**西安电子科技大学**

**Java程序设计 课程实验报告**

**实验名称 接口与实现**

经济与管理学院 信息管理与信息系统 班

成 绩

姓名 赵红玉 学号 21069100225

同作者

实验日期 2023 年 11 月 14 日

|  |
| --- |
| 指导教师评语：  实验内容：  实验效果：  实验报告：  考勤情况：  其他情况：  指导教师：  年 月 日 |

|  |
| --- |
| **实验报告内容基本要求及参考格式**  一、实验目的及小组分工情况  二、实验所用仪器（或实验环境）  三、实验方案设计  四、实验过程记录   1. 实验结果分析及实验心得 |

## 实验1歌手大赛

### 一、实验目的

掌握类如何实现接口。

### 二、实验要求

编写java程序，实现以下功能。歌手大赛计算选手成绩的办法是去掉一个最高分和一个最低分后再计算平均分，而学校评估一个班级的某科目考试成绩时，是计算全班同学的平均成绩。SongGame类和SchoolExam类都实现了ComputeAverage接口，但实现方式不同。

### 三、实验设计

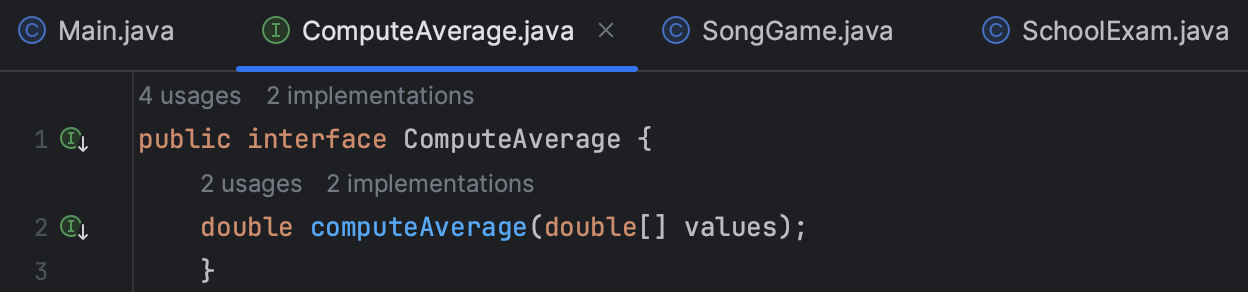
在这个实验中，ComputeAverage是一个接口，它定义了一个computeAverage()方法用于计算平均值。SongGame类和SchoolExam类都实现了ComputeAverage接口，并分别实现了自己的computeAverage()方法。在Main类的main方法中，分别创建了SongGame对象和SchoolExam对象，并调用它们的computeAverage()方法来计算选手成绩的平均分和学生成绩的平均值。

### 四、实验过程记录及代码

创建一个名为 ComputeAverage 的接口，其中定义了 computeAverage() 方法：

public interface ComputeAverage {

double computeAverage(double[] values);

}、

创建 SongGame 类，实现 ComputeAverage 接口，并实现自己的 computeAverage() 方法：

public class SongGame implements ComputeAverage {

@Override

public double computeAverage(double[] values) {

double sum = 0;

double min = Double.MAX\_VALUE;

double max = Double.MIN\_VALUE;

for (double value : values) {

sum += value;

min = Math.min(min, value);

max = Math.max(max, value);

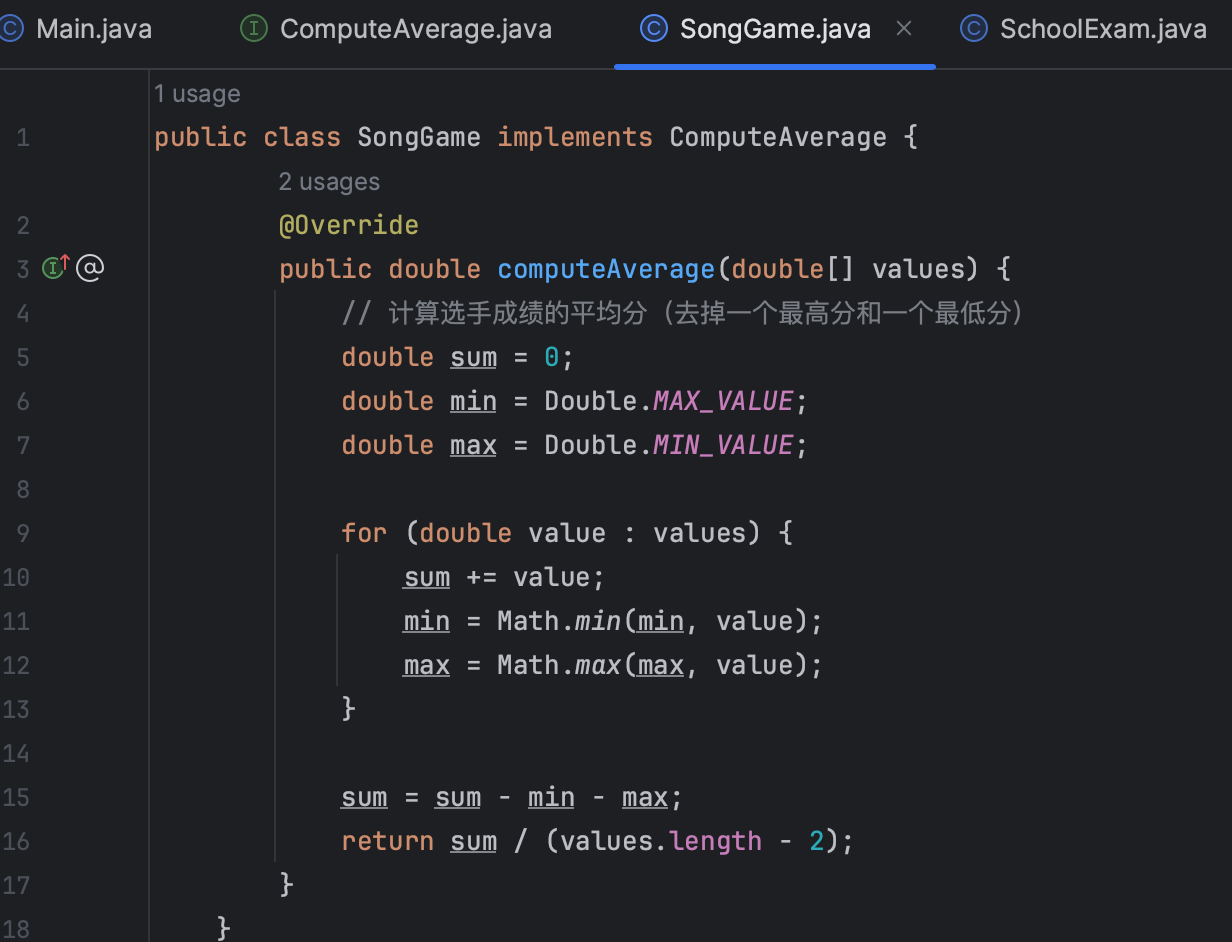
}

sum = sum - min - max;

return sum / (values.length - 2);

}

}



创建 SchoolExam 类，同样实现 ComputeAverage 接口，并实现自己的 computeAverage() 方法：

public class SchoolExam implements ComputeAverage {

@Override

public double computeAverage(double[] values) {

double sum = 0;

for (double value : values) {

sum += value;

}

return sum / values.length;

}

}



创建 Main 类，在其 main() 方法中创建 SongGame 和 SchoolExam 对象，并调用它们的 computeAverage() 方法计算平均值：

public class Main {

public static void main(String[] args) {

double[] songGameScores = {9.5, 8.8, 9.2, 9.7, 9.4};

ComputeAverage songGame = new SongGame();

Double songGameAverage = songGame.computeAverage(songGameScores);

System.out.println("Song Game Average: " + songGameAverage);

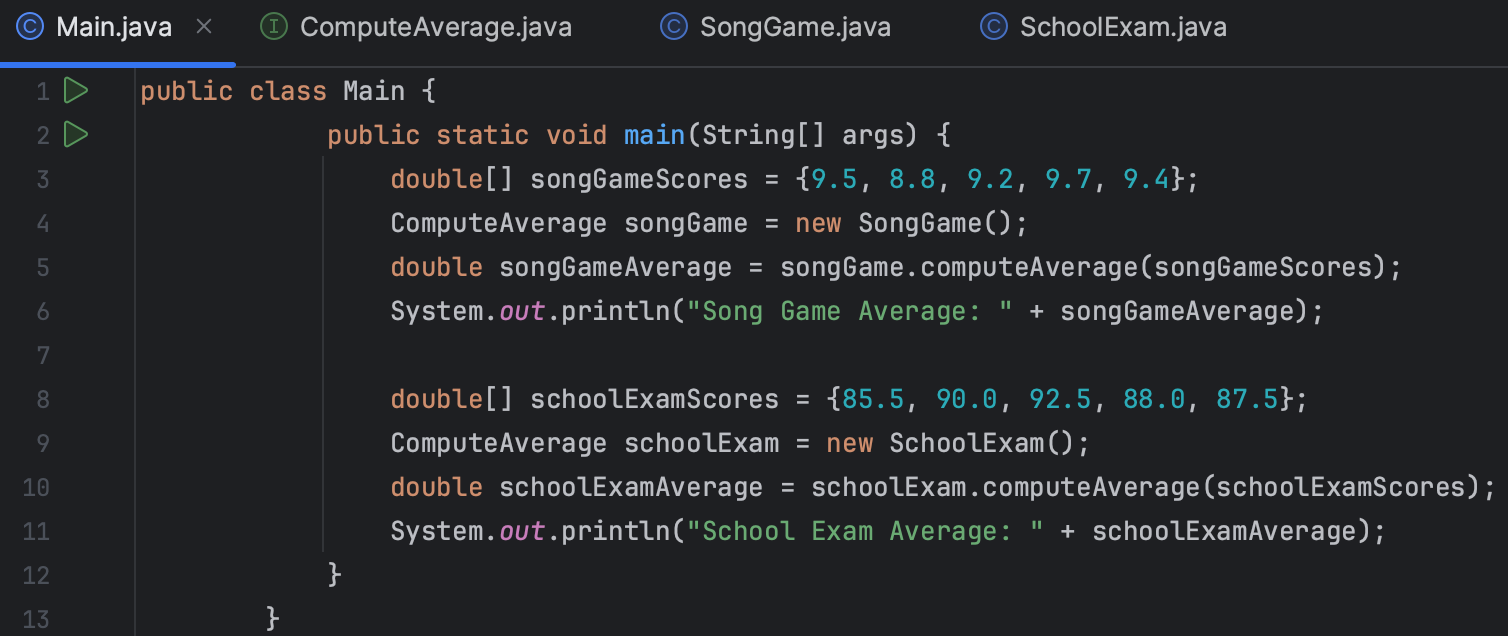
double[] schoolExamScores = {85.5, 90.0, 92.5, 88.0, 87.5};

ComputeAverage schoolExam = new SchoolExam();

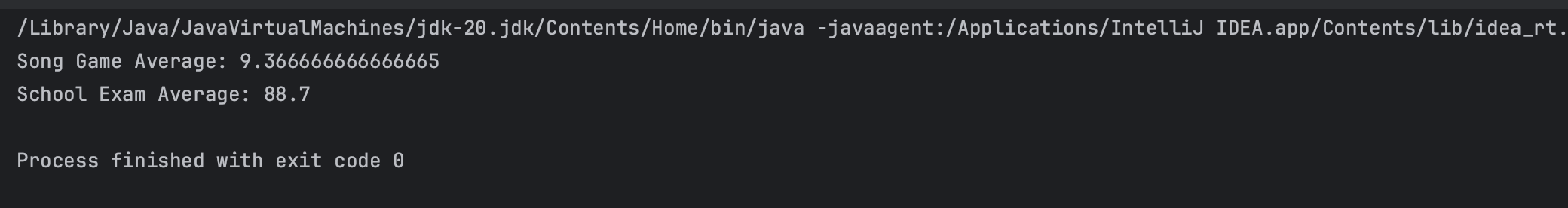
double schoolExamAverage = schoolExam.computeAverage(schoolExamScores);

System.out.println("School Exam Average: " + schoolExamAverage);

}

}

### 实验结果



### 五、实验结果分析及实验心得

实验结果分析：  
在歌手大赛实验中，实现了一个接口ComputeAverage，其中定义了computeAverage()方法用于计算平均值。SongGame类和SchoolExam类分别实现了该接口，并根据各自的需求实现了自己的computeAverage()方法。SongGame类通过去掉一个最高分和一个最低分后计算选手成绩的平均分，而SchoolExam类则计算全班同学的平均成绩。

实验心得：  
通过这个实验，学会了如何使用接口实现不同类的共同功能。接口提供了一种标准化的方式来定义类之间的交互，使得不同类可以根据自身的需求实现接口中的方法。在实验中，了解到接口的抽象性和灵活性，可以根据具体场景设计不同的实现方式。

## 实验2 货车的装载量

### 一、实验目的

本实验的目的是让学生掌握接口回调技术。

### 二、实验要求

按要求编写java程序。货车要装载一批货物，货物由电视机、计算机和洗衣机3种商品组成。卡车需要计算出整批货物的重量，具体需求如下。

（1）要求有一个ComputeWeight接口，该接口中有一个方法：

public double computeWeight ()。

（2）要求有3个实现该接口的类，即Television、Computer和 WashMachine。这3个类通过实现接口给出自重（可简单返回一个值表示货物的重量）。

（3）要求有一个Truck类，该类以ComputeWeight接口类型的数组作为成员（Truck类面向接口），那么该数组的单元就可以存放Television对象的引用、Computer对象的引用或WashMachine对象的引用。程序能输出Truck对象所装载的货物的总质量。

### 三、实验设计

在这个实验中，ComputeWeight是一个接口，它定义了一个computeWeight()方法用于计算物品的重量。Television、Computer和WashMachine类都实现了ComputeWeight接口，并实现了自己的computeWeight()方法来计算物品的重量。Truck类以ComputeWeight接口类型的数组作为成员，并实现了getTotalWeight()方法来计算所有货物的总重量。在Main类的main方法中，创建了一个Television对象、一个Computer对象和一个WashMachine对象，并将它们放入一个ComputerWeight数组中，然后创建了一个Truck对象，并调用它的getTotalWeight()方法来计算货物的总重量。

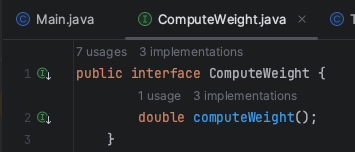
**四、实验过程及代码**

首先，创建一个名为 ComputeWeight 的接口，其中定义了 computeWeight() 方法：

public interface ComputeWeight {

double computeWeight();

}



创建 Television、Computer 和 WashMachine 类，它们分别实现了 ComputeWeight 接口，并实现了自己的 computeWeight() 方法：

class Television implements ComputeWeight {

public Television(){  
  
 }

private double weight;

private int quantity;

public Television(double weight, int quantity) {

this.weight = weight;

this.quantity = quantity;

}

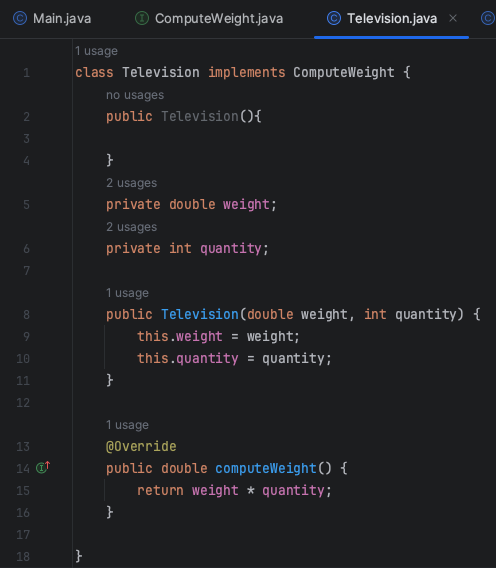
@Override

public double computeWeight() {

return weight \* quantity;

}

}



class Computer implements ComputeWeight {

public Computer(){  
  
 }

private double weight;

private int quantity;

public Computer(double weight, int quantity) {

this.weight = weight;

this.quantity = quantity;

}

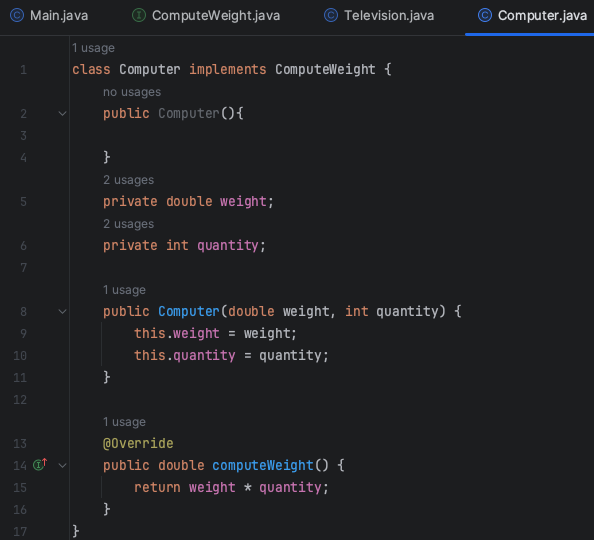
@Override

public double computeWeight() {

return weight \* quantity;

}

}



class WashMachine implements ComputeWeight {

public WashMachine(){  
  
 }

private double weight;

private int quantity;

public WashMachine(double weight, int quantity) {

this.weight = weight;

this.quantity = quantity;

}

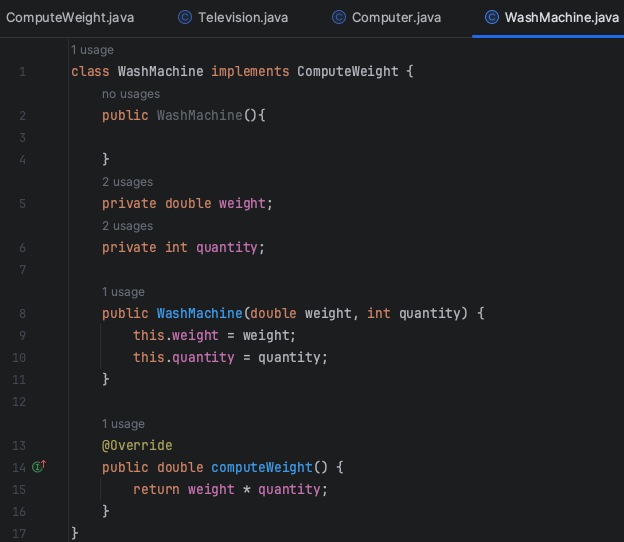
@Override

public double computeWeight() {

return weight \* quantity;

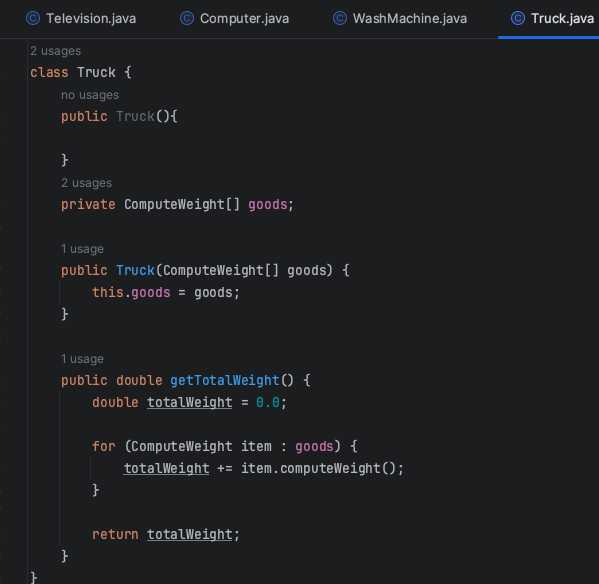
}

}



创建 Truck 类，以 ComputeWeight 接口类型的数组作为成员，并实现了 getTotalWeight() 方法来计算所有货物的总重量：

class Truck {  
 public Truck(){  
   
 }  
 private ComputeWeight[] goods;  
  
 public Truck(ComputeWeight[] goods) {  
 this.goods = goods;  
 }  
  
 public double getTotalWeight() {  
 double totalWeight = 0.0;  
  
 for (ComputeWeight item : goods) {  
 totalWeight += item.computeWeight();  
 }  
  
 return totalWeight;  
 }  
}



创建 Main 类，在其 main() 方法中创建 Television、Computer 和 WashMachine 对象，并将它们放入一个 ComputeWeight 数组中，然后创建 Truck 对象，并调用它的 getTotalWeight() 方法来计算货物的总重量：

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("请输入电视机的重量：");

double televisionWeight = scanner.nextDouble();

System.out.print("请输入电视机的数量：");

int televisionQuantity = scanner.nextInt();

System.out.print("请输入计算机的重量：");

double computerWeight = scanner.nextDouble();

System.out.print("请输入计算机的数量：");

int computerQuantity = scanner.nextInt();

System.out.print("请输入洗衣机的重量：");

double washMachineWeight = scanner.nextDouble();

System.out.print("请输入洗衣机的数量：");

int washMachineQuantity = scanner.nextInt();

scanner.close();

// 创建货物对象

ComputeWeight[] goods = {

new Television(televisionWeight, televisionQuantity),

new Computer(computerWeight, computerQuantity),

new WashMachine(washMachineWeight, washMachineQuantity)

};

// 创建货车对象

Truck truck = new Truck(goods);

double totalWeight = truck.getTotalWeight();

System.out.println("货车装载的货物总重量为：" + totalWeight);

}

}



### 实验结果



### 五、实验结果分析及实验心得

实验结果分析：  
在货车装载量实验中，实现了一个接口ComputeWeight，其中定义了computeWeight()方法用于计算物品的重量。Television、Computer和WashMachine类分别实现了该接口，并根据自身特点实现了各自的computeWeight()方法。Truck类以ComputeWeight接口类型的数组作为成员，通过遍历数组并调用各个物品的computeWeight()方法，计算出货车所装载货物的总质量。

实验心得：  
通过这个实验，学会了使用接口回调技术。通过接口回调，可以将不同类的实例传递给另一个类，并在需要时通过接口方法回调执行相应的操作。在实验中，了解到接口回调的灵活性和扩展性，可以实现不同类之间的松耦合，提高代码的可维护性和可扩展性。

## 实验3天气预报

### 一、实验目的

掌握面向接口编写思想。

### 二、实验要求

按要求编写java程序。天气可能出现不同的状态，要求用不同的接口封装天气的状态。具体要求如下：

1. 编写接口WeatherState，该接口有一个名字为void showState()的方法。
2. 编写Weather类，该类中有一个WeatherState声明的变量state。另外，该类有一个show()方法，在该方法中让接口state回调showState()的方法。
3. 编写若干个实现WeatherState接口的类，负责刻画天气的各种状态。
4. 编写主类，在主类中进行天气预报。

### 三、实验设计

在这个实验中，WeatherState是一个接口，它定义了一个showState()方法用于展示天气的状态。Sunny、Cloudy和Rainy类都实现了WeatherState接口，并实现了自己的showState()方法来展示不同的天气状态。Weather类有一个WeatherState类型的成员变量state，以及一个setState()方法用于设置天气状态，还有一个show()方法用于展示当前天气状态。在Main类的main方法中，创建了一个Weather对象，并分别设置为晴天、多云和雨天的状态，然后调用Weather对象的show()方法来展示当前天气状态。

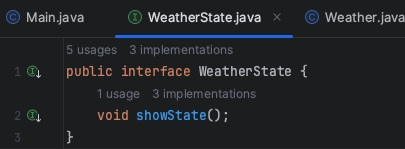
### 四、实验过程及代码

创建一个名为 WeatherState 的接口，其中定义了 showState() 方法：

public interface WeatherState {

void showState();

}



创建 Weather 类，其中包含一个 WeatherState 类型的成员变量 state，以及 setState() 方法和 show() 方法：

public class Weather {

private WeatherState state;

public void setState(WeatherState state) {

this.state = state;

}

public void show() {

state.showState();

}

}



创建实现 WeatherState 接口的 Sunny、Cloudy 和 Rainy 类，它们分别实现了自己的 showState() 方法：

public class Sunny implements WeatherState {

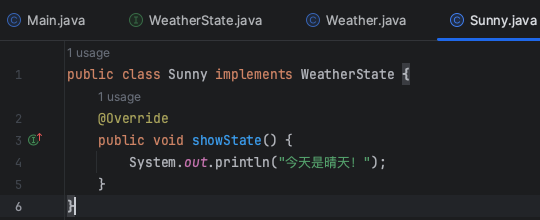
@Override

public void showState() {

System.out.println("今天是晴天！");

}

}



public class Cloudy implements WeatherState {

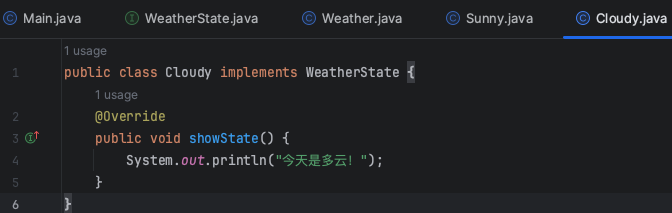
@Override

public void showState() {

System.out.println("今天是多云！");

}

}



public class Rainy implements WeatherState {

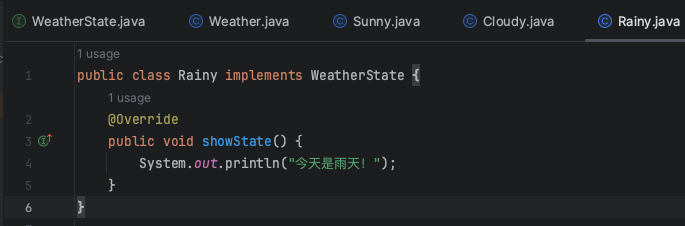
@Override

public void showState() {

System.out.println("今天是雨天！");

}

}



在 Main 类的 main() 方法中，创建一个 Weather 对象，并分别设置为晴天、多云和雨天的状态，然后调用 Weather 对象的 show() 方法来展示当前天气状态：

public class Main {

public static void main(String[] args) {

Weather weather = new Weather();

weather.setState(new Sunny());

weather.show();

weather.setState(new Cloudy());

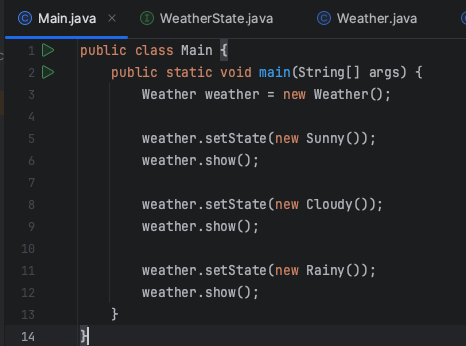
weather.show();

weather.setState(new Rainy());

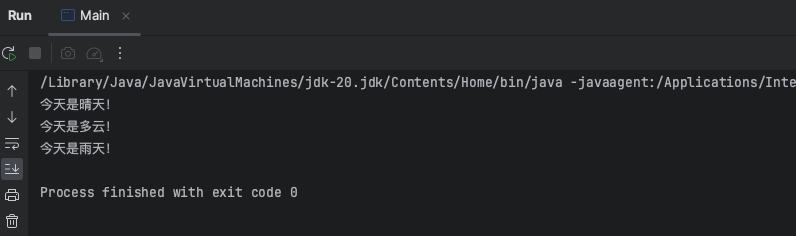
weather.show();

}

}



### 实验结果



### 五、实验结果分析及实验心得

实验结果分析：  
在天气预报实验中，定义了一个接口WeatherState，其中包含了一个showState()方法用于展示天气的状态。我们创建了几个实现了WeatherState接口的类，如Sunny、Cloudy和Rainy，它们分别实现了自己的showState()方法来展示不同的天气状态。Weather类包含一个WeatherState类型的成员变量state，通过调用state的showState()方法，实现了对当前天气状态的展示。

实验心得：  
通过这个实验，深入理解了面向接口编写的思想。接口提供了一种规范，使得不同类可以根据需求实现接口中定义的方法。在实验中，了解到面向接口编程的优势，可以实现代码的解耦和可扩展性，提高代码的灵活性和可维护性。

总结：  
这三个实验分别通过接口的方式，展示了不同领域的应用场景。通过实验，掌握了如何使用接口实现不同类之间的共同功能，如计算平均值、计算货物重量和展示天气状态。同时，还学会了接口回调技术，通过回调实现类与类之间的交互和松耦合。这些实验不仅加深了对接口和面向接口编程的理解，而且提升了编程技巧和解决问题的能力。