远动员练习

2018年6月29日 17:22

```
//教练与运动员案例

//定义一个说英语接口
interface SpeakEnglish{
	public abstract void speak();
}

//定义一个抽象类
abstract class Person{
	private String name;
	private int age;
	public Person(){}
	public Person(String name,int age){
		this.name = name;
		this.age = age;
	}
	public String getName(){
		return name;
	}
	public int getAge(){
		return age;
	}
	public void setName(String name){
		this.name = name;
	}
	public void setAge(int age){
		this.age = age;
	}
	public void sleep(){
		System.out.println("睡觉");
	}
	public abstract void eat();
}

//定义运动员抽象类
abstract class Player extends Person{
```

```
	public Player(){}
	public Player(String name,int age){
		super(name,age);
	}
	public abstract void study();
}

//定义教练抽象类
abstract class Coach extends Person{
	public Coach(){}
	public Coach(String name,int age){
		super(name,age);
	}
	public abstract void teach();
}

//定义乒乓球运动员具体类
class PingPangPlayer extends Player implements SpeakEnglish{
	public PingPangPlayer(){}
	public PingPangPlayer(String name,int age){
		super(name,age);
	}
	public void eat(){
		System.out.println("喝粥");
	}
	public void study(){
		System.out.println("打乒乓球");
	}
	public void speak(){
		System.out.println("说英语");
	}
}

//定义篮球运动员具体类
class BasketballPlayer extends Player{
	public BasketballPlayer(){}
	public BasketballPlayer(String name,int age){
		super(name,age);
	}
	public void eat(){
```

```
            System.out.println("喝牛奶");
        }
        public void study(){
            System.out.println("打篮球");
        }
}

//定义乒乓球教练具体类
class PingPangCoach extends Coach implements SpeakEnglish{
        public PingPangCoach(){}
        public PingPangCoach(String name,int age){
            super(name,age);
        }
        public void eat(){
            System.out.println("喝大米粥");
        }
        public void teach(){
            System.out.println("教打乒乓球");
        }
        public void speak(){
            System.out.println("说英语");
        }
}

//定义篮球教练具体类
class BasketballCoach extends Coach{
        public BasketballCoach(){}
        public BasketballCoach(String name,int age){
            super(name,age);
        }
        public void eat(){
            System.out.println("吃肉");
        }
        public void teach(){
            System.out.println("教打篮球");
        }
}

class InterfaceDemo{
        public static void main(String[] args){
```

```
        PingPangPlayer pp = new PingPangPlayer("王若潇",22);
        System.out.println(pp.getName()+"----"+pp.getAge());
        pp.eat();
        pp.study();
        pp.sleep();
        pp.speak();
    }
}
```

类名作为形式参数

2018年6月29日 17:23

匿名对象用法

new StudentDemo().method(new Student());

```
/*
形式参数：
	基本类型
	引用类型
		类（匿名对象）需要的是该类的对象
		抽象类
		接口
*/

class Student{
	public void study(){
		System.out.println("好好学习");
	}
}

class StudentDemo{
	public void method(Student s){
		s.study();
	}
}

class StudentTest{
	public static void main(String[] args){
		Student s = new Student();
		s.study();
		System.out.println("--------");
		StudentDemo sd = new StudentDemo();
		s = new Student();
		sd.method(s);
	}
}
```

抽象类名作为形式参数

2018年6月29日 17:41

需要该抽象类的子类对象

```
abstract class Person{
	public abstract void study();
}

class PersonDemo{
	public void method(Person p){
		p.study();
	}
}

//先定义一个具体类
class Student extends Person{
	public void study(){
		System.out.println("好好学习");
	}
}
class PersonTest{
	public static void main(String[] args){
		PersonDemo pd = new PersonDemo();
		Person p = new Student();
		pd.method(p);
	}
}
```

接口名作为形式参数

2018年6月29日 18:37

```
//定义一个爱好接口
interface Love{
	public abstract void love();
}

class LoveDemo{
	public void method(Love l){
		l.love();
	}
}

//定义具体类实现方法
class Teacher implements Love{
	public void love(){
		System.out.println("爱学术");
	}
}

class TeacherTest{
	public static void main(String[] args){
		LoveDemo ld = new LoveDemo();
		Love l = new Teacher();
		ld.method(l);
	}
}
```

类名为返回值类型

2018年6月29日 18:50

返回值类型：

- 基本类型：简单
- 引用类型：
  - 类
  - 抽象类
  - 接口

```
class Student{
	public void study(){
		System.out.println("好好学习");
	}
}

class StudentDemo{
	public Student getStudent(){
		return new Student();
	}
}

class StudentTest2{
	public static void main(String[] args){
		StudentDemo sd = new StudentDemo();
		Student s = sd.getStudent();
		s.study();
	}
}
```

抽象类名为返回值类型

2018年6月29日 19:06

返回的是抽象类的子类对象

```
abstract class Person{
	public abstract void study();
}

class PersonDemo{
	public Person getPerson(){
		return new Student();
	}
}

class Student extends Person{
	public void study(){
		System.out.print("学习");
	}
}

class PersonTest2{
	public static void main(String[] args){
		PersonDemo pd = new PersonDemo();
		Person p = pd.getPerson();
		p.study();
	}
}
```

接口为返回值类型

2018年6月29日 19:12

返回的是该接口的实现类对象

```
interface Love{
	public abstract void love();
}

class LoveDemo{
	public Love getLove(){
		return new Teacher();
	}
}

class Teacher implements Love{
	public void love(){
		System.out.println("爱学术");
	}
}

class TeacherTest2{
	public static void main(String[] args){
		LoveDemo ld = new LoveDemo();
		Love l = ld.getLove();
		l.love();
	}
}
```

链式编程

2018年6月29日 19:16

每次调用完毕后，返回的是一个对象

```
class Student{
	public void study(){
		System.out.println("好好学习");
	}
}

class StudentDemo{
	public Student getStudent(){
		return new Student();
	}
}

class StudentTest3{
	public static void main(String[] args){
		StudentDemo sd = new StudentDemo();
		sd.getStudent().study();//链式编程
	}
}
```

包概述

2018年6月29日 20:01

包：

1.其实就是文件夹

2.作用：

（1）把相同的类名放到不同的包中

（2）对类进行分类管理

方案1：按照功能分

方案2：按照模块分

package

包的定义及注意事项

2018年6月29日 20:08

```
/*
包的定义
	package 包名
	多级包用.分开即可
注意事项：
	package语句必须是程序的第一条可执行的代码
	package在一个java文件中只能有一个
	如果没有package，默认表示无包名
带包的编译和运行：
	A.手动式：
		1.编写一个带包的java文件
		2.通过javac命令编译该java文件
		3.手动创建包名
		4.把第二步的class文件放在第三步的最底层包中
		5.回到和包根目录在同一目录的地方，然后运行
			带包运行 java cn.itcast.HelloWorld
	B.自动式
		1.编写文件
		2.javac编译的时候带上-d即可
			javac -d . HelloWorld.java
		3.回到和包根目录在同一目录的地方，然后运行
			带包运行 java cn.itcast.HelloWorld
*/

package cn.itcast;

class HelloWorld{
	public static void main(String[] args){
		System.out.println("hello world");
	}
}
```

不同包之间的访问

2018年6月29日 20:43

```
package com.xiao;

public class Demo{
	public int sum(int a,int b){
		return a+b;
	}
}
```

```
package cn.itcast;

class Test{
	public static void main(String[] args){
		com.xiao.Demo d = new com.xiao.Demo();
		System.out.println(d.sum(3,4));
	}
}
```

```
javac -d . Demo.java
javac -d . Test.java
java cn.itcast.Test
```

导包

2018年6月30日 10:29

```
/*
导包:
	格式： import 包名;
	这种方法导入的是类的名称
	注意:用谁导谁
面试题:
	package import class 有没有顺序关系?
	有package>import>class
	package:只能有一个
	import：可以有多个
	class：可以有多个，建议是一个
*/

package cn.itcast;

import com.xiao.Demo;

class Test{
	public static void main(String[] args){
		Demo d = new Demo();
		System.out.println(d.sum(3,4));
	}
}
```

修饰符

2018年6月30日 10:36

```
/*
权限修饰符:
                    本类    同一个包下（子类或无关类）        不同包下（子类）                不同包下（无关类）
    private         Y
    默认            Y           Y
    protected       Y           Y                                   Y
    public          Y           Y                                   Y                       Y
*/
package com.xiao;
public class Father{
    private void show(){
        System.out.println("show");
    }

    void show2(){
        System.out.println("show2");
    }

    protected void show3(){
        System.out.println("show3");
    }

    public void show4(){
        System.out.println("show4");
    }

    public static void main(String[] args){
        Father f = new Father();
        f.show();
        f.show2();
        f.show3();
        f.show4();
    }
}
```

常见修饰符

2018年6月30日 13:56

修饰符

权限修饰符：private、默认的、protect、private

状态修饰符：static、final

抽象修饰符：abstract

类：
- 默认修饰符 public
- ~~private~~ ~~protected~~
- ~~static~~
- finial
- abstract

用的最多是 public

成员变量：都可以修饰（除 abstract）

public static final int X = 10;

最常用的是 private

构造方法：只能使用权限修饰符

最常用 public

成员方法：都可以修饰

最常用 public

除此以外的组合：

成员变量 public static final

内部类

2018年6月30日 14:13

把类定义在其它类的内部，这个类就被称为内部类

内部类的访问特点：
1.内部类可以直接访问外部类的成员，包括私有。
2.外部类要访问内部类的成员，必须创建对象

内部类位置

2018年6月30日 14:34

内部类位置
- 成员位置类
- 局部位置类

```
//成员位置类

class Outer{
	private int num = 10;
	//成员位置
	class Inner{
		public void show(){
			System.out.println(num);
		}
	}
}


class InnerClassDemo2{
	public static void main(String[] args){
		Outer.Inner oi = new Outer().new Inner();
		oi.show();
	}
}
```

成员内部类修饰符

2018年6月30日 14:58

```
//成员内部类的修饰符

class Body{
	private class Heart{
		public void operateor(){
			System.out.println("心脏搭桥");
		}
	}

	public void method(){
		Heart h = new Heart();
		h.operateor();
	}
}

class Outer{
	private int num =10;
	private static int num2 =10;
	//注意：静态内部类访问的外部类数据必须用静态修饰
	public static class Inner{
		public void show(){
			//System.out.println(num);
			System.out.println(num2);
		}
		public static void show2(){
			//System.out.println(num);
			System.out.println(num2);
		}
	}
}

class InnerClassDemo3{
	public static void main(String[] args){
		Body b = new Body();
		b.method();

		//成员内部类被静态修饰后的访问方式
		Outer.Inner oi = new Outer.Inner();
		oi.show();
		oi.show2();

		//show2的另一种调用方式
		Outer.Inner.show2();
	}
}
```

练习

```
/*
    面试题：
        要求请填空分别输出30，20，10。

    注意：
        1:内部类和外部类没有继承关系。
        2:通过外部类名限定this对象
            Outer.this
*/
class Outer {
    public int num = 10;
    class Inner {
        public int num = 20;
        public void show() {
            int num = 30;
            System.out.println(num);
            System.out.println(this.num);
            //System.out.println(new Outer().num);
            System.out.println(Outer.this.num);
        }
    }
}
class InnerClassTest {
    public static void main(String[] args) {
        Outer.Inner oi = new Outer().new Inner();
```

局部内部类

2018年6月30日 15:11

```
/*
局部内部类
	1.可以直接访问外部类的成员
	2.在局部位置可以创建内部类对象，通过对象调用内部类方法，来使用局部内部类功能
面试题：
	1.局部内部类访问局部变量必须用final修饰
	2.原因：局部变量随着方法的调用完毕而消失，而堆内存不会消失，用final修饰
	加入final后，变量变为常量，内存中储存的数据为20
*/
class Outer{
	private int num = 10;
	public void method(){
		final int num2 = 10
		class Inner{
			final void show(){
				System.out.println(num2);
			}
		}
		Inner i = new Inner();
		i.show();
	}
}

class InnerClassDemo4{
	public static void main(String[] args){
		Outer o = new Outer();
		o.method();
	}
}
```

匿名内部类

2018年6月30日 16:04

```
/*
匿名内部类：内部类的简化写法
前提：存在一个类或接口
格式：new 类名或接口名(){
        方法重写;
}
本质是继承了该类或实现了该接口的子类匿名对象
*/
interface Inter{
        public abstract void show();
        public abstract void show2();
}

class Outer{
        public void method(){
                /*
                一个方法的时候
                new Inter(){
                        public void show(){
                                System.out.println("show");
                        }
                }.show();
                */

                //两个方法的时候
                Inter i = new Inter(){ //多态
                        public void show(){
                                System.out.println("show");
                        }
                        public void show2(){
                                System.out.println("show2");
                        }
                };
                i.show();
                i.show2();
        }
```

```
}

class InnerClassDemo5{
	public static void main(String[] args){
		Outer o = new Outer();
		o.method();
	}
}
```

匿名内部类的应用

2018年6月30日 16:33

安卓中常用

```
//定义一个爱好接口
interface Love{
	public abstract void love();
}

class LoveDemo{
	public void method(Love l){
		l.love();
	}
}

//定义具体类实现方法
class Teacher implements Love{
	public void love(){
		System.out.println("爱学术");
	}
}

class TeacherTest{
	public static void main(String[] args){
		LoveDemo ld = new LoveDemo();
		Love l = new Teacher();
		ld.method(l);
		System.out.println("--------------");
		ld.method(new Love(){
			public void love(){
				System.out.println("爱学术");
			}
		});
	}
}
```