**第五次 作业**

**第一题**

* **有两个类Person和Student，它们存在继承关系。在Person中，有成员变量name，sex，age三个,它们类型分别为String、char、int，有一个带参数的构造方法Person(String, char, int)，用来对成员变量初始化。有成员方法setData(String, char, int)设置成员变量name, sex, age的值。getData()是不带参数且返回值由name、sex和age值构成的字符串的成员方法。**
* **Student类是Person类的子类，在Student中，有int类型的sID和classNo成员变量用来表示学生的学号和班级号。它有带五个参数的方法setData()和不带参数的方法getData()，setData()方法设置成员变量的值，而getData()则是返回五个成员变量的值组成的字符串。**
* **设计实现以上2个类，并另外设计一个包含入口main方法的类，对主要方法进行调用并得到输出结果**

源代码：

class Person {

    String name;

    char sex;

    int age;

    public Person(String name, char sex, int age) {

        this.name = name;

        this.sex = sex;

        this.age = age;

    }

    public void setData(String name, char sex, int age) {

        this.name = name;

        this.sex = sex;

        this.age = age;

    }

    public String getData() {

        return "Name: " + name + ", Sex: " + sex + ", Age: " + age;

    }

}

class Student extends *Person* {

    int sID;

    int classNo;

    public Student(String name, char sex, int age, int sID, int classNo) {

        super(name, sex, age);

        this.sID = sID;

        this.classNo = classNo;

    }

    public void setData(String name, char sex, int age, int sID, int classNo) {

        super.setData(name, sex, age);

        this.sID = sID;

        this.classNo = classNo;

    }

    public String getData() {

        return super.getData() + ", Student ID: " + sID + ", Class No: " + classNo;

    }

}

*// 包含main方法的类*

public class Main {

    public static void main(String[] args) {

*// 创建Person对象并调用方法*

        Person person = new Person("Alice", 'F', 30);

        System.out.println(person.getData()); *// 输出: Name: Alice, Sex: F, Age: 30*

*// 创建Student对象并调用方法*

        Student student = new Student("Bob", 'M', 20, 12345, 3);

        System.out.println(student.getData()); *// 输出: Name: Bob, Sex: M, Age: 20, Student ID: 12345, Class No: 3*

*// 使用setData更新Student对象的值*

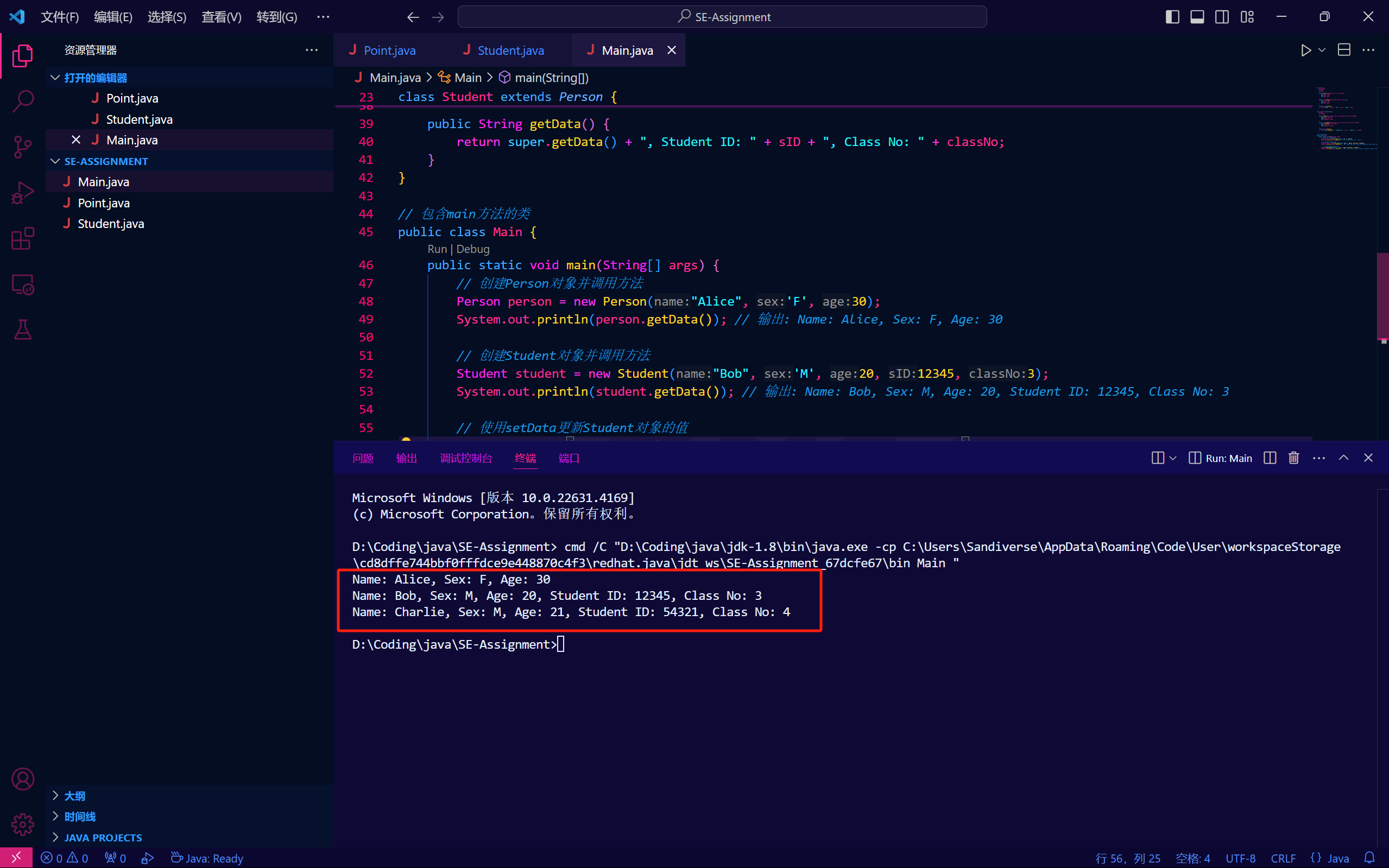
        student.setData("Charlie", 'M', 21, 54321, 4);

        System.out.println(student.getData()); *// 输出: Name: Charlie, Sex: M, Age: 21, Student ID: 54321, Class No: 4*

    }

}

运行结果截图：



**第二题**

* **将上一题Person类修改为抽象类**
* **类Student和类Teacher均分别是抽象类Person的子类。**
  + **类Student有成员变量name，sex，age，sID，speciality，其中sID表示学生的学号，speciality表示学生专业。**
  + **类Teacher有成员变量name，sex，age，tID，department，其中tID表示教师的编号，department表示教师所在的部门。**
* **设计实现以上3个类，其中学生和教师类自行实现部分基本功能。并另外设计一个包含入口main方法的类，对主要方法进行调用并得到输出结果**

源代码：

abstract class Person {

    String name;

    char sex;

    int age;

    public Person(String name, char sex, int age) {

        this.name = name;

        this.sex = sex;

        this.age = age;

    }

    public abstract String getData();

}

class Student extends *Person* {

    int sID;

    String speciality;

    public Student(String name, char sex, int age, int sID, String speciality) {

        super(name, sex, age);

        this.sID = sID;

        this.speciality = speciality;

    }

    public String getData() {

        return "Student [Name: " + name + ", Sex: " + sex + ", Age: " + age

                + ", Student ID: " + sID + ", Speciality: " + speciality + "]";

    }

}

class Teacher extends *Person* {

*// 教师特有的成员变量*

    int tID; *// 教师编号*

    String department; *// 部门*

    public Teacher(String name, char sex, int age, int tID, String department) {

        super(name, sex, age);

        this.tID = tID;

        this.department = department;

    }

    public String getData() {

        return "Teacher [Name: " + name + ", Sex: " + sex + ", Age: " + age

                + ", Teacher ID: " + tID + ", Department: " + department + "]";

    }

}

*// 包含main方法的类*

public class Main {

    public static void main(String[] args) {

        Student student = new Student("Alice", 'F', 20, 12345, "Computer Science");

        System.out.println(student.getData());

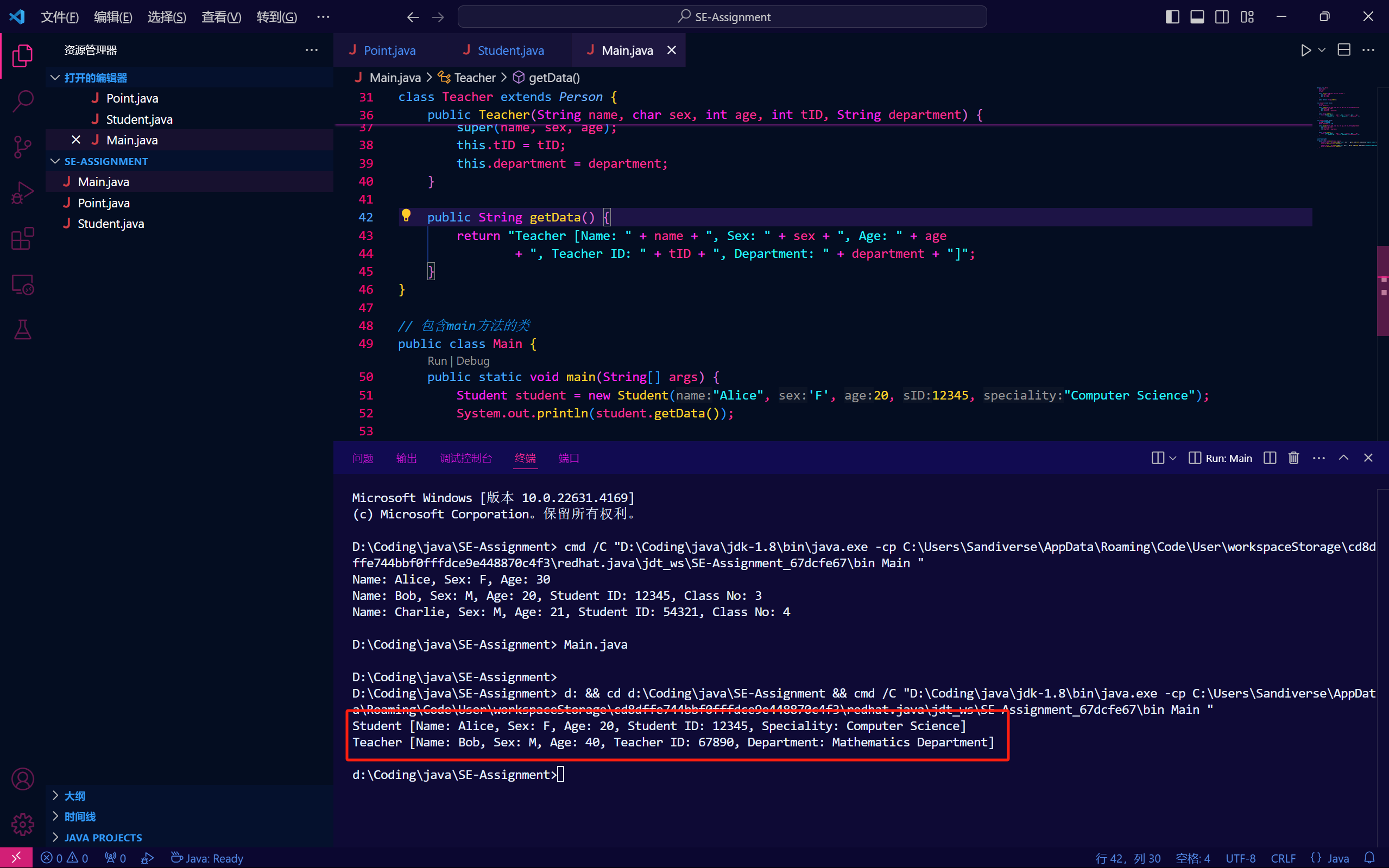
        Teacher teacher = new Teacher("Bob", 'M', 40, 67890, "Mathematics Department");

        System.out.println(teacher.getData());

    }

}

运行截图：



**第三题**

* **在前两题的基础上增加继承的层次**
* **大学生是学生的子类，增加专业、方向等属性，并有相应的get、set方法**
* **硕士研究生是大学生的子类，增加研究方向等属性，并有相应的方法**
* **自行思考并决定是否继续增加层次：本科生、专科生、高职生、博士研究生等等**
* **完成类的设计，并另外设计一个包含入口main方法的类，对不同层次的类的变量对实例的应用进行验证，并验证多态性**

源代码：

abstract class Person {

    String name;

    char sex;

    int age;

    public Person(String name, char sex, int age) {

        this.name = name;

        this.sex = sex;

        this.age = age;

    }

*// 抽象方法*

    public abstract String getData();

}

*// Student类继承自Person*

class Student extends *Person* {

    int sID;

    public Student(String name, char sex, int age, int sID) {

        super(name, sex, age);

        this.sID = sID;

    }

    public String getData() {

        return "Student [Name: " + name + ", Sex: " + sex + ", Age: " + age + ", Student ID: " + sID + "]";

    }

}

*// Undergraduate类继承自Student（本科生）*

class Undergraduate extends *Student* {

    String speciality;  *// 专业*

    String direction;   *// 方向*

    public Undergraduate(String name, char sex, int age, int sID, String speciality, String direction) {

        super(name, sex, age, sID);

        this.speciality = speciality;

        this.direction = direction;

    }

    public String getSpeciality() {

        return speciality;

    }

    public void setSpeciality(String speciality) {

        this.speciality = speciality;

    }

    public String getDirection() {

        return direction;

    }

    public void setDirection(String direction) {

        this.direction = direction;

    }

    public String getData() {

        return super.getData() + ", Speciality: " + speciality + ", Direction: " + direction;

    }

}

*// Graduate类继承自Undergraduate（研究生）*

class Graduate extends *Undergraduate* {

    String researchArea;  *// 研究方向*

    public Graduate(String name, char sex, int age, int sID, String speciality, String direction, String researchArea) {

        super(name, sex, age, sID, speciality, direction);

        this.researchArea = researchArea;

    }

    public String getResearchArea() {

        return researchArea;

    }

    public void setResearchArea(String researchArea) {

        this.researchArea = researchArea;

    }

    public String getData() {

        return super.getData() + ", Research Area: " + researchArea;

    }

}

*// Master类继承自Graduate（硕士研究生）*

class Master extends *Graduate* {

    public Master(String name, char sex, int age, int sID, String speciality, String direction, String researchArea) {

        super(name, sex, age, sID, speciality, direction, researchArea);

    }

    public String getData() {

        return "Master Student [" + super.getData() + "]";

    }

}

*// PhD类继承自Graduate（博士研究生）*

class PhD extends *Graduate* {

    public PhD(String name, char sex, int age, int sID, String speciality, String direction, String researchArea) {

        super(name, sex, age, sID, speciality, direction, researchArea);

    }

    public String getData() {

        return "PhD Student [" + super.getData() + "]";

    }

}

*// 包含main方法的类*

public class Main {

    public static void main(String[] args) {

*// 创建本科生对象*

        Undergraduate undergrad = new Undergraduate("Alice", 'F', 20, 12345, "Computer Science", "Software Engineering");

        System.out.println(undergrad.getData());

*// 输出: Student [Name: Alice, Sex: F, Age: 20, Student ID: 12345], Speciality: Computer Science, Direction: Software Engineering*

*// 创建硕士研究生对象*

        Master master = new Master("Bob", 'M', 25, 67890, "Physics", "Quantum Mechanics", "Quantum Computing");

        System.out.println(master.getData());

*// 输出: Master Student [Student [Name: Bob, Sex: M, Age: 25, Student ID: 67890], Speciality: Physics, Direction: Quantum Mechanics, Research Area: Quantum Computing]*

*// 创建博士研究生对象*

        PhD phd = new PhD("Charlie", 'M', 30, 13579, "Biology", "Genetics", "Gene Editing");

        System.out.println(phd.getData());

*// 输出: PhD Student [Student [Name: Charlie, Sex: M, Age: 30, Student ID: 13579], Speciality: Biology, Direction: Genetics, Research Area: Gene Editing]*

*// 验证多态性*

        Person person1 = undergrad;

        Person person2 = master;

        Person person3 = phd;

        System.out.println();

        System.out.println(person1.getData()); *// 调用子类 Undergraduate 的 getData()*

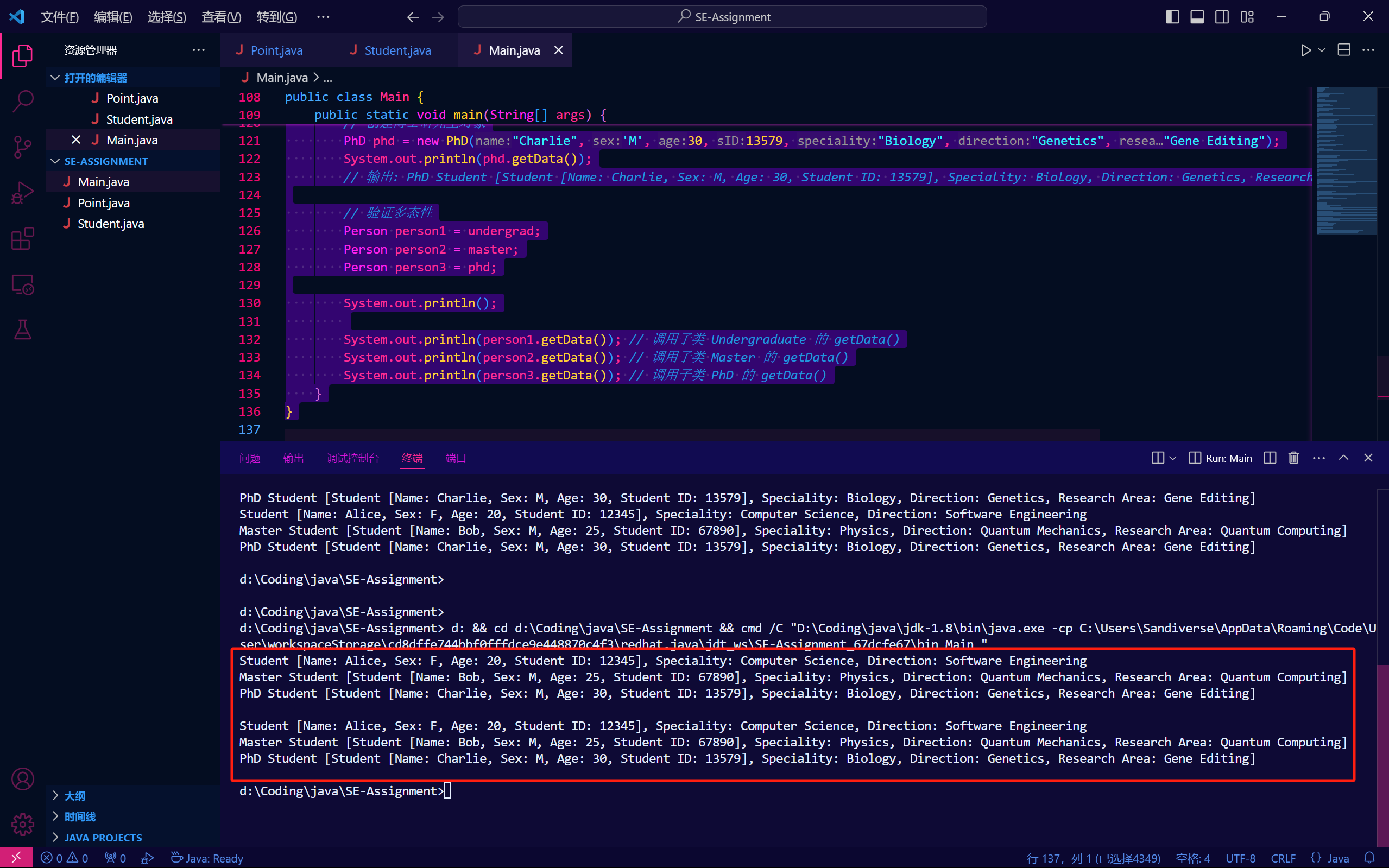
        System.out.println(person2.getData()); *// 调用子类 Master 的 getData()*

        System.out.println(person3.getData()); *// 调用子类 PhD 的 getData()*

    }

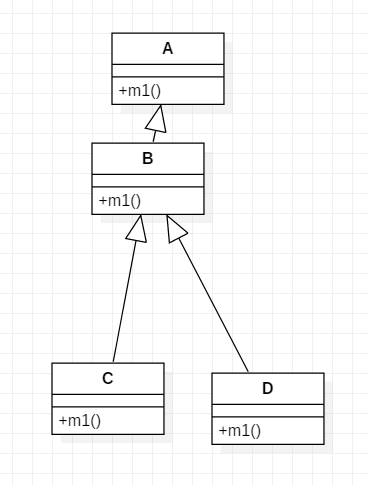
}

**运行截图：**



**第四题**

* **有如图所示的4个类，其继承（泛化）关系如图。编程该4个类。可以自行增加属性和方法。m1方法的功能自定，但必须被子类覆盖，且功能有所不同。**
* **另外设计一个类Test，在该类的main方法中尝试创建ABCD 4个类的对象和实例，实现多态的运行结果**
* **修改上述程序，分别用不同类型（ABCD）的对象(变量)访问不同的实例的方法，并尝试进行类型的强制转换，如果编译器报错，记录并理解错误信息，在作业中截错误图，并说明为什么出错。**

****

源代码：

*// 父类A*

class A {

*// m1方法将在子类中被覆盖*

    public void m1() {

        System.out.println("Method m1 in class A");

    }

}

*// 子类B继承自A*

class B extends *A* {

    public void m1() {

        System.out.println("Method m1 in class B");

    }

*// B类特有方法*

    public void methodB() {

        System.out.println("This is methodB in class B");

    }

}

*// 子类C继承自B*

class C extends *B* {

    @Override

    public void m1() {

        System.out.println("Method m1 in class C");

    }

*// C类特有方法*

    public void methodC() {

        System.out.println("This is methodC in class C");

    }

}

*// 子类D继承自B*

class D extends *B* {

    @Override

    public void m1() {

        System.out.println("Method m1 in class D");

    }

*// D类特有方法*

    public void methodD() {

        System.out.println("This is methodD in class D");

    }

}

*// 测试类Test*

class Test {

    public static void main(String[] args) {

*// 创建A、B、C、D类的对象*

        A a = new A();

        B b = new B();

        C c = new C();

        D d = new D();

*// 调用m1方法展示多态*

        System.out.println("======多态展示分割线======");

        A ref; *// 父类A的引用*

        ref = a;

        ref.m1(); *// 调用A的m1方法*

        ref = b;

        ref.m1(); *// 调用B的m1方法*

        ref = c;

        ref.m1(); *// 调用C的m1方法*

        ref = d;

        ref.m1(); *// 调用D的m1方法*

*// 强制类型转换和类型安全测试*

        System.out.println("======分割线：以下是强制类型转换和类型安全测试======");

        A obj = new B(); *// 父类引用指向子类对象*

        if (obj instanceof B) {

            B bObj = (B) obj; *// 强制转换为B类*

            bObj.methodB(); *// 调用B类的特有方法*

        }

        try {

            C cObj = (C) obj; *// 强制转换为C类，会抛出ClassCastException*

            cObj.methodC();

        } catch (ClassCastException e) {

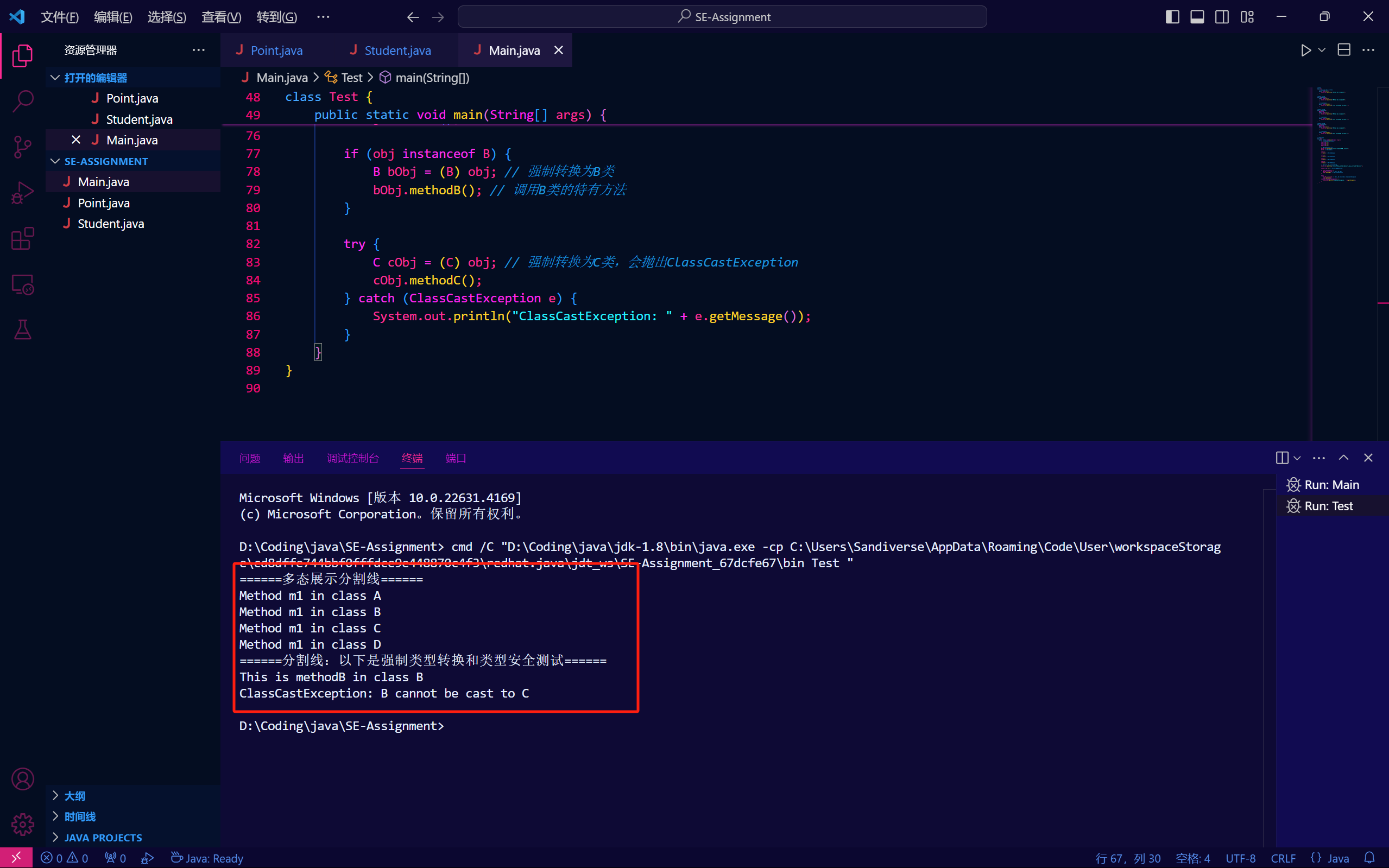
            System.out.println("ClassCastException: " + e.getMessage());

        }

    }

}

运行截图：



在尝试将 B 类的对象强制转换为 C 类时，虽然编译器允许这种转换，但在运行时会抛出 ClassCastException。这是因为 B 类的对象并不属于 C 类的实例，BC之间没有直接的继承关系**。**