**每日作业卷答案**

**就业班JavaSE第3天**

# 关卡1

## 训练案例1

### 训练描述

请阐述接口中成员的特点

### 操作步骤答案

1. 接口中的方法都是使用public abstract 修饰的,所以接口中的方法都是抽象的.

2. 接口中变量都是使用public static final 修饰的 所以接口中的变量也称之为常量，其值不能改变

3. 接口不可以直接创建对象,因为接口中的方法都是抽象方法,创建对象没有意义.

4. 实现接口的类,如果实现了接口中的所有抽象方法就可以创建对象,否则是抽象类不能创建对象.

## 训练案例2

### 训练描述

请阐述接口的特点.

### 操作步骤答案

1. 接口可以继承接口但类不能继承接口,只能实现接口
2. Java中的类不支持多继承,但是java 中的接口支持多继承.
3. Java支持继承父类的同时,再实现多个接口.
4. Java 允许父类的与实现接口的有相同的功能

## 训练案例3

### 训练描述

请阐述接口与抽象的区别

### 操作步骤答案

1. 相同点
   1. 都位于继承的顶端,用于被其他类实现或继承
   2. 都不能直接实例化对象
   3. 都可以包含抽象方法,其实现类或子类必须覆盖这些抽象方法
2. 区别
   1. 抽象类为部分方法提供实现,避免子类重复实现这些方法,提高代码重用性;接口只能包含抽象方法;
   2. 一个类只能继承一个直接父类(可能是抽象类),却可以实现多个接口;(接口弥补了Java的单继承)
   3. 抽象类为继承体系中的共性内容,接口为继承体系中的扩展功能
3. 如何选用
   1. 优先选用接口,尽量少用抽象类;
   2. 需要定义子类的行为,又要为子类提供共性功能时才选用抽象类;

## 训练案例4

### 训练描述

请阐述一下你对多态理解

### 操作步骤答案

1. 多态前提条件
   1. 必须子父类关系或类实现接口关系
   2. 必须有子类覆盖父类或接口方法
   3. 必须有父类引用指向子类对象
2. 多态有哪些注意事项?
   1. 成员变量编译看父类中是否存在,不存在编译失败
   2. 成员变量运行父类中的变量
   3. 成员方法编译看父类中是否存在,不存在编译失败
   4. 成员方法运行子类重写的方法
3. 多态的好处和弊端
   1. 好处:提高了代码的复用性与扩展性
   2. 弊端:不能调用子类特有方法
4. 多态的向上转型

* 向上转型：当有子类对象赋值给一个父类引用时，便是向上转型，多态本身就是向上转型的过程。

1. 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?
   1. 向下转型：一个已经向上转型的子类对象可以使用强制类型转换的格式，将父类引用转为子类引用，这个过程是向下转型
   2. 由于多态情况下不能使用子类(或实现类)特有方法,当需要使用子类(或实现类)特有方法就需要向下转型.
   3. 如果向下转型的引用变量,指向的不是所要转换的类型,在程序运行时就会报类型转换异常,在向下转型时,必须确保引用指向的是所要转换的类型,此时可以使用instanceof关键进行判断.

## 训练案例5

### 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

### 操作步骤答案

/\*

1.定义玩游戏(PlayGame)接口

a)声明抽象 playGame()

\*/

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

// a)声明抽象 playGame()

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

}

/\*

2.定义手机类(Phone)类,实现玩游戏(PlayGame)接口

a)属性: 品牌(brand),价格(price)

b)功能:

i.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

ii.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

c)提供 空参,有参构造;setters和getters方法

\*/

**public** **class** Phone **implements** PlayGame{

// a)属性: 品牌(brand),价格(price)

**private** String brand;

**private** **int** price;

// b)功能:

// i.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

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

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

}

// ii.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

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

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

}

// c)提供 空参,有参构造;setters和getters方法

**public** Phone() {

**super**();

}

**public** Phone(String brand, **int** price) {

**super**();

**this**.brand = brand;

**this**.price = price;

}

**public** String getBrand() {

**return** brand;

}

**public** **void** setBrand(String brand) {

**this**.brand = brand;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

## }

/\*

3.定义电脑(Computer)类实现玩游戏接口(PlayGame())

a)属性: 品牌(brand)和价格(price)

b)功能:

i.实现playGame()方法

1.输出格式: 使用6000元的雷神电脑玩游戏

ii.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

c)提供空参,有参构造,setters与getters

\*/

**public** **class** Computer **implements** PlayGame {

// a)属性: 品牌(brand)和价格(price)

**private** String brand;

**private** **int** price;

// b)功能:

// i.实现playGame()方法

// 1.输出格式: 使用6000元的雷神电脑玩游戏

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

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

}

// ii.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

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

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

}

// c)提供空参,有参构造,setters与getters

**public** Computer() {

**super**();

}

**public** Computer(String brand, **int** price) {

**super**();

**this**.brand = brand;

**this**.price = price;

}

**public** String getBrand() {

**return** brand;

}

**public** **void** setBrand(String brand) {

**this**.brand = brand;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

}

/\*

4.定义测试类Test

a)定义main方法

b)在main方法中

i.创建手机对象 p,品牌赋值为华为,价格赋值为3000

ii.调用手机对象 p的玩游戏功能

iii.调用手机对象 p的打电话功能

iv.创建电脑对象 c,品牌赋值为雷神,价格赋值为6000

v.调用电脑对象c的玩游戏功能

vi.调用电脑对象c的编程功能

\*/

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

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

// i.创建手机对象 p,品牌赋值为华为,价格赋值为3000

Phone p = **new** Phone("华为", 3000);

// ii.调用手机对象 p的玩游戏功能

p.playGame();

// iii.调用手机对象 p的打电话功能

p.call();

// iv.创建电脑对象 c,品牌赋值为雷神,价格赋值为6000

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

// v.调用电脑对象c的玩游戏功能

c.playGame();

// vi.调用电脑对象c的编程功能

c.coding();

}

}

## 训练案例6

### 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.

### 操作步骤答案

/\*

1.定义奥迪车类(Audi)

a)成员方法: 跑(run())

i.输出格式: 奥迪车在跑

\*/

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

// a)成员方法: 跑(run())

// i.输出格式: 奥迪车在跑

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

System.***out***.println("奥迪车在跑");

}

}

/\*

2.定义智能接口(Smart)

a)抽象方法: 自动泊车(automaticParking)和无人驾驶(automaticDrive)

\*/

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

// 自动泊车(automaticParking)

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

// 无人驾驶(automaticDrive)

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

}

/\*

3.定义智能奥迪车类(SmartAudi) 继承Audi实现Smart接口

a)成员方法

i.实现automaticParking方法

1.输出:智能奥迪车在自动泊车

ii.实现automaticDrive方法

1.输出:智能奥迪车在无人驾驶

\*/

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

// i.实现automaticParking方法

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

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

}

// ii.实现automaticDrive方法

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

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

}

}

/\*

4.定义测试类Test

a)提供main方法

b)在main方法中

i.创建Audi车对象 a,调用跑方法

ii.创建SmartAudi车对象 sa,调用跑,自动泊车,自动驾驶方法

iii.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

iv.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

v.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.

\*/

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

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

// i.创建Audi车对象 a,调用跑方法

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

a.run();

// ii.创建SmartAudi车对象 sa,调用跑,自动泊车,自动驾驶方法

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

sa.run();

sa.automaticParking();

sa.automaticDrive();

// iii.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

Audi aa = sa;

aa.run();

// aa.automaticParking(); 编译报错

// aa.automaticDrive(); 编译报错

// iv.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

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

SmartAudi saa = (SmartAudi) aa;

saa.automaticParking();

saa.automaticDrive();

}

// v.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.

Smart s = sa;

s.automaticDrive();

s.automaticParking();

// s.run(); 编译报错

}

}

# 关卡2

## 训练案例1

### 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.

### 操作步骤答案

/\*

1.定义玩游戏(PlayGame)接口

a)声明抽象方法 playGame()

\*/

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

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

}

/\*

2.定义手机类(Phone)类

a)属性: 品牌(brand),价格(price)

b)功能:

i.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

ii.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

c)提供 空参,有参构造;setters和getters方法

\*/

**public** **class** Phone {

// a)属性: 品牌(brand),价格(price)

**private** String brand;

**private** **int** price;

// b)功能:

// i.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

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

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

}

// ii.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

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

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

}

// c)提供 空参,有参构造;setters和getters方法

**public** Phone() {

**super**();

}

**public** Phone(String brand, **int** price) {

**super**();

**this**.brand = brand;

**this**.price = price;

}

**public** String getBrand() {

**return** brand;

}

**public** **void** setBrand(String brand) {

**this**.brand = brand;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

}

/\*

3.定义智能手机(SmartPhone)类继承手机类(Phone),实现玩游戏接口(PlayGame())

a)实现playGame()方法

i.在使用3000元的 华为 手机玩游戏

b)提供空参,有参构造

\*/

**public** **class** SmartPhone **extends** Phone **implements** PlayGame {

//a)实现playGame()方法

// i.在使用5000元的iPhone手机玩游戏

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

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

}

//b)提供空参,有参构造

**public** SmartPhone() {

**super**();

}

**public** SmartPhone(String brand, **int** price) {

**super**(brand, price);

}

}

/\*

4.定义平板(Pad)类,实现PlayGame接口

a)属性:品牌(brand),价格(price)

b)成员方法

i.实现playGame()方法

1.输出格式: 在使用2000元的iPad平板玩游戏

ii.特有方法: 听音乐(listenMusic)

1.输出格式: 在使用2000元的iPad平板玩听音乐

c)提供空参,有参构造,setters和getters方法

\*/

**public** **class** Pad **implements** PlayGame {

// a)属性:品牌(brand),价格(price)

**private** String brand;

**private** **int** price;

// b)成员方法

// i.实现playGame()方法

// 1.输出格式: 在使用2000元的iPad平板玩游戏

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

System.***out***.println("在使用"+price+"元的"+brand+"平板玩游戏");

}

// ii.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

**public** **void** listenMusic(){

System.***out***.println("在使用"+price+"元的"+brand+"平板听音乐");

}

// c)提供空参,有参构造,setters和getters方法

**public** Pad() {

**super**();

}

**public** Pad(String brand, **int** price) {

**super**();

**this**.brand = brand;

**this**.price = price;

}

**public** String getBrand() {

**return** brand;

}

**public** **void** setBrand(String brand) {

**this**.brand = brand;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

}

/\*

5.定义测试类Test

a)创建手机对象 njy,品牌赋值为诺基亚,价格赋值为100

b)调用手机对象 njy的打印电话和发短信功能,传入 景甜

c)创建智能手机对象 sp,品牌赋值为华为,价格赋值为3000

d)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

e)创建平板对象 p,品牌赋值为iPad,价格赋值为2000

f)调用平板p对象的听歌和玩游戏功能.

\*/

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

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

// a)创建手机对象 njy,品牌赋值为诺基亚,价格赋值为100

Phone njy = **new** Phone();

njy.setBrand("诺基亚");

njy.setPrice(100);

// b)调用手机对象 njy的打印电话和发短信功能,传入 景甜

njy.call("景甜");

njy.sendMessage("景甜");

// c)创建智能手机对象 sp,品牌赋值为华为,价格赋值为3000

SmartPhone sp = **new** SmartPhone("华为", 2000);

// d)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

sp.call("林思意");

sp.sendMessage("林思意");

sp.playGame();

// e)创建平板对象 p,品牌赋值为iPad,价格赋值为2000

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

// f)调用平板p对象的听歌和玩游戏功能.

p.listenMusic();

p.playGame();

}

}

## 训练案例2

### 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)

### 操作步骤答案

/\*

1.定义抽象人类(Person)

a)属性: 姓名(name),年龄(age)

b)抽象方法: eat()

c)提供:无参,有参构造方法和setters和getters方法

\*/

**public** **abstract** **class** Person {

// a)属性: 姓名(name),年龄(age)

**private** String name;

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

// b)抽象方法: eat()

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

// c)提供:无参,有参构造方法和setters和getters方法

**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;

}

}

/\*

2.定义运动接口(Sport)

a)抽象方法: playBasketball

\*/

**public** **interface** Sport {

**public** **abstract** **void** playBasketball();

}

/\*

3.定义老师类(Teacher)继承Person类

a) 成员方法:: 实现抽象方法eat()

i.输出格式: 年龄为30岁周晓斌老师,正在吃工作餐

b) 提供: 无参,有参构造方法

\*/

**public** **class** Teacher **extends** Person {

//a) 成员方法:: 实现抽象方法eat()

// i.输出格式: 年龄为30岁周晓斌老师,正在吃工作餐

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

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

}

// 提供: 无参,有参构造方法

**public** Teacher() {

**super**();

}

**public** Teacher(String name, **int** age) {

**super**(name, age);

}

}

/\*

4.定义运动老师类(SportTeacher)继承老师类(Teacher)实现运动接口

a)成员方法: 实现playBasketball的抽象方法

i.输出格式: 年龄为30岁 周晓斌 老师在打篮球

b)提供: 无参,有参构造方法

\*/

**public** **class** SportTeacher **extends** Teacher **implements** Sport {

//a)成员方法: 实现playBasketball的抽象方法

// i.输出格式: 年龄为30岁 周晓斌 老师在打篮球

**public** **void** playBasketball() {

System.***out***.println("年龄为"+getAge()+"岁 "+getName()+" 老师在打篮球");

}

//b)提供: 无参,有参构造方法

**public** SportTeacher() {

**super**();

}

**public** SportTeacher(String name, **int** age) {

**super**(name, age);

}

}

/\*

5.定义学生类(Student),继承人类(Person)

a)成员方法:现抽象方法eat()

i.输出格式: 年龄18岁的 叶知秋 在吃学生餐

b)提供: 无参,有参构造方法

\*/

**public** **class** Student **extends** Person{

//a)成员方法:现抽象方法eat()

// i.输出格式: 年龄18岁的 叶知秋 在吃学生餐

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

System.***out***.println("年龄"+getAge()+"岁的 "+getName()+" 在吃学生餐");

}

//b)提供: 无参,有参构造方法

**public** Student() {

**super**();

}

**public** Student(String name, **int** age) {

**super**(name, age);

}

}

/\*

5.定义学生类(Student),继承人类(Person)

a)成员方法:现抽象方法eat()

i.输出格式: 年龄18岁的 叶知秋 在吃学生餐

b)提供: 无参,有参构造方法

\*/

**public** **class** Student **extends** Person{

//a)成员方法:现抽象方法eat()

// i.输出格式: 年龄18岁的 叶知秋 在吃学生餐

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

System.***out***.println("年龄"+getAge()+"岁的 "+getName()+" 在吃学生餐");

}

//b)提供: 无参,有参构造方法

**public** Student() {

**super**();

}

**public** Student(String name, **int** age) {

**super**(name, age);

}

}

/\*

7.定义测试类Test

a)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

b)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

c)提供main方法

d)在main方法中

i.创建运动老师对象sp,年龄赋值为30,姓名赋值为周晓斌

i.思考这里为什么要传入Sport接口类型

ii.创建运动学生对象ss,年龄赋值为18,姓名赋值为叶知秋

i.思考这里为什么要传入顶层父类类型

iii.调用去运动goToSport方法传入对象sp

iv.调用去运动goToSport方法,传入对象ss

v.调用去食堂gotToRestaurant方法,传入sp

vi.调用去食堂goToRestaurant方法,传入ss

\*/

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

// a)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

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

s.playBasketball();

}

// b)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

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

p.eat();

}

// main方法

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

// i.创建运动老师对象sp,年龄赋值为30,姓名赋值为周晓斌

SportTeacher sp = **new** SportTeacher();

sp.setAge(30);

sp.setName("周晓斌");

// ii.创建运动学生对象ss,年龄赋值为18,姓名赋值为叶知秋

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

// iii.调用去运动goToSport方法传入对象sp

*goToSport*(sp);

// iv.调用去运动goToSport方法,传入对象ss

*goToSport*(ss);

// v.调用去食堂gotToRestaurant方法,传入sp

*goToRestaurant*(sp);

// vi.调用去食堂goToRestaurant方法,传入ss

*goToRestaurant*(ss);

}

}

## 训练案例3

### 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

### 操作步骤答案

/\*

1.定义动物类(Animal)

a)属性:颜色(color),年龄(age)

b)抽象方法: 吃(eat)

c)提供: 空参,带参构造;setters和getters方法

\*/

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

// a)属性:颜色(color),年龄(age)

**private** String color;

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

// b)抽象方法: 吃(eat)

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

// c)提供: 空参,带参构造;setters和getters方法

**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;

}

}

/\*

2.定义表演者接口(Actor)

a)抽象方法: play()

\*/

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

// 表演

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

}

/\*

3.定义老虎类(Tiger),继承动物类(Animal)

a)实现抽象方法: 吃(eat)

i.输出格式: 2岁的黄色老虎吃肉

b)提供: 空参,带参构造;

\*/

**public** **class** Tiger **extends** Animal {

//a)实现抽象方法: 吃(eat)

// i.输出格式: 2岁的黄色老虎吃肉

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

System.***out***.println(getAge()+"岁的"+getColor()+"老虎吃肉");

}

//b)提供: 空参,带参构造;

**public** Tiger() {

**super**();

}

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

**super**(color, age);

}

}

/\*

4.定义会表演的老虎类(ActedTiger),继承老虎类(Tiger),实现表演者接口

a)实现抽象方法表演(play)

i.输出格式: 2岁的黄色老虎在表演钻火圈

b)提供: 空参,带参构造;

\*/

**public** **class** ActedTiger **extends** Tiger **implements** Actor{

//a)实现抽象方法表演(play)

// i.输出格式: 2岁的黄色老虎在表演钻火圈

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

System.***out***.println(getAge()+"岁的"+getColor()+"老虎在表演钻火圈");

}

//b)提供: 空参,带参构造;

**public** ActedTiger() {

**super**();

}

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

**super**(color, age);

}

}

/\*

5.定义大猩猩类(Gorilla),继承动物类(Animal)

a)实现抽象方法: 吃(eat)

i.输出格式: 3岁的黑色大猩猩吃香蕉

b)提供: 空参,带参构造;

\*/

**public** **class** Gorilla **extends** Animal {

//a)实现抽象方法: 吃(eat)

// i.输出格式: 3岁的黑色大猩猩吃香蕉

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

System.***out***.println(getAge()+"岁的"+getColor()+"大猩猩吃香蕉");

}

//b)提供: 空参,带参构造;

**public** Gorilla() {

**super**();

}

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

**super**(color, age);

}

}

/\*

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

a)实现抽象方法表演(play)

i.输出格式: 3岁的黑色大猩猩在表演骑自行车

b)提供: 空参,带参构造;

\*/

**public** **class** ActedGorilla **extends** Gorilla **implements** Actor {

//a)实现抽象方法表演(play)

// i.输出格式: 3岁的黑色大猩猩在表演骑自行车

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

System.***out***.println(getAge()+"岁的"+getColor()+"大猩猩在表演骑自行车");

}

//b)提供: 空参,带参构造;

**public** ActedGorilla() {

**super**();

}

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

**super**(color, age);

}

}

/\*

7.定义鹦鹉类(Parrot),继承动物类(Animal)

a)实现抽象方法: 吃(eat)

i.输出格式: 1岁的绿色鹦鹉在吃小米

b)特有方法: 说话(say)

i.输出格式: 1岁的绿色鹦鹉在说你好

c)提供: 空参,带参构造;

\*/

**public** **class** Parrot **extends** Animal {

//a)实现抽象方法: 吃(eat)

// i.输出格式: 1岁的绿色鹦鹉在吃小米

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

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

}

//b)特有方法: 说话(say)

// i.输出格式: 1岁的绿色鹦鹉在说你好

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

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

}

//c)提供: 空参,带参构造;

**public** Parrot() {

**super**();

}

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

**super**(color, age);

}

}

/\*

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

a)实现抽象方法表演(play)

i.输出格式: 1岁的绿色鹦鹉在表演过跷跷板

b)提供: 空参,带参构造;

\*/

**public** **class** ActedParrot **extends** Parrot **implements** Actor {

//a)实现抽象方法表演(play)

// i.输出格式: 1岁的绿色鹦鹉在表演过跷跷板

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

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

}

//b)提供: 空参,带参构造;

**public** ActedParrot() {

**super**();

}

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

**super**(color, age);

}

}

/\*

9.定义动物园管理员类(ZooManager)

a)属性:姓名(name),年龄(age)

b)行为:

i.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

ii.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

a)调用a的play方法

b)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

\*/

**public** **class** ZooManager {

// a)属性:姓名(name),年龄(age)

**private** String name;

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

// b)行为:

// i.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

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

a.eat();

}

// ii.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

// a)调用a的play方法

// b)如果是鹦鹉,把a向下转型为Parrot类型,调用说话方法

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

// a)调用a的play方法

a.play();

// b)如果是鹦鹉,把a向下转型为Parrot类型,调用说话方法

**if**(a **instanceof** Parrot){

Parrot p = (Parrot) a;

p.say();

}

}

// c) 提供空参,有参构造方法与setters和getters方法

**public** ZooManager() {

**super**();

}

**public** ZooManager(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;

}

}

/\*

10.定义测试类Test

a)提供main方法

b)在main方法

i.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

ii.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

iii.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

iv.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zm对象的组织表演(letItShow(Actor a))方法,分别传入t,g,p对象

\*/

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

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

// i.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

ActedTiger t = **new** ActedTiger();

t.setAge(2);

t.setColor("黄色");

// ii.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

ActedGorilla g = **new** ActedGorilla();

g.setAge(3);

g.setColor("黑色");

// iii.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

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

// iv.创建物园管理员类(ZooManager)对象zm,把名称赋值邓超,年龄赋值30

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

// 1.调用zm对象的喂养动物(feed(Animal a))方法,分别传入t,g,p对象

zm.feed(t);

zm.feed(g);

zm.feed(p);

// 2.调用zm对象的组织表演(letItShow(Actor a))方法,分别传入t,g,p对象

zm.letItShow(t);

zm.letItShow(g);

zm.letItShow(p);

}

}

# 关卡3

## 训练案例1

### 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.

### 操作步骤

1. 定义飞机类Plane
   1. 成员方法:fly()
      1. 输出:飞机在飞
2. 定义接口类Firing接口
   1. 抽象方法:fire
3. 定义战斗机类(Battleplane) 继承 Plane 实现 Firing接口
   1. 实现抽象方法 fire
      1. 输出:战斗机在发射炮弹
4. 定义测试类(Test)
   1. 创建战斗机(Battleplane)的对象 bp,调用飞和发射炮弹功能
   2. 把bp赋值给 Plane p,
      1. 尝试调用fly和fire方法
      2. 先判断p是否是Battleplane的实例对象,如果是把p向下转型为Battleplane,然后调用fire方法
   3. 把bp赋值给 Fireing 类型 f 变量
      1. 尝试使用使用f调用fly和fire方法
      2. 先判断f是否是Battleplane的实例对象,如果是把f向下转为为Battleplane,然后调用fly方法

### 操作步骤答案

/\*

1.定义飞机类Plane

a)成员方法:fly()

i.输出:飞机在飞

\*/

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

// 飞

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

System.***out***.println("飞机在飞");

}

}

/\*

2.定义接口类Firing接口

a)抽象方法:fire

\*/

**public** **interface** Firing {

// a)抽象方法:发射 fire

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

}

/\*

3.定义战斗机类(Battleplane) 继承 Plane 实现 Firing接口

a)实现抽象方法 fire

i.输出:战斗机在发射炮弹

\*/

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

// 发射

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

System.***out***.println("战斗机在发射炮弹");

}

}

/\*

4.定义测试类(Test)

a)创建战斗机(Battleplane)的对象 bp,调用飞和发射炮弹功能

b)把bp赋值给 Plane p,

i.尝试调用fly和fire方法

ii.先判断p是否是Battleplane的实例对象,如果是把p向下转型为Battleplane,然后调用fire方法

c)把bp赋值给 Fireing 类型 f 变量

i.尝试使用使用f调用fly和fire方法

ii.先判断f是否是Battleplane的实例对象,如果是把f向下转为为Battleplane,然后调用fly方法

\*/

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

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

// a)创建战斗机(Battleplane)的对象 bp,调用飞和发射炮弹功能

Battleplane bp = **new** Battleplane();

bp.fly();

bp.fire();

// b)把bp赋值给 Plane p,

// i.尝试调用fly和fire方法

Plane p = bp;

p.fly();

// p.fire(); 编译报错,方法调用编译看左边的类型

// ii.先判断p是否是Battleplane的实例对象,如果是把p向下转型为Battleplane,然后调用fire方法

**if**(p **instanceof** Battleplane){

Battleplane bpp = (Battleplane) p;

bpp.fire();

}

// c)把bp赋值给 Fireing 类型 f 变量

Firing f = bp;

// i.尝试使用使用f调用fly和fire方法

f.fire();

// ii.先判断f是否是Battleplane的实例对象,如果是把f向下转为为Battleplane,然后调用fly方法

**if**(f **instanceof** Battleplane){

Battleplane pf = (Battleplane) f;

pf.fly();

}

}

}

## 训练案例2

### 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)

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)

### 操作步骤

总体思路:

1. 把猫和狗都有属性和行为抽取到动物类中
2. 把卖萌的额外功能提取高接口中

实现步骤:

1. 定义抽象动物类Animal
   1. 属性: 颜色(Color) ,年龄(age)
   2. 抽象方法: 吃(eat)
   3. 提供:空参,有参构造方法,setters和getters 方法
2. 定义接口卖萌接口(ActingCute)
   1. 抽象方法: 卖萌(actingCute)
3. 定义狗类(Dog)继承动物类(Animal)
   1. 实现父类抽象吃(eat)方法
      1. 输出格式: 2岁黑色的狗在肯骨头
   2. 特有方法: 看家(lookHome)
      1. 输出格式: 2岁黑色的狗在看家
   3. 提供:空参,有参构造方法
4. 定义会卖萌的狗类(ActingCuteDog) 继承狗类(Dog) 实现卖萌接口(ActingCute)
   1. 实现接口中的卖萌方法:actingCute())
      1. 输出格式: 2岁黑色的狗在装无辜
   2. 提供:空参,有参构造方法
5. 定义猫类(Cat) 继承动物类(Animal)
   1. 实现父类的吃(eat)方法
      1. 输出格式: 1岁白色的猫在吃鱼
   2. 特有方法:抓老鼠
      1. 输出格式: 1岁白色的猫在抓老鼠
   3. 提供:空参,有参构造方法
6. 定义会卖萌的猫类(ActingCuteCat) 继承猫类(Cat) 实现卖萌接口(ActionCute)
   1. 实现接口中的卖萌方法:actingCute())
      1. 输出格式: 1岁白色的猫在扮可怜
   2. 提供:空参,有参构造方法
7. 定义测试类(Test)
   1. 定义静态方法 去卖萌(goToActingCute(ActingCute ac))
      1. 在方法中:调用卖萌方法
      2. 如果传入的猫就调用猫的抓老鼠方法
      3. 如果传入的狗就调用狗的看家方法
   2. 定义静态方法 去吃饭(goToEat(Animal a))
      1. 在方法中调用吃方法
      2. 如果传入的猫就调用猫的抓老鼠方法
      3. 如果传入的狗就调用狗的看家方法
   3. 定义main方法
   4. 在main方法中
      1. 创建狗对象d,颜色赋值花色,年龄赋值3
      2. 创建会卖萌的狗对象 ad,颜色赋值为黑色,年龄赋值2
      3. 创建猫对象c,颜色赋值为蓝色,年龄赋值为4
      4. 创建会卖萌的猫对象,ac,颜色赋值白色,年龄赋值为1
      5. 调用4次去吃饭方法,分别传入对象d,ad,c,ac
      6. 调用2次去卖萌方法,分别传入ad,ac
      7. 调用2次去卖萌方法,分别传入d,c; 这里调用编译会通过吗? 思考为什么?

### 操作步骤答案

/\*

1.定义抽象动物类Animal

a)属性: 颜色(Color) ,年龄(age)

b)抽象方法: 吃(eat)

c)提供:空参,有参构造方法,setters和getters 方法

\*/

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

// a)属性: 颜色(Color) ,年龄(age)

**private** String color;

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

// b)抽象方法: 吃(eat)

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

// c)提供:空参,有参构造方法,setters和getters 方法

**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;

}

}

/\*

2.定义接口卖萌接口(ActingCute)

a)抽象方法: 卖萌(actingCute)

\*/

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

// 卖萌

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

}

/\*

3.定义狗类(Dog)继承动物类(Animal)

a)实现父类抽象吃(eat)方法

i.输出格式: 2岁黑色的狗在肯骨头

b)特有方法: 看家(lookHome)

i.输出格式: 2岁黑色的狗在看家

c)提供:空参,有参构造方法

\*/

**public** **class** Dog **extends** Animal {

// a)实现父类抽象吃(eat)方法

// i.输出格式: 2岁黑色的狗在肯骨头

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

System.***out***.println(getAge()+"岁"+getColor()+"的狗在肯骨头");

}

// b)特有方法: 看家(lookHome)

// i.输出格式: 2岁黑色的狗在看家

**public** **void** lookHome(){

System.***out***.println(getAge()+"岁"+getColor()+"的在看家");

}

// c)提供:空参,有参构造方法

**public** Dog() {

**super**();

}

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

**super**(color, age);

}

}

/\*

4.定义会卖萌的狗类(ActingCuteDog) 继承狗类(Dog) 实现卖萌接口(ActingCute)

a)实现接口中的卖萌方法:actingCute())

i.输出格式: 2岁黑色的狗在装无辜

b)提供:空参,有参构造方法

\*/

**public** **class** ActingCuteDog **extends** Dog **implements** ActingCute{

//a)实现接口中的卖萌方法:actingCute())

// i.输出格式: 2岁黑色的狗在装无辜

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

System.***out***.println(getAge()+"岁"+getColor()+"的狗在装无辜");

}

//b)提供:空参,有参构造方法

**public** ActingCuteDog() {

**super**();

}

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

**super**(color, age);

}

}

/\*

5.定义猫类(Cat) 继承动物类(Animal)

a)实现父类的吃(eat)方法

i.输出格式: 1岁白色的猫在吃鱼

b)特有方法:抓老鼠

i.输出格式: 1岁白色的猫在抓老鼠

c)提供:空参,有参构造方法

\*/

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

//a)实现父类的吃(eat)方法

// i.输出格式: 1岁白色的猫在吃鱼

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

System.***out***.println(getAge()+"岁"+getColor()+"的猫在吃鱼");

}

//b)特有方法:抓老鼠

// i.输出格式: 1岁白色的猫在抓老鼠

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

System.***out***.println(getAge()+"岁"+getColor()+"的猫在抓老鼠");

}

//c)提供:空参,有参构造方法

**public** Cat() {

**super**();

}

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

**super**(color, age);

}

}

/\*

6.定义会卖萌的猫类(ActingCuteCat) 继承猫类(Cat) 实现卖萌接口(ActionCute)

a)实现接口中的卖萌方法:actingCute())

i.输出格式: 1岁白色的猫在扮可怜

b)提供:空参,有参构造方法

\*/

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

//a)实现接口中的卖萌方法:actingCute())

// i.输出格式: 1岁白色的猫在扮可怜

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

System.***out***.println(getAge()+"岁"+getColor()+"的猫在扮可怜");

}

//b)提供:空参,有参构造方法

**public** ActingCuteCat() {

**super**();

}

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

**super**(color, age);

}

}

/\*

7.定义测试类(Test)

a)定义静态方法 去卖萌(goToActingCute(ActingCute ac))

i.在方法中:调用卖萌方法

ii.如果传入的猫就调用猫的抓老鼠方法

iii.如果传入的狗就调用狗的看家方法

b)定义静态方法 去吃饭(goToEat(Animal a))

i.在方法中:调用吃方法

ii.如果传入的猫就调用猫的抓老鼠方法

iii.如果传入的狗就调用狗的看家方法

c)定义main方法

d)在main方法中

i.创建狗对象d,颜色赋值花色,年龄赋值3

ii.创建会卖萌的狗对象 ad,颜色赋值为黑色,年龄赋值2

iii.创建猫对象c,颜色赋值为蓝色,年龄赋值为4

iv.创建会卖萌的猫对象,ac,颜色赋值白色,年龄赋值为1

v.调用4次去吃饭方法,分别传入对象d,ad,c,ac

vi.调用2次去卖萌方法,分别传入ad,ac

vii.调用2次去卖萌方法,分别传入d,c; 这里调用编译会通过吗? 思考为什么?

\*/

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

// a)定义静态方法 去卖萌(goToActingCute(ActingCute ac))

**public** **static** **void** goToActingCute(ActingCute ac){

// i.在方法中:调用吃方法

ac.actingCute();

// ii.如果传入的猫就调用猫的抓老鼠方法

**if**(ac **instanceof** Cat){

Cat c = (Cat) ac;

c.catchMouse();

}

// iii.如果传入的狗就调用狗的看家方法

**if**(ac **instanceof** Dog){

Dog d = (Dog) ac;

d.lookHome();

}

}

// b)定义静态方法 去吃饭(goToEat(Animal a))

**public** **static** **void** goToEat(Animal a){

// i.在方法中:调用吃方法

a.eat();

// ii.如果传入的猫就调用猫的抓老鼠方法

**if**(a **instanceof** Cat){

Cat c = (Cat) a;

c.catchMouse();

}

// iii.如果传入的狗就调用狗的看家方法

**if**(a **instanceof** Dog){

Dog d = (Dog) a;

d.lookHome();

}

}// c)定义main方法

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

// i.创建狗对象d,颜色赋值花色,年龄赋值3

Dog d = **new** Dog("花色", 3);

// ii.创建会卖萌的狗对象 ad,颜色赋值为黑色,年龄赋值2

ActingCuteDog ad = **new** ActingCuteDog("黑色", 2);

// iii.创建猫对象c,颜色赋值为蓝色,年龄赋值为4

Cat c = **new** Cat();

c.setAge(4);

c.setColor("蓝色");

// iv.创建会卖萌的猫对象,ac,颜色赋值白色,年龄赋值为1

ActingCuteCat ac = **new** ActingCuteCat();

ac.setAge(1);

ac.setColor("白色");

// v.调用4次去吃饭方法,分别传入对象d,ad,c,ac

*goToEat*(d);

*goToEat*(ad);

*goToEat*(c);

*goToEat*(ac);

// vi.调用2次去卖萌方法,分别传入ad,ac

*goToActingCute*(ad);

*goToActingCute*(ac);

// vii.调用2次去卖萌方法,分别传入d,c; 这里调用编译会通过吗? 思考为什么?

// goToActingCute(c); //编译失败 只有实现卖萌接口的类的对象,才可以作为参数传入.

// goToActingCute(d); //编译失败

}

}

## **训练案例3**

### 训练描述

请用代码描述

在某个公司每一个员工都有工号(id),姓名(name)和工资(salary),都要工作(work). 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(play)(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(play)(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(play)(玩魔术)

35岁的薛之谦老板会让安排员工工作和组织员工表演.

### 操作步骤描述

1. 定义抽象员工类(Employee)
   1. 属性:工号(id),姓名(name)和工资(salary)
   2. 抽象方法 (work)
   3. 提供空参,带参构造方法和setters,getters方法
2. 定义表演接口(Actor)
   1. 抽象方法:play()
3. 定义行政部员工类(AdminStaff) 继承 Employee
   1. 实现抽象方法work()
      1. 输出格式: 工号为001的行政部员工景甜在采购物品
   2. 提供空参,带参构造方法
4. 定义会表演的行政部员工类(ActedAdminStaff) 继承 AdminStaff 实现 Actor接口
   1. 实现抽象方法 play
      1. 输出格式: 工号为001的行政部员工景甜在跳舞
   2. 提供空参,带参构造方法
5. 定义开发部员工类(Developer) 继承 Employee
   1. 实现抽象方法work()
      1. 输出格式: 工号为010的开发部员工林俊杰在开发JavaEE应用
   2. 提供空参,带参构造方法
6. 定义会表演的开发部员工类(ActedDeveloper) 继承 Developer实现 Actor接口
   1. 实现抽象方法 play
      1. 输出格式: 工号为010的开发部员工林俊杰在唱歌
   2. 提供空参,带参构造方法
7. 定义财务部员工类(Treasurer) 继承 Employee
   1. 实现抽象方法work()
      1. 输出格式: 工号为100 的财务部员工林思意在对账
   2. 提供空参,带参构造方法
8. 定义会表演的财务部员工类(ActedTreasurer) 继承 Treasurer实现 Actor接口
   1. 实现抽象方法 play
      1. 输出格式: 工号为100的财务部员工林思意在表演魔术
   2. 提供空参,带参构造方法
9. 老板类Boss类
   1. 属性: 年龄,姓名
   2. 行为:
      1. 安排工作(letWork(Employee e)
         1. 调用e对象的工作方法
      2. 组织表演(letPlay(Actor a)
         1. 调用a对象的,表演方法.
   3. 提供空参,带参构造方法;setters和getters方法
10. 测试类Test
    1. 提供main方法
    2. 在main方法中
       1. 创建会表演的行政部员工类(ActedAdminStaff)对象 aas,工号赋值为001,姓名赋值为景甜,工资赋值5000
       2. 创建会表演的开发部员工类(ActedDeveloper) 对象ad,工号赋值为010,姓名赋值为林俊杰,工资赋值为10000
       3. 创建会表演的财务部员工类(ActedTreasurer) 对象 at,工号赋值为100,姓名赋值为林思意,工资赋值为8000
       4. 创建老板(Boss)对象,姓名赋值为薛之谦,年龄赋值为30
          1. 调用三次letWork(Employee e)方法,分别传入ass,ad,at
          2. 调用三次letPlay(Actor a)方法,分别传入ass,ad,at

### 操作步骤答案

/\*

1.定义抽象员工类(Employee)

a)属性:工号(id),姓名(name)和工资(salary)

b)抽象方法 (work)

c)提供空参,带参构造方法和setters,getters方法

\*/

**public** **abstract** **class** Employee {

// a)属性:工号(id),姓名(name)和工资(salary)

**private** String id;

**private** String name;

**private** **int** salary;

// b)抽象方法 (work)

**public** **abstract** **void** work();

// c)提供空参,带参构造方法和setters,getters方法

**public** Employee() {

**super**();

}

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

**super**();

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**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** **int** getSalary() {

**return** salary;

}

**public** **void** setSalary(**int** salary) {

**this**.salary = salary;

}

}

/\*

2.定义表演接口(Actor)

a)抽象方法:play()

\*/

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

// 表演

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

}

/\*

3.定义行政部员工类(AdminStaff) 继承 Employee

a)实现抽象方法work()

i.输出格式: 工号为001的行政部员工景甜在采购物品

b)提供空参,带参构造方法

\*/

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

//a)实现抽象方法work()

// i.输出格式: 工号为001的行政部员工景甜在采购物品

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

System.***out***.println("工号为"+getId()+"的行政部员工"+getName()+"在采购物品");

}

//b)提供空参,带参构造方法

**public** AdminStaff() {

**super**();

}

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

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

}

}

/\*

4.定义会表演的行政部员工类(ActedAdminStaff) 继承 AdminStaff 实现 Actor接口

a)实现抽象方法 play

i.输出格式: 工号为001的行政部员工景甜在跳舞

b)提供空参,带参构造方法

\*/

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

//a)实现抽象方法 play

// i.输出格式: 工号为001的行政部员工景甜在跳舞

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

System.***out***.println("工号为"+getId()+"的行政部员工"+getName()+"在跳舞");

}

//b)提供空参,带参构造方法

**public** ActedAdminStaff() {

**super**();

}

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

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

}

}

/\*

5.定义开发部员工类(Developer) 继承 Employee

a)实现抽象方法work()

i.输出格式: 工号为010的开发部员工林俊杰在开发JavaEE应用

b)提供空参,带参构造方法

\*/

**public** **class** Developer **extends** Employee {

//a)实现抽象方法work()

// i.输出格式: 工号为010的开发部员工林俊杰在开发JavaEE应用

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

System.***out***.println("工号为"+getId()+"的开发部员工"+getName()+"在开发JavaEE应用");

}

//b)提供空参,带参构造方法

**public** Developer() {

**super**();

}

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

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

}

}

/\*

6.定义会表演的开发部员工类(ActedDeveloper) 继承 Developer实现 Actor接口

a)实现抽象方法 play

i.输出格式: 工号为010的开发部员工林俊杰在唱歌

b)提供空参,带参构造方法

\*/

**public** **class** ActedDeveloper **extends** Developer **implements** Actor {

//a)实现抽象方法 play

// i.输出格式: 工号为010的行开发部员工林俊杰在唱歌

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

System.***out***.println("工号为"+getId()+"的开发部员工"+getName()+"在唱歌");

}

//b)提供空参,带参构造方法

**public** ActedDeveloper() {

**super**();

}

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

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

}

}

/\*

7.定义财务部员工类(Treasurer) 继承 Employee

a)实现抽象方法work()

i.输出格式: 工号为100 的财务部员工林思意在对账

b)提供空参,带参构造方法

\*/

**public** **class** Treasurer **extends** Employee {

//a)实现抽象方法work()

// i.输出格式: 工号为100 的财务部员工林思意在对账

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

System.***out***.println("工号为"+getId()+" 的财务部员工"+getName()+"在对账");

}

// b)提供空参,带参构造方法

**public** Treasurer() {

**super**();

}

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

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

}

}

/\*

8.定义会表演的财务部员工类(ActedTreasurer) 继承 Treasurer实现 Actor接口

a)实现抽象方法 play

i.输出格式: 工号为100的行财务部员工林思意在表演魔术

b)提供空参,带参构造方法

\*/

**public** **class** ActedTreasurer **extends** Treasurer **implements** Actor{

//a)实现抽象方法 play

// i.输出格式: 工号为100的行开发部员工林思意在表演魔术

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

System.***out***.println("工号为"+getId()+"的行财务部员工"+getName()+"在表演魔术");

}

//b)提供空参,带参构造方法

**public** ActedTreasurer() {

**super**();

}

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

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

}

}

/\*

9.老板类Boss类

a)属性: 年龄,姓名

b)行为:

i.安排工作(letWork(Employee e)

1.调用e对象的工作方法

ii.组织表演(letPlay(Actor a)

1.调用a对象的,表演方法.

c) 提供空参,带参构造方法;setters和getters方法

\*/

**public** **class** Boss {

// a)属性: 年龄,姓名

**private** String name;

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

// b)行为:

// i.安排工作(letWork(Employee e)

// 1.调用e对象的工作方法

**public** **void** letWork(Employee e){

e.work();

}

// ii.组织表演(letPlay(Actor a)

// 1.调用a对象的,表演方法.

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

a.play();

}

// c) 提供空参,带参构造方法;setters和getters方法

**public** Boss() {

**super**();

}

**public** Boss(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;

}

}

/\*

10.测试类Test

a)提供main方法

b)在main方法中

i.创建会表演的行政部员工类(ActedAdminStaff)对象 aas,工号赋值为001,姓名赋值为景甜,工资赋值5000

ii.创建会表演的开发部员工类(ActedDeveloper) 对象ad,工号赋值为010,姓名赋值为林俊杰,工资赋值为10000

iii.创建会表演的财务部员工类(ActedTreasurer) 对象 at,工号赋值为100,姓名赋值为林思意,工资赋值为8000

iv.创建老板(Boss)对象,姓名赋值为薛之谦,年龄赋值为30

1.调用三次letWork(Employee e)方法,分别传入ass,ad,at

2.调用三次letPlayo(Actor a)方法,分别传入ass,ad,at

\*/

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

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

// i.创建会表演的行政部员工类(ActedAdminStaff)对象 aas,工号赋值为001,姓名赋值为景甜,工资赋值5000

ActedAdminStaff aas = **new** ActedAdminStaff("001", "景甜", 5000);

// ii.创建会表演的开发部员工类(ActedDeveloper) 对象ad,工号赋值为010,姓名赋值为林俊杰,工资赋值为10000

ActedDeveloper ad = **new** ActedDeveloper();

ad.setId("010");

ad.setName("林俊杰");

ad.setSalary(10000);

// iii.创建会表演的财务部员工类(ActedTreasurer) 对象 at,工号赋值为100,姓名赋值为林思意,工资赋值为8000

ActedTreasurer at = **new** ActedTreasurer("100", "林思意", 8000);

// iv.创建老板(Boss)对象,姓名赋值为薛之谦,年龄赋值为30

Boss b = **new** Boss("薛之谦", 30);

// 1.调用三次letWork(Employee e)方法,分别传入aas,ad,at

b.letWork(aas);

b.letWork(ad);

b.letWork(at);

// 2.调用三次letPlayo(Actor a)方法,分别传入aas,ad,at

b.letPlay(aas);

b.letPlay(ad);

b.letPlay(at);

}

}