**每日作业卷**

**就业班JavaSE--day03【接口、多态】**

# 关卡1

## 训练案例1

### 训练考核知识点

接口

### 训练描述

请阐述接口中成员的特点

### 操作步骤描述

1. 接口中成员方法的特点是什么 ? 只描述所应该具备的方法，并没有具体实现

2. 接口中变量特点是什么? 接口中无法定义普通的成员变量，只能定义常量

3. 接口可以直接创建对象吗? 为什么? 不可以，接口本身是用来抽象的，实例化接口没有意义

4. 实现接口的类可以创建对象吗 ? 可以

## 训练案例2

### 训练考核知识点

接口

### 训练描述

请阐述接口的特点.

### 操作步骤描述

1. Java中接口可以继承吗 ? 可以
2. Java支持多继承吗 ? 不支持
3. Java支持继承父类的同时,再实现多个接口吗 ? 可以
4. Java 允许父类的与实现接口的有相同的功能吗 ? 可以

## 训练案例3

### 训练考核知识点

接口

抽象类

### 训练描述

请阐述接口与抽象的区别

### 操作步骤描述

1. 接口与抽象类的相同点是什么 ? 都不能被实例化

2. 接口与抽象类的区别是什么 ? 抽象类中可以有非抽象方法

接口中全部为抽象方法

3. 二者如何选用 ?  接口抽象的主要是行为，或者是功能， 抽象类抽象是行为和数据。

## 训练案例4

### 训练考核知识点

多态

### 训练描述

请阐述一下你对多态理解

### 操作步骤描述

1. 多态前提条件是什么? 是必须有子父类关系或者类实现接口关系
2. 多态有哪些注意事项? 不能重写父类的私有方法
3. 多态的好处和弊端是分别是什么? 利：提高了代码的扩展性，弊：前期定义的内容不能使用后期子类的特有内容
4. 什么是多态的向上转型 ? 将子类对象转为父类对象
5. 什么是多态的向下转型, 为什么要向下转型 , 向下转型需要注意什么?

父类对象转为子类对象，向下转型的前提是父类对象指向的是子类对象

## 训练案例5

### 训练考核知识点

接口定义

类实现接口

### 训练描述

请使用代码描述：

使用6000元的雷神电脑,编程和玩游戏

使用3000元的华为手机打电话和玩游戏

### 操作步骤描述

1. 定义玩游戏(PlayGame)接口
   1. 声明抽象 playGame()
2. 定义手机类(Phone)类,实现玩游戏(PlayGame)接口
   1. 属性: 品牌(brand),价格(price)
   2. 功能:
      1. 特有功能: 打电话(call())
         1. 输出格式: 在使用3000 元 华为手机打电话
      2. 实现接口功能: 玩游戏()
         1. 输出格式: 在使用3000 元 华为手机发短信
   3. 提供 空参,有参构造;setters和getters方法
3. 定义电脑(Computer)类实现玩游戏接口(PlayGame())
   1. 属性: 品牌(brand)和价格(price)
   2. 功能:
      1. 实现playGame()方法
         1. 输出格式: 使用6000元的雷神电脑玩游戏
      2. 特有功能: 编码(coding)
         1. 输出格式: 使用6000元的雷神电脑开发JavaEE应用
   3. 提供空参,有参构造;setters和getters方法
4. 定义测试类Test
   1. 定义main方法
   2. 在main方法中
      1. 创建手机对象 p,品牌赋值为华为,价格赋值为3000
      2. 调用手机对象 p的玩游戏功能
      3. 调用手机对象 p的打电话功能
      4. 创建电脑对象 c,品牌赋值为雷神,价格赋值为6000
      5. 调用电脑对象c的玩游戏功能
      6. 调用电脑对象c的编程功能

测试类：

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Phone phone = **new** Phone("华为", 3000);

phone.playGame();

phone.call();

Computer computer = **new** Computer("雷神", 6000);

computer.playGame();

computer.coding();

}

}

PlayGame接口：

**public** **interface** PlayGame {

**public** **abstract** **void** playGame();

}

Phone类：

**public** **void** call(){

System.***out***.println("在使用"+price+"元"+brand+"手机打电话");

}

@Override

**public** **void** playGame() {

System.***out***.println("在使用"+price+"元"+brand+"手机发短信");

}

Computer类：

**public** **void** coding(){

System.***out***.println("在使用"+price+"元"+brand+"电脑开发JavaEE应用");

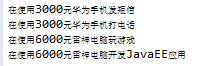
}

@Override

**public** **void** playGame() {

System.***out***.println("在使用"+price+"元"+brand+"电脑玩游戏");

}



## 训练案例6

### 训练考核知识点

继承

接口

类实现接口

多态(向上转型)

向下转型

### 训练描述

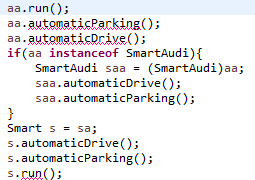
请使用代码描述:

奥迪车(Audi)都具有跑的功能，但是智能奥迪车(SmartAudi)除了具有跑的功能外，还具有自动泊车(automaticParking)和无人驾驶(automaticDrive)的功能！

要求:在测试类中创建Audi车对象和智能奥迪车对象,分别调用各自方法; 然后测试向上转型(转换为父类类型和实现的接口类型)和向下转型.

### 操作步骤描述

1. 定义奥迪车类(Audi)
   1. 成员方法: 跑(run())
      1. 输出格式: 奥迪车在跑
2. 定义智能接口(Smart)
   1. 抽象方法: 自动泊车(automaticParking)和无人驾驶(automaticDrive)
3. 定义智能奥迪车类(SmartAudi) 继承Audi实现Smart接口
   1. 成员方法
      1. 实现automaticParking方法
         1. 输出:智能奥迪车在自动泊车
      2. 实现automaticDrive方法
         1. 输出:智能奥迪车在无人驾驶
4. 定义测试类Test
   1. 提供main方法
   2. 在main方法中
      1. 创建Audi车对象 a,调用跑方法
      2. 创建SmartAudi车对象 sa,调用跑,自动泊车,自动驾驶方法
      3. 定义Audi类型的变量 aa 把sa赋值aa; 测试aa只能调用run方法,不能调用其他方法
      4. 判断如果aa是SmartAudi的实例对象, 把aa强制转换为saa;使用saa调用自动泊车和自动驾驶方法
      5. 定义Smart类型的变量 s,把sa赋值给s,测试只能调用自动泊车和自动驾驶方法,不能调用run方法.



测试类：

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Audi a = **new** Audi();

a.run();

SmartAudi sa = **new** SmartAudi();

sa.automaticParking();

sa.automaticDrive();

Audi aa = sa;

aa.run();

**if**(aa **instanceof** SmartAudi){

SmartAudi saa = (SmartAudi)aa;

saa.automaticDrive();

saa.automaticParking();

}

Smart s = sa;

s.automaticDrive();

s.automaticParking();

}

}

Audi类

**public** **class** Audi {

**public** **void** run(){

System.***out***.println("奥迪车在跑");

}

}

Smart接口：

**public** **interface** Smart {

**public** **abstract** **void** automaticParking();

**public** **abstract** **void** automaticDrive();

}

SmartAudi类：

**public** **class** SmartAudi **extends** Audi **implements** Smart{

@Override

**public** **void** automaticParking() {

System.***out***.println("智能奥迪车在自动泊车");

}

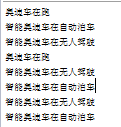
@Override

**public** **void** automaticDrive() {

System.***out***.println("智能奥迪车在无人驾驶");

}

}



# 关卡2

## 训练案例1

### 训练考核知识点

继承

接口定义

类实现接口

### 训练描述

请使用代码描述：

100元诺基亚手机(Phone)可以打电话(call)和发短信(sendMessage)

3000元的华为智能手机可以打电话,发短信和玩游戏(playGame)

2000元的iPad 平板,可以听歌(listenMusic)和玩游戏.

要求: 把手机的共性提取到父类中,特性的功能放到接口.

### 操作步骤描述

1. 定义玩游戏(PlayGame)接口
   1. 声明抽象方法 playGame()
2. 定义手机类(Phone)类
   1. 属性: 品牌(brand),价格(price)
   2. 功能:
      1. 打电话(call(String name))
         1. 输出格式: 在使用100 元 诺基亚 手机 给 景甜 打电话
      2. 发短信(sendMessage(String name))
         1. 输出格式: 在使用100 元 诺基亚 手机 给 景甜 发短信
   3. 提供 空参,有参构造;setters和getters方法
3. 定义智能手机(SmartPhone)类继承手机类(Phone),实现玩游戏接口(PlayGame())
   1. 实现playGame()方法
      1. 在使用3000元的 华为 手机玩游戏
   2. 提供空参,有参构造
4. 定义平板(Pad)类,实现PlayGame接口
   1. 属性:品牌(brand),价格(price)
   2. 成员方法
      1. 实现playGame()方法
         1. 输出格式: 在使用2000元的iPad平板玩游戏
      2. 特有方法: 听音乐(listenMusic)
         1. 输出格式: 在使用2000元的iPad平板听音乐
   3. 提供空参,有参构造,setters和getters方法
5. 定义测试类Test
   1. 创建手机对象 njy,品牌赋值为诺基亚,价格赋值为100
   2. 调用手机对象 njy的打印电话和发短信功能,传入 景甜
   3. 创建智能手机对象 sp,品牌赋值为华为,价格赋值为3000
   4. 调用智能数据对象sp的打电话,发短信功能都传入林思意, 调用玩游戏功能
   5. 创建平板对象 p,品牌赋值为iPad,价格赋值为2000
   6. 调用平板p对象的听歌和玩游戏功能.

测试类：

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Phone njy = **new** Phone("诺基亚", 100);

njy.call("景甜");

SmartPhone sp = **new** SmartPhone("华为", 3000);

sp.call("林思意");

sp.playGame();

Pad p = **new** Pad("ipad", 2000);

p.listenMusic();

p.playGame();

}

}

PlayGame接口：

**public** **interface** PlayGame {

**public** **abstract** **void** playGame();

}

Phone类：

**public** **void** call(String name){

System.***out***.println("在使用"+price+"元"+brand+"手机给"+name+"打电话");

}

**public** **void** sendMessage(String name){

System.***out***.println("在使用"+price+"元"+brand+"手机给"+name+"发短信");

}

SmartPhone类：

@Override

**public** **void** playGame() {

System.***out***.println("在使用"+getPrice()+"元"+getBrand()+"的手机玩游戏");

}

## 训练案例2

### 训练考核知识点

抽象类

接口

继承

多态

类名作为参数类型

接口作为参数类型

### 训练描述

请使用代码描述:

所有的学生都有年龄和姓名,都会吃饭(学生餐)和学习,但是有部分学生会打篮球

所有的老师都有年龄和姓名,都会吃饭(工作餐)和讲课,但有部分老师会打篮球

去打篮球,会打篮球的老师和学生可以. (提示通过在测试类中的静态方法实现)

去食堂吃饭,所有人都可以去(提示通过在测试类中的静态方法实现)

### 操作步骤描述

1. 定义抽象人类(Person)
   1. 属性: 姓名(name),年龄(age)
   2. 抽象方法: eat()
   3. 提供:无参,有参构造方法和setters和getters方法
2. 定义运动接口(Sport)
   1. 抽象方法: playBasketball
3. 定义老师类(Teacher)继承Person类
   1. 成员方法:: 实现抽象方法eat()
      1. 输出格式: 年龄为30岁周晓斌老师,正在吃工作餐
   2. 提供: 无参,有参构造方法
4. 定义运动老师类(SportTeacher)继承老师类(Teacher)实现运动接口
   1. 成员方法: 实现playBasketball的抽象方法
      1. 输出格式: 年龄为30岁 周晓斌 老师在打篮球
   2. 提供: 无参,有参构造方法
5. 定义学生类(Student),继承人类(Person)
   1. 成员方法:现抽象方法eat()
      1. 输出格式: 年龄18岁的 叶知秋 在吃学生餐
   2. 提供: 无参,有参构造方法
6. 定义运动学生类(SportStudent类),继承Student类,实现Sport接口
   1. 成员方法: 实现playBasketball的抽象方法
      1. 输出格式: 年龄为18岁 叶知秋 学生在打篮球
   2. 提供: 无参,有参构造方法
7. 定义测试类Test
   1. 提供静态方法去运动goToSport(Sport s) ,在方法中调用s的打篮球功能
      1. 思考这里为什么要传入Sport接口类型
   2. 提供静态方法去餐馆goToRestaurant(Person p),在方法中调用p对象的吃方法
      1. 思考这里为什么要传入顶层父类类型
   3. 提供main方法
   4. 在main方法中
      1. 创建运动老师对象sp,年龄赋值为30,姓名赋值为周晓斌
      2. 创建运动学生对象ss,年龄赋值为18,姓名赋值为叶知秋
      3. 调用去运动goToSport方法传入对象sp
      4. 调用去运动goToSport方法,传入对象ss
      5. 调用去食堂goToRestaurant方法,传入sp
      6. 调用去食堂goToRestaurant方法,传入ss

测试类：

**public** **class** Test {

**public** **static** **void** main(String[] args) {

SportTeacher sp = **new** SportTeacher("周晓斌", 30);

SportStudent ss= **new** SportStudent("叶知秋", 18);

*goToSport*(sp);

*goToSport*(ss);

*goToRestaurant*(sp);

*goToRestaurant*(ss);

}

**public** **static** **void** goToSport(Sport s) {

s.playBasketball();

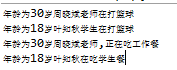
}

**public** **static** **void** goToRestaurant(Person p) {

p.eat();

}

}



## 训练案例3

### 训练考核知识点

抽象类

继承

接口

接口实现

多态

父类名作为参数类型

接口名作为参数类型

向下转型

### 训练描述

请用代码描述:

在动物园每一个动物都有颜色和年龄,都会吃

所有老虎都吃肉,但是有的老虎会表演(钻火圈)

所有的大猩猩都吃香蕉,但有大猩猩会表演(骑自行车)

所有的鹦鹉都吃小米和说话,但是有的鹦鹉会表演(过跷跷板)

在动物园中管理员有年龄和姓名,他会喂动物和组织会表演的动物进行演出,如果演出时鹦鹉,还会让鹦鹉说你好

### 操作步骤描述

1. 定义动物类(Animal)
   1. 属性:颜色(color),年龄(age)
   2. 抽象方法: 吃(eat)
   3. 提供: 空参,带参构造;setters和getters方法
2. 定义表演者接口(Actor)
   1. 抽象方法: play()
3. 定义老虎类(Tiger),继承动物类(Animal)
   1. 实现抽象方法: 吃(eat)
      1. 输出格式: 2岁的黄色老虎吃肉
   2. 提供: 空参,带参构造;
4. 定义会表演的老虎类(ActedTiger),继承老虎类(Tiger),实现表演者接口
   1. 实现抽象方法表演(play)
      1. 输出格式: 2岁的黄色老虎在表演钻火圈
   2. 提供: 空参,带参构造;
5. 定义大猩猩类(Gorilla),继承动物类(Animal)
   1. 实现抽象方法: 吃(eat)
      1. 输出格式: 3岁的黑色大猩猩吃香蕉
   2. 提供: 空参,带参构造;
6. 定义会表演的大猩猩(ActedGorilla),继承大猩猩类(Gorilla),实现表演者接口
   1. 实现抽象方法表演(play)
      1. 输出格式: 3岁的黑色大猩猩在表演骑自行车
   2. 提供: 空参,带参构造;
7. 定义鹦鹉类(Parrot),继承动物类(Animal)
   1. 实现抽象方法: 吃(eat)
      1. 输出格式: 1岁的绿色鹦鹉在吃小米
   2. 特有方法: 说话(say)
      1. 输出格式: 1岁的绿色鹦鹉在说你好
   3. 提供: 空参,带参构造;
8. 定义会会表演的鹦鹉类(ActedParrot),继承鹦鹉类(Parrot),实现表演者接口
   1. 实现抽象方法表演(play)
      1. 输出格式: 1岁的绿色鹦鹉在表演过跷跷板
   2. 提供: 空参,带参构造;
9. 定义动物园管理员类(ZooManager)
   1. 属性:姓名(name),年龄(age)
   2. 行为:
      1. 喂养动物(feed(Animal a)),调用吃的方法;
         1. 请思考此处为什么使用Animal作为参数类型而不是其子类
         2. 在方法内部,调用a对象的eat方法
      2. 组织演出(letItShow(Actor a))
         1. 请思考此处为什么使用Actor接口类型作为参数类型而不是其实现类
         2. 在方法内部
            1. 调用a的play方法
            2. 如果是鹦鹉,把a向下转型为Parrot类型,调用说话方法

c) 提供空参,有参构造方法与setters和getters方法

1. 定义测试类Test
   1. 提供main方法
   2. 在main方法
      1. 创建会表演的老虎类(ActedTiger)对象t,把颜色赋值为黄色,年龄赋值为2
      2. 创建会表演的大猩猩(ActedGorilla)对象g,把颜色赋值黑色,年龄赋值为3
      3. 创建会表演的鹦鹉类(ActedParrot)对象 p,把颜色赋值为绿色,年龄赋值1
      4. 创建物园管理员类(ZooManager)对象zm,把名称赋值邓超,年龄赋值30
         1. 调用三次zm对象的喂养动物(feed(Animal a))方法,分别传入t,g,p对象
         2. 调用三次zm对象的组织表演(letItShow(Actor a))方法,分别传入t,g,p对象

测试类：

**public** **class** Test {

**public** **static** **void** main(String[] args) {

ActedTiger t = **new** ActedTiger("黄色", 2);

ActedGorilla g=**new** ActedGorilla("黑色", 3);

ActedParrot p=**new** ActedParrot("绿色", 1);

ZooManager zm=**new** ZooManager("邓超", 30);

zm.feed(t);

zm.feed(g);

zm.feed(p);

zm.letItShow(t);

zm.letItShow(g);

zm.letItShow(p);

}

}

Animal类：

**public** **abstract** **class** Animal {

**private** String color;

**private** **int** age ;

**public** Animal() {

**super**();

}

**public** Animal(String color, **int** age) {

**super**();

**this**.color = color;

**this**.age = age;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** **abstract** **void** eat();

}

Actor接口：

**public** **interface** Actor {

**public** **void** play();

}

Parrot类为例子：

@Override

**public** **void** eat() {

System.***out***.println(getAge()+"岁的"+getColor()+"鹦鹉在吃小米");

}

**public** **void** say(){

System.***out***.println(getAge()+"岁的"+getColor()+"鹦鹉在说你好");

}

ActorParrot类：

@Override

**public** **void** play() {

System.***out***.println(getAge()+"岁的"+getColor()+"鹦鹉在表演过跷跷板");

}

ZooManager类：

**public** **void** feed(Animal a){

a.eat();

}

**public** **void** letItShow(Actor a){

a.play();

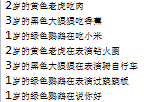
**if**(a **instanceof** ActedParrot){

Parrot parrot = (Parrot)a;

parrot.say();

}

}



# 关卡3

## 训练案例1

### 训练考核知识点

继承

接口

类实现接口

多态

向下转型

### 训练描述

请使用代码描述:

所有飞机(Plane)都具有飞(fly)的功能，但是战斗机(Battleplane)除了具有飞的功能外，还具有发射炮弹(fire)的功能！

要求:在测试类中创建战斗机对象,分别调用fly和fire方法; 然后测试向上转型(转换为父类类型和实现的接口类型)和向下转型.

测试类：

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Battleplane battleplane = **new** Battleplane();

battleplane.fly();

battleplane.fire();

System.***out***.println("----------");

System.***out***.println("向上转型父类：");

Plane plane = **new** Battleplane();

plane.fly();

System.***out***.println("----------");

System.***out***.println("向上转型接口：");

Bullet bullet = **new** Battleplane();

bullet.fire();

System.***out***.println("----------");

System.***out***.println("向下转型：");

Plane plane2 = plane;

Battleplane battleplane2 = (Battleplane)plane2;

battleplane2.fly();

battleplane2.fire();

}

}

Plane类：

**public** **class** Plane {

**public** **void** fly(){

System.***out***.println("飞机在飞");

}

}

战斗机类：

**public** **class** Battleplane **extends** Plane **implements** Bullet{

**public** **void** fly(){

System.***out***.println("战斗机在飞");

}

@Override

**public** **void** fire() {

System.***out***.println("战斗机发射炮弹");

}

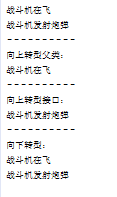
}

Bullet接口：

**public** **interface** Bullet {

**public** **void** fire();

}



## 训练案例2

### 训练考核知识点

抽象类

接口

继承

多态

类名作为参数类型

接口作为参数类型

向下转型

### 训练描述

请使用代码描述:

所有的猫都有颜色和年龄,都会吃(吃鱼)和抓老鼠,但是有部猫会卖萌(actingCute)

所有的狗都有颜色和年龄,都会吃(肯骨头)和看家,但有部分狗会卖萌(actingCute)

去卖萌,会卖萌的猫和狗可以. (提示通过在测试类中的静态方法实现)

如果传入的猫就调用猫的抓老鼠方法

如果传入的狗就调用狗的看家方法

去吃饭,所有动物都可以去(提示通过在测试类中的静态方法实现)

如果传入的猫就调用猫的抓老鼠方法

如果传入的狗就调用狗的看家方法

测试类：

**public** **class** Test {

**public** **static** **void** main(String[] args) {

ActingCuteCat ac = **new** ActingCuteCat("白色",2);

ActingCuteDog ad = **new** ActingCuteDog("黑色", 3);

*actCute*(ac);

*actCute*(ad);

*toEat*(ac);

*toEat*(ad);

}

**public** **static** **void** actCute(ActingCute actingCute) {

actingCute.actingCute();

**if**(actingCute **instanceof** Cat){

((Cat) actingCute).catchMouse();

}**else** {

((Dog)actingCute).lookHome();

}

}

**public** **static** **void** toEat(Animal animal){

animal.eat();

**if**(animal **instanceof** Cat){

((Cat) animal).catchMouse();

}**else** {

((Dog) animal).lookHome();

}

}

}

抽象类Animal：

**public** **abstract** **class** Animal {

**private** String color;

**private** **int** age;

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** Animal(String color, **int** age) {

**super**();

**this**.color = color;

**this**.age = age;

}

**public** Animal() {

**super**();

}

**public** **abstract** **void** eat();

}

以cat为例：

Cat类：

**public** **class** Cat **extends** Animal{

**public** Cat() {

**super**();

}

**public** Cat(String color, **int** age) {

**super**(color, age);

}

@Override

**public** **void** eat() {

System.***out***.println("猫吃鱼");

}

**public** **void** catchMouse(){

System.***out***.println("猫抓老鼠");

}

}

ActingCute接口：

**public** **interface** ActingCute {

**public** **void** actingCute();

}

ActingCuteCat类：

**public** **class** ActingCuteCat **extends** Cat **implements** ActingCute{

**public** ActingCuteCat() {

**super**();

}

**public** ActingCuteCat(String color, **int** age) {

**super**(color, age);

}

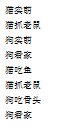
@Override

**public** **void** actingCute() {

System.***out***.println("猫卖萌");

}

}



### 训练案例3

### 训练考核知识点

抽象类

接口

继承

多态

类名作为参数类型

接口作为参数类型

### 训练描述

请用代码描述

在某个公司每一个员工都有工号,姓名和工资,都要工作. 按照做工作的内容不同分为行政部(AdminStaff),开发部(Developer)和财务部(Treasurer)

所有行政部人员都要工作(采购),但是有的行政部员工会表演(跳舞)

所有的开发部人员都要工作(编程),但有开发部员工会表演(唱歌)

所有的财务部人员都要工作(对账),但是有的财务部员工表演(玩魔术)

35岁的薛之谦老板会安排员工工作和组织员工表演.

测试类：

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Boss boss = **new** Boss("薛之谦",30);

AdminStaff a = **new** AdminStaff(11, "a", 1000);

AdminStaffDance as= **new** AdminStaffDance(12, "b", 2000);

Developer d = **new** Developer(13, "c", 5000);

DeveloperSing ds= **new** DeveloperSing(14, "d", 10000);

Treasurer t = **new** Treasurer(15, "e", 3000);

TreasurerMagic tm = **new** TreasurerMagic(16, "f", 3500);

System.***out***.println("---------工作----------");

boss.goToWork(a);

boss.goToWork(as);

boss.goToWork(d);

boss.goToWork(ds);

boss.goToWork(t);

boss.goToWork(tm);

System.***out***.println("---------表演----------");

boss.goToAct(as);

boss.goToAct(ds);

boss.goToAct(tm);

}

}

Employee类写了一个抽象方法

public abstract void work();

以行政部为例：

AdminStaff类：

**public** **class** AdminStaff **extends** Employee{

**public** AdminStaff() {

**super**();

}

**public** AdminStaff(**int** id, String name, **int** salary) {

**super**(id, name, salary);

}

@Override

**public** **void** work() {

System.***out***.println("行政部工号为"+getId()+"工资为"+getSalary()+"的"+getName()+"在采购");

}

}

Actor接口：

**public** **interface** Actor {

**public** **void** play();

}

AdminStaffDance类：

**public** **class** AdminStaffDance **extends** AdminStaff **implements** Actor{

**public** AdminStaffDance() {

**super**();

}

**public** AdminStaffDance(**int** id, String name, **int** salary) {

**super**(id, name, salary);

}

@Override

**public** **void** play() {

System.***out***.println("行政部工号为"+getId()+"工资为"+getSalary()+"的"+getName()+"在跳舞");

}

}

Boss类实现了两个方法：

**public** **void** goToWork(Employee employee){

employee.work();

}

**public** **void** goToAct(Actor actor){

actor.play();

}

运行结果：

