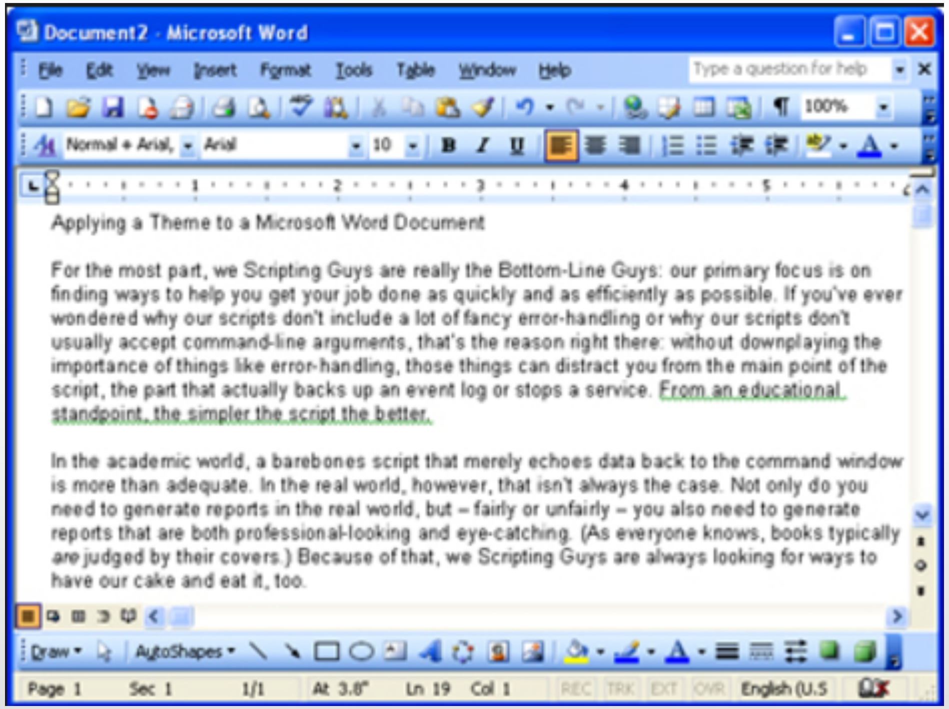
Threads

Object Oriented Programming (OOP)
2nd Year – Semester 1
By Udara Samaratunge

What is a Thread?

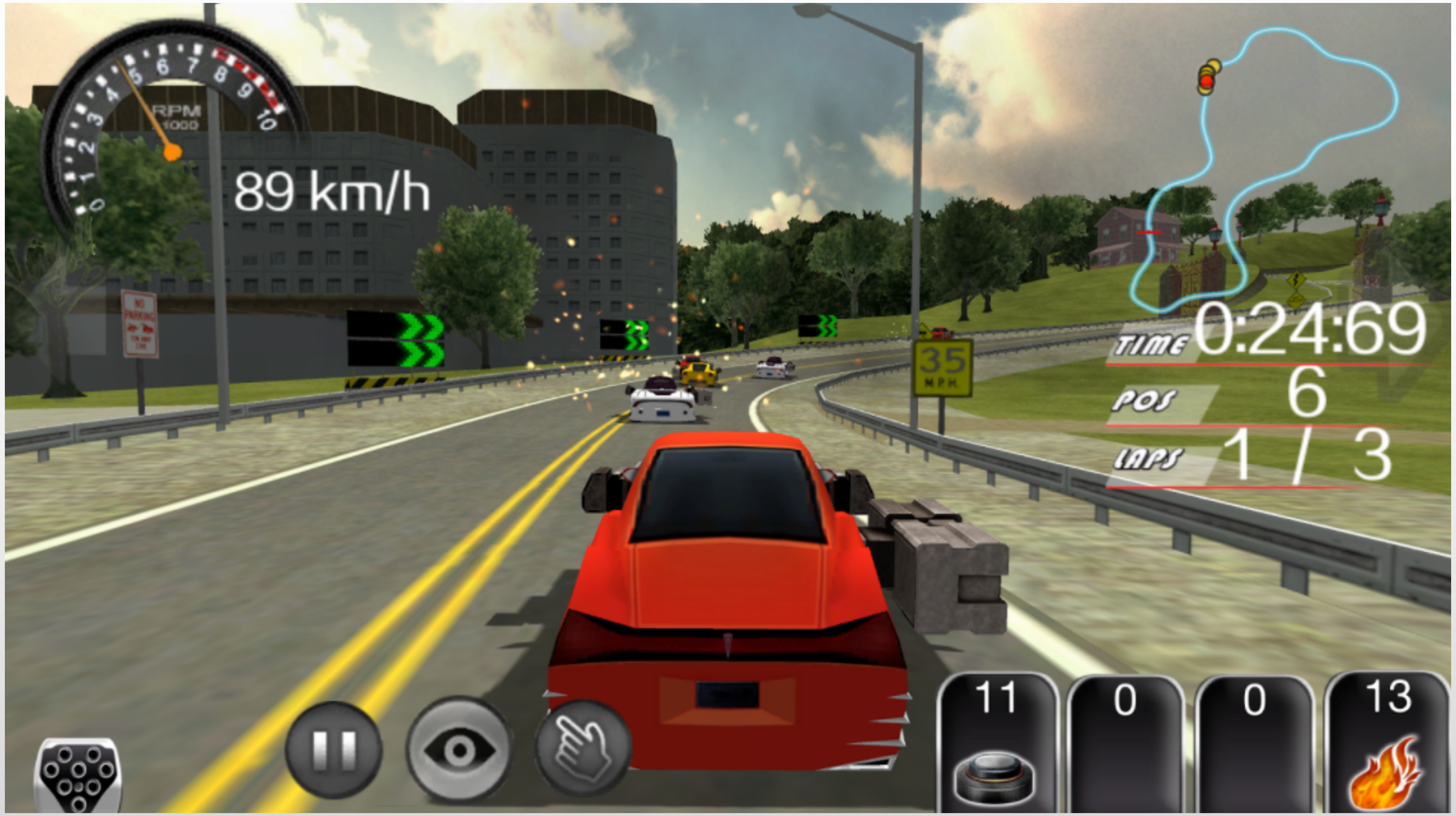
Examples:-


Document2 - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Type a question for help
100%
Normal + Arial, Arial 10
Draw AutoShapes
Page 1 Sec 1 1/1 At 3.8" Ln 19 Col 1 REC TRK EXT OVR English (U.S

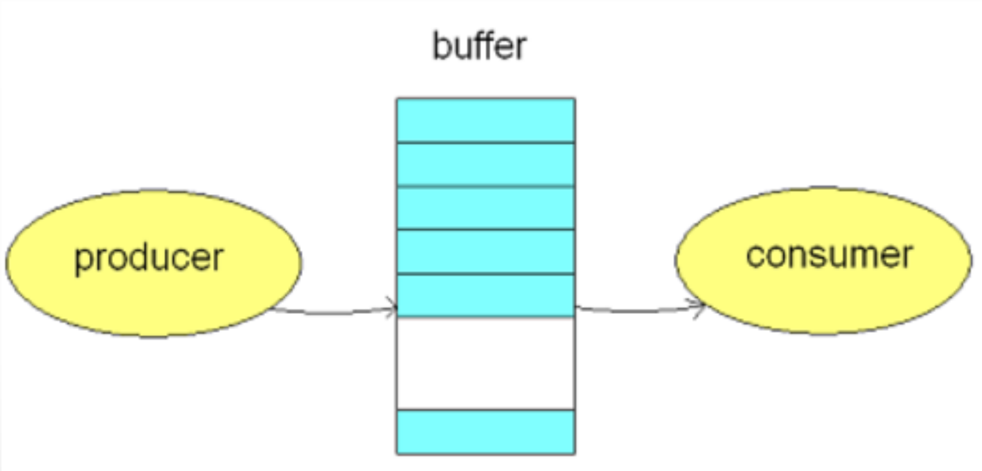
Applying a Theme to a Microsoft Word Document

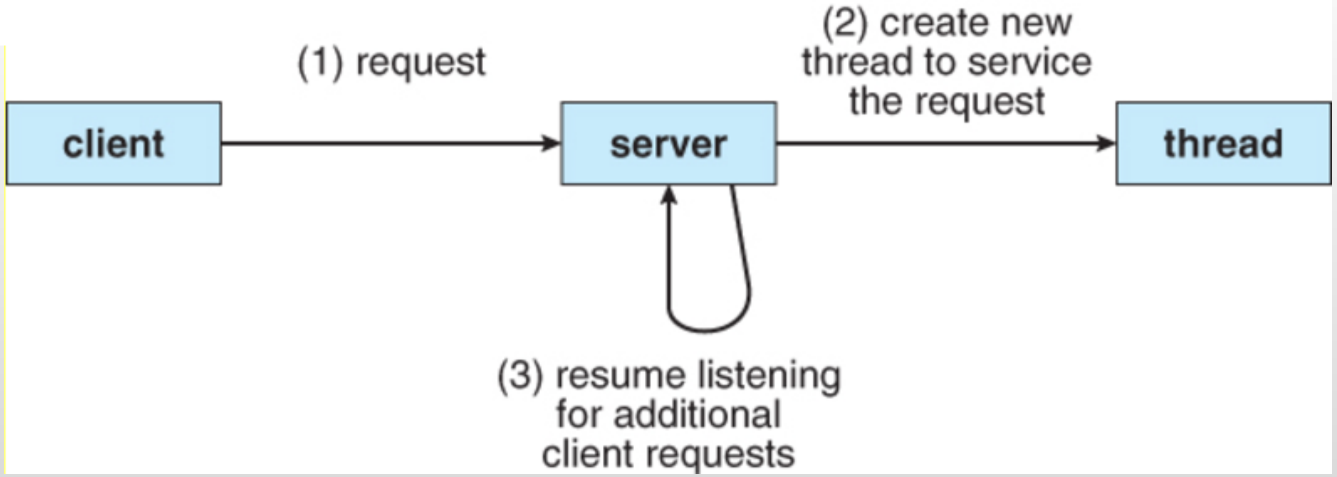
For the most part, we Scripting Guys are really the Bottom-Line Guys: our primary focus is on finding ways to help you get your job done as quickly and as efficiently as possible. If you've ever wondered why our scripts don't include a lot of fancy error-handling or why our scripts don't usually accept command-line arguments, that's the reason right there: without downplaying the importance of things like error-handling, those things can distract you from the main point of the script, the part that actually backs up an event log or stops a service. From an educational standpoint, the simpler the script the better.

In the academic world, a barebones script that merely echoes data back to the command window is more than adequate. In the real world, however, that isn't always the case. Not only do you need to generate reports in the real world, but – fairly or unfairly – you also need to generate reports that are both professional-looking and eye-catching. (As everyone knows, books typically are judged by their covers.) Because of that, we Scripting Guys are always looking for ways to have our cake and eat it, too.



Other Scenarios Threads can be Applied



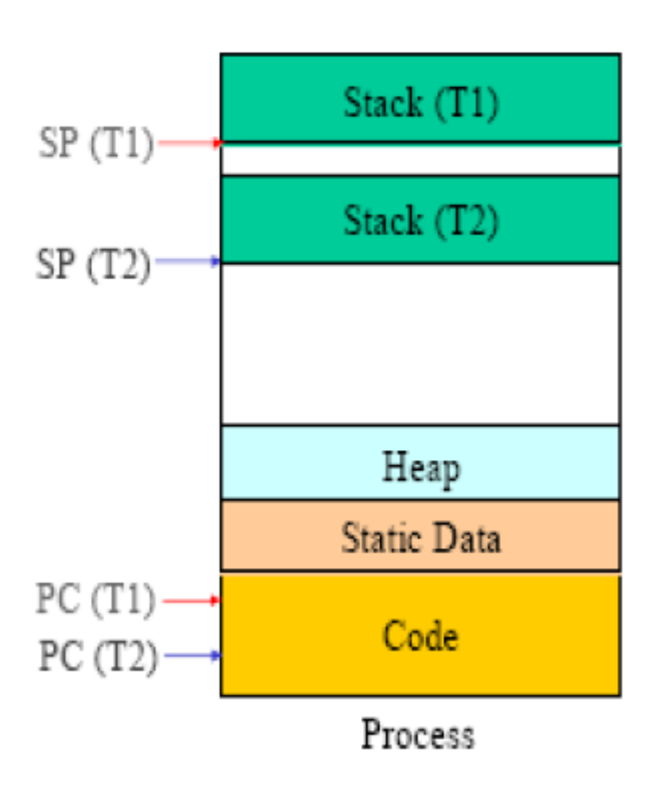

(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

Thread Vs. Process

- **Threads** are easier to create than **processes** since they don't require a separate address space.
- Threads are considered lightweight because they use far less resources than processes.
- Processes are typically independent, while threads exist as subsets of a process
- Processes have separate address spaces, whereas threads share their address space
- **Context switching** between threads in the same process is typically faster than **context switching** between processes.

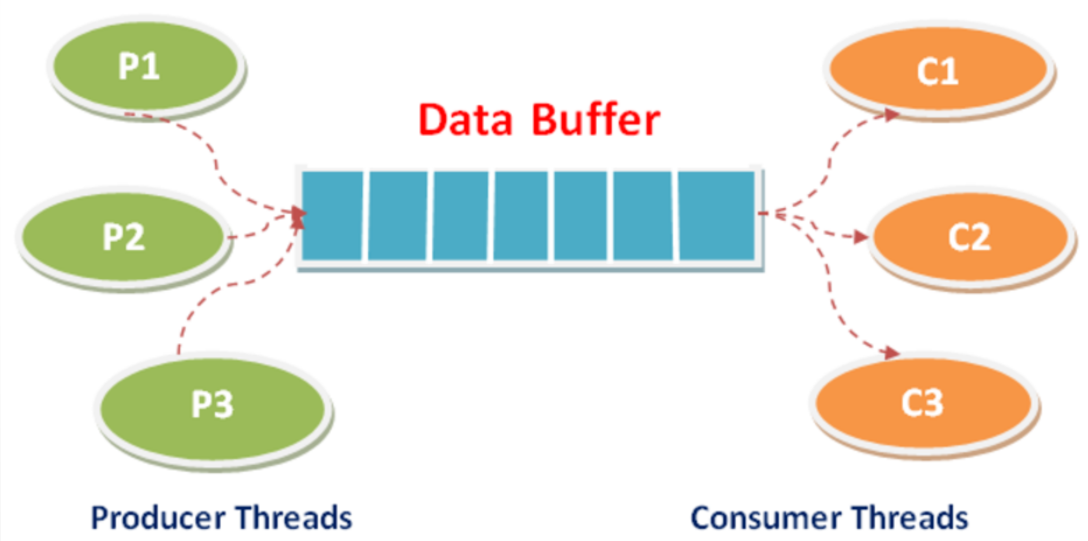
Thread Vs. Process

- A **Thread** in execution works with
  - thread ID
  - Registers (program counter and working register set)
  - Stack (for procedure call parameters, local variables etc.)
- A thread *shares* with other threads a process's (to which it belongs to)
  - Code section
  - Data section (static + heap)
  - Permissions
  - Other resources (e.g. files)

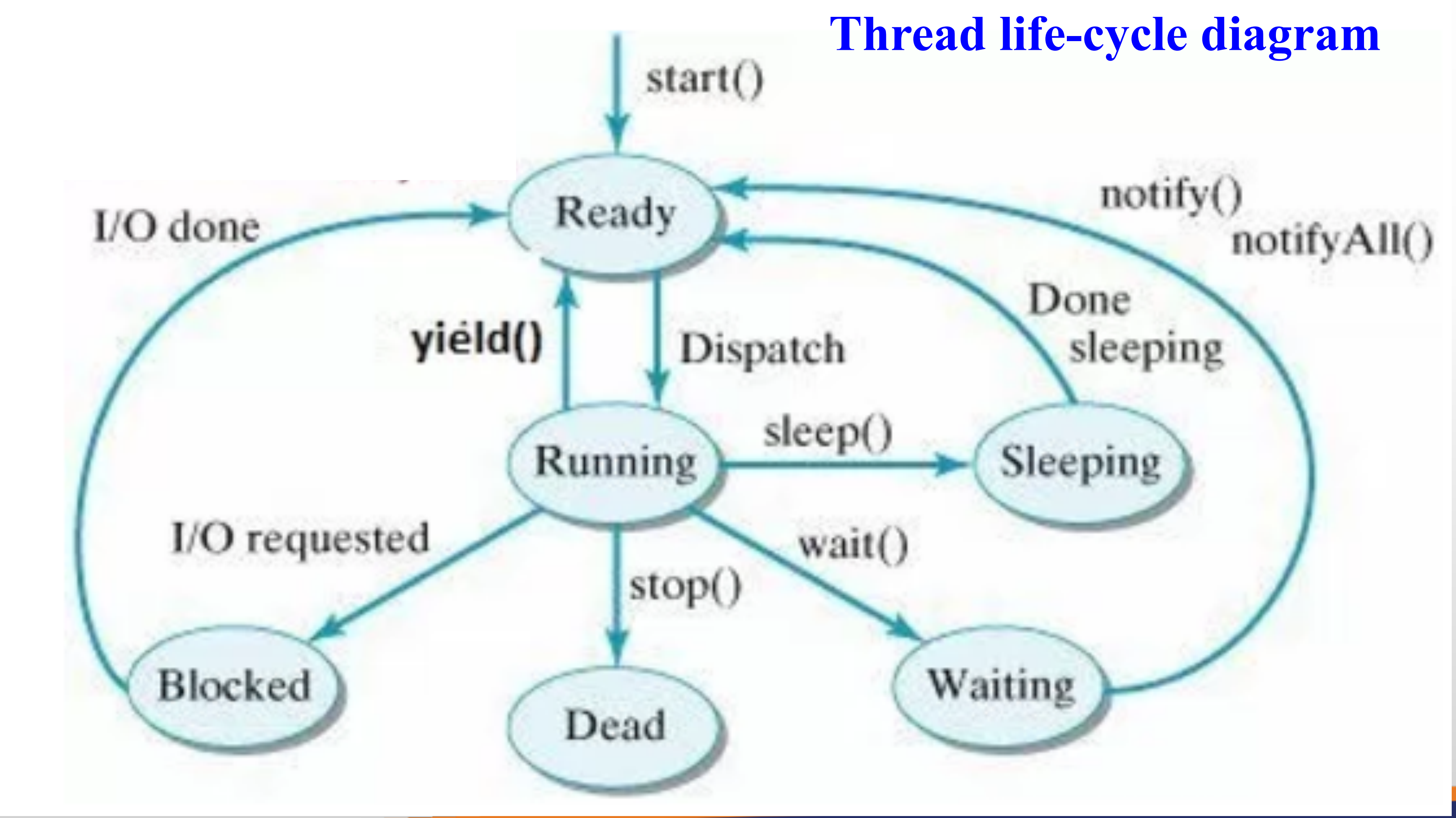


Process with 2 threads

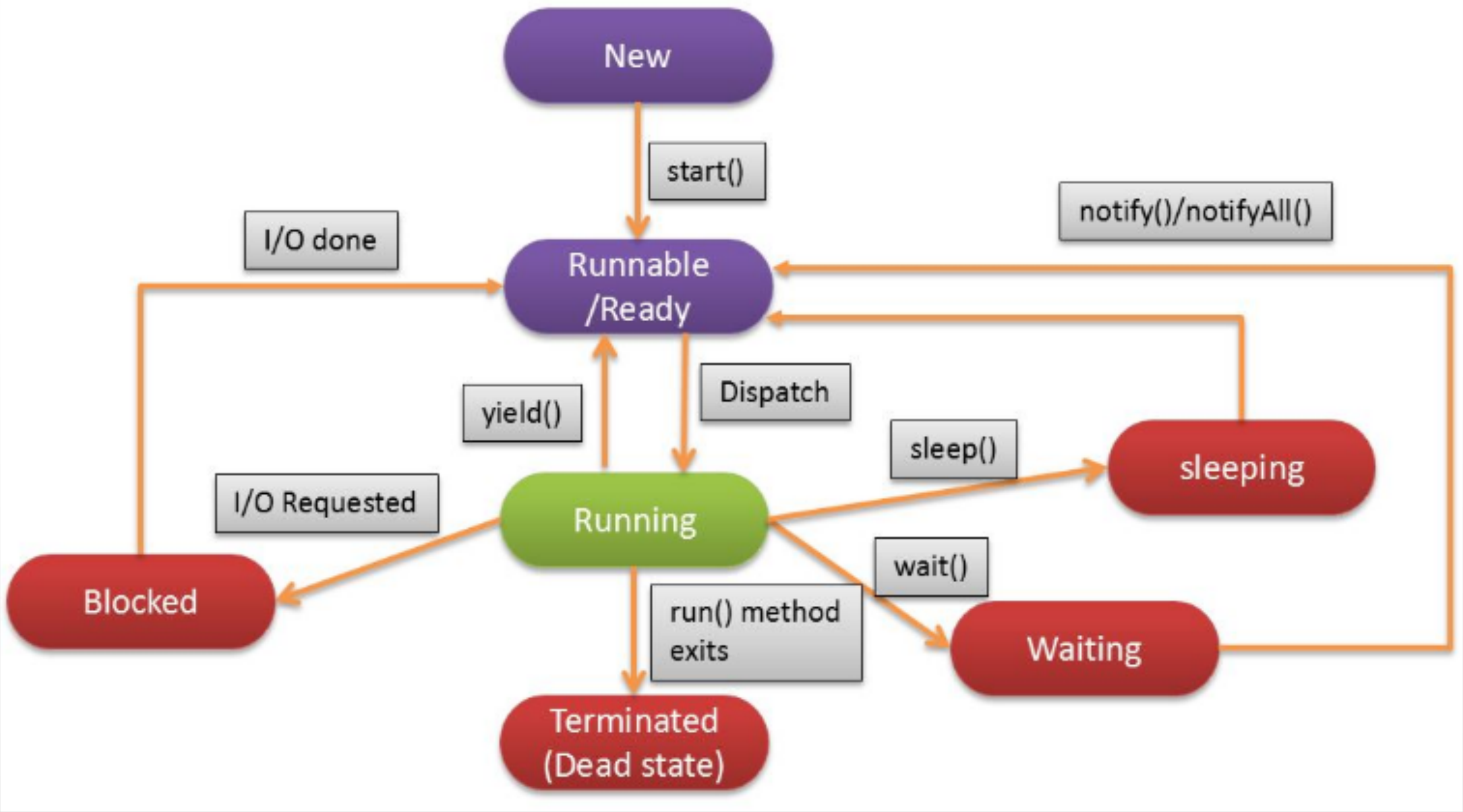
Multi-threaded Environment



Thread Life Cycle

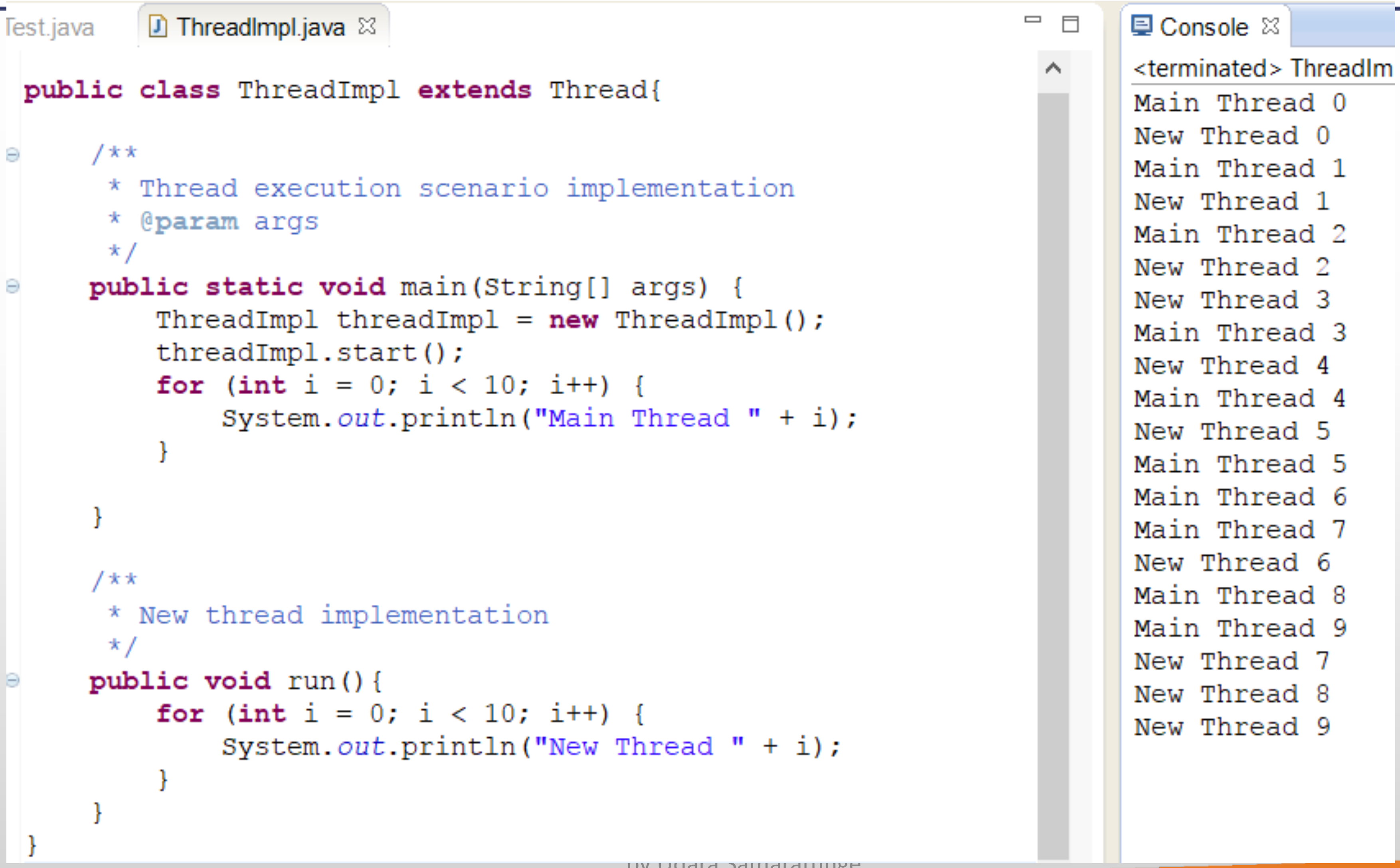


Thread States



Thread Implementation

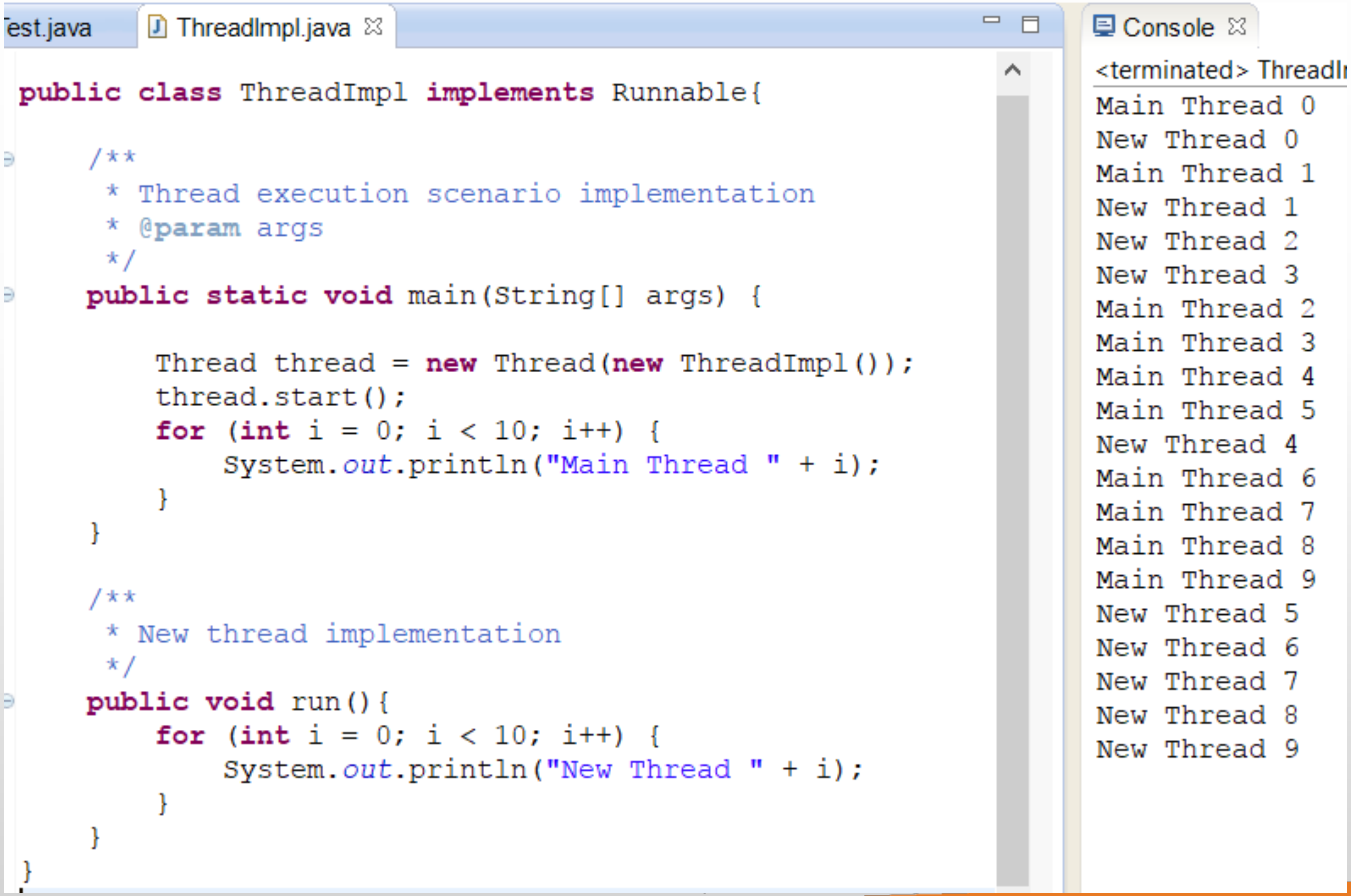
Extends Thread class



```java
public class ThreadImpl extends Thread{

    /**
     * Thread execution scenario implementation
     * @param args
     */
    public static void main(String[] args) {
        ThreadImpl threadImpl = new ThreadImpl();
        threadImpl.start();
        for (int i = 0; i < 10; i++) {
            System.out.println("Main Thread " + i);
        }

    }

    /**
     * New thread implementation
     */
    public void run(){
        for (int i = 0; i < 10; i++) {
            System.out.println("New Thread " + i);
        }
    }
}
```

```
Main Thread 0
New Thread 0
Main Thread 1
New Thread 1
Main Thread 2
New Thread 2
New Thread 3
Main Thread 3
New Thread 4
Main Thread 4
New Thread 5
Main Thread 5
Main Thread 6
Main Thread 7
New Thread 6
Main Thread 8
Main Thread 9
New Thread 7
New Thread 8
New Thread 9
```

Implements Runnable Interface

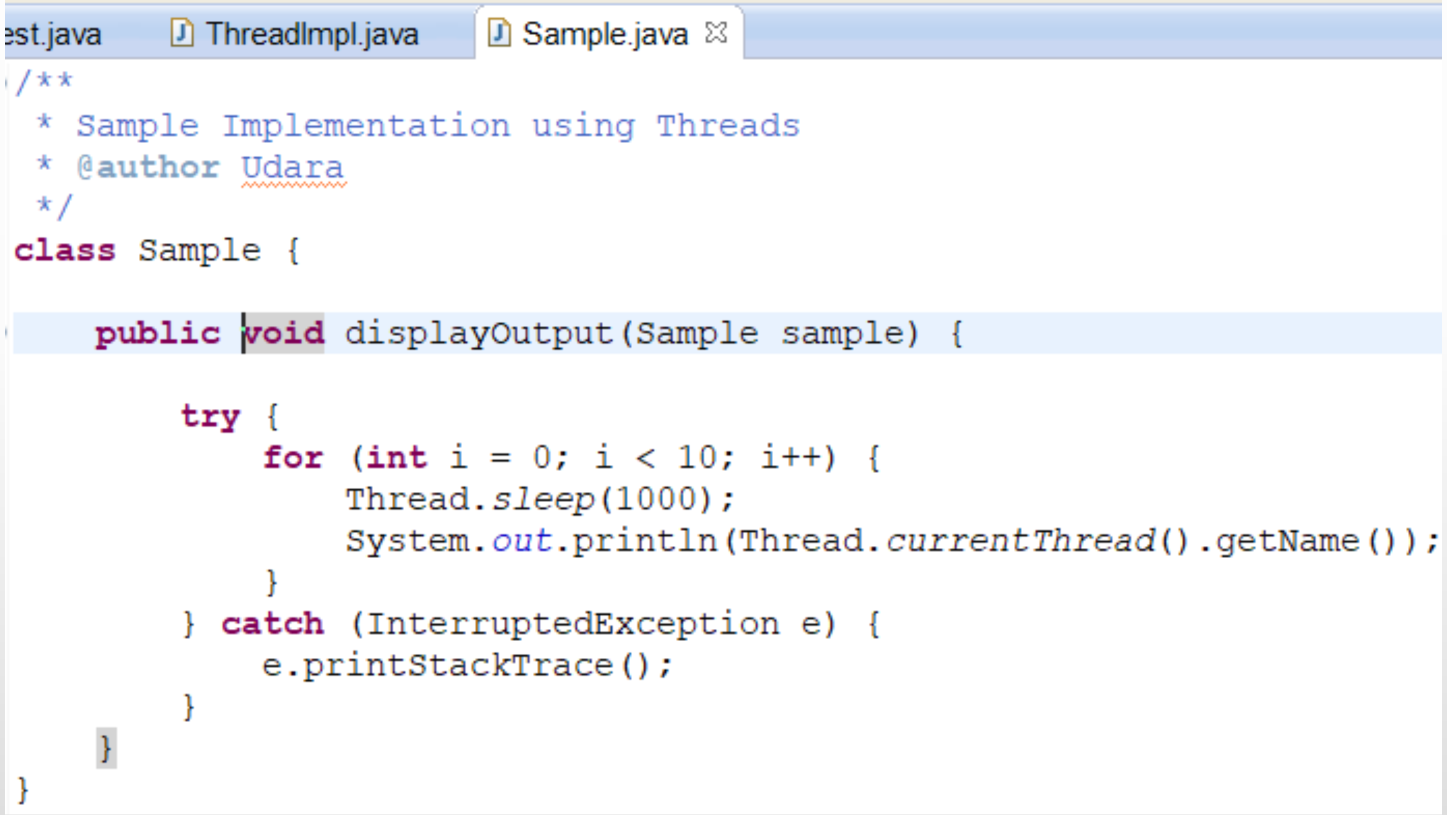


```
public class ThreadImpl implements Runnable{

    /**
     * Thread execution scenario implementation
     * @param args
     */
    public static void main(String[] args) {

        Thread thread = new Thread(new ThreadImpl());
        thread.start();
        for (int i = 0; i < 10; i++) {
            System.out.println("Main Thread " + i);
        }
    }

    /**
     * New thread implementation
     */
    public void run(){
        for (int i = 0; i < 10; i++) {
            System.out.println("New Thread " + i);
        }
    }
}
```

```
Main Thread 0
New Thread 0
Main Thread 1
New Thread 1
New Thread 2
New Thread 3
Main Thread 2
Main Thread 3
Main Thread 4
Main Thread 5
New Thread 4
Main Thread 6
Main Thread 7
Main Thread 8
Main Thread 9
New Thread 5
New Thread 6
New Thread 7
New Thread 8
New Thread 9
```

Thread Synchronization

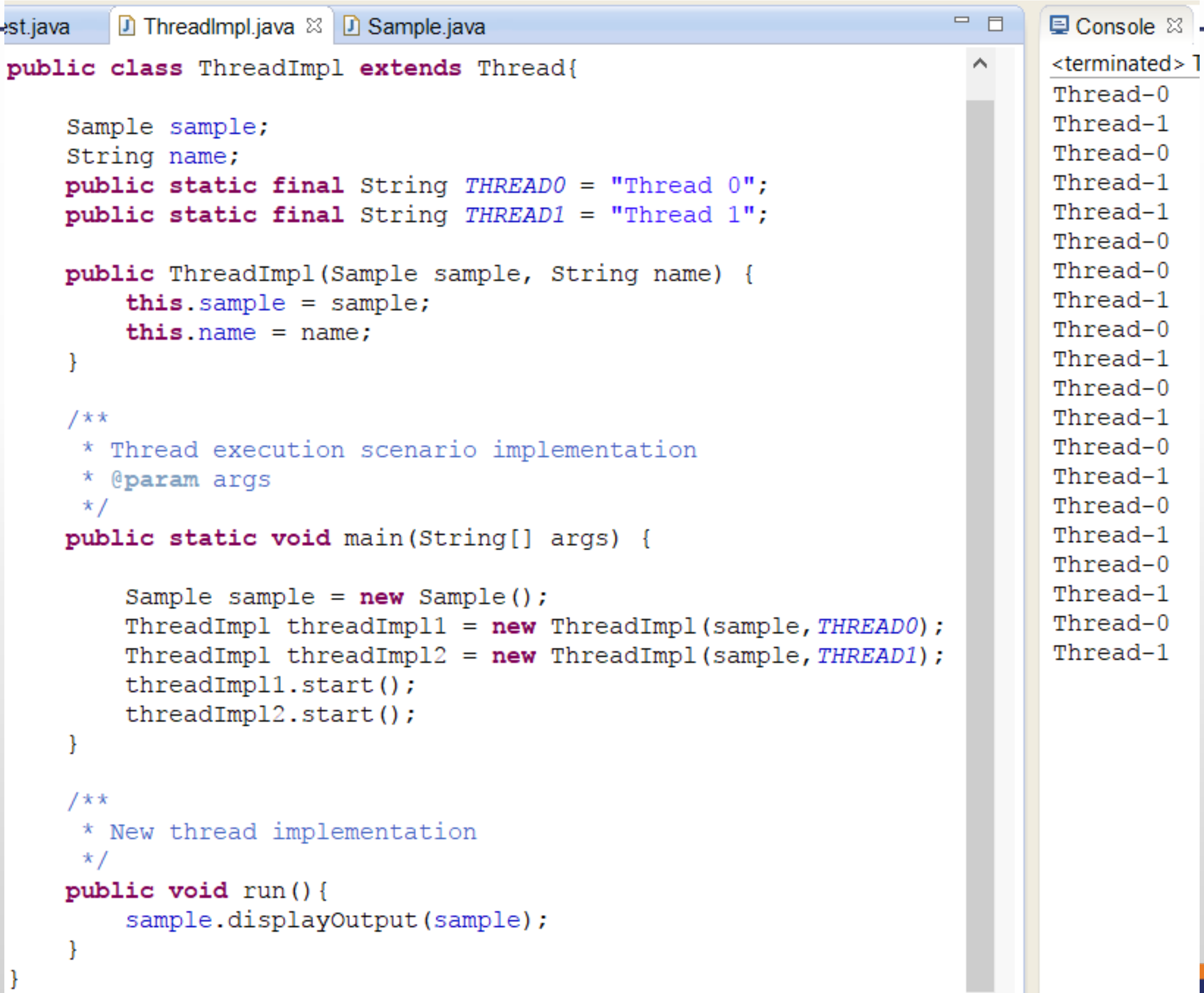
Threads are not synchronized



```
/**
 * Sample Implementation using Threads
 * @author Udara
 */
class Sample {

    public void displayOutput(Sample sample) {

        try {
            for (int i = 0; i < 10; i++) {
                Thread.sleep(1000);
                System.out.println(Thread.currentThread().getName());
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

- When Thread sleeps it throws InterruptedException
- When Thread sleep it keeps the lock with it

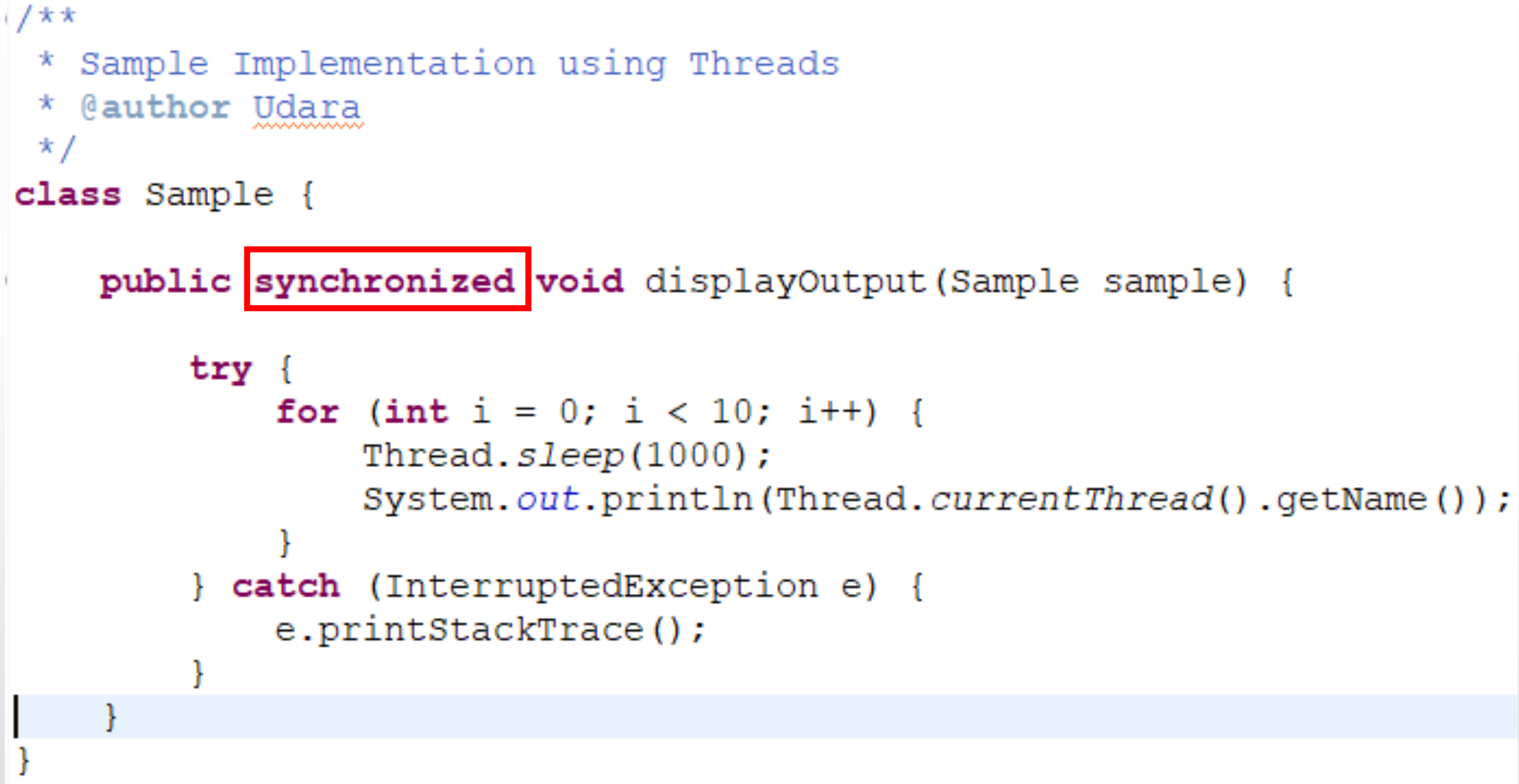
Threads are not synchronized



```java
public class ThreadImpl extends Thread{

    Sample sample;
    String name;
    public static final String THREAD0 = "Thread 0";
    public static final String THREAD1 = "Thread 1";

    public ThreadImpl(Sample sample, String name) {
        this.sample = sample;
        this.name = name;
    }

    /**
     * Thread execution scenario implementation
     * @param args
     */
    public static void main(String[] args) {

        Sample sample = new Sample();
        ThreadImpl threadImpl1 = new ThreadImpl(sample,THREAD0);
        ThreadImpl threadImpl2 = new ThreadImpl(sample,THREAD1);
        threadImpl1.start();
        threadImpl2.start();
    }

    /**
     * New thread implementation
     */
    public void run(){
        sample.displayOutput(sample);
    }
}
```

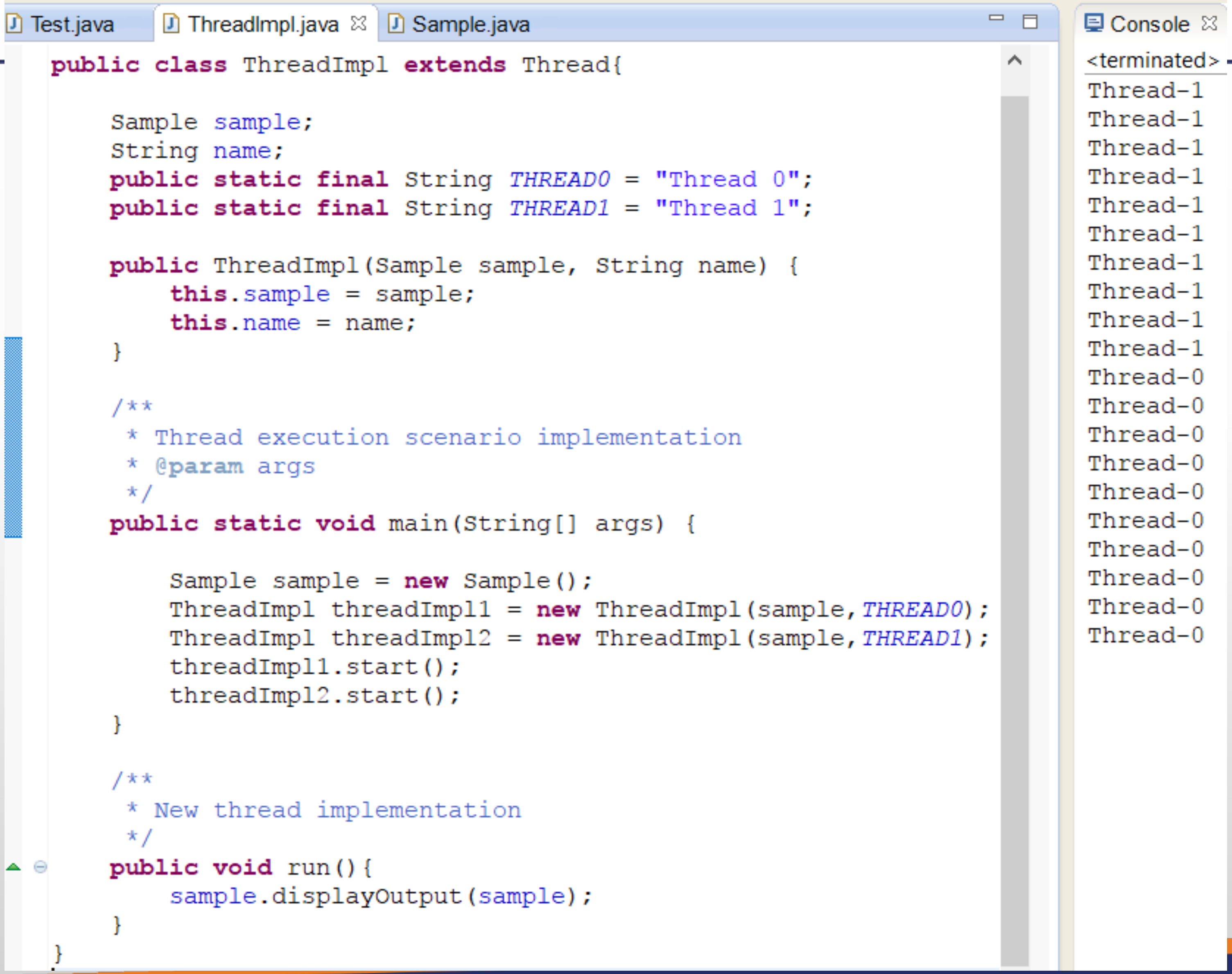
```
Thread-0
Thread-1
Thread-0
Thread-1
Thread-1
Thread-0
Thread-0
Thread-1
Thread-0
Thread-1
Thread-0
Thread-1
Thread-0
Thread-1
Thread-0
Thread-1
Thread-0
Thread-1
Thread-0
Thread-1
```

Thread Synchronized Method



```
/**
 * Sample Implementation using Threads
 * @author Udara
 */
class Sample {

    public synchronized void displayOutput(Sample sample) {

        try {
            for (int i = 0; i < 10; i++) {
                Thread.sleep(1000);
                System.out.println(Thread.currentThread().getName());
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

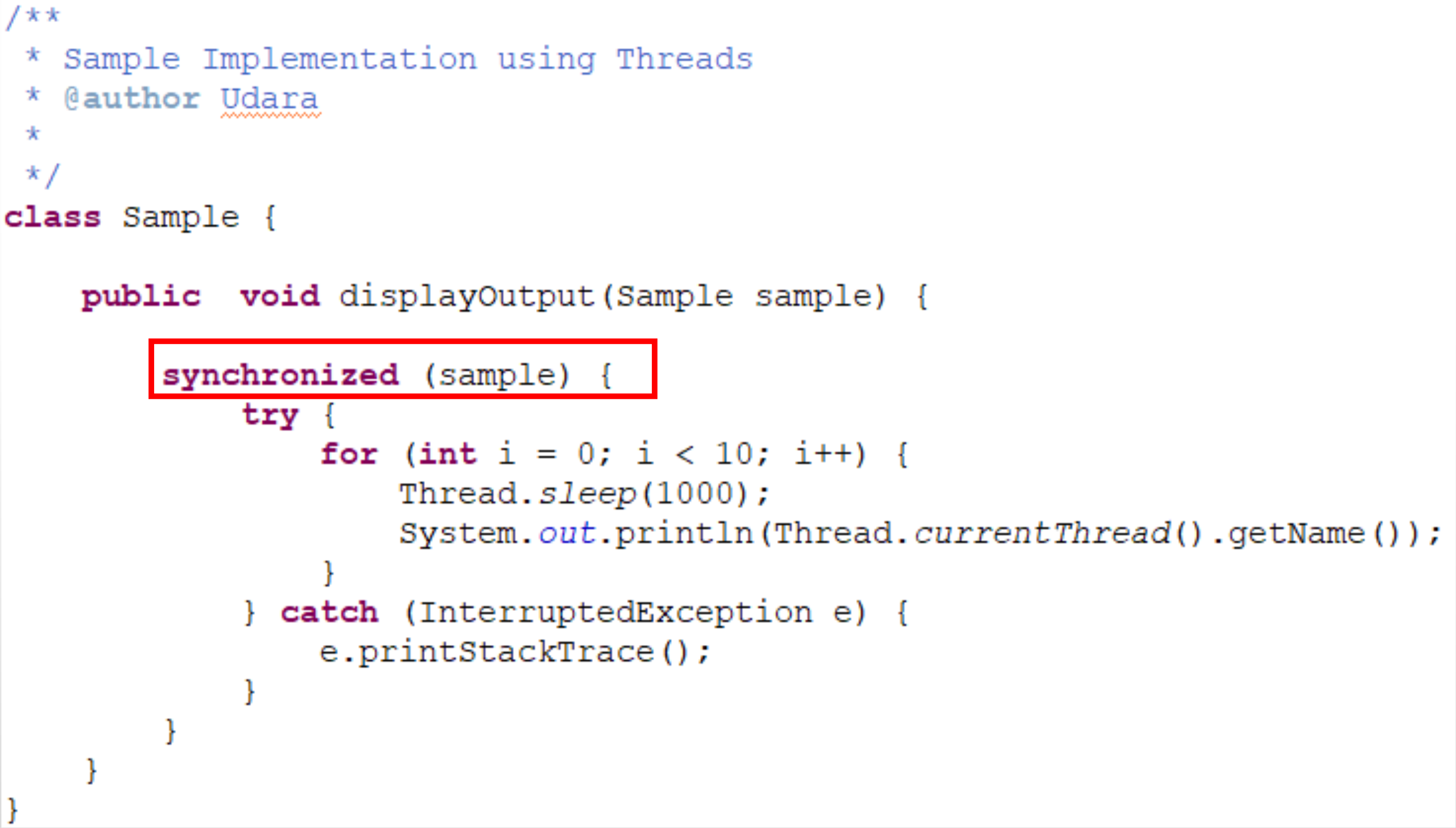
Thread Synchronization



```
public class ThreadImpl extends Thread{

    Sample sample;
    String name;
    public static final String THREAD0 = "Thread 0";
    public static final String THREAD1 = "Thread 1";

    public ThreadImpl(Sample sample, String name) {
        this.sample = sample;
        this.name = name;
    }

    /**
     * Thread execution scenario implementation
     * @param args
     */
    public static void main(String[] args) {

        Sample sample = new Sample();
        ThreadImpl threadImpl1 = new ThreadImpl(sample,THREAD0);
        ThreadImpl threadImpl2 = new ThreadImpl(sample,THREAD1);
        threadImpl1.start();
        threadImpl2.start();
    }

    /**
     * New thread implementation
     */
    public void run(){
        sample.displayOutput(sample);
    }
}
```

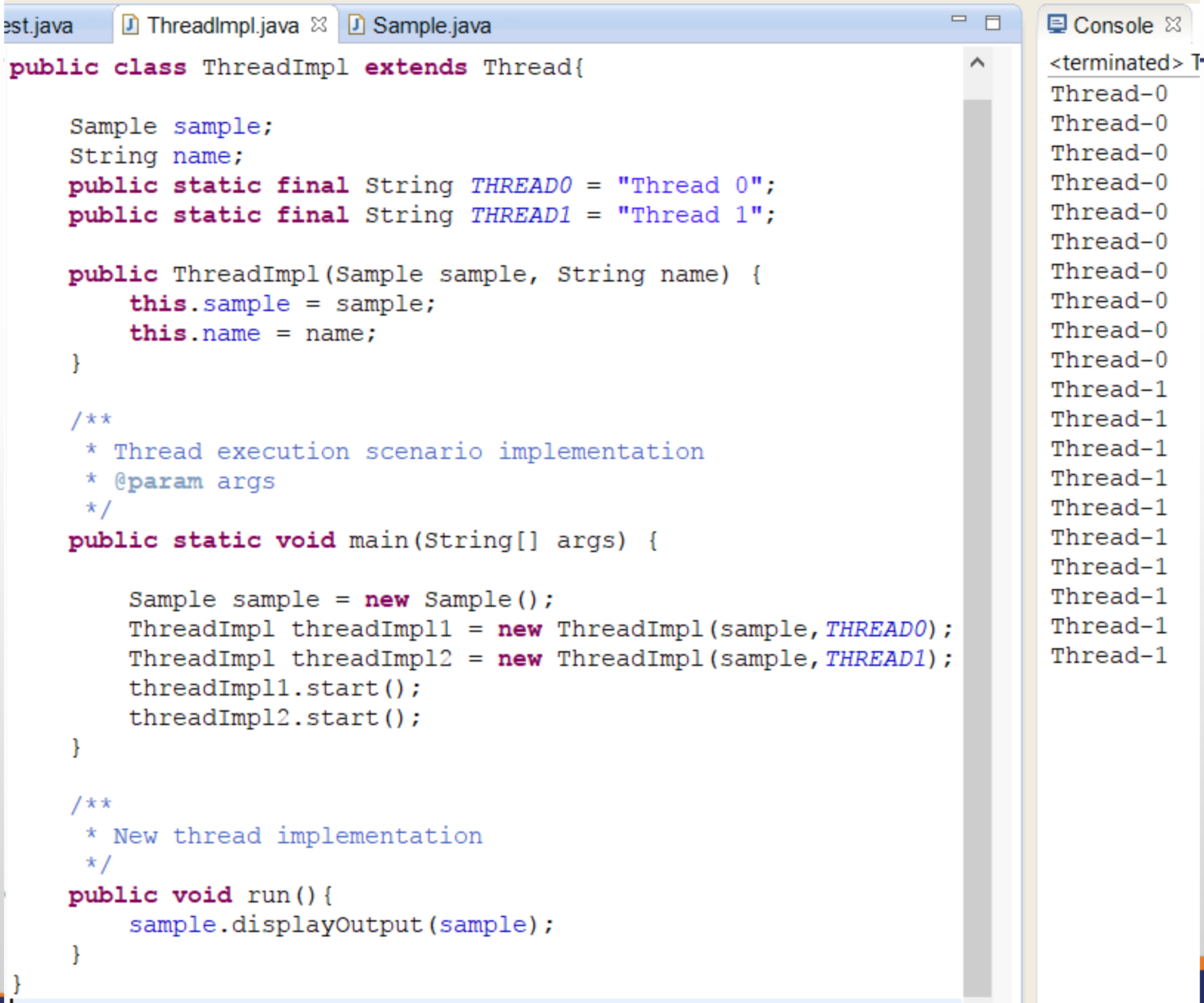
```
<terminated>
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
```

Thread Synchronization block



```
/**
 * Sample Implementation using Threads
 * @author Udara
 *
 */
class Sample {

    public  void displayOutput(Sample sample) {

        synchronized (sample) {
            try {
                for (int i = 0; i < 10; i++) {
                    Thread.sleep(1000);
                    System.out.println(Thread.currentThread().getName());
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

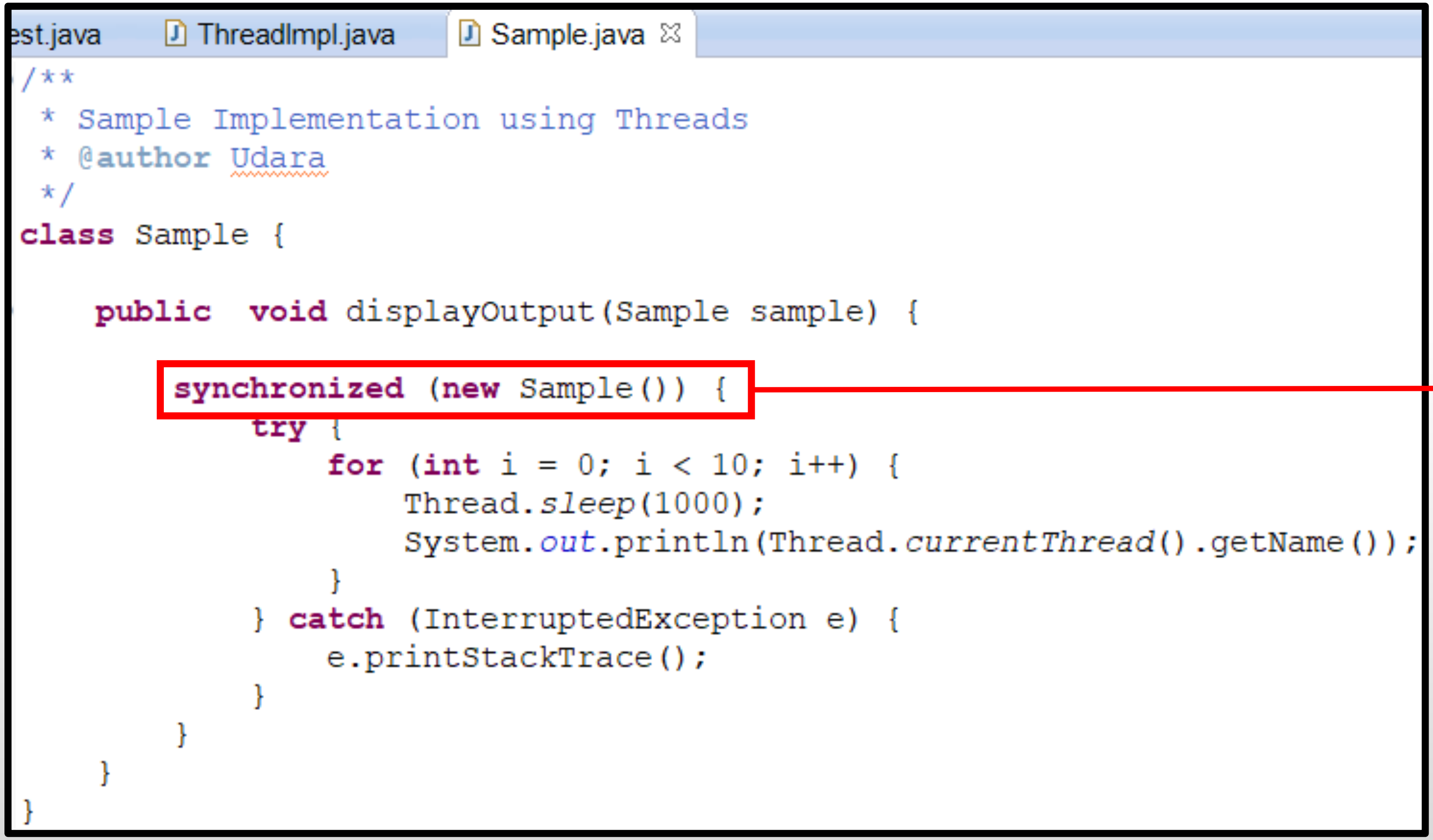
Thread Synchronization block



```java
public class ThreadImpl extends Thread{

    Sample sample;
    String name;
    public static final String THREAD0 = "Thread 0";
    public static final String THREAD1 = "Thread 1";

    public ThreadImpl(Sample sample, String name) {
        this.sample = sample;
        this.name = name;
    }

    /**
     * Thread execution scenario implementation
     * @param args
     */
    public static void main(String[] args) {

        Sample sample = new Sample();
        ThreadImpl threadImpl1 = new ThreadImpl(sample,THREAD0);
        ThreadImpl threadImpl2 = new ThreadImpl(sample,THREAD1);
        threadImpl1.start();
        threadImpl2.start();
    }

    /**
     * New thread implementation
     */
    public void run(){
        sample.displayOutput(sample);
    }
}
```

```
<terminated> T
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
```

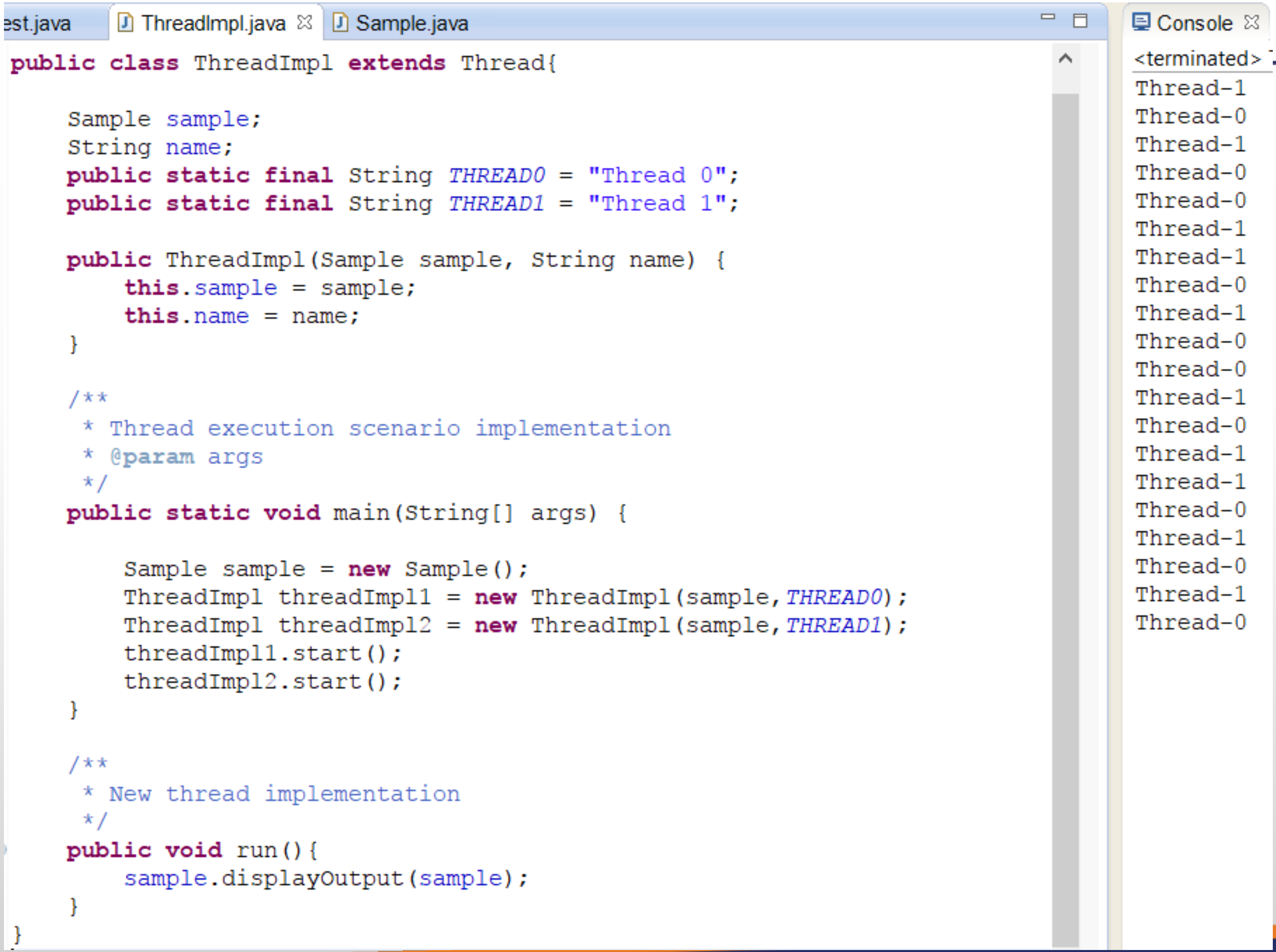
Thread Synchronization block with lock change



```java
/**
 * Sample Implementation using Threads
 * @author Udara
 */
class Sample {

    public  void displayOutput(Sample sample) {

        synchronized (new Sample()) {
            try {
                for (int i = 0; i < 10; i++) {
                    Thread.sleep(1000);
                    System.out.println(Thread.currentThread().getName());
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Each Thread get separate object to lock. So method not synchronized

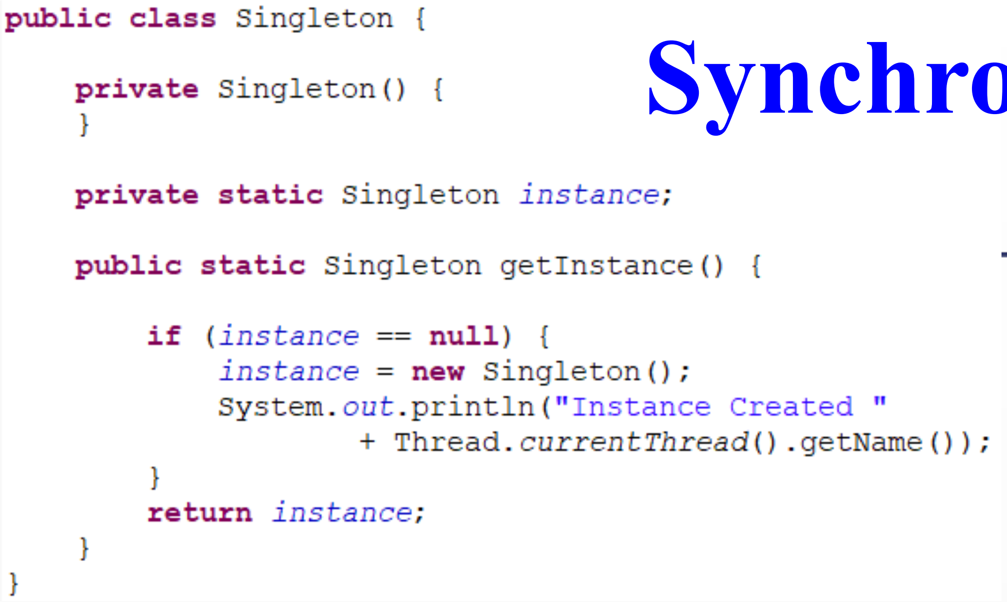
Thread Synchronization block with lock change



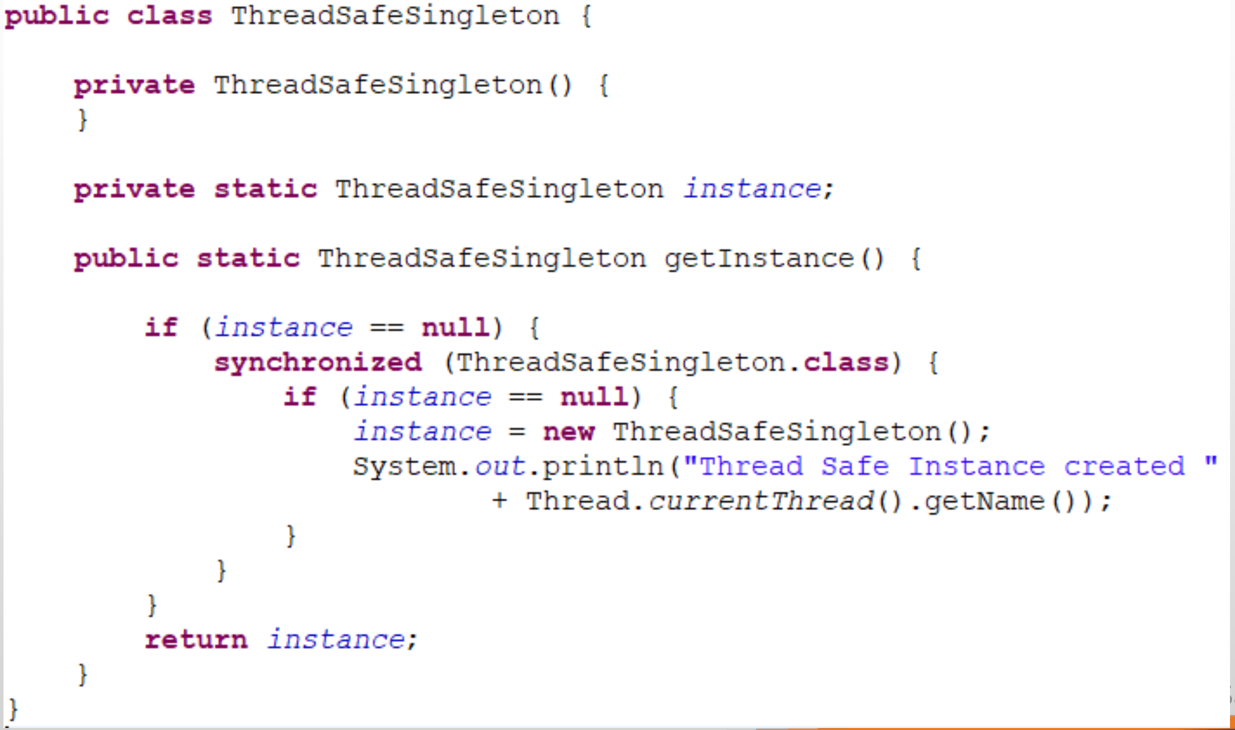
```java
public class ThreadImpl extends Thread{

    Sample sample;
    String name;
    public static final String THREAD0 = "Thread 0";
    public static final String THREAD1 = "Thread 1";

    public ThreadImpl(Sample sample, String name) {
        this.sample = sample;
        this.name = name;
    }

    /**
     * Thread execution scenario implementation
     * @param args
     */
    public static void main(String[] args) {

        Sample sample = new Sample();
        ThreadImpl threadImpl1 = new ThreadImpl(sample,THREAD0);
        ThreadImpl threadImpl2 = new ThreadImpl(sample,THREAD1);
        threadImpl1.start();
        threadImpl2.start();
    }

    /**
     * New thread implementation
     */
    public void run(){
        sample.displayOutput(sample);
    }
}
```

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

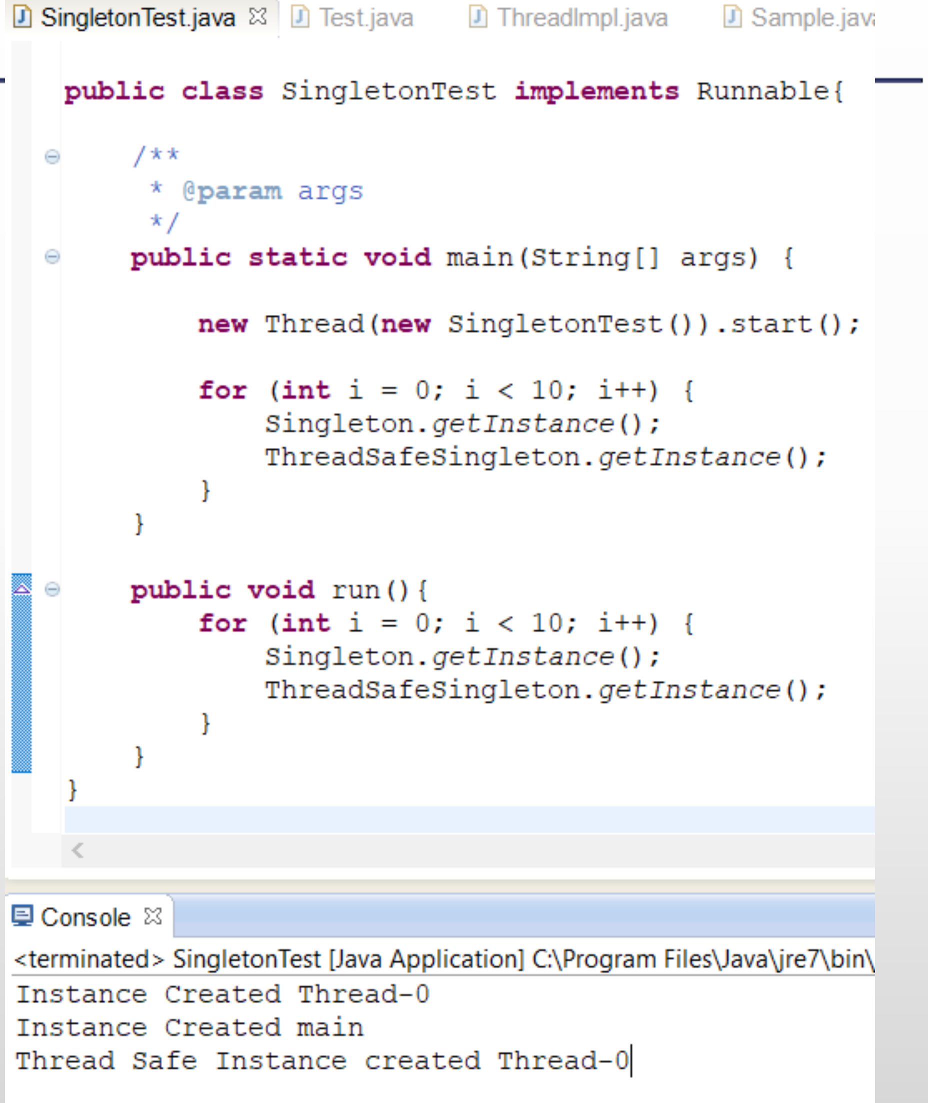
Synchronized block importance



```
public class Singleton {

    private Singleton() {
    }

    private static Singleton instance;

    public static Singleton getInstance() {

        if (instance == null) {
            instance = new Singleton();
            System.out.println("Instance Created "
                    + Thread.currentThread().getName());
        }
        return instance;
    }
}
```



```
public class ThreadSafeSingleton {

    private ThreadSafeSingleton() {
    }

    private static ThreadSafeSingleton instance;

    public static ThreadSafeSingleton getInstance() {

        if (instance == null) {
            synchronized (ThreadSafeSingleton.class) {
                if (instance == null) {
                    instance = new ThreadSafeSingleton();
                    System.out.println("Thread Safe Instance created "
                            + Thread.currentThread().getName());
                }
            }
        }
        return instance;
    }
}
```

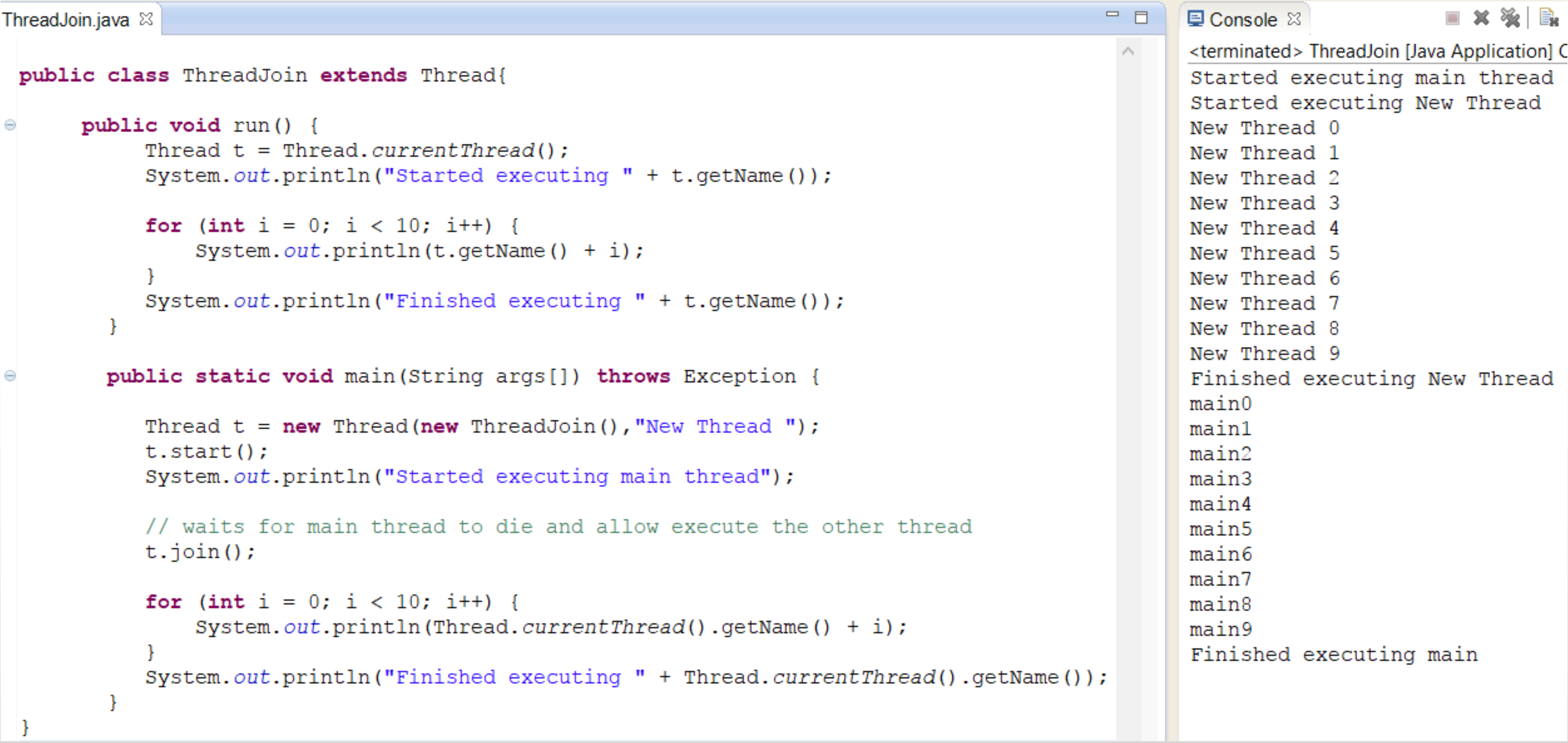


```
public class SingletonTest implements Runnable{

    /**
     * @param args
     */
    public static void main(String[] args) {

        new Thread(new SingletonTest()).start();

        for (int i = 0; i < 10; i++) {
            Singleton.getInstance();
            ThreadSafeSingleton.getInstance();
        }
    }

    public void run(){
        for (int i = 0; i < 10; i++) {
            Singleton.getInstance();
            ThreadSafeSingleton.getInstance();
        }
    }
}
```

```
<terminated> SingletonTest [Java Application] C:\Program Files\Java\jre7\bin\
Instance Created Thread-0
Instance Created main
Thread Safe Instance created Thread-0
```

Thread Join method

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

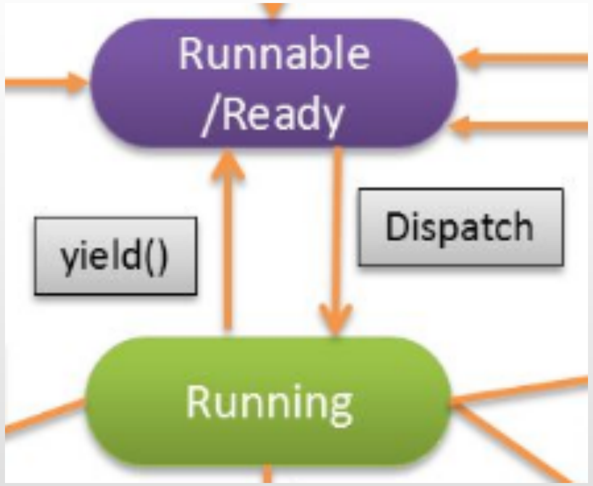


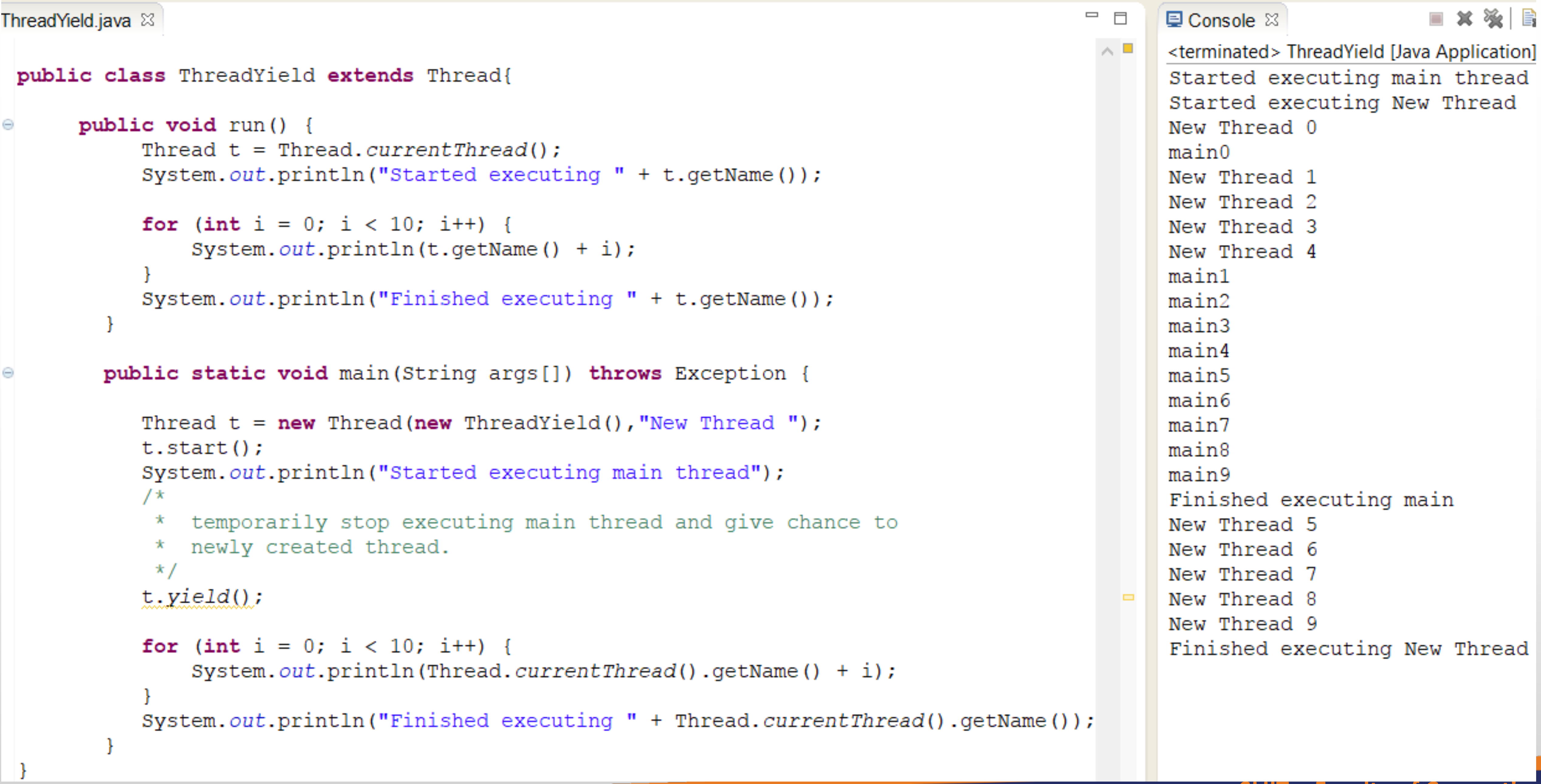
```java
public class ThreadJoin extends Thread{

        public void run() {
            Thread t = Thread.currentThread();
            System.out.println("Started executing " + t.getName());

            for (int i = 0; i < 10; i++) {
                System.out.println(t.getName() + i);
            }
            System.out.println("Finished executing " + t.getName());
        }

        public static void main(String args[]) throws Exception {

            Thread t = new Thread(new ThreadJoin(),"New Thread ");
            t.start();
            System.out.println("Started executing main thread");

            // waits for main thread to die and allow execute the other thread
            t.join();

            for (int i = 0; i < 10; i++) {
                System.out.println(Thread.currentThread().getName() + i);
            }
            System.out.println("Finished executing " + Thread.currentThread().getName());
        }
}
```

```
Started executing main thread
Started executing New Thread
New Thread 0
New Thread 1
New Thread 2
New Thread 3
New Thread 4
New Thread 5
New Thread 6
New Thread 7
New Thread 8
New Thread 9
Finished executing New Thread
main0
main1
main2
main3
main4
main5
main6
main7
main8
main9
Finished executing main
```

Thread Yield method

- Yield() is used to give the other threads of the same priority a chance to execute
- This causes current running thread to move to runnable state. [running state to ready state]





```
public class ThreadYield extends Thread{

     public void run() {
          Thread t = Thread.currentThread();
          System.out.println("Started executing " + t.getName());

          for (int i = 0; i < 10; i++) {
              System.out.println(t.getName() + i);
          }
          System.out.println("Finished executing " + t.getName());
     }

       public static void main(String args[]) throws Exception {

          Thread t = new Thread(new ThreadYield(),"New Thread ");
          t.start();
          System.out.println("Started executing main thread");
          /*
           *  temporarily stop executing main thread and give chance to
           *  newly created thread.
           */
          t.yield();

          for (int i = 0; i < 10; i++) {
              System.out.println(Thread.currentThread().getName() + i);
          }
          System.out.println("Finished executing " + Thread.currentThread().getName());
     }
}
```

```
<terminated> ThreadYield [Java Application]
Started executing main thread
Started executing New Thread
New Thread 0
main0
New Thread 1
New Thread 2
New Thread 3
New Thread 4
main1
main2
main3
main4
main5
main6
main7
main8
main9
Finished executing main
New Thread 5
New Thread 6
New Thread 7
New Thread 8
New Thread 9
Finished executing New Thread
```

Thread wait and notify

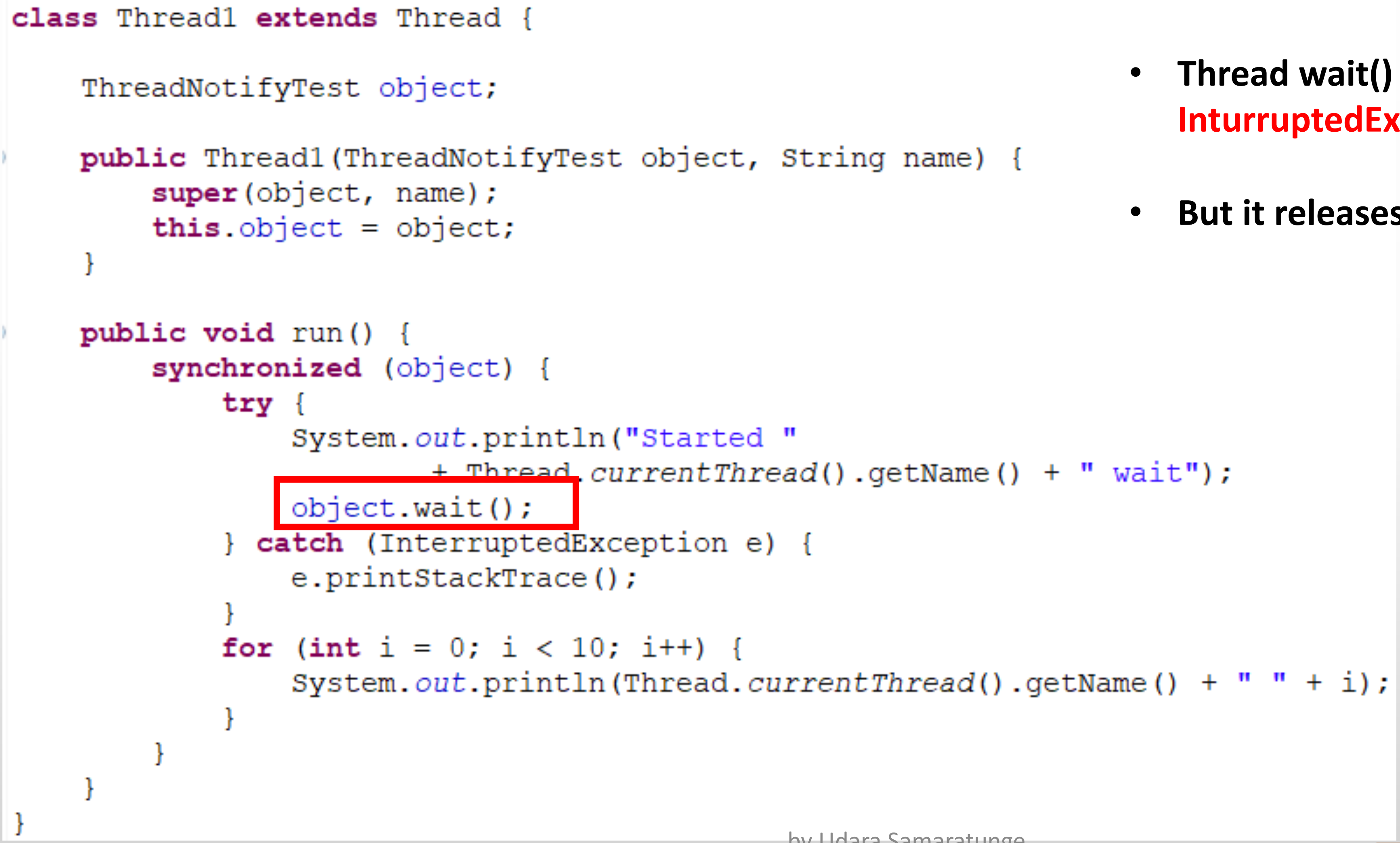
- Once thread executes wait() method it releases the lock and state changed from **Runnable** to **waiting state**.

- Other thread can acquire the lock and continue execution.

- Once notify() method get executed the waited thread move to ready state and resume its execution.

- notifyAll() This wakes up all the threads that called wait() on the same object.

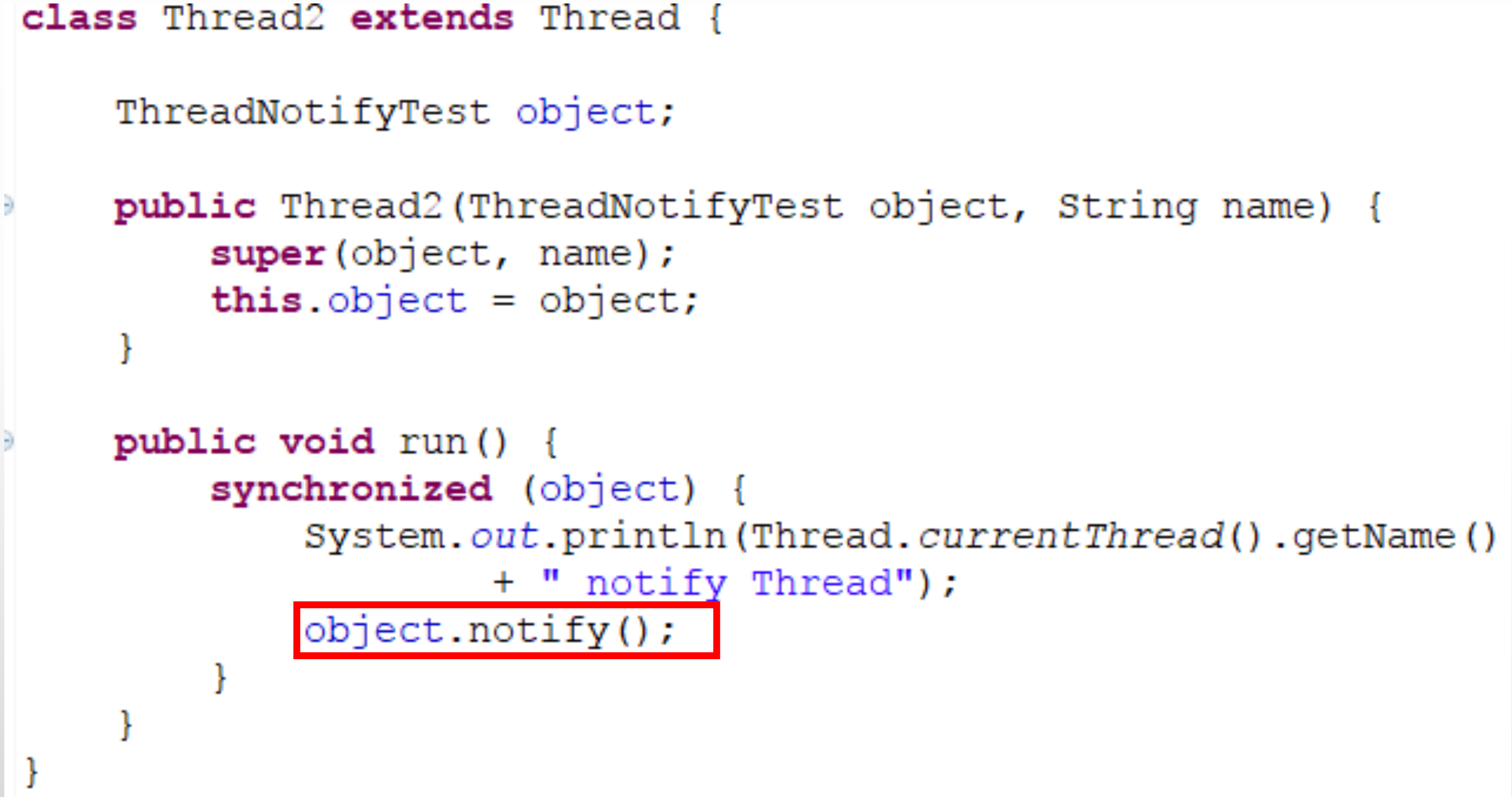
Thread wait and notify



```
class Thread1 extends Thread {

    ThreadNotifyTest object;

    public Thread1(ThreadNotifyTest object, String name) {
        super(object, name);
        this.object = object;
    }

    public void run() {
        synchronized (object) {
            try {
                System.out.println("Started "
                        + Thread.currentThread().getName() + " wait");
                object.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            for (int i = 0; i < 10; i++) {
                System.out.println(Thread.currentThread().getName() + " " + i);
            }
        }
    }
}
```

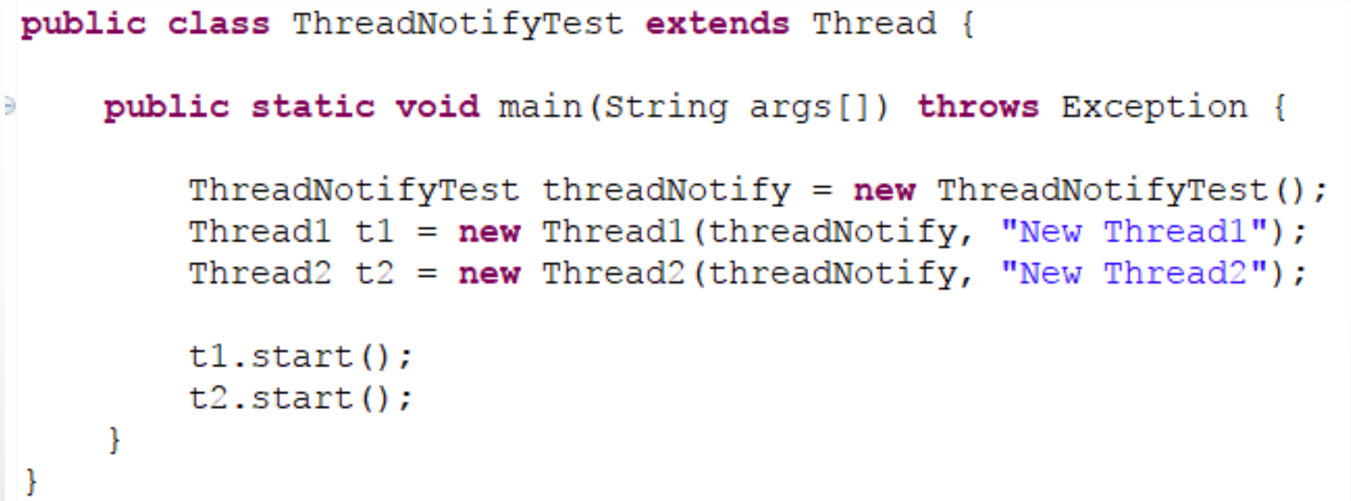
- **Thread wait() method throw an InturruptedException**
- **But it releases the lock**

Thread wait and notify



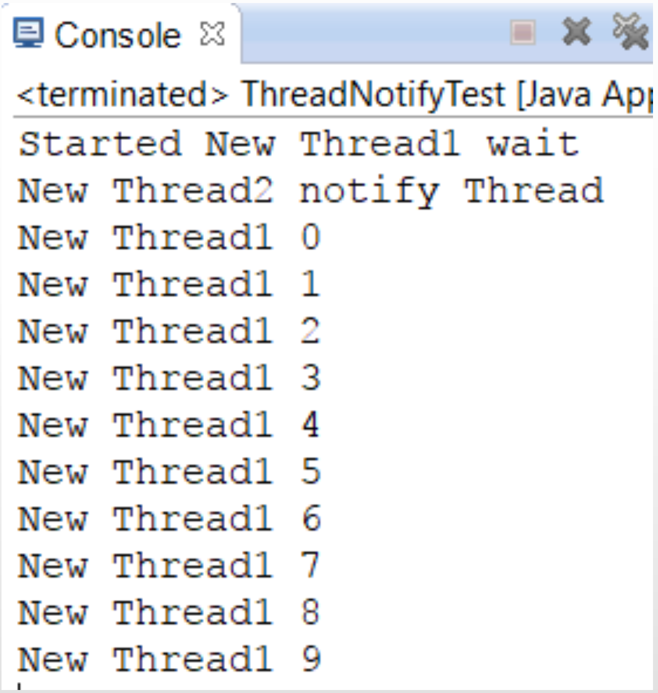
```
class Thread2 extends Thread {

    ThreadNotifyTest object;

    public Thread2(ThreadNotifyTest object, String name) {
        super(object, name);
        this.object = object;
    }

    public void run() {
        synchronized (object) {
            System.out.println(Thread.currentThread().getName()
                    + " notify Thread");
            object.notify();
        }
    }
}
```

Thread wait and notify



```java
public class ThreadNotifyTest extends Thread {

    public static void main(String args[]) throws Exception {

        ThreadNotifyTest threadNotify = new ThreadNotifyTest();
        Thread1 t1 = new Thread1(threadNotify, "New Thread1");
        Thread2 t2 = new Thread2(threadNotify, "New Thread2");

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

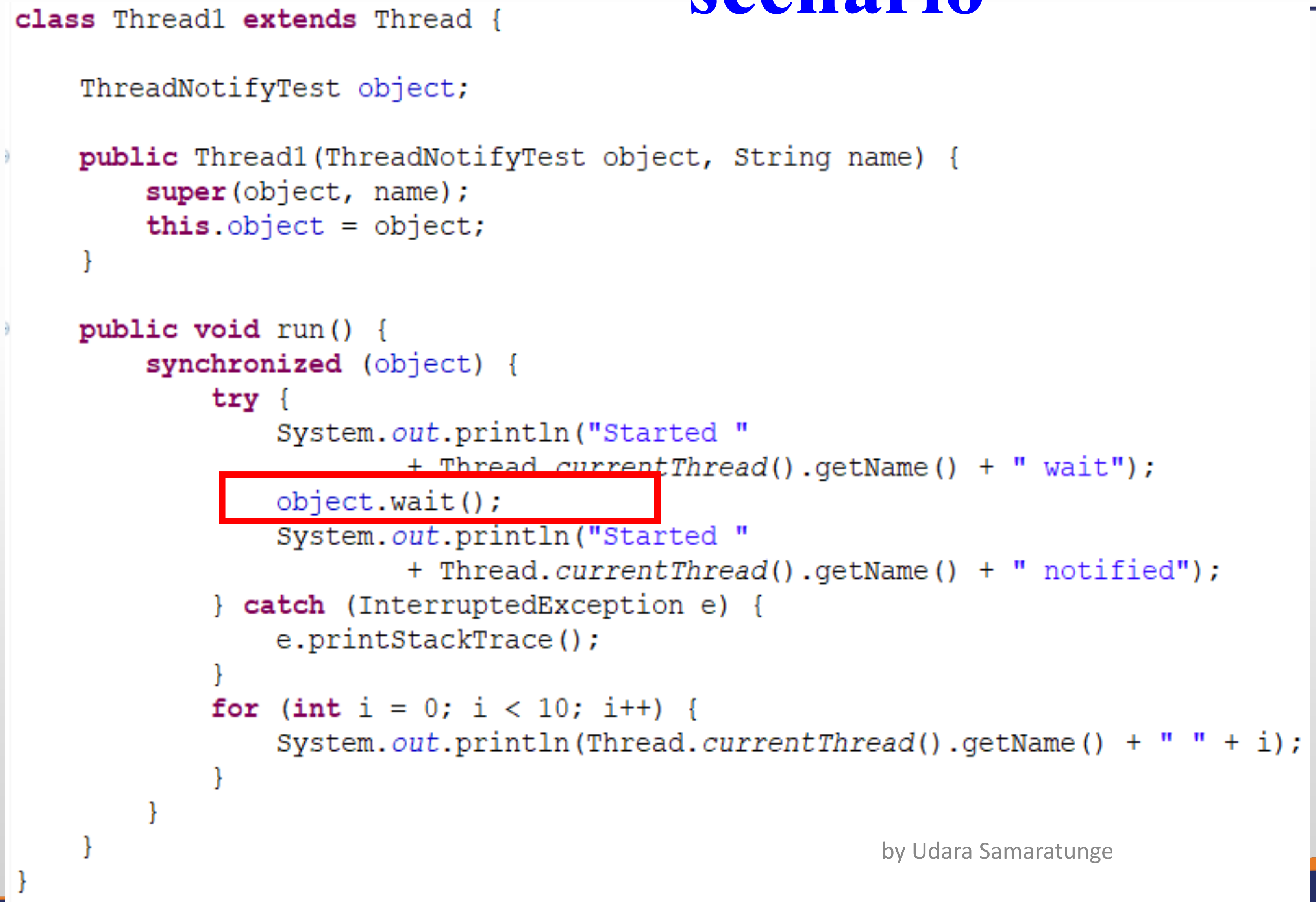
Response



<terminated> ThreadNotifyTest [Java Ap

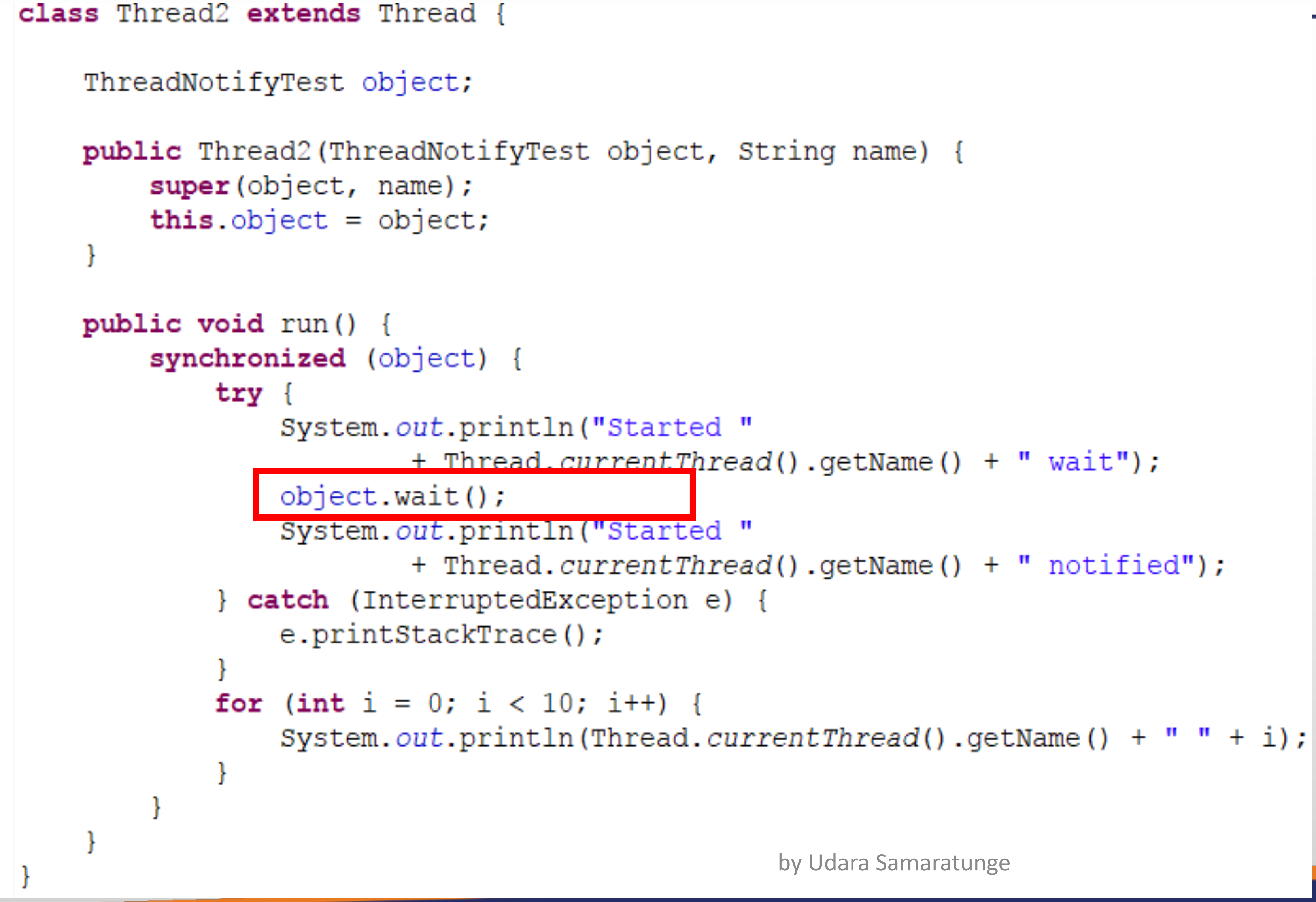
```
Started New Thread1 wait
New Thread2 notify Thread
New Thread1 0
New Thread1 1
New Thread1 2
New Thread1 3
New Thread1 4
New Thread1 5
New Thread1 6
New Thread1 7
New Thread1 8
New Thread1 9
```

Example for notifyAll with multi-threaded scenario



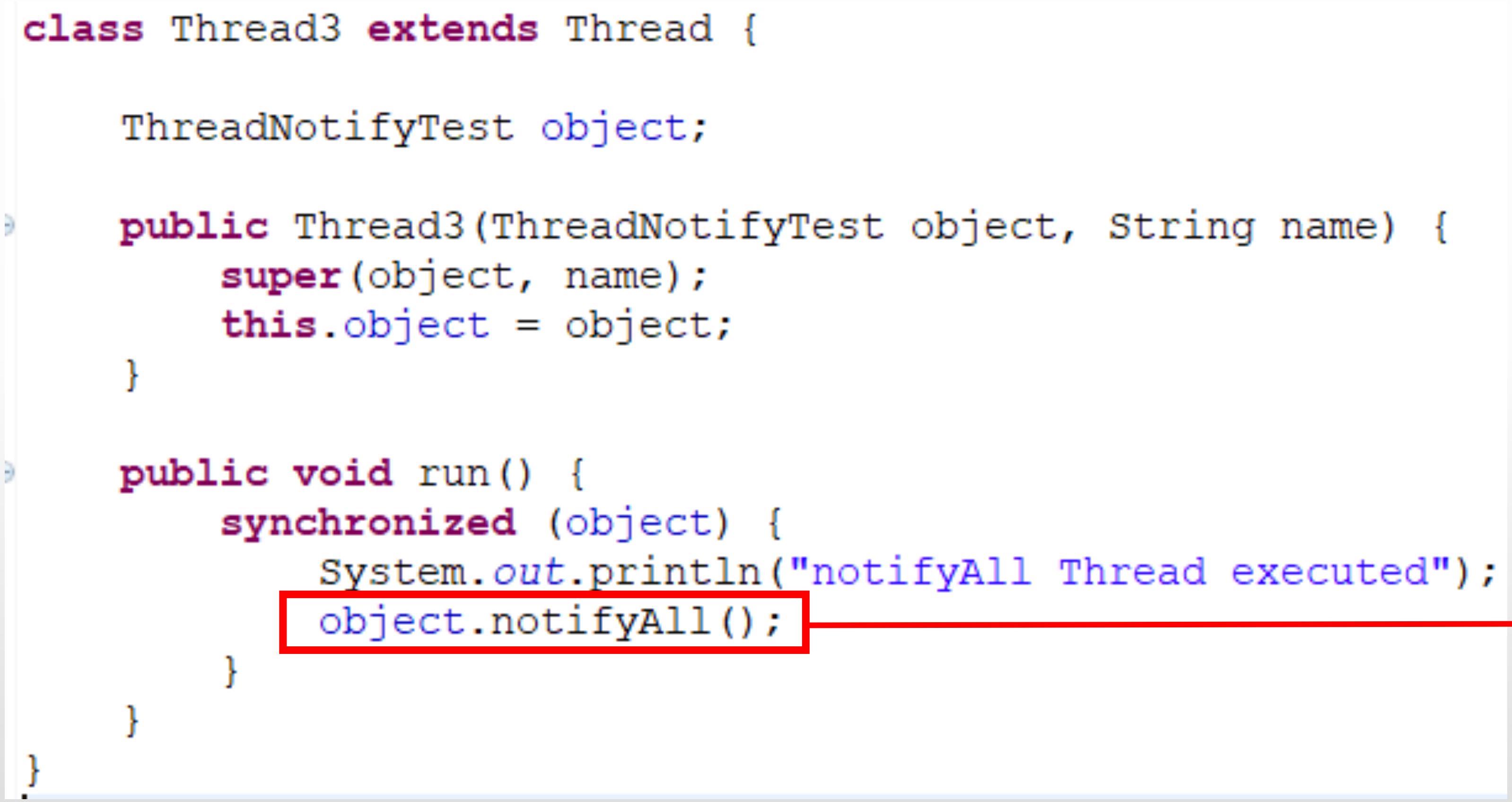
```
class Thread1 extends Thread {

    ThreadNotifyTest object;

    public Thread1(ThreadNotifyTest object, String name) {
        super(object, name);
        this.object = object;
    }

    public void run() {
        synchronized (object) {
            try {
                System.out.println("Started "
                        + Thread.currentThread().getName() + " wait");
                object.wait();
                System.out.println("Started "
                        + Thread.currentThread().getName() + " notified");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            for (int i = 0; i < 10; i++) {
                System.out.println(Thread.currentThread().getName() + " " + i);
            }
        }
    }
}
```

Example for notifyAll with multi-threaded scenario



```
class Thread2 extends Thread {

    ThreadNotifyTest object;

    public Thread2(ThreadNotifyTest object, String name) {
        super(object, name);
        this.object = object;
    }

    public void run() {
        synchronized (object) {
            try {
                System.out.println("Started "
                        + Thread.currentThread().getName() + " wait");
                object.wait();
                System.out.println("Started "
                        + Thread.currentThread().getName() + " notified");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            for (int i = 0; i < 10; i++) {
                System.out.println(Thread.currentThread().getName() + " " + i);
            }
        }
    }
}
```

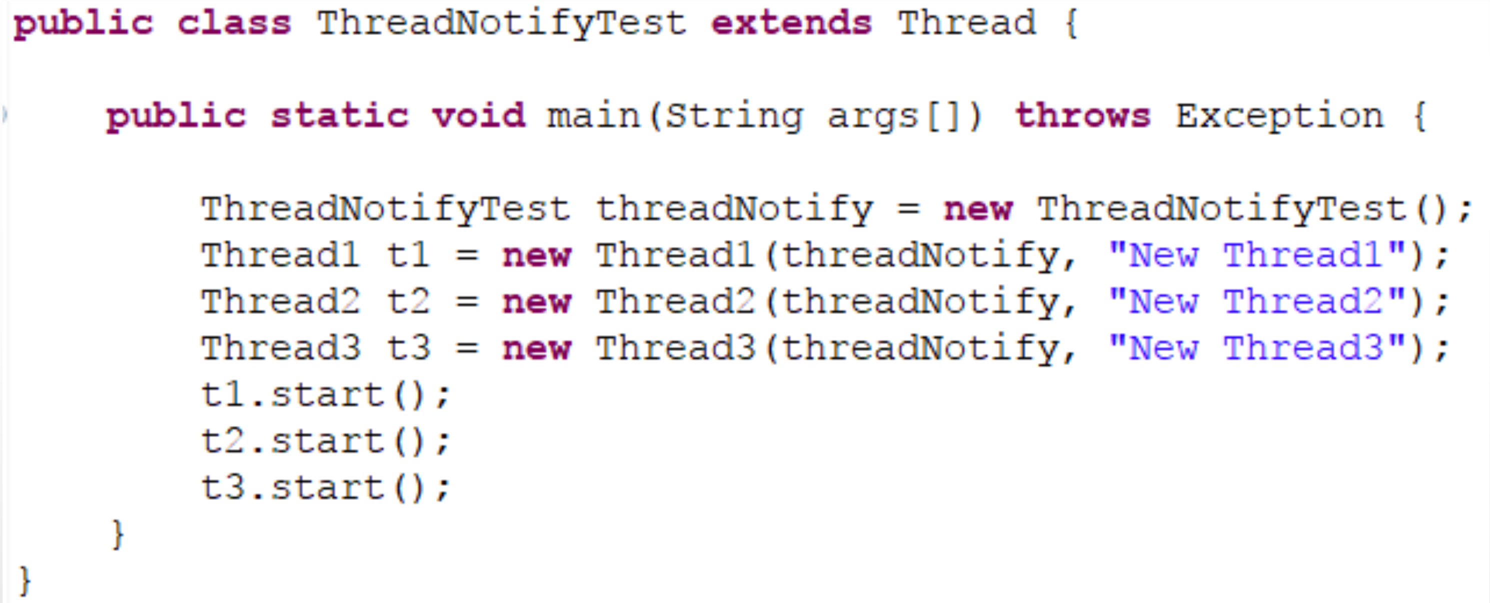
Thread notifyAll() method



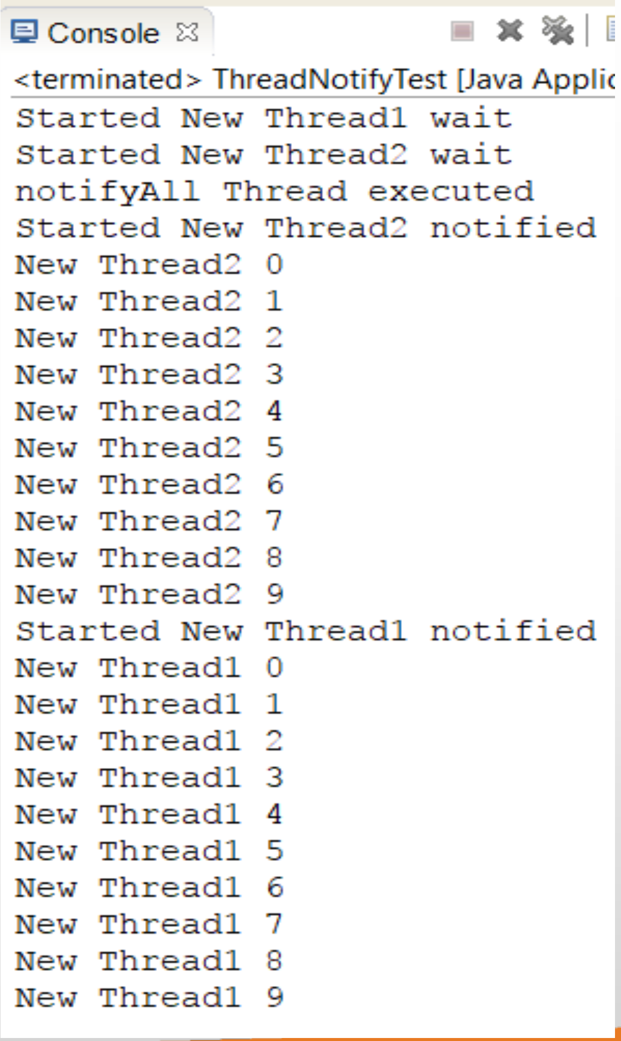
```
class Thread3 extends Thread {

    ThreadNotifyTest object;

    public Thread3(ThreadNotifyTest object, String name) {
        super(object, name);
        this.object = object;
    }

    public void run() {
        synchronized (object) {
            System.out.println("notifyAll Thread executed");
            object.notifyAll();
        }
    }
}
```

This method awake all threads which are waiting with the same objects

Output of notifyAll Scenario

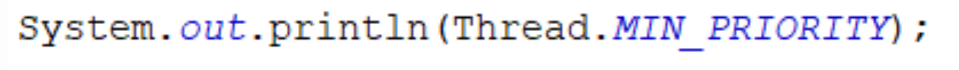


```
public class ThreadNotifyTest extends Thread {

    public static void main(String args[]) throws Exception {

        ThreadNotifyTest threadNotify = new ThreadNotifyTest();
        Thread1 t1 = new Thread1(threadNotify, "New Thread1");
        Thread2 t2 = new Thread2(threadNotify, "New Thread2");
        Thread3 t3 = new Thread3(threadNotify, "New Thread3");
        t1.start();
        t2.start();
        t3.start();
    }
}
```



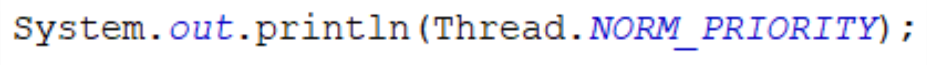
```
<terminated> ThreadNotifyTest [Java Applic
Started New Thread1 wait
Started New Thread2 wait
notifyAll Thread executed
Started New Thread2 notified
New Thread2 0
New Thread2 1
New Thread2 2
New Thread2 3
New Thread2 4
New Thread2 5
New Thread2 6
New Thread2 7
New Thread2 8
New Thread2 9
Started New Thread1 notified
New Thread1 0
New Thread1 1
New Thread1 2
New Thread1 3
New Thread1 4
New Thread1 5
New Thread1 6
New Thread1 7
New Thread1 8
New Thread1 9
```

Thread Priority



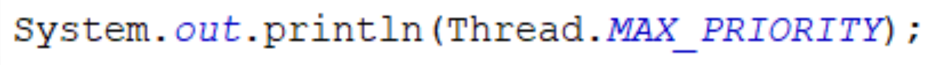
```
System.out.println(Thread.MIN_PRIORITY);
```

=> 1



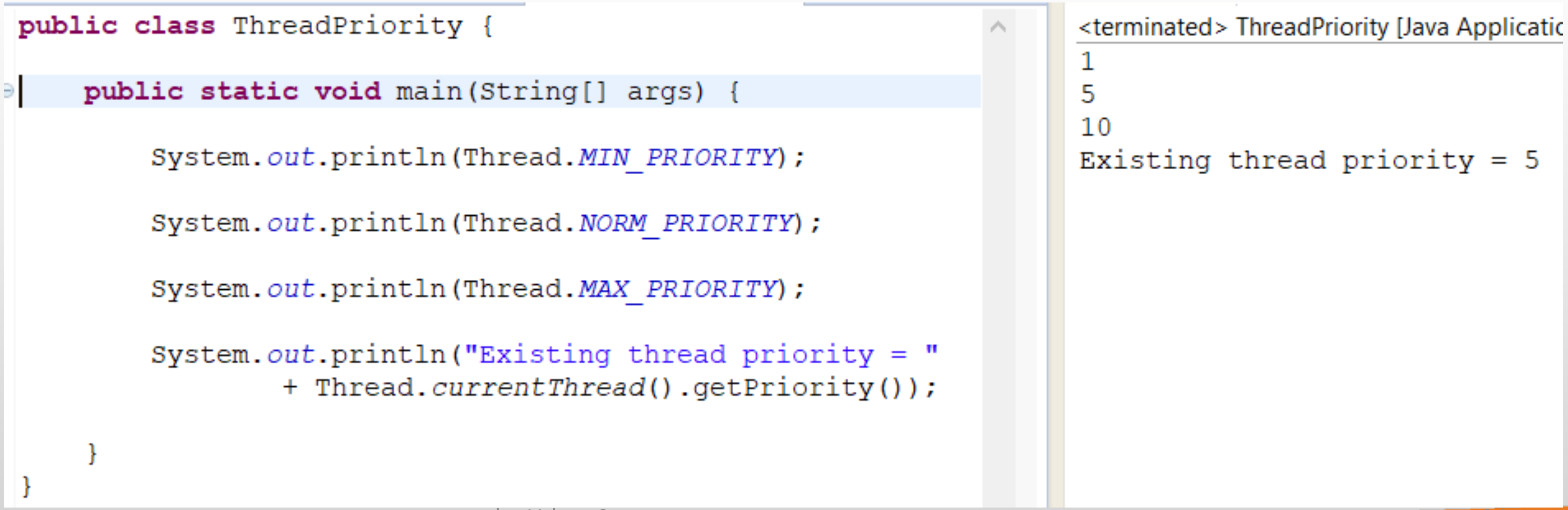
```
System.out.println(Thread.NORM_PRIORITY);
```

=> 5



```
System.out.println(Thread.MAX_PRIORITY);
```

=> 10



```
public class ThreadPriority {

    public static void main(String[] args) {

        System.out.println(Thread.MIN_PRIORITY);

        System.out.println(Thread.NORM_PRIORITY);

        System.out.println(Thread.MAX_PRIORITY);

        System.out.println("Existing thread priority = "
                + Thread.currentThread().getPriority());

    }
}
```

```
1
5
10
Existing thread priority = 5
```

Daemon Threads

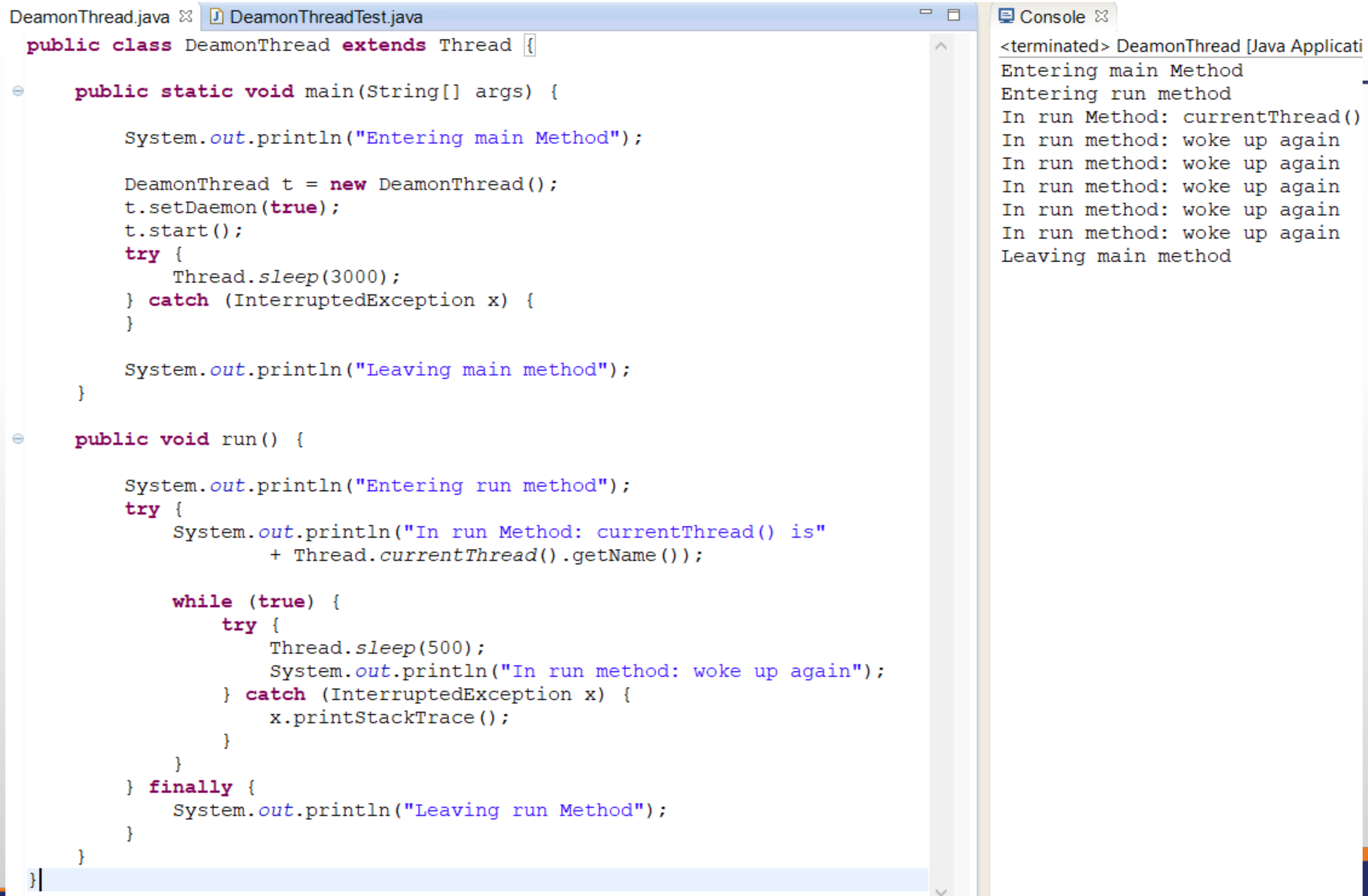
- Daemon Threads are “backgroung threads”.
  - That **provides service to other threads**, e.g. **The garbage collection thread.**
- The Java VM will not exit if non-daemon threads are executing
- The Java VM will exit if only Daemon threads are executing
- Daemon thread die when the Java VM exits.

Daemon Thread Example

- Since newly created thread is daemon thread when main thread completes its execution JavaVM will not wait until Daemon thread completes its execution.

- So it exit and Daemon thread automatically Die

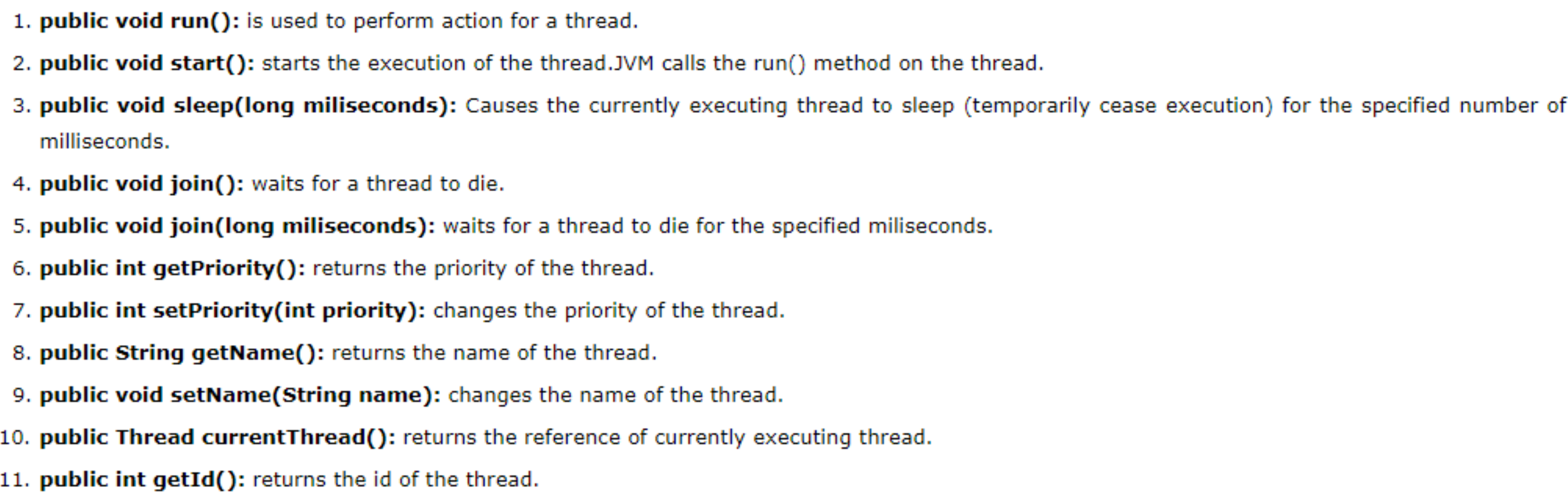
Daemon Thread Example



```java
public class DeamonThread extends Thread {

    public static void main(String[] args) {

        System.out.println("Entering main Method");

        DeamonThread t = new DeamonThread();
        t.setDaemon(true);
        t.start();
        try {
            Thread.sleep(3000);
        } catch (InterruptedException x) {
        }

        System.out.println("Leaving main method");
    }

    public void run() {

        System.out.println("Entering run method");
        try {
            System.out.println("In run Method: currentThread() is"
                    + Thread.currentThread().getName());

            while (true) {
                try {
                    Thread.sleep(500);
                    System.out.println("In run method: woke up again");
                } catch (InterruptedException x) {
                    x.printStackTrace();
                }
            }
        } finally {
            System.out.println("Leaving run Method");
        }
    }
}
```

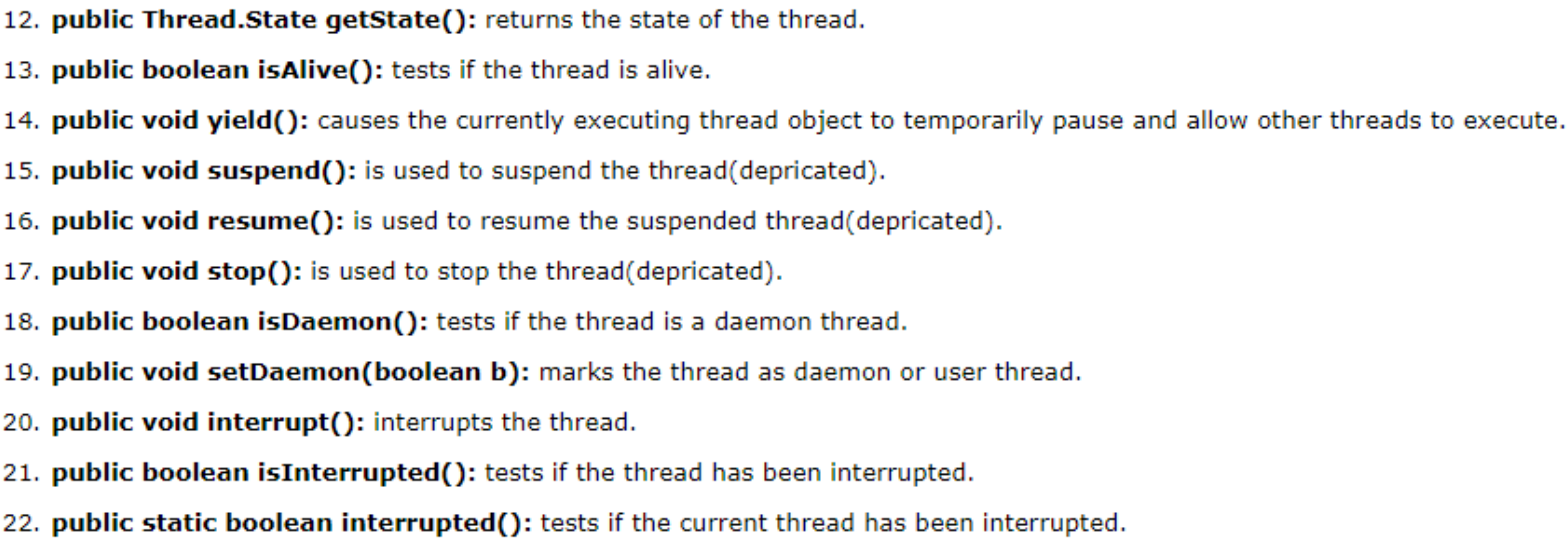
```
<terminated> DeamonThread [Java Applicati
Entering main Method
Entering run method
In run Method: currentThread()
In run method: woke up again
In run method: woke up again
In run method: woke up again
In run method: woke up again
In run method: woke up again
Leaving main method
```

Commonly used methods for Thread class



1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.

Commonly used methods for Thread class



12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

The End