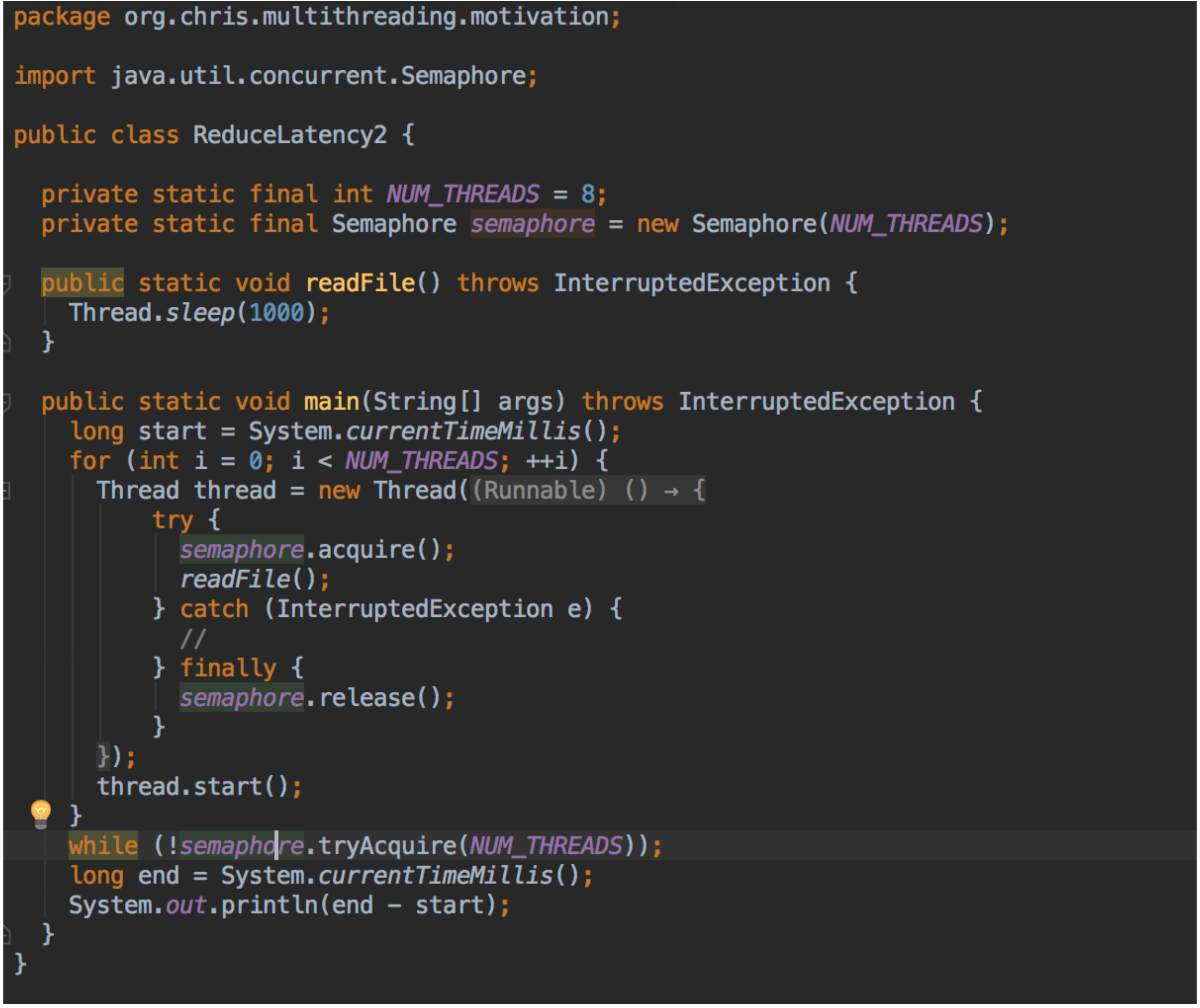
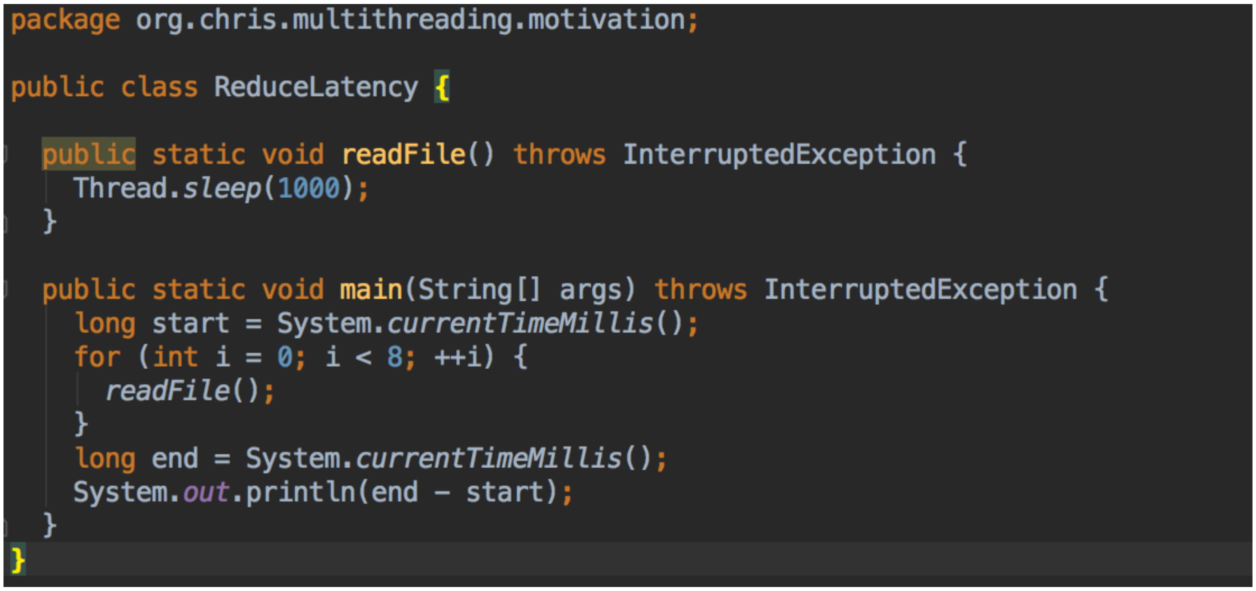
Motivation

Multithreading can be used to reduce latency. Reading 8 files from ~8s to ~1s.

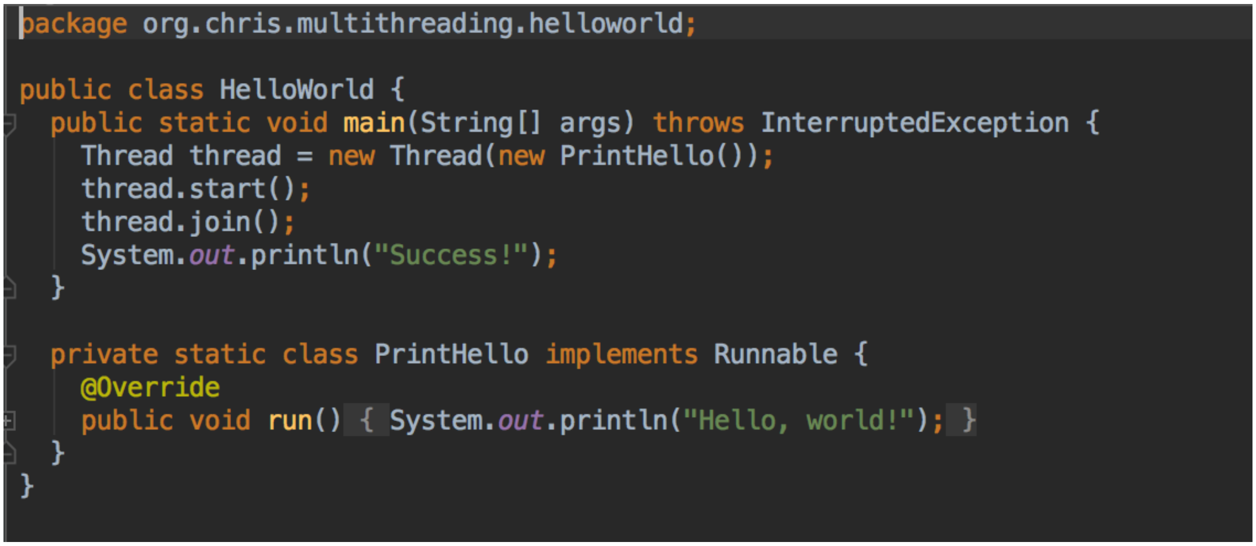


```java
package org.chris.multithreading.motivation;

import java.util.concurrent.Semaphore;

public class ReduceLatency2 {

  private static final int NUM_THREADS = 8;
  private static final Semaphore semaphore = new Semaphore(NUM_THREADS);

  public static void readFile() throws InterruptedException {
    Thread.sleep(1000);
  }

  public static void main(String[] args) throws InterruptedException {
    long start = System.currentTimeMillis();
    for (int i = 0; i < NUM_THREADS; ++i) {
      Thread thread = new Thread((Runnable) () → {
          try {
            semaphore.acquire();
            readFile();
          } catch (InterruptedException e) {
            //
          } finally {
            semaphore.release();
          }
      });
      thread.start();
    }
    while (!semaphore.tryAcquire(NUM_THREADS));
    long end = System.currentTimeMillis();
    System.out.println(end - start);
  }
}
```

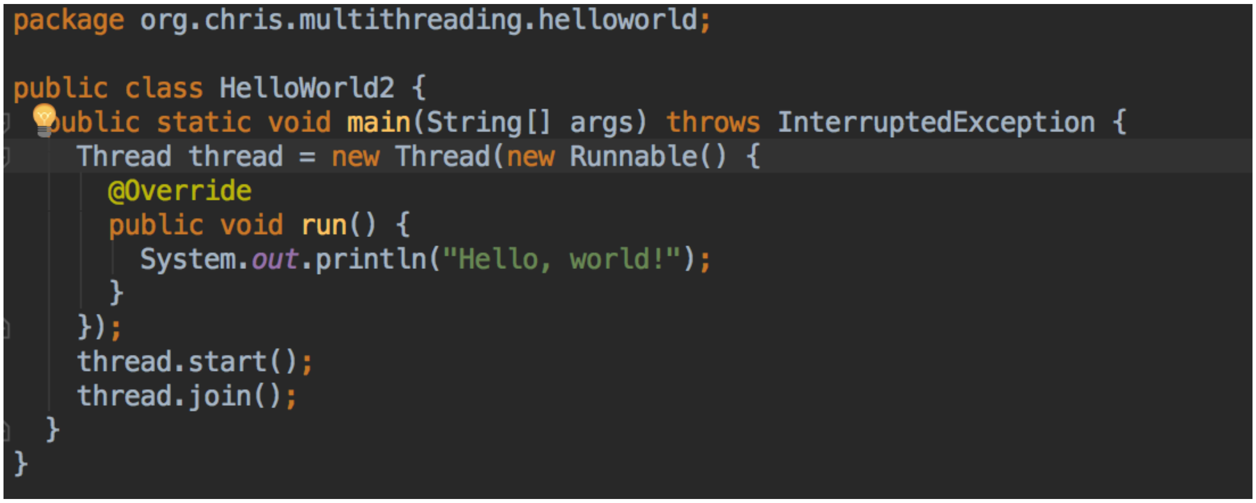


```java
package org.chris.multithreading.motivation;

public class ReduceLatency {

  public static void readFile() throws InterruptedException {
    Thread.sleep(1000);
  }

  public static void main(String[] args) throws InterruptedException {
    long start = System.currentTimeMillis();
    for (int i = 0; i < 8; ++i) {
      readFile();
    }
    long end = System.currentTimeMillis();
    System.out.println(end - start);
  }
}
```

Hello World



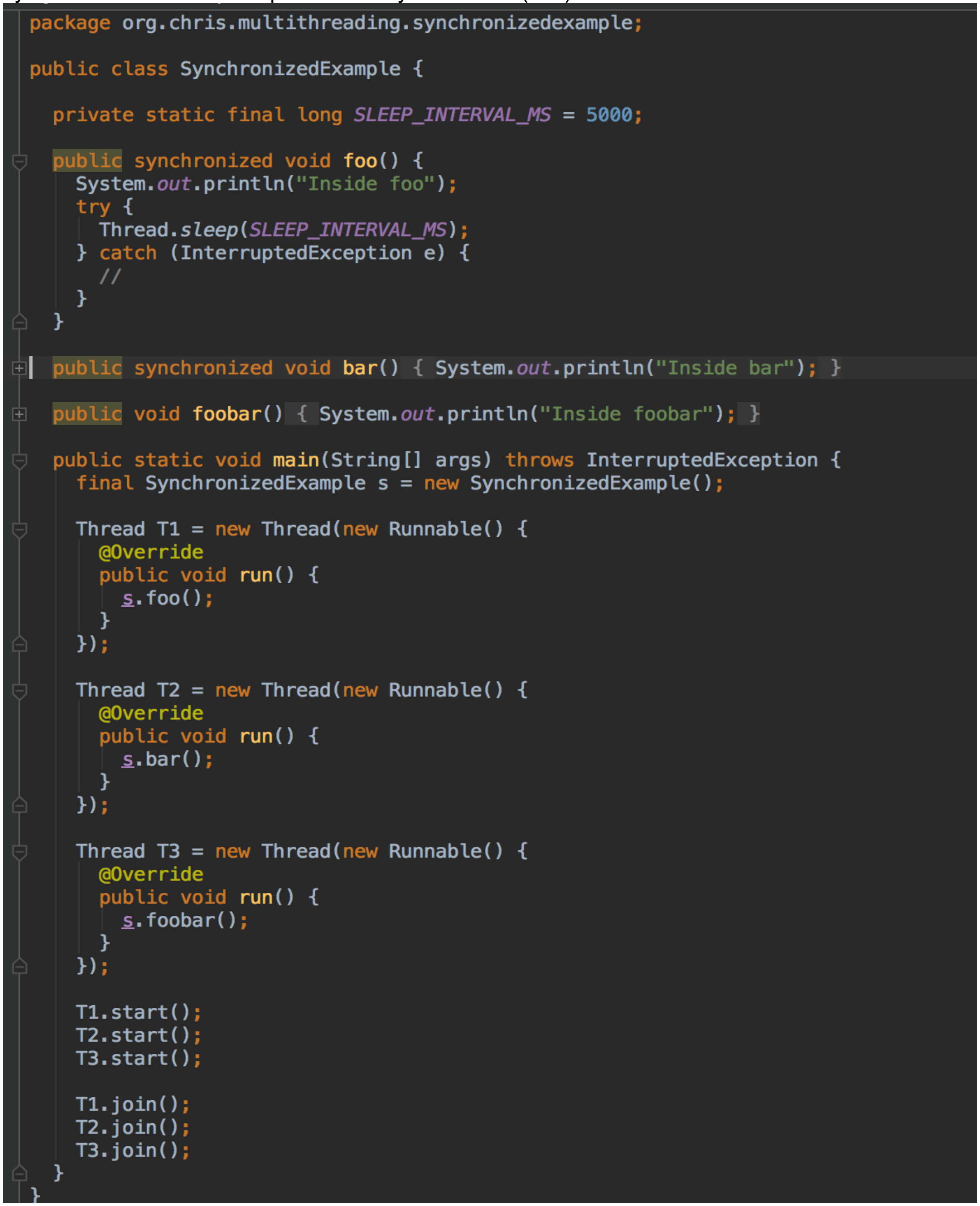
```java
package org.chris.multithreading.helloworld;

public class HelloWorld {
  public static void main(String[] args) throws InterruptedException {
    Thread thread = new Thread(new PrintHello());
    thread.start();
    thread.join();
    System.out.println("Success!");
  }

  private static class PrintHello implements Runnable {
    @Override
    public void run() { System.out.println("Hello, world!"); }
  }
}
```



```java
package org.chris.multithreading.helloworld;

public class HelloWorld2 {
  public static void main(String[] args) throws InterruptedException {
    Thread thread = new Thread(new Runnable() {
      @Override
      public void run() {
        System.out.println("Hello, world!");
      }
    });
    thread.start();
    thread.join();
  }
}
```

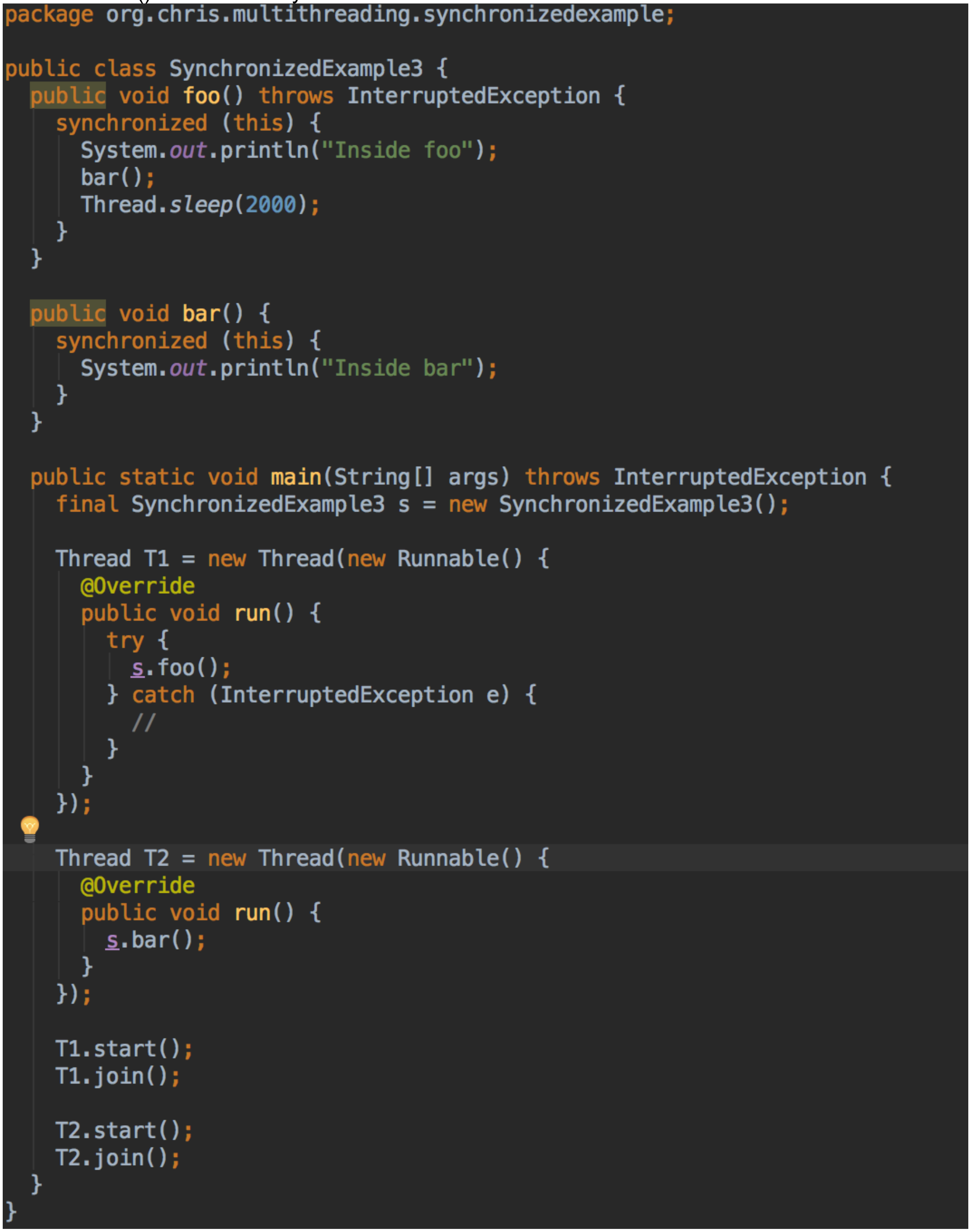
Synchronized

synchronized method is equivalent to synchronized(this).



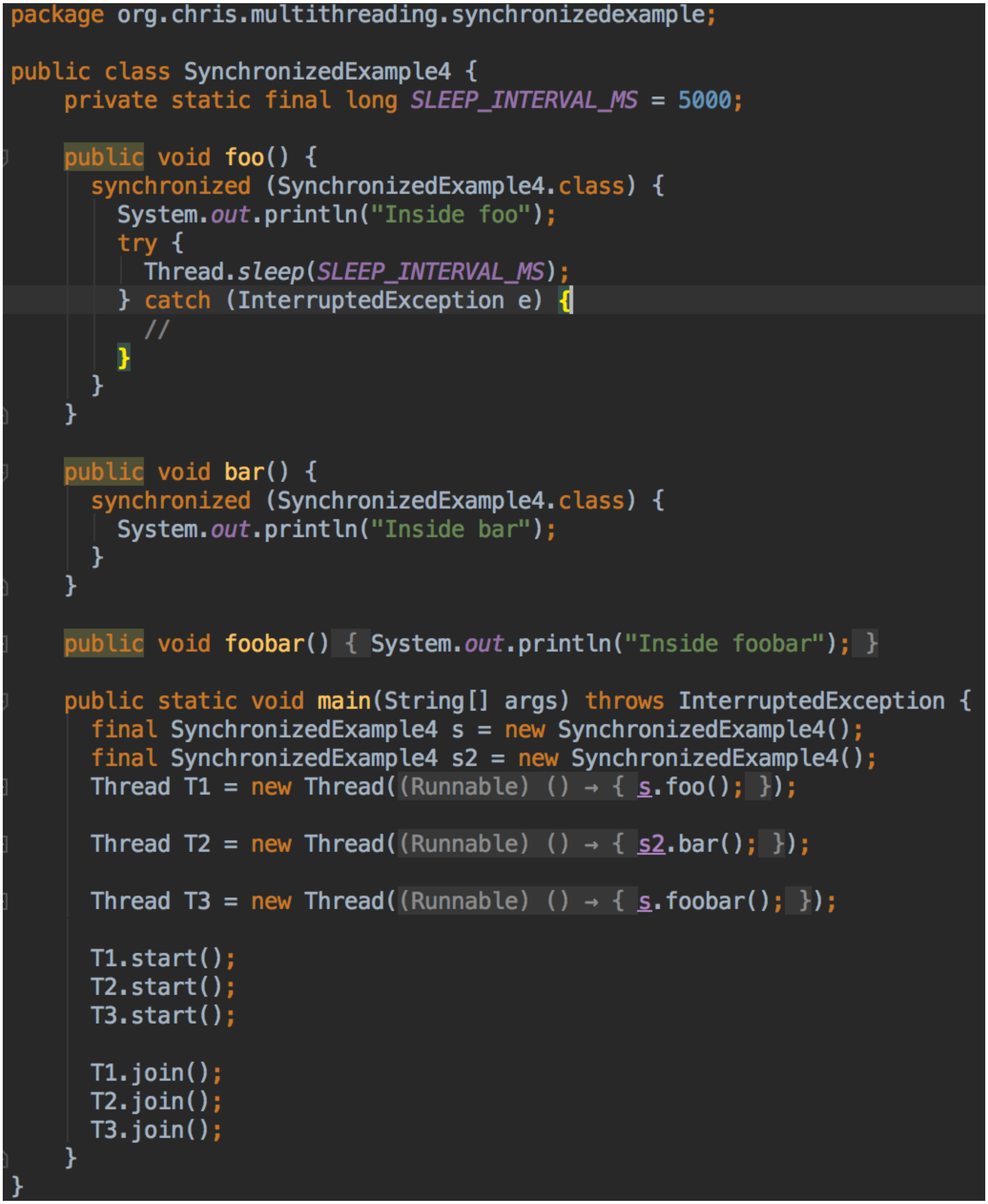
```java
package org.chris.multithreading.synchronizedexample;

public class SynchronizedExample {

  private static final long SLEEP_INTERVAL_MS = 5000;

  public synchronized void foo() {
    System.out.println("Inside foo");
    try {
      Thread.sleep(SLEEP_INTERVAL_MS);
    } catch (InterruptedException e) {
      //
    }
  }

  public synchronized void bar() { System.out.println("Inside bar"); }

  public void foobar() { System.out.println("Inside foobar"); }

  public static void main(String[] args) throws InterruptedException {
    final SynchronizedExample s = new SynchronizedExample();

    Thread T1 = new Thread(new Runnable() {
      @Override
      public void run() {
        s.foo();
      }
    });

    Thread T2 = new Thread(new Runnable() {
      @Override
      public void run() {
        s.bar();
      }
    });

    Thread T3 = new Thread(new Runnable() {
      @Override
      public void run() {
        s.foobar();
      }
    });

    T1.start();
    T2.start();
    T3.start();

    T1.join();
    T2.join();
    T3.join();
  }
}
```

Synchronized is reentrant to the same thread. Here foo() calls bar() which is also protected by synchronized(this). A thread T that enters foo can enter bar as well. Other threads will be blocked if foo() is executed by T.

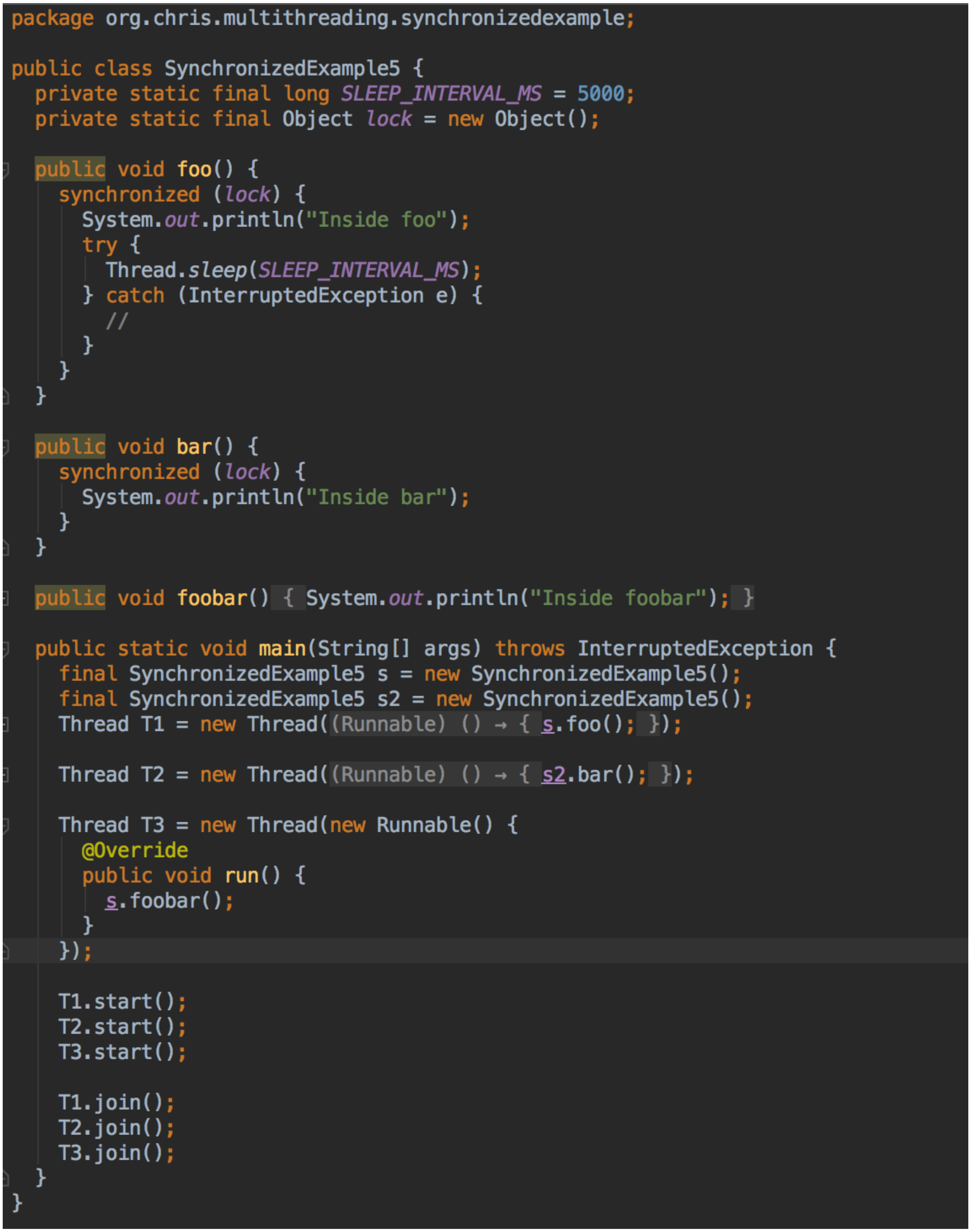


```java
package org.chris.multithreading.synchronizedexample;

public class SynchronizedExample3 {
  public void foo() throws InterruptedException {
    synchronized (this) {
      System.out.println("Inside foo");
      bar();
      Thread.sleep(2000);
    }
  }

  public void bar() {
    synchronized (this) {
      System.out.println("Inside bar");
    }
  }

  public static void main(String[] args) throws InterruptedException {
    final SynchronizedExample3 s = new SynchronizedExample3();

    Thread T1 = new Thread(new Runnable() {
      @Override
      public void run() {
        try {
          s.foo();
        } catch (InterruptedException e) {
          //
        }
      }
    });

    Thread T2 = new Thread(new Runnable() {
      @Override
      public void run() {
        s.bar();
      }
    });

    T1.start();
    T1.join();

    T2.start();
    T2.join();
  }
}
```

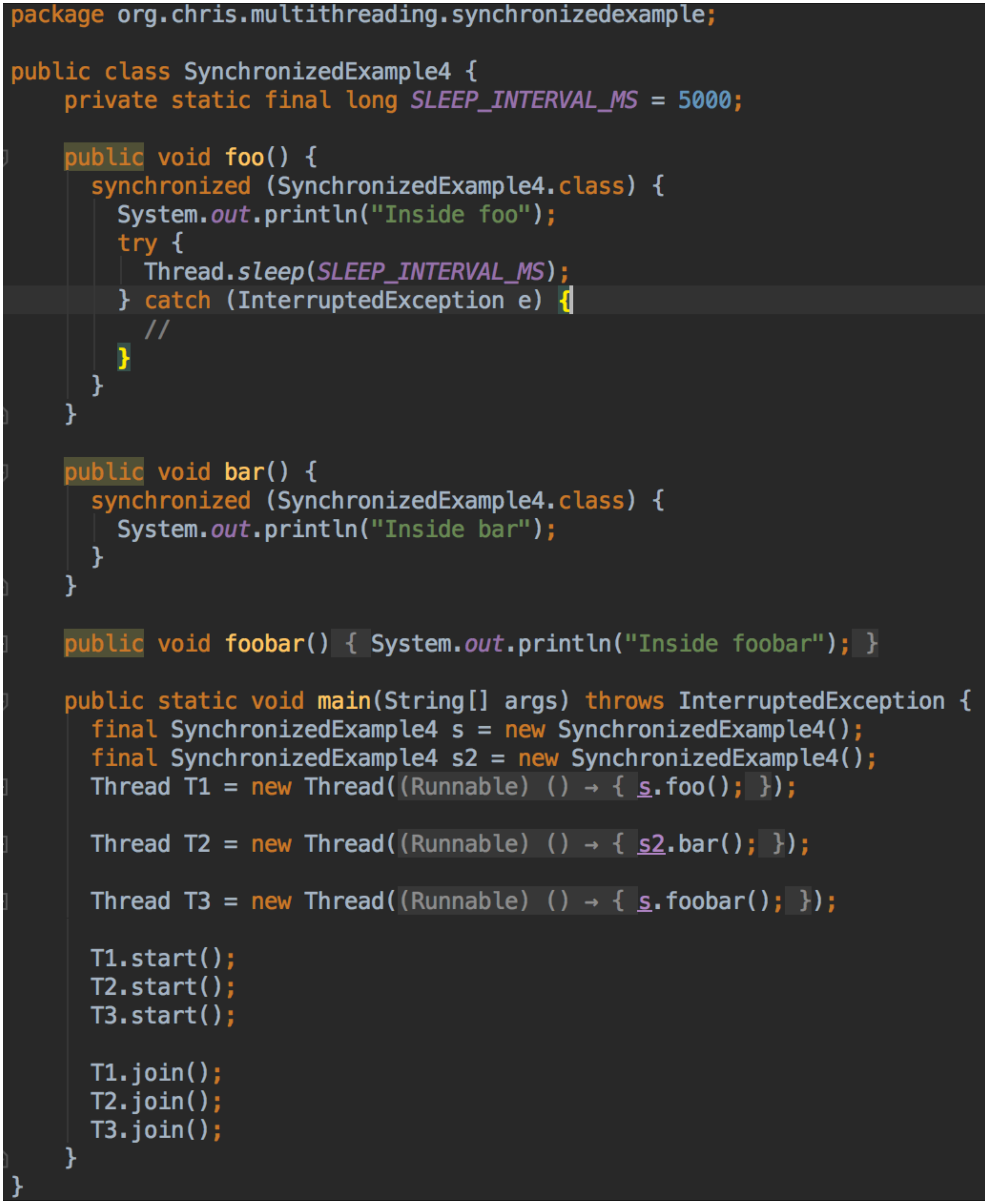
Synchronized on a method for all instances of a class.



```java
package org.chris.multithreading.synchronizedexample;

public class SynchronizedExample4 {
    private static final long SLEEP_INTERVAL_MS = 5000;

    public void foo() {
      synchronized (SynchronizedExample4.class) {
        System.out.println("Inside foo");
        try {
          Thread.sleep(SLEEP_INTERVAL_MS);
        } catch (InterruptedException e) {
          //
        }
      }
    }

    public void bar() {
      synchronized (SynchronizedExample4.class) {
        System.out.println("Inside bar");
      }
    }

    public void foobar() { System.out.println("Inside foobar"); }

    public static void main(String[] args) throws InterruptedException {
      final SynchronizedExample4 s = new SynchronizedExample4();
      final SynchronizedExample4 s2 = new SynchronizedExample4();
      Thread T1 = new Thread((Runnable) () → { s.foo(); });

      Thread T2 = new Thread((Runnable) () → { s2.bar(); });

      Thread T3 = new Thread((Runnable) () → { s.foobar(); });

      T1.start();
      T2.start();
      T3.start();

      T1.join();
      T2.join();
      T3.join();
    }
}
```



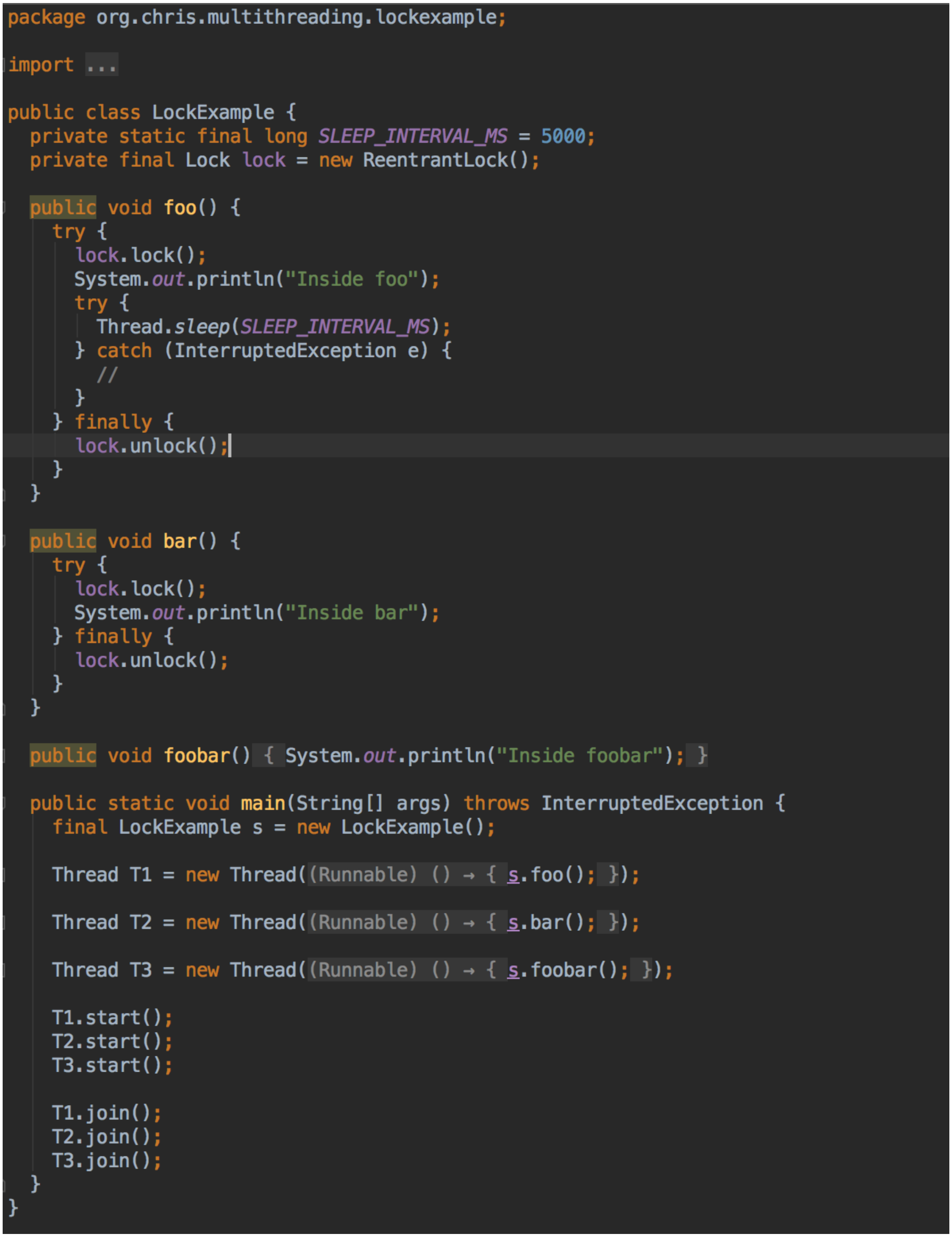
```java
package org.chris.multithreading.synchronizedexample;

public class SynchronizedExample5 {
  private static final long SLEEP_INTERVAL_MS = 5000;
  private static final Object lock = new Object();

  public void foo() {
    synchronized (lock) {
      System.out.println("Inside foo");
      try {
        Thread.sleep(SLEEP_INTERVAL_MS);
      } catch (InterruptedException e) {
        //
      }
    }
  }

  public void bar() {
    synchronized (lock) {
      System.out.println("Inside bar");
    }
  }

  public void foobar() { System.out.println("Inside foobar"); }

  public static void main(String[] args) throws InterruptedException {
    final SynchronizedExample5 s = new SynchronizedExample5();
    final SynchronizedExample5 s2 = new SynchronizedExample5();
    Thread T1 = new Thread((Runnable) () → { s.foo(); });

    Thread T2 = new Thread((Runnable) () → { s2.bar(); });

    Thread T3 = new Thread(new Runnable() {
      @Override
      public void run() {
        s.foobar();
      }
    });

    T1.start();
    T2.start();
    T3.start();

    T1.join();
    T2.join();
    T3.join();
  }
}
```



```
package org.chris.multithreading.synchronizedexample;

public class SynchronizedExample4 {
    private static final long SLEEP_INTERVAL_MS = 5000;

    public void foo() {
      synchronized (SynchronizedExample4.class) {
        System.out.println("Inside foo");
        try {
          Thread.sleep(SLEEP_INTERVAL_MS);
        } catch (InterruptedException e) {
          //
        }
      }
    }

    public void bar() {
      synchronized (SynchronizedExample4.class) {
        System.out.println("Inside bar");
      }
    }

    public void foobar() { System.out.println("Inside foobar"); }

    public static void main(String[] args) throws InterruptedException {
      final SynchronizedExample4 s = new SynchronizedExample4();
      final SynchronizedExample4 s2 = new SynchronizedExample4();
      Thread T1 = new Thread((Runnable) () → { s.foo(); });

      Thread T2 = new Thread((Runnable) () → { s2.bar(); });

      Thread T3 = new Thread((Runnable) () → { s.foobar(); });

      T1.start();
      T2.start();
      T3.start();

      T1.join();
      T2.join();
      T3.join();
    }
}
```

Reentrant Lock

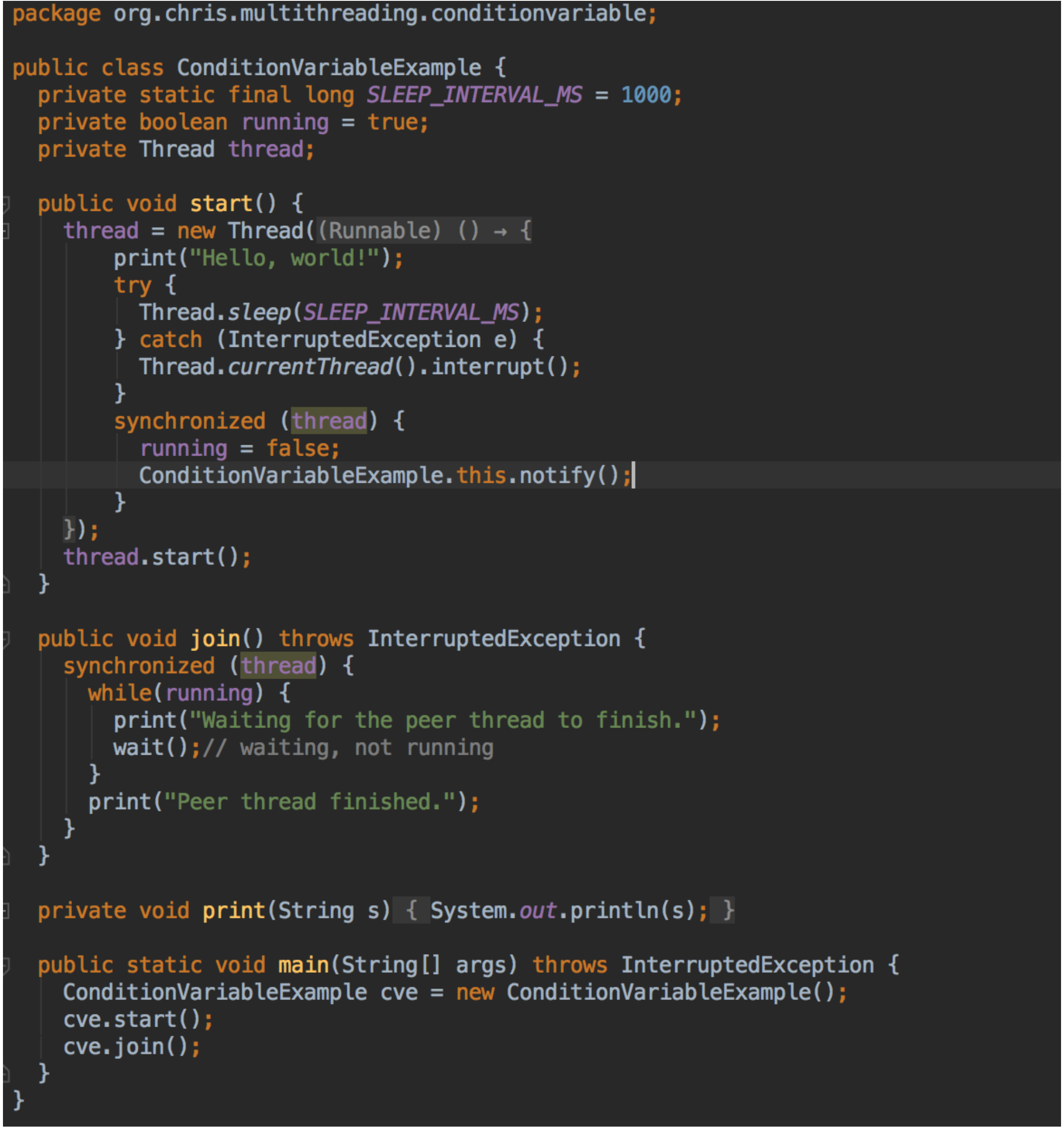
Reentrant lock is similar to synchronized keyword. It is recommended that the lock is released in the finally block.



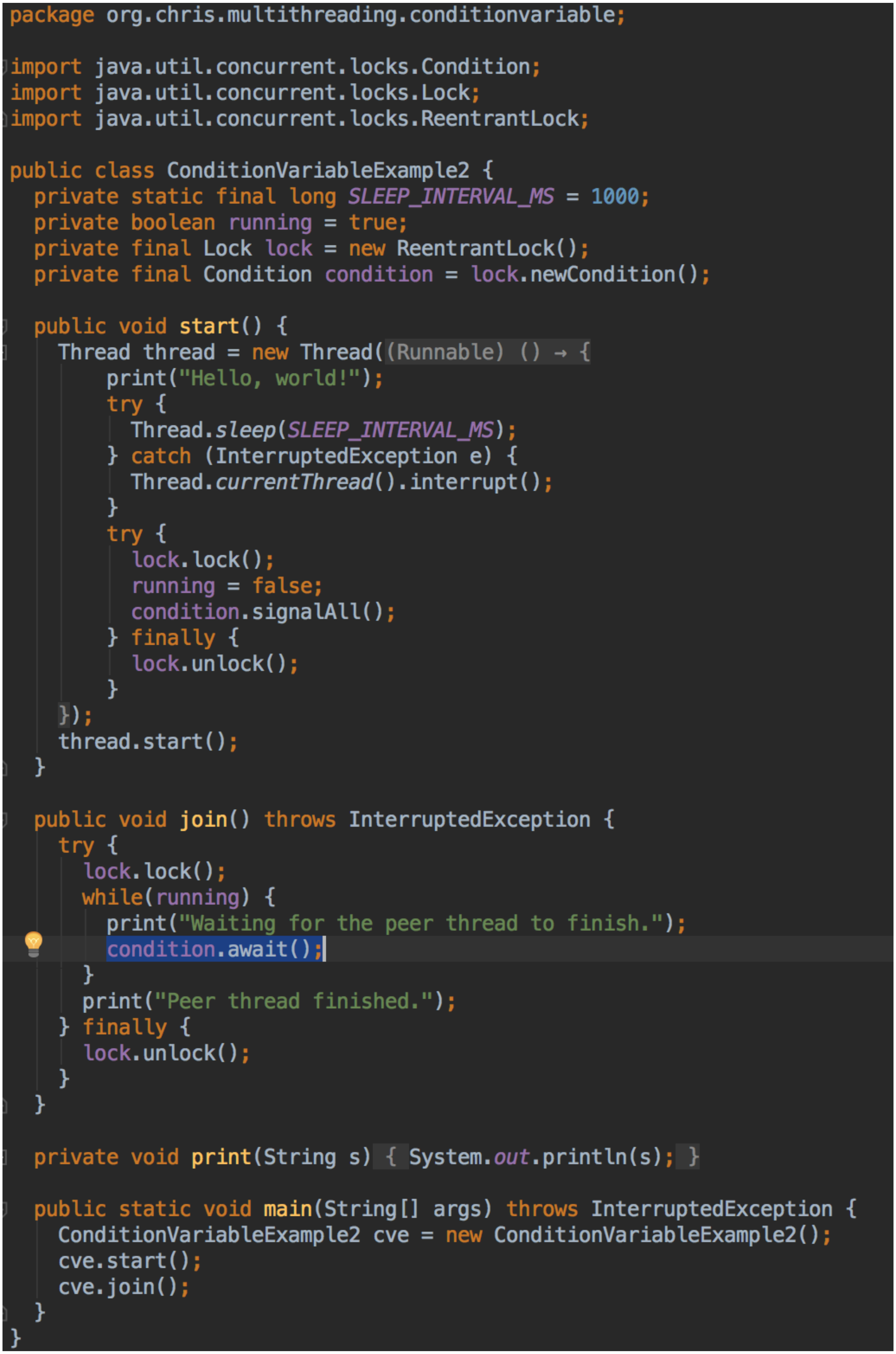
```java
package org.chris.multithreading.lockexample;

import ...

public class LockExample {
  private static final long SLEEP_INTERVAL_MS = 5000;
  private final Lock lock = new ReentrantLock();

  public void foo() {
    try {
      lock.lock();
      System.out.println("Inside foo");
      try {
        Thread.sleep(SLEEP_INTERVAL_MS);
      } catch (InterruptedException e) {
        //
      }
    } finally {
      lock.unlock();
    }
  }

  public void bar() {
    try {
      lock.lock();
      System.out.println("Inside bar");
    } finally {
      lock.unlock();
    }
  }

  public void foobar() { System.out.println("Inside foobar"); }

  public static void main(String[] args) throws InterruptedException {
    final LockExample s = new LockExample();

    Thread T1 = new Thread((Runnable) () → { s.foo(); });

    Thread T2 = new Thread((Runnable) () → { s.bar(); });

    Thread T3 = new Thread((Runnable) () → { s.foobar(); });

    T1.start();
    T2.start();
    T3.start();

    T1.join();
    T2.join();
    T3.join();
  }
}
```

Condition Variable

This code uses condition variable to implement the join method.



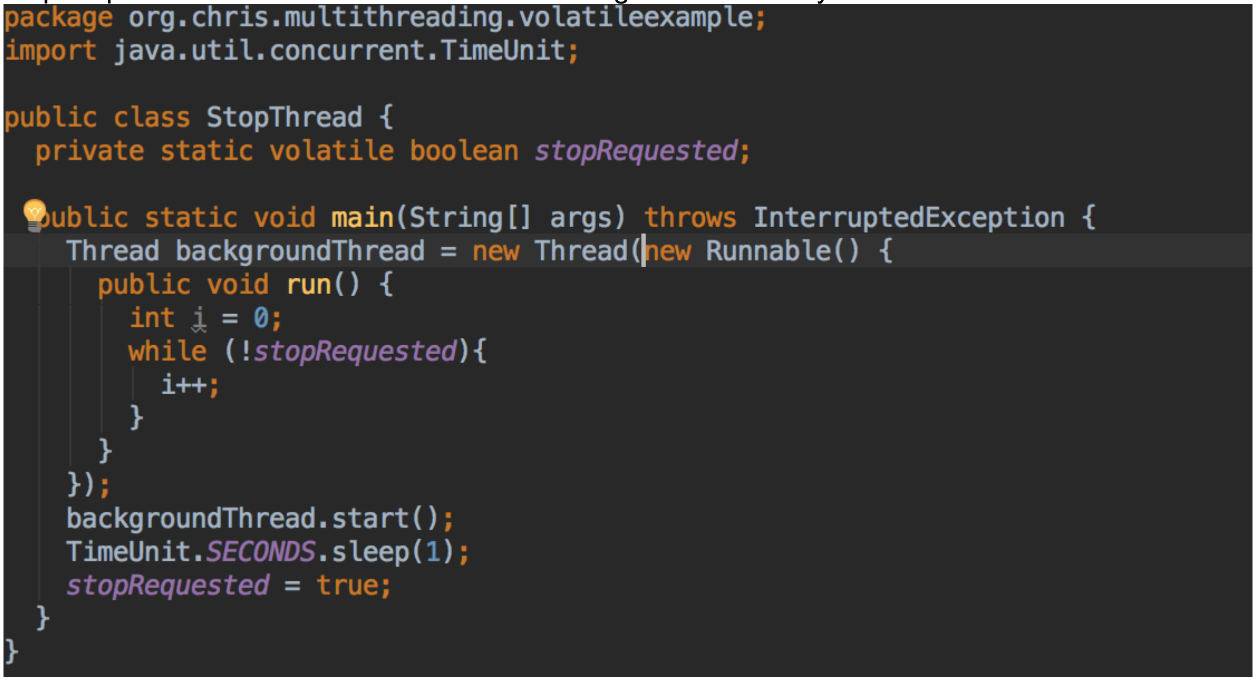
```
package org.chris.multithreading.conditionvariable;

public class ConditionVariableExample {
  private static final long SLEEP_INTERVAL_MS = 1000;
  private boolean running = true;
  private Thread thread;

  public void start() {
    thread = new Thread((Runnable) () → {
        print("Hello, world!");
        try {
          Thread.sleep(SLEEP_INTERVAL_MS);
        } catch (InterruptedException e) {
          Thread.currentThread().interrupt();
        }
        synchronized (thread) {
          running = false;
          ConditionVariableExample.this.notify();
        }
    });
    thread.start();
  }

  public void join() throws InterruptedException {
    synchronized (thread) {
      while(running) {
        print("Waiting for the peer thread to finish.");
        wait();// waiting, not running
      }
      print("Peer thread finished.");
    }
  }

  private void print(String s) { System.out.println(s); }

  public static void main(String[] args) throws InterruptedException {
    ConditionVariableExample cve = new ConditionVariableExample();
    cve.start();
    cve.join();
  }
}
```



```java
package org.chris.multithreading.conditionvariable;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class ConditionVariableExample2 {
  private static final long SLEEP_INTERVAL_MS = 1000;
  private boolean running = true;
  private final Lock lock = new ReentrantLock();
  private final Condition condition = lock.newCondition();

  public void start() {
    Thread thread = new Thread((Runnable) () → {
        print("Hello, world!");
        try {
          Thread.sleep(SLEEP_INTERVAL_MS);
        } catch (InterruptedException e) {
          Thread.currentThread().interrupt();
        }
        try {
          lock.lock();
          running = false;
          condition.signalAll();
        } finally {
          lock.unlock();
        }
    });
    thread.start();
  }

  public void join() throws InterruptedException {
    try {
      lock.lock();
      while(running) {
        print("Waiting for the peer thread to finish.");
        condition.await();
      }
      print("Peer thread finished.");
    } finally {
      lock.unlock();
    }
  }

  private void print(String s) { System.out.println(s); }

  public static void main(String[] args) throws InterruptedException {
    ConditionVariableExample2 cve = new ConditionVariableExample2();
    cve.start();
    cve.join();
  }
}
```

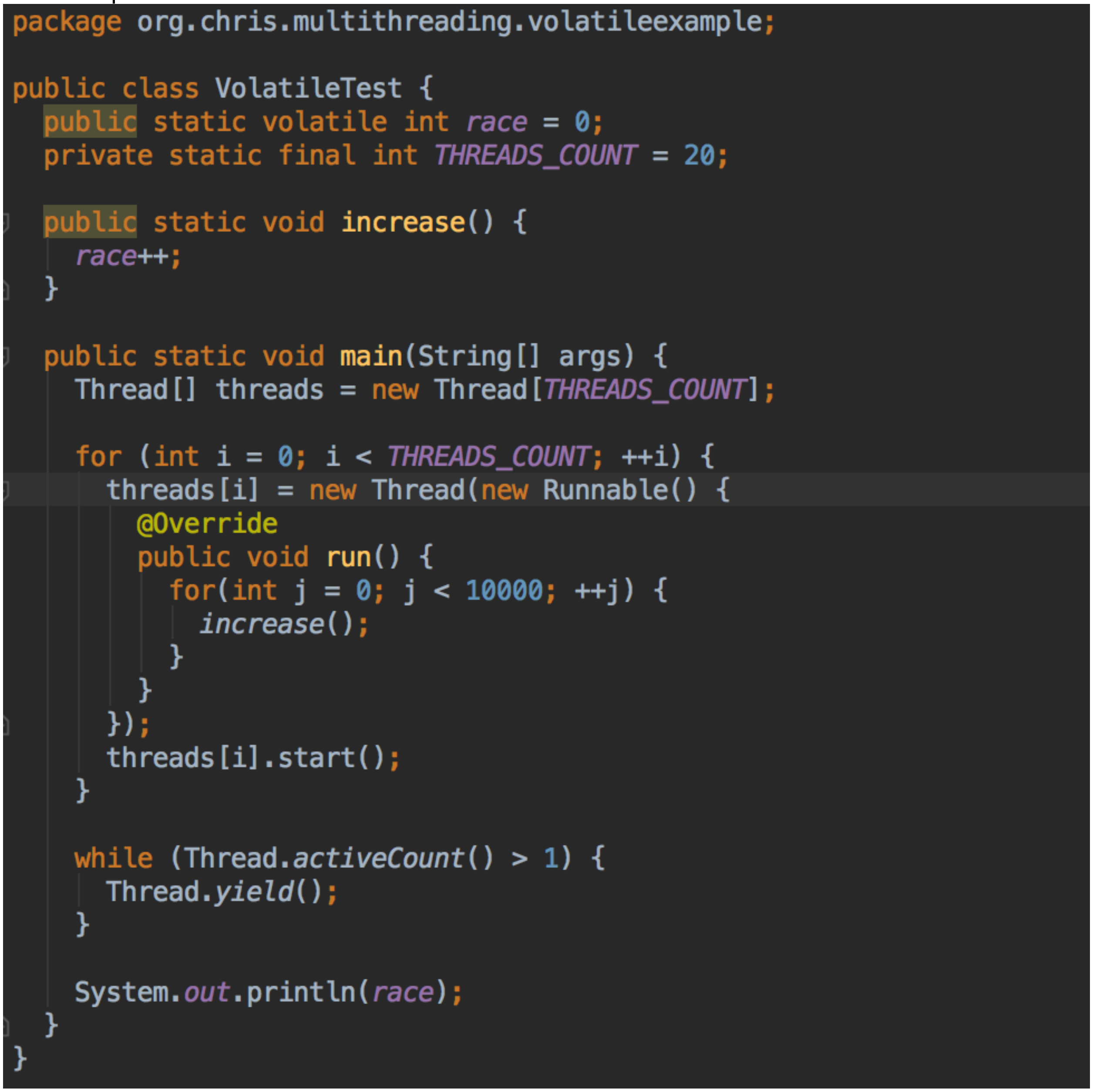
Volatile Variable

stopRequested needs to be volatile for its change to be seen by the other thread.

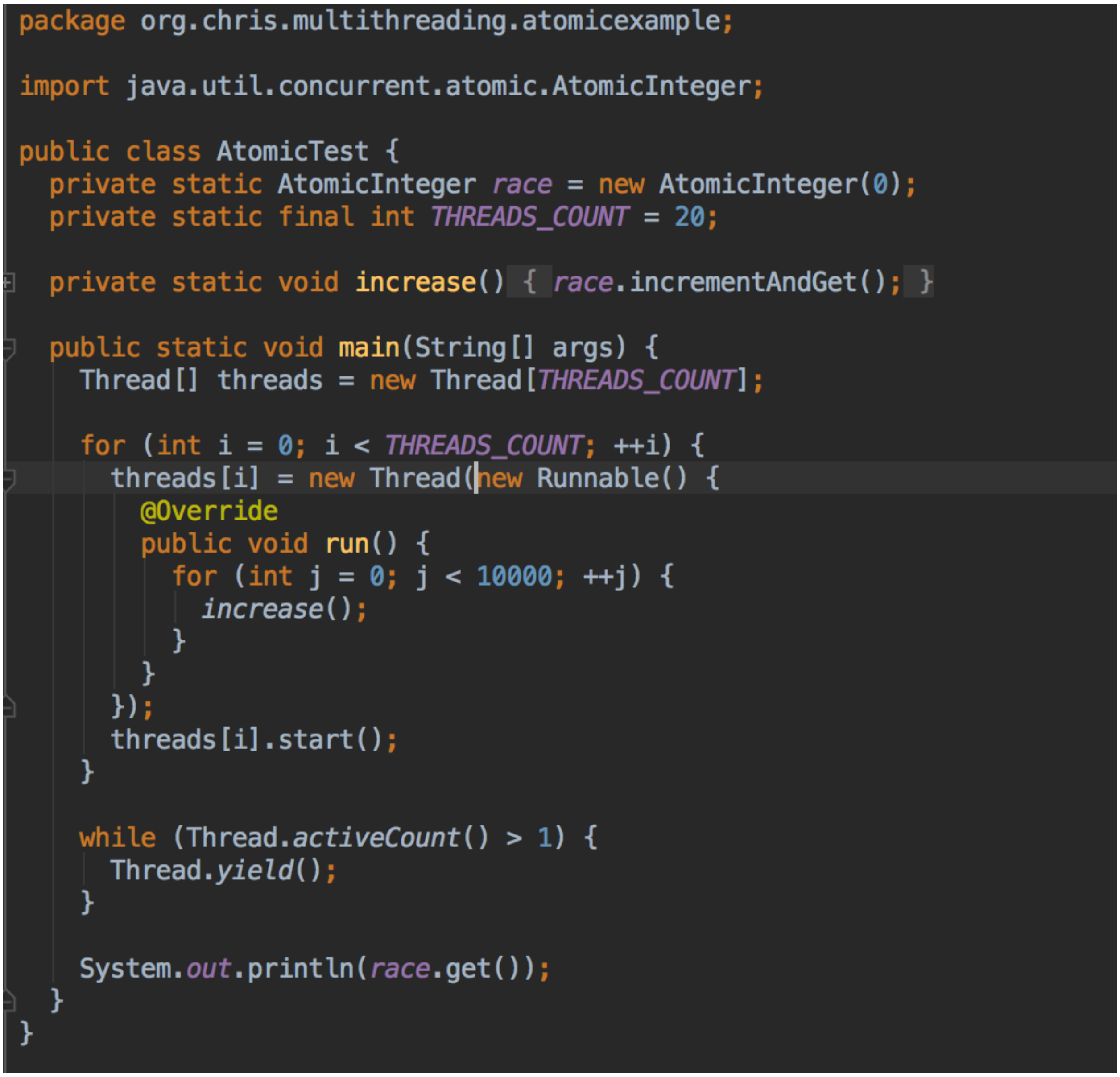


```java
package org.chris.multithreading.volatileexample;
import java.util.concurrent.TimeUnit;

public class StopThread {
  private static volatile boolean stopRequested;

  public static void main(String[] args) throws InterruptedException {
    Thread backgroundThread = new Thread(new Runnable() {
      public void run() {
        int i = 0;
        while (!stopRequested){
          i++;
        }
      }
    });
    backgroundThread.start();
    TimeUnit.SECONDS.sleep(1);
    stopRequested = true;
  }
}
```

race needs to be an atomic variable. Volatile can only guarantee that when reading race, the value is up to date.



```
package org.chris.multithreading.volatileexample;

public class VolatileTest {
  public static volatile int race = 0;
  private static final int THREADS_COUNT = 20;

  public static void increase() {
    race++;
  }

  public static void main(String[] args) {
    Thread[] threads = new Thread[THREADS_COUNT];

    for (int i = 0; i < THREADS_COUNT; ++i) {
      threads[i] = new Thread(new Runnable() {
        @Override
        public void run() {
          for(int j = 0; j < 10000; ++j) {
            increase();
          }
        }
      });
      threads[i].start();
    }

    while (Thread.activeCount() > 1) {
      Thread.yield();
    }

    System.out.println(race);
  }
}
```



```java
package org.chris.multithreading.atomicexample;

import java.util.concurrent.atomic.AtomicInteger;

public class AtomicTest {
  private static AtomicInteger race = new AtomicInteger(0);
  private static final int THREADS_COUNT = 20;

  private static void increase() { race.incrementAndGet(); }

  public static void main(String[] args) {
    Thread[] threads = new Thread[THREADS_COUNT];

    for (int i = 0; i < THREADS_COUNT; ++i) {
      threads[i] = new Thread(new Runnable() {
        @Override
        public void run() {
          for (int j = 0; j < 10000; ++j) {
            increase();
          }
        }
      });
      threads[i].start();
    }

    while (Thread.activeCount() > 1) {
      Thread.yield();
    }

    System.out.println(race.get());
  }
}
```