

# day04 【面向对象回顾、练习】

---

## 今日内容

---

- 面向对象复习
- 面向对象练习

## 教学目标

---

- ☐ 能够完成面向对象练习1
- ☐ 能够完成面向对象练习2
- ☐ 能够完成面向对象练习3
- ☐ 能够完成面向对象练习4
- ☐ 能够完成面向对象练习5
- ☐ 能够完成面向对象练习6
- ☐ 能够完成面向对象练习7

## 第一章 面向对象复习

---

### 1.1 如何定义类

---

类的定义格式如下:

```
修饰符 class 类名{  
    // 1.成员变量（属性）  
    // 2.成员方法（行为）  
    // 3.构造器（初始化类的对象数据的）  
}
```

例如:

```
public class Student {  
    // 1.成员变量  
    public String name;  
    public char sex; // '男' '女'  
    public int age;  
  
    public void eat() {  
        System.out.println(name + "在吃饭");  
    }  
}
```

### 1.2 如何通过类创建对象

---

```
类名 对象名称 = new 类名();
```

例如:

```
Student stu = new Student();
```

## 1.3 封装

### 封装的步骤

- 1.使用 `private` 关键字来修饰成员变量。
- 2.使用 `public` 修饰getter和setter方法。

### 封装的实现

1. `private`修饰成员变量

```
public class Student {  
    private String name;  
    private int age;  
}
```

2. `public`修饰getter和setter方法

```
public class Student {  
    private String name;  
    private int age;  
  
    public void setName(String n) {  
        name = n;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setAge(int a) {  
        if (a > 0 && a < 200) {  
            age = a;  
        } else {  
            System.out.println("年龄非法!");  
        }  
    }  
  
    public int getAge() {  
        return age;  
    }  
}
```

## 1.4 构造方法

### 构造方法的作用

完成对象数据的初始化。

### 构造方法的格式

```
public class 类名 {  
    修饰符 类名(参数) {  
    }  
}
```

### 构造方法的应用

首先定义一个学生类，代码如下：

```
public class Student {  
    // 1.成员变量  
    public String name;  
    public int age;  
  
    // 2.构造器  
    public Student() {  
        System.out.println("无参数构造器被调用");  
    }  
}
```

接下来通过调用构造器得到两个学生对象。

```
public class CreateStu02 {  
    public static void main(String[] args) {  
        // 创建一个学生对象  
        // 类名 变量名称 = new 类名();  
        Student s1 = new Student();  
        // 使用对象访问成员变量，赋值  
        s1.name = "张三";  
        s1.age = 20 ;  
  
        // 使用对象访问成员变量 输出值  
        System.out.println(s1.name);  
        System.out.println(s1.age);  
  
        Student s2 = new Student();  
        // 使用对象访问成员变量 赋值  
        s2.name = "李四";  
        s2.age = 18 ;  
        System.out.println(s2.name);  
        System.out.println(s2.age);  
    }  
}
```

## 1.5 this关键字的作用

### this关键字的作用

this代表所在类的当前对象的引用（地址值），即代表当前对象。

### this关键字的应用

#### 用于普通的getter与setter方法

this出现在实例方法中，谁调用这个方法（哪个对象调用这个方法），this就代表谁（this就代表哪个对象）。

```
public class Student {
    private String name;
    private int age;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        if (age > 0 && age < 200) {
            this.age = age;
        } else {
            System.out.println("年龄非法!");
        }
    }

    public int getAge() {
        return age;
    }
}
```

#### 用于构造方法中

this出现在构造器中，代表构造器正在初始化的那个对象。

```
public class Student {
    private String name;
    private int age;

    // 无参数构造方法
    public Student() {}

    // 有参数构造方法
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

## 1.6 继承

### 继承格式

通过 `extends` 关键字，可以声明一个子类继承另外一个父类，定义格式如下：

```
class 父类 {  
    ...  
}  
  
class 子类 extends 父类 {  
    ...  
}
```

### 继承好处

1. 提高**代码的复用性**（减少代码冗余，相同代码重复利用）。
2. 使类与类之间产生了关系。

## 1.7 抽象

### 抽象方法

使用 `abstract` 关键字修饰方法，该方法就成了抽象方法，抽象方法只包含一个方法名，而没有方法体。

定义格式：

修饰符 `abstract` 返回值类型 `方法名`（参数列表）；

代码举例：

```
public abstract void run();
```

### 抽象类

如果一个类包含抽象方法，那么该类必须是抽象类。

定义格式：

```
abstract class 类名字 {  
  
}
```

代码举例：

```
public abstract class Animal {  
    public abstract void run();  
}
```

## 1.8 final关键字

---

**final**：不可改变。可以用于修饰类、方法和变量。

- 类：被修饰的类，不能被继承。
- 方法：被修饰的方法，不能被重写。
- 变量：被修饰的变量，不能被重新赋值。

### 修饰类

```
final class 类名 {  
  
}
```

### 修饰方法

```
修饰符 final 返回值类型 方法名(参数列表){  
    //方法体  
}
```

### 修饰变量

```
final 数据类型 变量名
```

## 1.9 static关键字

---

### 修饰变量

static修饰的成员被多个对象共享。

```
static 数据类型 变量名;
```

### 修饰方法

```
修饰符 static 返回值类型 方法名 (参数列表){  
    // 执行语句  
}
```

被static修饰的成员可以并且建议通过**类名直接访问**。

## 1.10 接口

---

接口的内部主要就是**封装了方法**，包含抽象方法（JDK 7及以前），默认方法和静态方法（JDK 8）。

## 定义格式

```
public interface 接口名称 {  
    // 抽象方法  
    // 默认方法  
    // 静态方法  
}
```

## 类实现接口

```
class 类名 implements 接口名 {  
    // 重写接口中抽象方法【必须】  
    // 重写接口中默认方法【可选】  
}
```

## 1.11 匿名对象

**什么是匿名对象：就是指"没有名字"的对象。**

有名字的对象：

```
Scanner sc = new Scanner(System.in);
```

匿名对象：

```
new Scanner(System.in);
```

### 匿名对象的使用情景

链式编程：

```
public class Demo{  
    public static void main(String[] args){  
        //如果我们只需要从控制台接收一次数据-一个年龄值  
        System.out.println("请输入你的年龄：");  
        int age = new Scanner(System.in).nextInt();//匿名对象  
        System.out.println("你的年龄是：" + age);  
    }  
}
```

作为实参：

```
public class Demo{
    public static void main(String[] args){
        int age = getAge(new Scanner(System.in)); //匿名对象
        System.out.println("你的年龄是: " + age);
    }
    //以下方法接收一个Scanner对象，用于从控制台接收一个年龄值并返回
    public static int getAge(Scanner sc){
        System.out.println("请输入你的年龄: ");
        int age = sc.nextInt();
        return age
    }
}
```

## 第二章 面向对象练习

### 练习1

#### 需求

定义一个扑克类Card

属性：

- 花色
- 点数

构造方法：

- 满参构造方法

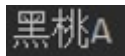
成员方法：

- showCard方法：打印牌面信息

定义测试类

在main方法中使用满参构造创建Card对象card，并调用showCard方法

代码实现，效果如图所示：



#### 案例代码

Card类



```

public class Card {
    private String ds; // 点数
    private String hs; // 花色

    public Card() {
    }

    public void showCard() {
        System.out.println(ds + hs);
    }
}

```

## 测试类

```

public class Test01 {
    public static void main(String[] args) {
        Card card = new Card();
        card.ds = "A";
        card.hs = "黑桃";
        card.showCard();
    }
}

```

## 练习2

### 需求

- 定义两个类，经理类Manager，程序员类Coder
- Coder类：
  - 属性：姓名，工号，薪资
  - 构造方法：无参构造方法
  - 成员方法：
    - intro方法：打印姓名，工号信息
    - showSalary方法：打印薪资信息
    - work方法：打印"正在努力写代码...."
- Manager类：
  - 属性：姓名，工号，薪资，奖金
  - 构造方法：无参构造方法
  - 成员方法：
    - intro方法：打印姓名，工号信息
    - showSalary方法：打印薪资和奖金信息
    - work方法：打印"正在努力的做着管理工作,分配任务,检查员工提交上来的代码....."
- 定义测试类，创建Manager对象，创建Coder对象，并测试
- 代码实现，效果如图所示：

```
经理姓名:James  
工号:9527  
基本工资为15000,奖金为3000  
正在努力的做着管理工作,分配任务,检查员工提交上来的代码.....  
=====  
程序员姓名:Kobe  
工号:0025  
基本工资为10000,奖金无  
正在努力写代码.....
```

## 案例代码

Coder类

```
public class Coder {  
    private String name;  
    private String id;  
    private int salary; // 基本工资  
  
    public Coder() {  
    }  
  
    public void showSalary() {  
        System.out.println("基本工资为" + salary + ",奖金无");  
    }  
  
    public void intro() {  
        System.out.println("程序员姓名:" + name);  
        System.out.println("工号:" + id);  
    }  
  
    public void work() {  
        System.out.println("正在努力写代码....");  
    }  
}
```

Manager类

```
public class Manager {  
    private String name;  
    private String id;  
    private int salary; // 基本工资  
    private int bouns; // 奖金  
  
    public Manager() {  
    }  
  
    public void showSalary() {  
        System.out.println("基本工资为" + salary + ",奖金为" + bouns);  
    }  
  
    public void intro() {  
        System.out.println("经理姓名:" + name);  
    }  
}
```

```

        System.out.println("工号:" + id);
    }

    public void work() {
        System.out.println("正在努力的做着管理工作,分配任务,检查员工提交上来的代
码.....");
    }
}

```

## 测试类

```

public class Test02 {
    public static void main(String[] args) {
        Manager m = new Manager();
        m.name = "James";
        m.id = "9527";
        m.salary = 15000;
        m.bouns = 3000;
        m.intro();
        m.showSalary();
        m.work();

        System.out.println("=====");
        Coder c = new Coder();
        c.name = "Kobe";
        c.id = "0025";
        c.salary = 10000;
        c.intro();
        c.showSalary();
        c.work();
    }
}

```

## 练习3

### 需求

请使用继承定义以下类:

程序员(Coder)

成员变量: 姓名, 年龄

成员方法: 吃饭, 睡觉, 敲代码

老师(Teacher)

成员变量: 姓名, 年龄

成员方法: 吃饭, 睡觉, 上课

将程序员和老师中相同的内容(姓名, 年龄, 吃饭, 睡觉)抽取到父类Person中

效果如图所示:

马化腾吃饭  
马化腾睡觉  
马化腾敲代码

-----  
马云吃饭  
马云睡觉  
马云上课

## 案例代码

Person类

```
public class Person {  
    String name;  
    int age;  
  
    public void eat() {  
        System.out.println(name + "吃饭");  
    }  
  
    public void sleep() {  
        System.out.println(name + "睡觉");  
    }  
}
```

Coder类

```
public class Coder extends Person {  
  
    // 敲代码  
    public void coding() {  
        System.out.println(name + "敲代码");  
    }  
}
```

Teacher类

```
public class Teacher extends Person {  
  
    public void teach() {  
        System.out.println(name + "上课");  
    }  
}
```

测试类

```
public class Test03 {
```

```

public static void main(String[] args) {
    // 创建Coder对象,并设置成员变量的值
    Coder c = new Coder();
    c.name = "马化腾";
    c.age = 45;
    // 调用Coder对象的eat()方法
    c.eat();
    // 调用Coder对象的sleep()方法
    c.sleep();
    // 调用Coder对象的coding()方法
    c.coding();

    System.out.println("-----");
    // 创建Teacher对象,并设置成员变量的值
    Teacher t = new Teacher();
    t.name = "马云";
    t.age = 50;
    // 调用Teacher对象的eat()方法
    t.eat();
    // 调用Teacher对象的sleep()方法
    t.sleep();
    // 调用Teacher对象的teach()方法
    t.teach();
}
}

```

## 练习4

### 需求

请使用 继承, 抽象方法, 抽象类 定义以下类:

#### 1. 经理

成员变量: 工号, 姓名, 工资

成员方法: 工作(管理其他人), 吃饭(吃鱼)

#### 2. 厨师

成员变量: 工号, 姓名, 工资

成员方法: 工作(炒菜), 吃饭(吃肉)

效果如图所示:

```

工号为:m110, 姓名为:老王工资为:10000.0的经理在吃鱼
工号为:m110, 姓名为:老王工资为:10000.0的经理在工作, 管理其他人
工号为:c110, 姓名为:小王工资为:6000.0的厨师在吃肉
工号为:c110, 姓名为:小王工资为:6000.0的厨师在工作, 炒菜

```

### 案例代码

抽象的Employee类

```

abstract class Employee {

```

```

// 工号属性,姓名属性,工资属性
private String id;
private String name;
private double salary;

public Employee() {
}

public Employee(String id, String name, double salary) {
    this.id = id;
    this.name = name;
    this.salary = salary;
}

// 抽象的工作方法
public abstract void work();

// 抽象的吃饭方法
public abstract void eat();

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public double getSalary() {
    return salary;
}

public void setSalary(double salary) {
    this.salary = salary;
}
}

```

## Manager类

```

// 经理继承员工,重写工作方法,和吃饭方法
class Manager extends Employee {
    public Manager() {
    }

    public Manager(String id, String name, double salary) {
        super(id, name, salary);
    }
}

```

```

        @Override
        public void work() {
            System.out.println("工号为:" + getId() + ",姓名为:" + getName() + "工资为:"
+ getSalary() + "的经理在工作,管理其他人");
        }

        @Override
        public void eat() {
            System.out.println("工号为:" + getId() + ",姓名为:" + getName() + "工资为:"
+ getSalary() + "的经理在吃鱼");
        }
    }
}

```

## Cook类

```

class Cook extends Employee {
    public Cook() {
    }

    public Cook(String id, String name, double salary) {
        super(id, name, salary);
    }

    @Override
    public void work() {
        System.out.println("工号为:" + getId() + ",姓名为:" + getName() + "工资为:"
+ getSalary() + "的厨师在工作,炒菜");
    }

    @Override
    public void eat() {
        System.out.println("工号为:" + getId() + ",姓名为:" + getName() + "工资为:"
+ getSalary() + "的厨师在吃肉");
    }
}

```

## 测试类

```

public class Test04 {
    public static void main(String[] args) {
        // 创建Manager对象
        Manager m = new Manager("m110", "老王", 10000);
        // 调用Manager对象的eat方法
        m.eat();
        // 调用Manager对象的work方法
        m.work();

        // 创建Cook对象
        Cook c = new Cook("c110", "小王", 6000);
        // 调用Cook对象的eat方法
        c.eat();
        // 调用Cook对象的work方法
    }
}

```

```
        c.work();
    }
}
```

## 练习5

### 需求

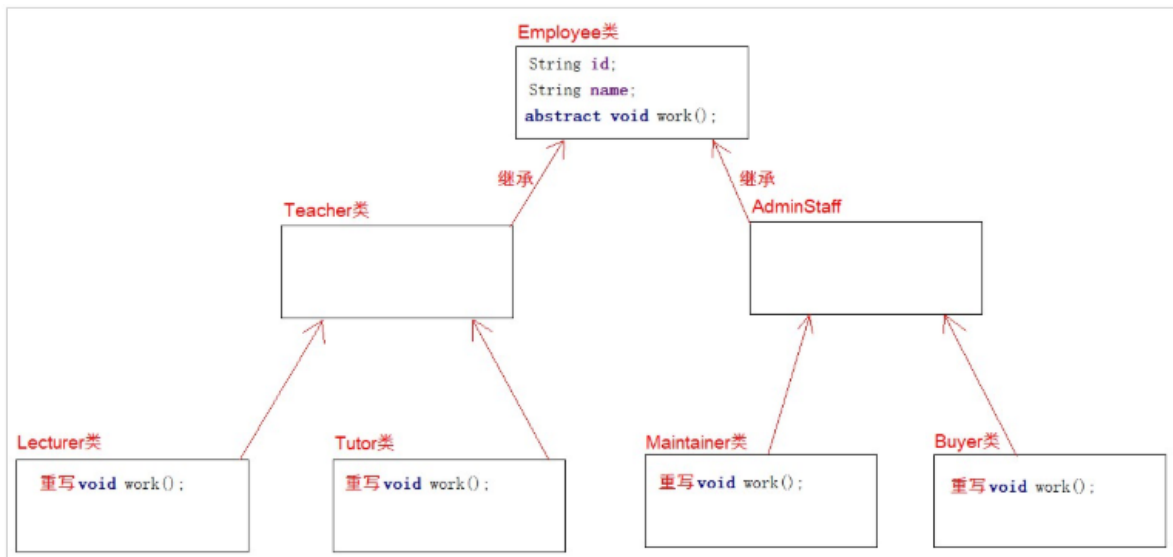
请使用 继承, 抽象方法, 抽象类 定义以下类:

1. 在传智播客有很多员工(Employee), 按照工作内容不同分教研部员工(Teacher)和行政部员工(AdminStaff)
2. 教研部根据教学的方式不同又分为讲师(Lecturer)和助教(Tutor)
3. 行政部根据负责事项不同, 又分为维护专员(Maintainer), 采购专员(Buyer)
4. 公司的每一个员工都编号, 姓名和其负责的工作
5. 每个员工都有工作的功能, 但是具体的工作内容又不一样, 在向上抽取的时候定义为抽象方法

效果如图所示:

```
工号为 666 的讲师 傅红雪 在讲课
工号为 668的助教 顾棋 在帮助学生解决问题
工号为 686 的维护专员 庖丁 在解决不能共享屏幕问题
工号为 888 的采购专员 景甜 在采购音响设备
```

类之间的关系如下:



### 案例代码

Employee类

```
/*
1. 定义抽象类员工类(Employee)
a) 成员变量: 工号(id), 姓名(name)
b) 抽象方法: void work();
c) 提供无参和带参的构造方法以及setters和getters
```



```

*/
public abstract class Employee {
    // a)成员变量: 工号(id),姓名(name)
    private String id;
    private String name;

    // b)抽象方法: void work();
    public abstract void work();

    // c)提供无参和带参的构造方法以及setters和getters
    public Employee() {
        super();
    }

    public Employee(String id, String name) {
        super();
        this.id = id;
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```

## Teacher类

```

/*
2. 定义抽象类教研部员工 (Teacher) 类继承员工类(Employee)
*/
public abstract class Teacher extends Employee{
    // a)提供无参和带参的构造方法
    public Teacher() {
        super();
    }

    public Teacher(String id, String name) {
        super(id, name);
    }
}

```

## AdminStaff类

```
/*
3.定义抽象类行政部员工(AdminStaff)类继承员工类(Employee)
*/
public abstract class AdminStaff extends Employee {
// a)提供无参和带参的构造方法
    public AdminStaff() {
        super();
    }

    public AdminStaff(String id, String name) {
        super(id, name);
    }
}
```

## Lecturer类

```
/*
4.定义讲师(Lecturer)类继承研部员工(Teacher)类
*/
public class Lecturer extends Teacher {
// a)提供无参和带参的构造方法
    public Lecturer() {
        super();
    }

    public Lecturer(String id, String name) {
        super(id, name);
    }

// b)实现抽象方法: void work();
//      输出格式: 工号为 666 的讲师 傅红雪 在讲课
    public void work() {
        System.out.println("工号为 "+getId()+" 的讲师 "+getName()+" 在讲课");
    }
}
```

## Tutor类

```
/*
5.定义助教(Tutor)类继承研部员工(Teacher)类
*/
public class Tutor extends Teacher{
// a)提供无参和带参的构造方法
    public Tutor() {
        super();
    }

    public Tutor(String id, String name) {
        super(id, name);
    }

// b)实现抽象方法: void work();
```

```
// i.输出格式：工号为 668的助教 顾棋 在帮助学生解决问题
public void work() {
    System.out.println("工号为 "+getId()+"的助教 "+getName()+" 在帮助学生解决问题");
}
}
```

## Maintainer类

```
/*
6.定义维护专员(Maintainer)类继承行政部员工(AdminStaff)类
*/
public class Maintainer extends AdminStaff {
// a)提供无参和带参的构造方法
    public Maintainer() {
        super();
    }

    public Maintainer(String id, String name) {
        super(id, name);
    }
// b)实现抽象方法：void work();
// i.输出格式：工号为 686 的维护专员 庖丁 在解决不能共享屏幕问题
    public void work() {
        System.out.println("工号为 "+getId()+" 的维护专员 "+getName()+" 在解决不能共享屏幕问题");
    }
}
```

## Buyer类

```
/*
7.定义采购专员(Buyer) 类继承行政部员工(AdminStaff)类
*/
public class Buyer extends AdminStaff {
// a)提供无参和带参的构造方法
    public Buyer() {
        super();
    }

    public Buyer(String id, String name) {
        super(id, name);
    }
// b)实现抽象方法：void work();
// 输出格式： 工号为 888 的采购专员 景甜 在采购音响设备
    public void work() {
        System.out.println("工号为 "+getId()+" 的采购专员 "+getName()+" 在采购音响设备");
    }
}
```

## 测试类

```
/*
8. 定义测试类Test
*/
public class Test5 {
    public static void main(String[] args) {
        // a) 创建讲师对象l, 把工号赋值为666, 姓名赋值为"傅红雪"
        Lecturer l = new Lecturer("666", "傅红雪");
        // b) 调用讲师对象l的工作方法
        l.work();

        // c) 创建助教对象 t, 把工号赋值为668, 姓名赋值为"顾棋"
        Tutor t = new Tutor("668", "顾棋");
        // d) 调用助教对象t的工作方法
        t.work();

        // e) 创建维护专员对象 m, 把工号赋值为686, 姓名赋值为"庖丁"
        Maintainer m = new Maintainer("686", "庖丁");
        // f) 调用维护专员对象m的工作方法
        m.work();

        // g) 创建采购专员对象 b, 把工号赋值为888, 姓名赋值为"景甜"
        Buyer b = new Buyer("888", "景甜");
        // h) 调用采购专员对象b的工作方法
        b.work();
    }
}
```

## 练习6

### 需求

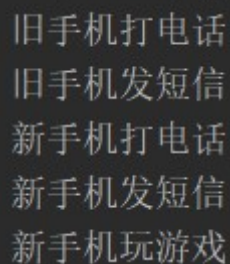
请使用 继承, 接口, 定义以下类:

两个手机类OldPhone和NewPhone都有call()和sendMessage()方法. 定义接口Play, Play中有一个抽象的玩游戏的方法playGame(), NewPhone实现Play接口有玩游戏的功能;

要求:

分别测试OldPhone和NewPhone的call()和sendMessage()方法, 再测试新手机palyGame()的方法

效果如图所示:



```
旧手机打电话
旧手机发短信
新手机打电话
新手机发短信
新手机玩游戏
```

### 案例代码

Play接口

```
// 1.定义接口Play
interface Play {
    // 2.在Play中定义一个抽象的玩游戏的方法playGame()
    public abstract void playGame();
}
```

## OldPhone类

```
// 3.定义OldPhone类
class OldPhone {
    // 4.在OldPhone类中定义call()和sendMessage()方法
    public void call() {
        System.out.println("旧手机打电话");
    }

    public void sendMessage() {
        System.out.println("旧手机发短信");
    }
}
```

## NewPhone类

```
// 5.定义NewPhone类,继承OldPhone,实现Play接口
class NewPhone extends OldPhone implements Play {
    @Override
    public void playGame() {
        System.out.println("新手机玩游戏");
    }

    @Override
    public void call() {
        System.out.println("新手机打电话");
    }

    @Override
    public void sendMessage() {
        System.out.println("新手机发短信");
    }
}
```

## 测试类

```
public class Test06 {
    public static void main(String[] args) {
        // 6.创建旧手机对象
        OldPhone oldPhone = new OldPhone();
        // 7.使用旧手机打电话
        oldPhone.call();
        // 8.使用旧手机发信息
        oldPhone.sendMessage();
    }
}
```

```

// 9.创建新手机对象
NewPhone newPhone = new NewPhone();
// 10.使用新手机打电话
newPhone.call();
// 11.使用新手机发信息
newPhone.sendMessage();
// 12.使用新手机玩游戏
newPhone.playGame();
    }
}

```

## 练习7

### 需求

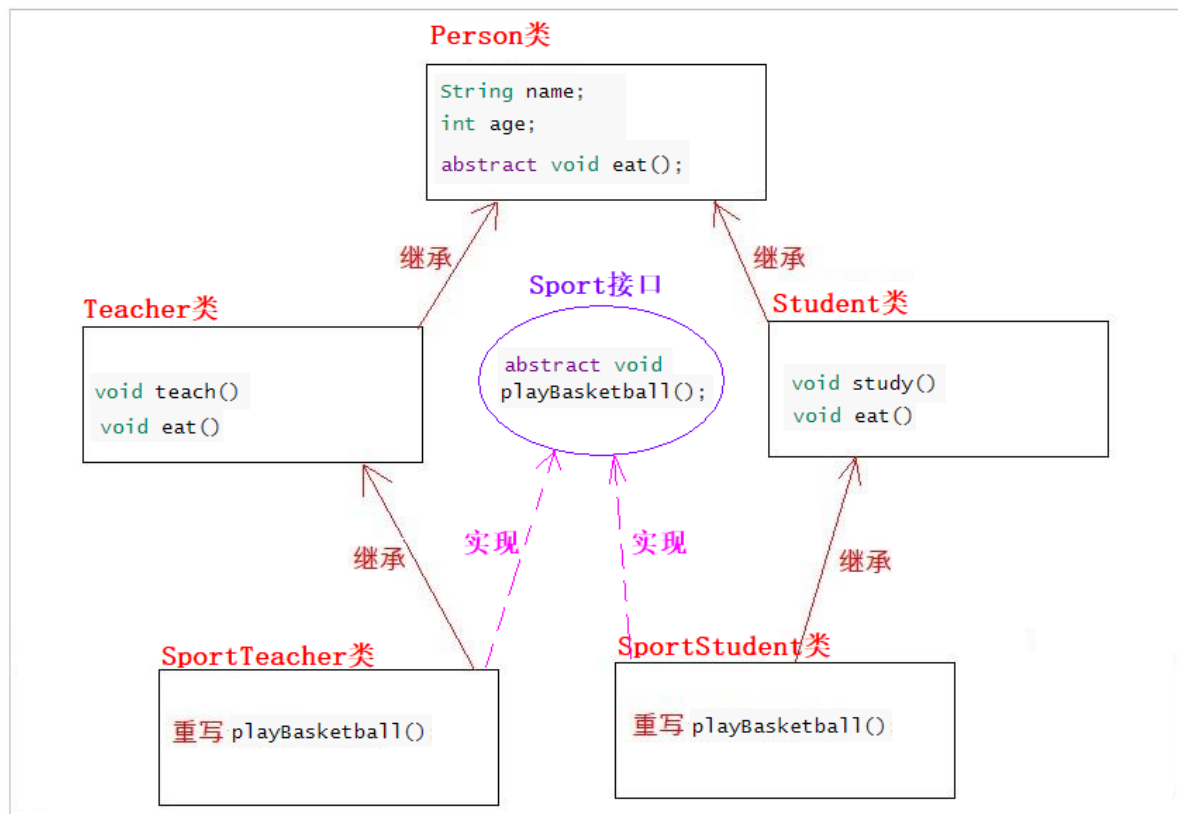
请使用 继承，接口，定义以下类：

学生都有年龄和姓名属性，有吃饭(学生餐)和学习方法，但是有部分学生会打篮球  
 老师都有年龄和姓名属性，有吃饭(工作餐)和讲课方法，但是有部分老师会打篮球  
 定义一个方法模拟去打篮球，只要会打篮球的人都可以传入。（提示通过在测试类中定义一个方法参数为接口）

效果如图所示：

年龄为35岁 大姚 的老师在打篮球  
 年龄为21岁 王中王 的学生在打篮球

类和接口之间的关系如下：



### 案例代码

## 抽象的Person类

```
// 1. 定义Person类
abstract class Person {
    // 2. Person类包含name, age属性和抽象的eat方法
    private String name;
    private int age;

    public abstract void eat();

    public Person() {
        super();
    }

    public Person(String name, int age) {
        super();
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

## Sport接口

```
// 3. 定义Sport接口, 包含playBasketball方法
interface Sport {
    public abstract void playBasketball();
}
```

## Teacher类

```
// 4. 定义Teacher类继承Person类, 重写抽象方法eat()
class Teacher extends Person {
    public void eat() {
        System.out.println();
    }
}
```

```

    public void teach() {
        System.out.println(getName() + "老师在讲课");
    }

    public Teacher() {
        super();
    }

    public Teacher(String name, int age) {
        super(name, age);
    }
}

```

## SportTeacher类

```

// 5.定义SportTeacher类继承Teacher类,实现Sport接口,重写Sport接口中的playBasketball方法
class SportTeacher extends Teacher implements Sport {
    public void playBasketball() {
        System.out.println("年龄为" + getAge() + "岁 " + getName() + " 的老师在打篮球");
    }

    public SportTeacher() {
        super();
    }

    public SportTeacher(String name, int age) {
        super(name, age);
    }
}

```

## Student类

```

// 6.定义Student类继承Person类,重写抽象方法eat()
class Student extends Person {
    public void eat() {
        System.out.println("年龄" + getAge() + "岁的 " + getName() + " 在吃学生餐");
    }

    public void study() {
        System.out.println(getName() + "学生在学习");
    }

    public Student() {
        super();
    }

    public Student(String name, int age) {
        super(name, age);
    }
}

```



## SportStudent类

```
// 7.定义SportStudent类继承Student类,实现Sport接口,重写Sport接口中的playBasketball方法
class SportStudent extends Student implements Sport {
    public SportStudent() {
    }

    public SportStudent(String name, int age) {
        super(name, age);
    }

    public void playBasketball() {
        System.out.println("年龄为" + getAge() + "岁 " + getName() + " 的学生在打篮球");
    }
}
```

## 测试类

```
public class Test07 {
    public static void main(String[] args) {
        // 9.在main方法中创建普通的老师t1,姓名为马云,年龄为45岁
        Teacher t1 = new Teacher("马云", 45);
        // 10.在main方法中创建会打篮球的老师t2,姓名为大姚,年龄为35岁
        SportTeacher t2 = new SportTeacher("大姚", 35);

        // 11.在main方法中创建普通的学生s1,姓名为小王,年龄为20
        Student s1 = new Student("小王", 20);
        // 12.在main方法中创建会打篮球的学生s2,姓名为王中王,年龄为21
        SportStudent s2 = new SportStudent("王中王", 21);

        // 13.在main方法中调用goToSport方法.传入t1,t2,s1,s2四个对象.我们会发现只有实现
        Sport接口的对象才能传入
        // goToSport(t1); // 没有实现Sport接口不能传入
        goToSport(t2);
        // goToSport(s1); // 没有实现Sport接口不能传入
        goToSport(s2);
    }

    // 8.在测试类中定义静态的goToSport方法,参数为Sport接口类型
    public static void goToSport(Sport s){
        // 在goToSport方法中调用传入参数的playBasketball方法
        s.playBasketball();
    }
}
```