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 运行所需要的相关基础环境。

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

```java
// Patient.java
public class Patient extends Thread {
    private String name;

    public Patient(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        System.out.println("患者 " + name + " 正在挂号...");
        try {
            Thread.sleep(1000); // 模拟挂号耗时
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("患者 " + name + " 挂号完成。");
    }
}

// 测试类
public class PatientTest {
    public static void main(String[] args) {
        Patient patient1 = new Patient("张三");
        Patient patient2 = new Patient("李四");
        Patient patient3 = new Patient("王五");

        patient1.start();
        patient2.start();
        patient3.start();
    }
}
```

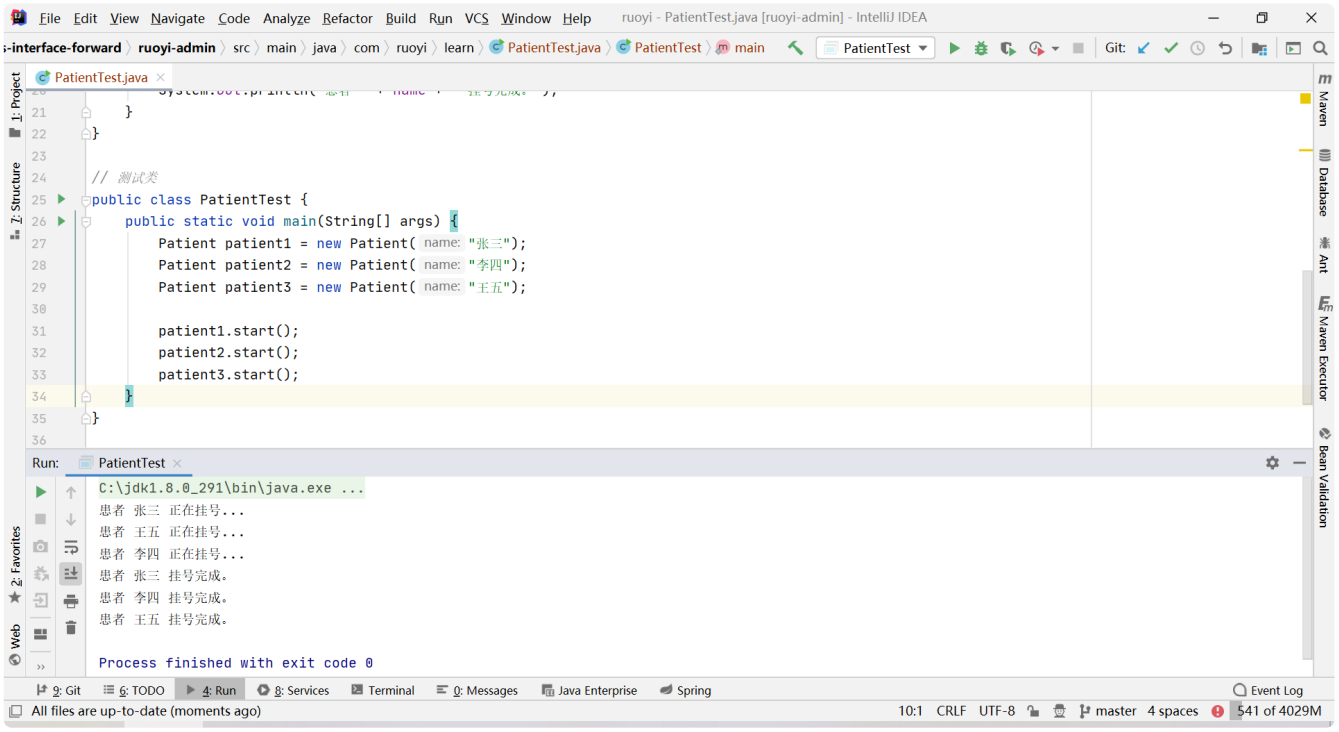

图1 患者挂号过程功能展示

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

```java
// Doctor.java
public class Doctor implements Runnable {
    private String name;

    public Doctor(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        System.out.println("医生 " + name + " 正在接诊...");
        try {
            Thread.sleep(2000); // 模拟接诊耗时
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("医生 " + name + " 接诊完成。");
    }
}

// 测试类
public class DoctorTest {
    public static void main(String[] args) {
        Doctor doctor1 = new Doctor("王医生");
        Doctor doctor2 = new Doctor("李医生");

        Thread thread1 = new Thread(doctor1);
        Thread thread2 = new Thread(doctor2);

        thread1.start();
        thread2.start();
    }
}
```

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

```java
// Hospital.java
public class Hospital {
    private int availableNumbers = 10; // 剩余挂号数量

    // 挂号方法
    public synchronized void register(Patient patient) {
        if (availableNumbers > 0) {
            System.out.println("患者 " + patient.getName() + " 挂号成功，剩余号数: " + --availableNumbers);
        } else {
            System.out.println("患者 " + patient.getName() + " 挂号失败，号已挂完。");
        }
    }
}

// Patient.java
public class Patient extends Thread {
    private String name;
    private Hospital hospital;

    public Patient(String name, Hospital hospital) {
        this.name = name;
        this.hospital = hospital;
    }

    @Override
    public void run() {
        hospital.register(this);
    }
}

// 测试类
public class HospitalTest {
    public static void main(String[] args) {
        Hospital hospital = new Hospital();

        for (int i = 1; i <= 15; i++) {
            Patient patient = new Patient("患者" + i, hospital);
            patient.start();
        }
```

```
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

```java
// Clinic.java
public class Clinic {
    private boolean isDoctorAvailable = false;

    // 患者等待医生接诊
    public synchronized void waitForDoctor(Patient patient) {
        while (!isDoctorAvailable) {
            try {
                System.out.println("患者 " + patient.getName() + " 正在等待医生...");
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("患者 " + patient.getName() + " 被医生接诊。");
    }

    // 医生接诊患者
    public synchronized void consultPatient(Doctor doctor) {
        System.out.println("医生 " + doctor.getName() + " 开始接诊。");
        isDoctorAvailable = true;
        notifyAll();
    }
}

// Patient.java
public class Patient extends Thread {
    private String name;
    private Clinic clinic;

    public Patient(String name, Clinic clinic) {
        this.name = name;
        this.clinic = clinic;
    }

    @Override
    public void run() {
        clinic.waitForDoctor(this);
    }
}
```

```java

// Doctor.java
public class Doctor extends Thread {
    private String name;
    private Clinic clinic;

    public Doctor(String name, Clinic clinic) {
        this.name = name;
        this.clinic = clinic;
    }

    @Override
    public void run() {
        clinic.consultPatient(this);
    }
}

// 测试类
public class ClinicTest {
    public static void main(String[] args) {
        Clinic clinic = new Clinic();

        Patient patient1 = new Patient("张三", clinic);
        Patient patient2 = new Patient("李四", clinic);
        Doctor doctor = new Doctor("王医生", clinic);

        patient1.start();
        patient2.start();
        doctor.start();
    }
}
```

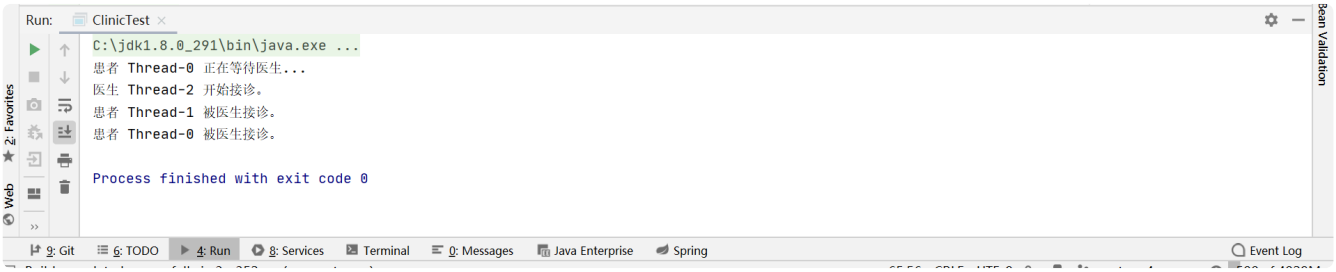

图2 医生接诊功能展示

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

```java
import java.util.Timer;
import java.util.TimerTask;

// PatientMonitor.java
public class PatientMonitor {
    public static void main(String[] args) {
        Timer timer = new Timer();

        // 定时任务：每5秒检查一次患者状态
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                System.out.println("正在检查患者状态...");
                // 模拟检查逻辑
                System.out.println("患者状态正常。");
            }
        }, 0, 5000); // 延迟0毫秒，每隔5000毫秒执行一次
    }
}
```

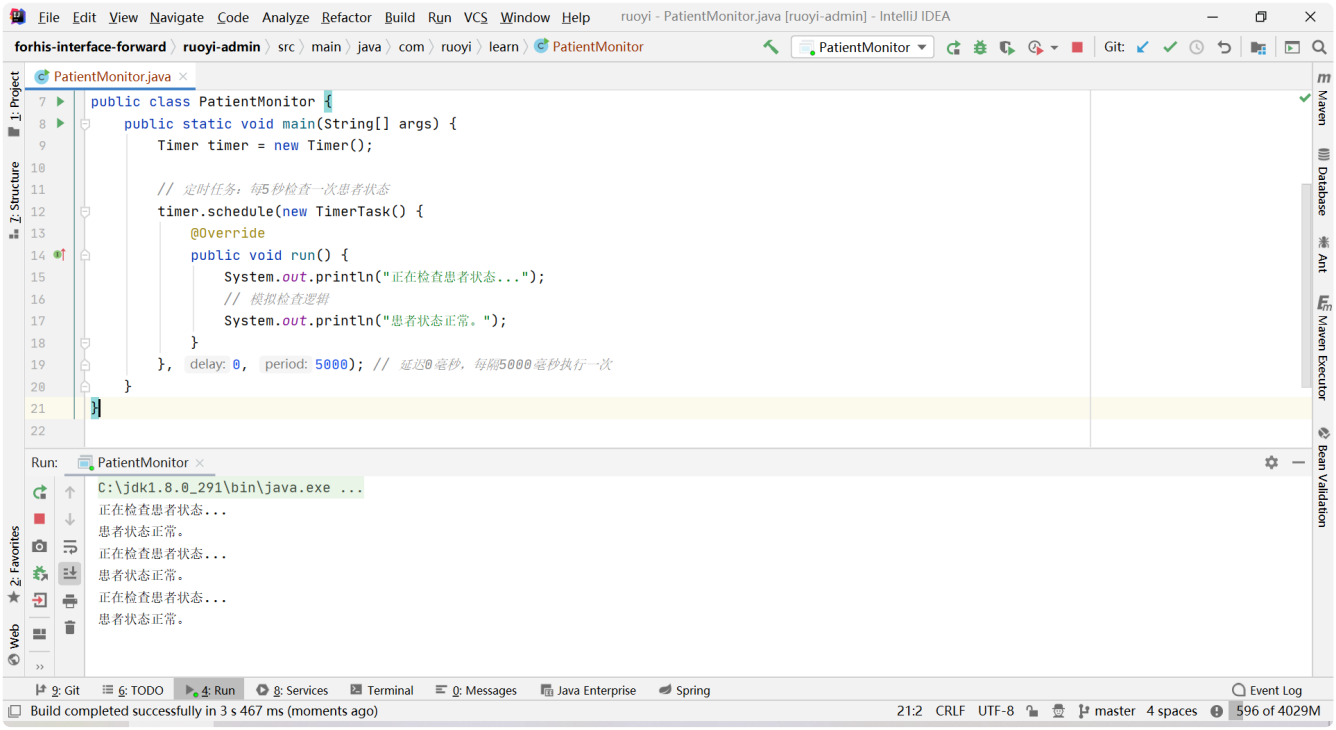

图3 定时任务功能展示

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。