Java 核心技术(进阶)

第五章 Java多线程和并发编程

第四节 Java多线程管理

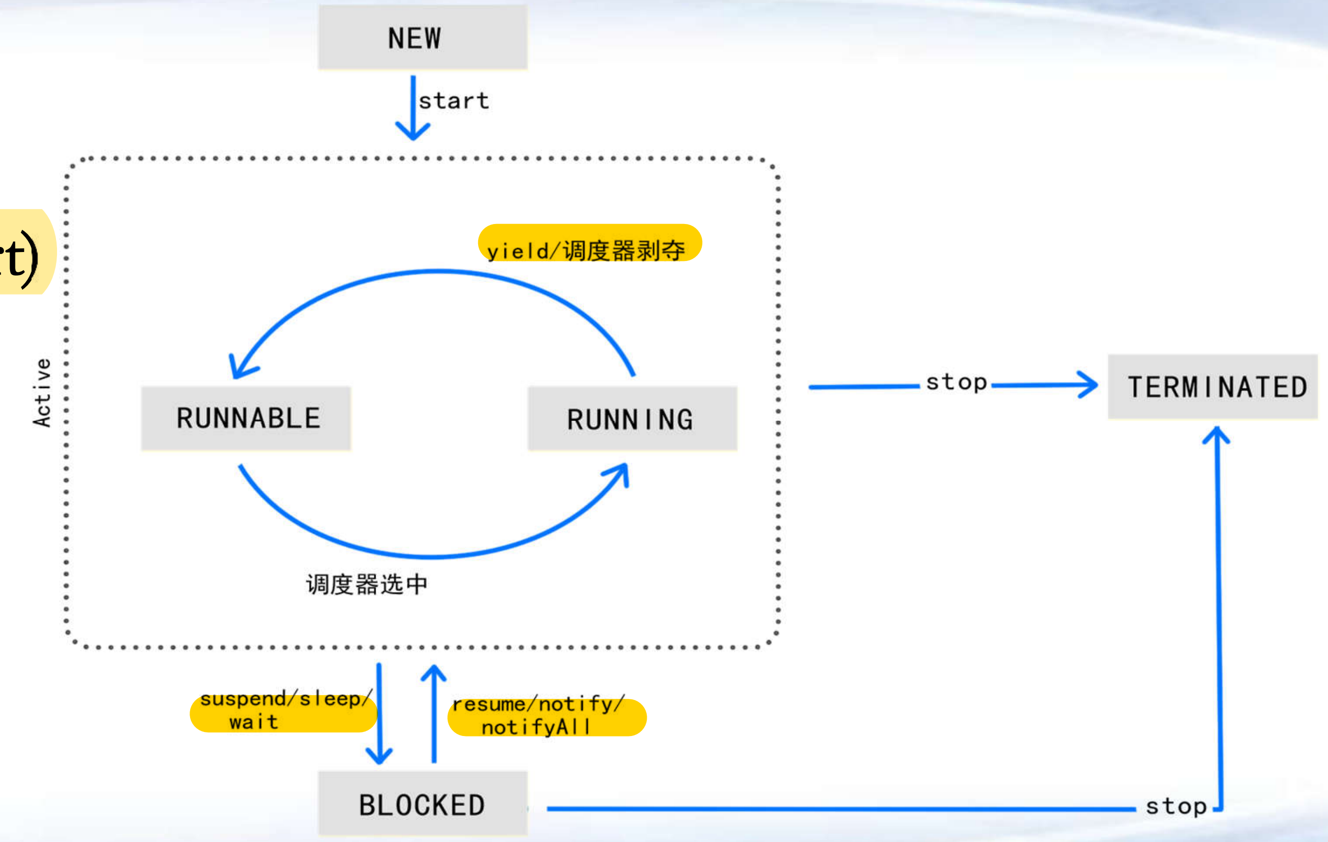
华东师范大学 陈良育

多线程管理(1)

- 线程类
  - 通过继承Thread或实现Runnable
  - 通过start方法，调用run方法，run方法工作
  - 线程run结束后，线程退出
- 粗粒度：子线程与子线程之间、和main线程之间缺乏同步
- 细粒度：线程之间有同步协作
  - 等待
  - 通知/唤醒
  - 终止

多线程管理(2)

- 线程状态
  - NEW 刚创建(new)
  - RUNNABLE 就绪态(start)
  - RUNNING 运行中(run)
  - BLOCK 阻塞(sleep)
  - TERMINATED 结束



多线程管理(3)

- Thread的部分API已经废弃
  - 暂停和恢复 suspend/resume
  - 消亡 stop/destroy
- 线程阻塞/和唤醒
  - sleep，时间一到，自己会醒来
  - wait/notify/notifyAll，等待，需要别人来唤醒
  - join，等待另外一个线程结束
  - interrupt，向另外一个线程发送中断信号，该线程收到信号，会触发InterruptedException(可解除阻塞)，并进行下一步处理

多线程管理(4)

- 线程被动地暂停和终止
  - 依靠别的线程来拯救自己 ☹☹☹
  - 没有及时释放资源
- 线程主动暂停和终止
  - 定期监测共享变量
  - 如果需要暂停或者终止，先释放资源，再主动动作☺ ☺ ☺
  - 暂停：Thread.sleep()，休眠
  - 终止：run方法结束，线程终止

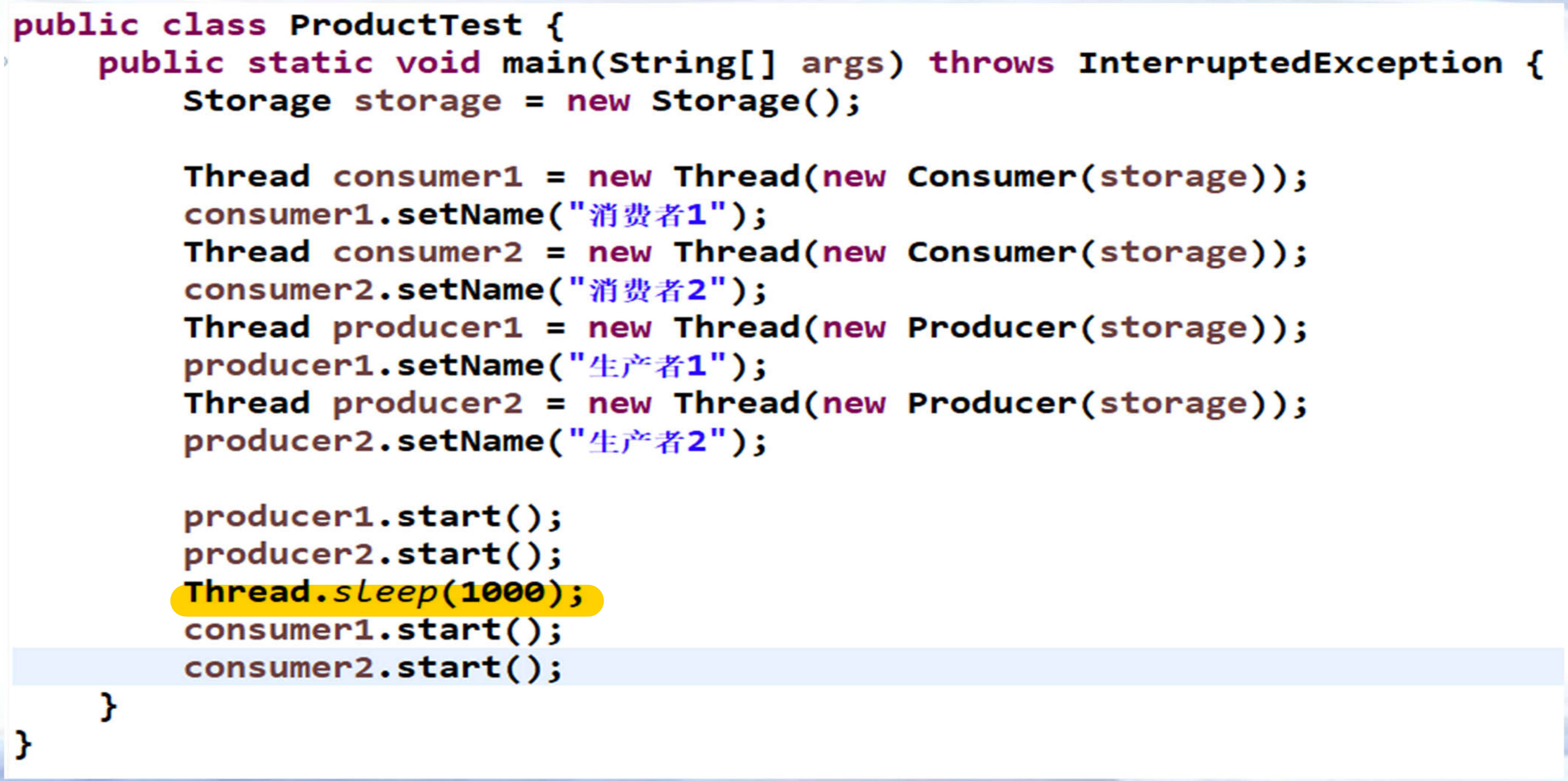
多线程管理(5)

- 多线程死锁
  - 每个线程互相持有别人需要的锁(哲学家吃面问题)
  - 预防死锁，对资源进行等级排序
- 守护(后台)线程
  - 普通线程的结束，是run方法运行结束
  - 守护线程的结束，是run方法运行结束，或main函数结束
  - 守护线程永远不要访问资源，如文件或数据库等
- 线程查看工具 jvisualvm

总结

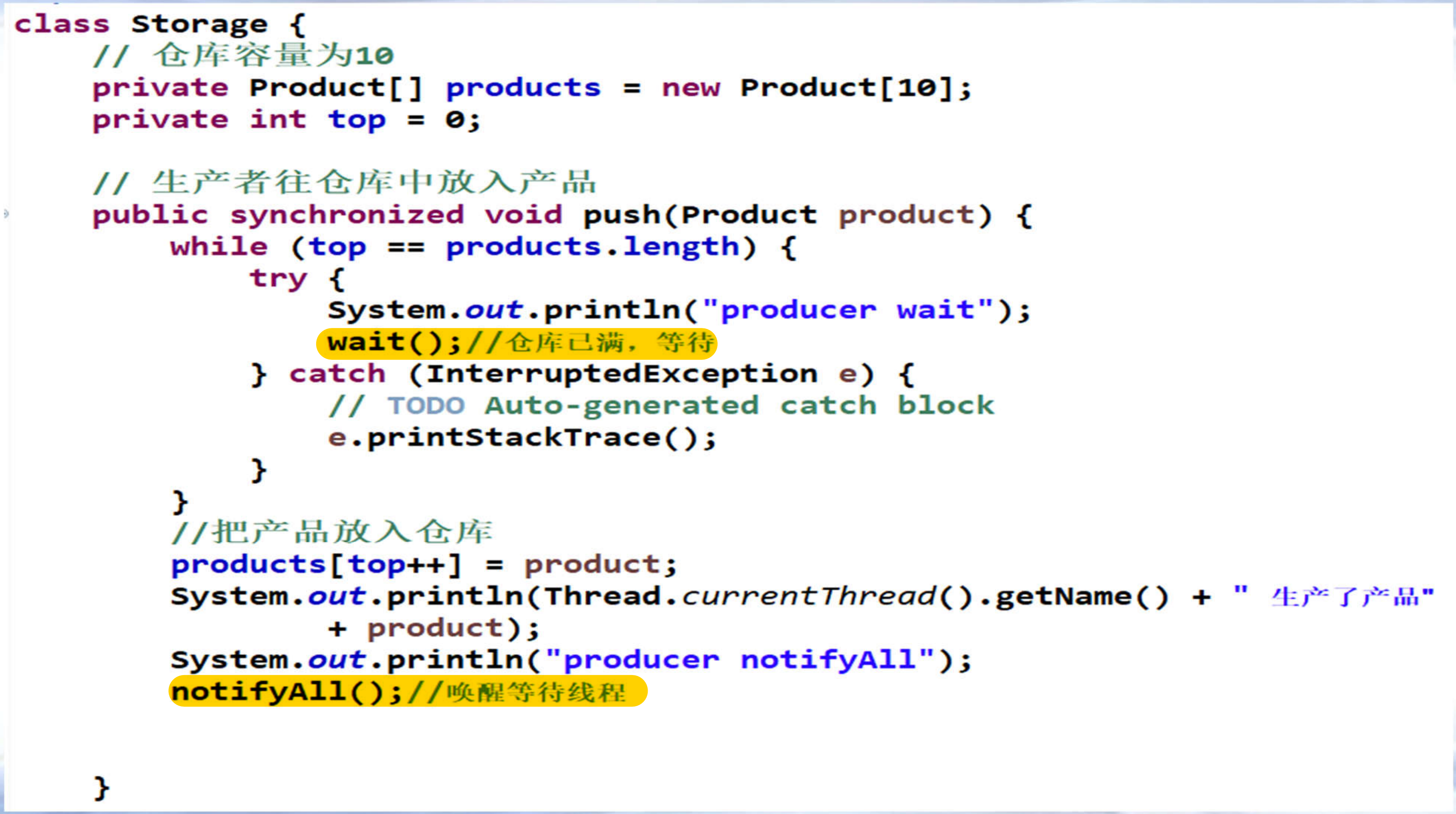
- 总结
  - 了解线程的多个状态
  - 了解线程协作机制
  - 线程协作尽量简单化，采用粗粒度协作
  - 了解死锁和后台线程概念
  - 使用jvisualvm查看线程执行情况

代码(1) ProductTest.java



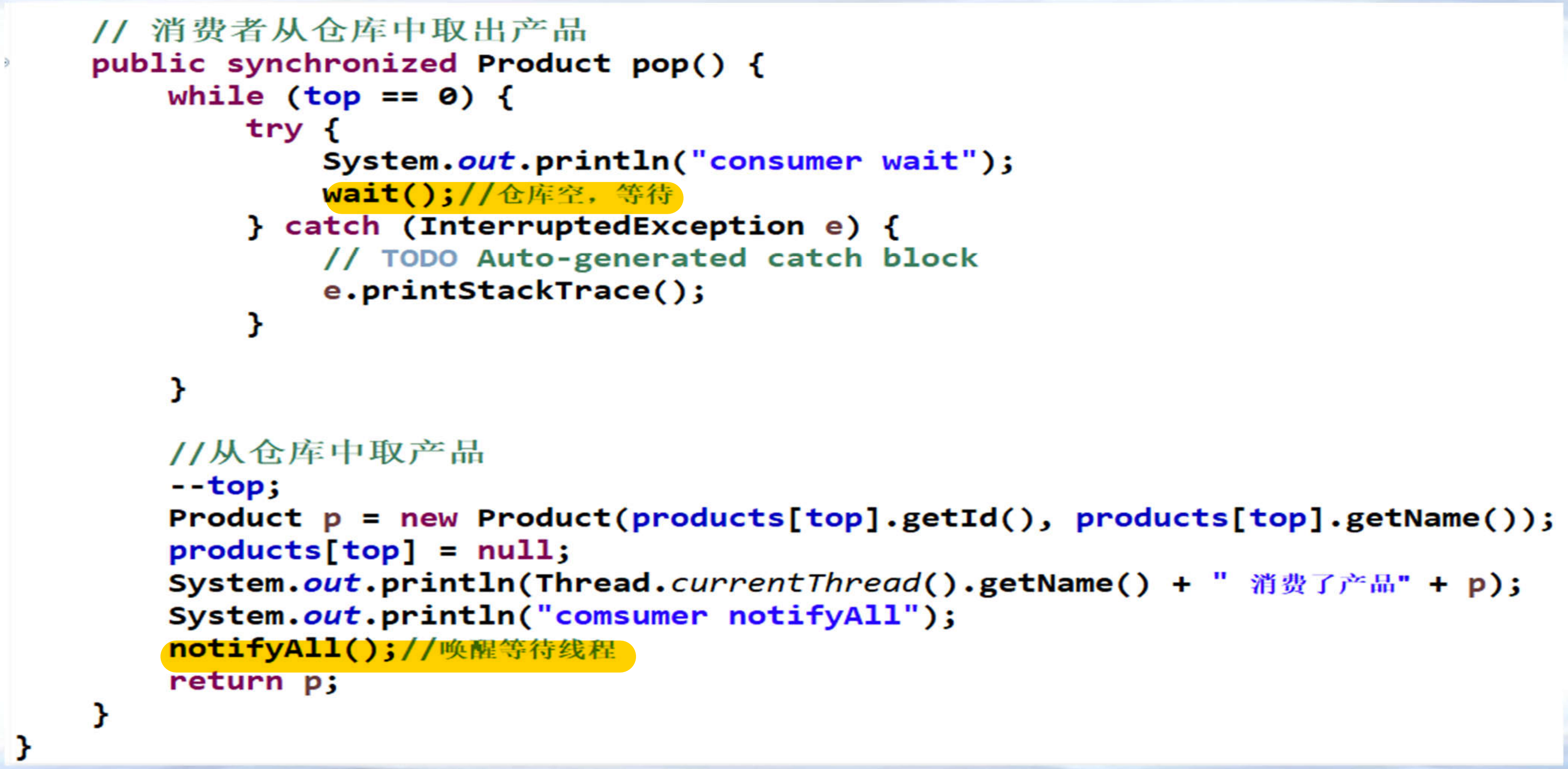
```java
public class ProductTest {
    public static void main(String[] args) throws InterruptedException {
        Storage storage = new Storage();

        Thread consumer1 = new Thread(new Consumer(storage));
        consumer1.setName("消费者1");
        Thread consumer2 = new Thread(new Consumer(storage));
        consumer2.setName("消费者2");
        Thread producer1 = new Thread(new Producer(storage));
        producer1.setName("生产者1");
        Thread producer2 = new Thread(new Producer(storage));
        producer2.setName("生产者2");

        producer1.start();
        producer2.start();
        Thread.sleep(1000);
        consumer1.start();
        consumer2.start();
    }
}
```

代码(2) Storage.java



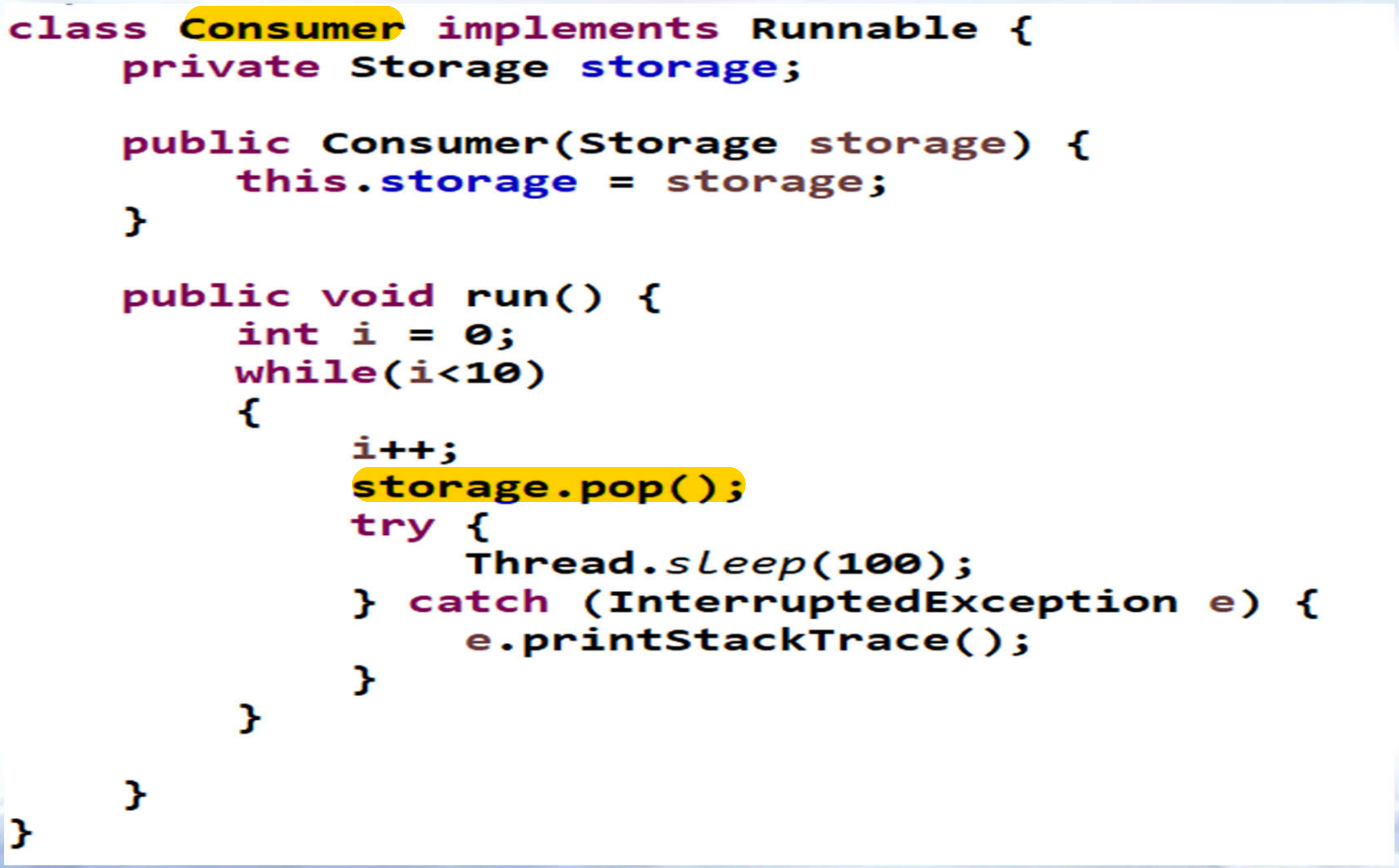
```java
class Storage {
    // 仓库容量为10
    private Product[] products = new Product[10];
    private int top = 0;

    // 生产者往仓库中放入产品
    public synchronized void push(Product product) {
        while (top == products.length) {
            try {
                System.out.println("producer wait");
                wait();//仓库已满，等待
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        //把产品放入仓库
        products[top++] = product;
        System.out.println(Thread.currentThread().getName() + " 生产了产品"
                + product);
        System.out.println("producer notifyAll");
        notifyAll();//唤醒等待线程


    }
```

代码(3) Storage.java



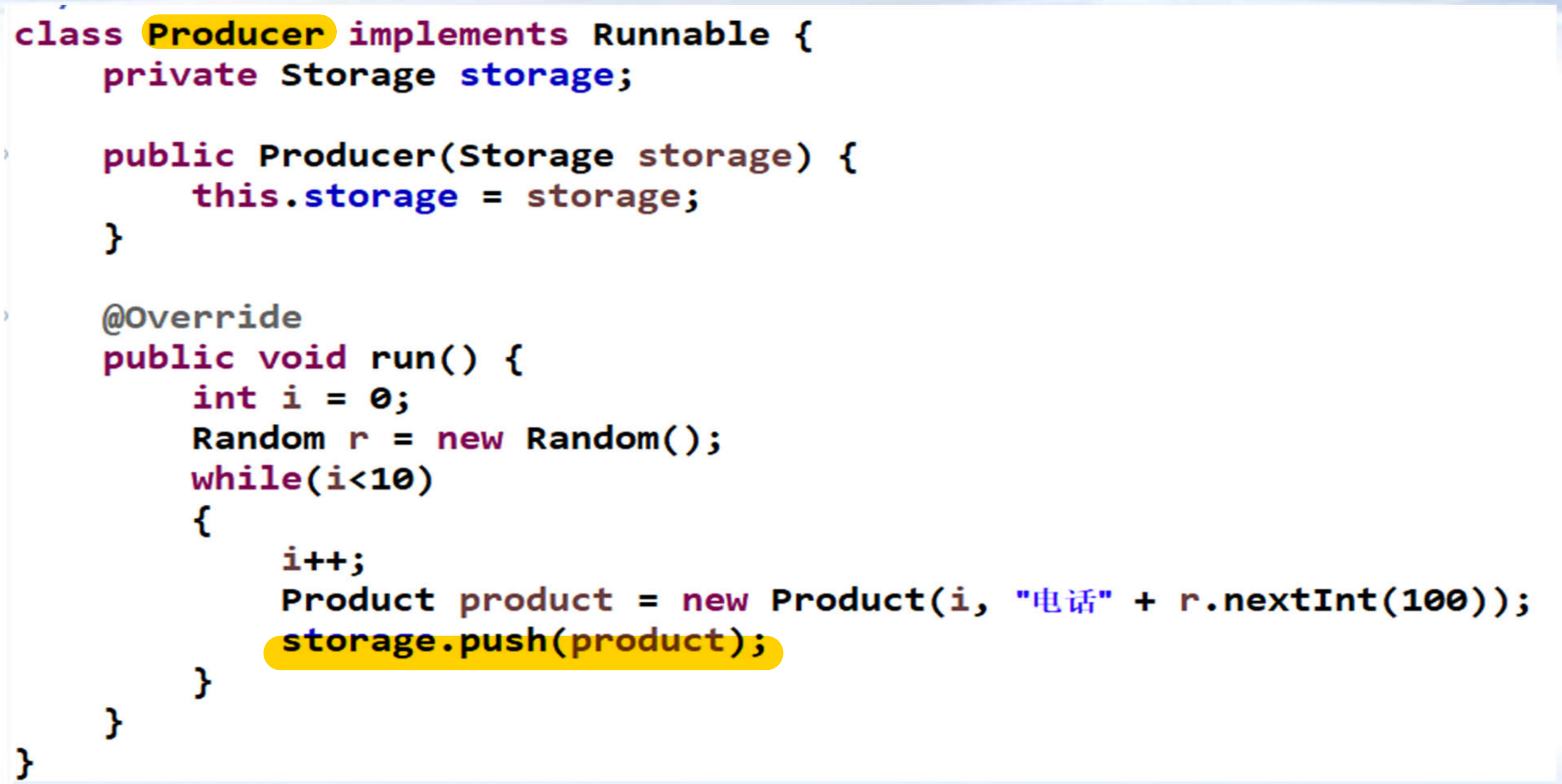
```
    // 消费者从仓库中取出产品
    public synchronized Product pop() {
        while (top == 0) {
            try {
                System.out.println("consumer wait");
                wait();//仓库空，等待
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }

        //从仓库中取产品
        --top;
        Product p = new Product(products[top].getId(), products[top].getName());
        products[top] = null;
        System.out.println(Thread.currentThread().getName() + " 消费了产品" + p);
        System.out.println("comsumer notifyAll");
        notifyAll();//唤醒等待线程
        return p;
    }
}
```

代码(4) Consumer.java



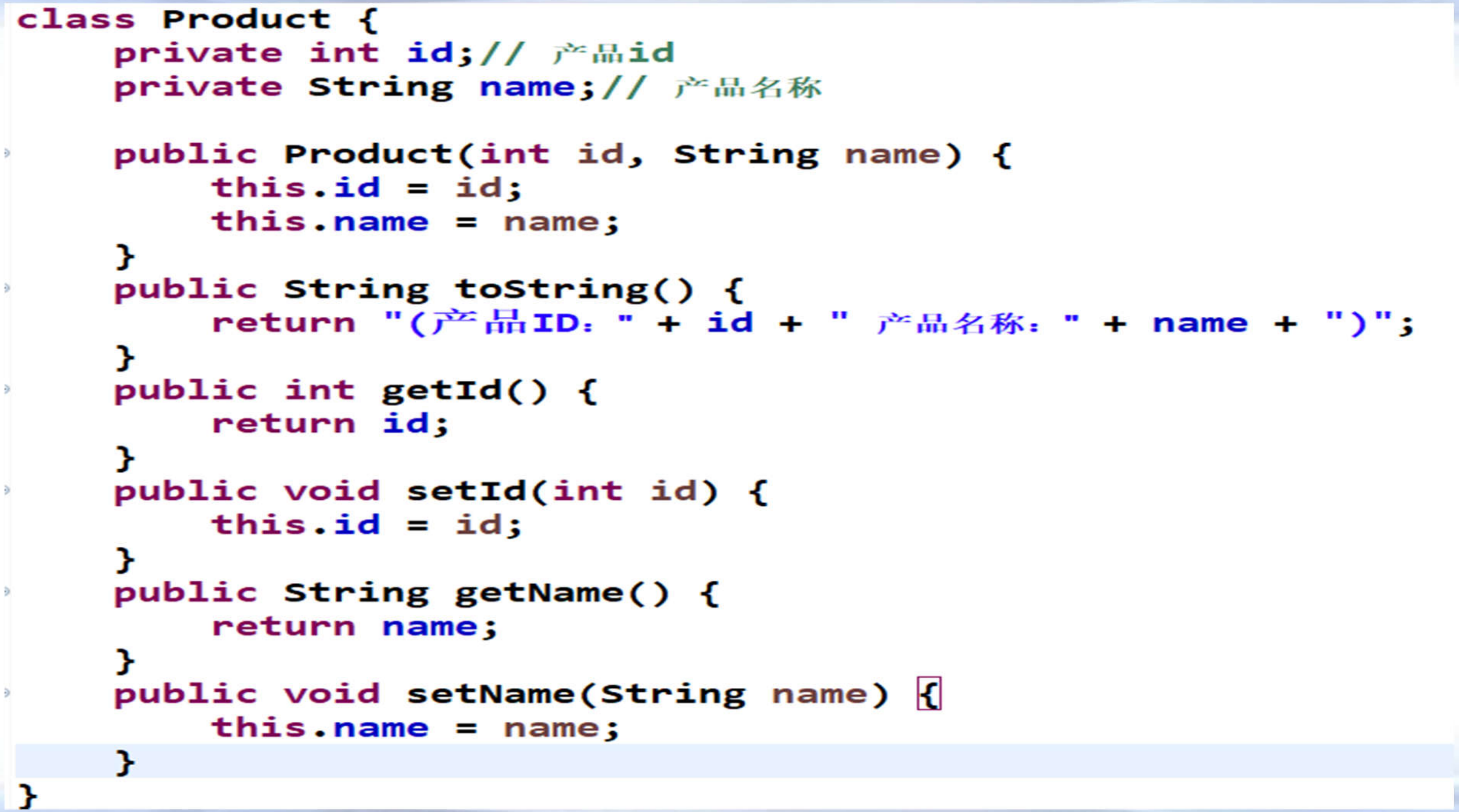
```java
class Consumer implements Runnable {
    private Storage storage;

    public Consumer(Storage storage) {
        this.storage = storage;
    }

    public void run() {
        int i = 0;
        while(i<10)
        {
            i++;
            storage.pop();
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }
}
```

代码(5) Producer.java



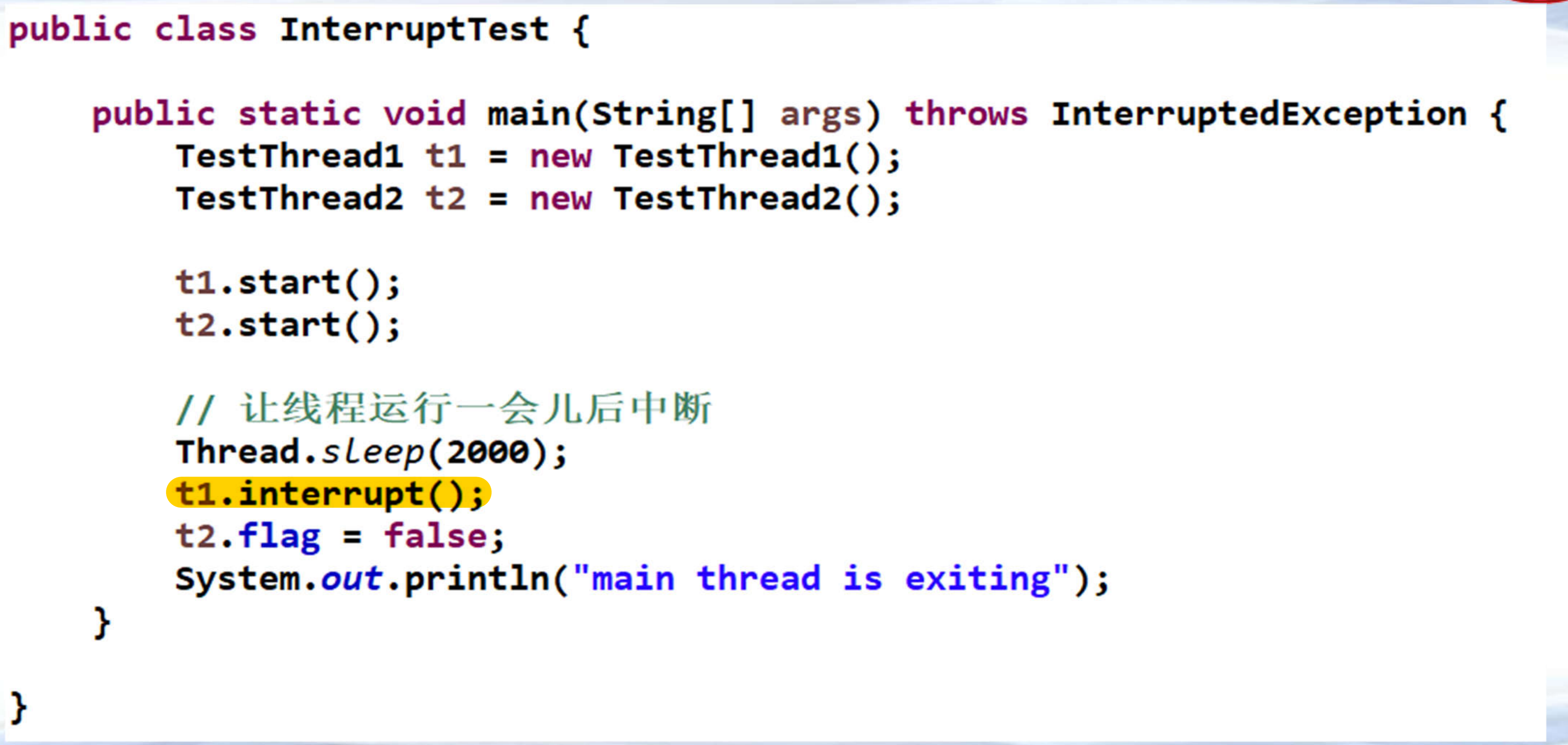
```java
class Producer implements Runnable {
    private Storage storage;

    public Producer(Storage storage) {
        this.storage = storage;
    }

    @Override
    public void run() {
        int i = 0;
        Random r = new Random();
        while(i<10)
        {
            i++;
            Product product = new Product(i, "电话" + r.nextInt(100));
            storage.push(product);
        }
    }
}
```

代码(6) Product.java



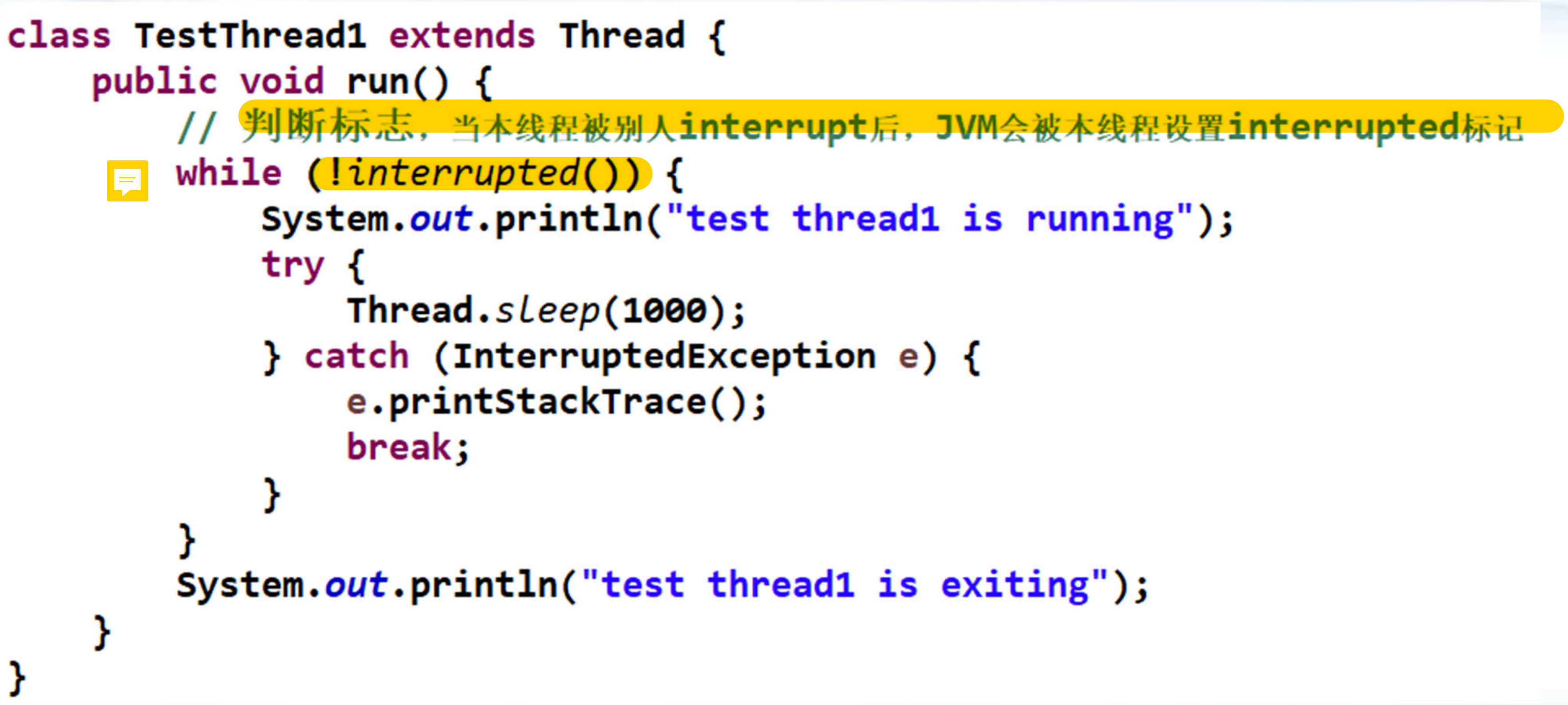
```
class Product {
    private int id;// 产品id
    private String name;// 产品名称

    public Product(int id, String name) {
        this.id = id;
        this.name = name;
    }
    public String toString() {
        return "(产品ID: " + id + " 产品名称: " + name + ")";
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

代码(7) InterruptTest.java



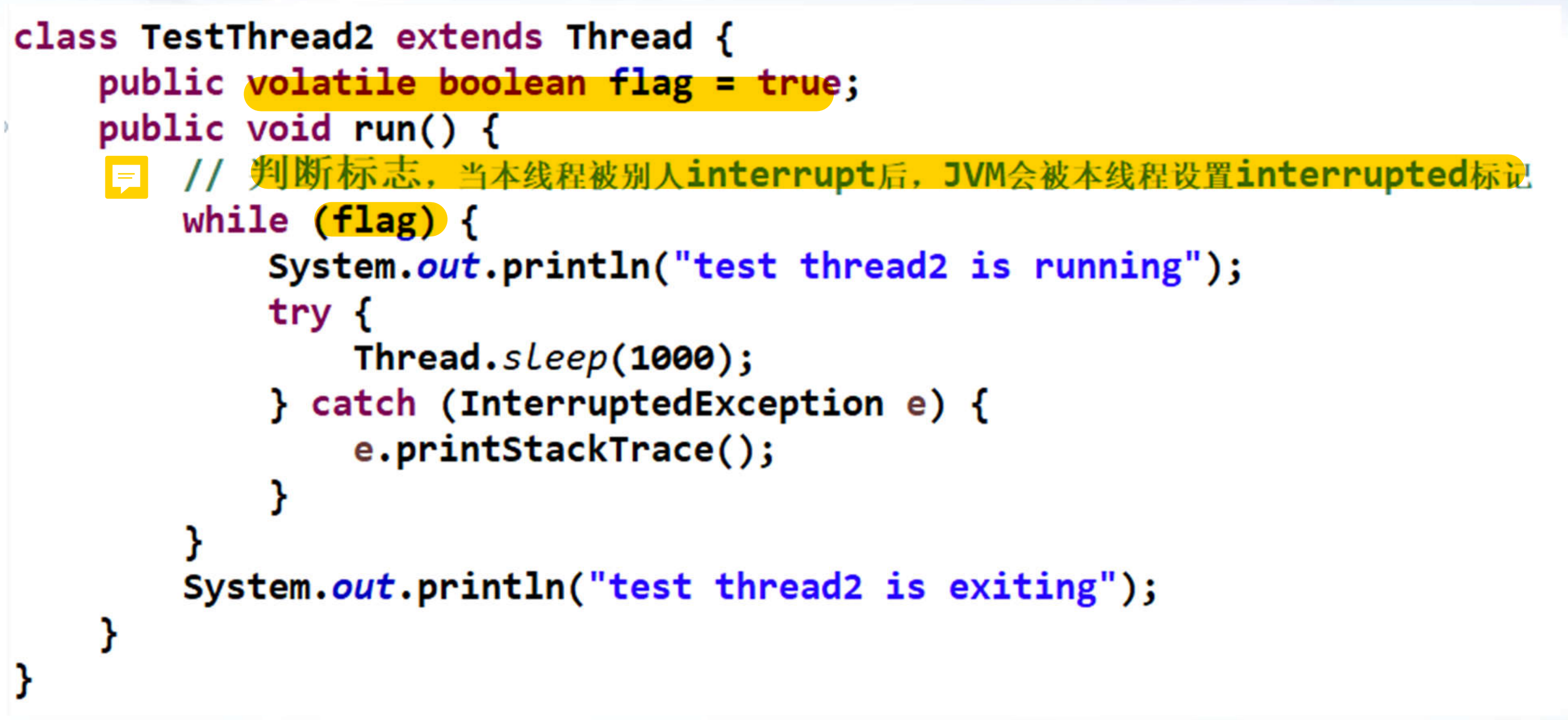
```java
public class InterruptTest {

    public static void main(String[] args) throws InterruptedException {
        TestThread1 t1 = new TestThread1();
        TestThread2 t2 = new TestThread2();

        t1.start();
        t2.start();

        // 让线程运行一会儿后中断
        Thread.sleep(2000);
        t1.interrupt();
        t2.flag = false;
        System.out.println("main thread is exiting");
    }

}
```

代码(8) TestThread1.java



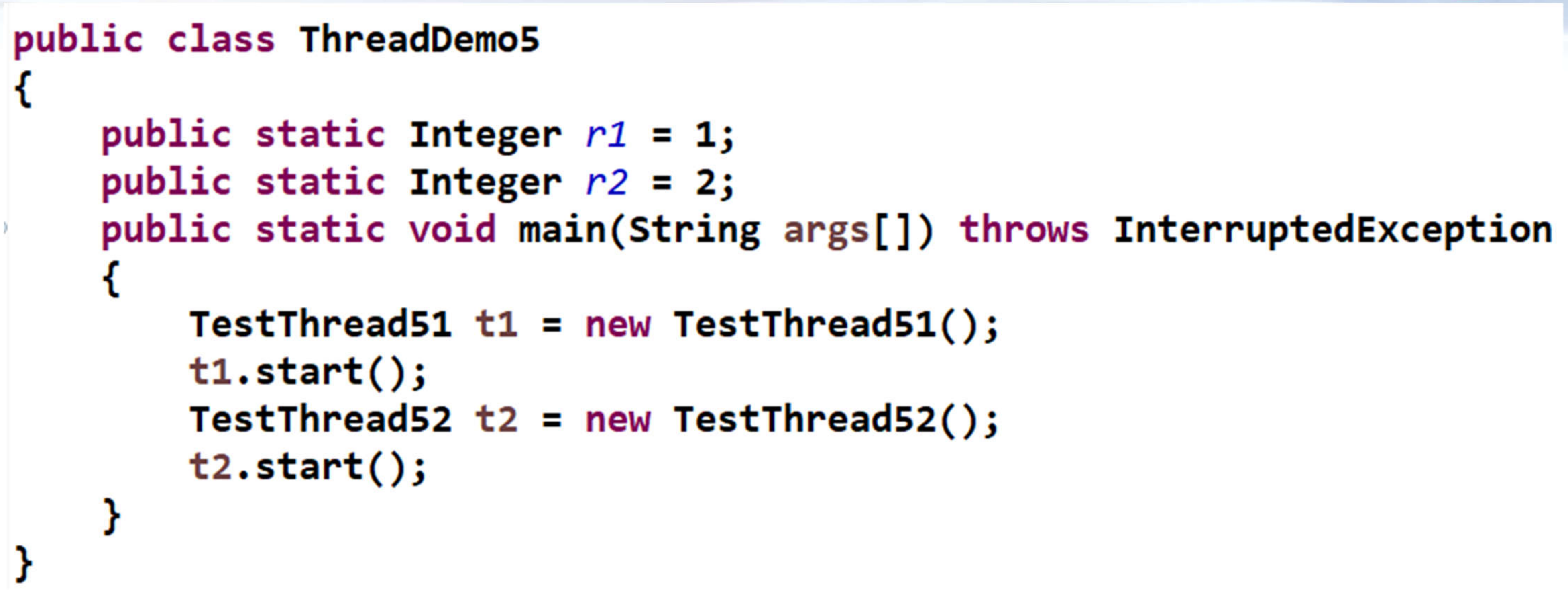
```java
class TestThread1 extends Thread {
    public void run() {
        // 判断标志，当本线程被别人interrupt后，JVM会被本线程设置interrupted标记
        while (!interrupted()) {
            System.out.println("test thread1 is running");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
                break;
            }
        }
        System.out.println("test thread1 is exiting");
    }
}
```

代码(9) TestThread2.java



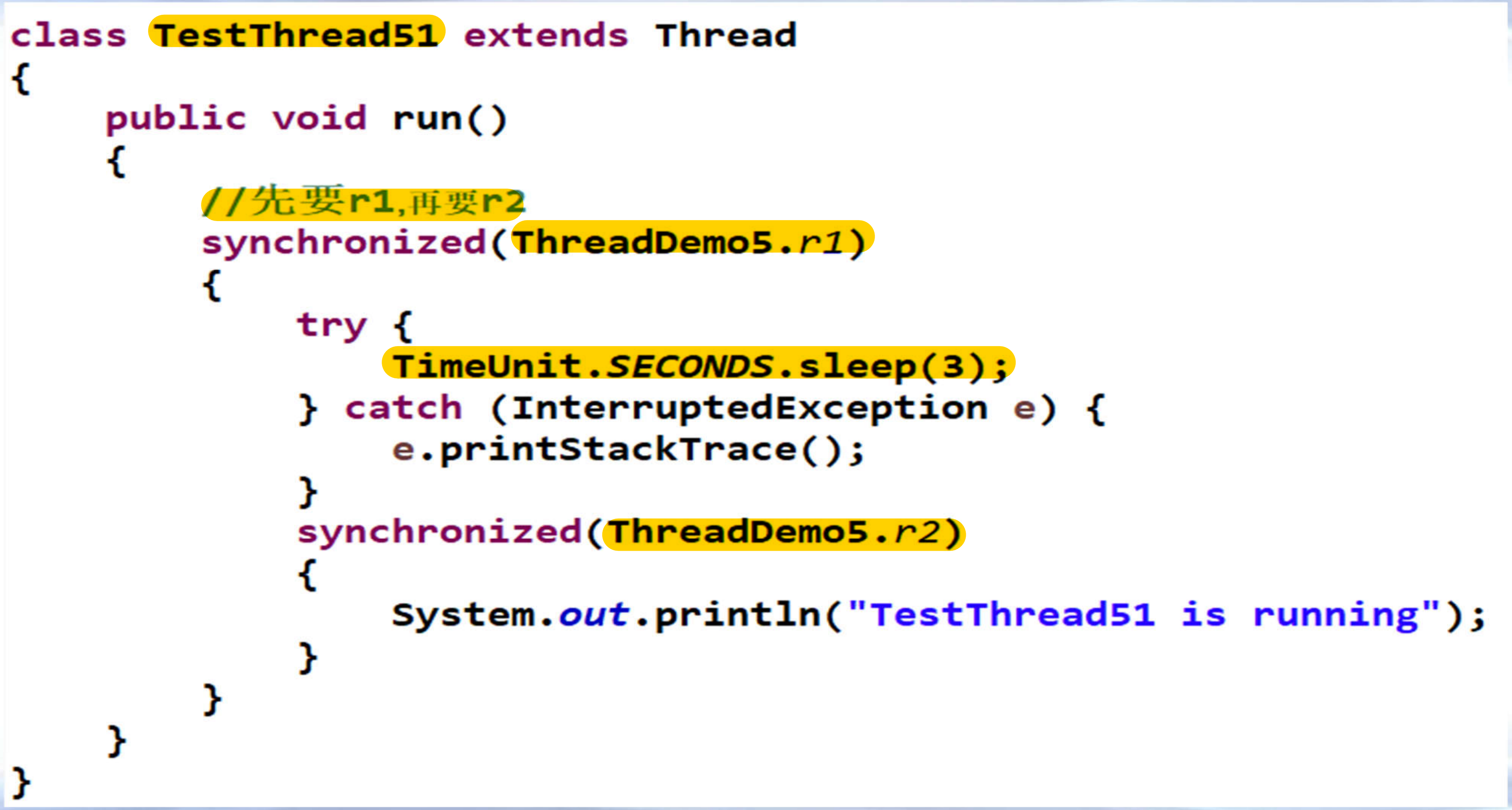
```java
class TestThread2 extends Thread {
    public volatile boolean flag = true;
    public void run() {
        // 判断标志，当本线程被别人interrupt后，JVM会被本线程设置interrupted标记
        while (flag) {
            System.out.println("test thread2 is running");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("test thread2 is exiting");
    }
}
```

代码(10) ThreadDemo5.java



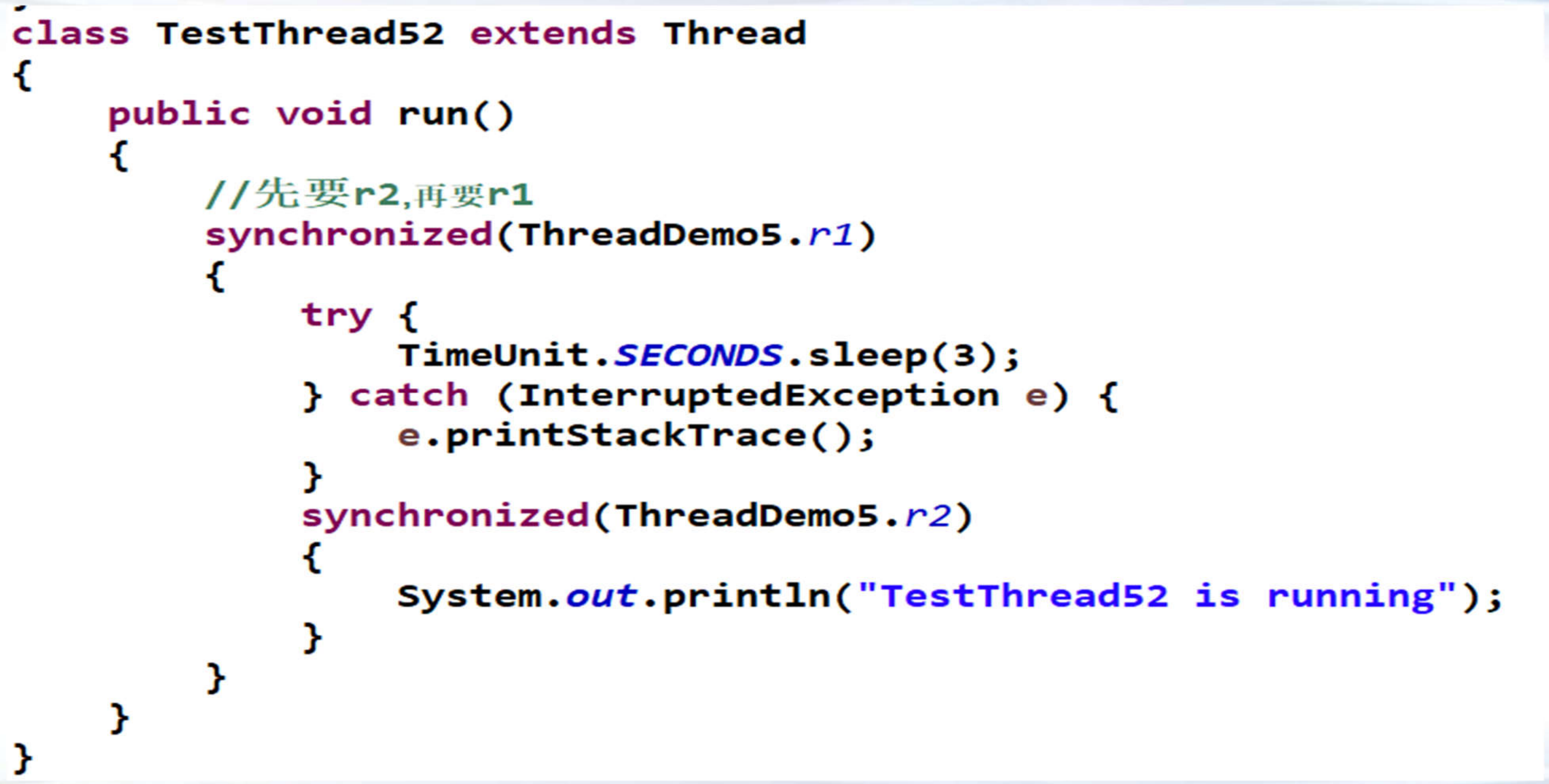
```
public class ThreadDemo5
{
    public static Integer r1 = 1;
    public static Integer r2 = 2;
    public static void main(String args[]) throws InterruptedException
    {
        TestThread51 t1 = new TestThread51();
        t1.start();
        TestThread52 t2 = new TestThread52();
        t2.start();
    }
}
```

代码(11) TestThread51.java



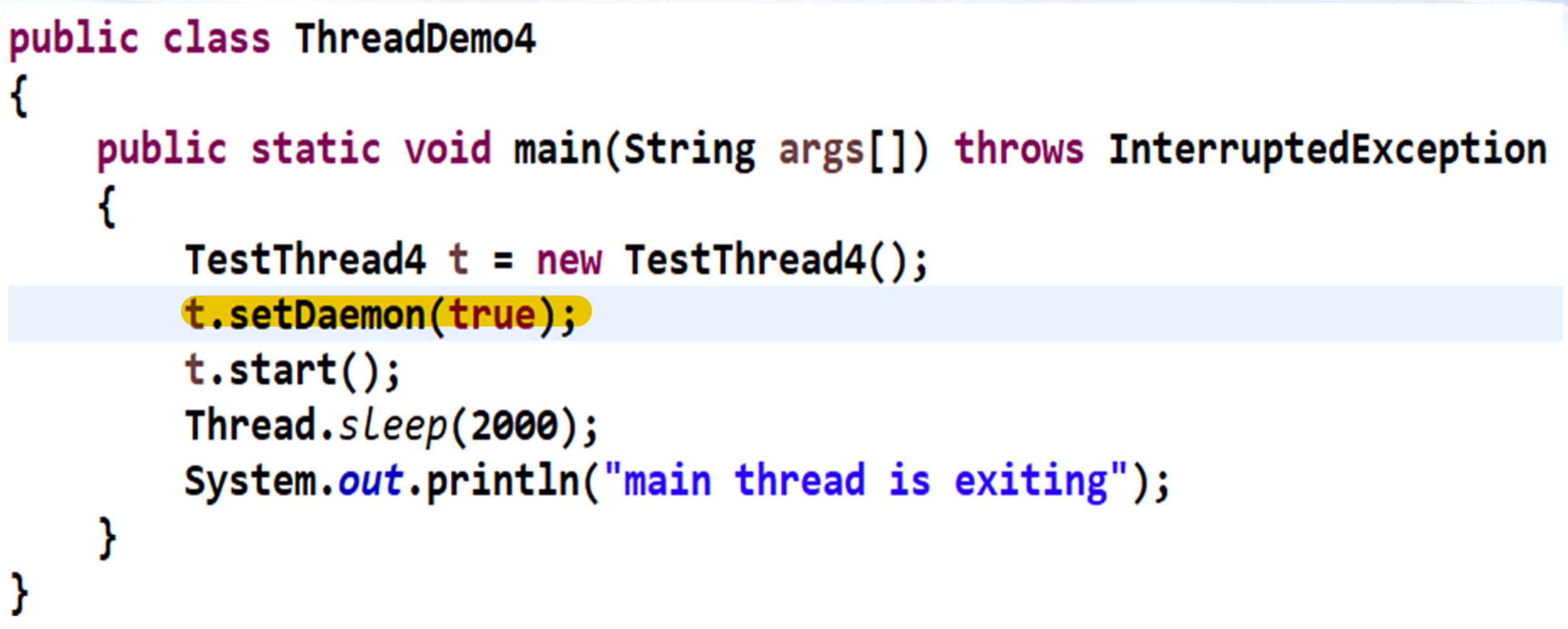
```java
class TestThread51 extends Thread
{
    public void run()
    {
        //先要r1,再要r2
        synchronized(ThreadDemo5.r1)
        {
            try {
                TimeUnit.SECONDS.sleep(3);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            synchronized(ThreadDemo5.r2)
            {
                System.out.println("TestThread51 is running");
            }
        }
    }
}
```

代码(12) TestThread52.java



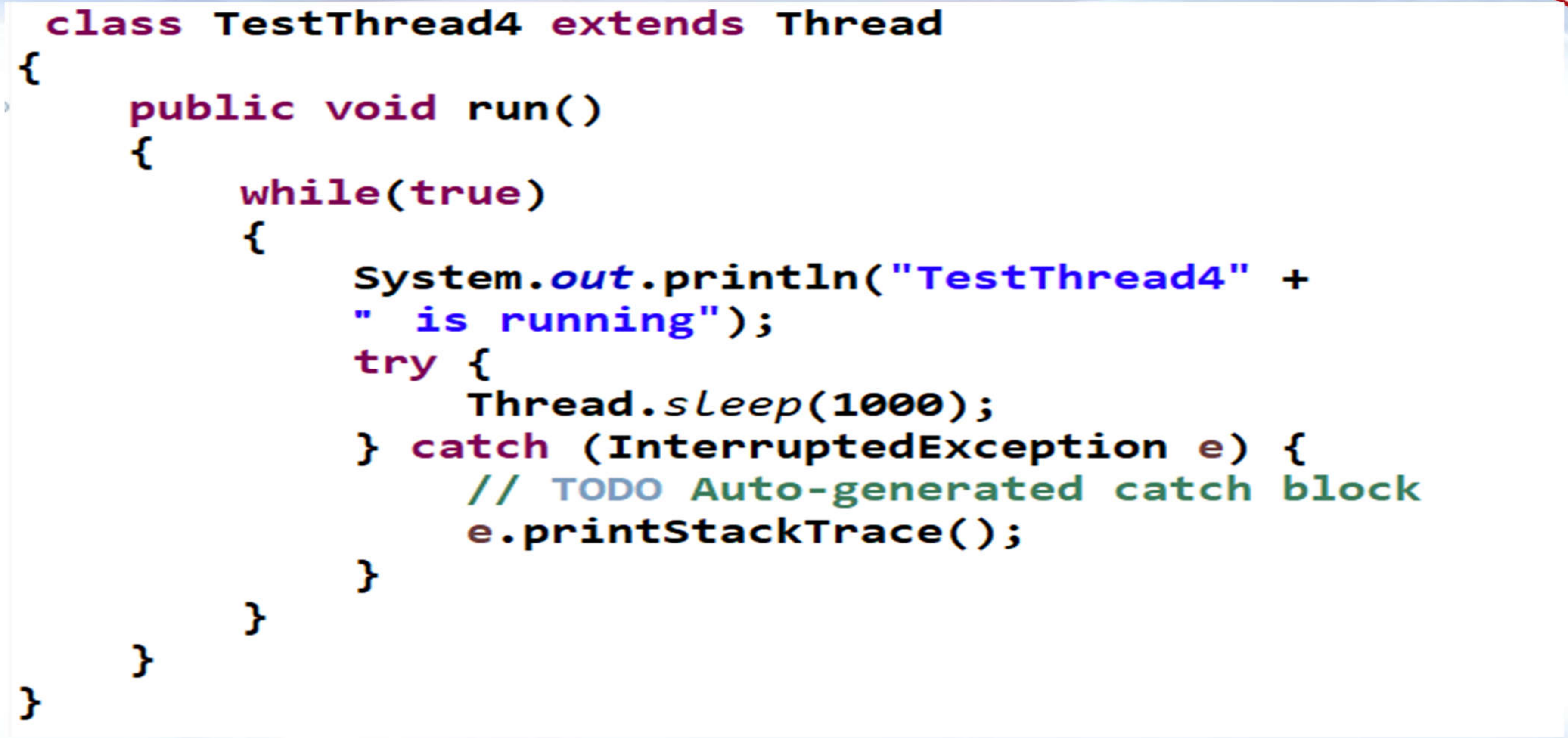
```java
class TestThread52 extends Thread
{
    public void run()
    {
        //先要r2,再要r1
        synchronized(ThreadDemo5.r1)
        {
            try {
                TimeUnit.SECONDS.sleep(3);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            synchronized(ThreadDemo5.r2)
            {
                System.out.println("TestThread52 is running");
            }
        }
    }
}
```

代码(13) ThreadDemo4.java



```
public class ThreadDemo4
{
    public static void main(String args[]) throws InterruptedException
    {
        TestThread4 t = new TestThread4();
        t.setDaemon(true);
        t.start();
        Thread.sleep(2000);
        System.out.println("main thread is exiting");
    }
}
```

代码(14) TestThread4.java



```java
 class TestThread4 extends Thread
{
    public void run()
    {
        while(true)
        {
            System.out.println("TestThread4" +
            " is running");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

谢 谢！