华 南 农 业 大 学 数 学 与 信 息（软 件） 学 院

《操作系统课程实验》成绩单

开设时间：2023 学年第二学期

<table>
<tr><td>班级</td><td></td><td>学号</td><td></td><td>姓名</td><td></td></tr>
<tr><td>题目</td><td colspan="5">题目三 进程同步与互斥</td></tr>
<tr><td>自我评价</td><td colspan="5">通过本次实验，我加深了对进程同步和死锁问题的理解。哲学家就餐问题生动形象地展示了进程同步和死锁的产生原因及危害。每个哲学家代表一个进程，筷子代表所需资源。当多个进程互相占有对方所需的资源时，就会导致死锁发生。本实验通过模拟不同的解决策略，让我更好地掌握了预防和避免死锁的方法。
本实验需要模拟多个哲学家同时就餐的场景，因此需要用到多线程编程。通过实践，我掌握了如何创建多线程、线程同步的方法以及如何控制线程的执行。同时也体会到了多线程编程的复杂性，比如需要注意线程安全等问题。
总的来说，通过这次实验，我不仅巩固和加深了对操作系统相关知识的理解，还培养和提高了编程、调试和解决复杂问题的能力，获益匪浅。对于将来的学习和工作，这些经验都将是宝贵的财富。</td></tr>
<tr><td>教师评语</td><td colspan="5">评价指标:
● 题目内容和要求完成情况 优 □ 良 □ 中 □ 差 □
● 对算法原理的理解程度 优 □ 良 □ 中 □ 差 □
● 程序设计水平 优 □ 良 □ 中 □ 差 □
● 程序运行效果及正确性 优 □ 良 □ 中 □ 差 □
● 课程设计报告结构清晰 优 □ 良 □ 中 □ 差 □
● 报告中总结和分析详尽 优 □ 良 □ 中 □ 差 □</td></tr>
<tr><td>成绩</td><td colspan="2"></td><td>教师签名</td><td colspan="2"></td></tr>
</table>

题目三 进程同步与互斥

目录

一、 需求分析

本项目目的是模拟哲学家进餐问题，通过设计不同的方案，解决哲学家进餐死锁问题。本系统分成以下模块。

1. 选择栏：用户可以通过选择栏进行哲学家问题模拟初始化、策略切换、软件退出；

2. 模拟主界面：以图像、文字、终端等形式输出哲学家的活动过程，可视化哲学家的思考、等待和吃饭的状态。

3. 操作面板：包括开始、暂停、速度设置以及音乐按钮。

4. 解决方法：

表 1 解决策略及描述

策略名	策略描述
默认策略	这是最基本的策略，哲学家首先随机获取左侧或右侧的筷子。如果两个筷子都获取成功，哲学家就开始吃饭。
最多四人持筷子	该策略在策略 1 的基础上添加了一个条件，只有当筷子的使用者少于四个时，哲学家才尝试获取筷子。
按照奇偶 ID 拾取筷子	该策略根据哲学家的 ID 来决定获取筷子的顺序。如果哲学家的 ID 是奇数，那么他首先获取左边的筷子，然后获取右边的筷子。如果哲学家的 ID 是偶数，那么他首先获取右边的筷子，然后获取左边的筷子。
同时拾取 2 根筷子	该策略是在策略 1 的基础上，将获取筷子的操作放在一个同步块中，即同时获取左右两边的筷子。这样可以保证哲学家在获取筷子时不会被其他哲学家打断。

二、 概要设计

本系统基于 Java 开发，使用了 Java 下的 Thread 库、Swing 库，以每一个独立的线程来达标一个哲学家。用随机数 `(Math.random() * sleepTime)` 模拟哲学家思考时间和用餐时间。

1. 类图展示

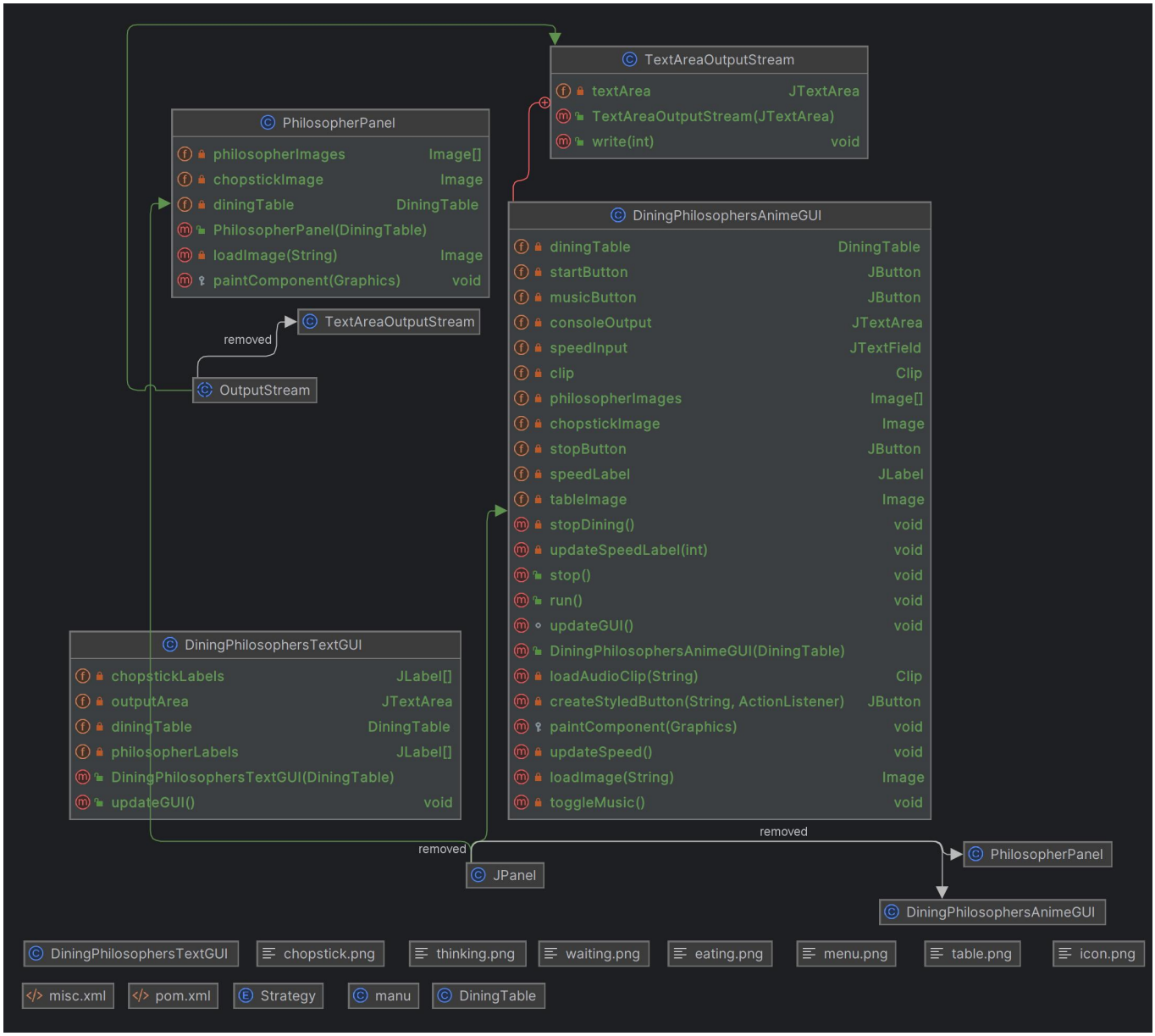


图 1 GUI 类图

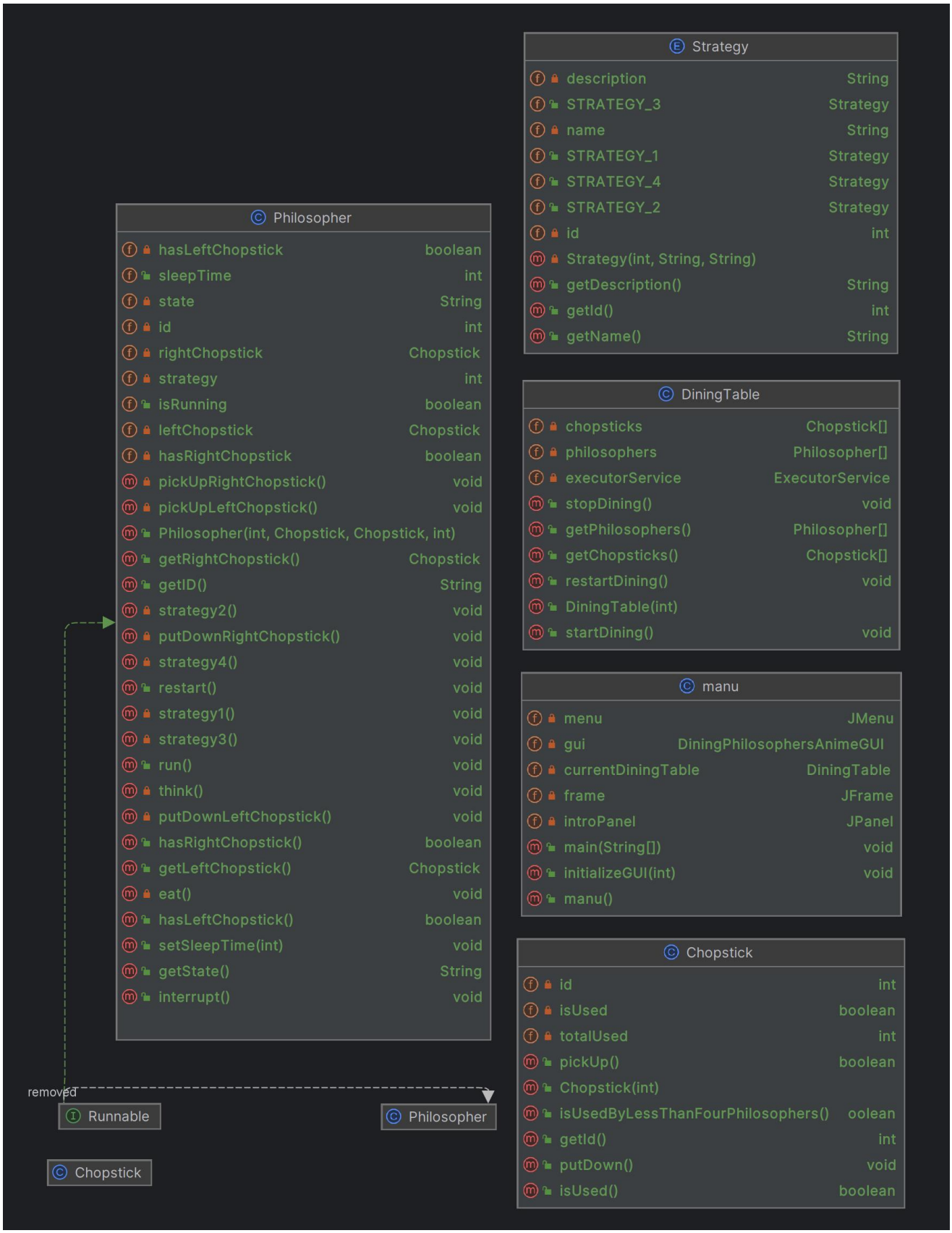

Strategy
description String
STRATEGY_3 Strategy
name String
STRATEGY_1 Strategy
STRATEGY_4 Strategy
STRATEGY_2 Strategy
id int
Strategy(int, String, String)
getDescription() String
getId() int
getName() String
Philosopher
hasLeftChopstick boolean
sleepTime int
state String
id int
rightChopstick Chopstick
strategy int
isRunning boolean
leftChopstick Chopstick
hasRightChopstick boolean
pickUpRightChopstick() void
pickUpLeftChopstick() void
Philosopher(int, Chopstick, Chopstick, int)
getRightChopstick() Chopstick
getID() String
strategy2() void
putDownRightChopstick() void
strategy4() void
restart() void
strategy1() void
strategy3() void
run() void
think() void
putDownLeftChopstick() void
hasRightChopstick() boolean
getLeftChopstick() Chopstick
eat() void
hasLeftChopstick() boolean
setSleepTime(int) void
getState() String
interrupt() void
DiningTable
chopsticks Chopstick[]
philosophers Philosopher[]
executorService ExecutorService
stopDining() void
getPhilosophers() Philosopher[]
getChopsticks() Chopstick[]
restartDining() void
DiningTable(int)
startDining() void
manu
menu JMenu
gui DiningPhilosophersAnimeGUI
currentDiningTable DiningTable
frame JFrame
introPanel JPanel
main(String[]) void
initializeGUI(int) void
manu()
Chopstick
id int
isUsed boolean
totalUsed int
pickUp() boolean
Chopstick(int)
isUsedByLessThanFourPhilosophers() oolean
getId() int
putDown() void
isUsed() boolean
removed
Runnable
Philosopher
Chopstick

图 2 哲学家进餐类图

2. 主程序流程图

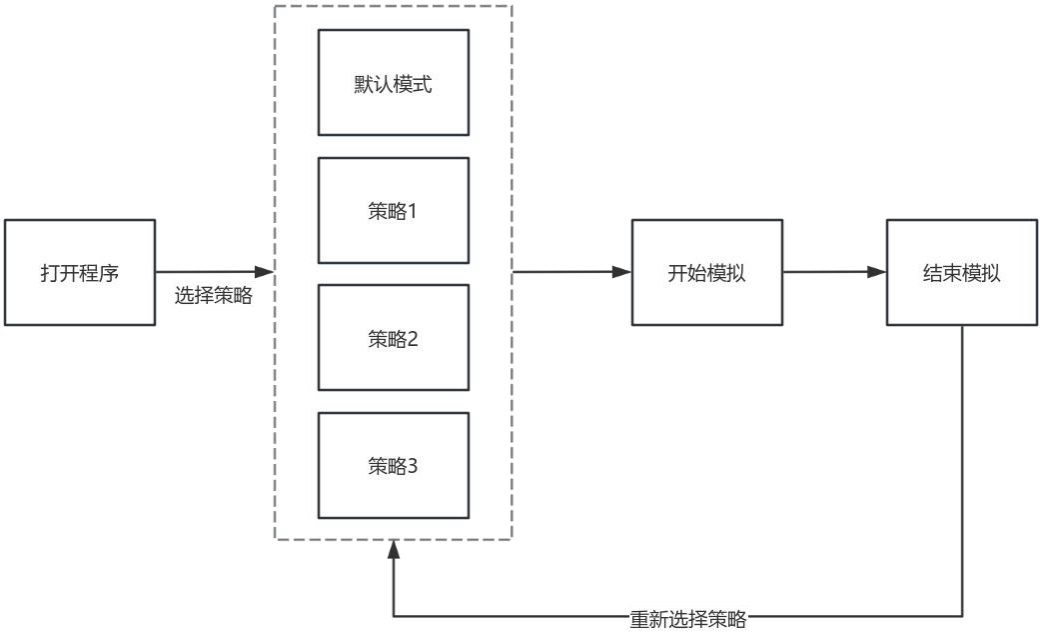


图 3 主程序流程图

四种策略具体详见表 1

三、 详细设计

1. Chopstick 类伪代码

```
class Chopstick:
    state: String  // 筷子的状态

    method pickUp():  // 拿起筷子的方法
        set state to "in use"  // 将状态设为“使用中”

    method putDown():  // 放下筷子的方法
        set state to "available"  // 将状态设为“可用”
```

2. Philosopher 类伪代码

```
class Philosopher:
    id: int  // 哲学家的编号
    leftChopstick: Chopstick  // 左边的筷子
    rightChopstick: Chopstick  // 右边的筷子
    diningTable: DiningTable  // 餐桌

    method run():  // 哲学家线程运行方法
        while not diningTable.isStopped:  // 如果餐桌没有停止
            think()  // 哲学家思考
            tryToEat()  // 尝试进餐
```

```
    method think():  // 思考的方法
        sleep for a random time  // 随机思考一段时间

    method tryToEat():  // 尝试进餐的方法
        if diningTable.tryToPickChopsticks(id):  // 尝试拿筷子
            eat()  // 开始进餐
            diningTable.putDownChopsticks(id)  // 放下筷子

    method eat():
        sleep for a random time  // 随机进餐一段时间
```

3. DiningTable 类伪代码

```
class DiningTable:
    chopsticks: list of Chopstick  // 筷子列表
    philosophers: list of Philosopher  // 哲学家列表
    strategy: int  // 选择的策略
    isStopped: boolean  // 餐桌是否停止

    constructor DiningTable(strategy: int):  // 餐桌的构造方法
        initialize chopsticks  // 初始化筷子
        initialize philosophers  // 初始化哲学家
        set strategy  // 设置策略

    method startDining():  // 开始进餐的方法
        start philosopher threads  // 启动哲学家线程

    method stopDining():  // 停止进餐的方法
        set isStopped to true  // 设置餐桌为停止状态
        stop philosopher threads  // 停止哲学家线程

    method tryToPickChopsticks(philosopherId: int) -> boolean:  // 尝试拿
筷子的方法
        if strategy == 1:
            return tryBasicStrategy(philosopherId)  // 使用策略 1
        if strategy == 2:
            return tryLimitedStrategy(philosopherId)  // 使用策略 2
        if strategy == 3:
            return tryOddEvenStrategy(philosopherId)  // 使用策略 3
        if strategy == 4:
            return trySynchronizedStrategy(philosopherId)  // 使用策略 4

    method putDownChopsticks(philosopherId: int):  // 放下筷子的方法
        release chopsticks  // 释放筷子
```

4. DiningPhilosophersAnimeGUI 类伪代码

```
class DiningPhilosophersAnimeGUI:
    diningTable: DiningTable  // 餐桌

    constructor DiningPhilosophersAnimeGUI(diningTable: DiningTable):  // GUI 的构造方法
        initialize GUI components  // 初始化 GUI 组件

    method run():  // 运行 GUI 的方法
        while not diningTable.isStopped:  // 如果餐桌没有停止
            updateGUI()  // 更新 GUI

    method updateGUI():  // 更新 GUI 的方法
        update philosopher and chopstick states in GUI  // 更新哲学家和筷子的状态

    method stop():  // 停止 GUI 的方法
        stop GUI updates  // 停止 GUI 更新
```

5. menu 类伪代码

```
class manu:
    frame: JFrame  // 主窗口
    gui: DiningPhilosophersAnimeGUI  // GUI
    currentDiningTable: DiningTable  // 当前餐桌
    menu: JMenu  // 菜单
    introPanel: JPanel  // 介绍面板

    constructor manu():  // 构造方法
        create JFrame  // 创建主窗口
        set default close operation  // 设置默认关闭操作
        set size and location  // 设置窗口大小和位置
        load and set icon  // 加载并设置图标
        create and setup introPanel  // 创建并设置介绍面板
        create and add introLabel to introPanel  // 创建并添加介绍标签
        add introPanel to frame  // 将介绍面板添加到窗口
        create JMenuBar  // 创建菜单栏
        create JMenu and add to JMenuBar  // 创建菜单并添加到菜单栏
        create exit menu item and add action listener  // 创建退出菜单项并添加事件监听
        add exit menu item to JMenuBar  // 将退出菜单项添加到菜单栏
        set JMenuBar to frame  // 设置菜单栏到窗口
        create menu items for each strategy  // 为每个策略创建菜单项
        add action listeners to menu items  // 为菜单项添加事件监听
        make frame visible  // 使窗口可见

    method initializeGUI(strategy: int):  // 初始化 GUI 的方法
```

```
        if gui is not null:
            gui.stop()  // 停止当前 GUI 及其相关线程
            remove gui from frame  // 从窗口移除 GUI

        if currentDiningTable is not null:
            currentDiningTable.stopDining()  // 停止所有哲学家线程

        create new DiningTable with strategy  // 创建新的餐桌
        create new DiningPhilosophersAnimeGUI with diningTable  // 创建新
的 GUI
        add gui to frame  // 将 GUI 添加到窗口
        start GUI update thread  // 启动 GUI 更新线程
```

四、 调试分析

1. 设计过程的经验和体会

本次实验选择了哲学家就餐问题，这个问题的算法难度并不高，但模拟 5 个哲学家的同时运行是一个重要的问题。

在研究多线程的实现方法时，我发现只需要在需要使用多线程的类上实现 Runnable 接口，并在 run()方法中实现多线程调用即可。我定义了一个 Chopstick 类，该类表示筷子（即资源），里面有一个 isUsed 布尔变量，用作筷子是否已被使用的信号量。

界面设计是本实验中最困难的部分，需要考虑控件布局，并找到适当的图片来表示哲学家的不同状态。在实现过程中，我逐步学习并应用了多线程的使用，巩固了课本中关于死锁的知识，并熟悉了 Swing 库的界面设计。

2. 调试过程中遇到的问题及其解决方法

问题一：观测不到死锁状态

在设计运行程序的过程中，我发现死锁的发生频率非常低。很难观测到死锁状态的发生，这可能是因为每一个哲学家都遵循固定的获取资源的顺序，即先尝试获取左侧筷子、再尝试获取右侧筷子的资源获取策略。同时每个线程获取资源

的时间也是固定。

为了解决上述的问题，我做了以下三处修改，①简化锁顺序的设计，通过随机获取左右侧筷子，从而增加死锁发生的概率。②引入随机性，使得线程获取资源的顺序和时间不确定，从而增加死锁发生的概率。例如，让线程在获取每个资源之前随机等待一段时间。③引入了 sleepTime 参数，使得可以修改线程的随机等待时间范围(0, sleepTime]

即便如此，也很难观测到死锁状态的发生，程序在后台挂起一段时间后，控制台输出了 55780 次信息才产生死锁状态。

```
55769 Philosopher 4 is Eating
55770 Philosopher 4 is Waiting for putting down right chopstick with id 0
55771 Philosopher 4 is Waiting for putting down left chopstick with id 4
55772 Philosopher 4 is Thinking
55773 Philosopher 4 is Waiting for picking up right chopstick with id 0
55774 Philosopher 1 is Waiting for putting down right chopstick with id 2
55775 Philosopher 1 is Waiting for putting down left chopstick with id 1
55776 Philosopher 1 is Thinking
55777 Philosopher 1 is Waiting for picking up right chopstick with id 2
55778 Philosopher 2 is Waiting for picking up right chopstick with id 3
55779 Philosopher 3 is Waiting for picking up right chopstick with id 4
55780 Philosopher 0 is Waiting for picking up right chopstick with id 1
55781
```

图 4 日志记录信息

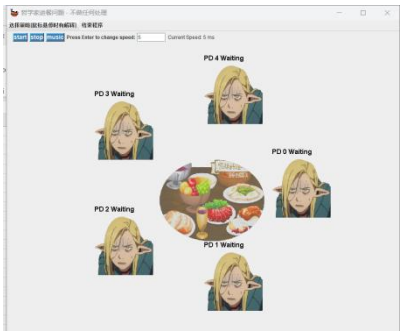


图 5 模拟出现死锁[1]

问题二：程序结束问题

在设计程序的第一阶段，由于启用线程是用循环与 run 方法实现的，所以我仍然沿用了循环与 interrupt 方法。

```
public void interrupt() throws InterruptedException {
    if (hasLeftChopstick) {
        putDownLeftChopstick();
    }
    if (hasRightChopstick) {
        putDownRightChopstick();
    }
    state = "Thinking";
    Thread.currentThread().interrupt();
}
```

[1] 当程序陷入死锁状态时，chopstick 有概率会消失，但可以通过 Log 日志获取 chopstick 此时的具体状态

在程序设计的后续阶段，用户可以通过控制 sleepTime 以及后台挂起程序的方式，观测到死锁的发生。然而，我发现在死锁发生后，使用 stop 方法并不能结束线程，这是因为 interrupt 方法本身并不能直接关闭死锁，它是以向目标线程发送一个中断信号，通知该线程被中断实现关闭操作。由于在按下 stop 方法后，我观测到每个哲学家并没有执行放下筷子的操作。因此确定 Thread.currentThread().interrupt()方法本身没有被调用。

综上所述，我采用了 executorService.shutdownNow()的方法，该方法可以尝试关闭执行器服务，并停止当前执行的任务。如果某个任务因为死锁或其他原因而无法响应中断，shutdownNow() 方法会尝试通过中断线程来停止这些任务。因此，它在某种程度上可以帮助解决由于死锁导致的执行器服务无法正常关闭的问题。

五、 使用说明

打开程序，在左上角的策略模拟选项选择策略。当鼠标悬停在选项时，会有

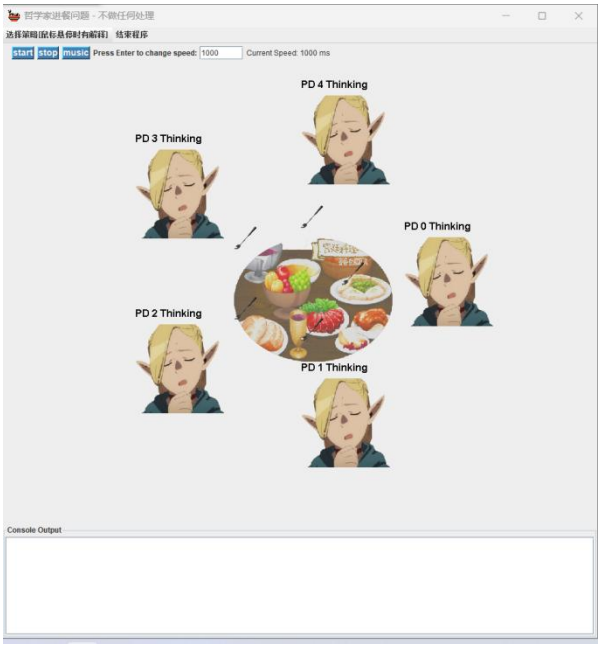
图 6 程序操作说明

各个策略的使用说明。有四种策略可以选择，四种策略详见表 1。点击左上角的开始按钮进行模拟。点击终止按钮，结束程序。在重新选择策略之前，需要结束模拟。当控制台不输出新的信息时，报名当前模拟操作已结束。

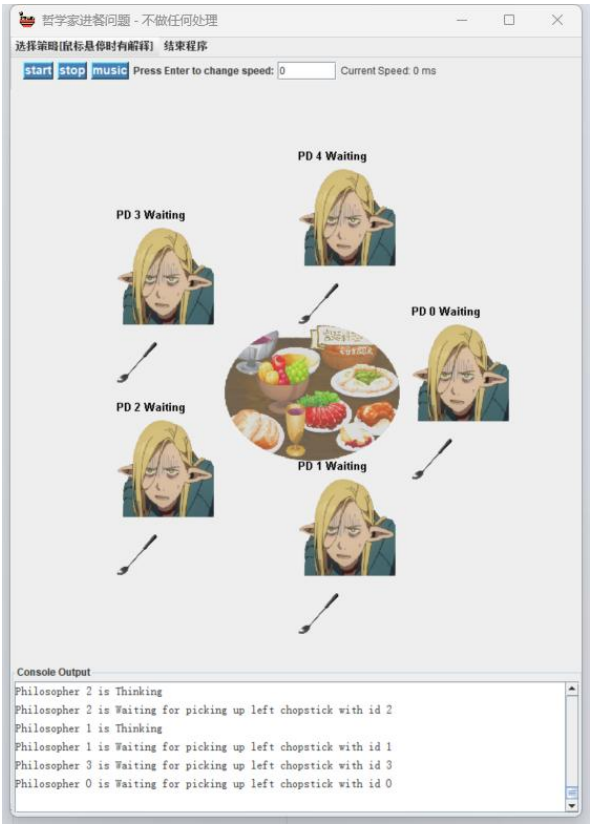
六、 测试与运行结果

1. 初始界面

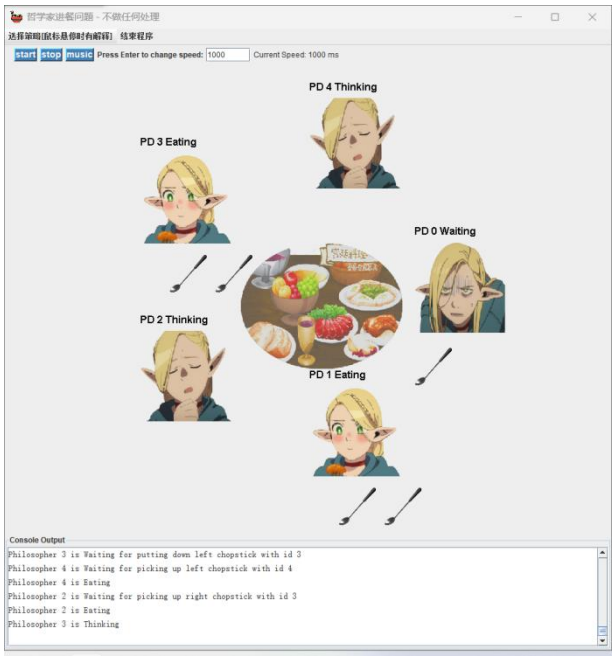
2. 开始运行界面（默认方案）



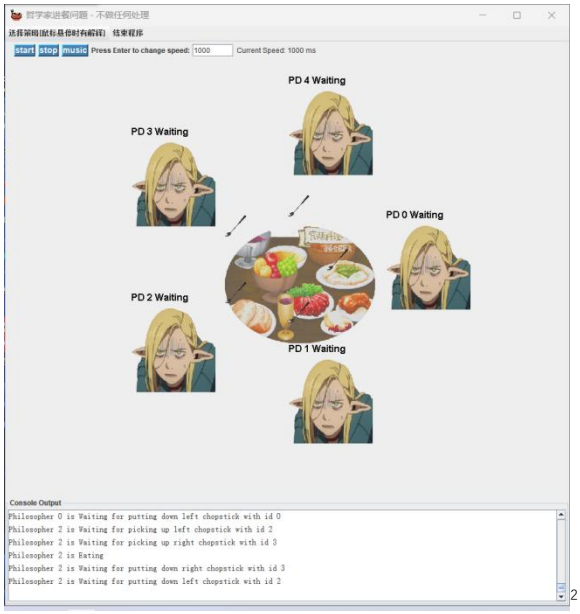
3. 死锁状态（默认方案）



4. 运行状态



5. 结束状态



[2]

[2] 与死锁状态不同，chopstick 变大表示该资源被使用，线程正常终止时的 chopstick 为正常大小。

七、 附录

```java
package main.java;

public class Chopstick {
    private final int id;
    private boolean isUsed;
    private static int totalUsed = 0; // 全局计数器

    public Chopstick(int id) {
        this.id = id;
        this.isUsed = false;
    }

    public synchronized boolean pickUp() {
        while (isUsed) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        isUsed = true;
        totalUsed++;
        return true;
    }

    public synchronized void putDown() {
        isUsed = false;
        totalUsed--;
        notifyAll();
    }

    public int getId() {
        return id;
    }

    public boolean isUsed() {
        return isUsed;
    }

    public static synchronized boolean isUsedByLessThanFourPhilosophers() {
        return totalUsed < 4;
    }
}
```

```java
package main.java;

import javax.sound.sampled.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
import java.io.*;
import java.net.URL;

public class DiningPhilosophersAnimeGUI extends JPanel {
    private final DiningTable diningTable;
    private final Image[] philosopherImages;
    private final Image chopstickImage;
    private final Image tableImage;

    private final JButton startButton;
    private final JButton stopButton;
    private final JButton musicButton;
    private final JTextField speedInput;
    private final JTextArea consoleOutput;
    private final JLabel speedLabel;

    private Clip clip;

    public DiningPhilosophersAnimeGUI(DiningTable diningTable) throws
FileNotFoundException {
        this.diningTable = diningTable;
        this.philosopherImages = new Image[3];

        // Load images
        this.chopstickImage = loadImage("/chopstick.png");
        this.tableImage = loadImage("/table.png");
        this.philosopherImages[0] = loadImage("/thinking.png");
        this.philosopherImages[1] = loadImage("/eating.png");
        this.philosopherImages[2] = loadImage("/waiting.png");

        this.speedInput = new JTextField("1000", 6);
        this.speedInput.addActionListener(e -> updateSpeed());

        this.speedLabel = new JLabel();
        updateSpeedLabel(1000);

        this.startButton = createStyledButton("start", e ->
```

```
diningTable.startDining());
        this.stopButton = createStyledButton("stop", e -> stopDining());
        this.musicButton = createStyledButton("music", e -> toggleMusic());

        this.clip = loadAudioClip("Sleep Walking Orchestra - BUMP OF C.wav");

        this.consoleOutput = new JTextArea(7, 30);
        this.consoleOutput.setEditable(false);
        this.consoleOutput.setFont(new Font("Monospaced", Font.PLAIN, 14));
        JScrollPane scrollPane = new JScrollPane(consoleOutput);

        JPanel consolePanel = new JPanel(new BorderLayout());
        consolePanel.setBorder(BorderFactory.createTitledBorder("Console Output"));
        consolePanel.add(scrollPane, BorderLayout.CENTER);

        this.setLayout(new BorderLayout());
        JPanel controlPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        controlPanel.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));
        controlPanel.add(startButton);
        controlPanel.add(stopButton);
        controlPanel.add(musicButton);
        controlPanel.add(new JLabel("Press Enter to change speed:"));
        controlPanel.add(speedInput);
        controlPanel.add(speedLabel);

        this.add(controlPanel, BorderLayout.NORTH);
        this.add(consolePanel, BorderLayout.SOUTH);

        System.setOut(new PrintStream(new TextAreaOutputStream(consoleOutput)));
        System.setErr(new PrintStream(new TextAreaOutputStream(consoleOutput)));
    }

    private JButton createStyledButton(String text, ActionListener actionListener) {
        JButton button = new JButton(text);
        button.addActionListener(actionListener);
        button.setFont(new Font("Arial", Font.BOLD, 14));
        button.setBackground(new Color(70, 130, 180));
        button.setForeground(Color.WHITE);
        button.setFocusPainted(false);
        button.setBorder(BorderFactory.createRaisedBevelBorder());
        return button;
    }

    private void updateSpeed() {
```

```
        try {
            int speed = Integer.parseInt(speedInput.getText());
            Philosopher.setSleepTime(speed);
            updateSpeedLabel(speed);
            diningTable.restartDining();
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(this, "Please enter a valid number",
"Invalid Input", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void updateSpeedLabel(int speed) {
        speedLabel.setText("Current Speed: " + speed + " ms");
        speedLabel.setFont(new Font("Arial", Font.PLAIN, 12));
    }

    private void stopDining() {
        Philosopher.isRunning = false;
        try {
            diningTable.stopDining();
        } catch (InterruptedException ex) {
            throw new RuntimeException(ex);
        }
    }

    private void toggleMusic() {
        if (clip != null) {
            if (clip.isRunning()) {
                clip.stop();
            } else {
                clip.setFramePosition(0);
                clip.start();
            }
        }
    }

    private Clip loadAudioClip(String path) {
        try {
            URL url = getClass().getClassLoader().getResource(path);
            if (url == null) {
                System.err.println("File not found: " + path);
                return null;
            }
            AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
```

```
            Clip clip = AudioSystem.getClip();
            clip.open(audioIn);
            return clip;
        } catch (UnsupportedAudioFileException | IOException |
LineUnavailableException e) {
            e.printStackTrace();
            return null;
        }
    }

    private Image loadImage(String path) {
        URL url = getClass().getResource(path);
        if (url == null) {
            System.err.println("File not found: " + path);
            return null;
        }
        return new ImageIcon(url).getImage();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g.create();

        // 设计时的宽度和高度
        int originalWidth = 700;
        int originalHeight = 800;

        // 当前面板的宽度和高度
        int currentWidth = getWidth();
        int currentHeight = getHeight();

        // 计算宽度和高度的比例
        double widthRatio = (double) currentWidth / originalWidth;
        double heightRatio = (double) currentHeight / originalHeight;
        double scale = Math.min(widthRatio, heightRatio);

        int centerX = currentWidth / 2;
        int centerY = currentHeight / 2 - (int) (80 * scale);
        int radius = (int) (200 * scale);

        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
0.75f));
        g2d.drawImage(tableImage, centerX - (int) (90 * scale), centerY - (int) (60 *
```

```
scale), (int) (220 * scale), (int) (170 * scale), this);
        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
1.0f));

        for (int i = 0; i < 5; i++) {
            double angle = 2 * Math.PI / 5 * i;
            int x = (int) (centerX + radius * Math.cos(angle));
            int y = (int) (centerY + radius * Math.sin(angle));

            Philosopher philosopher = diningTable.getPhilosophers()[i];
            String state = philosopher.getState();

            Image philosopherImage = philosopherImages[0];
            if ("Eating".equals(state)) {
                philosopherImage = philosopherImages[1];
            } else if ("Waiting".equals(state)) {
                philosopherImage = philosopherImages[2];
            }
            g2d.drawImage(philosopherImage, x - (int) (60 * scale), y - (int) (60 *
scale), (int) (120 * scale), (int) (120 * scale), this);

            g2d.setColor(Color.BLACK);
            g2d.setFont(new Font("Arial", Font.BOLD, (int) (14 * scale)));
            g2d.drawString("PD " + philosopher.getID() + " " + state, x - (int) (60 *
scale), y - (int) (70 * scale));

            int philosopherLeft = x - (int) (50 * scale);
            int philosopherRight = x + (int) (50 * scale);

            Chopstick chopstick = diningTable.getChopsticks()[i];
            int chopstickX = (int) (centerX + (radius - (120 * scale)) *
Math.cos(angle));
            int chopstickY = (int) (centerY + (radius - (120 * scale)) *
Math.sin(angle));

            if (chopstick.isUsed()) {
                g2d.setColor(Color.WHITE);
                if (philosopher.hasLeftChopstick() &&
philosopher.hasRightChopstick()) {
                    g2d.drawImage(chopstickImage, philosopherLeft + (int) (20 *
scale), y + (int) (70 * scale), (int) (65 * scale), (int) (65 * scale), this);
                    g2d.drawImage(chopstickImage, philosopherRight - (int) (20 *
scale), y + (int) (70 * scale), (int) (65 * scale), (int) (65 * scale), this);
                } else if (philosopher.hasLeftChopstick() && chopstick ==
```

```
philosopher.getLeftChopstick()) {
                    g2d.drawImage(chopstickImage, philosopherLeft - (int) (20 *
scale), y + (int) (75 * scale), (int) (65 * scale), (int) (65 * scale), this);
                } else if (philosopher.hasRightChopstick() && chopstick ==
philosopher.getRightChopstick()) {
                    g2d.drawImage(chopstickImage, philosopherRight + (int) (20 *
scale), y + (int) (75 * scale), (int) (65 * scale), (int) (65 * scale), this);
                }
            } else {
                g2d.setColor(Color.BLACK);
                g2d.drawImage(chopstickImage, chopstickX - (int) (30 * scale),
chopstickY - (int) (30 * scale), (int) (40 * scale), (int) (40 * scale), this);
            }
        }
        g2d.dispose();
    }


    public void run() {
        // Main loop
        while (true) {
            updateGUI();
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }

    void updateGUI() {
        repaint();
    }

    public void stop() {
        if (clip != null) {
            clip.stop();
            clip.close();
        }
    }

    class TextAreaOutputStream extends OutputStream {
        private final JTextArea textArea;
```

```
        public TextAreaOutputStream(JTextArea textArea) {
            this.textArea = textArea;
        }

        @Override
        public void write(int b) {
            textArea.append(String.valueOf((char) b));
            textArea.setCaretPosition(textArea.getDocument().getLength());
        }
    }
}
```

```
package main.java;

public class Philosopher implements Runnable {
    private final int id;
    private final Chopstick leftChopstick;
    private final Chopstick rightChopstick;
    private String state;
    private final int strategy;
    private boolean hasLeftChopstick = false;
    private boolean hasRightChopstick = false;
    public static int sleepTime = 1000;
    public static boolean isRunning = true; // Add this line

    public Philosopher(int id, Chopstick leftChopstick, Chopstick rightChopstick, int
strategy) {
        this.id = id;
        this.leftChopstick = leftChopstick;
        this.rightChopstick = rightChopstick;
        this.state = "Thinking";
        this.strategy = strategy;
    }

    public static void setSleepTime(int speed) {
        sleepTime = speed;
    }


    @Override
    public void run() {
        try {
            isRunning = true;
```

```
            while (!Thread.currentThread().isInterrupted() && isRunning) { // Check
isRunning
                think();
                switch (strategy) {
                    case 1:
                        strategy1();
                        break;
                    case 2:
                        strategy2();
                        break;
                    case 3:
                        strategy3();
                        break;
                    case 4:
                        strategy4();
                        break;
                }
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    private void think() throws InterruptedException {
        state = "Thinking";
        System.out.println("Philosopher " + id + " is " + state);
        logger.logMessage("Philosopher " + id + " is " + state);

        Thread.sleep((long) (Math.random() * sleepTime));
    }

    private void eat() throws InterruptedException {
        if (hasLeftChopstick && hasRightChopstick) {
            state = "Eating";
            System.out.println("Philosopher " + id + " is " + state);
            logger.logMessage("Philosopher " + id + " is " + state);

            Thread.sleep((long) (Math.random() * sleepTime));
        }
    }

    private void pickUpLeftChopstick() throws InterruptedException {
        state = "Waiting";
        System.out.println("Philosopher " + id + " is " + state + " for picking up left
```

```
chopstick with id " + leftChopstick.getId());
        logger.logMessage("Philosopher " + id + " is " + state + " for picking up left
chopstick with id " + leftChopstick.getId());

        if (leftChopstick.pickUp()) {
            hasLeftChopstick = true;
            state = "Waiting";
        }
    }

    private void pickUpRightChopstick() throws InterruptedException {
        state = "Waiting";
        System.out.println("Philosopher " + id + " is " + state + " for picking up right
chopstick with id " + rightChopstick.getId());
        logger.logMessage("Philosopher " + id + " is " + state + " for picking up right
chopstick with id " + rightChopstick.getId());

        if (rightChopstick.pickUp()) {
            hasRightChopstick = true;
            state = "Waiting";
        }
    }

    private void putDownRightChopstick() throws InterruptedException {
        rightChopstick.putDown();
        hasRightChopstick = false;
        state = "Waiting";
        System.out.println("Philosopher " + id + " is " + state + " for putting down
right chopstick with id " + rightChopstick.getId());
        logger.logMessage("Philosopher " + id + " is " + state + " for putting down right
chopstick with id " + rightChopstick.getId());
    }

    private void putDownLeftChopstick() throws InterruptedException {
        leftChopstick.putDown();
        hasLeftChopstick = false;
        state = "Waiting";
        System.out.println("Philosopher " + id + " is " + state + " for putting down
left chopstick with id " + leftChopstick.getId());
        logger.logMessage("Philosopher " + id + " is " + state + " for putting down left
chopstick with id " + leftChopstick.getId());
    }
```

```
    private void strategy1() throws InterruptedException {
        if (Math.random() < 0.5) {
            synchronized (leftChopstick) {
                pickUpLeftChopstick();
                synchronized (rightChopstick) {
                    pickUpRightChopstick();
                    eat();
                    putDownRightChopstick();
                }
                putDownLeftChopstick();
            }
        } else {
            synchronized (rightChopstick) {
                pickUpRightChopstick();
                synchronized (leftChopstick) {
                    pickUpLeftChopstick();
                    eat();
                    putDownLeftChopstick();
                }
                putDownRightChopstick();
            }
        }
    }

    private void strategy2() throws InterruptedException {
        synchronized (rightChopstick) {
            if (Chopstick.isUsedByLessThanFourPhilosophers()) {
                pickUpRightChopstick();
                synchronized (leftChopstick) {
                    pickUpLeftChopstick();
                    eat();
                    putDownLeftChopstick();
                }
                putDownRightChopstick();
            }
        }
    }

    private void strategy3() throws InterruptedException {
        if (id % 2 == 1) {
            synchronized (leftChopstick) {
                pickUpLeftChopstick();
                synchronized (rightChopstick) {
                    pickUpRightChopstick();
```

```
                eat();
                putDownRightChopstick();
            }
            putDownLeftChopstick();
        }
    } else {
        synchronized (rightChopstick) {
            pickUpRightChopstick();
            synchronized (leftChopstick) {
                pickUpLeftChopstick();
                eat();
                putDownLeftChopstick();
            }
            putDownRightChopstick();
        }
    }
}

private void strategy4() throws InterruptedException {
    synchronized (leftChopstick) {
        synchronized (rightChopstick) {
            pickUpLeftChopstick();
            pickUpRightChopstick();
            eat();
            putDownRightChopstick();
            putDownLeftChopstick();
        }
    }
}

public void interrupt() throws InterruptedException {
    if (hasLeftChopstick) {
        putDownLeftChopstick();
    }
    if (hasRightChopstick) {
        putDownRightChopstick();
    }
    state = "Thinking";
    Thread.currentThread().interrupt();
}

public String getState() {
    return state;
}
```

```
    public boolean hasRightChopstick() {
        return hasRightChopstick;
    }

    public boolean hasLeftChopstick() {
        return hasLeftChopstick;
    }

    public Chopstick getLeftChopstick() {
        return leftChopstick;
    }

    public Chopstick getRightChopstick() {
        return rightChopstick;
    }

    public String getID() {
        return String.valueOf(id);
    }

    public void restart() {
        hasLeftChopstick = false;
        hasRightChopstick = false;
        state = "Thinking";
    }
}
```

```
package main.java;

import javax.swing.*;
import java.awt.*;
import java.io.FileNotFoundException;
import java.net.URL;

public class manu {
    private JFrame frame;
    private DiningPhilosophersAnimeGUI gui;
    private DiningTable currentDiningTable;
    private JMenu menu;
    private JPanel introPanel;

    public manu() {
```

```
        frame = new JFrame("哲学家进餐问题 - 介绍界面");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(700, 850);
        frame.setLocationRelativeTo(null);

        URL iconUrl = getClass().getResource("/icon.png");
        ImageIcon icon = new ImageIcon(iconUrl);
        frame.setIconImage(icon.getImage());
        JPanel introPanel = new JPanel();
        introPanel.setLayout(new BoxLayout(introPanel, BoxLayout.Y_AXIS));
        introPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

        URL menuUrl = getClass().getResource("/menu.png");
        ImageIcon imageIcon = new ImageIcon(menuUrl);
        // 设置图片大小
        imageIcon.setImage(imageIcon.getImage().getScaledInstance(300, 300,
Image.SCALE_DEFAULT));

        JLabel imageLabel = new JLabel(imageIcon);
        imageLabel.setAlignmentX(Component.CENTER_ALIGNMENT);

        introPanel.add(imageLabel);

        JLabel introLabel = new JLabel(
                "<html><body style='font-size:13px; font-family:Microsoft YaHei;
color: #333333;'>" +
                        "<div style='text-align: left;'>" +
                        "欢迎使用哲学家就餐模拟程序<br>" +
                        "功能：实现了延迟设置，多种策略的哲学家就餐问题模拟
<br><br><br>" +
                        "版本：1.2<br>" +
                        "开发次数：2 次<br>" +

                        "[ README ]<br>" +
                        "所有功能均已通过多轮测试<br>" +
                        "选中策略后，即可开始进行模拟<br>" +
                        "在切换策略时，请确保已经按下了停止按钮<br>" +
                        "将鼠标悬停在一些功能上会有提示<br>" +
                        "叉子变大表示被选中，叉子变小表示未被选中<br>" +
                        "sleepTime 决定了线程的读写持续时间范围 [0,
sleepTime]<br><br>" +
                        "感觉 Swing 库比 Fx 库要难多了....." +
                        "</div></body></html>");
        introLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
```

```
        introPanel.add(Box.createVerticalStrut(20)); // 增加空白
        introPanel.add(introLabel);

        frame.add(introPanel, BorderLayout.CENTER); // 更改为居中显示
        frame.setVisible(true);


        JMenuBar menuBar = new JMenuBar();
        menu = new JMenu("选择策略[鼠标悬停时有解释]");
        menuBar.add(menu);

        JMenuItem exitMenuItem = new JMenuItem("结束程序");
        exitMenuItem.addActionListener(e -> System.exit(0)); // 当菜单项被点击时，结束程序

        // 将菜单项添加到菜单栏
        menuBar.add(exitMenuItem);

        frame.setJMenuBar(menuBar);
        frame.setVisible(true);

        final boolean[] isFirstTime = {true};

        for (Strategy strategy : Strategy.values()) {
            JMenuItem menuItem = new JMenuItem(strategy.getName());
            menuItem.setToolTipText(strategy.getDescription());
            menuItem.addActionListener(e -> {
                try {
                    if (isFirstTime[0]) {
                        frame.remove(introPanel);
                        isFirstTime[0] = false;
                    }
                    initializeGUI(strategy.getId());
                    frame.setTitle("哲学家进餐问题 - " + strategy.getName());
                    frame.revalidate();
                    frame.repaint();
                } catch (FileNotFoundException ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(frame, "Error initializing GUI: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                } catch (InterruptedException ex) {
                    throw new RuntimeException(ex);
                }
```

```
            });
            menu.add(menuItem);
        }

        frame.setJMenuBar(menuBar);
        frame.setVisible(true);
    }

    public void initializeGUI(int strategy) throws FileNotFoundException,
InterruptedException {
        if (gui != null) {
            gui.stop();
            frame.remove(gui);
        }

        if (currentDiningTable != null) {
            currentDiningTable.stopDining();
        }

        currentDiningTable = new DiningTable(strategy);
        gui = new DiningPhilosophersAnimeGUI(currentDiningTable);
        frame.add(gui);

        logger.logMessage("Initializing GUI with strategy: " + strategy);
        System.out.println("Initializing GUI with strategy: " + strategy);

        new Thread(() -> {
            gui.run();
        }).start();
    }


    public static void main(String[] args) {
        logger.setupLogger();
        SwingUtilities.invokeLater(manu::new);
    }
}

enum Strategy {
    STRATEGY_1(1, "不做任何处理", "<html>策略 1：<br>这是最基本的策略，哲学家首先随
机获取左侧或右侧的筷子。<br>如果两个筷子都获取成功，哲学家就开始吃饭。</html>"),
    STRATEGY_2(2, "最多 4 人持筷子", "<html>策略 2：<br>这个策略在策略 1 的基础上添加
了一个条件，只有当筷子的使用者少于四个时，哲学家才尝试获取筷子。<br>这是为了避免死锁，
因为如果五个哲学家都持有一个筷子，那么他们都无法开始吃饭。</html>"),
```

```
    STRATEGY_3(3, "按号奇偶持筷", "<html>策略 3：<br>这个策略根据哲学家的 ID 来决定获取筷子的顺序。<br>如果哲学家的 ID 是奇数，那么他首先获取左边的筷子，然后获取右边的筷子。<br>如果哲学家的 ID 是偶数，那么他首先获取右边的筷子，然后获取左边的筷子。<br>这是为了避免死锁，因为至少有一个哲学家能够获取到两个筷子。</html>"),
    STRATEGY_4(4, "同时拿两支筷子", "<html>策略 4：<br>这个策略是在策略 1 的基础上，将获取筷子的操作放在一个同步块中。<br>这样可以保证哲学家在获取筷子时不会被其他哲学家打断。<br>这是为了避免死锁，因为哲学家在获取筷子时不会被其他哲学家打断，所以他总是能够获取到两个筷子。</html>");

    private final int id;
    private final String name;
    private final String description;

    Strategy(int id, String name, String description) {
        this.id = id;
        this.name = name;
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return description;
    }
}
```

```
package main.java;

public class Philosopher implements Runnable {
    private final int id;
    private final Chopstick leftChopstick;
    private final Chopstick rightChopstick;
    private String state;
    private final int strategy;
    private boolean hasLeftChopstick = false;
    private boolean hasRightChopstick = false;
    public static int sleepTime = 1000;
```

```
    public static boolean isRunning = true; // Add this line

    public Philosopher(int id, Chopstick leftChopstick, Chopstick rightChopstick, int
strategy) {
        this.id = id;
        this.leftChopstick = leftChopstick;
        this.rightChopstick = rightChopstick;
        this.state = "Thinking";
        this.strategy = strategy;
    }

    public static void setSleepTime(int speed) {
        sleepTime = speed;
    }


    @Override
    public void run() {
        try {
            isRunning = true;
            while (!Thread.currentThread().isInterrupted() && isRunning) { // Check
isRunning
                think();
                switch (strategy) {
                    case 1:
                        strategy1();
                        break;
                    case 2:
                        strategy2();
                        break;
                    case 3:
                        strategy3();
                        break;
                    case 4:
                        strategy4();
                        break;
                }
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    private void think() throws InterruptedException {
```

```
        state = "Thinking";
        System.out.println("Philosopher " + id + " is " + state);
        logger.logMessage("Philosopher " + id + " is " + state);

        Thread.sleep((long) (Math.random() * sleepTime));
    }

    private void eat() throws InterruptedException {
        if (hasLeftChopstick && hasRightChopstick) {
            state = "Eating";
            System.out.println("Philosopher " + id + " is " + state);
            logger.logMessage("Philosopher " + id + " is " + state);

            Thread.sleep((long) (Math.random() * sleepTime));
        }
    }

    private void pickUpLeftChopstick() throws InterruptedException {
        state = "Waiting";
        System.out.println("Philosopher " + id + " is " + state + " for picking up left
chopstick with id " + leftChopstick.getId());
        logger.logMessage("Philosopher " + id + " is " + state + " for picking up left
chopstick with id " + leftChopstick.getId());

        if (leftChopstick.pickUp()) {
            hasLeftChopstick = true;
            state = "Waiting";
        }
    }

    private void pickUpRightChopstick() throws InterruptedException {
        state = "Waiting";
        System.out.println("Philosopher " + id + " is " + state + " for picking up right
chopstick with id " + rightChopstick.getId());
        logger.logMessage("Philosopher " + id + " is " + state + " for picking up right
chopstick with id " + rightChopstick.getId());

        if (rightChopstick.pickUp()) {
            hasRightChopstick = true;
            state = "Waiting";
        }
    }

    private void putDownRightChopstick() throws InterruptedException {
```

```
        rightChopstick.putDown();
        hasRightChopstick = false;
        state = "Waiting";
        System.out.println("Philosopher " + id + " is " + state + " for putting down
right chopstick with id " + rightChopstick.getId());
        logger.logMessage("Philosopher " + id + " is " + state + " for putting down right
chopstick with id " + rightChopstick.getId());
    }

    private void putDownLeftChopstick() throws InterruptedException {
        leftChopstick.putDown();
        hasLeftChopstick = false;
        state = "Waiting";
        System.out.println("Philosopher " + id + " is " + state + " for putting down
left chopstick with id " + leftChopstick.getId());
        logger.logMessage("Philosopher " + id + " is " + state + " for putting down left
chopstick with id " + leftChopstick.getId());
    }


    private void strategy1() throws InterruptedException {
        if (Math.random() < 0.5) {
            synchronized (leftChopstick) {
                pickUpLeftChopstick();
                synchronized (rightChopstick) {
                    pickUpRightChopstick();
                    eat();
                    putDownRightChopstick();
                }
                putDownLeftChopstick();
            }
        } else {
            synchronized (rightChopstick) {
                pickUpRightChopstick();
                synchronized (leftChopstick) {
                    pickUpLeftChopstick();
                    eat();
                    putDownLeftChopstick();
                }
                putDownRightChopstick();
            }
        }
    }
```

```
    private void strategy2() throws InterruptedException {
        synchronized (rightChopstick) {
            if (Chopstick.isUsedByLessThanFourPhilosophers()) {
                pickUpRightChopstick();
                synchronized (leftChopstick) {
                    pickUpLeftChopstick();
                    eat();
                    putDownLeftChopstick();
                }
                putDownRightChopstick();
            }
        }
    }

    private void strategy3() throws InterruptedException {
        if (id % 2 == 1) {
            synchronized (leftChopstick) {
                pickUpLeftChopstick();
                synchronized (rightChopstick) {
                    pickUpRightChopstick();
                    eat();
                    putDownRightChopstick();
                }
                putDownLeftChopstick();
            }
        } else {
            synchronized (rightChopstick) {
                pickUpRightChopstick();
                synchronized (leftChopstick) {
                    pickUpLeftChopstick();
                    eat();
                    putDownLeftChopstick();
                }
                putDownRightChopstick();
            }
        }
    }

    private void strategy4() throws InterruptedException {
        synchronized (leftChopstick) {
            synchronized (rightChopstick) {
                pickUpLeftChopstick();
                pickUpRightChopstick();
                eat();
```

```
                putDownRightChopstick();
                putDownLeftChopstick();
            }
        }
    }

    public void interrupt() throws InterruptedException {
        if (hasLeftChopstick) {
            putDownLeftChopstick();
        }
        if (hasRightChopstick) {
            putDownRightChopstick();
        }
        state = "Thinking";
        Thread.currentThread().interrupt();
    }

    public String getState() {
        return state;
    }

    public boolean hasRightChopstick() {
        return hasRightChopstick;
    }

    public boolean hasLeftChopstick() {
        return hasLeftChopstick;
    }

    public Chopstick getLeftChopstick() {
        return leftChopstick;
    }

    public Chopstick getRightChopstick() {
        return rightChopstick;
    }

    public String getID() {
        return String.valueOf(id);
    }

    public void restart() {
        hasLeftChopstick = false;
        hasRightChopstick = false;
```

```
        state = "Thinking";
    }
}
```