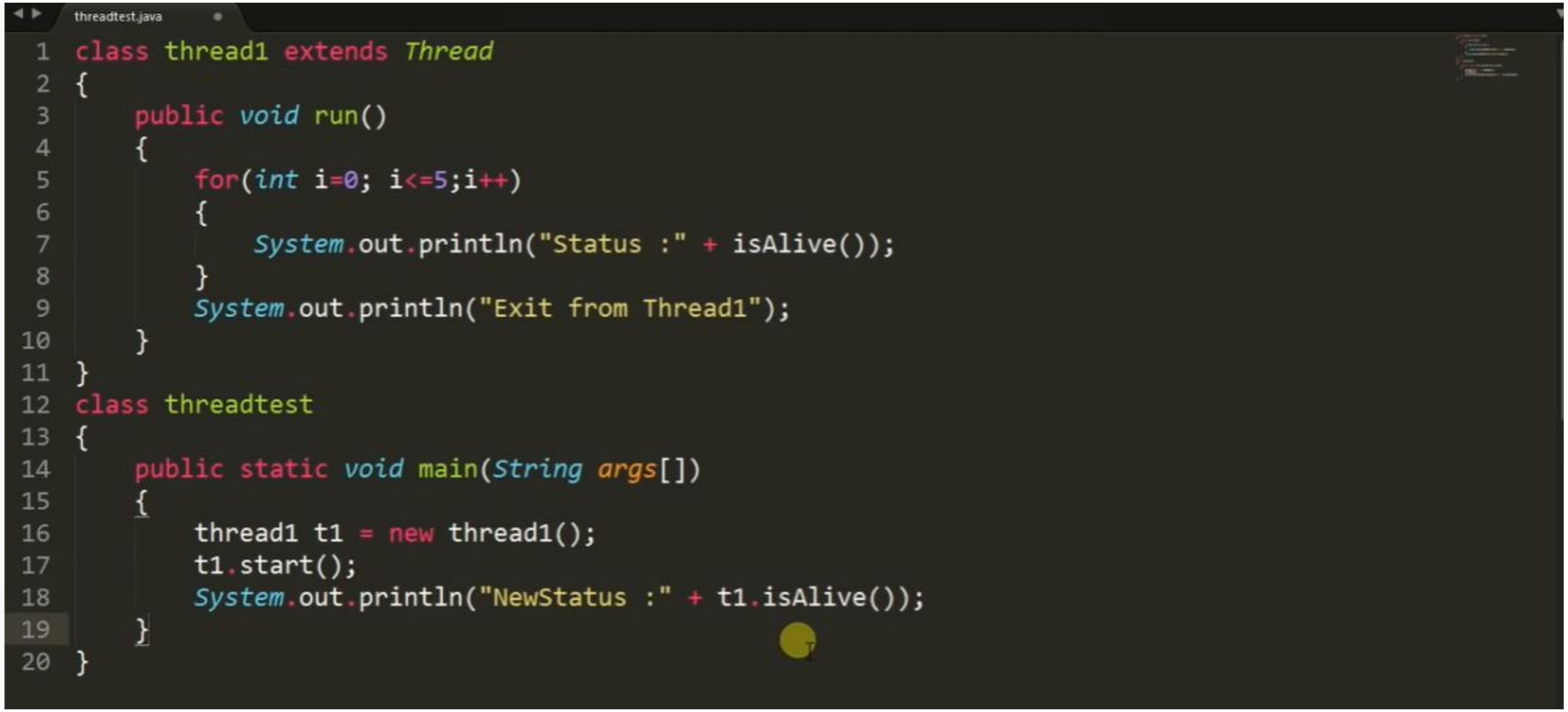
isAlive() and join() method

- If a Thread want to wait until the completion of other thread then join() method must call.

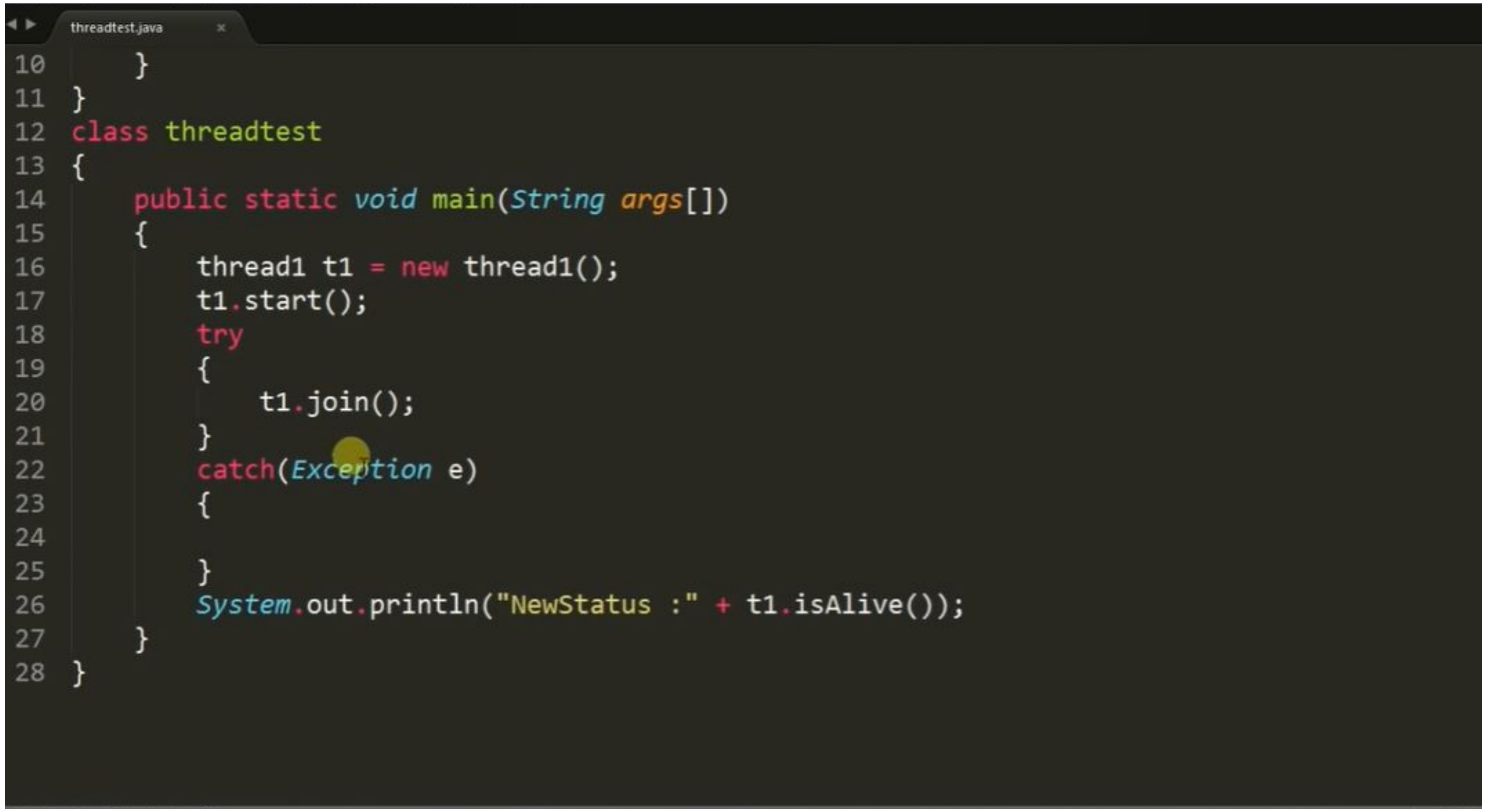


- The join() method waits for a thread to die.
- In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

isAlive()


threadtest.java

```java
class thread1 extends Thread
{
	public void run()
	{
		for(int i=0; i<=5;i++)
		{
			System.out.println("Status :" + isAlive());
		}
		System.out.println("Exit from Thread1");
	}
}
class threadtest
{
	public static void main(String args[])
	{
		thread1 t1 = new thread1();
		t1.start();
		System.out.println("NewStatus :" + t1.isAlive());
	}
}
```


threadtest.java

```
    }
}
class threadtest
{
    public static void main(String args[])
    {
        thread1 t1 = new thread1();
        t1.start();
        try
        {
            t1.join();
        }
        catch(Exception e)
        {

        }
        System.out.println("NewStatus :" + t1.isAlive());
    }
}
```

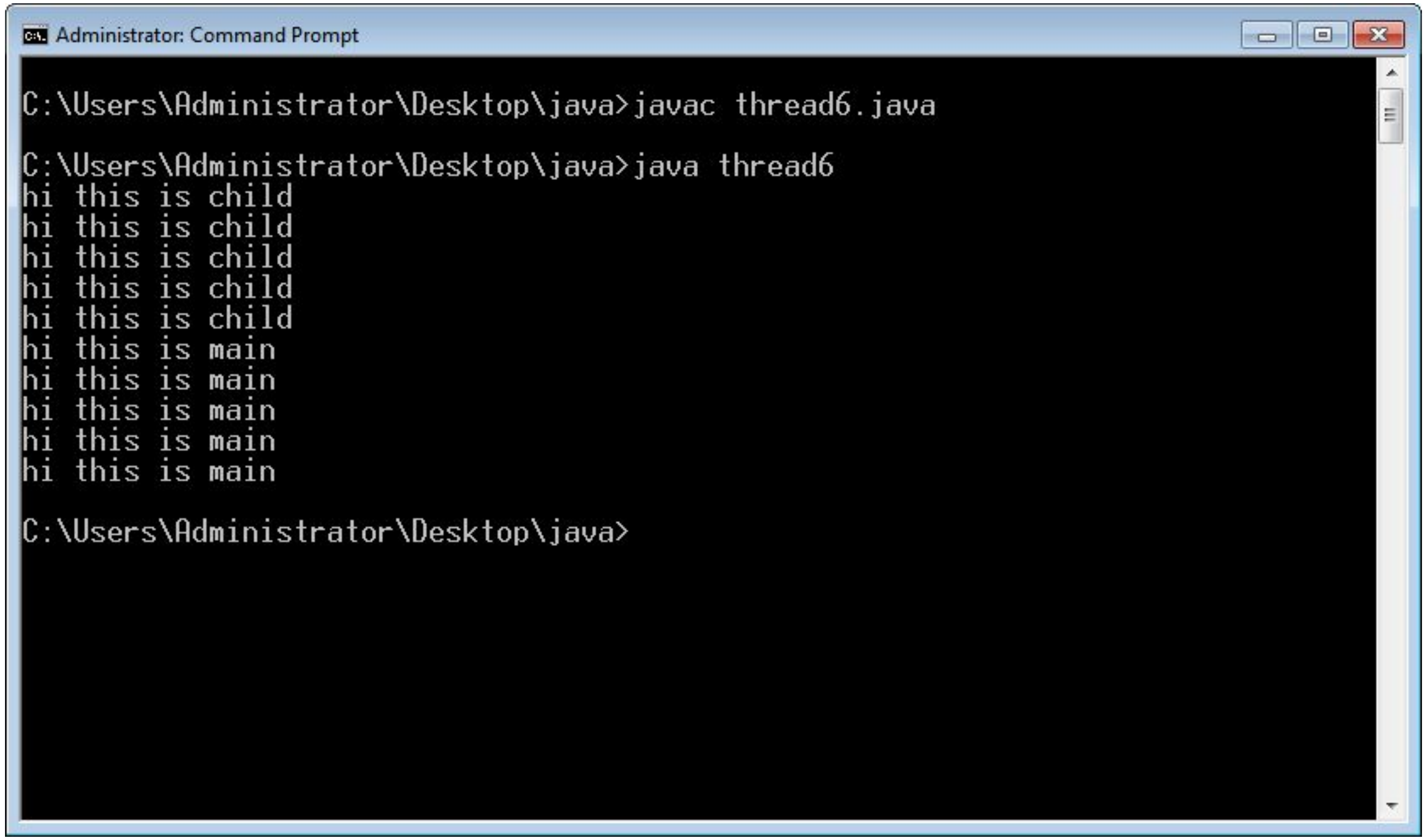
3 types of join() methods in Thread class

1. public final void join() throws InterruptedException
2. public final void join(long) throws InterruptedException
3. public final void join(long ms, int ns) throws InterruptedException

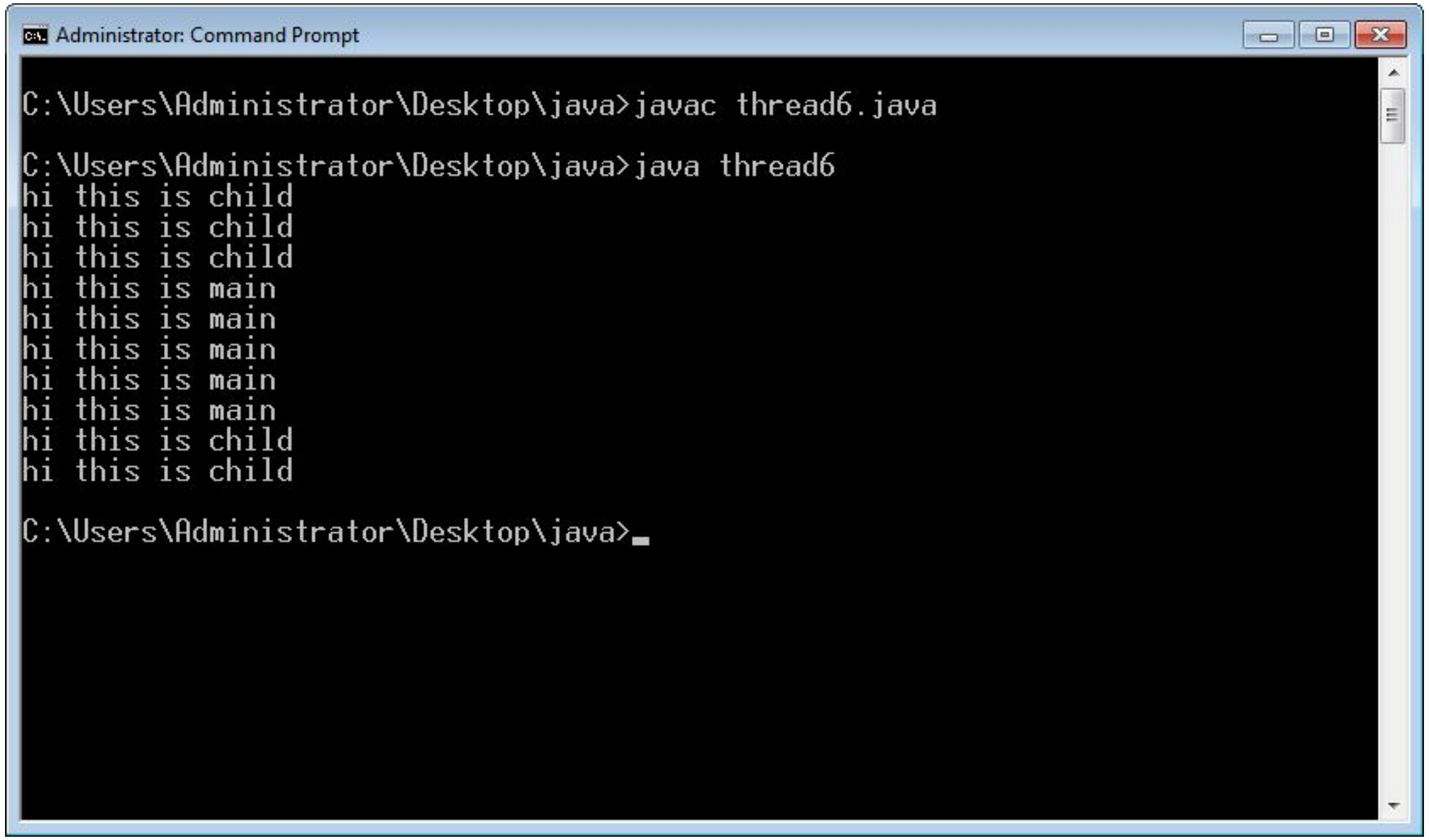
Example1

```
class thread6 extends Thread{
public void run(){
 for(int i=1;i<=5;i++){
 System.out.println(“hi this is child thread”);
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
}
 }
```

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

 thread6 t2=new thread6();

 t2.start();
              try{
                     t2.join();
                     //t2.join(1500);
                 }
              catch(Exception 
e){System.out.println(e);}


              for(int i=1;i<=5;i++)
                     {
                            System.out.println("hi this is 
main");
                     }

}
```


Administrator: Command Prompt

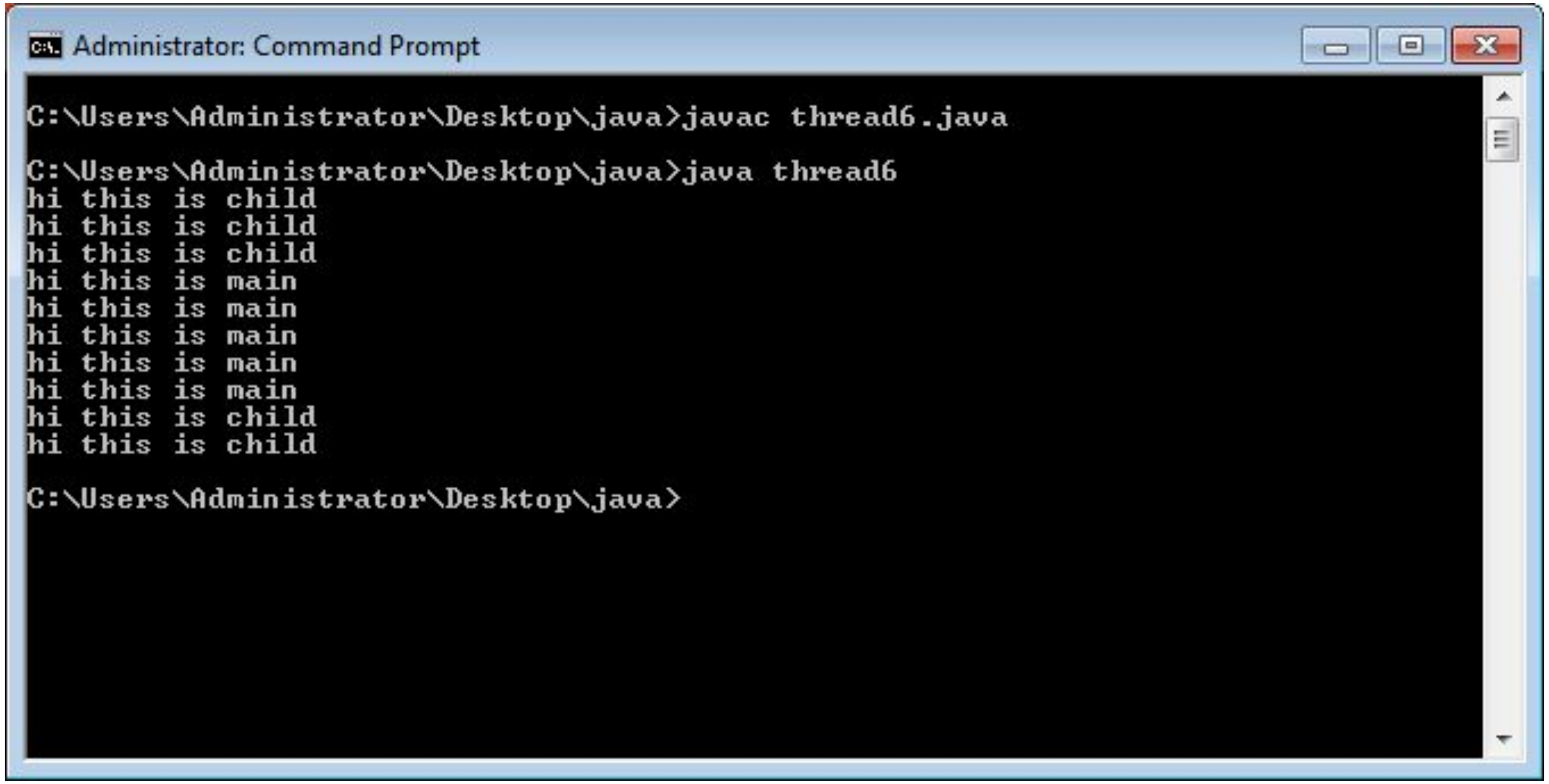
```
C:\Users\Administrator\Desktop\java>javac thread6.java

C:\Users\Administrator\Desktop\java>java thread6
hi this is child
hi this is child
hi this is child
hi this is child
hi this is child
hi this is main
hi this is main
hi this is main
hi this is main
hi this is main

C:\Users\Administrator\Desktop\java>
```



```
C:\Users\Administrator\Desktop\java>javac thread6.java

C:\Users\Administrator\Desktop\java>java thread6
hi this is child
hi this is child
hi this is child
hi this is main
hi this is main
hi this is main
hi this is main
hi this is main
hi this is child
hi this is child

C:\Users\Administrator\Desktop\java>_
```

- When the join() method is invoked, the current thread stops its execution and go into the wait state. The current thread remains in the wait state until the thread on which the join() method is invoked called is dead or the wait for the specified time frame(in milliseconds + nanos) is over.

Another Small change in join() method-
t.join(4000, 2000);



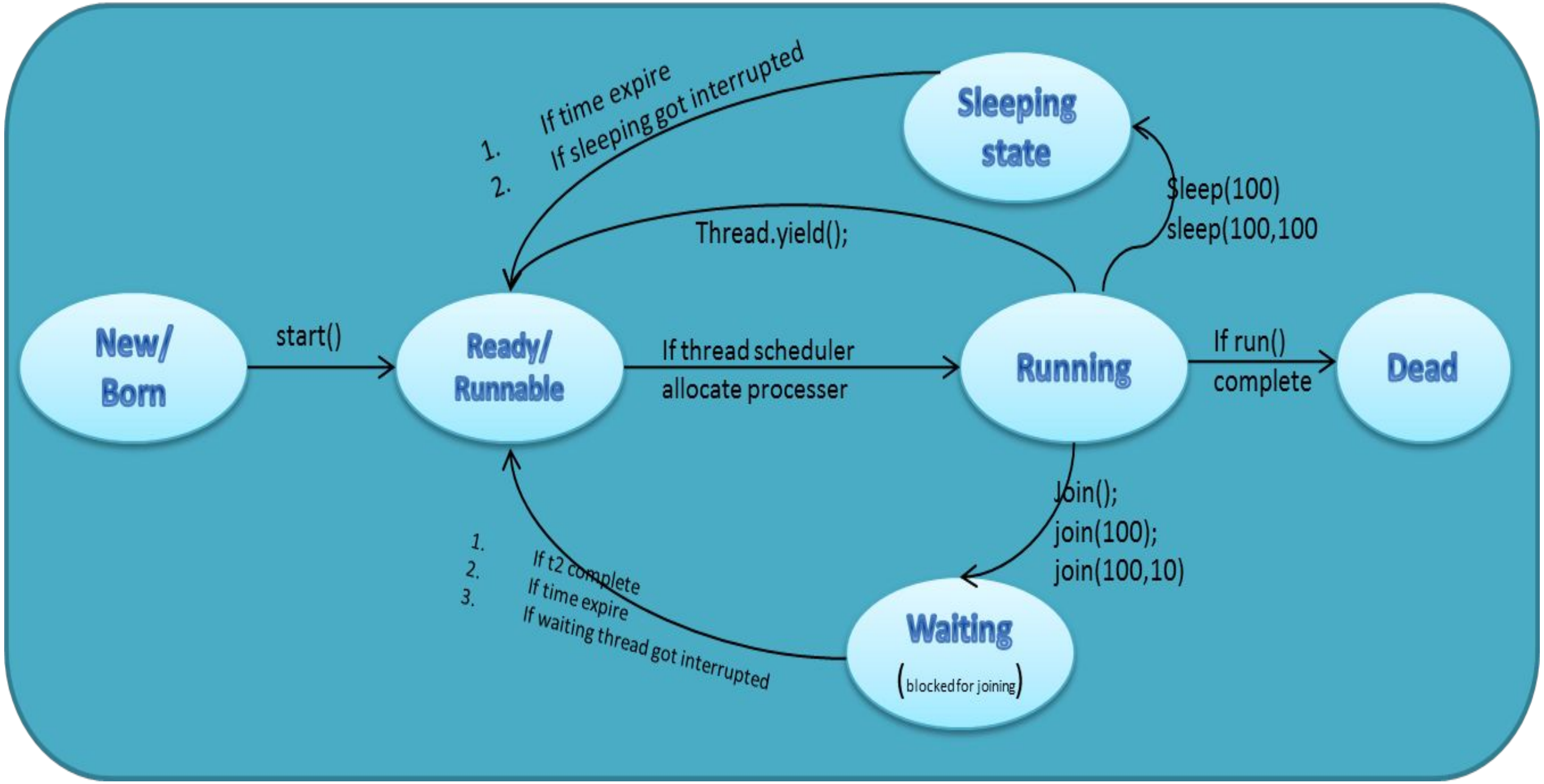
Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac thread6.java

C:\Users\Administrator\Desktop\java>java thread6
hi this is child
hi this is child
hi this is child
hi this is main
hi this is main
hi this is main
hi this is main
hi this is main
hi this is child
hi this is child

C:\Users\Administrator\Desktop\java>
```

Example1

```
class MyThread extends Thread{
static Thread mt;
public void run(){
 try{
    mt.join();
   }catch(Exception e){System.out.println(e);}


 for(int i=1;i<=5;i++){
 System.out.println(“hi this is child thread”);
}
 }
```

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

 MyThread mt = Thread.CurrentThread();
MyThread t2 = new MyThread();


 t2.start();

            for(int i=1;i<=5;i++)
                  {
                        System.out.println("hi this is
main");
      Thread.sleep(3000);
                  }

}
}
```



New/ Born
start()
Ready/ Runnable
If thread scheduler allocate processer
Running
If run() complete
Dead
Thread.yield();
Sleeping state
Sleep(100)
sleep(100,100
1. If time expire
2. If sleeping got interrupted
Join();
join(100);
join(100,10)
Waiting
(blocked for joining)
1. If t2 complete
2. If time expire
3. If waiting thread got interrupted

	Yield()	Join()	Sleep()
Purpose	This method allows the current thread to release its lock of the object to other thread with same priority.	The join method allows one thread to wait for the completion of another.	This is used to sleep current thread to suspend its execution for a specified period and it doesn't release the lock of the object.
Is it overloaded	No	Yes	Yes
Is it Final?	No	Yes	No
Does it throw IntertuptedException?	No	Yes	Yes
Is it native?	Yes	No	Sleep(long ms) - yes Sleep(long ms, int ns) - no
Is it static?	Yes	No	Yes

Most valuable concept in multithreading is Synchronization

- People know multithreading but not good in synchronization
- Little bit difficult

Few basic point about **synchronized** modifier

- Class can’t be **synchronized**
- Variables can’t be **synchronized**

- Method can be **synchronized**
- Blocks can be **synchronized**

What is the purpose of synchronization?

- **For example-** if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file.

- If resources are shared
- Railway Tatkal reservation – very good example
- Bank transaction using ATM, Online etc.

Synchronization in Java

- Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- The synchronization is mainly used to
    - To prevent thread interference.
    - To prevent data inconsistency problem.

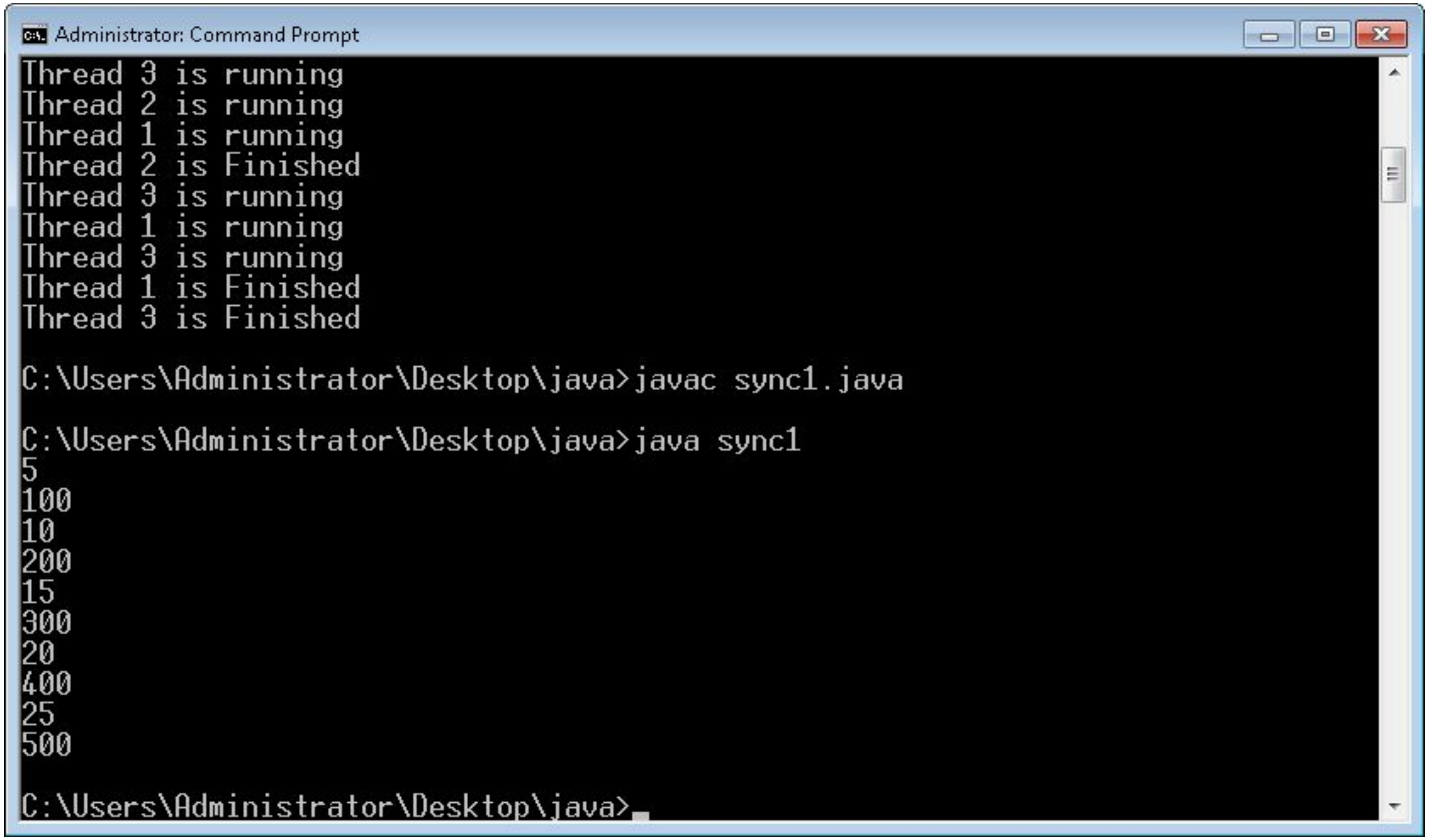
-

```
class Table{
void printTable(int n){//method not synchronized
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
    Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}
  }  }}
```

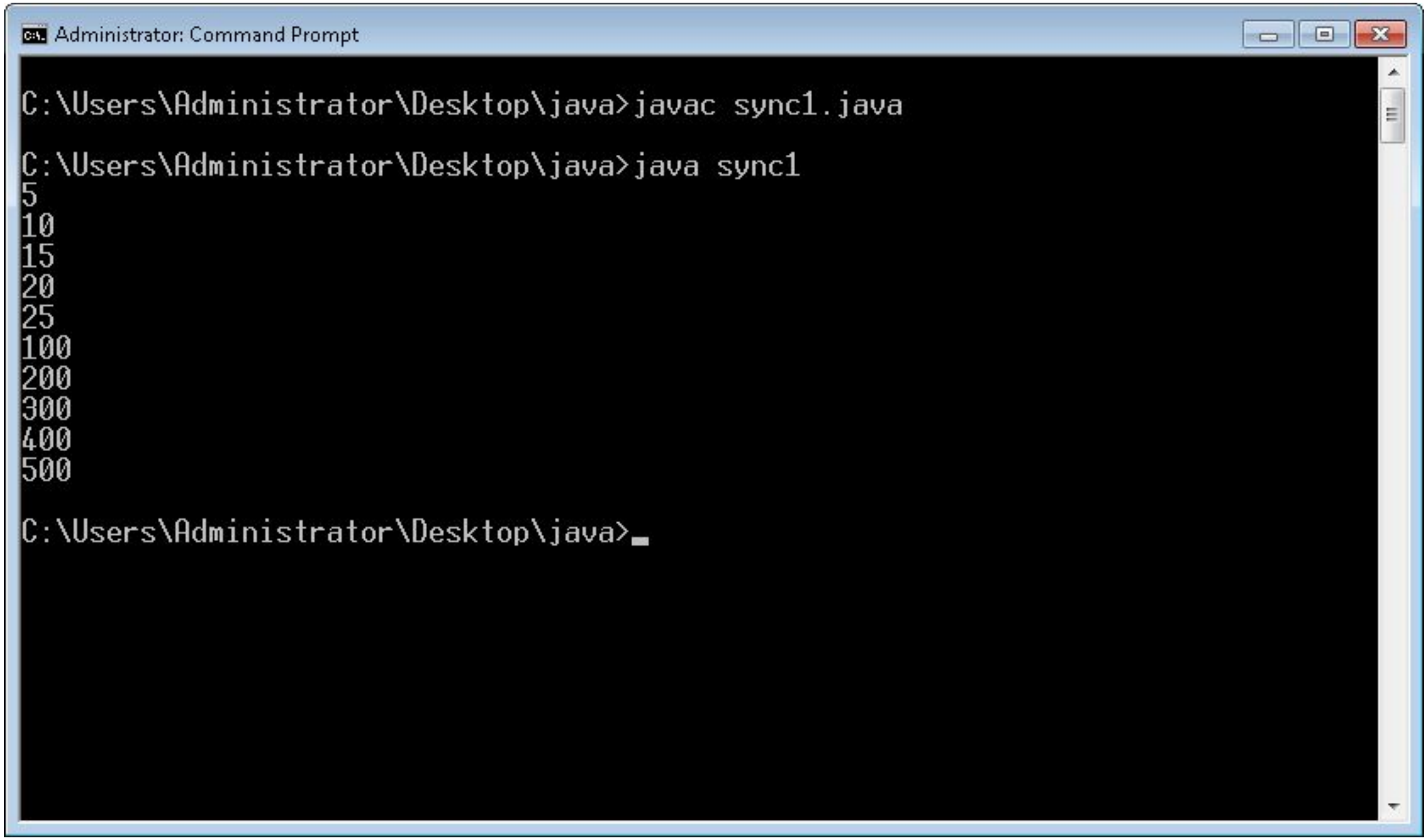
```
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
```

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

```
class TestSynchronization1{
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();
}
}
```



```
Thread 3 is running
Thread 2 is running
Thread 1 is running
Thread 2 is Finished
Thread 3 is running
Thread 1 is running
Thread 3 is running
Thread 1 is Finished
Thread 3 is Finished

C:\Users\Administrator\Desktop\java>javac sync1.java

C:\Users\Administrator\Desktop\java>java sync1
5
100
10
200
15
300
20
400
25
500

C:\Users\Administrator\Desktop\java>
```

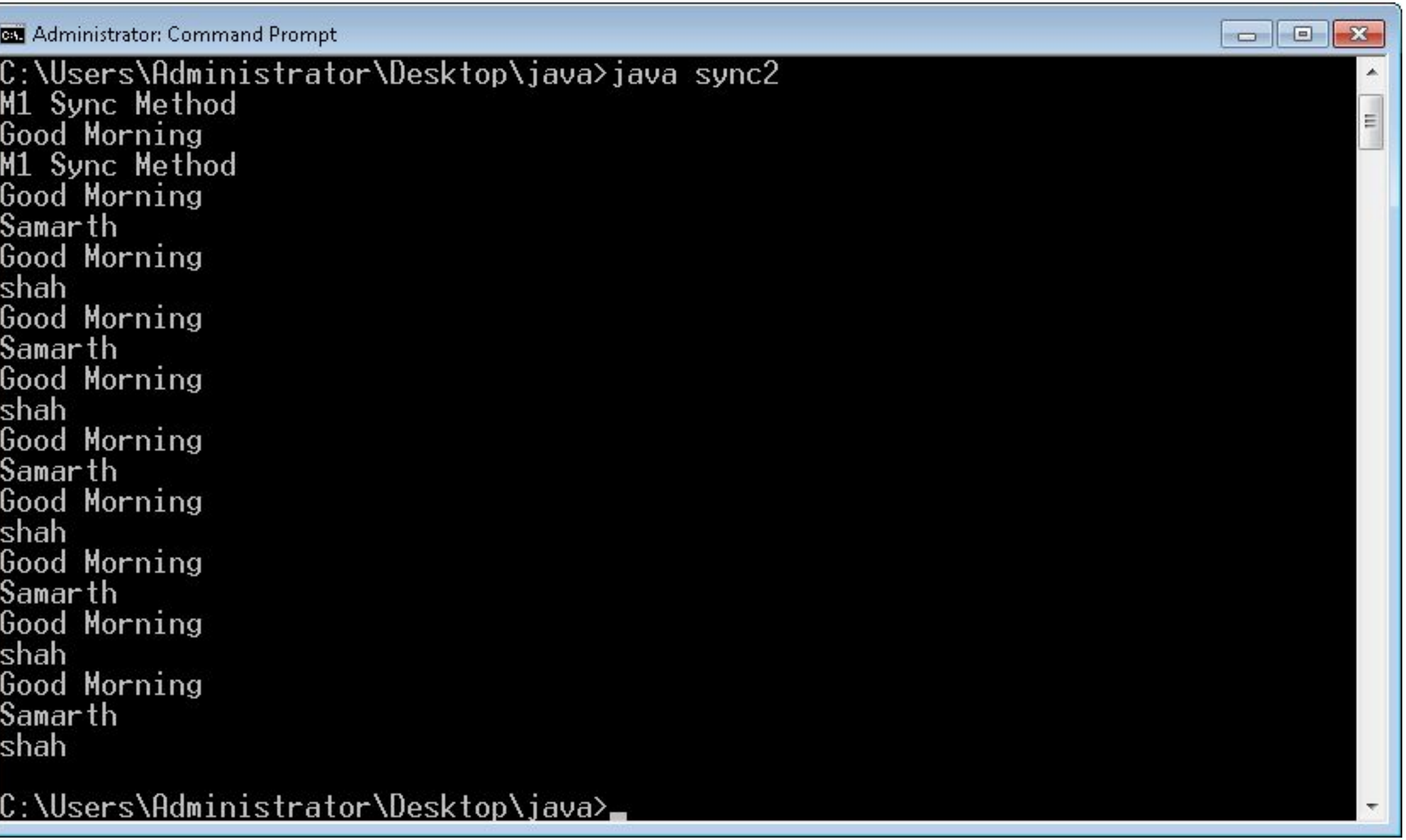

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac sync1.java

C:\Users\Administrator\Desktop\java>java sync1
5
10
15
20
25
100
200
300
400
500

C:\Users\Administrator\Desktop\java>
```

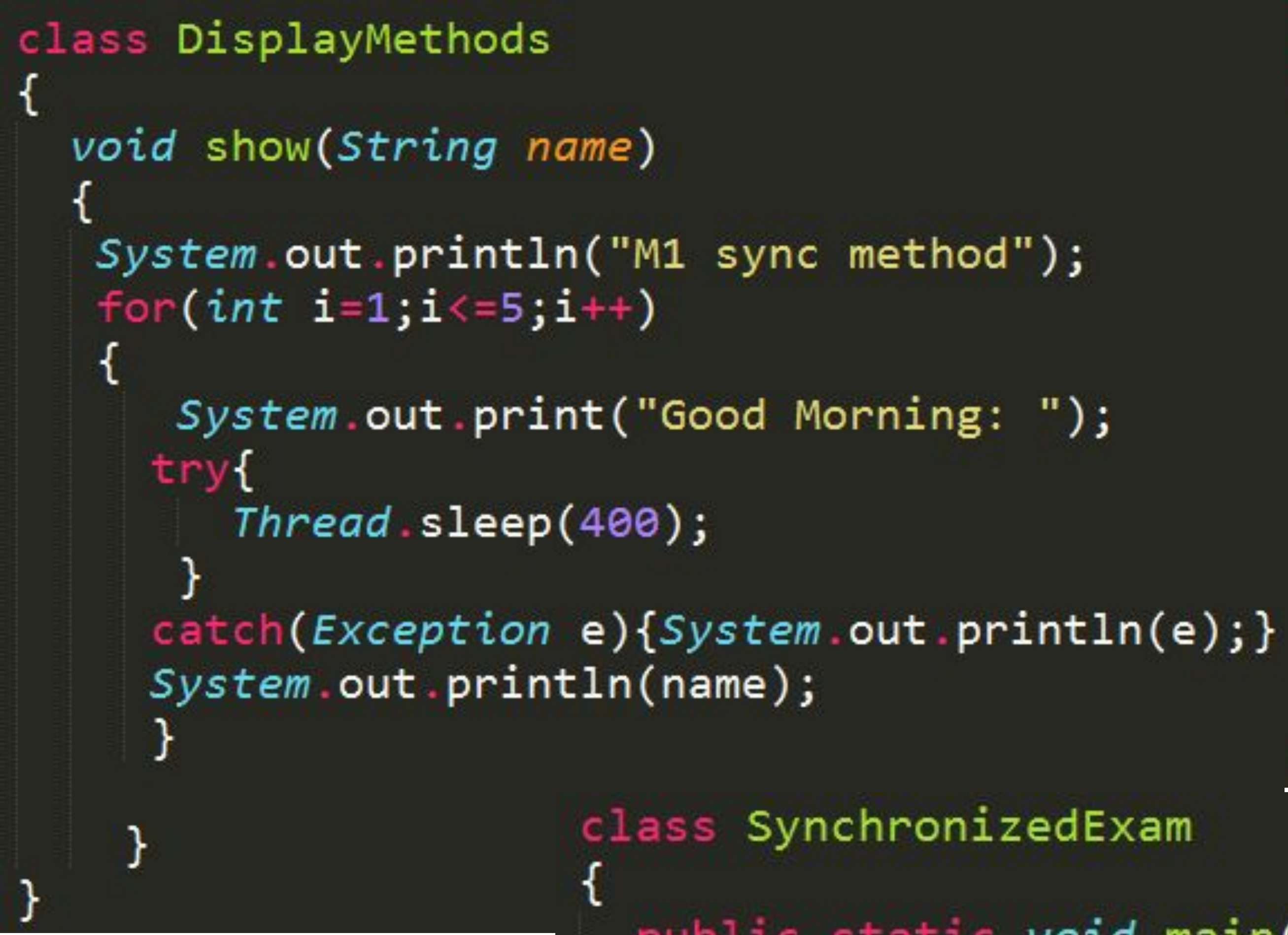
```
class display{
 void show(String name){//method
not synchronized
  System.out.println("M1 Sync
Method");

for(int i=1;i<=5;i++){
     System.out.println("Good
Morning");
     try{
      Thread.sleep(400);
     }catch(Exception
e){System.out.println(e);}
System.out.println(name);
   }

 }
}
```

```
class MyThread1 extends Thread{
display d;
String name;
MyThread1(display d,String name){
this.d=d;
this.name=name;
}
public void run(){
d.show(name);
}

}
```

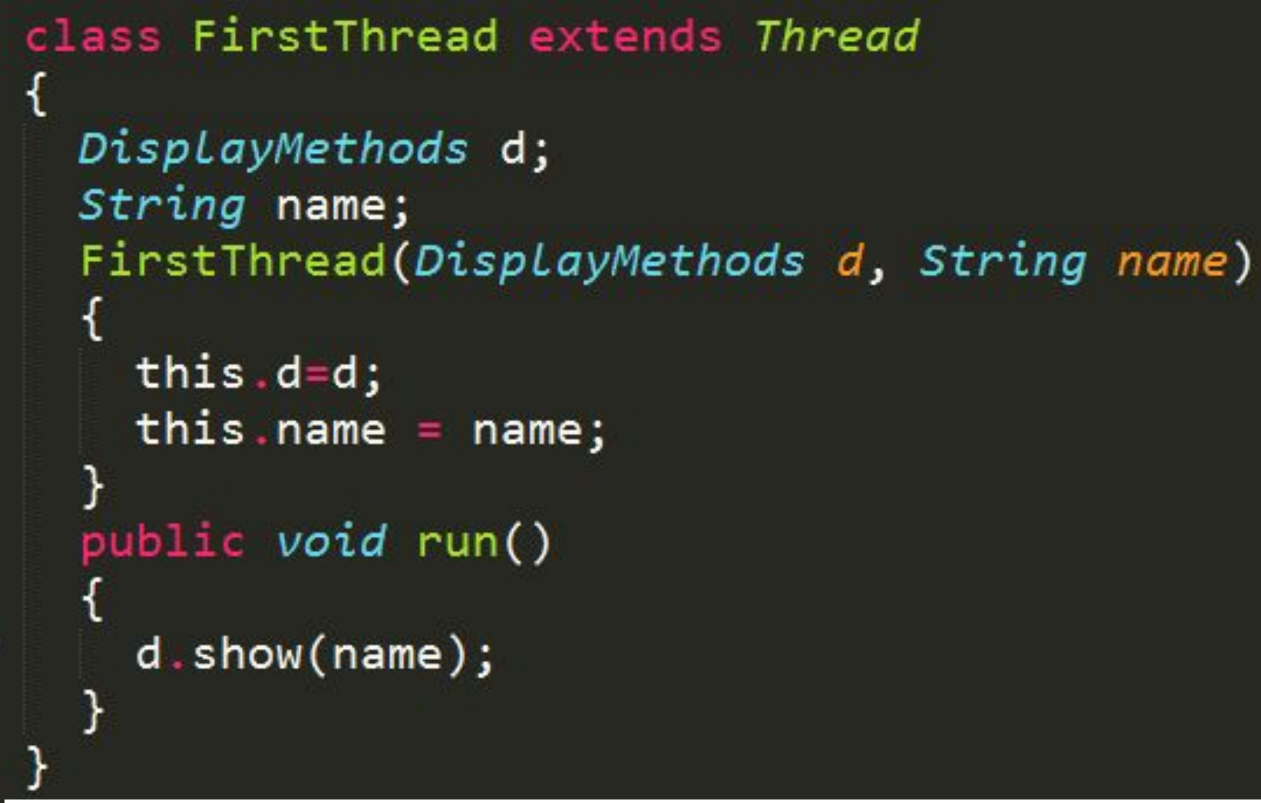
```
class sync2{
public static void main(String args[]){
display d = new display();//only one object
MyThread1 t1=new MyThread1(d,"Samarth");
MyThread1 t2=new MyThread1(d,"shah");
t1.start();
t2.start();
}
}
```


Administrator: Command Prompt

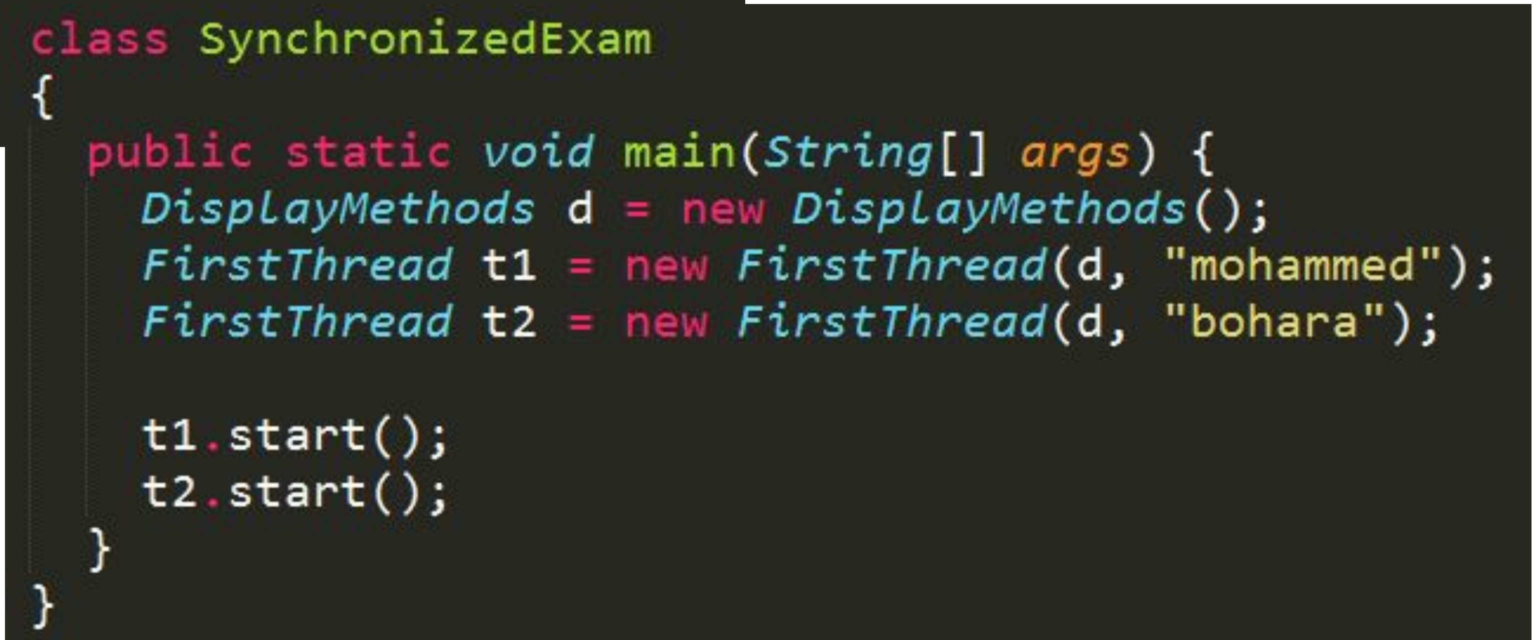
```
C:\Users\Administrator\Desktop\java>java sync2
M1 Sync Method
Good Morning
M1 Sync Method
Good Morning
Samarth
Good Morning
shah
Good Morning
Samarth
Good Morning
shah
Good Morning
Samarth
Good Morning
shah
Good Morning
Samarth
Good Morning
shah
Good Morning
Samarth
shah

C:\Users\Administrator\Desktop\java>
```



```
class DisplayMethods
{
  void show(String name)
  {
   System.out.println("M1 sync method");
   for(int i=1;i<=5;i++)
   {
      System.out.print("Good Morning: ");
     try{
       Thread.sleep(400);
      }
     catch(Exception e){System.out.println(e);}
     System.out.println(name);
     }

    }
}
```



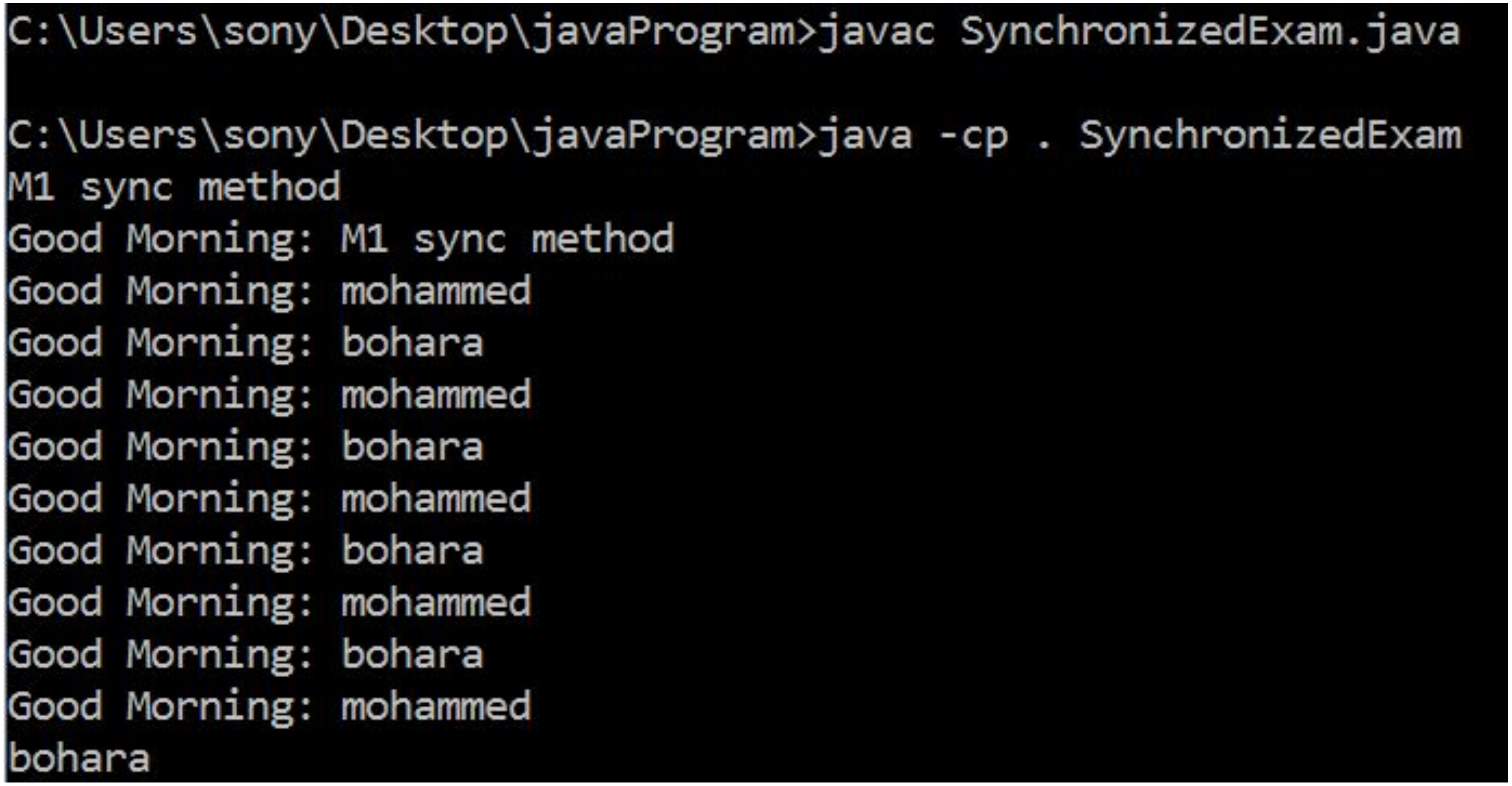
```
class FirstThread extends Thread
{
  DisplayMethods d;
  String name;
  FirstThread(DisplayMethods d, String name)
  {
    this.d=d;
    this.name = name;
  }
  public void run()
  {
    d.show(name);
  }
}
```



```
class SynchronizedExam
{
  public static void main(String[] args) {
    DisplayMethods d = new DisplayMethods();
    FirstThread t1 = new FirstThread(d, "mohammed");
    FirstThread t2 = new FirstThread(d, "bohara");

    t1.start();
    t2.start();
  }
}
```

Example

Output



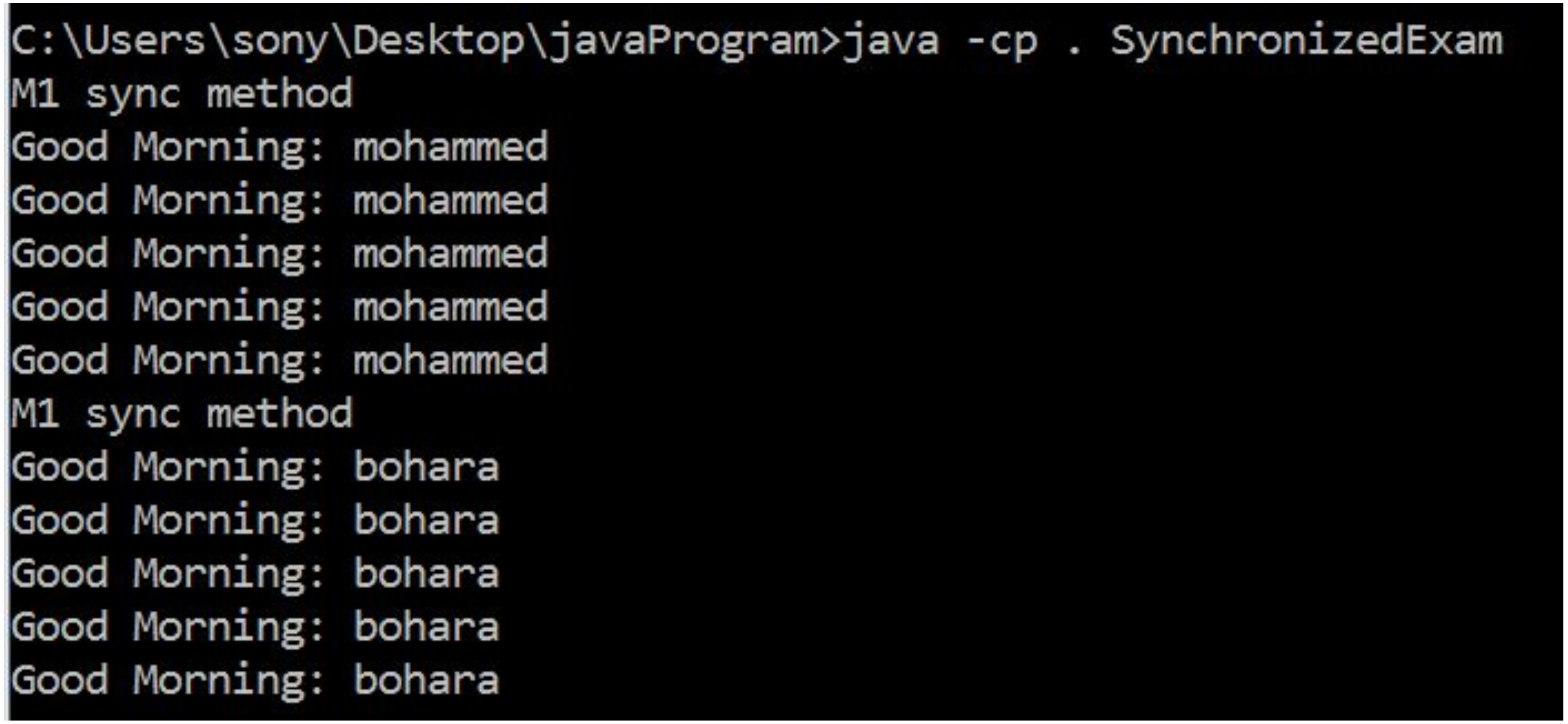
```
C:\Users\sony\Desktop\javaProgram>javac SynchronizedExam.java

C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedExam
M1 sync method
Good Morning: M1 sync method
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
bohara
```

Irregular outcome

Now add synchronized modifier with method

synchronized void show(String name){ }

Output

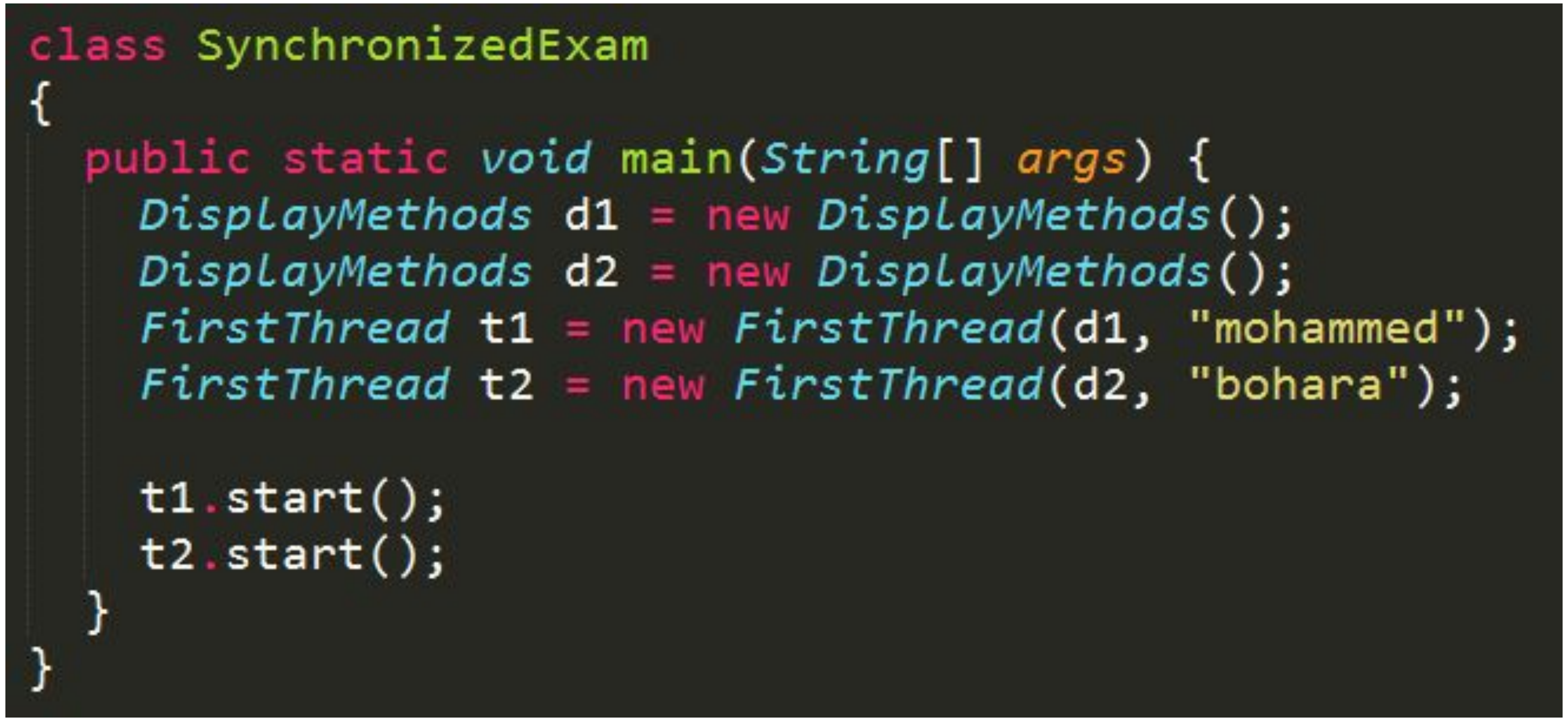


```
C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedExam
M1 sync method
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
M1 sync method
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
```

regular outcome

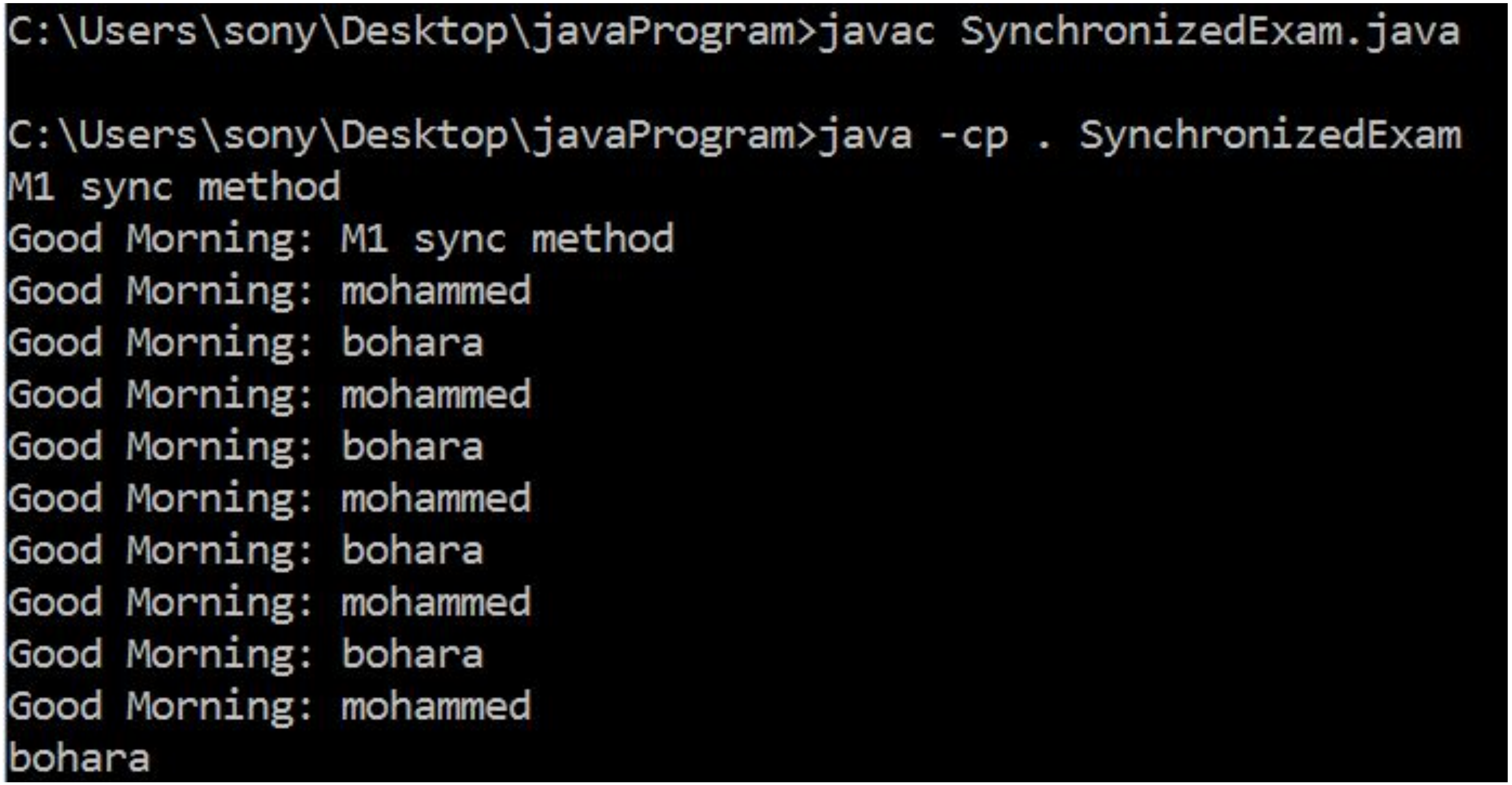
Lets do small modification in previous program

Create two object of DisplayMethod and assign it to different object and check the outcome.



```
class SynchronizedExam
{
  public static void main(String[] args) {
    DisplayMethods d1 = new DisplayMethods();
    DisplayMethods d2 = new DisplayMethods();
    FirstThread t1 = new FirstThread(d1, "mohammed");
    FirstThread t2 = new FirstThread(d2, "bohara");

    t1.start();
    t2.start();
  }
}
```

Output



```
C:\Users\sony\Desktop\javaProgram>javac SynchronizedExam.java

C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedExam
M1 sync method
Good Morning: M1 sync method
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
Good Morning: bohara
Good Morning: mohammed
bohara
```

Irregular outcome- even method is synchronized

One solution for such problem

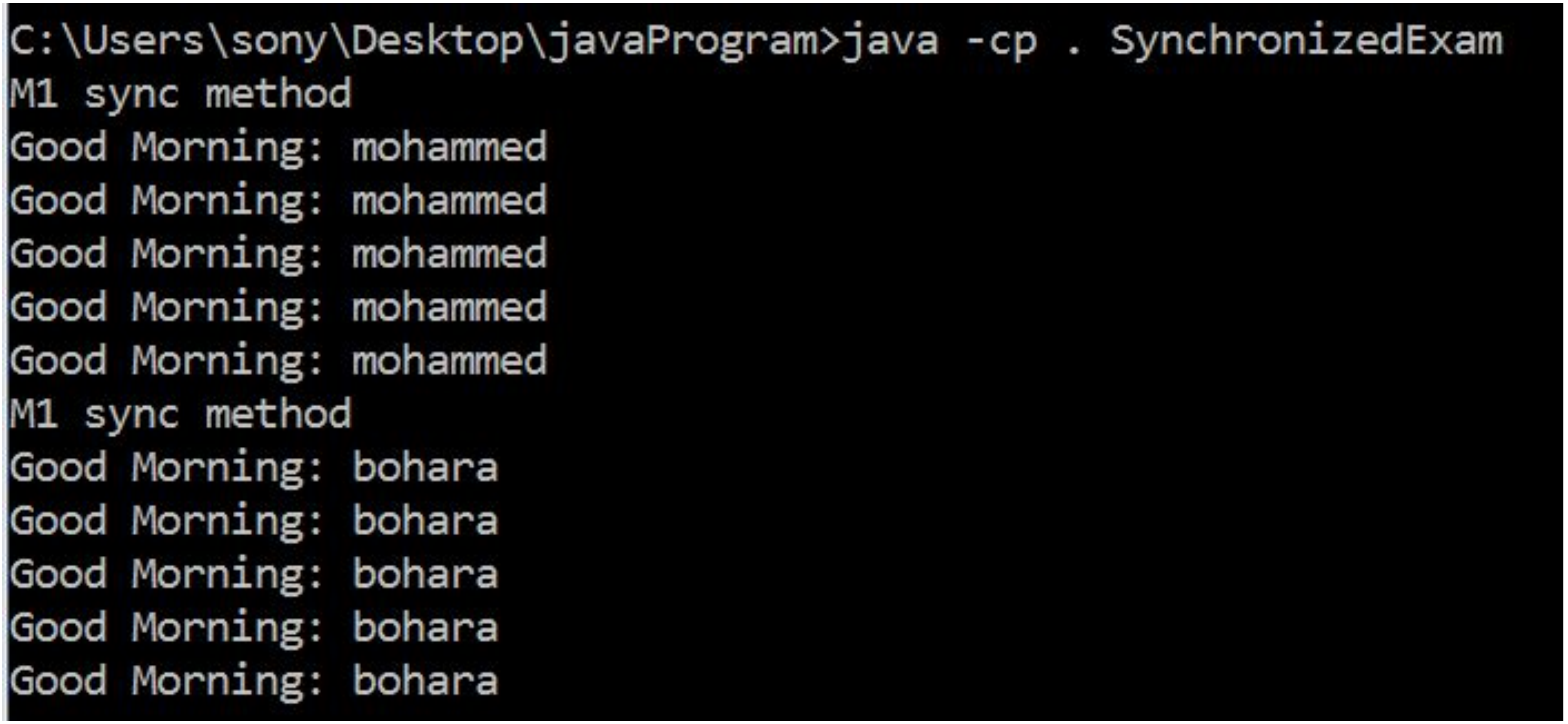
- Make method static and synchronized both

```
synchronized static void show(String name)
```

Check the outcome again

Output

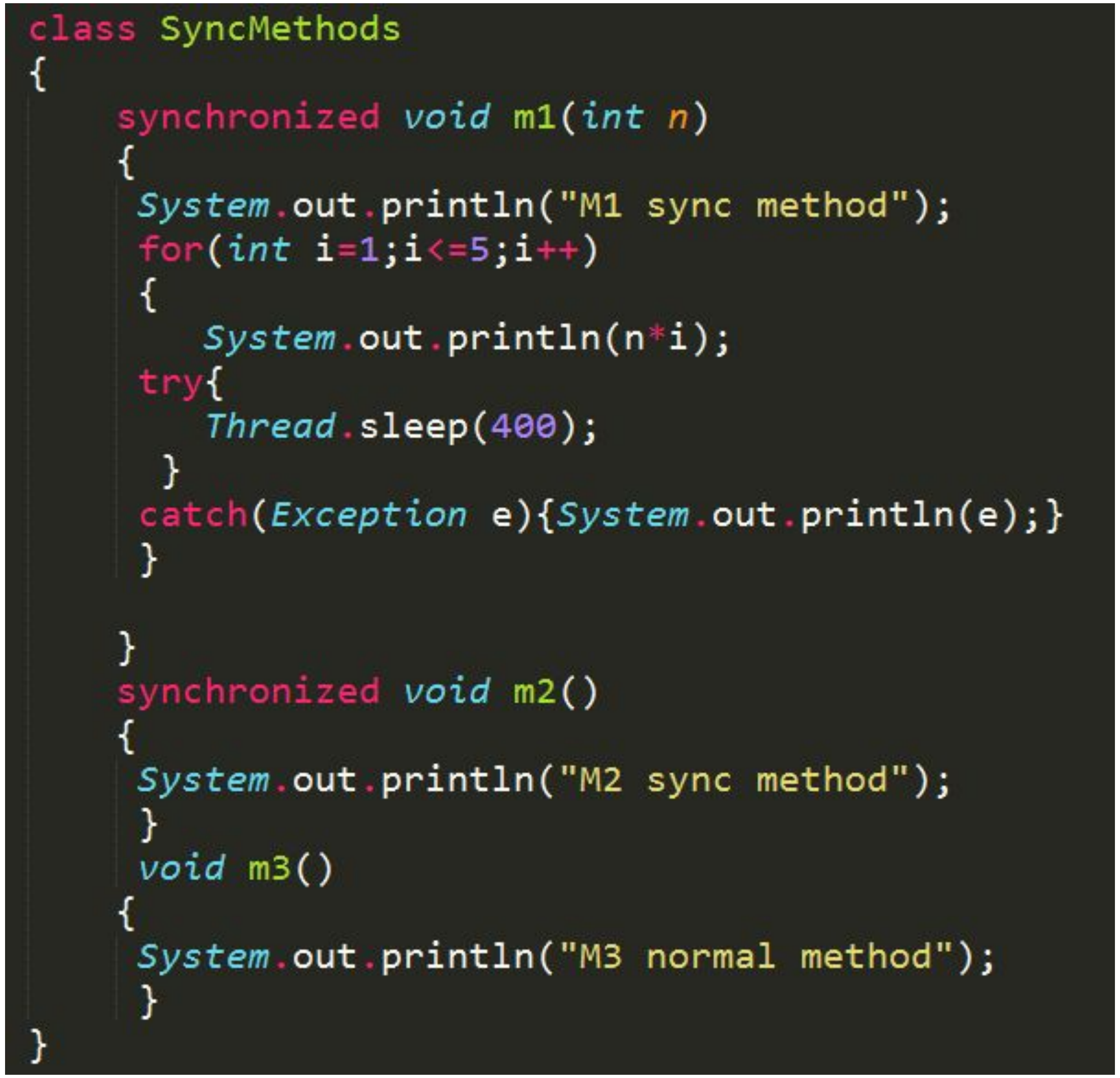
Called as class level lock



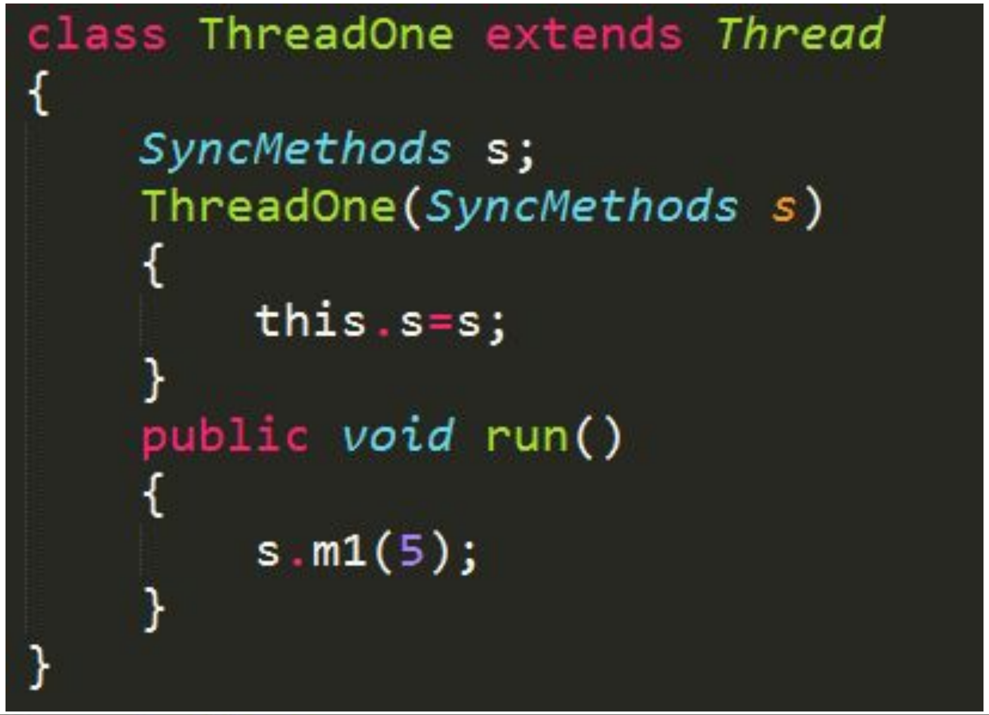
```
C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedExam
M1 sync method
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
M1 sync method
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
```

regular outcome- bcs it will make class synchronized

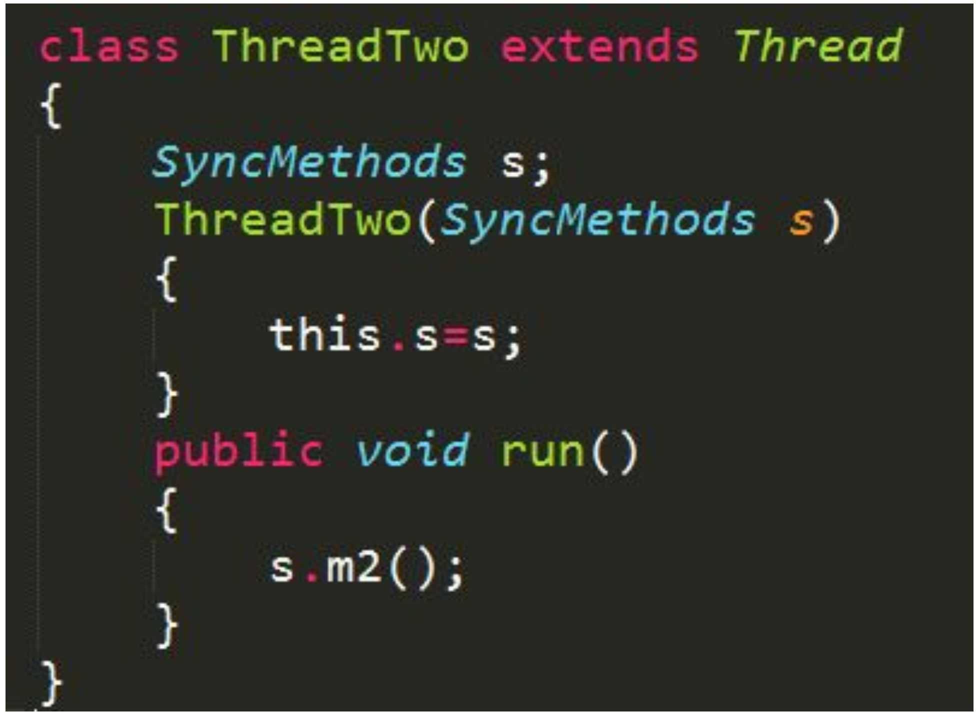
Create two synchronized method and one normal method



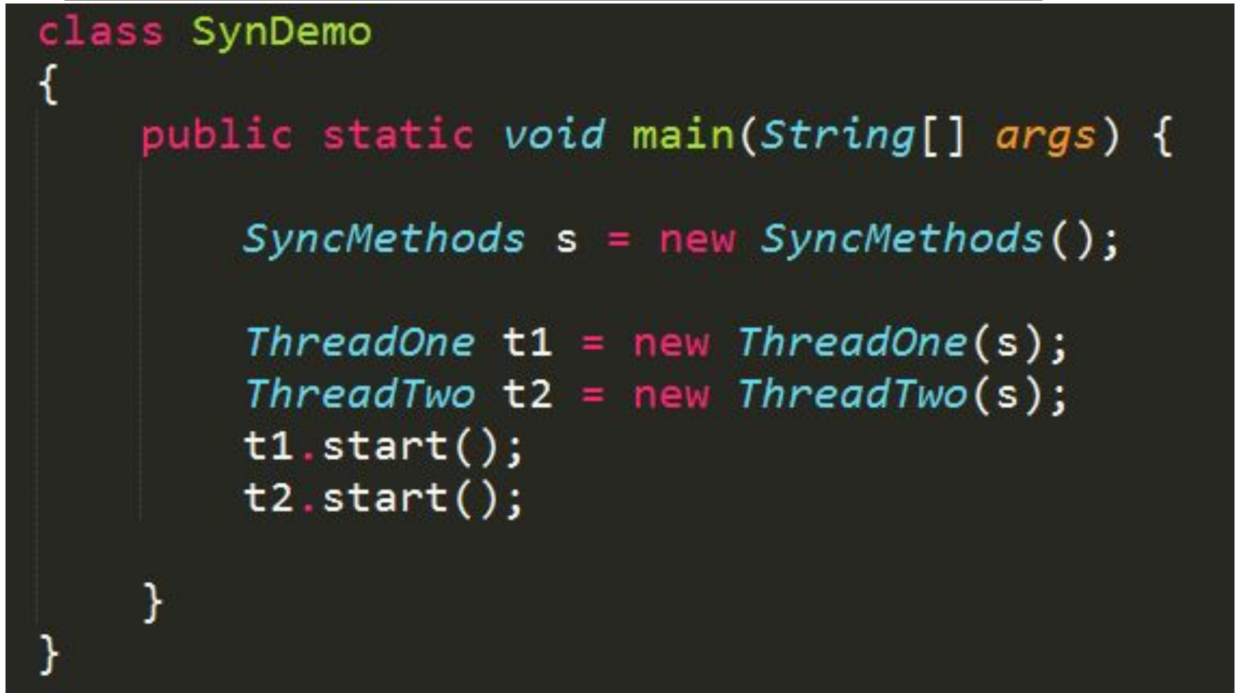
```
class SyncMethods
{
    synchronized void m1(int n)
    {
     System.out.println("M1 sync method");
     for(int i=1;i<=5;i++)
     {
        System.out.println(n*i);
     try{
        Thread.sleep(400);
      }
     catch(Exception e){System.out.println(e);}
     }

    }
    synchronized void m2()
    {
     System.out.println("M2 sync method");
     }
     void m3()
    {
     System.out.println("M3 normal method");
     }
}
```



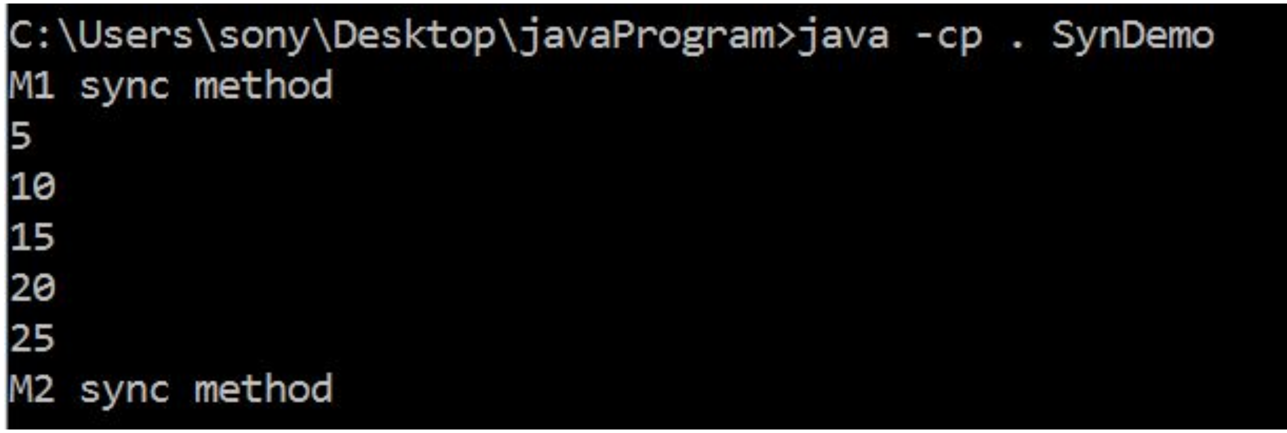
```java
class ThreadOne extends Thread
{
    SyncMethods s;
    ThreadOne(SyncMethods s)
    {
        this.s=s;
    }
    public void run()
    {
        s.m1(5);
    }
}
```



```java
class ThreadTwo extends Thread
{
    SyncMethods s;
    ThreadTwo(SyncMethods s)
    {
        this.s=s;
    }
    public void run()
    {
        s.m2();
    }
}
```

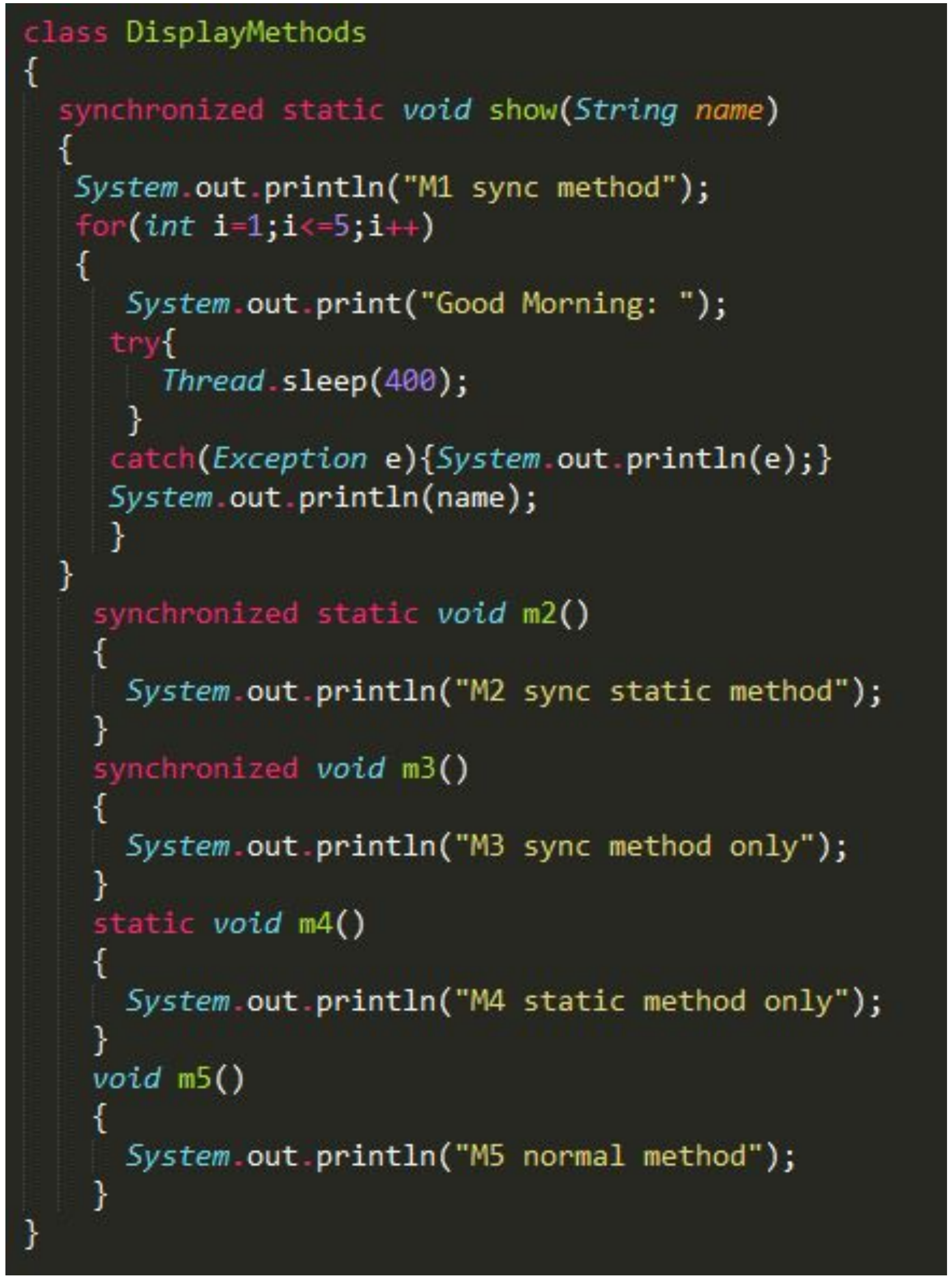


```java
class SynDemo
{
    public static void main(String[] args) {

        SyncMethods s = new SyncMethods();

        ThreadOne t1 = new ThreadOne(s);
        ThreadTwo t2 = new ThreadTwo(s);
        t1.start();
        t2.start();

    }
}
```



```
C:\Users\sony\Desktop\javaProgram>java -cp . SynDemo
M1 sync method
5
10
15
20
25
M2 sync method
```

Do one exercise



```
class DisplayMethods
{
  synchronized static void show(String name)
  {
   System.out.println("M1 sync method");
   for(int i=1;i<=5;i++)
   {
      System.out.print("Good Morning: ");
     try{
        Thread.sleep(400);
      }
     catch(Exception e){System.out.println(e);}
     System.out.println(name);
     }
  }
    synchronized static void m2()
    {
      System.out.println("M2 sync static method");
    }
    synchronized void m3()
    {
      System.out.println("M3 sync method only");
    }
    static void m4()
    {
      System.out.println("M4 static method only");
    }
    void m5()
    {
      System.out.println("M5 normal method");
    }
}
```

Synchronized block

Three ways

1. Get the lock of current object
2. Get a lock of particular object ‘b’
3. Get a class level lock ‘DisplayMethods.class’

Get the lock of current object

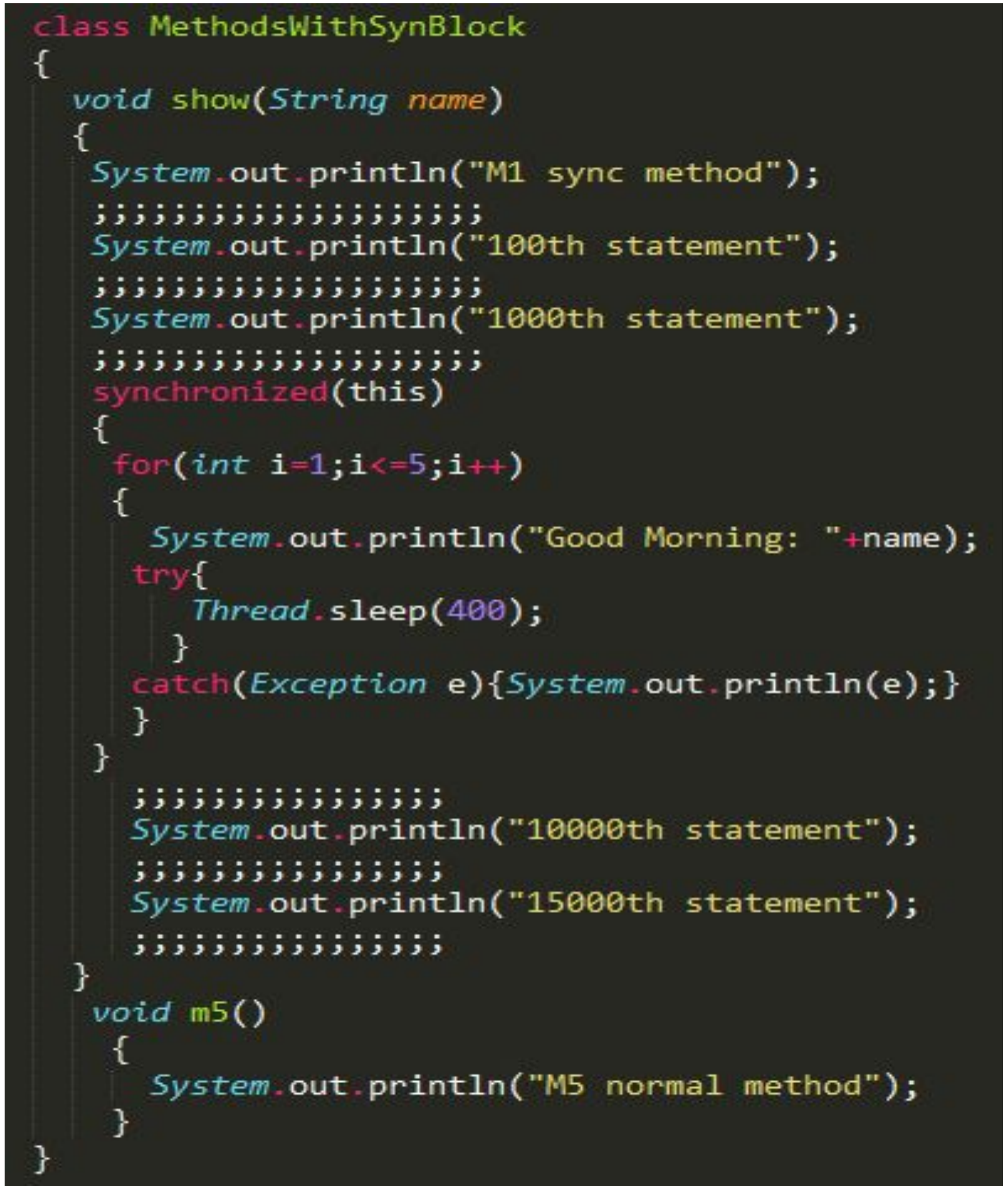
```
synchronized(this)
{
}
```

Get a lock of particular object ‘b’

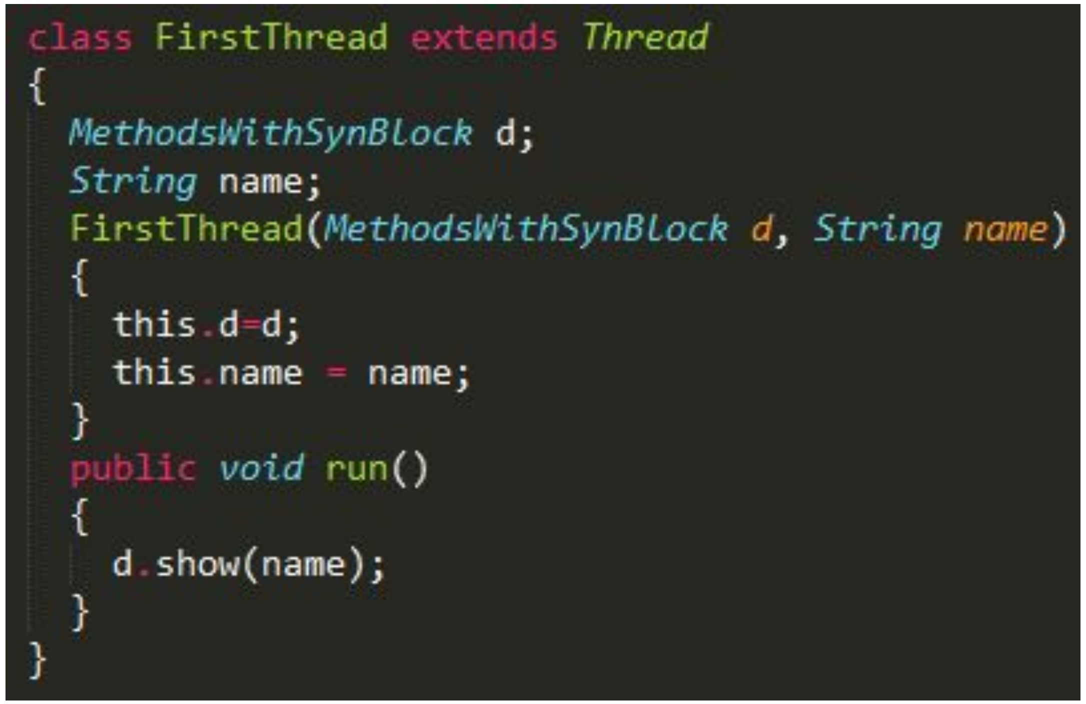
```
synchronized(b)
{
}
```

Get a class level lock ‘DisplayMethods.class’

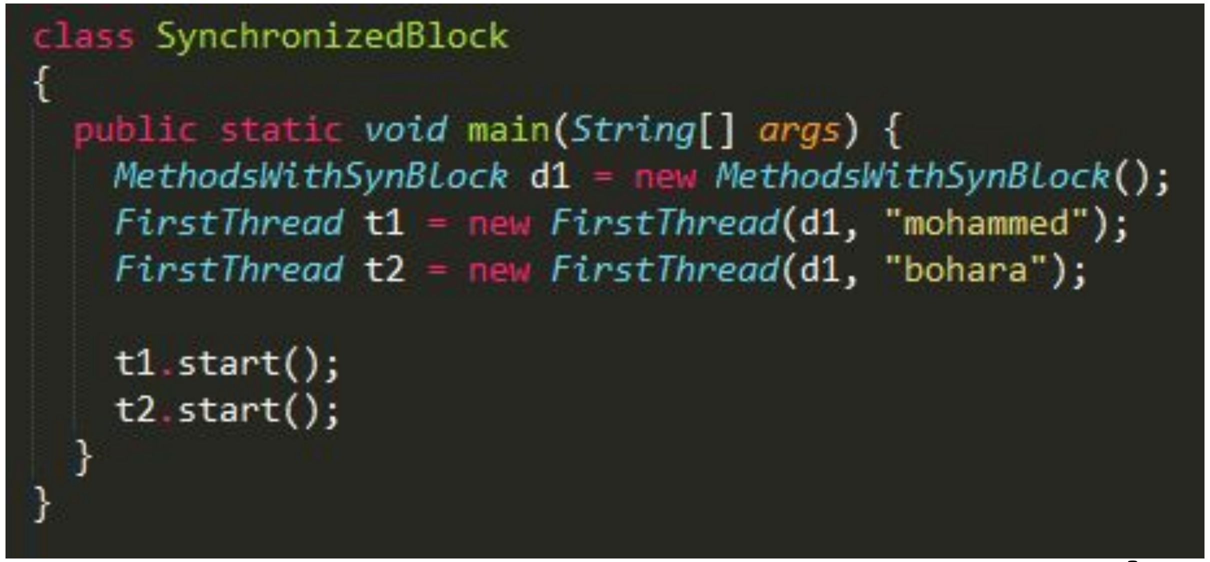
```
synchronized(DisplayMethods.class)
{
}
```



```
class MethodsWithSynBlock
{
  void show(String name)
  {
   System.out.println("M1 sync method");
   ;;;;;;;;;;;;;;;;;;;;
   System.out.println("100th statement");
   ;;;;;;;;;;;;;;;;;;;;
   System.out.println("1000th statement");
   ;;;;;;;;;;;;;;;;;;;;
   synchronized(this)
   {
    for(int i=1;i<=5;i++)
    {
     System.out.println("Good Morning: "+name);
     try{
       Thread.sleep(400);
       }
     catch(Exception e){System.out.println(e);}
     }
   }
     ;;;;;;;;;;;;;;;;
     System.out.println("10000th statement");
     ;;;;;;;;;;;;;;;;
     System.out.println("15000th statement");
     ;;;;;;;;;;;;;;;;
  }
   void m5()
    {
     System.out.println("M5 normal method");
    }
}
```



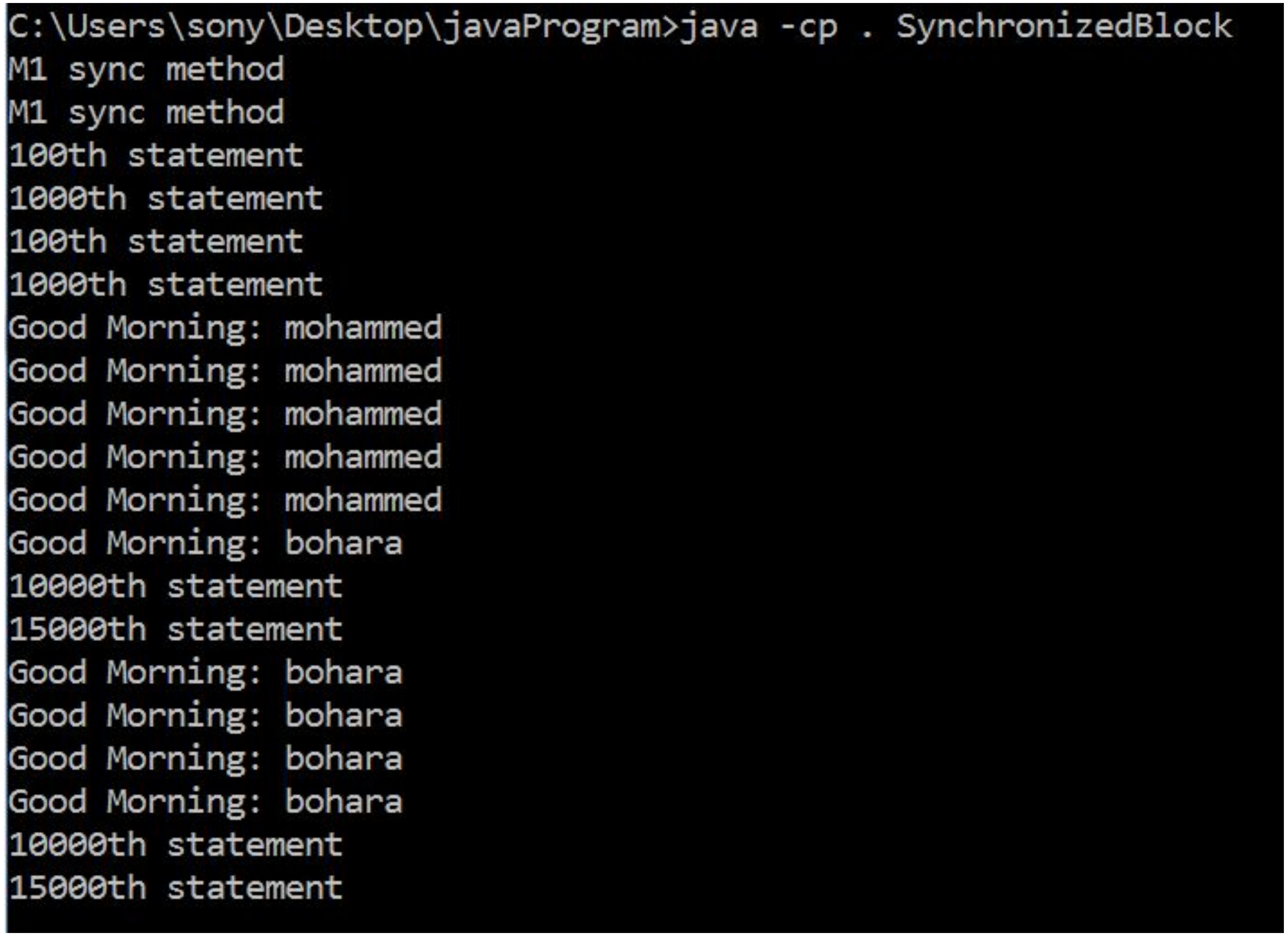
```
class FirstThread extends Thread
{
  MethodsWithSynBlock d;
  String name;
  FirstThread(MethodsWithSynBlock d, String name)
  {
    this.d=d;
    this.name = name;
  }
  public void run()
  {
    d.show(name);
  }
}
```



```
class SynchronizedBlock
{
  public static void main(String[] args) {
    MethodsWithSynBlock d1 = new MethodsWithSynBlock();
    FirstThread t1 = new FirstThread(d1, "mohammed");
    FirstThread t2 = new FirstThread(d1, "bohara");

    t1.start();
    t2.start();
  }
}
```

Example

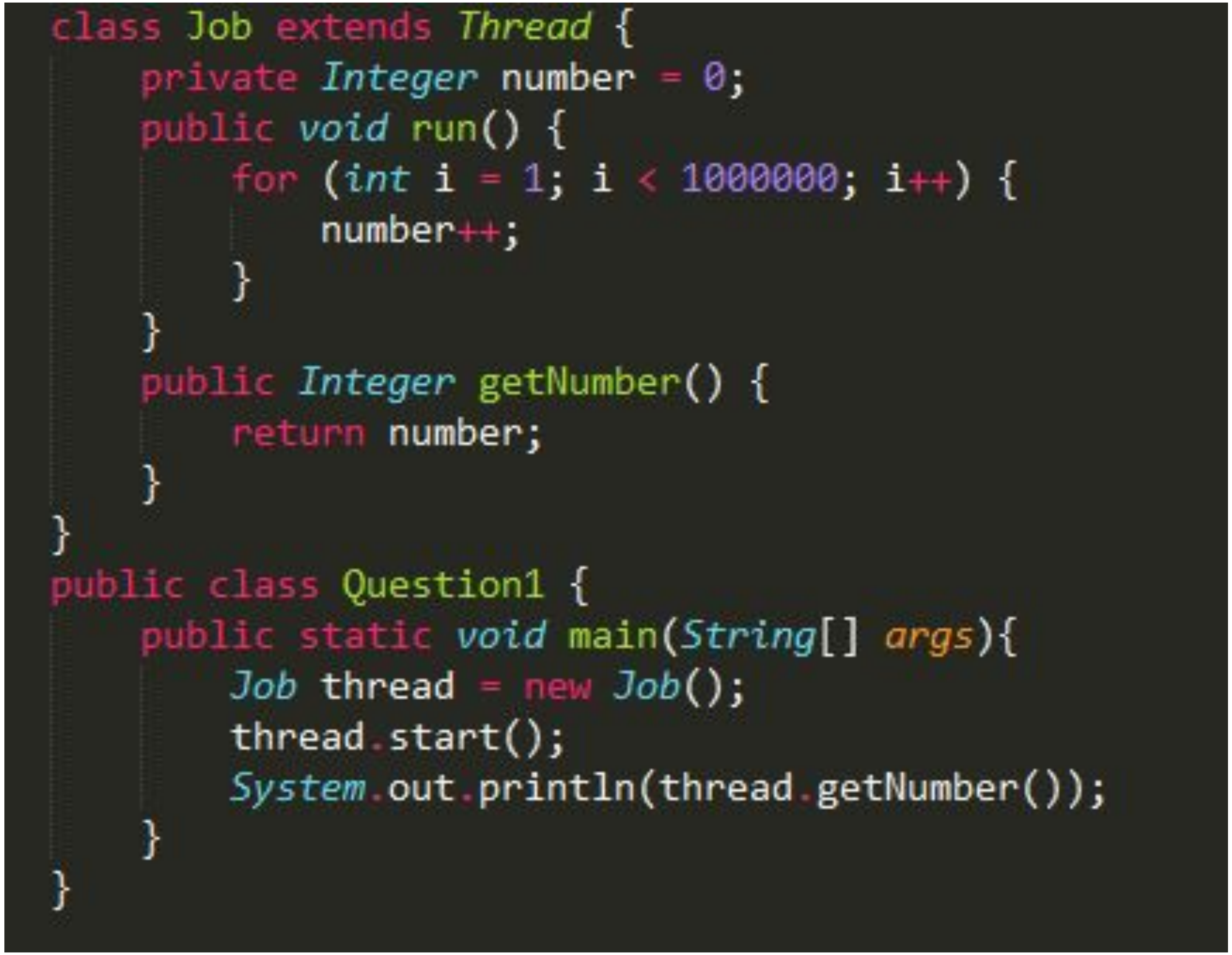
Output



```
C:\Users\sony\Desktop\javaProgram>java -cp . SynchronizedBlock
M1 sync method
M1 sync method
100th statement
1000th statement
100th statement
1000th statement
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: mohammed
Good Morning: bohara
10000th statement
15000th statement
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
Good Morning: bohara
10000th statement
15000th statement
```

Any Question

Q-1



```
class Job extends Thread {
    private Integer number = 0;
    public void run() {
        for (int i = 1; i < 1000000; i++) {
            number++;
        }
    }
    public Integer getNumber() {
        return number;
    }
}
public class Question1 {
    public static void main(String[] args){
        Job thread = new Job();
        thread.start();
        System.out.println(thread.getNumber());
    }
}
```

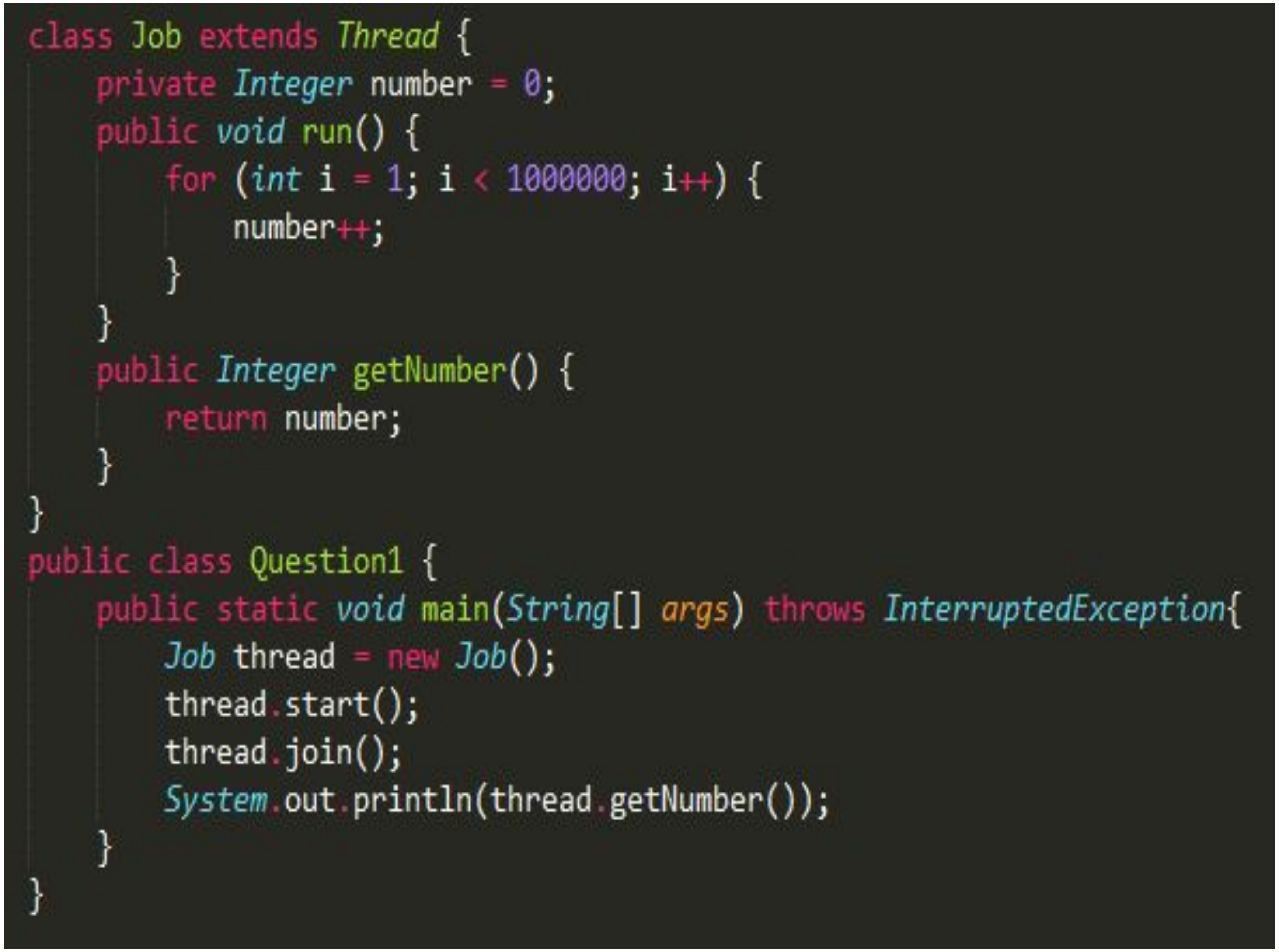
What is the output?

A. It prints 0.

B. It prints 999999.

C. The output is not guaranteed to be any of the above.

Q-2



```
class Job extends Thread {
    private Integer number = 0;
    public void run() {
        for (int i = 1; i < 1000000; i++) {
            number++;
        }
    }
    public Integer getNumber() {
        return number;
    }
}
public class Question1 {
    public static void main(String[] args) throws InterruptedException{
        Job thread = new Job();
        thread.start();
        thread.join();
        System.out.println(thread.getNumber());
    }
}
```

What is the output?

A. It prints 0.

B. It prints 999999.

C. The output is not guaranteed to be any of the above.

Q-3 **What is the output?**

```
public class Threads3 implements Runnable {
    public void run() {
        System.out.print("running");
    }
    public static void main(String[] args) {
        Thread t = new Thread(new Threads3());
        t.run();
        t.run();
        t.start();
    }
}
```

A. Compilation fails.

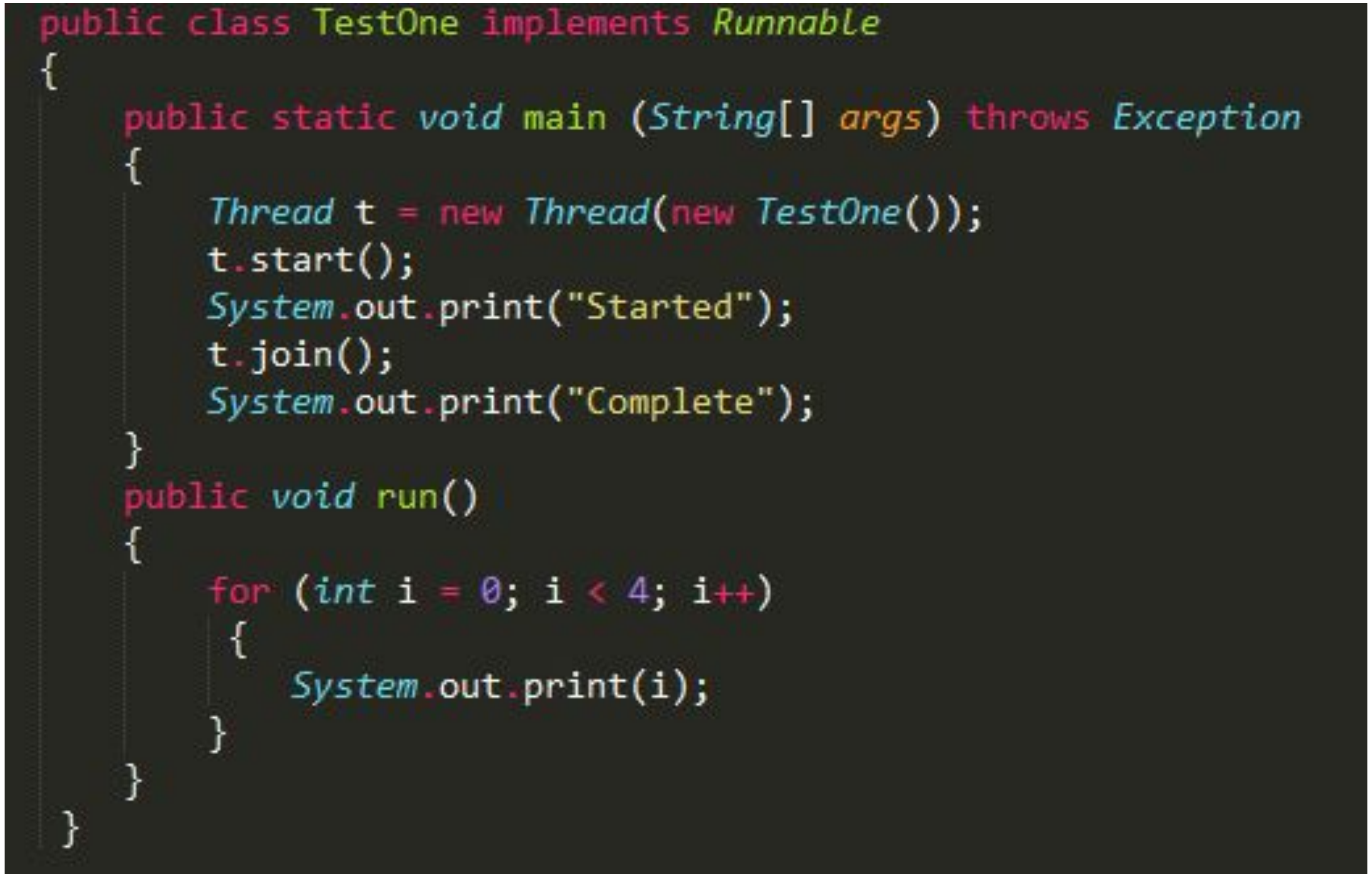
B. An exception is thrown at runtime.

C. The code executes and prints "running".

D. The code executes and prints "runningrunning".

E. The code executes and prints "runningrunningrunning".

Q-4 **What is the output?**



```
public class TestOne implements Runnable
{
    public static void main (String[] args) throws Exception
    {
        Thread t = new Thread(new TestOne());
        t.start();
        System.out.print("Started");
        t.join();
        System.out.print("Complete");
    }
    public void run()
    {
        for (int i = 0; i < 4; i++)
         {
            System.out.print(i);
         }
    }
 }
```

A. Compilation fails.

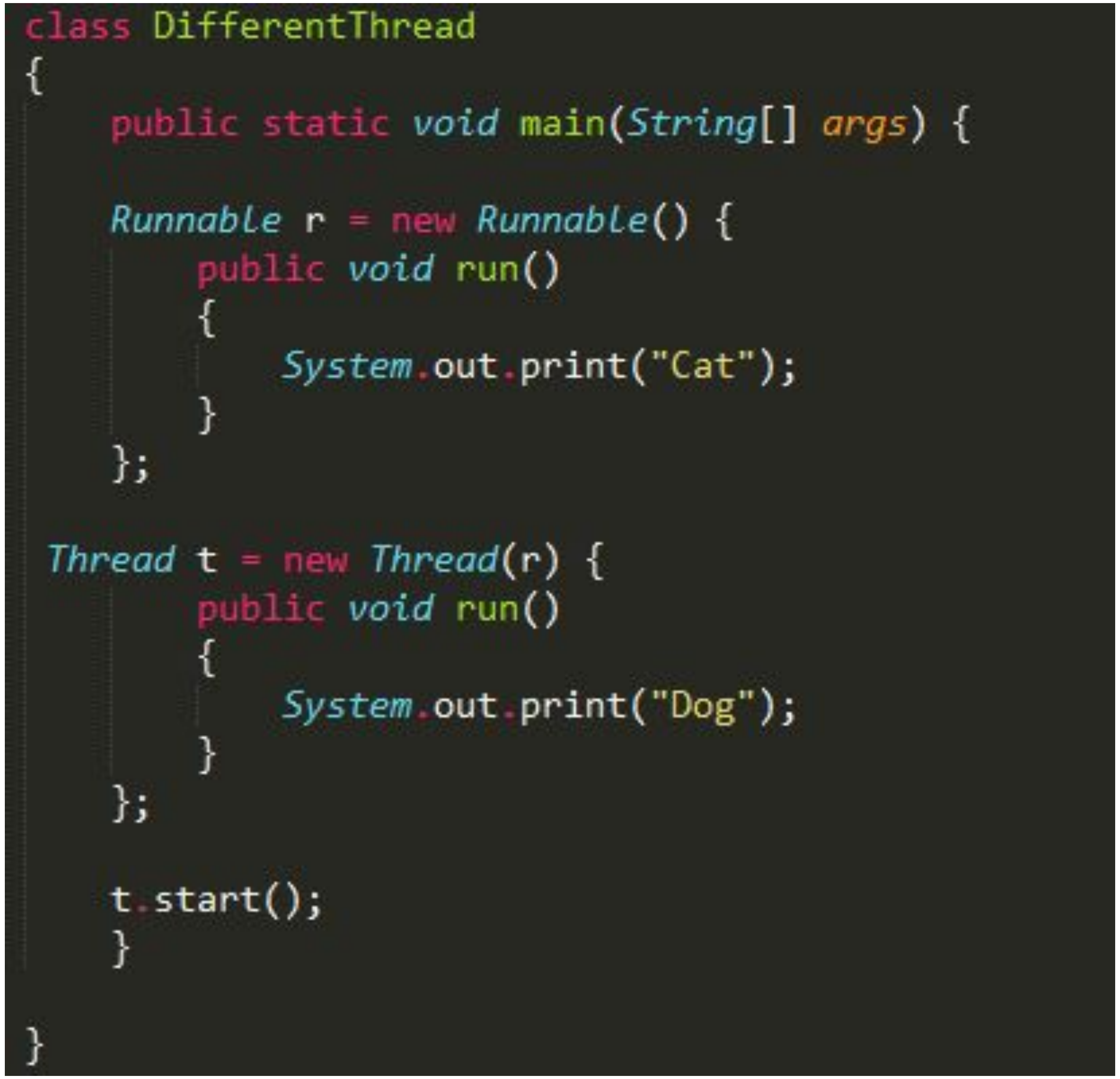
B. An exception is thrown at runtime.

C. The code executes and prints "StartedComplete".

D. The code executes and prints "StartedComplete0123".

E. The code executes and prints "Started0123Complete".

Q-5 **What is the output?**



```
class DifferentThread
{
    public static void main(String[] args) {

    Runnable r = new Runnable() {
        public void run()
        {
            System.out.print("Cat");
        }
    };

 Thread t = new Thread(r) {
        public void run()
        {
            System.out.print("Dog");
        }
    };

    t.start();
    }

}
```

A. Cat

B. Dog

C. Compilation fails.

D. The code runs with no output.

E. An exception is thrown at runtime.