**每日作业卷答案**

**就业班JavaSE第4天**

传智播客.黑马程序员

# 关卡1

## 训练案例1

### 训练描述

请阐述对static关键字理解

### 操作步骤答案

1. static一般用于修饰类的成员(成员变量和成员方法)
2. static的特点:
   1. 被static修饰的成员属于类，不属于单个这个类的某个对象
   2. static修饰的成员被多个对象共享
   3. static修饰的成员属于类，但是会影响每一个对象。
   4. 被static修饰的成员又叫类成员，不叫对象的成员。
3. static修饰的成员使用
   1. 被static修饰的成员可以并且建议通过类名直接访问
      1. 访问静态方法:
         1. 类名.静态方法名(实参列表)
      2. 访问静态属性:
         1. 类名.静态成员变量名
   2. 也可以通过对象访问,但是不建议
      1. 访问静态方法
         1. 对象.静态方法名(实参列表)
      2. 访问静态成员变量
         1. 对象.静态成员变量名

## 训练案例2

### 训练描述

请阐述你对静态代码块的理解

### 操作步骤答案

1. 静态代码块定义在类中方法外
2. 静态代码块的格式是 static{ 语句体 }
3. 它优先于主方法执行、优先于构造代码块执行，当以任意形式第一次使用到该类时执行
4. 该类不管创建多少对象，静态代码块只执行一次。
5. 可用于给静态变量赋值，用来给类进行初始化。

## 训练案例3

### 训练描述

请阐述你对final关键字的理解

### 操作步骤答案

1. final关键字是一个修饰符,可以修饰类,方法,局部变量,成员变量
2. final特点
   1. final修饰类不可以被继承，但是可以继承其他类。
   2. final修饰的方法不可以被覆盖,但父类中没有被final修饰方法，子类覆盖后可以加final
   3. final修饰的变量称为常量，这些变量只能赋值一次
   4. 引用类型的变量值为对象地址值，地址值不能更改，但是地址内的对象属性值可以修改
   5. 修饰成员变量，需要在创建对象前赋值，否则报错。(当没有显式赋值时，多个构造方法的均需要为其赋值。)

## 训练案例4

### 训练描述

请阐述你对包的理解

### 操作步骤答案

1. 包的作用:类似文件夹,用于管理项目中的类
2. 命名规则: 全部小写,多级包使用.隔开,一般是公司域名倒过来写;
3. 声明格式:package 包名.包名.包名…;
4. 声明位置:必须写在程序有效代码的第一行（注释不算）
5. 访问包中的类
   1. 用含有包名的类全名（包名.类名),前提是要有访问权限
   2. 类的简化访问:
      1. 如果访问的是同包中或java.lang包中的类,可以直接通过类名访问
   3. 当我们要使用的类，与当前程序不在同一个包中（即不同文件夹中），要访问的类必须用public修饰才可访问
6. 导包
   1. 作用: 简化使用类; 导入包后可以直接通过类名访问不同包中的类
   2. 格式: import 包名.类名;
   3. 位置: 包声明语句下面,所有类定义语句上面.

## 训练案例5

### 训练描述

请阐述你对四种访问权限修饰符的理解

### 操作步骤答案

1. 四种访问权限分别是:public,protected,默认权限(空),私有权限
2. 可以修饰类,方法,成员变量
3. 特点:
   1. 被private修饰的只能在本类中使用
   2. 如果没有写权限修饰符就是默认权限,可以在本包的类中使用
   3. 被protected修饰的,可以在本包的类或其子类中使用
   4. 被public修饰的,可以在所有类中使用
4. 使用
   1. 要想仅能在本类中访问使用private修饰
   2. 要想本包中的类都可以访问不加修饰符即可
   3. 要想本包中的类与其他包中的子类可以访问使用protected修饰
   4. 要想所有包中的所有类都可以访问使用public修饰
      1. 注意：如果类用public修饰，则类名必须与文件名相同。一个文件中只能有一个public修饰的类

## 训练案例6

### 训练描述

请阐述内部类的概念

### 操作步骤答案

1. 将类写在其他类的内部，可以写在其他类的成员位置和局部位置，这时写在其他类内部的类就称为内部类。其他类也称为外部类。
2. 在描述事物时，若一个事物内部还包含其他可能包含的事物，比如在描述汽车时，汽车中还包含这发动机，这时发动机就可以使用内部类来描述
3. 内部类分为成员内部类与局部内部类
   1. 成员内部类，定义在外部类中的成员位置。与类中的成员变量相似，可通过外部类对象进行访问
   2. 局部内部类: 定义在方法中. 与局部变量类似,只能在本方法中使用.

## 训练案例7

### 训练描述

请阐述你对成员内部类的理解

### 操作步骤答案

1. 定义位置: 成员内部类，定义在外部类中的成员位置。与类中的成员变量相似，可通过外部类对象进行访问
2. 定义格式:

class 外部类 {

修饰符 class 内部类 {

//其他代码

}

}

1. 创建对象(访问方式):

外部类名.内部类名 变量名 = new 外部类名().new 内部类名();

## 训练案例8

### 训练描述

请阐述你对匿名内部类的理解

### 操作步骤答案

1. 匿名内部类的两个含义
   1. 临时定义某一指定类型的子类(实现类)
   2. 定义后即刻创建刚刚定义的这个子类(实现类)的对象
2. 作用: 匿名内部类是创建某个类型子类(实现类)对象的快捷方式

## 训练案例9

### 训练描述

请阐述你引用数据类型作为参数和返回值理解

### 操作步骤答案

1. 普通类:
   1. 作为参数类型,当调用方法的时候,需要传入的是该类的对象
   2. 作为返回值类型,返回的值是这个类的对象
2. 抽象类
   1. 作为参数类型,当调用方法的时候,需要传入的是该类的实现所有抽象方法的子类的对象
   2. 作为返回值类型,返回的值是该类的实现所有抽象方法的子类的对象
3. 接口
   1. 作为参数类型,当调用方法的时候,需要传入的是该接口的实现类的对象
   2. 作为返回值类型,返回的值是该接口的实现类的对象.

## 训练案例10

### 训练描述

请说出下面代码执行结果并绘制内存图

**public** **class** Student {

// 学校

**public** **static** String *school* = "清华大学";

// 姓名

**private** String name;

// 年龄

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

// 构造方法

**public** Student() {

**super**();

}

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

**super**();

**this**.name = name;

**this**.age = age;

}

}

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

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

Student s1 = **new** Student("景甜",22);

Student s2 = **new** Student("林思意",18);

System.***out***.println(s1.*school*);

s1.*school* = " 传智播客";

System.***out***.println(s2.*school*);

}

}

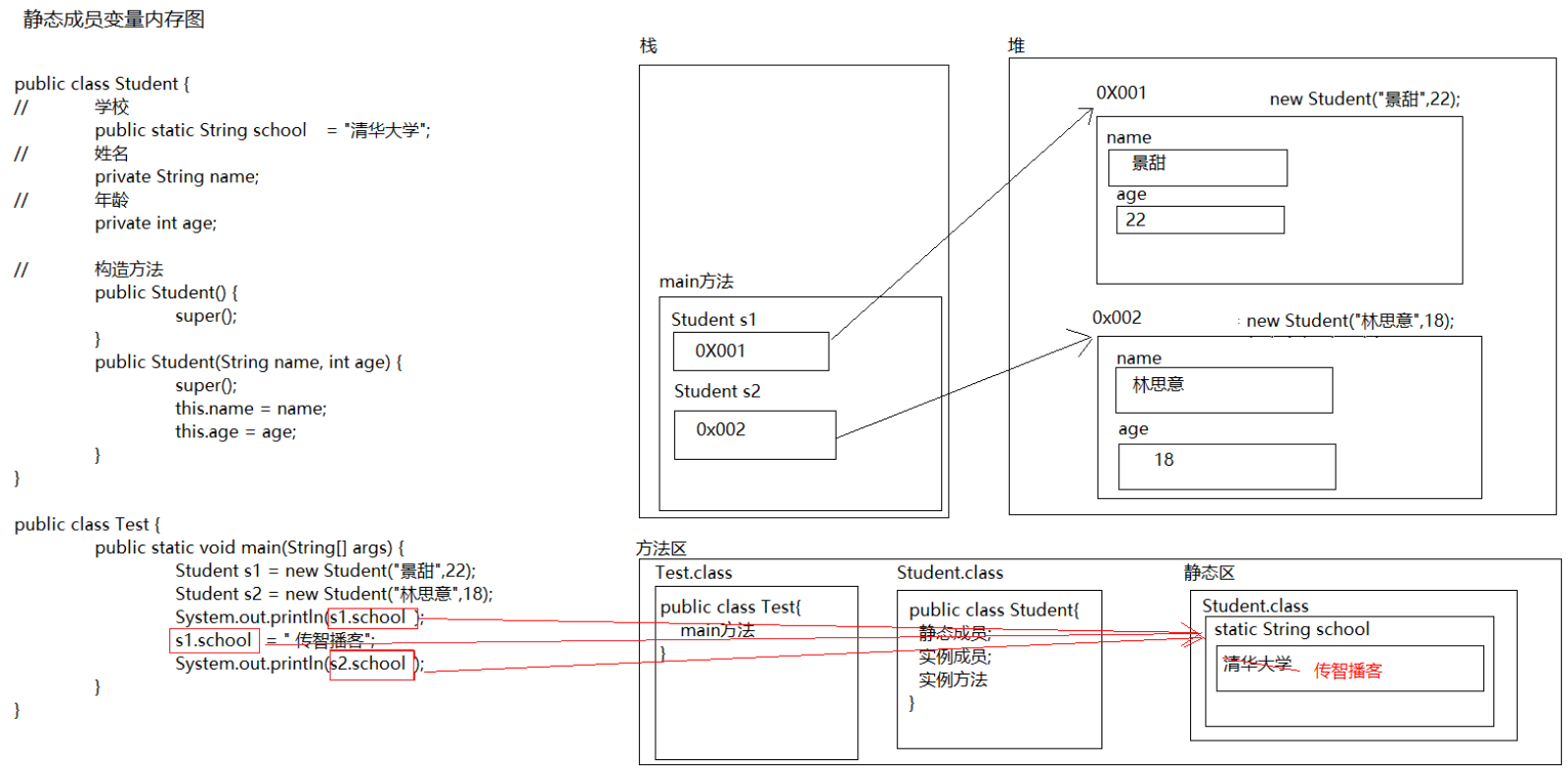
### 操作步骤答案

1. 输出结果:

清华大学

传智播客

1. 内存图



## 训练案例11

### 训练描述

看下面的代码说出运行结果

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

**static** {

System.***out***.println("Test 静态代码块");

}

**public** Test(){

System.***out***.println("Test 构造方法");

}

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

System.***out***.println("Test main方法");

Test t = **new** Test();

}

}

### 操作步骤答案

1. 执行结果:

Test 静态代码块

Test main方法

Test 构造方法

1. 执行流程
   1. 当前程序启动的时候,会首先加载main方法所在的类
   2. 在加载类的时候,会执行静态代码块中的代码,输出Test 静态代码块
   3. 接下来执行main方法,输出: Test main方法
   4. 最后使用空参构造创建Test对象,执行空参的构造方法,输出Test 构造方法

## 训练案例12

### 训练描述

请用代码描述:

汽车有两种状态,一种是停止,一种运行; 汽车的内部都一个发动机;发动机有一个工作的功能; 如果汽车的运行状态发动机就飞速旋转,如果汽车是停止状态,发动机停止工作

要求: 使用成员内部类.

### 操作步骤答案

/\*

1.定义一个汽车类(Car)

a)成员变量: 状态(status),true表示运行状态一种,false表示停止状态.

b)成员内部类: 发动机(Engine)

i.成员方法:work()

1.如果汽车的运行状态打印发动机就飞速旋转

2.否则打印发动机停止工作

c)提供:状态(status)的getter和setter方法

\*/

**public** **class** Car {

// a)成员变量: 状态(status),true表示运行状态一种,false表示停止状态.

**private** **boolean** status;

// b)成员内部类: 发动机(Engine)

**public** **class** Engine{

// i.成员方法:work()

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

// 1.如果汽车的运行状态打印发动机就飞速旋转

**if**(status){

System.***out***.println("发动机就飞速旋转");

}**else**{

// 2.否则打印发动机停止工作

System.***out***.println("发动机停止工作");

}

}

}

// 状态(status)的getter和setter方法

**public** **boolean** isStatus() {

**return** status;

}

**public** **void** setStatus(**boolean** status) {

**this**.status = status;

}

}

/\*

2.定义测试类(Test)

a)提供main方法

b)在main方法中

i.创建汽车对象 c

ii.通过汽车对象c创建该汽车的发动机对象e

iii.调用发动机e对象的work()方法

iv.把汽车对象c的状态设置为true

v.调用发动机e对象的work()方法

\*/

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

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

// i.创建汽车对象 c

Car c = **new** Car();

// ii.通过汽车对象c创建该汽车的发动机对象e

Car.Engine e = c.**new** Engine();

// iii.调用发动机e对象的work()方法

e.work();

// iv.把汽车对象c的状态设置为true

c.setStatus(**true**);

// v.调用发动机e对象的work()方法

e.work();

}

}

# 关卡2

## 训练案例1

### 训练描述

请简述abstract不能和那些关键一起使用,为什么.

### 操作步骤答案

1. abstract 不可以与private关键字一起使用,因为abstract修饰的要求子类实现(覆盖),而private修饰的只能在本来中使用. 相互矛盾
2. abstract 不可以与static关键字一起使用,因为abstract修饰的要求子类实现(覆盖),而使用static修饰的属与类不属与对象,java中类的东西都不能覆盖, 相互矛盾
3. abstract 不可以与final关键字一起使用, 因为abstract修饰的要求子类实现(覆盖),而final是最终的,它修饰的东西不能被覆盖(重写) ; 相互矛盾

## 训练案例2

### 训练描述

看下面的程序,说出的它的执行结果以及执行流程

**public** **class** Student {

**static** {

System.***out***.println("Student 类的静态代码块");

}

**public** Student(){

System.***out***.println("Student 类的构造方法");

}

**public** Student(String methodName){

System.***out***.println("在"+methodName+"方法中创建Student对象调用的");

}

}

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

**static** {

System.***out***.println("Test 类的静态代码块");

}

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

System.***out***.println("执行main方法");

// 创建学生对象

Student s1 = **new** Student("main");

}

}

提示: 类只有被使用到的时候才会加载

### 操作步骤答案

1. 输出结果

Test 类的静态代码块

执行main方法

Student 类的静态代码块

在main方法中创建对象调用的

1. 执行流程
   1. 加载main方法所在的类Test
   2. 执行Test的静态代码块,输出: Test 类静态代码块
   3. 执行main方法,输出: 执行main方法
   4. 在使用类创建对象之前,首先要把这个加载到内存,所以接下来先加载Student类
   5. 执行Student类的静态代码块:输出:Student类的静态代码块
   6. 接下来执行创建对象代码的,执行Student的有参构造:输出:在main方法中创建Student对象调用的

## 训练案例3

### 训练描述

请写代码验证final关键字的特点

### 操作步骤答案

//1.验证:final修饰类不可以被继承，但是可以继承其他类

//被final修饰的类

**public** **class** Final\_1 {}

//验证:被final修饰的类可以继承其他类

**final** **class** FinalClass **extends** Final\_1{ }

//class Sub extends FinalClass{ } //编译报错

/\*

错误原因: The type Sub cannot subclass the final class FinalClass

//翻译: 被final修饰的类不能有子类; 也就是说被final修饰的类不能被继承

\*/

/\*

2.验证: final修饰的方法不可以被覆盖,但父类中没有被final修饰方法，子类覆盖后可以加final

\*/

**public** **class** Final\_2 {

// 1.被final修饰的成员方法 finalMethod

**public** **final** **void** finalMethod(){}

// 2.普通方法 normalMethod

**public** **void** normalMethod(){}

}

**class** Sub2 **extends** Final\_2{

// final修饰的方法不可以被覆盖

// public final void finalMethod(){}

// 父类中没有被final修饰方法，子类覆盖后可以加final

**public** **final** **void** normalMethod(){}

}

/\*

3.验证: final修饰的变量称为常量，这些变量只能赋值一次

\*/

**public** **class** Final\_3 {

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

**final** **int** a = 10;

// 被final修饰变量为常量,只能赋值一次

// a = 20; //编译报错:The final local variable a cannot be assigned

**final** **int** b;

b = 5;

// 被final修饰变量为常量,只能赋值一次

// b = 10; //编译标错:The final local variable b may already have been assigned

}

}

/\*

4. 验证: 引用类型的变量值为对象地址值，地址值不能更改，但是地址内的对象属性值可以修改

\*/

**public** **class** Final\_4 {

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

// 定义final修饰引用变量,并进行初始化

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

// 引用类型的变量值为对象地址值，地址值不能更改

// c = new Cat();//编译报错

// 地址内的对象属性值可以修改

c.age = 2;

}

}

//定义辅助类:Cat

**class** Cat{

**public** **int** age;

}

/\*

5. 验证:修饰成员变量，需要在创建对象前赋值，否则报错。(当没有显式赋值时，多个构造方法的均需要为其赋值。

\*/

**public** **class** Final\_5 {

// 定义final成员变量a,并显示赋值

**public** **final** **int** a = 10;

**public** **final** **int** b;

**public** Final\_5() {

// 如果把这句话注释,编译报错: The blank final field b may not have been initialized

b = 20;

}

**public** Final\_5(**int** a,**int** b) {

// 如果把这句话注释,编译报错: The blank final field b may not have been initialized

**this**.b = b;

// this.a = a; //被final修饰成员变量只能赋值一次

}

}

## 训练案例4

### 训练描述

请用代码描述:

在某新型公司,每一个员工都有工号,姓名; 都要工作,但是每一个员工的工作内容都不一样;

要求:使用匿名内部类创建对象

### 操作步骤答案

/\*

1.定义抽象类(Employee)

a)属性: 工号(id),姓名(name)

b)行为: 抽象方法work()

c)要求: 提供setters和gettters方法

\*/

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

// 工号

**private** String id;

// 姓名

**private** String name;

// 行为

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

// 提供setters和gettters方法

**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;

}

}

/\*

2.定义测试类Test

a)提供main方法

b)在main方法中

i.使用匿名内部类创建Employee的子类对象 e;并重写work()方法

1.输出格式:工号为001的员工范冰冰在努力的敲代码

ii.使用setXxx方法给e的工号和名称赋值

iii.调用e的work()方法

\*/

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

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

// i.使用匿名内部类创建Employee的子类对象 e;并重写work()方法

// 1.输出格式:工号为001的员工范冰冰在努力的敲代码

Employee e = **new** Employee() {

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

System.***out***.println("工号为"+getId()+"的员工"+getName()+"在努力的敲代码");

}

};

// ii.使用setXxx方法给e的工号和名称赋值

e.setId("001");

e.setName("范冰冰");

// iii.调用e的work()方法

e.work();

}

}

## 训练案例5

### 训练描述

请用代码描述：

某市举办一场运动会,只要是会跑的都可以参加.

要求: 使用到接口,接口作为参数,匿名内部类.

提示: 把跑功能抽取到接口中 ,在测试类中提供参加运动会的静态方法

### 操作步骤答案

/\*

1.定义接口(Sport)

a)抽象方法: 跑(run)

\*/

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

// a)抽象方法: 跑(run)

**public** **abstract** **void** run();

}

/\*

1.定义接口(Sport)

a)抽象方法: 跑(run)

2.定义测试类Test

a)定义静态方法:进入运行会(enter(Sport s)),在方法中调用s的run方法

b)提供main方法

c)在main方法中

i.调用enter(Sport s),传入使用Sport接口的匿名内部类创建的对象;

ii.在匿名内部类中重写run方法

1.输出: 参加运动会,奔跑吧

\*/

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

// a)定义静态方法:进入运行会(enter(Sport s)),在方法中调用s的run方法

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

s.run();

}

// ii.在匿名内部类中重写run方法

// 1.输出: 参加运动会,奔跑吧

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

// i.调用enter(Sport s),传入使用Sport接口的匿名内部类创建的对象;

*enter*(**new** Sport() {

// ii.在匿名内部类中重写run方法

// 1.输出: 参加运动会,奔跑吧

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

System.***out***.println("参加运动会,奔跑吧");

}

});

}

}

## 训练案例6

### 训练描述

请用代码描述：

某高校举成立一个大球俱乐部,只要是会打篮球,踢足球,打排球的可以参加.

要求: 使用到接口,接口作为参数,匿名内部类.

提示: 把打篮球,踢足球,打排球功能抽取到接口中 ,在测试类中提供加入俱乐部的静态方法,参数为接口类;

### 操作步骤答案

/\*

1.定义接口(Player)

a)抽象方法: 打篮球(playBasketball),踢足球(playFootball),打排球(playVolleyball)

\*/

**public** **interface** Player {

// 打篮球(playBasketball)

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

// 踢足球(playFootball)

**public** **abstract** **void** playFootball();

// 打排球(playVolleyball)

**public** **abstract** **void** playVolleyball();

}

/\*

2.定义测试类Test

a)定义静态方法:进入运行会(join(Player p)),在方法中调用p的 打篮球,踢足球,打排球 方法

b)提供main方法

c)在main方法中

i.调用join(Player p),传入使用Player 接口的匿名内部类创建的匿名对象;

ii.在匿名内部类中

1.重写playBasketball()方法

a)输出: 在打篮球

2.重写playFootball()方法

a)输出: 在踢足球

3.重写playVolleyball()方法

a)输出: 在打排球

\*/

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

// a)定义静态方法:进入运行会(join(Player p)),在方法中调用p的 打篮球,踢足球,打排球 方法

**public** **static** **void** join(Player p){

p.playBasketball();

p.playFootball();

p.playVolleyball();

}

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

// i.调用join(Player p),传入使用Player 接口的匿名内部类创建的匿名对象;

*join*(**new** Player() {

// ii.在匿名内部类中

// 1.重写playBasketball()方法

// a)输出: 在打篮球

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

System.***out***.println("打篮球");

}

// 2.重写playFootball()方法

// a)输出: 在踢足球

**public** **void** playFootball() {

System.***out***.println("踢足球");

}

// 3.重写playVolleyball()方法

// a)输出: 在打排球

**public** **void** playVolleyball() {

System.***out***.println("打排球");

}

});

}

}

# 关卡3

## 训练案例1

### 训练描述

看下面的程序说出执行结果并阐述程序的执行流程

**public** **class** Student {

**static** {

System.***out***.println("Student 类的静态代码块");

}

**public** Student(){

System.***out***.println("Student 类的构造方法");

}

**public** Student(String methodName){

System.***out***.println("在"+methodName+"创建Student对象调用的");

}

}

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

// 实例(对象)成员变量

Student s = **new** Student("成员变量显式初始化的时候");

// 静态代码块

**static** {

System.***out***.println("Test 类的静态代码块");

}

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

System.***out***.println("main方法");

// 创建学生对象

Student s1 = **new** Student("main方法中");

// 创建Test对象

Test t = **new** Test();

}

}

提示: 1. 类只有被使用就会加载到内存中

2. 只有创建对象才会为成员变量分配存储空间

### 操作步骤答案

1. 输出结果

Test 类的静态代码块

main方法

Student 类的静态代码块

在main方法中创建Student对象调用的

在成员变量显式初始化的时候创建Student对象调用的

1. 执行流程
   1. 首先加载main方法所在的类Test.class
   2. 然后执行Test类中的静态代码块,输出: Test 类的静态代码块
   3. 接下来执行main方法,输出: mian方法
   4. 创建Student对象之前,先要加载Student.class
   5. 然后执行Student类中的静态代码块,输出: Student 类的静态代码块
   6. 接下来执行main方法中创建Student对象,调用的构造方法,输出:在main方法中创建Student对象调用的
   7. 最后执行创建Test的对象的代码,在创建Test对象时候,会为成员变量分配存储空间,并执行后面的创建Student对象的语句,输出: 在成员变量显式初始化的时候创建Student对象调用的

## 训练案例2

### 训练描述

请使用代码描述:

某人养了几只宠物,每个宠物都有名字,每一个宠物都会吃,但是每个宠物吃的东西都不一样; 宠物1 叫Tom吃鱼,宠物2 叫小黑 啃骨头.

要求: 用到抽象类和匿名内部类

### 操作步骤

1. 定义抽象类宠物(Pet)

1. 属性： 名称：name

2. 抽象方法: eat()

3. 提供带参构造和setXxx和getXxx方法

2. 定义测试类Test

1. 提供main方法

2. 在main方法中

1. 使用匿名内部类利用Pet空参构造,创建对象p1

2. 在匿名内部类中重写eat()方法,输出格式: 宠物1叫Tom在吃鱼,

3. 把p1的name赋值为Tom

1. 调用p1的eat()方法
2. 使用匿名内部类,利用Pet的有参构造,创建对象
   1. 重写eat()方法,输出: 宠物2 小黑在肯骨头
   2. 使用该匿名对象调用eat()方法

### 操作步骤答案

/\*

1. 定义抽象类宠物(Pet)

1. 属性： 名称：name

2. 抽象方法: eat()

3. 提供带参构造和setXxx和getXxx方法

\*/

**public** **abstract** **class** Pet {

// 姓名

**private** String name;

// 抽象方法: eat()

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

// 提供带参构造和setXxx和getXxx方法

**public** Pet() {

}

**public** Pet(String name) {

**this**.name = name;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

}

/\*

2. 定义测试类Test

1. 提供main方法

2. 在main方法中

1. 使用匿名内部类利用Pet空参构造,创建对象p1

2. 在匿名内部类中重写eat()方法,输出格式: 宠物1叫Tom在吃鱼,

3. 把p1的name赋值为Tom

4. 调用p1的eat()方法

5. 使用匿名内部类,利用Pet的有参构造,创建对象

a)重写eat()方法,输出: 宠物2 小黑在肯骨头

b)使用该匿名对象调用eat()方法

\*/

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

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

// 1. 使用匿名内部类利用Pet空参构造,创建对象p1

Pet p1 = **new** Pet() {

// 2. 在匿名内部类中重写eat()方法,输出格式: 宠物1叫Tom在吃鱼,

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

System.***out***.println("宠物1叫"+getName()+"在吃鱼,");

}

};

// 3. 把p1的name赋值为Tom

p1.setName("Tom");

// 4. 调用p1的eat()方法

p1.eat();

// 5. 使用匿名内部类,利用Pet的有参构造,创建对象,名称赋值为小黑

**new** Pet("小黑") {

// a)重写eat()方法,输出: 宠物2 小黑在肯骨头

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

System.***out***.println(" 宠物2 "+getName()+"在肯骨头");

}

// b)使用该匿名对象调用eat()方法

}.eat();

}

}

## 训练案例3

### 训练描述

请使用代码描述

某公司成立一个游泳俱乐部,只要是会游泳的都可以加入;

要求: 使用到接口,接口作为参数,匿名内部类.

提示: 把游泳功能抽取到接口中 ,在测试类中提供参加运动会的静态方法

### 操作步骤描述

1. 定义游泳者接口(Swimmer)
   1. 抽象方法: 游泳swim
2. 定义测试类Test
   1. 定义加入游泳俱乐部的方法: join(Swimmer s)
      1. 在方法中调用s的swim方法
   2. 提供main方法
   3. 在main方法中
      1. 调用join(Swimmer s) 传入使用接口Swimmer的匿名内部类创建的对象
      2. 在匿名内部类中重写swim方法: 输出:愉快的游泳

### 操作步骤答案

/\*

1.定义游泳者接口(Swimmer)

a)抽象方法: 游泳swim

\*/

**public** **interface** Swimmer {

// a) 抽象方法: 游泳swim

**public** **abstract** **void** swim();

}

/\*

2.定义测试类Test

a)定义加入游泳俱乐部的方法: join(Swimmer s)

i.在方法中调用s的swim方法

b)提供main方法

c)在main方法中

i.调用join(Swimmer s) 传入使用接口Swimmer的匿名内部类创建的对象

ii.在匿名内部类中重写swim方法: 输出:愉快的游泳

\*/

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

// a)定义加入游泳俱乐部的方法: join(Swimmer s)

**public** **static** **void** join(Swimmer s){

// i.在方法中调用s的swim方法

s.swim();

}

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

// i.调用join(Swimmer s) 传入使用接口Swimmer的匿名内部类创建的对象

*join*(**new** Swimmer() {

// ii.在匿名内部类中重写swim方法: 输出:愉快的游泳

**public** **void** swim() {

System.***out***.println("愉快的游泳");

}

});

}

}