**哲学家就餐问题**

1. **课题内容和要求**

课题内容：

哲学家就餐问题是在计算机科学中的一个经典问题，用来描述进程并发执行时产生的同步问题。在1971年，著名的计算机科学家 Edsger W. Dijkstra 给学生出的一个同步问题，即假设有五台计算机都试图访问五台共享的磁带机。不久之后， Tony Hoare 进一步将这个问题重新表述为哲学家就餐问题。这个问题可以用来解释同步和死锁。阅读操作系统教材，了解哲学家就餐问题，了解进程同步和死锁的相关概念。请使用信号量机制，实现哲学家就餐问题

课题要求：

(1) 查阅相关资料，了解信号量操作相关系统调用；

(2) 实现哲学家就餐问题；

(3) 模拟实现哲学家就餐问题中死锁的出现；

(4) 为了避免死锁的出现，进一步修改程序，使得不出现死锁。（如：限定并发进程的数目， 限定取左右叉子的顺序等等）

(5) 要有简单的交互界面。

(6) 可根据自己能力，在完成以上基本要求后，对程序功能进行适当扩充。 (7) 撰写报告，对所采用的算法、程序结构和主要函数过程以及关键变量进

行详细的说明； 对程序的调试过程所遇到的问题进行回顾和分析，对测试和运行结果进行分析；总结软件设计和实习的经验和体会，进一步改进的设想。

(8) 提供关键程序的清单、源程序及可执行文件和相关的软件说明；能所涉

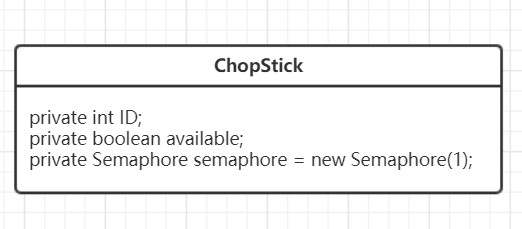
及的关联约束。

1. **需求分析**

哲学家进餐问题由Dijkstra提出，问题描述有五个哲学家共用一张圆桌，分别坐在周围的五张椅子上，在圆桌上有五个碗和五支筷子，他们的生活方式是交替地进行思考和进餐。平时，一个哲学家进行思考，饥饿时便试图取用其左右最靠近他的筷子，只有在他拿到两只筷子时才能进餐。进餐完毕，放下筷子继续思考。当系统中两个或者多个进程无限期地等待永远不会发生的条件，系统就会处于停滞状态。这时我们就程进程处于死锁状态。

经分析可知，放在桌子上的筷子是临界资源，在一段时间内只允许一位哲学家使用。为了实现对筷子的互斥，可以用一个信号量表示一只筷子。当哲学家饥饿时总是先去拿左筷子，成功后在拿右筷子。当五位哲学家同时拿起左筷子，这是每位哲学家都没有右筷子可以拿，就会造成死锁，所以最多只允许4位哲学家同时去拿左筷子，只有拿到左筷子才能继续拿右筷子。拿到两双筷子并用餐之后，先释放左筷子，在释放右筷子，这样就可以避免死锁操作。

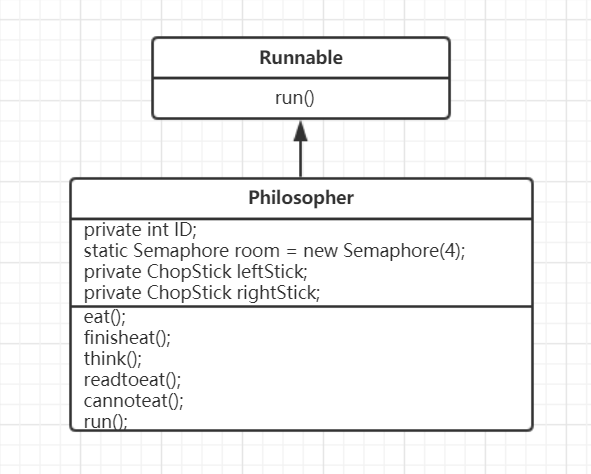
1. **概要设计**
2. **筷子类设计**



布尔类型值available表示筷子是否被占用；

信号量semaphore用来实现筷子的互斥选取

1. **哲学家类设计**



哲学家类实现了runnable接口，用来进行并发控制。

私有成员leftStick，rightStick表示哲学家拿起的左筷子和右筷子。

设置信号量room控制最多允许四个哲学家同时拿左筷子；

Run（）方法实现哲学家就餐问题

1. **界面设计**

命令行显示界面，选择进行的操作

**四、详细设计**

**1,筷子类实现**

**public** **class** ChopStick {

**private** **int** ID;

**private** **boolean** available;

**private** Semaphore semaphore = **new** Semaphore(1);

**public** ChopStick(**int** ID){

**this**.ID = ID;

**this**.available = **true**;

**this**.semaphore = **new** Semaphore(1);

}

…

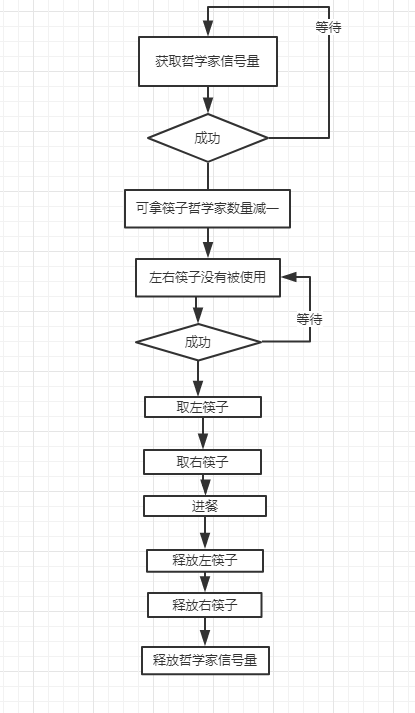
get（） and set();

…

}

**2,哲学家类实现**

**2.1哲学家就餐问题流程图:**



2.2代码实现：

public class Philosopher implements Runnable{

private int ID;

static Semaphore room = new Semaphore(4);

private ChopStick leftStick;

private ChopStick rightStick;

public Philosopher(int ID, ChopStick chopStickLeft, ChopStick chopStickRight){

this.ID = ID;

this.leftStick = chopStickLeft;

this.rightStick = chopStickRight;

}

…

Get() and set();

…

public void eat(){

getLeftChopStick();

getRightChopStick();

System.out.println("哲学家"+ this.getId() + "正在用餐。。。");

}

public void think(){

System.out.println("哲学家" + this.getId() + "正在思考。。。");

}

public void finishEat(){

System.out.println("哲学家" + this.getId() + "用餐结束,正在思考。。。");

dropLeftChopStick();

dropRightChopStick();

}

public void readyToEat(){

System.out.println("哲学家" + this.getId() + "饿了准备用餐。。。");

}

public void cannotEat(){

System.out.println("哲学家" + this.getId() + "缺少筷子，不能用餐，等待。。。");

}

public void run(){

try{

room.acquire();

this.readyToEat();

if(this.leftStick.getSemaphore().availablePermits() == 0 ||

this.leftStick.getSemaphore().availablePermits() == 0){

this.cannotEat();

}

this.leftStick.getSemaphore().acquire();

System.out.println("哲学家"+this.getId()+"获取左筷子："+this.leftStick.getID());

Thread.sleep(1000 \* 1);

this.rightStick.getSemaphore().acquire();

System.out.println("哲学家"+this.getId()+"获取右筷子："+this.rightStick.getID());

this.eat();

Thread.sleep(1000 \* 2);

this.finishEat();

this.leftStick.getSemaphore().release();

System.out.println("哲学家"+this.getId()+"放下左筷子："+this.leftStick.getID());

this.rightStick.getSemaphore().release();

System.out.println("哲学家"+this.getId()+"放下右筷子："+this.leftStick.getID());

room.release();

}catch(InterruptedException ex){

ex.toString();

}

}

}

3主**函数：**

public class Test {

public static void main(String[] args){

Scanner input = new Scanner(System.in);

menu();

int choice = input.nextInt();

while(choice != 1){

if(choice == 0){

ChopStick[] chopStick = new ChopStick[5];

for(int i = 0; i < 5; i ++){

chopStick[i] = new ChopStick(i);

}

ExecutorService excutor = Executors.newFixedThreadPool(5);

Philosopher ph0 = new Philosopher(0, chopStick[0], chopStick[1]);

excutor.execute(new Philosopher(0, chopStick[0], chopStick[1]));

excutor.execute(new Philosopher(1, chopStick[1], chopStick[2]));

excutor.execute(new Philosopher(2, chopStick[2], chopStick[3]));

excutor.execute(new Philosopher(3, chopStick[3], chopStick[4]));

excutor.execute(new Philosopher(4, chopStick[4], chopStick[0]));

excutor.shutdown();

}

choice = input.nextInt();

menu();

}

}

public static void menu(){

System.out.println("0: 演示");

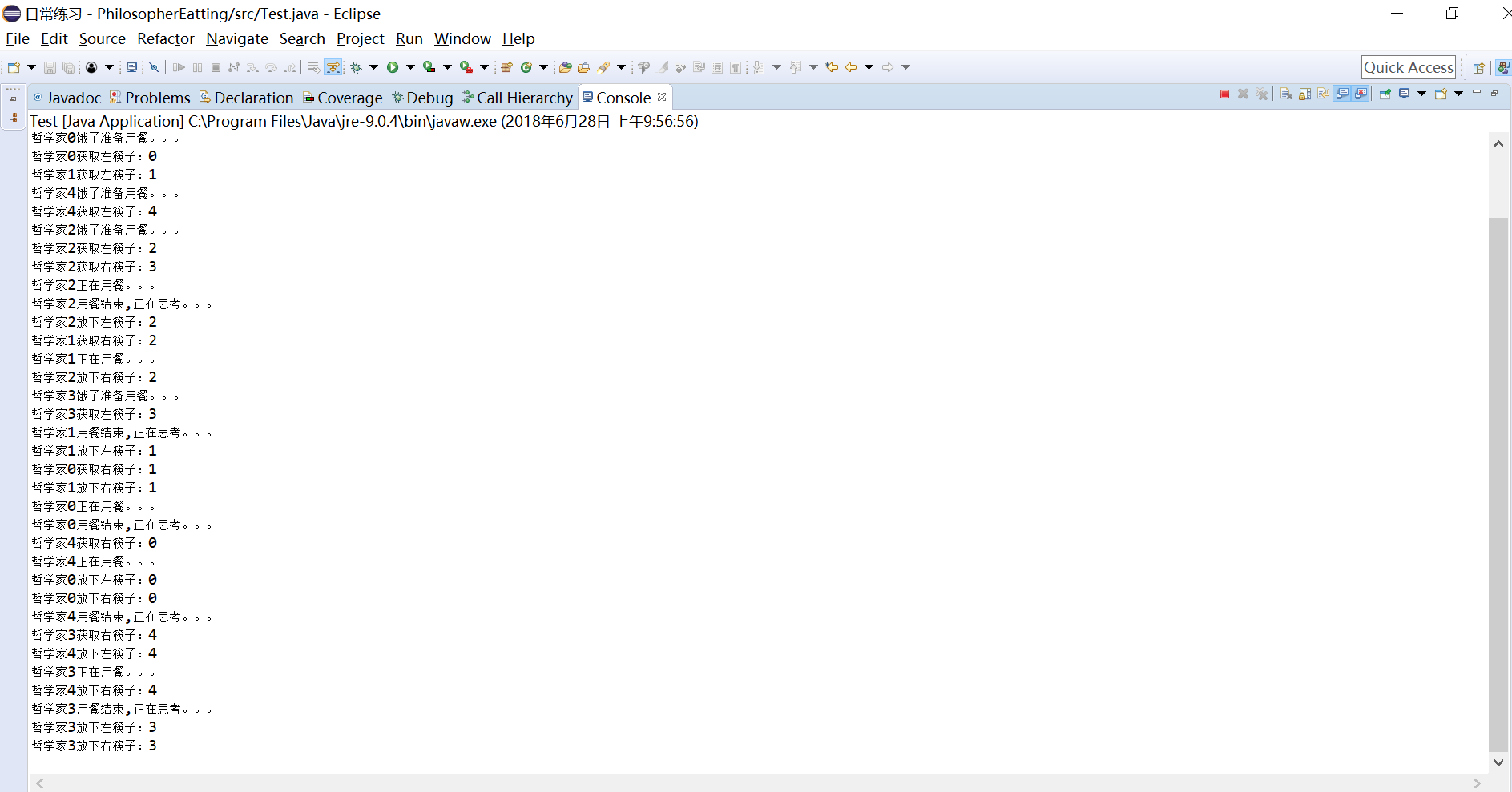
System.out.println("1: 结束");

}

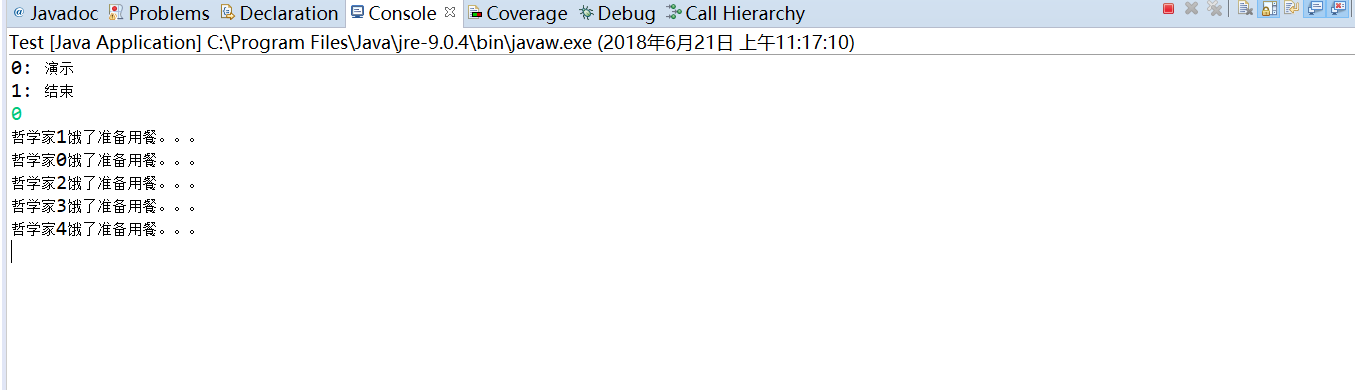
}

**五丶测试数据及其结果分析**

1. **正常运行截图：**



1. **死锁问题截图：**



**六丶调试过程中的问题**

开始的时候，无法模拟出死锁的问题，不管有没有限制线程数，都可以运行结束，找了一下原因可能是每个进程执行的时间太快，并没有实际的并发，后来添加了Thread.sleep(1000 \* 1)来观察死锁出现的情况，解决了这一问题。

调试界面的时候，菜单总是和每一条运行结果随机出现，我觉得是线程同步的原因，后来在显示菜单前加了一个等待输入的命令，这样就会在显示完结果之后才能选择下一步做什么。

**七、参考文献和查阅的资料**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文档名称 | 版本号 | 日期 | 出版单位/来源 | 备注 |
| 操作系统教程 | 第一版 | 2009年9月 | 人民邮电出版社 |  |

**八、课程设计总结**

（一）遇到的难点及解决办法

1. 拿到题目的时候看到涉及到系统调用，一时没有思路，后来老师说用虚 拟机，我就想到了用java来做，java有自带的JVM虚拟机，而且开发也很方便，很省事。

2.在最开始的时候没想好怎么设计，本来想放在一个主函数里面的就行，后来想想还是面向对象来设计实现，于是设计了筷子类，哲学家类来模拟就餐问题，这样也符合java面向对象编程。

（二）心得与体会

通过本次的程序设计，我有了很大的收获，之前学习操作系统这门课程的时候，觉得里面的有些东西好像不是很实用，没行到这么快就用上了。拿到课题之后，从开始的设计，到编码，在到调试运行，真正的将理论与实践结合起来。也真正理解到了动手实践的重要性。

由于时间仓促，设计难免还有一些不成熟和不完善的地方，但通过本次课题设计，使我对这进程间的同步互斥有了进一步的了解，也的确体会到了前所未有的充实感和自豪感，也对自己的学习和实践能力有了充分的自信。

最后，感谢段老师这两周以来的帮助，没有老师的指导，我的进展不会那么顺利，再次表示感谢。