《Java 程序设计》实验报告六

学生姓名：李季鸿　　班级：2021 级计科本（3）班　　学号：20213002624

实验地点：9 教 404　　指导教师：张春元

实验日期：2023-04-10　　共 2 学时

实验环境：Win10+JDK1.8+IntelliJ IDEA 2022.1.1

1. 实验目的

学习接口与实现、内部类，掌握集成化开发工具工程的创建。

2. 实验内容

（1）用集成化开发工具完成实验教材 P53 实验 1 内容。
（2）用集成化开发工具完成实验教材 P55 实验 2 内容。
（3）用集成化开发工具完成实验教材 P57 实验 3 内容。
（4）用集成化开发工具完成实验教材 P62 实验 1 内容。

3. 实验过程

报告撰写具体要求：截屏显示或直接写出实验 1 至实验 4 的源码和运行结果。

实验内容（1）：

```
1. package _6_;
2.
3. /**
4.  * \* Created with IntelliJ IDEA.
5.  * \* User: lijihong
6.  * \* Date: 2023-04-10
7.  * \* Time: 16:35
8.  * \
9.  */
10.interface CompurerAverage {
11.    public double average(double x[]);
12.}
13.
14.class Gymnastics implements CompurerAverage {
15.    public double average(double x[]) {
16.        int count = x.length;
17.        double aver = 0, temp = 0;
18.        for (int i = 0; i < count; i++) {
19.            for (int j = i; j < count; j++) {
20.                if (x[j] < x[i]) {
21.                    temp = x[j];
22.                    x[j] = x[i];
23.                    x[i] = temp;
24.                }
```

```
            }
        }
        for (int i = 1; i < count - 1; i++) {
            aver = aver + x[i];
        }
        if (count > 2)
            aver = aver / (count - 2);
        else
            aver = 0;
        return aver;
    }
}

class School implements CompurerAverage {
    public double average(double x[]) {
        int count = x.length;
        double aver = 0;
        for (int i = 0; i < count; i++) {
            aver = aver + x[i];
        }
        aver = aver / count;
        return aver;
    }
}

public class Estimator {
    public static void main(String args[]) {
        double a[] = {9.89, 9.88, 9.99, 9.12, 9.69, 9.76, 8.97};
        double b[] = {89, 56, 78, 90, 100, 77, 56, 45, 36, 79, 98
   };
        CompurerAverage computer;
        computer = new Gymnastics();
        double result = computer.average(a);
        System.out.printf("%n");
        System.out.printf("体操选手最后得分:%5.3f\n", result);
        computer = new School();
        result = computer.average(b);
        System.out.printf("班级考试平均分数:%-5.2f", result);
    }
}
```

```
体操选手最后得分:9.668
班级考试平均分数:73.09
进程已结束,退出代码0
```

实验后的练习：

```
D:\java_study_codes\src\_6_\Estimator.java:38
java: _6_.School不是抽象的, 并且未覆盖_6_.CompurerAverage中的抽象方法average(double[])
```

通过此次实验我们可以知道在类中怎么实现接口，接口体中所有的常量的访 问权限一定都是 public，所有的抽象方法的访问权限一定都是 public；一个类可以实现多个接口，关键字为 implements，且如果一个非抽象类实现了某个接口，那么这个类必须重写该接口的所有方法；如果把实现某一接口的类创建的对象的引用赋给该接口声明的接口变量中，那么该接口变量就可以调用被类实现的接口方法；在实验后的练习中编译时会提示 School 不是抽象的，并且未覆盖 ComputerAverage 中的抽象方法。

实验内容（2）：

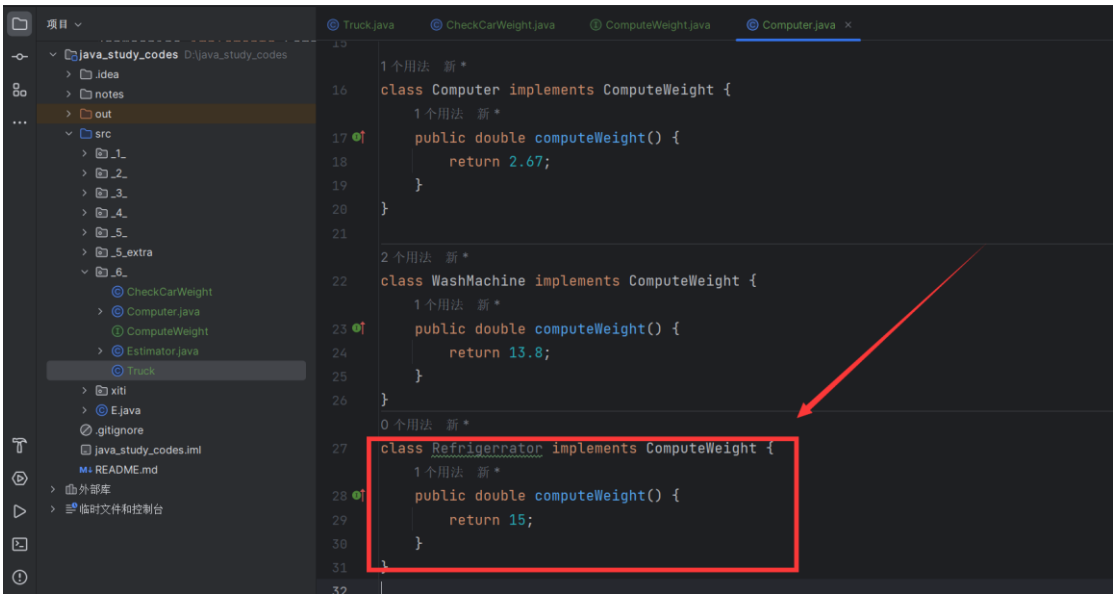
```java
package _6_;

interface ComputeWeight {
    public double computeWeight();
}
package _6_;

/**
 * \* Created with IntelliJ IDEA.
 * \* User: lijihong
 * \* Date: 2023-04-10
 * \* Time: 16:47
 * \
 */
class Television implements ComputeWeight {
    public double computeWeight() {
        return 3.5;
    }
}

class Computer implements ComputeWeight {
    public double computeWeight() {
        return 2.67;
    }
}
```

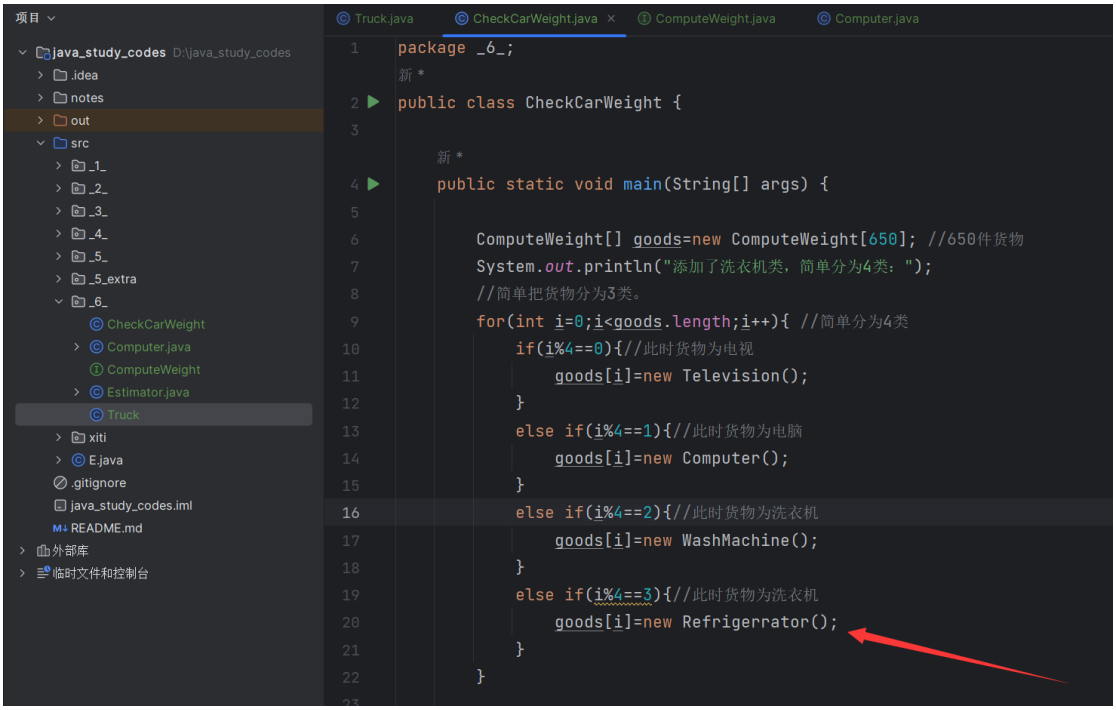
```
26.
27.class WashMachine implements ComputeWeight {
28.    public double computeWeight() {
29.        return 13.8;
30.    }
31.}
32.package _6_;
33.
34.public class Truck {
35.    ComputeWeight[] goods; //用ComputeWeight 接口类型的数组作为数据成员
36.    double totalweights = 0;
37.
38.    //构造方法
39.    Truck(ComputeWeight[] goods) {
40.        this.goods = goods;
41.    }
42.
43.    //
44.    public void setgoods(ComputeWeight[] goods) {
45.        this.goods = goods;
46.    }
47.
48.    //计算货物总重量。
49.    //每一件货物都有一个自重，所有货物的自重加起来就是这批货物（goods 数组）的总重（totalweights）。
50.    public double getTotalweights() {
51.        totalweights = 0;
52.        for (int i = 0; i < goods.length; i++) {
53.            totalweights = totalweights + goods[i].computeWeight();
54.        }
55.        return totalweights;
56.    }
57.}
58.package _6_;
59.public class CheckCarWeight {
60.
61.    public static void main(String[] args) {
62.
63.        ComputeWeight[] goods=new ComputeWeight[650]; //650 件货物
64.
65.        //简单把货物分为3 类。
```

```
        for(int i=0;i<goods.length;i++){ //简单分为三类
            if(i%3==0){//此时货物为电视
                goods[i]=new Television();
            }
            else if(i%3==1){//此时货物为电脑
                goods[i]=new Computer();
            }
            else if(i%3==2){//此时货物为洗衣机
                goods[i]=new WashMachine();
            }
        }

        //货车 truck1，装着 goods 这组货物。
        Truck truck =new Truck(goods);
        System.out.printf("\n 货车装载的货物总重量:%-8.5f kg\n",truck.getTotalweights());
        goods = new ComputeWeight[68]; //68 件货物
        for (int i = 0; i < goods.length; i++) {
            if(i % 2 == 0) goods[i] = new Television();
            else goods[i] =  new WashMachine();
        }
        truck.setgoods(goods);
        System.out.printf("\n 货车装载的货物总重量:%-8.5f kg\n",truck.getTotalweights());
        /*
    对于数组 goods 中的每一个 good[i]，都已分配给不同的类的对象（电视/电脑/洗衣机），
  则调用 truck1.getTotalweights()，具体执行也是对应类重写的 getTotalweights()方法。
   */
    }
}
```

```
货车装载的货物总重量:4319.69000 kg

货车装载的货物总重量:588.20000 kg

进程已结束,退出代码0
```

实验后的练习：






```
添加了洗衣机类，简单分为4类：

货车装载的货物总重量:5671.31000 kg

货车装载的货物总重量:588.20000 kg

进程已结束,退出代码0
```

实验证明，添加了之后不用修改 Truck 类。

实验内容（3）：

```
1. package _6_.shiyan3;
2.
3. public interface DogState {
4.     public void showState();
5. }
6. package _6_.shiyan3;
7.
8. /**
9.  * \* Created with IntelliJ IDEA.
10. * \* User: lijihong
11. * \* Date: 2023-04-10
12. * \* Time: 17:11
13. * \
14. */
15.public class Dog {
16.    DogState state;//声明接口的变量
17.    public void show(){
18.        state.showState();//调用接口里面的函数
19.    }
20.    public void setState(DogState s){
21.        state=s;
22.    }
23.}
24.package _6_.shiyan3;
25.
26./**
27. * \* Created with IntelliJ IDEA.
28. * \* User: lijihong
29. * \* Date: 2023-04-10
30. * \* Time: 17:18
31. * \
32. */
33.public class SoflyState implements DogState {
34.    public void showState(){
35.        System.out.println("听主人的命令");
36.    }
37.}
38.package _6_.shiyan3;
39.
40./**
```

```
41. * \* Created with IntelliJ IDEA.
42. * \* User: lijihong
43. * \* Date: 2023-04-10
44. * \* Time: 17:13
45. * \
46. */
47.class MeetEnemyState implements DogState{//实现接口的类
48.
49.    @Override
50.    public void showState() {
51.        System.out.println("遇到敌人狂叫！");
52.    }
53.}
54.package _6_.shiyan3;
55.
56./**
57. * \* Created with IntelliJ IDEA.
58. * \* User: lijihong
59. * \* Date: 2023-04-10
60. * \* Time: 17:15
61. * \
62. */
63.public class MeetFriendState implements DogState{//实现接口的类
64.    @Override
65.    public void showState() {
66.        System.out.println("遇到朋友晃动尾巴表示欢迎！");
67.    }
68.}
69.package _6_.shiyan3;
70.
71./**
72. * \* Created with IntelliJ IDEA.
73. * \* User: lijihong
74. * \* Date: 2023-04-10
75. * \* Time: 17:16
76. * \
77. */
78.public class MeetAnotherDog implements DogState{
79.    @Override
80.    public void showState() {
81.        System.out.println("嬉戏");
82.    }
83.}
```

```java
package _6_.shiyan3;

/**
 * \* Created with IntelliJ IDEA.
 * \* User: lijihong
 * \* Date: 2023-04-10
 * \* Time: 17:17
 * \
 */
public class CheckDogState {
    public static void main(String[] args) {
        Dog yelloDog = new Dog();
        System.out.print("狗在主人前面：");
        yelloDog.setState(new SoflyState());
        yelloDog.show();
        System.out.print("狗遇到敌人：");
         yelloDog.setState(new MeetEnemyState());
         yelloDog.show();
         System.out.print("狗遇到朋友：");
         yelloDog.setState(new MeetFriendState());
         yelloDog.show();
         System.out.print("狗遇到同伴：");
         yelloDog.setState(new MeetAnotherDog());
         yelloDog.show();
    }
}
```

```
狗在主人前面：听主人的命令
狗遇到敌人：遇到敌人狂叫！
狗遇到朋友：遇到朋友晃动尾巴表示欢迎！
狗遇到同伴：嬉戏

进程已结束,退出代码0
```

实验后的练习：
可以先定义一个接口：Water:

```java
package _6_.shiyan3_extra;

2 个用法  2 个实现  新 *
public interface Water {
    0 个用法  2 个实现  新 *
    public void increaseTemperature(double degree);
    0 个用法  2 个实现  新 *
    public void decreaseTemperature(double degree);
    0 个用法  2 个实现  新 *
    public String getState();
}

```

在这个接口中，我们定义了三个方法：

increaseTemperature：增加水的温度

decreaseTemperature：减少水的温度

getState：获取水的状态

接着，我们可以定义一个 ColdWater 类，用于表示水温低于 4 度时的状态：

```java
package _6_.shiyan3_extra;

/**
 * \* Created with IntelliJ IDEA.
 * \* User: lijihong
 * \* Date: 2023-04-10
 * \* Time: 17:27
 * \
 */
public class ColdWater implements Water {
    private double temperature;

    public ColdWater() {
        this.temperature = 0;
    }

    public void increaseTemperature(double degree) {
        this.temperature += degree;
    }

    public void decreaseTemperature(double degree) {
        this.temperature -= degree;
        if (this.temperature < 0) {
            this.temperature = 0;
        }
```

```
    }

    public String getState() {
        if (this.temperature < 4) {
            return "cold";
        } else if (this.temperature >= 4 && this.temperature < 100) {
            return "liquid";
        } else {
            return "gas";
        }
    }
}
```

在 ColdWater 类中，我们实现了 Water 接口，并且定义了一个 temperature 变量，用于表示水的温度。在 increaseTemperature 方法中，我们可以增加水的温度；在 decreaseTemperature 方法中，我们可以减少水的温度，并且如果水的温度低于 0 度，就将其设置为 0 度；在 getState 方法中，我们可以根据水的温度返回水的状态，如果水温低于 4 度，则状态为“cold”，如果水温在 4 度和 100 度之间，则状态为“liquid”，如果水温高于 100 度，则状态为“gas”。
接着，我们可以定义一个 HotWater 类，用于表示水温高于 100 度时的状态：

```
package _6_.shiyan3_extra;

/**
 * \* Created with IntelliJ IDEA.
 * \* User: lijihong
 * \* Date: 2023-04-10
 * \* Time: 17:27
 * \
 */
public class HotWater implements Water {
    private double temperature;

    public HotWater() {
        this.temperature = 100;
    }

    public void increaseTemperature(double degree) {
        this.temperature += degree;
        if (this.temperature > 100) {
            this.temperature = 100;
        }
    }
```

```
23.
24.     public void decreaseTemperature(double degree) {
25.         this.temperature -= degree;
26.     }
27.
28.     public String getState() {
29.         if (this.temperature < 4) {
30.             return "cold";
31.         } else if (this.temperature >= 4 && this.temperature < 10
    0) {
32.             return "liquid";
33.         } else {
34.             return "gas";
35.         }
36.     }
37. }
```

在 HotWater 类中，我们也实现了 Water 接口，并且定义了一个 temperature 变量，用于表示水的温度。在 increaseTemperature 方法中，我们可以增加水的温度，并且如果水的温度高于 100 度，就将其设置为 100 度；在 decreaseTemperature 方法中，我们可以减少水的温度；在 getState 方法中，我们可以根据水水的温度返回水的状态，如果水温低于 4 度，则状态为“cold”，如果水温在 4 度和 100 度之间，则状态为“liquid”，如果水温高于 100 度，则状态为“gas”。

最后，我们可以编写一个测试程序，来模拟水杯中水的温度变化和状态变化：

```
1. public class WaterTest {
2.     public static void main(String[] args) {
3.         Water water = new ColdWater();
4.         System.out.println(water.getState()); // 输出 cold
5.
6.         water.increaseTemperature(10);
7.         System.out.println(water.getState()); // 输出 liquid
8.
9.         water.increaseTemperature(90);
10.        System.out.println(water.getState()); // 输出 gas
11.
12.        water.decreaseTemperature(50);
13.        System.out.println(water.getState()); // 输出 liquid
14.
15.        water = new HotWater();
16.        System.out.println(water.getState()); // 输出 gas
17.
18.        water.decreaseTemperature(20);
```

```
        System.out.println(water.getState()); // 输出 liquid

        water.decreaseTemperature(110);
        System.out.println(water.getState()); // 输出 cold
    }
}
```

在这个测试程序中，我们首先创建一个 ColdWater 对象，并且输出其状态为“cold”。接着，我们增加水的温度，再次输出水的状态，这次状态变为了“liquid”。然后，我们继续增加水的温度，输出状态变为“gas”。接着，我们减少水的温度，状态变回“liquid”。然后，我们创建一个 HotWater 对象，并且输出其状态为“gas”。接着，我们减少水的温度，状态变为“liquid”。最后，我们再次减少水的温度，状态变为“cold”。
这个程序使用了面向接口的思想，将水的状态抽象成了一个接口，并且通过不同的实现类来表示不同的状态。这样做的好处是，我们可以很方便地扩展程序，添加新的水的状态，而不需要修改现有的代码。同时，这个程序还使用了多态的特性，可以让我们在运行时动态地切换不同的实现类，从而模拟不同温度下水的状态的变化。
后续发现，希望获得当前状态的温度，于是，在接口中添加了：public double getTemperature();
在 ColdWater.java 和 HotWater.java 中分别加上了：

```
public double getTemperature(){
    return this.temperature;
}
```

用来获取当前水温，输出为：

```
冷水最初状态：cold 0.0 度
加了10°：liquid 10.0 度
加了90°：gas 100.0 度
减了50°：liquid 50.0 度
热水最初状态：gas 100.0 度
减了20°：liquid 80.0 度
减了110°：cold -30.0 度

进程已结束,退出代码0
```

实验内容（4）：

```
package _6_.shiyan4;

import javax.swing.*;

/**
 * \* Created with IntelliJ IDEA.
 * \* User: lijihong
 * \* Date: 2023-04-10
```

```
 * \* Time: 17:48
 * \
 */
public class MobileShop {
    InnerPurchaseMoney purchaseMoney1;
    InnerPurchaseMoney purchaseMoney2;
    private int mobileAmount; // 手机的数量
    MobileShop(){
        purchaseMoney1 = new InnerPurchaseMoney(20000);
        purchaseMoney2 = new InnerPurchaseMoney(10000);
    }
    void setMobileAmount(int m){
        mobileAmount = m;
    }
    int getMobileAmount(){
        return mobileAmount;
    }
    class InnerPurchaseMoney{
        int moneyValue;
        InnerPurchaseMoney(int m){
            moneyValue = m;
        }
        void buyMobile(){
            if(moneyValue >= 20000){
                mobileAmount = mobileAmount - 6;
                System.out.println("用价值" + moneyValue + "的内部购物卷买了 6 部手机");
            }
            else if(moneyValue < 20000 && moneyValue >= 10000){
                mobileAmount = mobileAmount - 3;
                System.out.println("用价值" + moneyValue + "的内部购物卷买了 3 部手机");
            }
        }
    }
}
package _6_.shiyan4;

/**
 * \* Created with IntelliJ IDEA.
 * \* User: lijihong
 * \* Date: 2023-04-10
 * \* Time: 17:56
```

```
50. * \
51. */
52.public class NewYear {
53.    public static void main(String[] args) {
54.        MobileShop shop = new MobileShop();
55.        shop.setMobileAmount(30);
56.        System.out.println("手机店目前有
   " + shop.getMobileAmount() + "部手机");
57.        shop.purchaseMoney1.buyMobile();
58.        shop.purchaseMoney2.buyMobile();
59.        System.out.println("手机店目前有
   " + shop.getMobileAmount() + "部手机");
60.    }
61.}
```

```
手机店目前有30部手机
用价值20000的内部购物卷买了6部手机
用价值10000的内部购物卷买了3部手机
手机店目前有21部手机
```

实验后的练习：

以下是一个简单的实例，使用 Java 内部类模拟一个学生选课的场景。Course 类表示课程，Student 类表示学生，Enrollment 类表示学生的选课记录。每个学生可以选多门课程，每门课程可以有多个学生选修。在 Student 类中，我们定义了一个内部类 Enrollment，用于表示学生的选课记录。

Enrollment 类是 Student 类的内部类，表示学生的选课记录。在 Enrollment 类中，我们只保留了一个 Course 对象，表示学生选修的课程。

Enrollment 类中有一个 getCourse 方法，用于返回学生选修的课程对象。在 Student 类的 enroll 和 withdraw 方法中，我们分别创建和删除一个 Enrollment 对象，并将其添加到或从 enrollmentList 中删除。

接下来，我们可以在 Main 类中使用 Student 和 Course 类模拟学生选课的过程。

```
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        Course math101 = new Course("Math 101", 1);
        Course english101 = new Course("English 101", 2);

        Student alice = new Student("Alice");
        alice.enroll(math101);
```

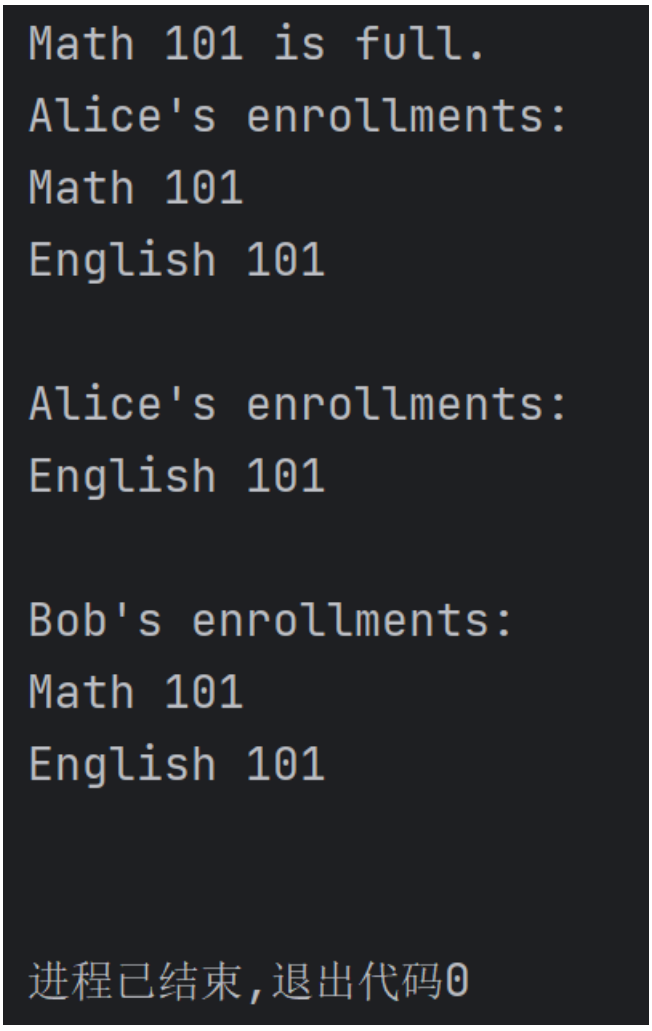
```
        alice.enroll(english101);
        alice.enroll(math101);

        alice.printEnrollments();

        alice.withdraw(math101);
        alice.withdraw(math101);

        alice.printEnrollments();

        Student bob = new Student("Bob");
        bob.enroll(math101);
        bob.enroll(english101);

        bob.printEnrollments();
    }
}

class Course {
    private String name;
    private int capacity;
    private List<Student> students;

    public Course(String name, int capacity) {
        this.name = name;
        this.capacity = capacity;
        this.students = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public int getCapacity() {
        return capacity;
    }

    public int getNumStudents() {
        return students.size();
    }

    public boolean isFull() {
        return getNumStudents() >= capacity;
```

```
    }

    public void addStudent(Student student) {
        students.add(student);
    }

    public void removeStudent(Student student) {
        students.remove(student);
    }
}

class Student {
    private String name;
    private List<Course> courses;

    public Student(String name) {
        this.name = name;
        this.courses = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public void enroll(Course course) {
        if (course.isFull()) {
            System.out.println(course.getName() + " is full.");
            return;
        }
        if (!courses.contains(course)) {
            courses.add(course);
            course.addStudent(this);
        }
    }

    public void withdraw(Course course) {
        if (courses.contains(course)) {
            courses.remove(course);
            course.removeStudent(this);
        }
    }

    public void printEnrollments() {
```

```
            System.out.println(name + "'s enrollments:");
            for (Course course : courses) {
                System.out.println(course.getName());
            }
            System.out.println();
        }
}
```

在这个例子中，我们创建了两门课程：Math 101 和 English 101。然后创建了两个学生：Alice 和 Bob。Alice 首先选择了 Math 101 和 English 101，然后再次选择了 Math 101。当她尝试再次选择 Math 101 时，因为该课程已经满员，系统会输出一条“Math 101 is full.”的消息。然后，Alice 打印了她的选课记录，然后从 Math 101 中撤回了她的选课。再次打印她的选课记录后，Math 101 已经不在记录中了。最后，Bob 选择了 Math 101 和 English 101，并打印了他的选课记录。

运行结果如下图所示：



4. 实验总结

写出实验中的心得体会（对第 6 章理论课重点简述）。

接口与抽象类的区别

☞ 数据域区别：在接口中只能定义常量；抽象类的数据域

则既可定义常量、也可定义变量。

☞ 方法的区别：接口中的方法必须是公有抽象的实例方法、default 实例方法、static 方法和 priavte 方法，没有其它类型的方法（包括构造方法也不存在）；抽象类中的方法没有限制（存在构造方法，但不能用其实例化对象），但其抽象方法必须是可访问的抽象实例方法。

☞ 继承的区别：接口之间可以存在多重继承，接口不能继承类；类之间只能单重继承，并且可以实现多个接口。

什么时候使用接口？

接口和抽象类都可以抽象出重要的行为标准，该行为标准用抽象方法来表示。那什么情况下使用接口？

子类和父类之间是“强是(is-a)”关系，明显的父子关系用类来模拟；类与接口之间是“弱是(like-a)”关系，一般用接口来定义不相关的多个类的共有属性(并不要求这些类间关系是父子类关系)。此外，通过接口还可以来实现类间所不具备的多重继承关系。