OBJECT ORIENTED PROGRAMMING USING JAVA

OUTLINE

- Multithreading in java
- Thread
- Scheduler
- Thread lifecycle

WHAT IS MULTITHREADING??

- Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.
- Each part of such program is called a thread. So, threads are light-weight processes within a process.
- Each of the threads can run in parallel.

APPLICATION OF MULTITHREADING?

- Mobile application usage and updation.
- Browsing displaying images in webpages.
- Websites displaying ads.
- Webcrawlers.

MOTIVATION OF MULTITHREADING?

- We know sequential programming
  - Certain task block. E.g. read() in I/O
    - Blocking halts CPU time thus wasting CPU time.

Solution: Concurrent programming

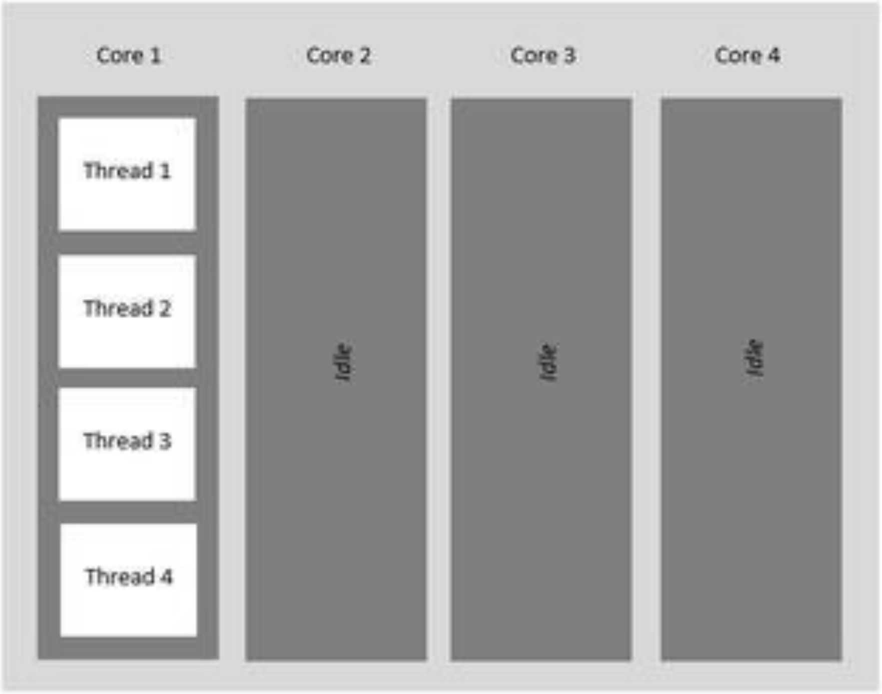
Why???

Helps in better recourse utilization and throughput.

“But Concurrent programming is complex”

EXAMPLE:

Sequential Stream



Core 1

Thread 1

Thread 2

Thread 3

Thread 4

Core 2

Idle

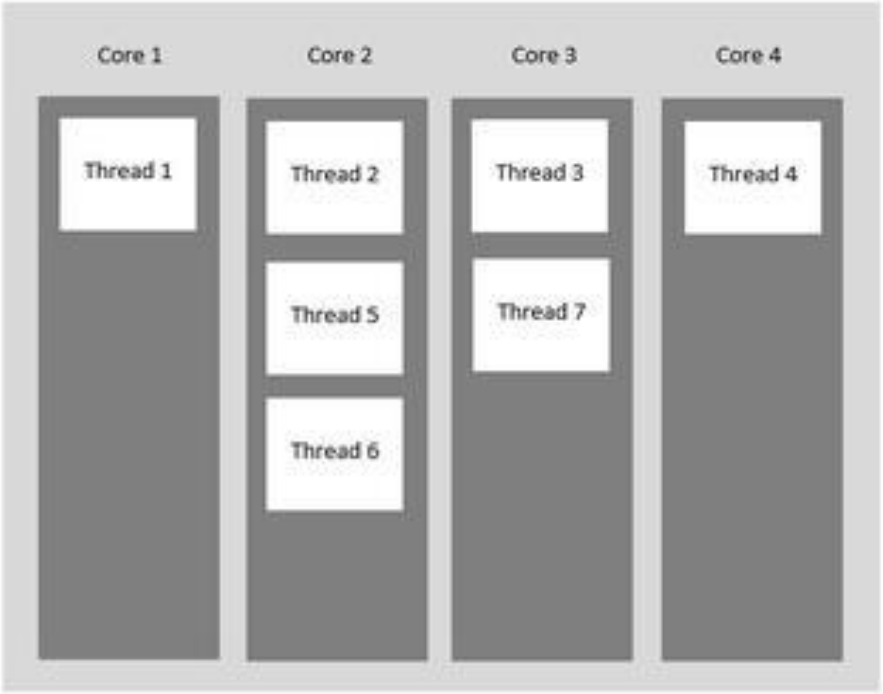
Core 3

Idle

Core 4

Idle

Parallel Stream



Core 1

Thread 1

Core 2

Thread 2

Thread 5

Thread 6

Core 3

Thread 3

Thread 7

Core 4

Thread 4

THREADS

- Post-os era- computers ran a single program.
- OS- multiple processes to run on concurrent
  - Multitasking : switching CPU from processes to another(illusion of parallelism: does not imply parallelism-)

“Better resource utilization”

“ Better resource utilization leads to the development of threads”

THREADS

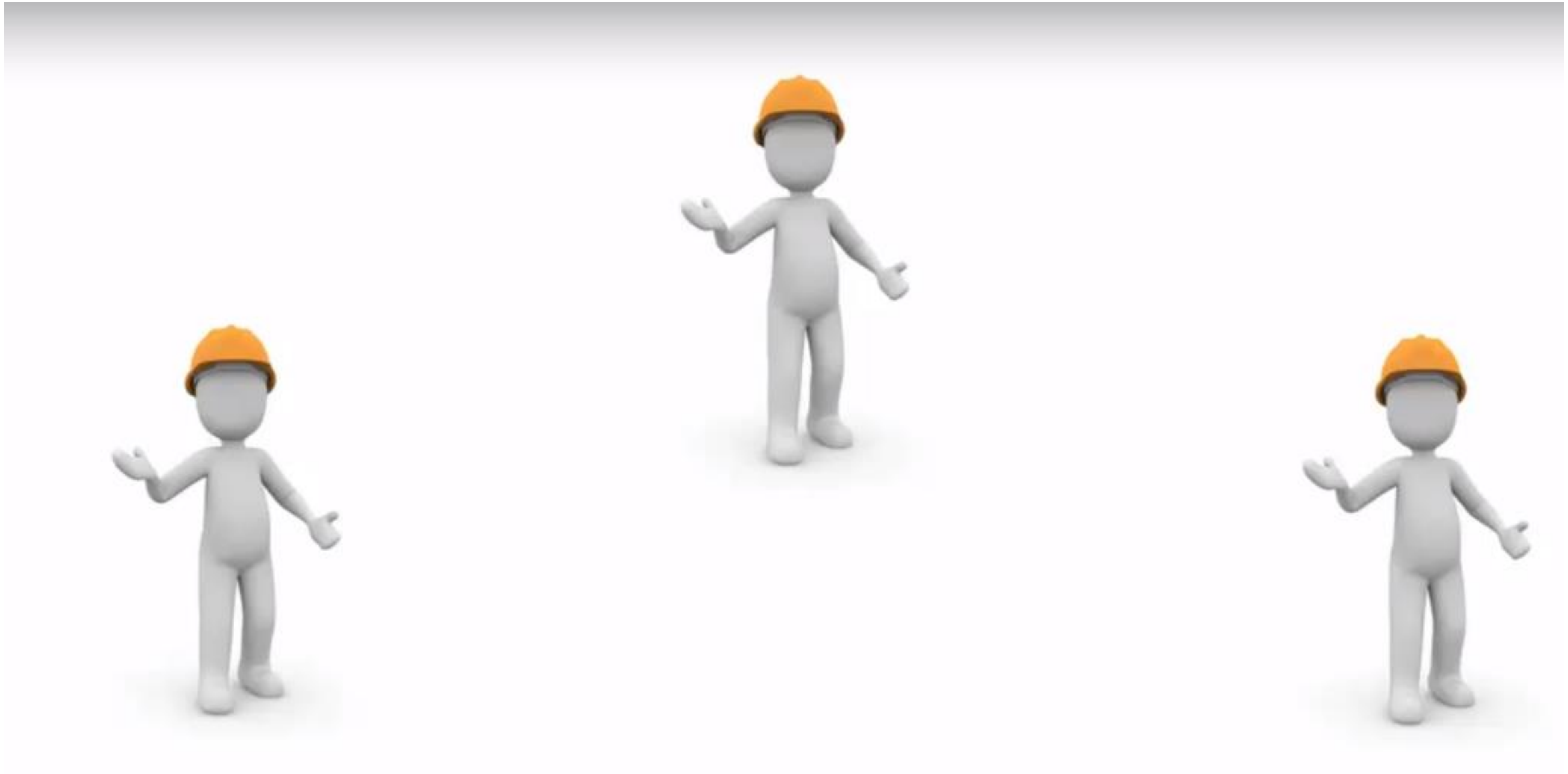
- Single sequential flow of control within a process.
- Light weight processes.
- A process can have multiple threads.
- Threads share processes wide recourses e.g. Memory.
- Threads has its own PC, Stack and local variable.

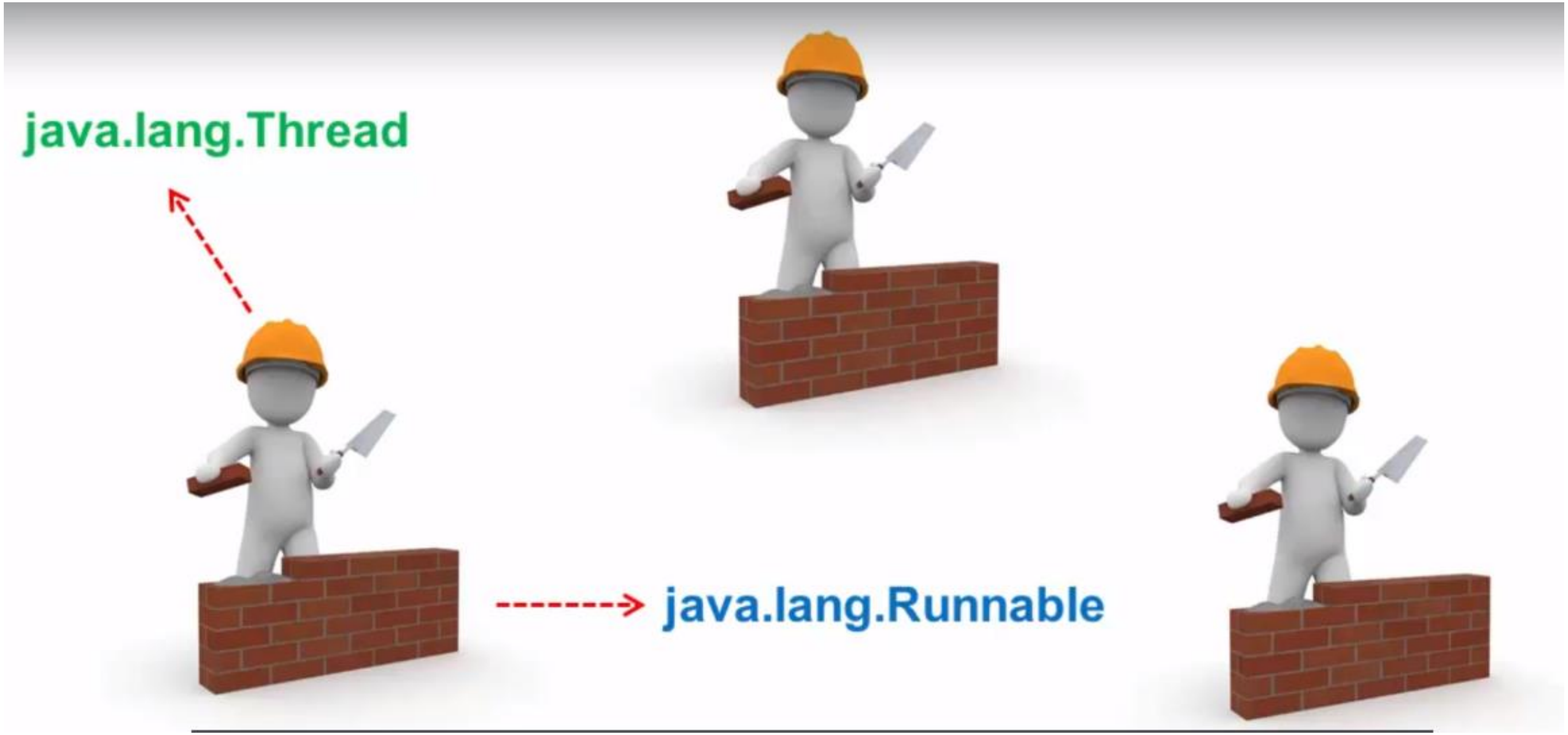
BENEFITS OF THREADS

- Exploiting multiple processors (thus achieving parallelism);
- Always loosely coupled designs.
- Better throughput even in single CPU machines.

THREADS TYPES

- Daemon threads
    - Background threads which are useful for tasks such as garbage collection
- Non-Daemon
  - Created within application
  - Main thread: created by JVM to run main()
  - JVM will not terminate if at least one non-daemon thread is running





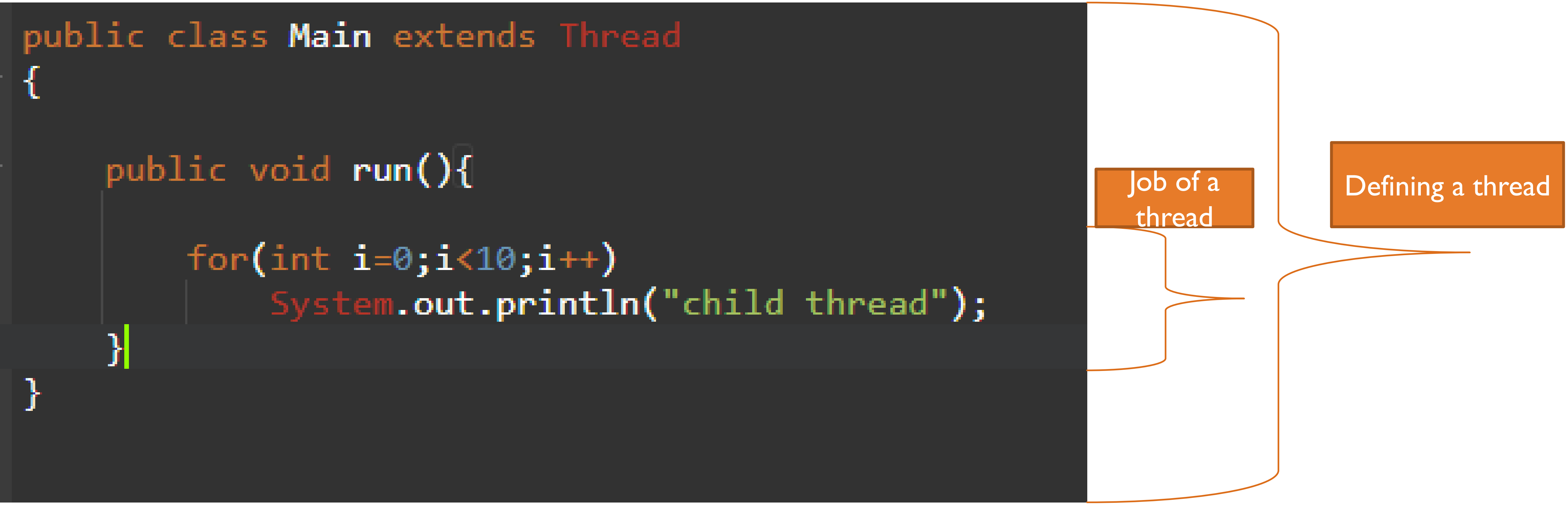
java.lang.Thread

java.lang.Runnable

LAUNCHING A THREAD

- **Case 1: By extending thread class**
- **Case 2: By implementing runnable interface**
- Create Task
  - Runnable task=new MyRunnable();
  - Runnable has exactly one method run()
- Creating a thread with task
  - Thread thread=new Thread(task); // NEW
- Start Thread
  - Thread.start(); //Runnable
  - New call stack with run() is created for thread

CASE 1: BY EXTENDING THREAD CLASS



```java
public class Main extends Thread
{

    public void run(){

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

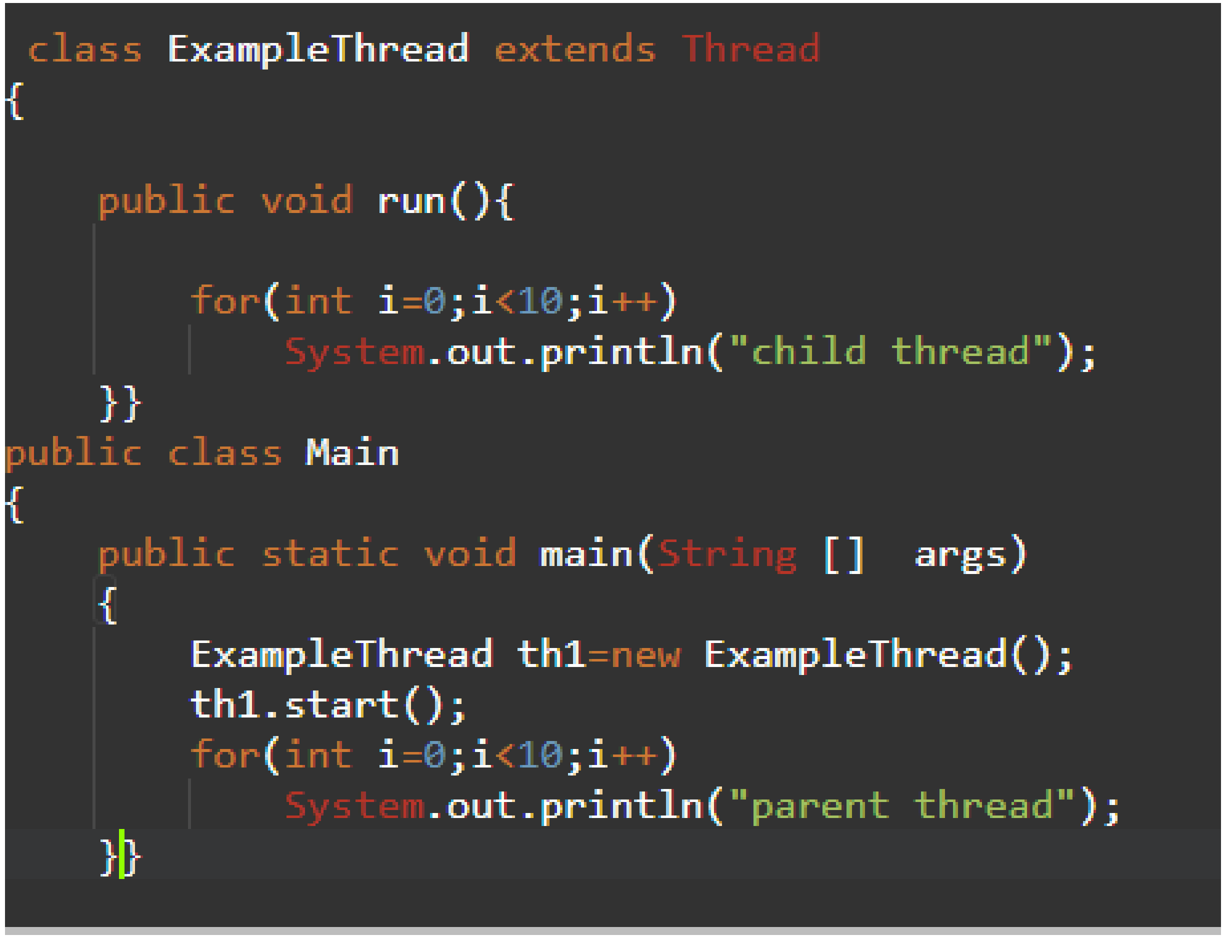
Job of a thread

Defining a thread

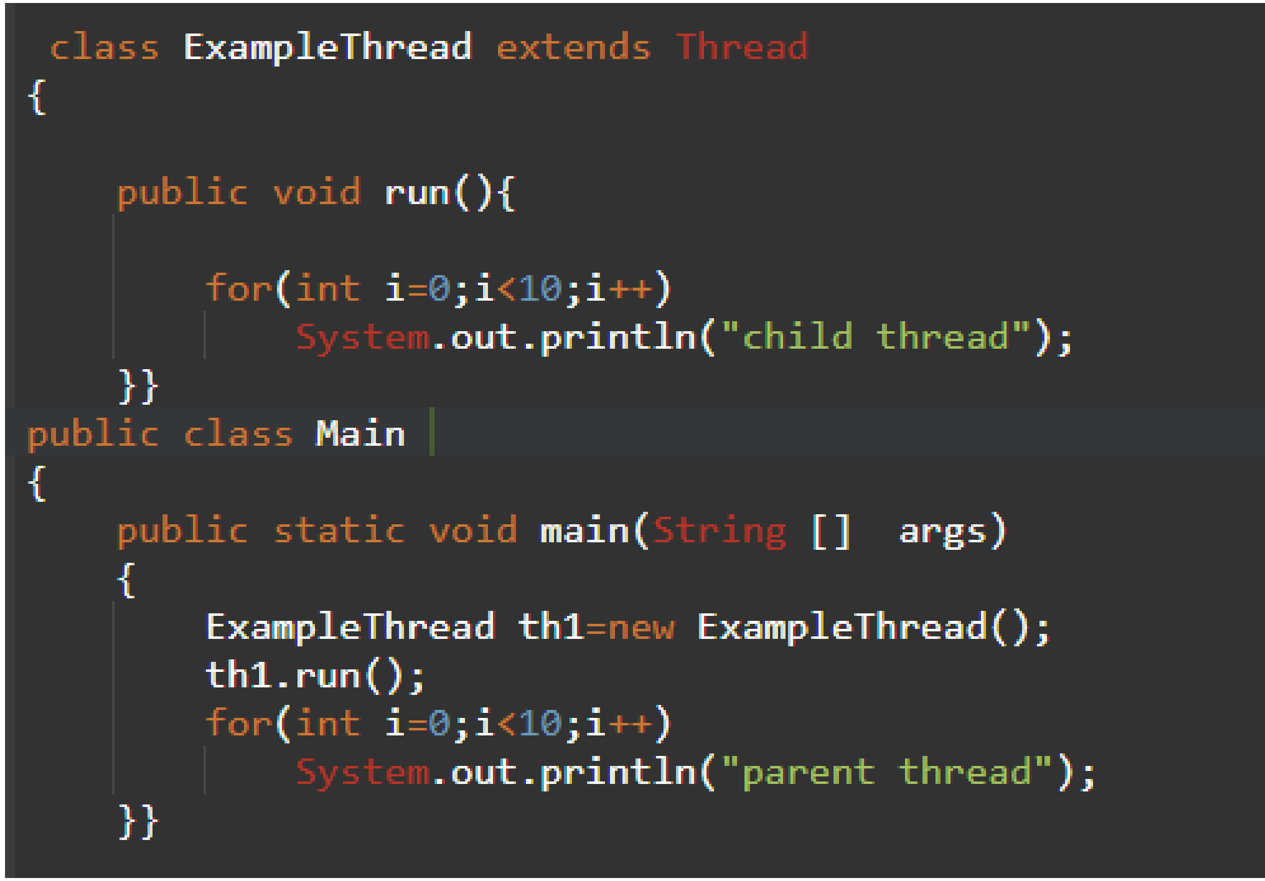
THREAD SCHEDULER

- It is the part of jvm.
- It is responsible to schedule threads.
- If multiple threads are waiting to get a chance of execution , in which order threads are executed is done by thread scheduler.
- We cant expect exact algorithm followed by thread scheduler it is varied from jvm to jvm, hence we cant expect thread execution order and exact order.
- Hence whenever there is no guarantee of exact output, but we can provide all possible outputs.

DIFFERENCE BETWEEN t.start() vs t.run()

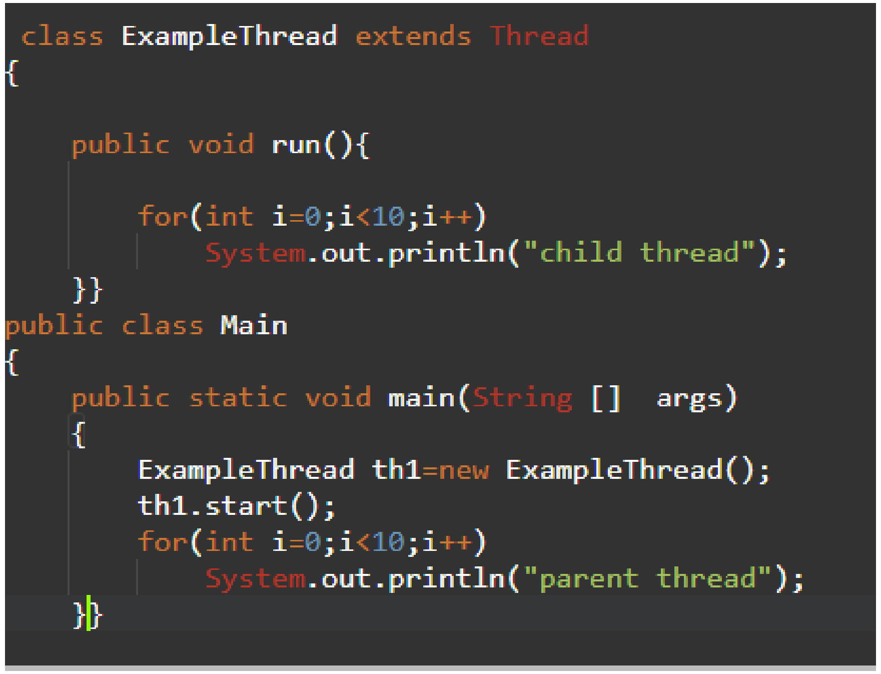


```
 class ExampleThread extends Thread
{

    public void run(){

        for(int i=0;i<10;i++)
            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        ExampleThread th1=new ExampleThread();
        th1.start();
        for(int i=0;i<10;i++)
            System.out.println("parent thread");
    }}
```



```
 class ExampleThread extends Thread
{

    public void run(){

        for(int i=0;i<10;i++)
            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        ExampleThread th1=new ExampleThread();
        th1.run();
        for(int i=0;i<10;i++)
            System.out.println("parent thread");
    }}
```

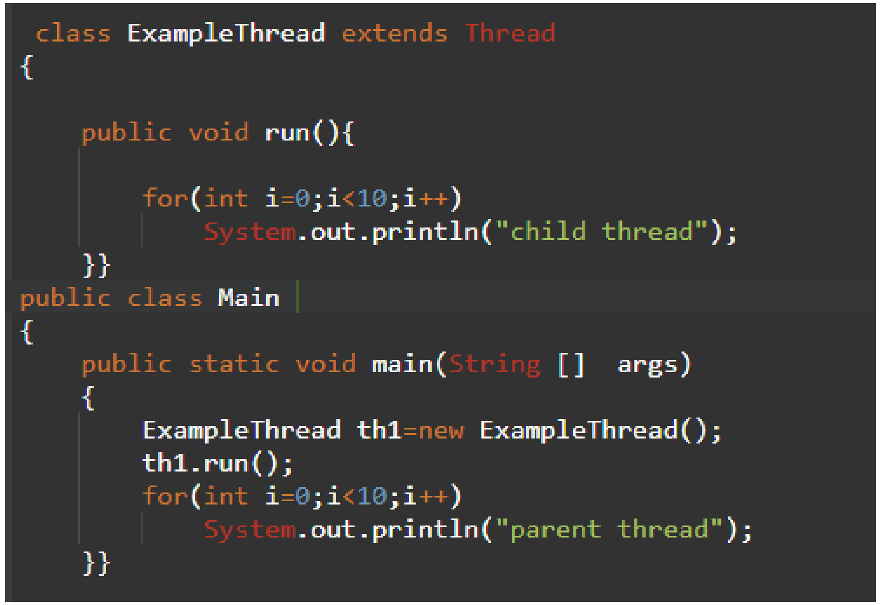
DIFFERENCE BETWEEN t.start() vs t.run()



```
 class ExampleThread extends Thread
{

    public void run(){

        for(int i=0;i<10;i++)
            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        ExampleThread th1=new ExampleThread();
        th1.start();
        for(int i=0;i<10;i++)
            System.out.println("parent thread");
    }}
```

Number of thread: 2

- Here a new thread is created which is responsible for the execution of run method
- Overloading not possible.
- It is mandatory to override run() method. Otherwise don’t use thread.

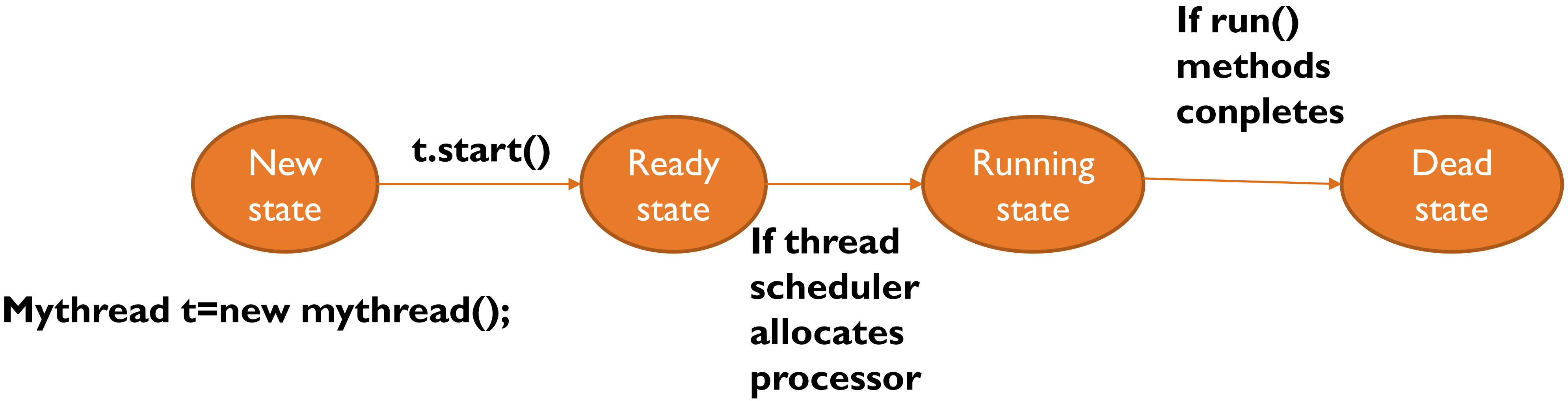


```
 class ExampleThread extends Thread
{

    public void run(){

        for(int i=0;i<10;i++)
            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        ExampleThread th1=new ExampleThread();
        th1.run();
        for(int i=0;i<10;i++)
            System.out.println("parent thread");
    }}
```

Number of thread: 1

- Here a new thread is not created and is executed in a normal method call.
- Overloading possible.

THREAD LIFECYCLE



New state

t.start()

Ready state

If thread scheduler allocates processor

Running state

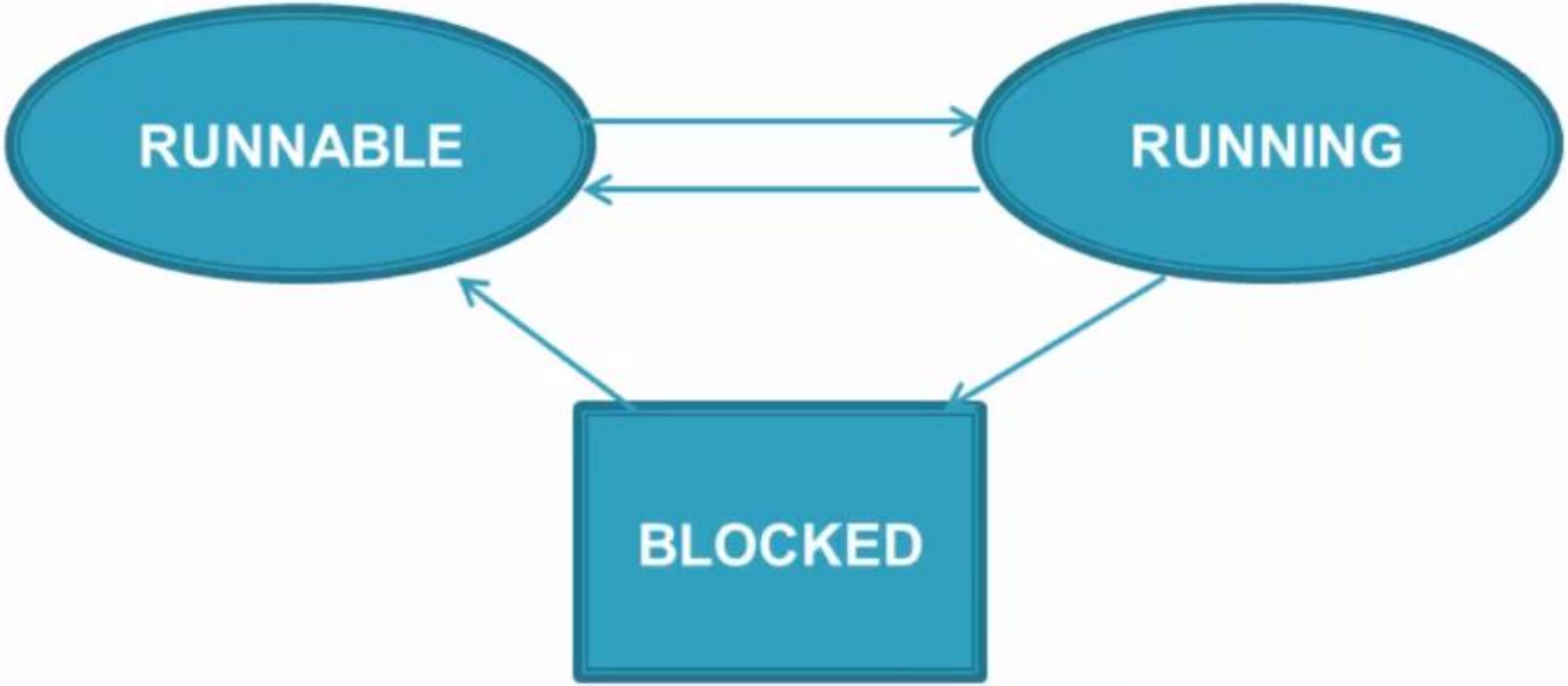
If run() methods conpletes

Dead state

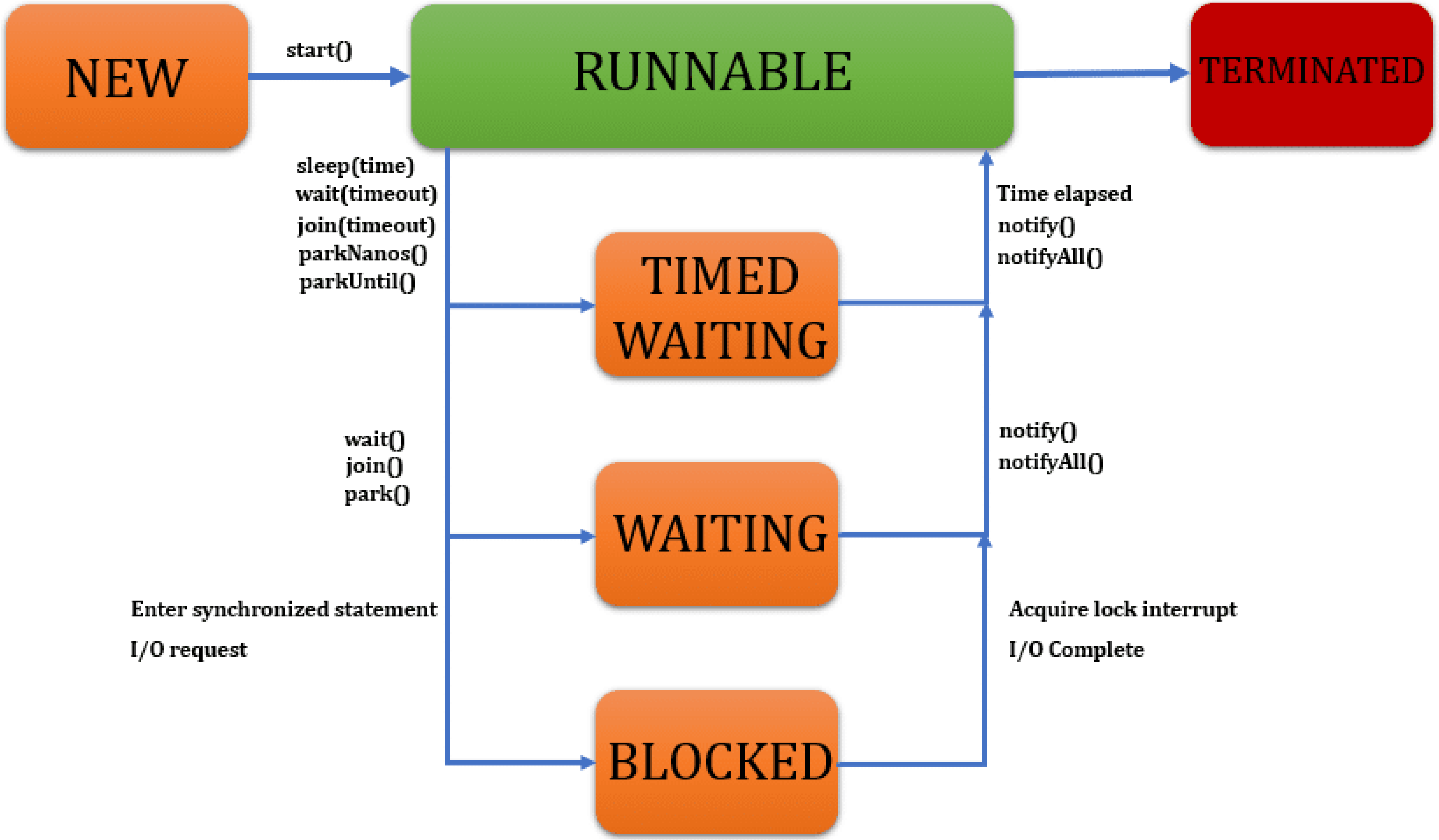
Mythread t=new mythread();

Note: after starting a thread if we again start a thread we will get a RE: illegalthreadstateelements

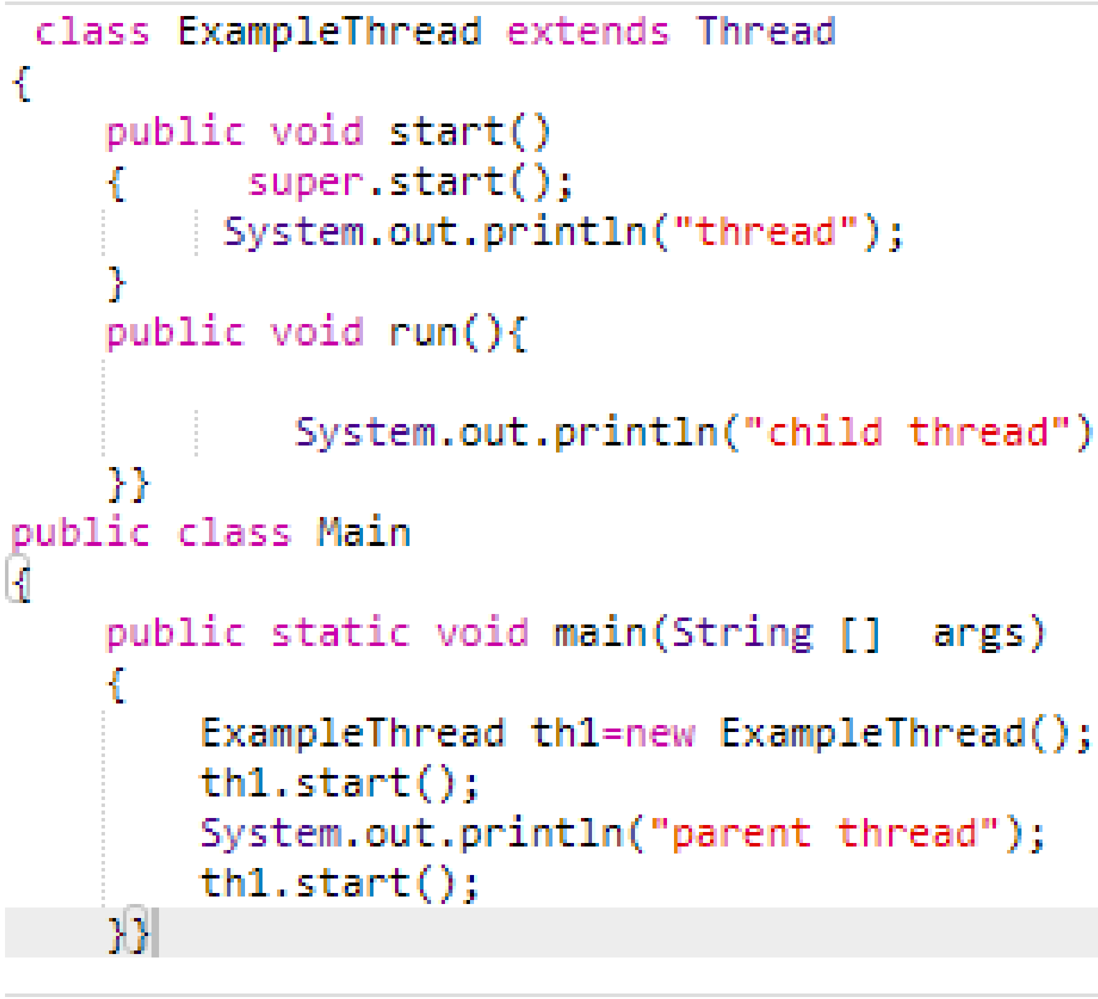
THREAD SCHEDULER



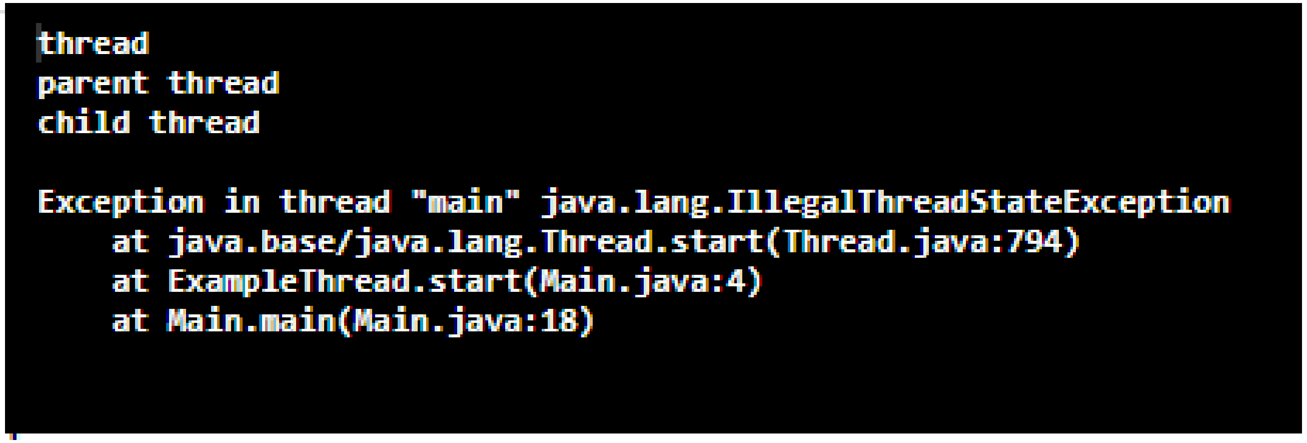
THREAD LIFECYCLE



EXAMPLE

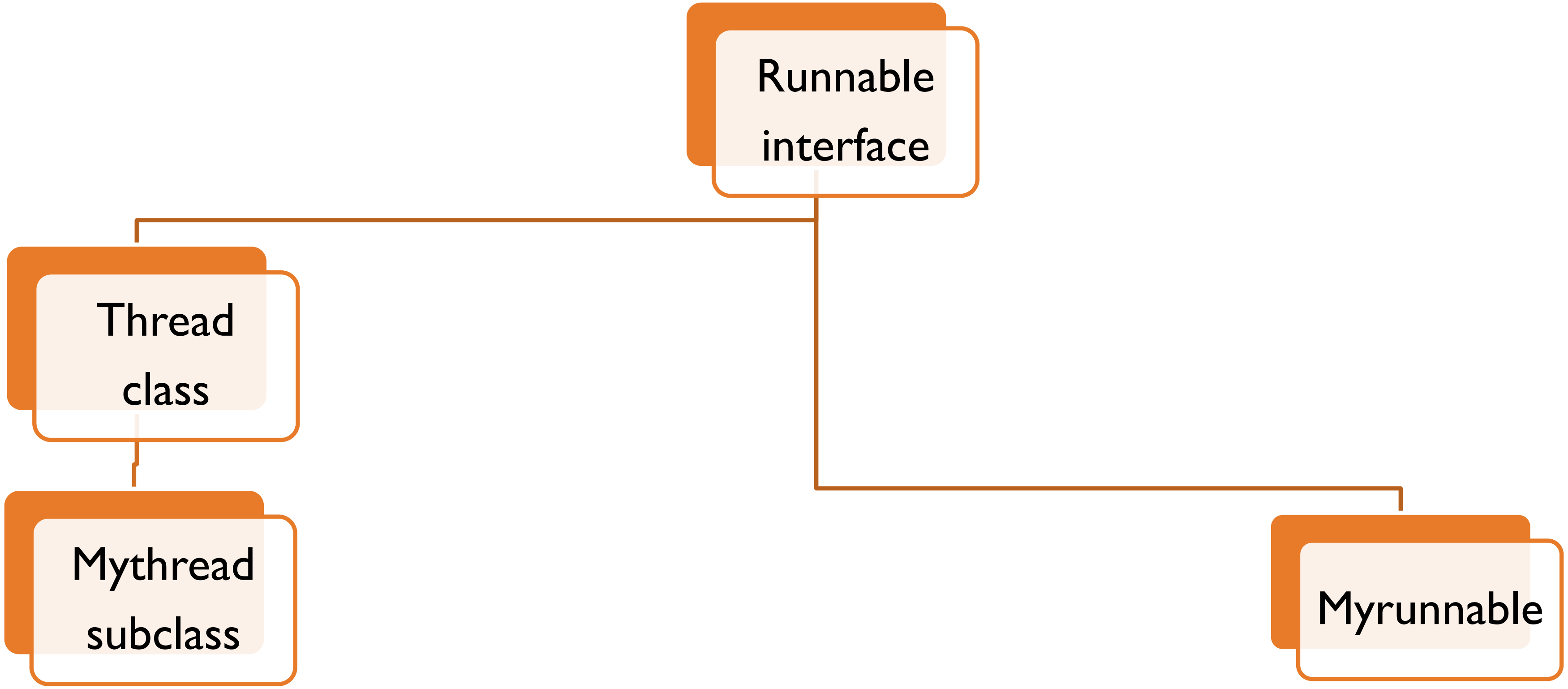


```
 class ExampleThread extends Thread
{
    public void start()
    {     super.start();
         System.out.println("thread");
    }
    public void run(){

            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        ExampleThread th1=new ExampleThread();
        th1.start();
        System.out.println("parent thread");
        th1.start();
    }}
```



```
thread
parent thread
child thread

Exception in thread "main" java.lang.IllegalThreadStateException
	at java.base/java.lang.Thread.start(Thread.java:794)
	at ExampleThread.start(Main.java:4)
	at Main.main(Main.java:18)
```

DEFINE A THREAD BY RUNNABLE INTERFACE



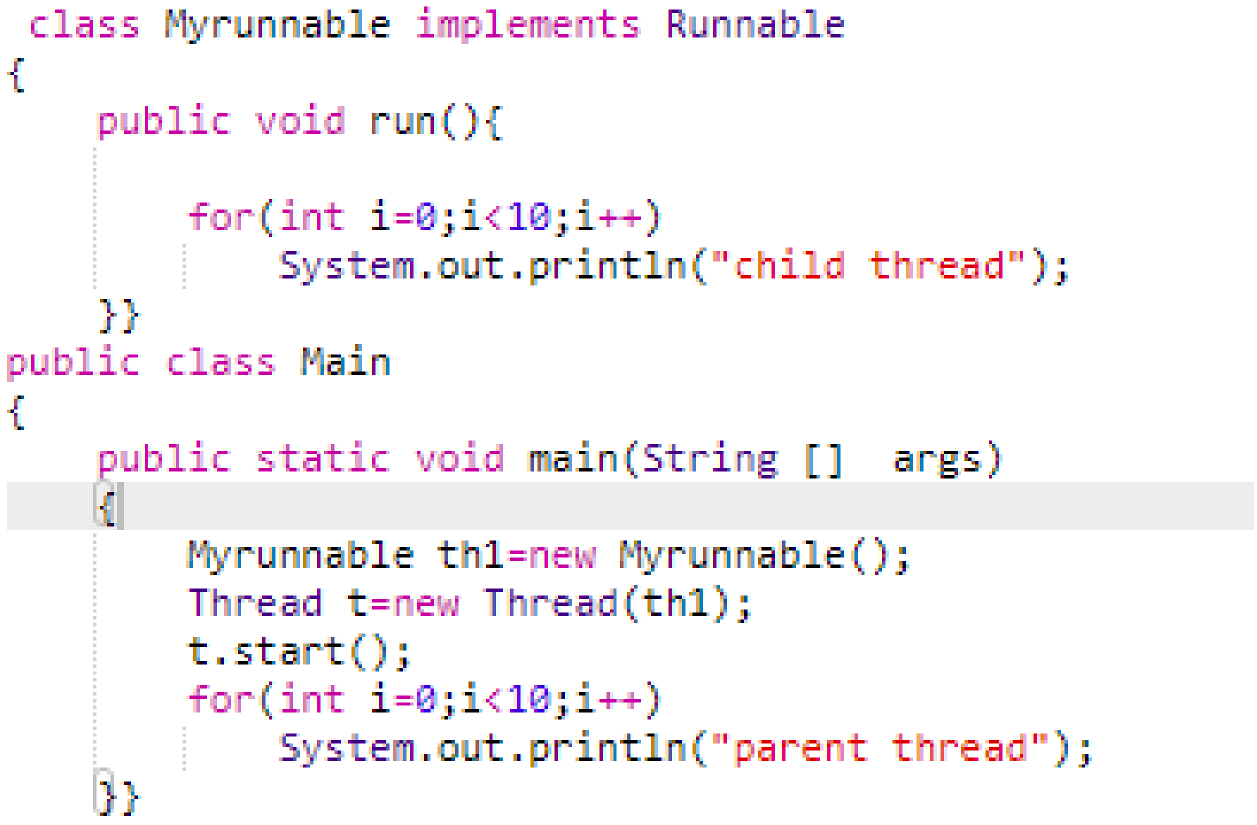
Runnable interface

Thread class

Mythread subclass

Myrunnable

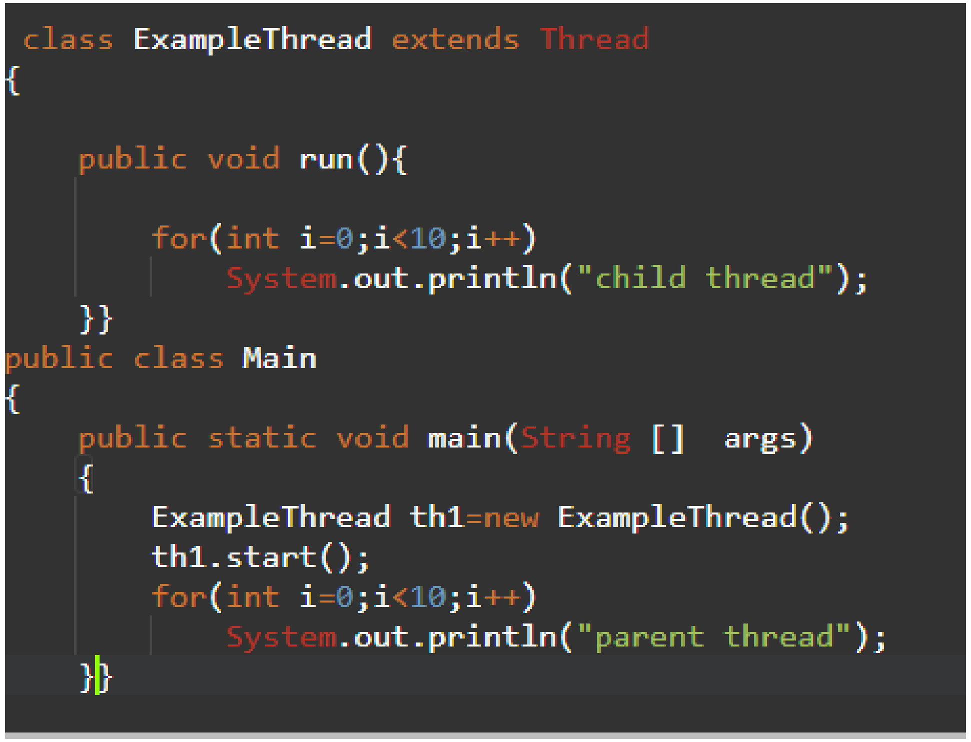
DEFINE A THREAD BY RUNNABLE INTERFACE



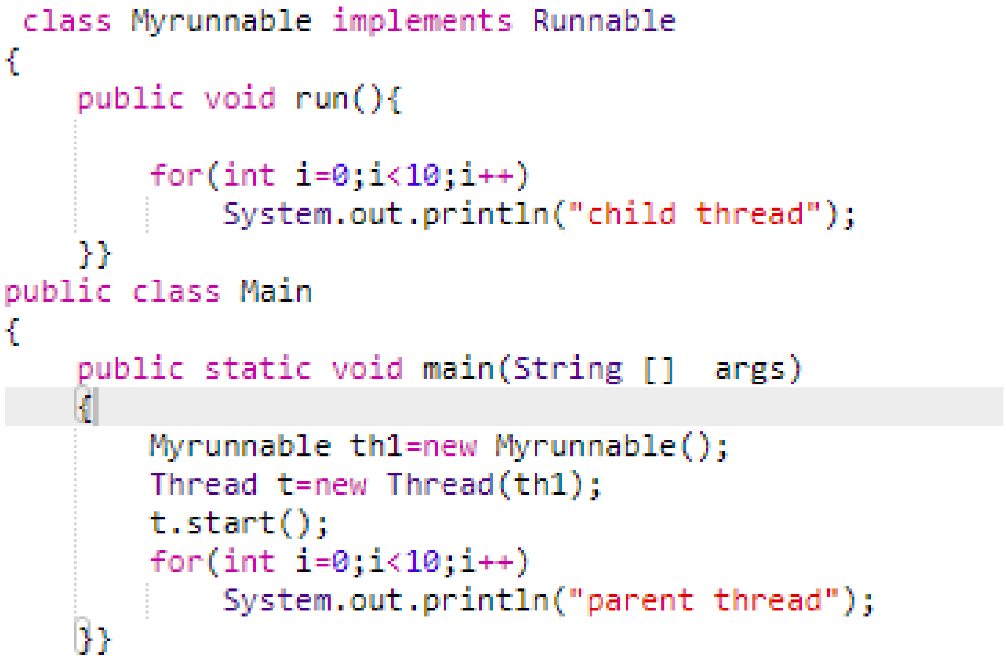
```
 class Myrunnable implements Runnable
{
    public void run(){

        for(int i=0;i<10;i++)
            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        Myrunnable th1=new Myrunnable();
        Thread t=new Thread(th1);
        t.start();
        for(int i=0;i<10;i++)
            System.out.println("parent thread");
    }}
```

WHICH APPROACH WE SHOULD WE PREFER/RECOMMEND??

- **Case 1: By extending thread class**
- **Case 2: By implementing runnable interface**



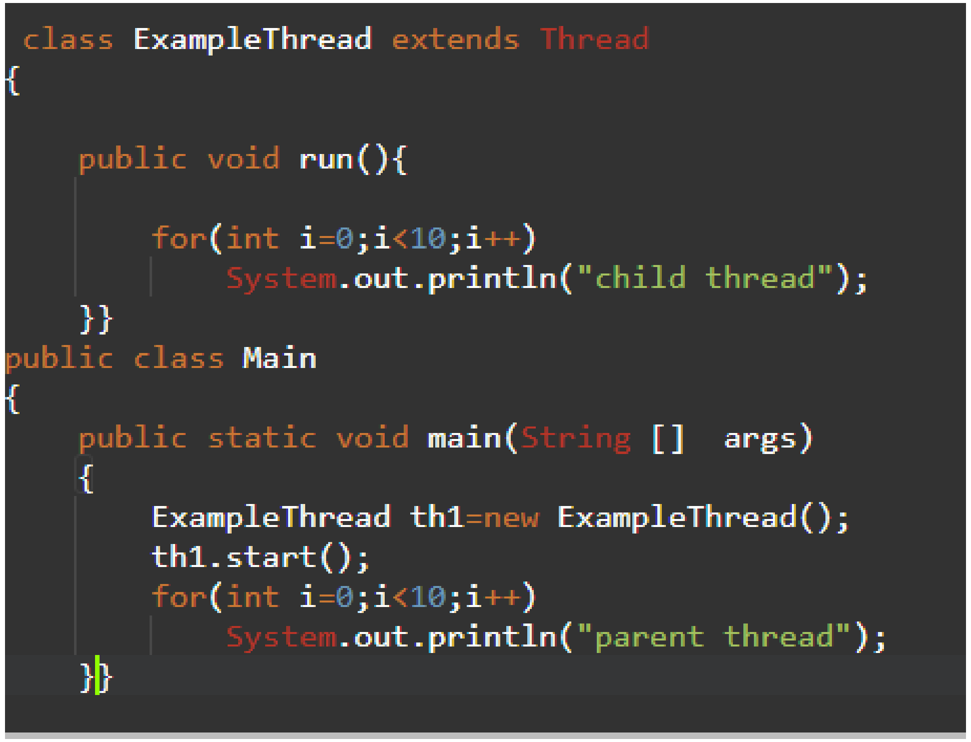
```
 class ExampleThread extends Thread
{

    public void run(){

        for(int i=0;i<10;i++)
            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        ExampleThread th1=new ExampleThread();
        th1.start();
        for(int i=0;i<10;i++)
            System.out.println("parent thread");
    }}
```



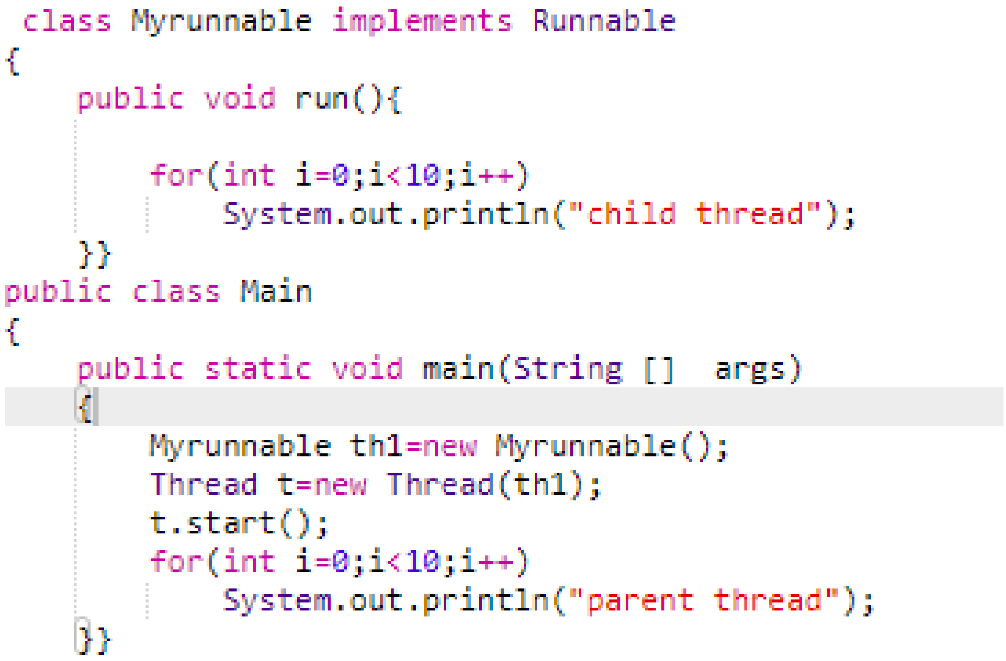
```
 class Myrunnable implements Runnable
{
    public void run(){

        for(int i=0;i<10;i++)
            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        Myrunnable th1=new Myrunnable();
        Thread t=new Thread(th1);
        t.start();
        for(int i=0;i<10;i++)
            System.out.println("parent thread");
    }}
```

WHICH APPROACH WE SHOULD WE PREFER/RECOMMEND??

- **Case 1: By extending thread class**
- **Case 2: By implementing runnable interface**



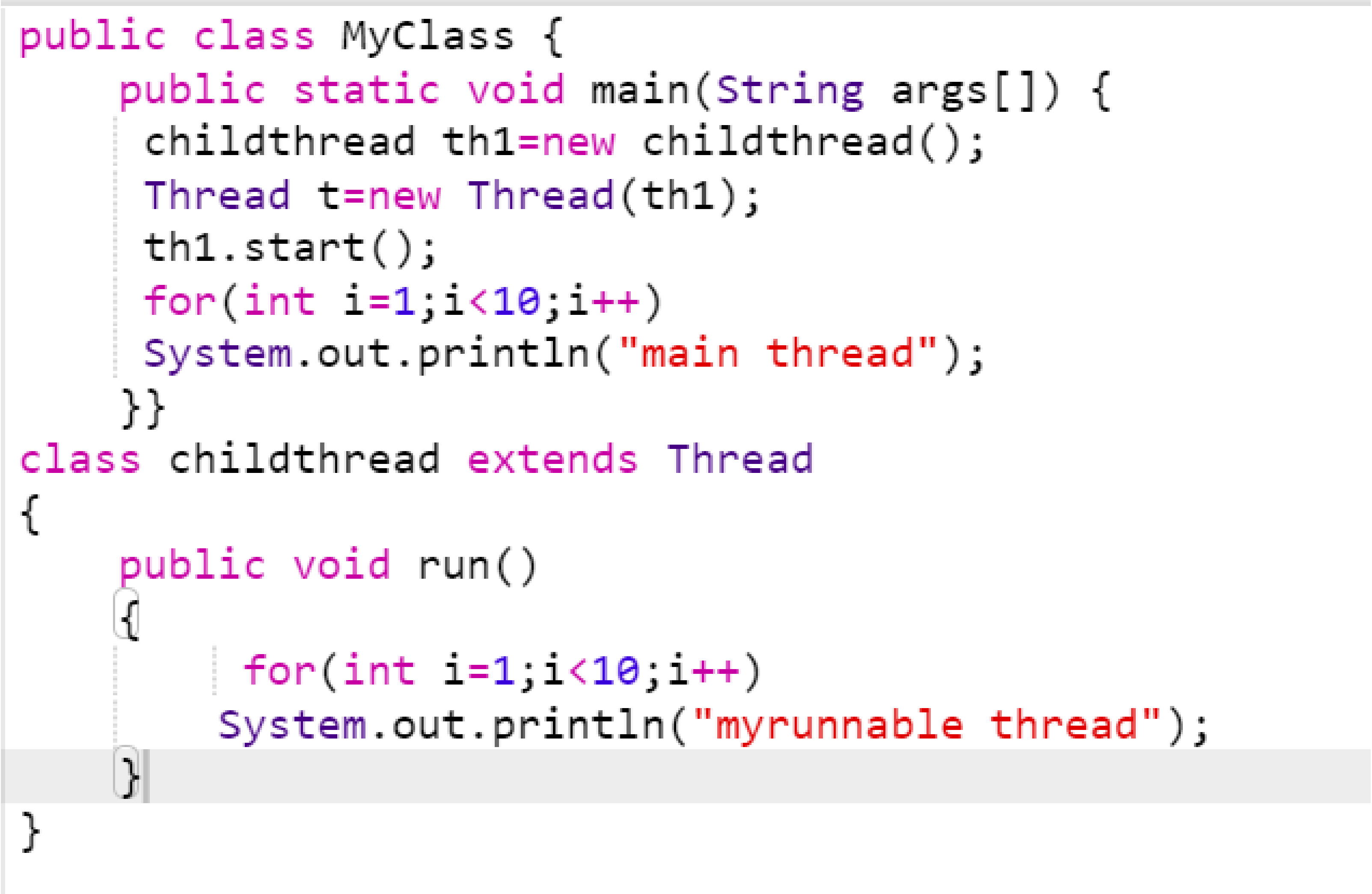
```
 class ExampleThread extends Thread
{

    public void run(){

        for(int i=0;i<10;i++)
            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        ExampleThread th1=new ExampleThread();
        th1.start();
        for(int i=0;i<10;i++)
            System.out.println("parent thread");
    }}
```



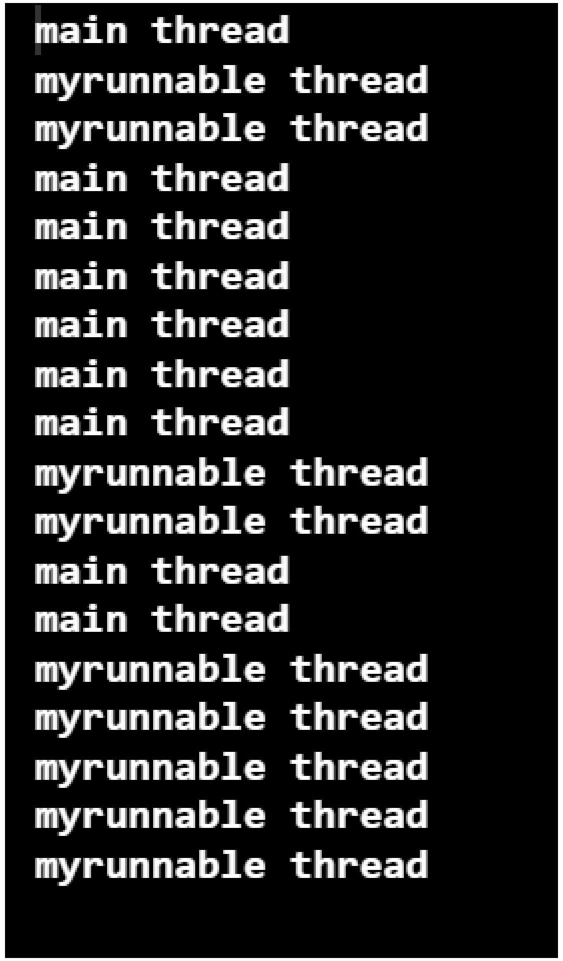
```
 class Myrunnable implements Runnable
{
    public void run(){

        for(int i=0;i<10;i++)
            System.out.println("child thread");
    }}
public class Main
{
    public static void main(String []  args)
    {
        Myrunnable th1=new Myrunnable();
        Thread t=new Thread(th1);
        t.start();
        for(int i=0;i<10;i++)
            System.out.println("parent thread");
    }}
```

THREAD CLASS CONSTRUCTOR

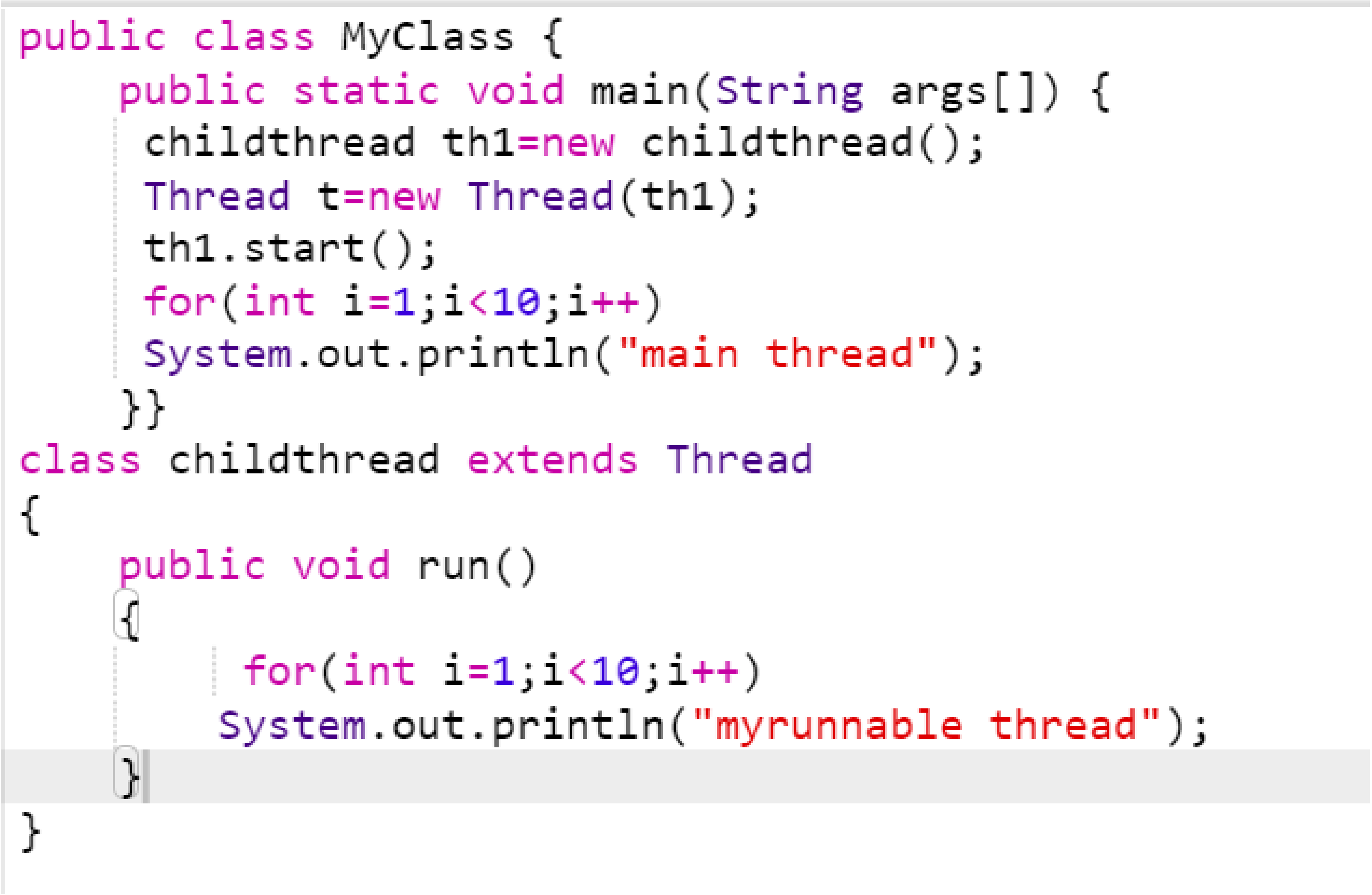
1. **Thread t=new Thread()**
2. **Thread t=new Thread(Runnable r)**
3. **Thread t=new Thread(String name)**
4. **Thread t=new Thread(Runnable target, String name)**
5. **Thread t=new Thread(ThreadGroup group, String name)**
6. **Thread t=new Thread(ThreadGroup group, Runnable target)**
7. **Thread t=new Thread(ThreadGroup group, Runnable target, String name)**
8. **Thread t=new Thread(ThreadGroup group, Runnable target, String name, long stackSize)**



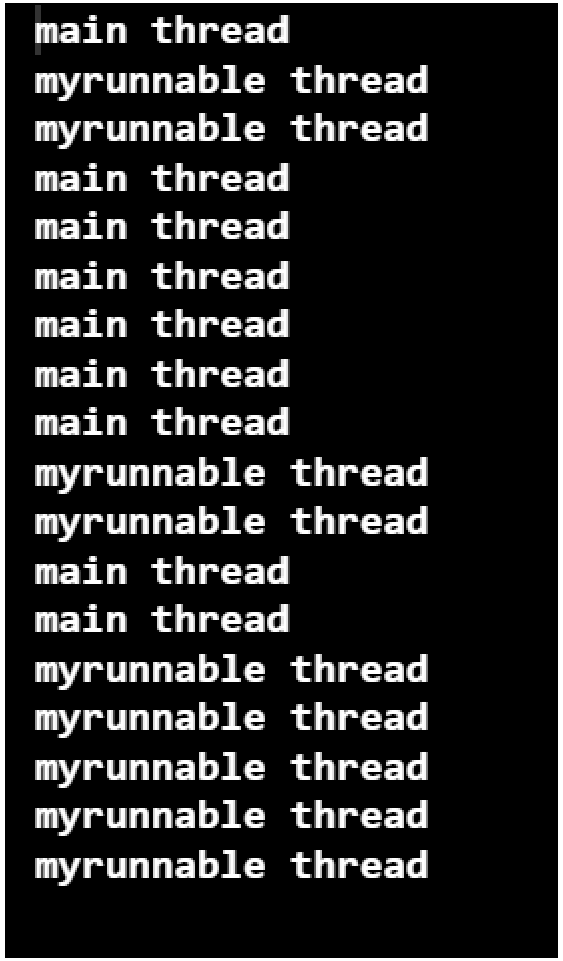
```
public class MyClass {
    public static void main(String args[]) {
     childthread th1=new childthread();
     Thread t=new Thread(th1);
     th1.start();
     for(int i=1;i<10;i++)
     System.out.println("main thread");
    }}
class childthread extends Thread
{
    public void run()
    {
         for(int i=1;i<10;i++)
        System.out.println("myrunnable thread");
    }
}
```



```
main thread
myrunnable thread
myrunnable thread
main thread
main thread
main thread
main thread
main thread
main thread
myrunnable thread
myrunnable thread
main thread
main thread
myrunnable thread
myrunnable thread
myrunnable thread
myrunnable thread
myrunnable thread
```



```
public class MyClass {
    public static void main(String args[]) {
     childthread th1=new childthread();
     Thread t=new Thread(th1);
     th1.start();
     for(int i=1;i<10;i++)
     System.out.println("main thread");
    }}
class childthread extends Thread
{
    public void run()
    {
         for(int i=1;i<10;i++)
       System.out.println("myrunnable thread");
    }
}
```



```
main thread
myrunnable thread
myrunnable thread
main thread
main thread
main thread
main thread
main thread
main thread
myrunnable thread
myrunnable thread
main thread
main thread
myrunnable thread
myrunnable thread
myrunnable thread
myrunnable thread
myrunnable thread
```

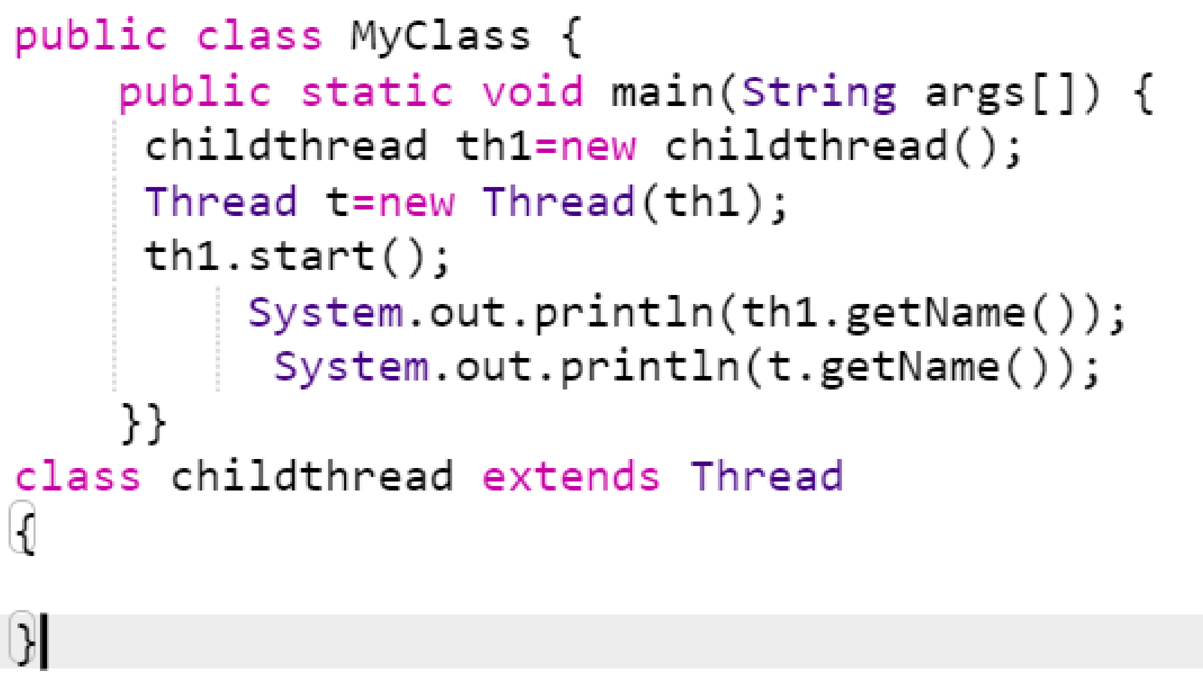
THREAD CLASS METHODS

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.

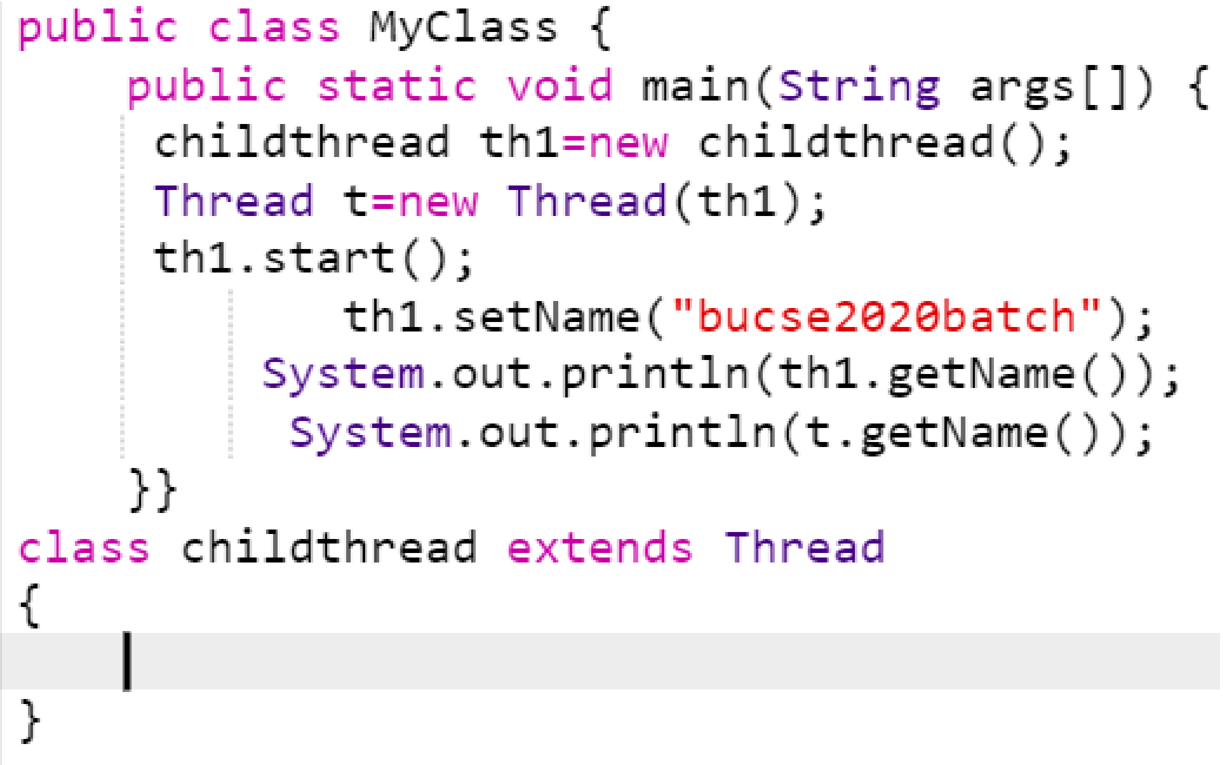
THREAD CLASS METHODS

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.

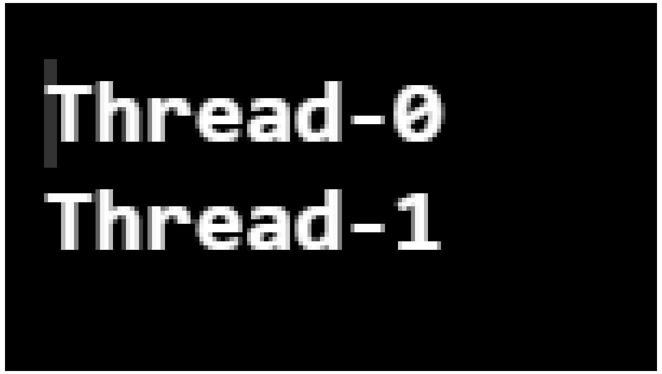
GETNAME AND SETNAME METHOD



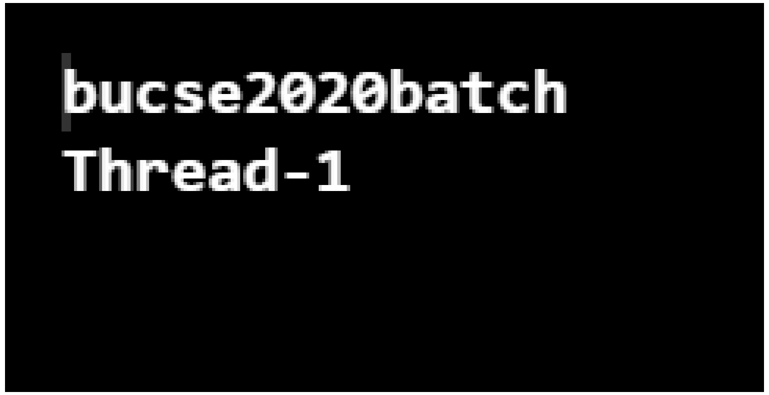
```java
public class MyClass {
    public static void main(String args[]) {
     childthread th1=new childthread();
     Thread t=new Thread(th1);
     th1.start();
         System.out.println(th1.getName());
          System.out.println(t.getName());
    }}
class childthread extends Thread
{

}
```



```java
public class MyClass {
    public static void main(String args[]) {
     childthread th1=new childthread();
     Thread t=new Thread(th1);
     th1.start();
            th1.setName("bucse2020batch");
         System.out.println(th1.getName());
          System.out.println(t.getName());
    }}
class childthread extends Thread
{

}
```

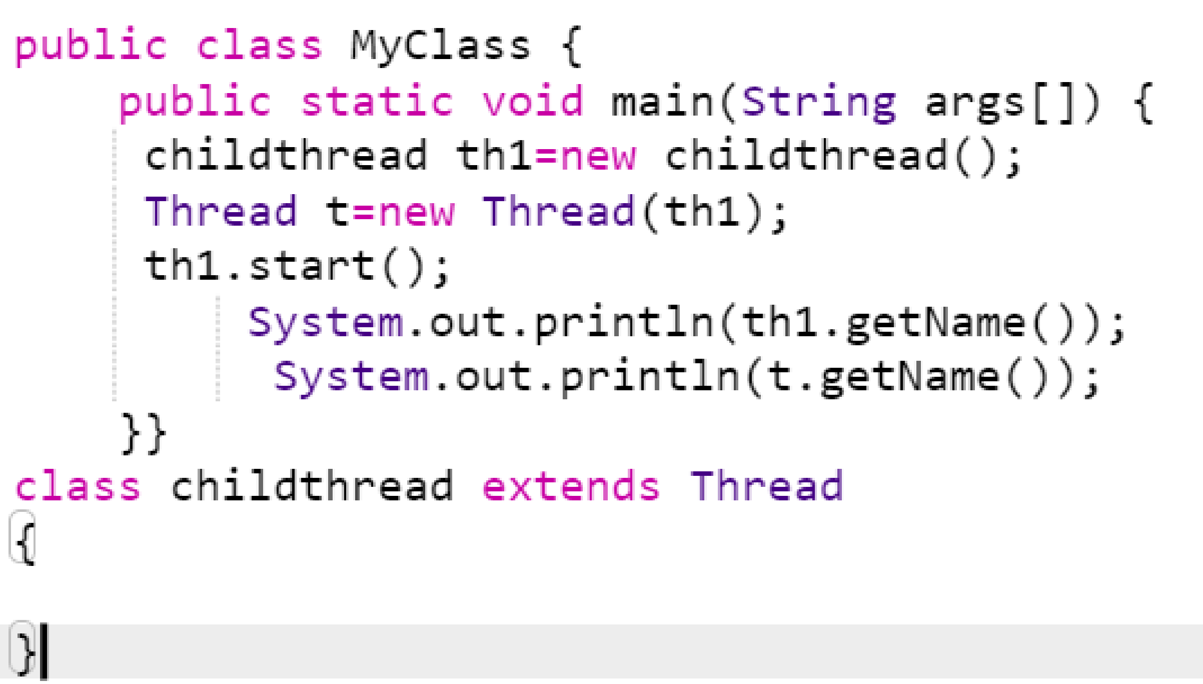


```
Thread-0
Thread-1
```

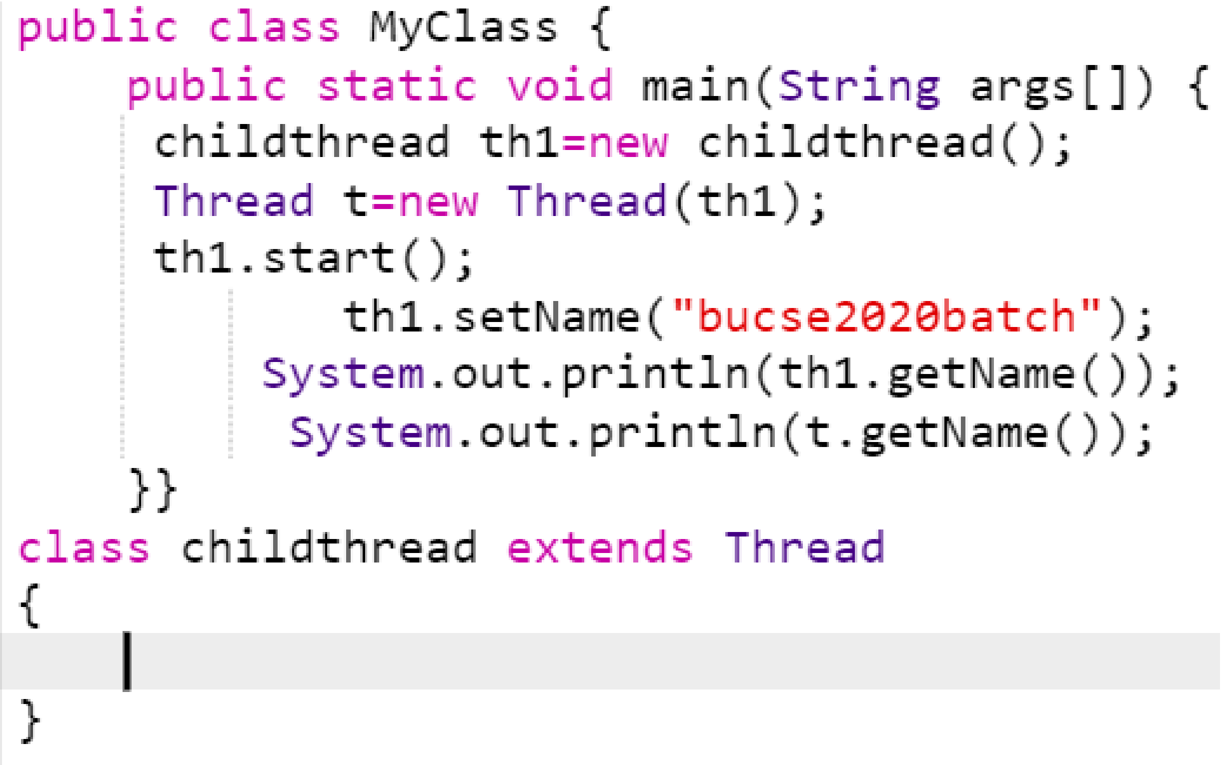


```
bucse2020batch
Thread-1
```

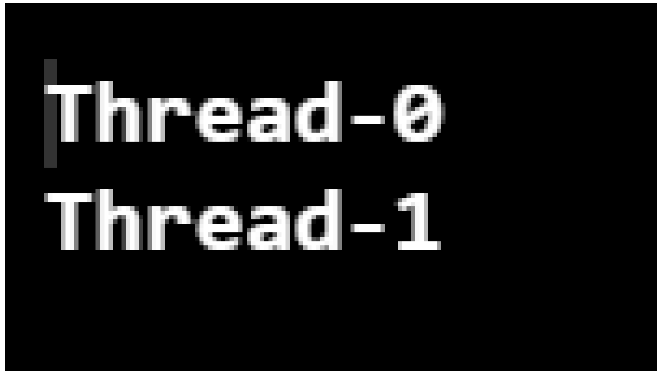
GETNAME AND SETNAME METHOD



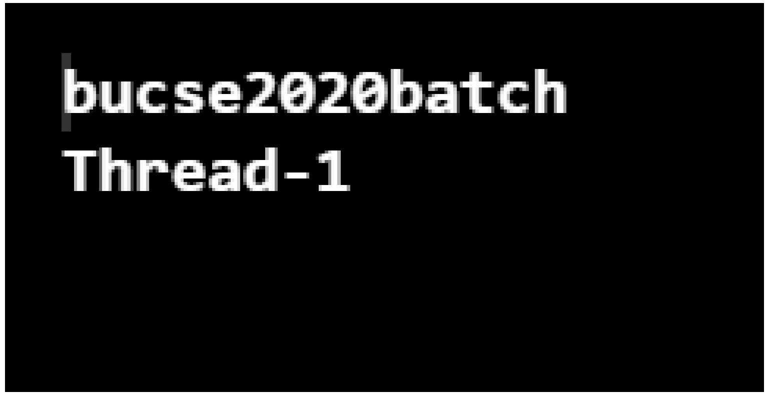
```java
public class MyClass {
    public static void main(String args[]) {
     childthread th1=new childthread();
     Thread t=new Thread(th1);
     th1.start();
         System.out.println(th1.getName());
          System.out.println(t.getName());
    }}
class childthread extends Thread
{

}
```



```java
public class MyClass {
    public static void main(String args[]) {
     childthread th1=new childthread();
     Thread t=new Thread(th1);
     th1.start();
            th1.setName("bucse2020batch");
         System.out.println(th1.getName());
          System.out.println(t.getName());
    }}
class childthread extends Thread
{

}
```



```
Thread-0
Thread-1
```



```
bucse2020batch
Thread-1
```

THREAD PRIORITY

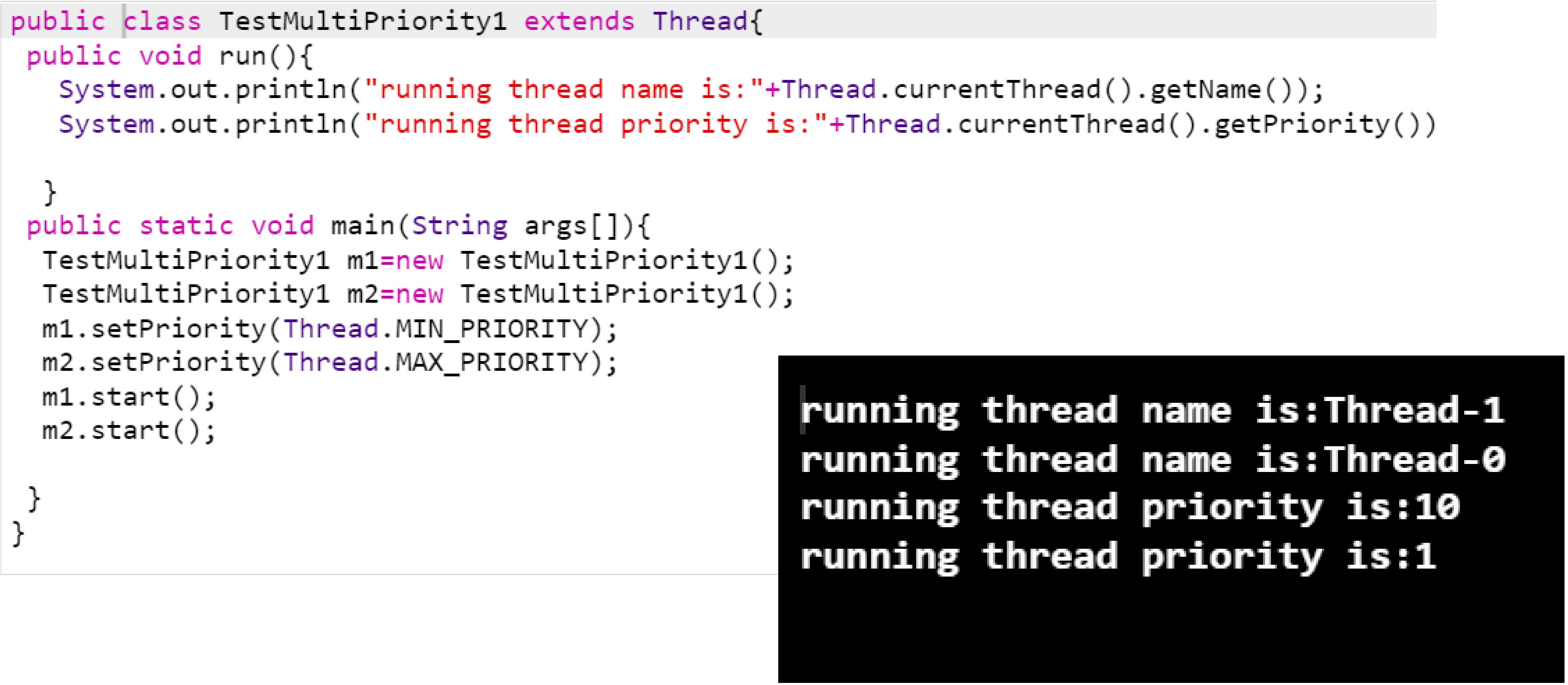
- It may be default priority set by jvm
- It may be customized priority set by programmer

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

THREAD PRIORITY: EXAMPLE



```java
public 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();

 }
}
```

```
running thread name is:Thread-1
running thread name is:Thread-0
running thread priority is:10
running thread priority is:1
```

METHODS THROUGH WHICH WE CAN PREVENT THREAD EXECUTION

1

yield()

2

join()

3

sleep()

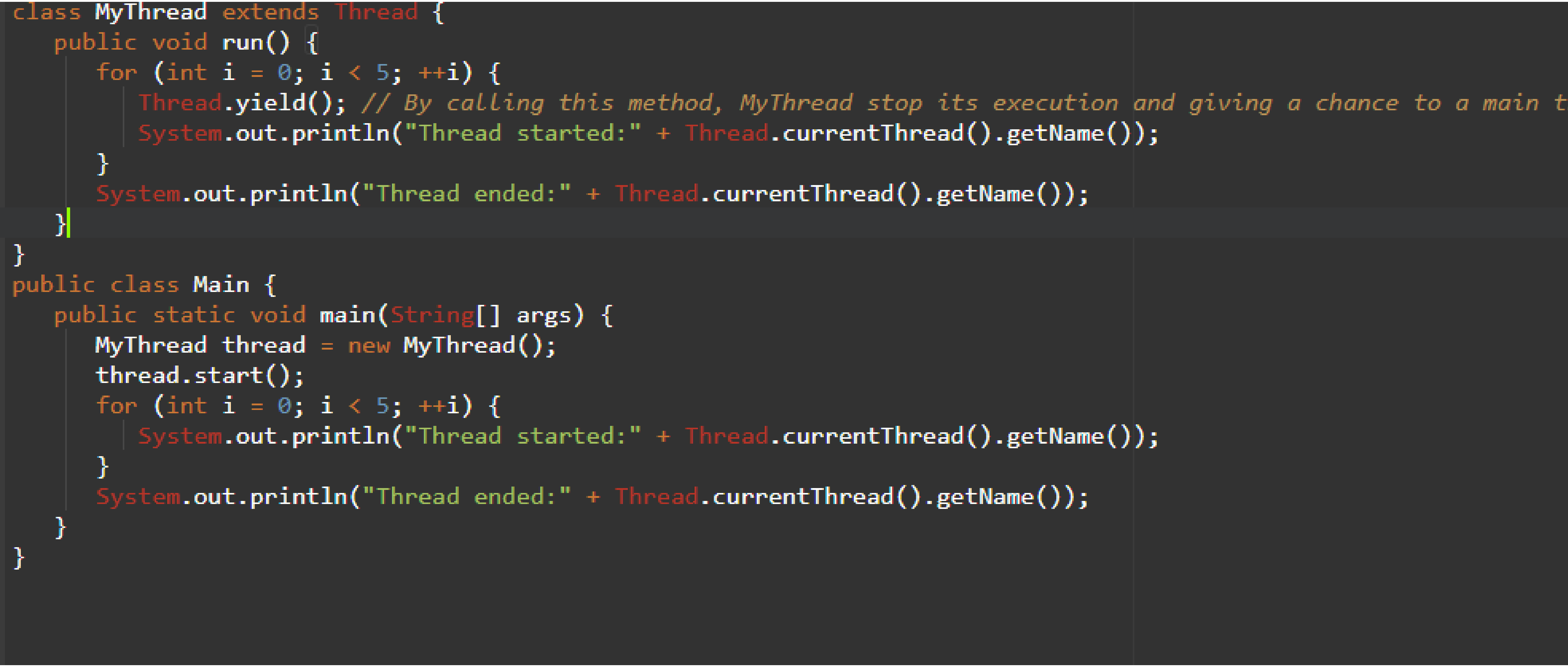
YIELD():

- A **yield()** method is a **static** method of **Thread** class and it can stop the currently executing thread and will give a chance to **other waiting threads of the same priority.**
- If in case there are no waiting threads or if all the waiting threads have **low priority** then the same thread will continue its execution.
- The advantage of **yield()** method is to get a chance to execute other waiting threads so if our current thread takes more time to execute and allocate processor to other threads.
- We can use the yield() method to temporarily release time for other threads. For example,

```
public void run() {
  for (int i = 1; i <= lastNum; i++) {
    System.out.print(" " + i);
    Thread.yield();
  }
}
```

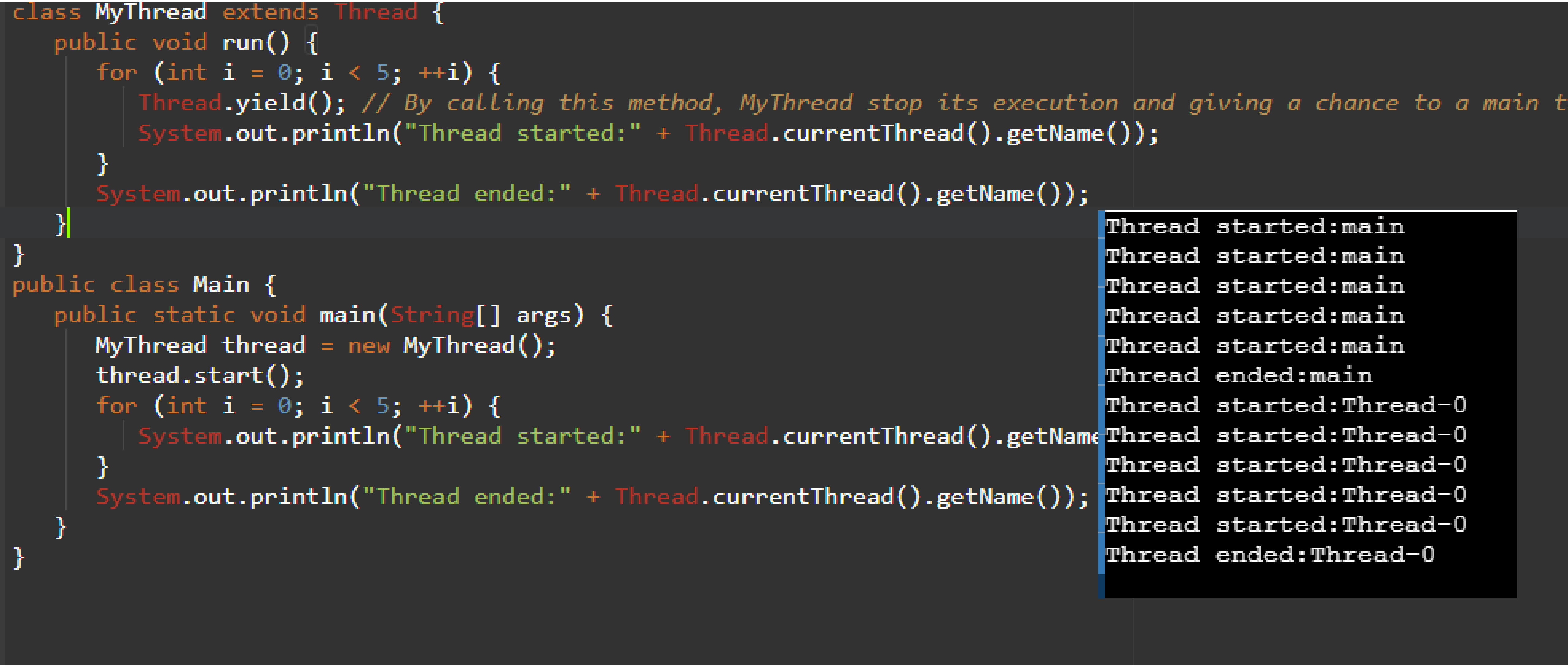
- Every time a number is printed, the current thread is yielded. So, the numbers are printed after the characters.

YIELD(): EXAMPLE



```
class MyThread extends Thread {
    public void run() {
        for (int i = 0; i < 5; ++i) {
            Thread.yield(); // By calling this method, MyThread stop its execution and giving a chance to a main t
            System.out.println("Thread started:" + Thread.currentThread().getName());
        }
        System.out.println("Thread ended:" + Thread.currentThread().getName());
    }
}
public class Main {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start();
        for (int i = 0; i < 5; ++i) {
            System.out.println("Thread started:" + Thread.currentThread().getName());
        }
        System.out.println("Thread ended:" + Thread.currentThread().getName());
    }
}
```

YIELD(): EXAMPLE



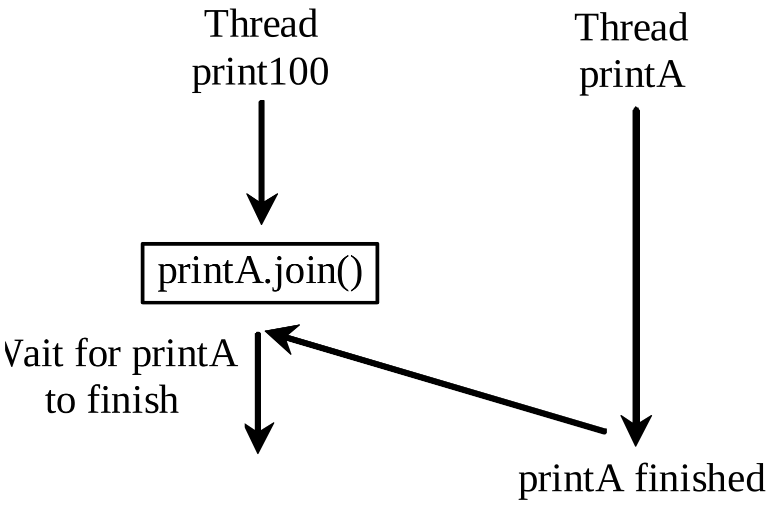
```java
class MyThread extends Thread {
    public void run() {
        for (int i = 0; i < 5; ++i) {
            Thread.yield(); // By calling this method, MyThread stop its execution and giving a chance to a main t
            System.out.println("Thread started:" + Thread.currentThread().getName());
        }
        System.out.println("Thread ended:" + Thread.currentThread().getName());
    }
}
public class Main {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start();
        for (int i = 0; i < 5; ++i) {
            System.out.println("Thread started:" + Thread.currentThread().getName
        }
        System.out.println("Thread ended:" + Thread.currentThread().getName());
    }
}
```

```
Thread started:main
Thread started:main
Thread started:main
Thread started:main
Thread started:main
Thread ended:main
Thread started:Thread-0
Thread started:Thread-0
Thread started:Thread-0
Thread started:Thread-0
Thread started:Thread-0
Thread ended:Thread-0
```

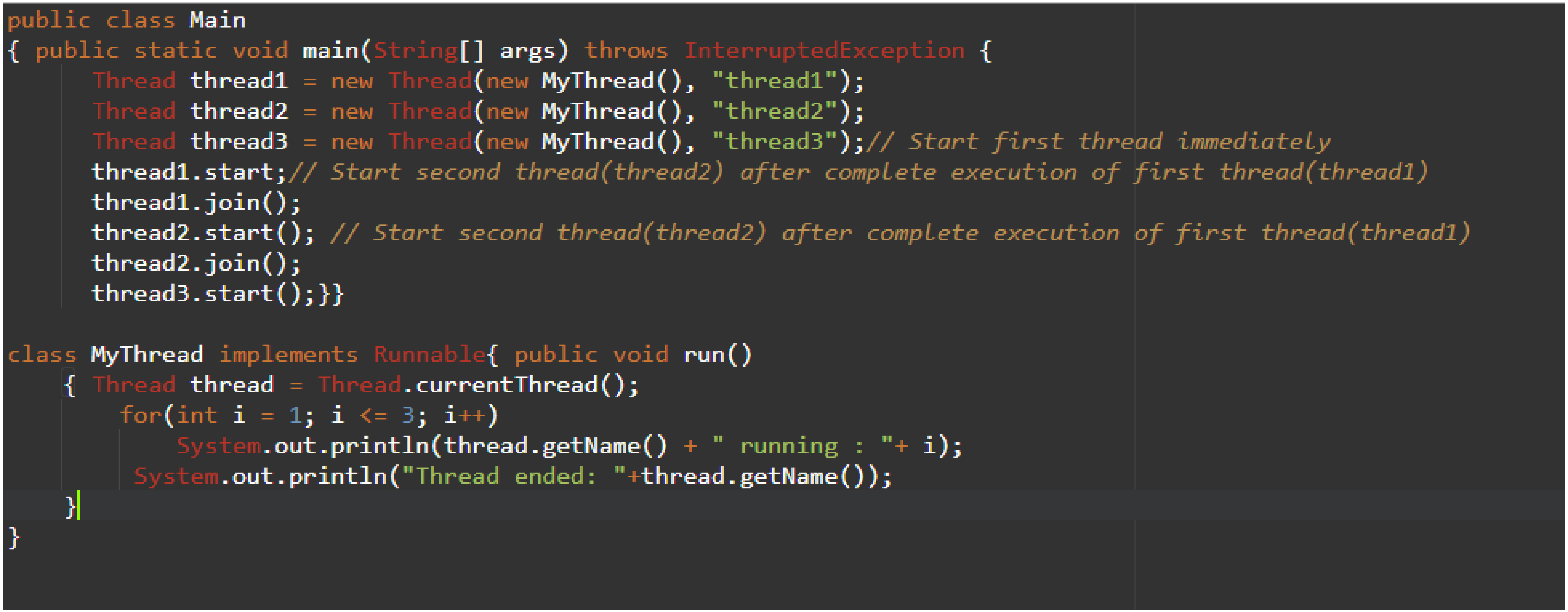
JOIN():

- 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.
- If a thread want to wait until completing some other thread then we should go for join method.
- For example if a thread t1 wants to wait until completing t2 then t1 has to call t2.join().
- If t1 executes t2.join then immediately t1 will be entered into waiting state until t2 completes.
- Once t2 completes then t1 can continue its execution.

```
public void run() {
  Thread thread4 = new Thread(
    new PrintChar('c', 40));
  thread4.start();
  try {
    for (int i = 1; i <= lastNum; i++) {
      System.out.print(" " + i);
      if (i == 50) thread4.join();
    }
  }
  catch (InterruptedException ex) {
  }
```

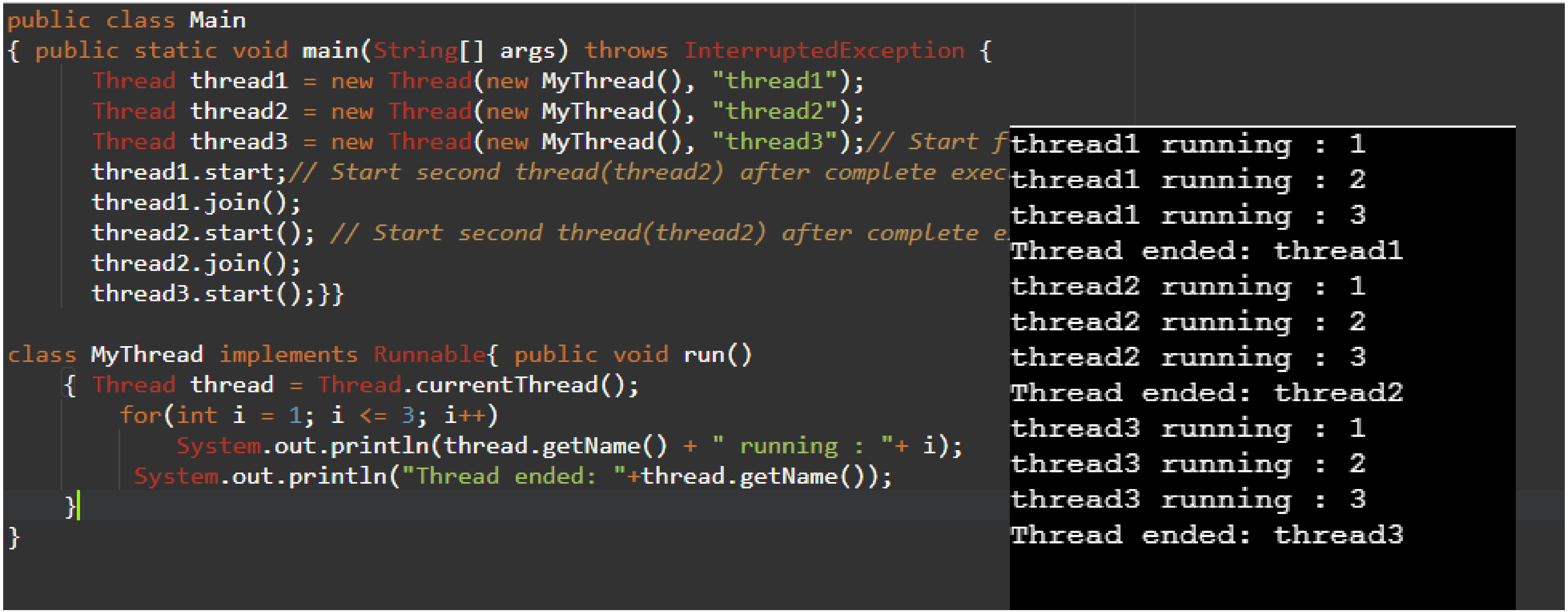


JOIN():EXAMPLE



```
public class Main
{ public static void main(String[] args) throws InterruptedException {
      Thread thread1 = new Thread(new MyThread(), "thread1");
      Thread thread2 = new Thread(new MyThread(), "thread2");
      Thread thread3 = new Thread(new MyThread(), "thread3");// Start first thread immediately
      thread1.start;// Start second thread(thread2) after complete execution of first thread(thread1)
      thread1.join();
      thread2.start(); // Start second thread(thread2) after complete execution of first thread(thread1)
      thread2.join();
      thread3.start();}}

class MyThread implements Runnable{ public void run()
    { Thread thread = Thread.currentThread();
        for(int i = 1; i <= 3; i++)
            System.out.println(thread.getName() + " running : "+ i);
         System.out.println("Thread ended: "+thread.getName());
    }
}
```

JOIN():EXAMPLE



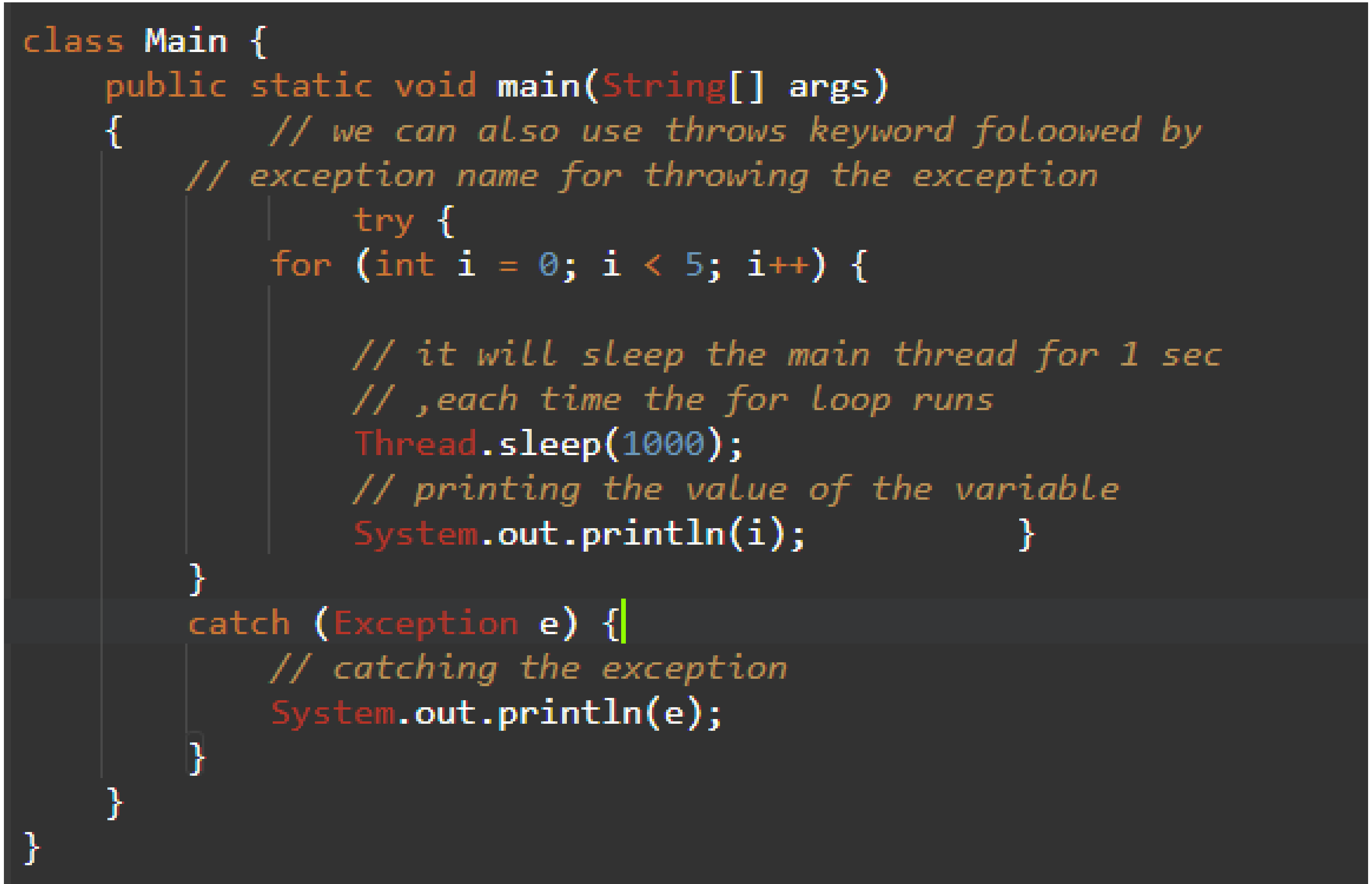
```
public class Main
{ public static void main(String[] args) throws InterruptedException {
      Thread thread1 = new Thread(new MyThread(), "thread1");
      Thread thread2 = new Thread(new MyThread(), "thread2");
      Thread thread3 = new Thread(new MyThread(), "thread3");// Start f
      thread1.start;// Start second thread(thread2) after complete exec
      thread1.join();
      thread2.start(); // Start second thread(thread2) after complete e
      thread2.join();
      thread3.start();}}

class MyThread implements Runnable{ public void run()
    { Thread thread = Thread.currentThread();
        for(int i = 1; i <= 3; i++)
            System.out.println(thread.getName() + " running : "+ i);
         System.out.println("Thread ended: "+thread.getName());
    }
}
```

```
thread1 running : 1
thread1 running : 2
thread1 running : 3
Thread ended: thread1
thread2 running : 1
thread2 running : 2
thread2 running : 3
Thread ended: thread2
thread3 running : 1
thread3 running : 2
thread3 running : 3
Thread ended: thread3
```

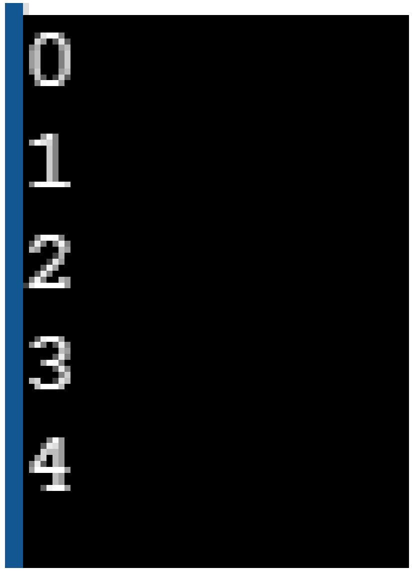
SLEEP():

- The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
- The java.lang.Thread.sleep(long millis) method causes the currently executing thread to sleep for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.
- Method Whenever Thread.sleep() functions to execute, it always pauses the current thread execution.
- If any other thread interrupts when the thread is sleeping, then **InterruptedException** will be thrown.
- If the system is busy, then the actual time the thread will sleep will be more as compared to that passed while calling the sleep method and if the system has less load, then the actual sleep time of the thread will be close to that passed while calling sleep() method.

SLEEP():EXAMPLE

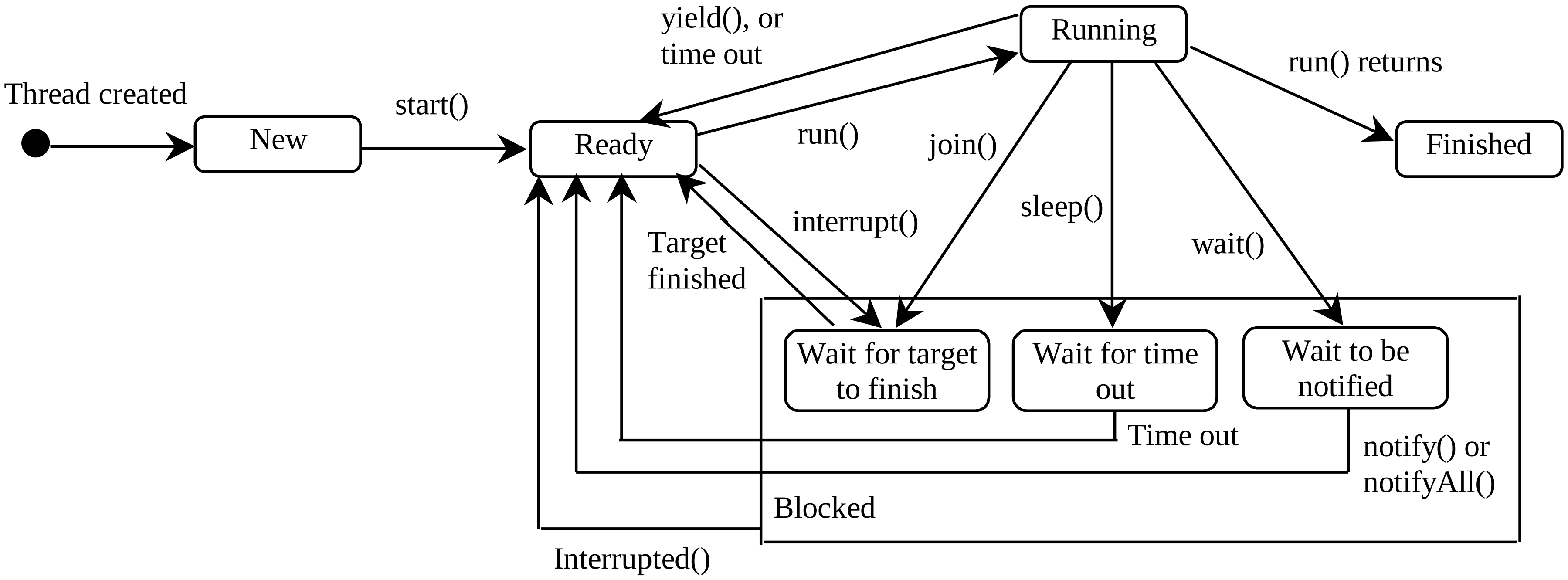


```java
class Main {
    public static void main(String[] args)
    {       // we can also use throws keyword foloowed by
        // exception name for throwing the exception
                try {
            for (int i = 0; i < 5; i++) {

                // it will sleep the main thread for 1 sec
                // ,each time the for loop runs
                Thread.sleep(1000);
                // printing the value of the variable
                System.out.println(i);           }
        }
        catch (Exception e) {
            // catching the exception
            System.out.println(e);
        }
    }
}
```



```
0
1
2
3
4
```

THREADS STATES



SYNCHRONIZATION

- Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.
- So it needs to be made sure by some synchronization method that only **one thread can access the resource at a given point of time.**
- **Cause race condition**
- Why do we require Synchronization?

Ans: To achieve **Consistency.**

Step	balance	thread[i]	thread[j]
1	0	newBalance = bank.getBalance() + 1;	
2	0		newBalance = bank.getBalance() + 1;
3	1	bank.setBalance(newBalance);	
4	1		bank.setBalance(newBalance);

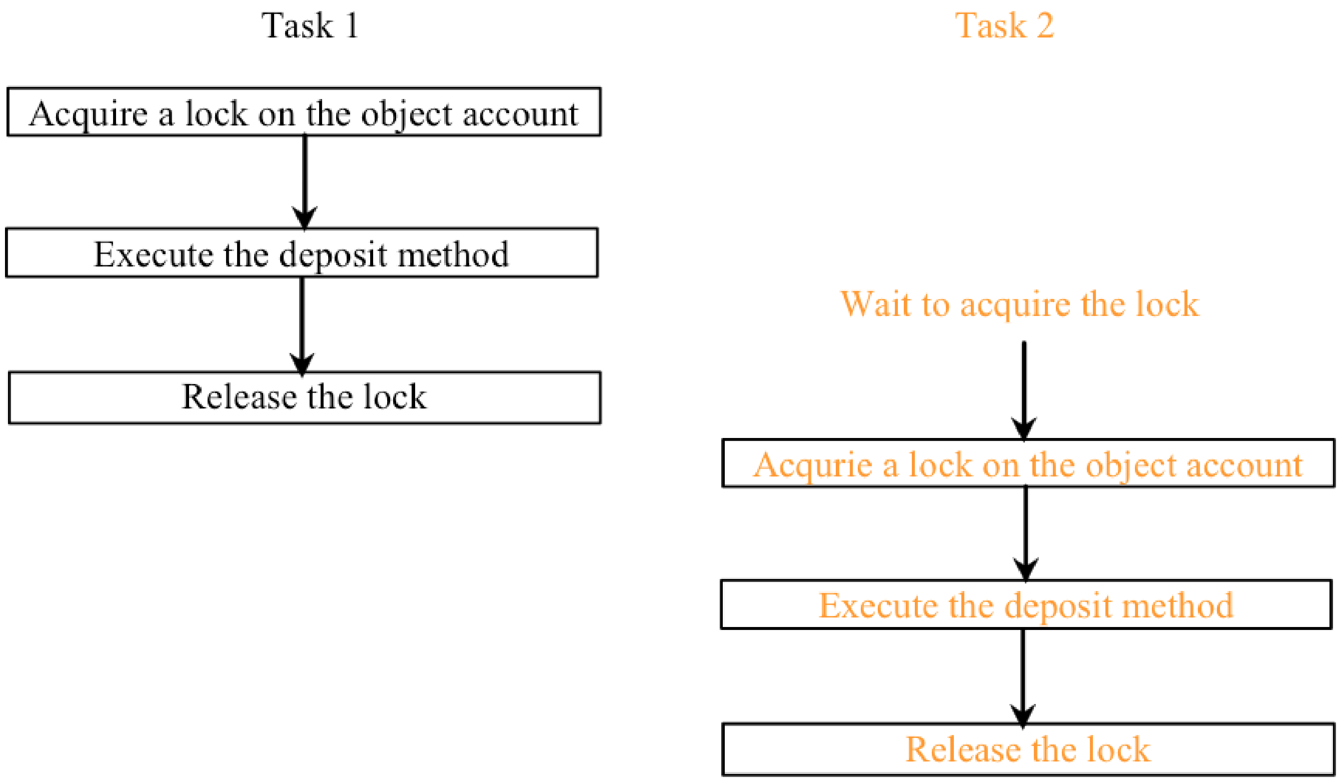
THE SYNCHRONIZED KEYWORD

- The synchronized keyword can be used to mark four different types of blocks:
  - Instance methods
  - Static methods
  - Code blocks inside instance methods
  - Code blocks inside static methods
- To avoid race conditions, threads must be prevented from simultaneously entering certain part of the program, known as critical region.
- The critical is the entire deposit method. You can use the synchronized keyword to synchronize the method so that only **one thread can access the method at a time.**

```
public synchronized void deposit(double amount) {
 int newBalance= balance + amount;
Balance = newBalance;
}
```

SYNCHRONIZING INSTANCE METHODS AND STATIC METHODS

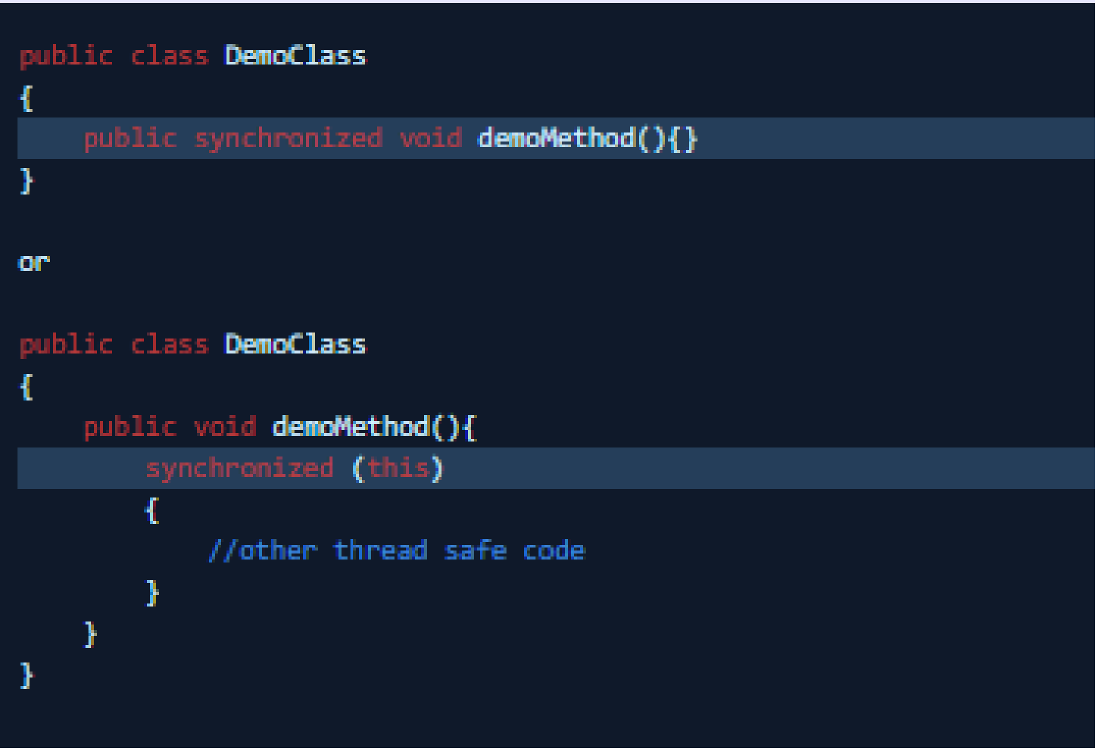
- With the deposit method synchronized, the preceding scenario cannot happen. If Task 2 starts to enter the method, and Task 1 is already in the method, Task 2 is blocked until Task 1 finishes the method.



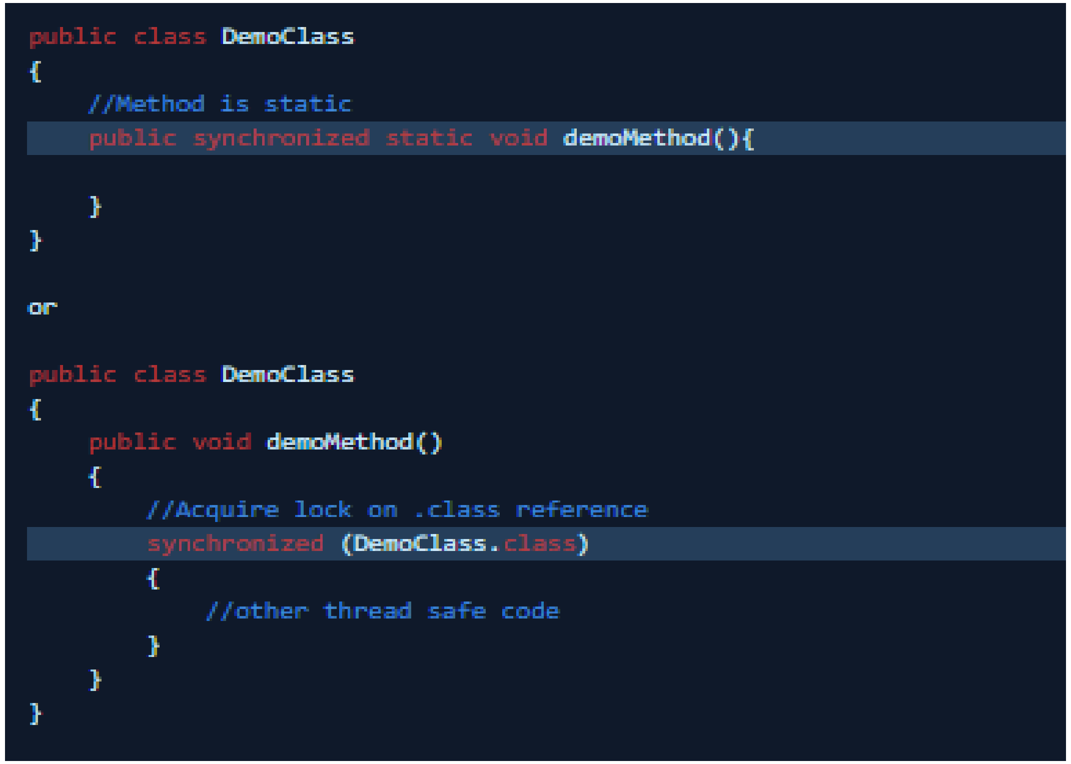
Task 1
Task 2
Acquire a lock on the object account
Execute the deposit method
Release the lock
Wait to acquire the lock
Acqurie a lock on the object account
Execute the deposit method
Release the lock

THE SYNCHRONIZED KEYWORD

- The synchronized keyword can be used to mark four different types of blocks:
  - Instance methods
  - Static methods
  - Code blocks inside instance methods
  - Code blocks inside static methods

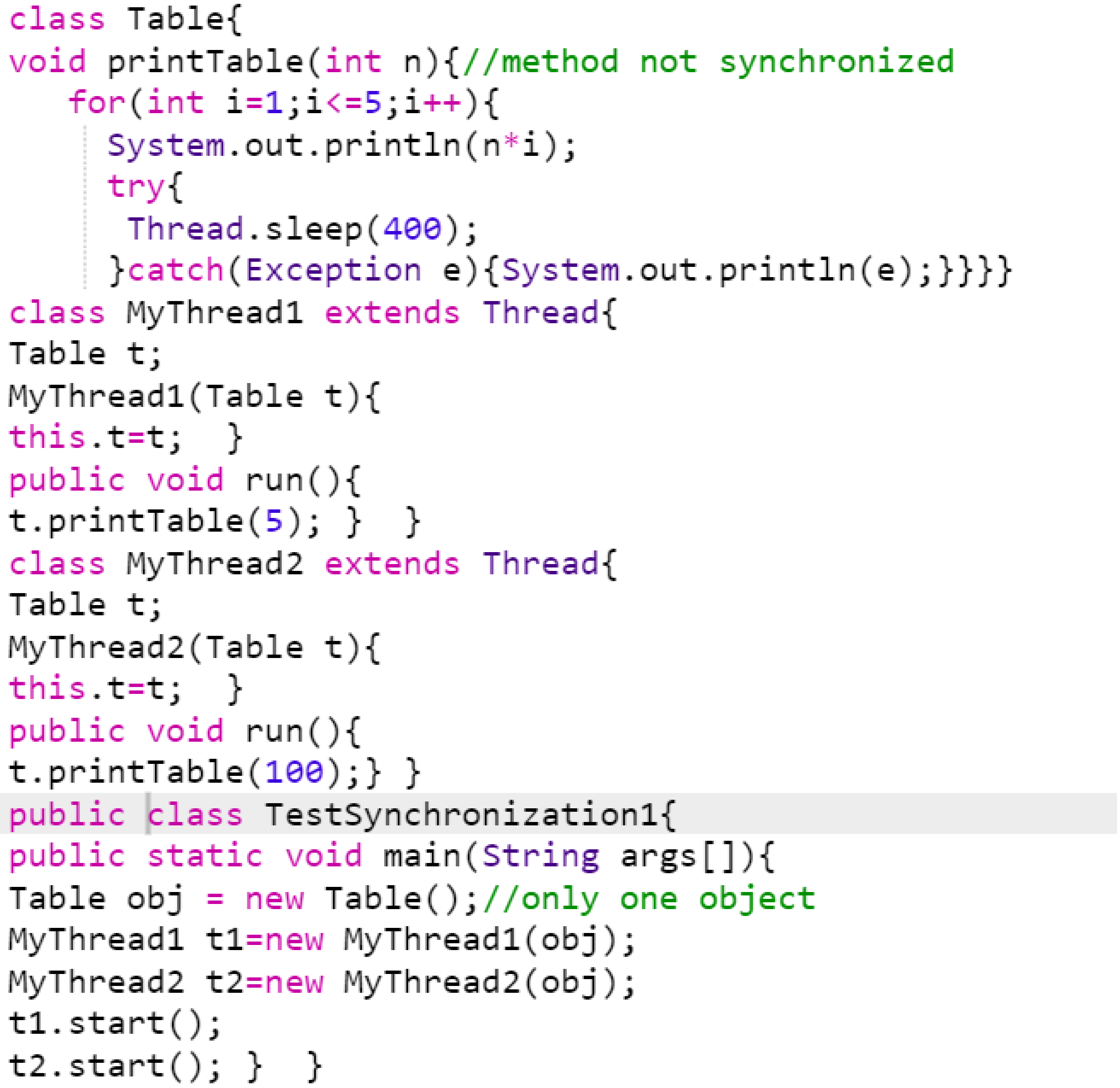


```
public class DemoClass
{
    public synchronized void demoMethod(){}
}

or

public class DemoClass
{
    public void demoMethod(){
        synchronized (this)
        {
            //other thread safe code
        }
    }
}
```

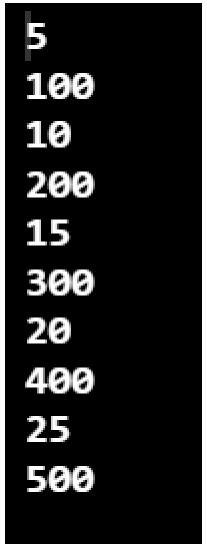


```
public class DemoClass
{
    //Method is static
    public synchronized static void demoMethod(){

    }
}

or

public class DemoClass
{
    public void demoMethod()
    {
        //Acquire lock on .class reference
        synchronized (DemoClass.class)
        {
            //other thread safe code
        }
    }
}
```

EXAMPLE:

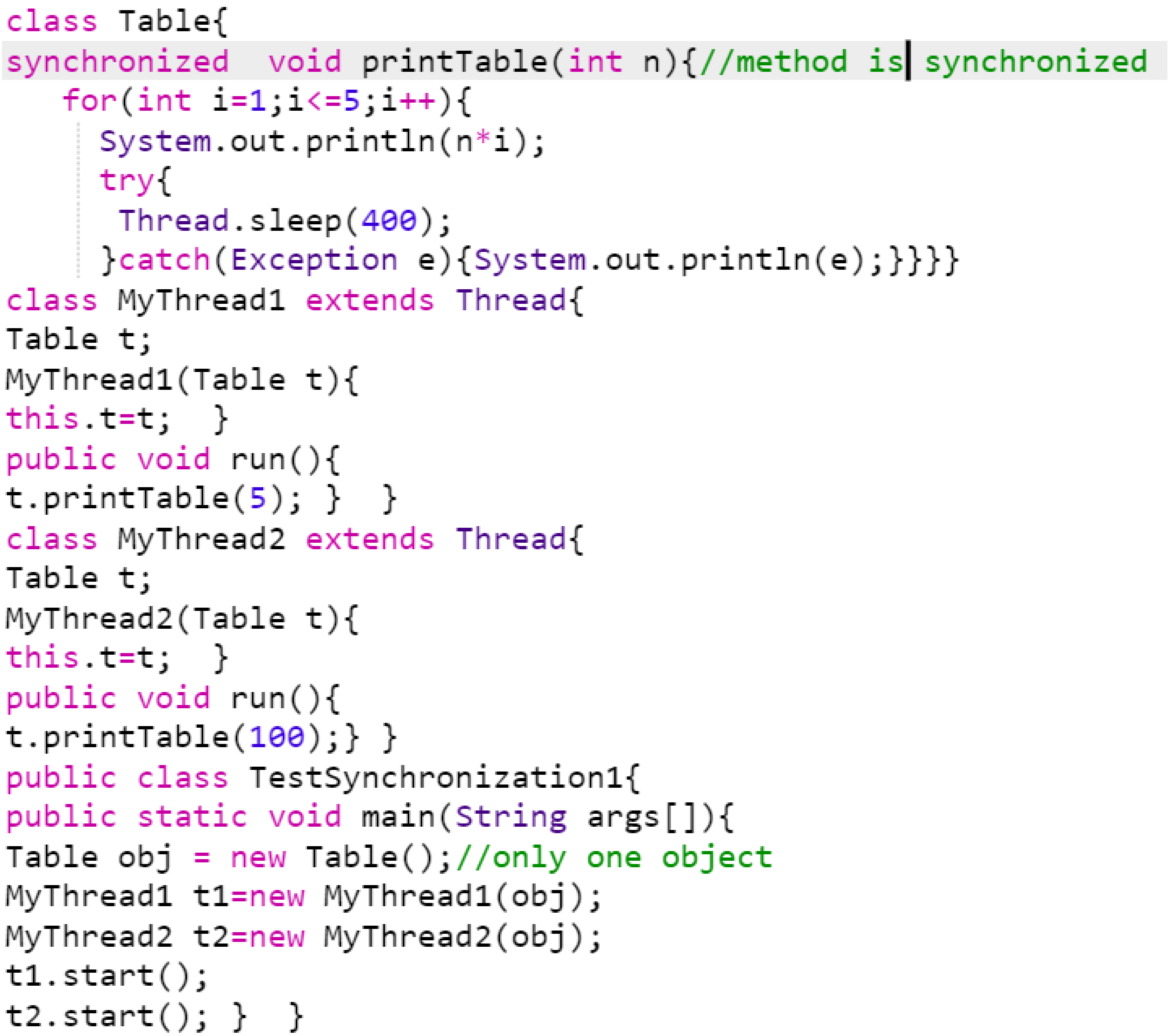


```
class Table{
void printTable(int n){//method not synchronized
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}}}}
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;  }
public void run(){
t.printTable(5); }  }
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;  }
public void run(){
t.printTable(100);} }
public 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(); }  }
```

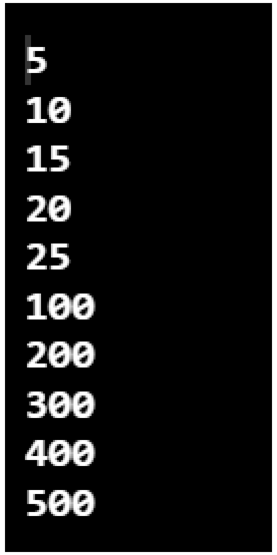


```
5
100
10
200
15
300
20
400
25
500
```

EXAMPLE:



```
class Table{
synchronized  void printTable(int n){//method is 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);} }
public 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(); }  }
```



```
5
10
15
20
25
100
200
300
400
500
```

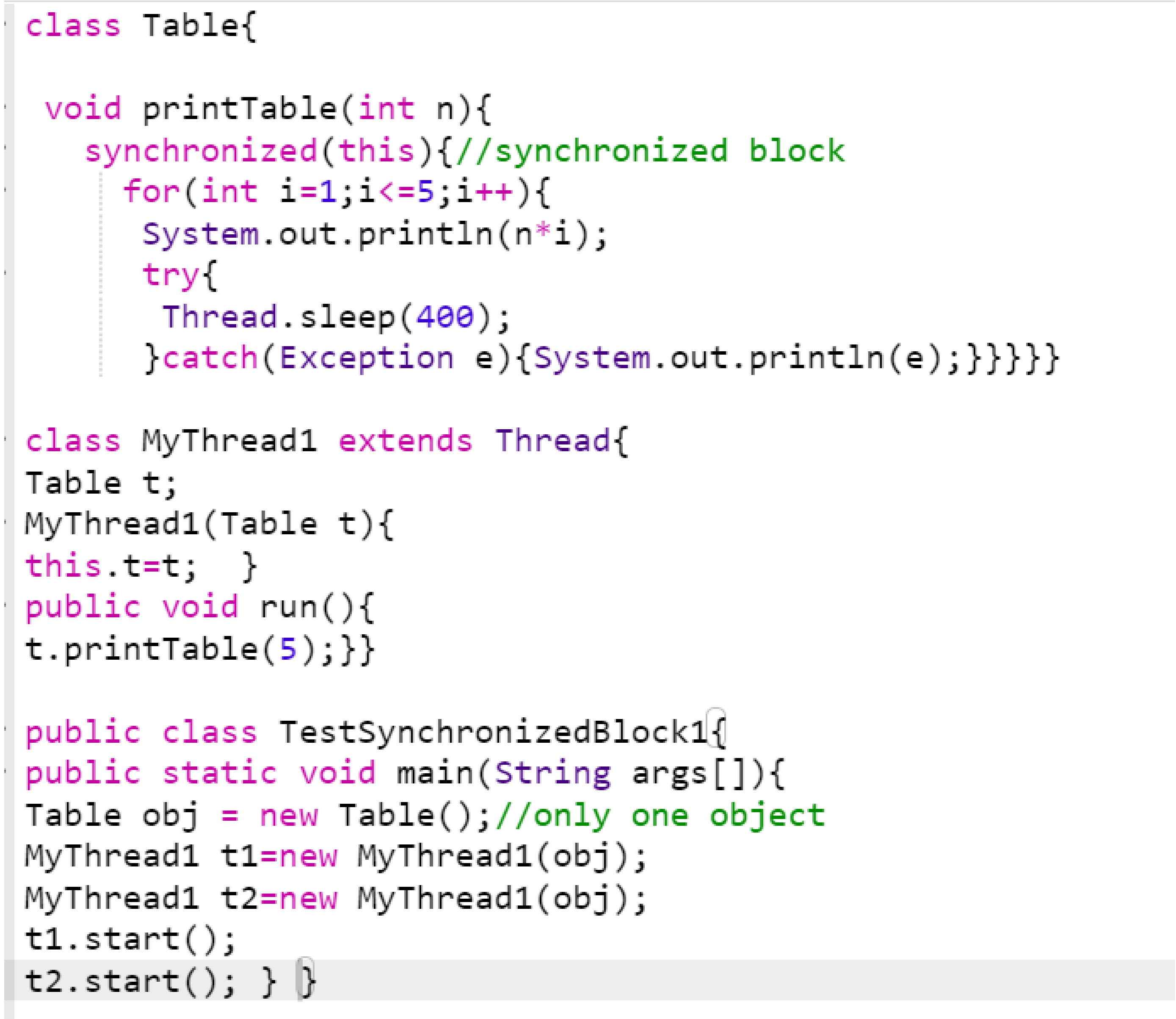
SYNCHRONIZED BLOCK IN JAVA

- Synchronized block can be used to perform synchronization on any specific resource of the method.
- Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
- If you put all the codes of the method in the synchronized block, it will work same as the synchronized method

Syntax to use synchronized block:

```
synchronized (object reference expression) {
  //code block
}
```

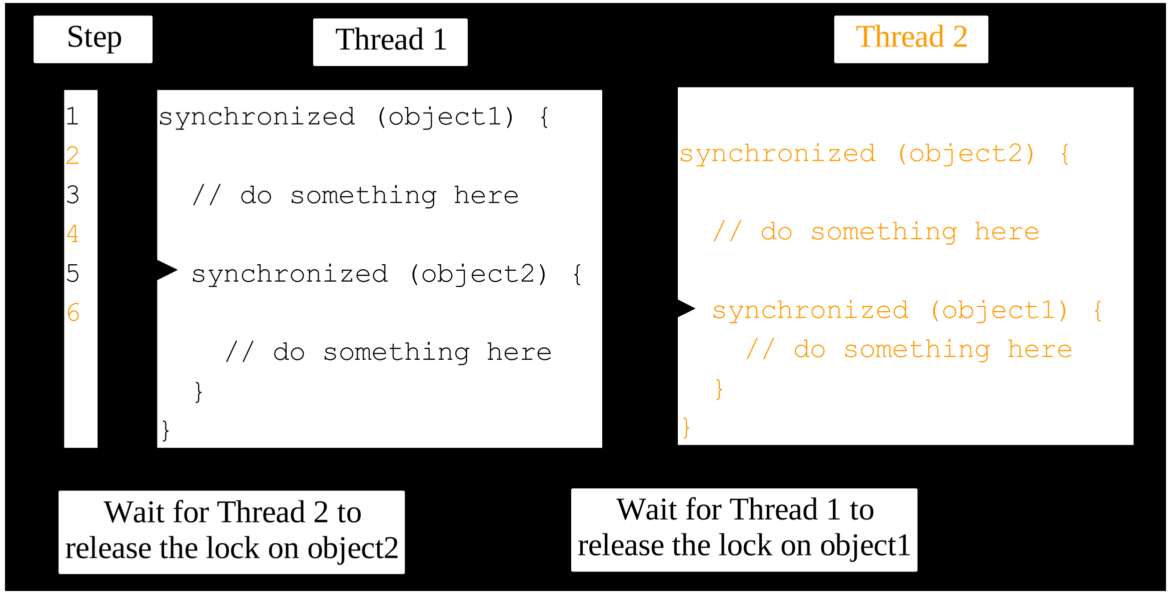
EXAMPLE:



```
class Table{

 void printTable(int n){
   synchronized(this){//synchronized block
     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);}}

public class TestSynchronizedBlock1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread1 t2=new MyThread1(obj);
t1.start();
t2.start(); } }
```

DEADLOCK

- Sometimes two or more threads need to acquire the locks on several shared objects. This could cause deadlock, in which each thread has the lock on one of the objects and is waiting for the lock on the other object. Consider the scenario with two threads and two objects, as shown. Thread 1 acquired a lock on object1 and Thread 2 acquired a lock on object2. Now Thread 1 is waiting for the lock on object2 and Thread 2 for the lock on object1. The two threads wait for each other to release the in order to get the lock, and neither can continue to run.



```
synchronized (object1) {

  // do something here

  synchronized (object2) {

    // do something here
  }
}
```

```
synchronized (object2) {

  // do something here

  synchronized (object1) {
    // do something here
  }
}
```

Wait for Thread 2 to release the lock on object2

Wait for Thread 1 to release the lock on object1

PREVENTING DEADLOCK

- Deadlock can be easily avoided by using a simple technique known as resource ordering. With this technique, you assign an order on all the objects whose locks must be acquired and ensure that each thread acquires the locks in that order. For the example, suppose the objects are ordered as object1 and object2. Using the resource ordering technique, Thread 2 must acquire a lock on object1 first, then on object2. Once Thread 1 acquired a lock on object1, Thread 2 has to wait for a lock on object1. So Thread 1 will be able to acquire a lock on object2 and no deadlock would occur.

THANK YOU

?