

实验 07：多线程

一、实验目的

1. 掌握多线程的基本概念，理解线程的状态、调度与优先级。
2. 学会通过继承 `Thread` 类和实现 `Runnable` 接口编写多线程程序。
3. 理解线程资源的同步问题，掌握 `synchronized` 关键字和 `wait()`、`notify()` 方法的使用。
4. 熟悉 Java 的任务定时处理，学会使用 `Timer` 和 `TimerTask`。
5. 综合运用多线程的知识，设计并实现一个较为复杂的医院患者管理系统。

二、实验学时

2 学时

三、实验类型

验证性

四、实验需求

1、硬件

每人配备计算机 1 台，建议优先使用个人计算机开展实验。

2、软件

安装 IntelliJ IDEA，以及 Java 运行所需的相关基础环境。

3、网络

本地主机能够访问互联网。

4、工具

无。

五、实验任务

1. 继承 `Thread` 类实现多线程：模拟多个患者同时挂号。
2. 实现 `Runnable` 接口编写多线程：模拟多个医生同时接诊。
3. 线程资源的同步处理：使用 `synchronized` 关键字解决临界资源问题。
4. 线程通信：使用 `wait()` 和 `notify()` 方法实现患者与医生的交互。
5. 任务定时处理：使用 `Timer` 和 `TimerTask` 实现定时任务，如定期检查患者状态。

六、实验内容及步骤

任务 1：继承 `Thread` 类实现多线程

步骤：

1. 设计一个 `Patient` 类继承 `Thread` 类，模拟患者挂号的过程。

示例代码：

Java

```
1 // Patient.java
2 public class Patient extends Thread {
3     private String name;
4
5     public Patient(String name) {
6         this.name = name;
7     }
8
9     @Override
10    public void run() {
11        System.out.println("患者 " + name + " 正在挂号...");
12        try {
13            Thread.sleep(1000); // 模拟挂号耗时
14        } catch (InterruptedException e) {
15            e.printStackTrace();
16        }
17        System.out.println("患者 " + name + " 挂号完成。");
18    }
19 }
20
21 // 测试类
22 public class PatientTest {
23     public static void main(String[] args) {
24         Patient patient1 = new Patient("张三");
25         Patient patient2 = new Patient("李四");
26         Patient patient3 = new Patient("王五");
27
28         patient1.start();
29         patient2.start();
30         patient3.start();
31     }
32 }
```

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** Shows the file `PatientTest.java` under the `ruoyi-admin` project.
- Code Editor:** Displays the `PatientTest` class with a main method that creates three `Patient` objects and starts them.
- Run Tab:** Shows the output of the run command: "患者 张三 正在挂号..." followed by three lines of "挂号完成".
- Bottom Status Bar:** Shows the path `C:\jdk1.8.0_291\bin\java.exe ...`, the process exit code `0`, and other system information like date and time.

图1 患者挂号过程功能展示

任务 2：实现 `Runnable` 接口编写多线程

步骤：

1. 设计一个 `Doctor` 类实现 `Runnable` 接口，模拟医生接诊的过程。

示例代码：

Java

```
1 // Doctor.java
2 public class Doctor implements Runnable {
3     private String name;
4
5     public Doctor(String name) {
6         this.name = name;
7     }
8
9     @Override
10    public void run() {
11        System.out.println("医生 " + name + " 正在接诊...");
12        try {
13            Thread.sleep(2000); // 模拟接诊耗时
14        } catch (InterruptedException e) {
15            e.printStackTrace();
16        }
17        System.out.println("医生 " + name + " 接诊完成。");
18    }
19 }
20
21 // 测试类
22 public class DoctorTest {
23     public static void main(String[] args) {
24         Doctor doctor1 = new Doctor("王医生");
25         Doctor doctor2 = new Doctor("李医生");
26
27         Thread thread1 = new Thread(doctor1);
28         Thread thread2 = new Thread(doctor2);
29
30         thread1.start();
31         thread2.start();
32     }
33 }
```

任务 3：线程资源的同步处理

步骤：

1. 设计一个 `Hospital` 类，模拟医院挂号系统的临界资源问题。
2. 使用 `synchronized` 关键字解决多线程并发访问问题。

示例代码：

Java

```
1 // Hospital.java
2 public class Hospital {
3     private int availableNumbers = 10; // 剩余挂号数量
4
5     // 挂号方法
6     public synchronized void register(Patient patient) {
7         if (availableNumbers > 0) {
8             System.out.println("患者 " + patient.getName() + " 挂号成功,
剩余号数: " + --availableNumbers);
9         } else {
10            System.out.println("患者 " + patient.getName() + " 挂号失败,
号已挂完。");
11        }
12    }
13 }
14
15 // Patient.java
16 public class Patient extends Thread {
17     private String name;
18     private Hospital hospital;
19
20     public Patient(String name, Hospital hospital) {
21         this.name = name;
22         this.hospital = hospital;
23     }
24
25     @Override
26     public void run() {
27         hospital.register(this);
28     }
29 }
30
31 // 测试类
32 public class HospitalTest {
33     public static void main(String[] args) {
34         Hospital hospital = new Hospital();
35
36         for (int i = 1; i <= 15; i++) {
37             Patient patient = new Patient("患者" + i, hospital);
38             patient.start();
39         }
40     }
41 }
```

```
40      }
41 }
```

任务 4：线程通信

步骤：

1. 使用 `wait()` 和 `notify()` 方法实现患者与医生的交互。
2. 模拟患者等待医生接诊的过程。

示例代码：

Java

```
1 // Clinic.java
2 public class Clinic {
3     private boolean isDoctorAvailable = false;
4
5     // 患者等待医生接诊
6     public synchronized void waitForDoctor(Patient patient) {
7         while (!isDoctorAvailable) {
8             try {
9                 System.out.println("患者 " + patient.getName() + " 正在
10                等待医生... ");
11                 wait();
12             } catch (InterruptedException e) {
13                 e.printStackTrace();
14             }
15             System.out.println("患者 " + patient.getName() + " 被医生接
16                诊。");
17         }
18         // 医生接诊患者
19         public synchronized void consultPatient(Doctor doctor) {
20             System.out.println("医生 " + doctor.getName() + " 开始接诊。");
21             isDoctorAvailable = true;
22             notifyAll();
23         }
24     }
25
26 // Patient.java
27 public class Patient extends Thread {
28     private String name;
29     private Clinic clinic;
30
31     public Patient(String name, Clinic clinic) {
32         this.name = name;
33         this.clinic = clinic;
34     }
35
36     @Override
37     public void run() {
38         clinic.waitForDoctor(this);
39     }
40 }
```

```

41
42 // Doctor.java
43 public class Doctor extends Thread {
44     private String name;
45     private Clinic clinic;
46
47     public Doctor(String name, Clinic clinic) {
48         this.name = name;
49         this.clinic = clinic;
50     }
51
52     @Override
53     public void run() {
54         clinic.consultPatient(this);
55     }
56 }
57
58 // 测试类
59 public class ClinicTest {
60     public static void main(String[] args) {
61         Clinic clinic = new Clinic();
62
63         Patient patient1 = new Patient("张三", clinic);
64         Patient patient2 = new Patient("李四", clinic);
65         Doctor doctor = new Doctor("王医生", clinic);
66
67         patient1.start();
68         patient2.start();
69         doctor.start();
70     }
71 }

```



图2 医生接诊功能展示

任务 5：任务定时处理

步骤：

1. 使用 `Timer` 和 `TimerTask` 实现定时任务，如定期检查患者状态。

示例代码：

Java

```
1 import java.util.Timer;
2 import java.util.TimerTask;
3
4 // PatientMonitor.java
5 public class PatientMonitor {
6     public static void main(String[] args) {
7         Timer timer = new Timer();
8
9         // 定时任务：每5秒检查一次患者状态
10        timer.schedule(new TimerTask() {
11            @Override
12            public void run() {
13                System.out.println("正在检查患者状态... ");
14                // 模拟检查逻辑
15                System.out.println("患者状态正常。");
16            }
17        }, 0, 5000); // 延迟0毫秒，每隔5000毫秒执行一次
18    }
19 }
```

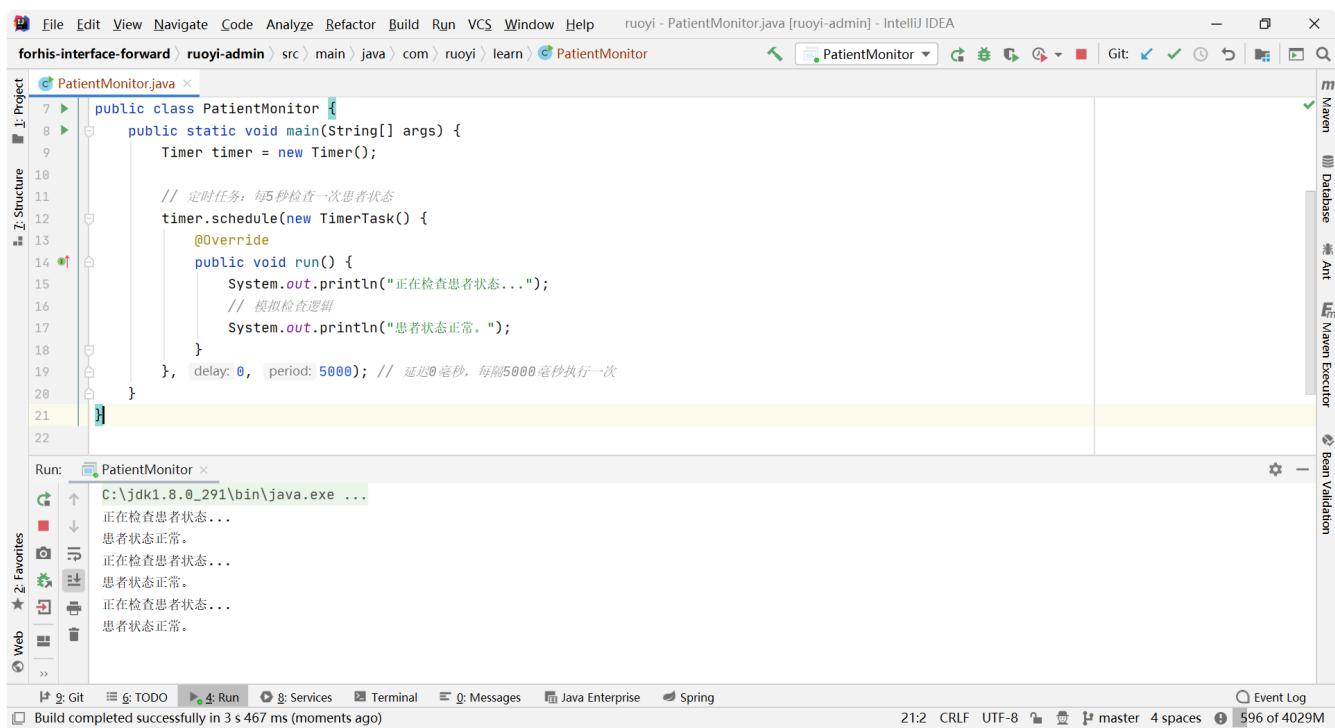


图3 定时任务功能展示

七、实验考核

本实验考核采用【实验随堂查】方式开展。

每个实验完成后，在实验课上通过现场演示的方式向实验指导教师进行汇报，并完成现场问答交流。

每个实验考核满分 100 分，其中实验成果汇报 60 分，现场提问交流 40 分。

实验考核流程：

- (1) 学生演示汇报实验内容的完成情况，实验指导老师现场打分。
- (2) 指导老师结合实验内容进行提问，每位学生提问 2-3 个问题，根据回答的情况现场打分。
- (3) 实验考核结束后，进行公布成绩。