第08天 java面向对象

今日内容介绍

* final和static关键字
* 抽象类
* 接口的概述以及练习
* 包的概述和权限修饰符

# final和static关键字

## final关键字的概述及特点

* final关键字是最终的意思，可以修饰类，成员变量，成员方法。
  + 修饰类，类不能被继承
  + 修饰变量，变量就变成了常量，只能被赋值一次
  + 修饰方法，方法不能被重写

### 案例代码一

package com.itheima;

/\*

public final class Father {

}

\*/

public class Father {

public final void method() {

System.out.println("method father");

}

}

package com.itheima;

public class Son extends Father {

public final int age = 20;

public void show() {

//age = 10;

System.out.println(age);

}

/\*

@Override

public void method() {

System.out.println("method son");

}

\*/

}

package com.itheima;

/\*

\* final：是一个关键字，表示最终的意思。可以用来修饰类，修饰变量，修饰方法。

\* 修饰类：表明该类是最终类，不能被继承

\* 修饰变量：表明该变量是常量，不能再次被赋值

\* 修饰方法：表明该方法是最终方法，不能被重写

\*/

public class FinalDemo {

public static void main(String[] args) {

Son s = new Son();

s.show();

}

}

## static关键字的概述及特点

### 静态的概述

当在定义类的时候，类中都会有相应的属性和方法。而属性和方法都是通过创建本类对象调用的。当在调用对象的某个方法时，这个方法没有访问到对象的特有数据时，方法创建这个对象有些多余。可是不创建对象，方法又调用不了，这时就会想，那么我们能不能不创建对象，就可以调用方法呢？

可以的，我们可以通过static关键字来实现。static它是静态修饰符，一般用来修饰类中的成员。

### 静态的特点

* + 被类的所有对象共享
    - 这也是我们判断是否使用静态关键字的条件
  + 可以通过类名调用
  + 优先于对象存在
  + 随着类的加载而加载

### 案例代码二

package com.itheima\_01;

/\*

\* static：是一个关键字，静态的意思。可以用来修饰成员变量和成员方法。

\* static修饰成员的特点：

\* A:被类的所有对象共享。

\* 其实也是判断一个成员是否应该用static修饰的条件。

\* B:可以通过类名直接访问

\* C:优先于对象存在

\* D:随着类的加载而加载

\*/

public class StaticDemo {

public static void main(String[] args) {

Student.graduateFrom = "传智学院";

Student s1 = new Student();

s1.name = "林青霞";

s1.age = 30;

//s1.graduateFrom = "传智学院";

s1.show();

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

Student s2 = new Student();

s2.name = "刘德华";

s2.age = 28;

//s2.graduateFrom = "传智学院";

s2.show();

}

}

package com.itheima\_01;

public class Student {

public String name;

public int age;

//public String graduateFrom; //毕业院校

public static String graduateFrom; //毕业院校

public void show() {

System.out.println(name+"---"+age+"---"+graduateFrom);

}

}

## static方法的访问特点及注意事项

* 静态方法的访问特点
  + 静态方法只能访问静态的成员变量和静态的成员方法
  + 静态方法的注意事项
  + 在静态方法中是没有this,super关键字的
    - 静态的内容是随着类的加载而加载，this和super是随着对象的创建而存在。

### 案例代码三

package com.itheima\_02;

/\*

\* 非静态的成员方法：

\* 能访问静态的成员变量

\* 能访问非静态的成员变量

\* 能访问静态的成员方法

\* 能访问非静态的成员方法

\*

\* 静态的成员方法：

\* 能访问静态的成员变量

\* 能访问静态的成员方法

\*

\* 注意事项：

\* 静态成员方法中不能出现this,super这样的关键字。

\* 原因是：静态是随着类的加载而加载，this,super这样的关键字是随着对象的创建而存在。

\* 先进内存的，不能访问后进内存的。

\*/

public class Student {

//非静态的成员变量

private String name = "林青霞";

//静态的成员变量

private static int age = 30;

//非静态的成员方法

public void show() {

this.name = "刘德华";

System.out.println(name);

System.out.println(age);

show2();

show4();

}

public void show2() {}

//静态的成员方法

public static void show3() {

//this.age

//this.name

//System.out.println(name);

System.out.println(age);

//show2();

show4();

}

public static void show4() {}

}

# 抽象类

## 抽象类概述

当编写一个类时，我们往往会为该类定义一些方法，这些方法是用来描述该类的功能具回想前面我们的猫狗案例，提取出了一个动物类，这个时候我们是可以通过Animal a = new Animal()来创建动物对象的，其实这是不对的。为什么呢?因为，我说动物，你知道我说的是什么动物吗?只有看到了具体的动物，你才知道，这是什么动物。 所以说，动物本身并不是一个具体的事物，而是一个抽象的事物。只有真正的猫，狗才是具体的动物。同理，我们也可以推想，不同的动物吃的东西应该是不一样的，所以，我们不应该在动物类中给出具体体现，而是应该给出一个声明即可。在Java中，一个没有方法体的方法应该定义为抽象方法，而类中如果有抽象方法，该类必须定义为抽象类。

### 案例代码四

package com.itheima\_01;

public class AnimalDemo {

public static void main(String[] args) {

/\*

Animal a = new Animal();

a.eat();

\*/

}

}

package com.itheima\_01;

public abstract class Animal {

/\*

public void eat() {

System.out.println("吃东西");

}

\*/

//抽象方法

public abstract void eat();

}

## 抽象类的特点

抽象类和抽象方法必须用abstract关键字修饰

格式:

public abstract class 类名 {}

public abstract void eat();

抽象类不一定有抽象方法，有抽象方法的类一定是抽象类

抽象类不能实例化

那么，抽象类如何实例化呢?

按照多态的方式，由具体的子类实例化。其实这也是多态的一种，抽象类多态。

抽象类的子类

要么是抽象类

要么重写抽象类中的所有抽象方法

### 案例代码五

package com.itheima\_02;

/\*

\* 抽象类的特点：

\* A:抽象类和抽象方法必须使用abstract关键字修饰

\* B:抽象类中不一定有抽象方法，有抽象方法的类一定是抽象类

\* C:抽象类不能实例化

\* 抽象类如何实例化呢?

\* 参照多态的方式，通过子类对象实例化。

\* D:抽象类的子类

\* 要么重写抽象类中的所有抽象方法

\* 要么是抽象类

\*/

public class AnimalDemo {

public static void main(String[] args) {

//创建对象

//Animal a = new Animal();

//按照多态的形式实例化抽象类

Animal a = new Cat();

a.eat();

a.sleep();

}

}

package com.itheima\_02;

public abstract class Dog extends Animal {

}

package com.itheima\_02;

public class Cat extends Animal {

@Override

public void eat() {

System.out.println("猫吃鱼");

}

}

package com.itheima\_02;

//抽象类

public abstract class Animal {

//抽象方法

public abstract void eat();

public void sleep() {

System.out.println("睡觉");

}

}

## 抽象类的成员的特点

* 成员变量
  + 可以是变量
  + 也可以是常量
* 构造方法
  + 有构造方法，但是不能实例化
  + 那么，构造方法的作用是什么呢?
    - 用于子类访问父类数据的初始化
* 成员方法
  + 可以有抽象方法 限定子类必须完成某些动作
  + 也可以有非抽象方法 提高代码复用性

### 案例代码六

package com.itheima\_03;

/\*

\* 抽象类的成员特点：

\* 成员变量：

\* 有成员变量，成员变量可以是变量，也可以是常量。

\* 构造方法：

\* 有构造方法。

\* 抽象类中构造方法的作用?

\* 用于子类访问父类数据的初始化。

\* 成员方法：

\* 有成员方法，成员方法可以是抽象的，也可以是非抽象的。

\* 抽象方法：限定子类必须完成某些动作

\* 非抽象方法：提高代码的复用性

\*/

public class PersonDemo {

public static void main(String[] args) {

Person p = new Student();

p.show();

}

}

package com.itheima\_03;

public class Student extends Person {

@Override

public void eat() {

System.out.println("学生吃米饭");

}

}

package com.itheima\_03;

public abstract class Person {

private int age = 20;

private final String country = "中国";

public Person() {}

public Person(int age) {

this.age = age;

}

public void show() {

age = 30;

System.out.println(age);

//country = "美国";

System.out.println(country);

}

public abstract void eat();

}

## 抽象类的练习之老师案例

* 老师案例
  + 具体事物：基础班老师，就业班老师
  + 共性：姓名，年龄，讲课。

### 案例代码七

package com.itheima\_04;

/\*

\* 分析：从具体到抽象

\* 实现：从抽象到具体

\* 使用：使用的是具体的类的对象

\*

\* 分析：

\* package com.itheima\_04;

/\*

\* 分析：从具体到抽象

\* 实现：从抽象到具体

\* 使用：使用的是具体的类的对象

\*

\* 分析：

\* 基础班老师：

\* 成员变量：name,age

\* 构造方法：无参，带参

\* 成员方法：getXxx(),setXxx(),teach(){}

\* 就业班老师：

\* 成员变量：name,age

\* 构造方法：无参，带参

\* 成员方法：getXxx(),setXxx(),teach(){}

\*

\* 抽象的老师类：

\* 成员变量：name,age

\* 构造方法：无参，带参

\* 成员方法：getXxx(),setXxx(),teach();

\*/

public class TeacherDemo {

public static void main(String[] args) {

//使用的是具体的类的对象

//BasicTeacher

//多态形式的测试

Teacher t = new BasicTeacher();

t.setName("林青霞");

t.setAge(30);

System.out.println(t.getName()+"---"+t.getAge());

t.teach();

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

t = new BasicTeacher("林青霞", 30);

System.out.println(t.getName()+"---"+t.getAge());

t.teach();

}

} 基础班老师：

\* 成员变量：name,age

\* 构造方法：无参，带参

\* 成员方法：getXxx(),setXxx(),teach(){}

\* 就业班老师：

\* 成员变量：name,age

\* 构造方法：无参，带参

\* 成员方法：getXxx(),setXxx(),teach(){}

\*

\* 抽象的老师类：

\* 成员变量：name,age

\* 构造方法：无参，带参

\* 成员方法：getXxx(),setXxx(),teach();

\*/

public class TeacherDemo {

public static void main(String[] args) {

//使用的是具体的类的对象

//BasicTeacher

//多态形式的测试

Teacher t = new BasicTeacher();

t.setName("林青霞");

t.setAge(30);

System.out.println(t.getName()+"---"+t.getAge());

t.teach();

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

t = new BasicTeacher("林青霞", 30);

System.out.println(t.getName()+"---"+t.getAge());

t.teach();

}

}

package com.itheima\_04;

//就业班老师

public class WorkTeacher extends Teacher {

public WorkTeacher() {}

public WorkTeacher(String name,int age) {

super(name,age);

}

@Override

public void teach() {

System.out.println("就业班老师讲解JavaEE的内容");

}

}

package com.itheima\_04;

//基础班老师

public class BasicTeacher extends Teacher {

public BasicTeacher() {}

public BasicTeacher(String name,int age) {

super(name,age);

}

@Override

public void teach() {

System.out.println("基础班老师讲解JavaSE的内容");

}

}

package com.itheima\_04;

//抽象的老师类

public abstract class Teacher {

private String name;

private int age;

public Teacher() {}

public Teacher(String name,int age) {

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;

}

//抽象方法

public abstract void teach();

}

# 接口的概述以及练习

## 接口的概述

继续回到我们的猫狗案例，我们想想狗一般就是看门，猫一般就是作为宠物了。但是，现在有很多的驯养员或者是驯兽师，可以训练出：猫钻火圈，狗跳高，狗做计算等。而这些额外的动作，并不是所有猫或者狗一开始就具备的，这应该属于经过特殊的培训训练出来的。所以，这些额外的动作定义到动物类中就不合适，也不适合直接定义到猫或者狗中，因为只有部分猫狗具备这些功能。所以，为了体现事物功能的扩展性，Java中就提供了接口来定义这些额外功能，并不给出具体实现，将来哪些猫狗需要被训练，只需要这部分猫狗把这些额外功能实现即可。

## 接口的特点

* 接口用关键字interface表示
  + 格式：public interface 接口名 {}
  + 类实现接口用implements表示
    - 格式：public class 类名 implements 接口名 {}
  + 接口不能实例化
    - 那么，接口如何实例化呢?
    - 按照多态的方式，由具体的实现类实例化。其实这也是多态的一种，接口多态。
  + 接口的实现类
    - 要么是抽象类

要么重写接口中的所有抽象方法

### 案例代码八

package com.itheima\_01;

/\*

\* 接口的特点：

\* A:定义接口使用的是interface关键字

\* B:类和接口之间是实现关系，用implements关键字表示

\* C:接口不能实例化

\* 接口有没有其他的方式实例化呢?

\* 参照多态的形式使用实现类来实例化。

\* D:接口的实现类

\* 要么重写接口中的所有的抽象方法

\* 要么是一个抽象类

\*

\* 多态的几种形式：

\* 具体类多态(几乎不用)

\* 抽象类多态(常用)

\* 接口多态(最常用)

\*/

public class InterfaceDemo {

public static void main(String[] args) {

//Jumpping j = new Jumpping();

//接口多态的形式实例化

Jumpping j = new Cat();

j.jump();

}

}

package com.itheima\_01;

public class Cat implements Jumpping {

@Override

public void jump() {

System.out.println("猫可以跳高了");

}

}

package com.itheima\_01;

public abstract class Dog implements Jumpping {

}

package com.itheima\_01;

//定义了一个跳高的接口

public interface Jumpping {

//抽象方法

public abstract void jump();

}

## 接口的成员特点

* 成员变量
  + 只能是常量
  + 默认修饰符 public static final
* 构造方法
  + 没有，因为接口主要是扩展功能的，而没有具体存在
* 成员方法
  + 只能是抽象方法
  + 默认修饰符 public abstract

### 案例代码九

package com.itheima\_02;

/\*

\* 接口的成员特点：

\* 成员变量：

\* 有成员变量，而且变量只能是常量。

\* 默认修饰符：public static final

\* 构造方法：

\* 没有构造方法。

\* 成员方法：

\* 有成员方法，而且都是抽象的。

\* 默认修饰符：public abstract

\*

\* Object:是类层次结构的根类，所有的类都直接的或者间接的继承自该类。

\*/

public class InterfaceDemo {

public static void main(String[] args) {

//按照多态的形式创建接口对象

Inter i = new InterImpl();

//i.num = 30;

//System.out.println(i.num);

//i.num2 = 40;

//System.out.println(i.num2);

System.out.println(Inter.num);

System.out.println(Inter.num2);

}

}

package com.itheima\_02;

//public class InterImpl implements Inter

public class InterImpl extends Object implements Inter {

public InterImpl() {

super();

}

@Override

public void method() {

}

}

package com.itheima\_02;

public interface Inter {

public int num = 10;

public final int num2 = 20;

public static final int num3 = 30;

//public Inter() {}

//public void show() {}

public abstract void method();

}

## 类与类\_类与接口\_接口与接口的关系

* 类与类
  + 继承关系，只能单继承，但是可以多层继承
* 类与接口
  + 实现关系，可以单实现，也可以多实现。还可以在继承一个类的同时实现多个接口
* 接口与接口
  + 继承关系，可以单继承，也可以多继承

### 案例代码十

package com.itheima\_03;

/\*

\* 类与类：

\* 继承关系，只能单继承，可以多层继承。

\*

\* 类与接口：

\* 实现关系，可以单实现，也可以多实现。

\* 还可以在继承一个类的同时实现多个接口。

\*

\* 接口与接口：

\* 继承关系，可以单继承，也可以多继承。

\*/

public class InterfaceDemo {

}

package com.itheima\_03;

public interface Sister extends Father,Mother {

}

package com.itheima\_03;

public class Son extends Object implements Father,Mother {

}

package com.itheima\_03;

public interface Mother {

}

package com.itheima\_03;

public interface Father {

}

## 抽象类与接口的区别

* 成员区别

抽象类 变量,常量;有抽象方法;抽象方法,非抽象方法

接口 常量;抽象方法

* 关系区别

类与类 继承，单继承

类与接口 实现，单实现，多实现

接口与接口 继承，单继承，多继承

* 设计理念区别

抽象类 被继承体现的是：”is a”的关系。共性功能

接口 被实现体现的是：”like a”的关系。扩展功能

### 案例代码十

package com.itheima\_04;

/\*

\* 抽象类和接口的区别：

\* A:成员区别

\* 抽象类：

\* 成员变量：可以是变量，也可以是常量

\* 构造方法：有

\* 成员方法：可以是抽象方法，也可以是非抽象方法

\* 接口：

\* 成员变量：只能是常量

\* 成员方法：只能是抽象方法

\* B:关系区别

\* 类与类：继承关系，只能单继承，可以多层继承

\* 类与接口：实现关系，可以单实现，也可以多实现

\* 接口与接口：继承关系，可以单继承，也可以多继承

\* C:设计理念的区别

\* 抽象类 被继承体现的是："is a" 抽象类中定义的是继承体系的共性功能

\* 接口 被实现体现的是："like a" 接口中定义的是该体系的扩展功能

\*

\* 举例：

\* 猫，动物

\* 猫，跳高运动员

\*/

public class InterfaceDemo {

}

## 接口的练习

### 接口的练习之猫狗案例

#### 案例代码十一

package com.itheima\_01;

/\*

\* 需求：猫狗案例,让所有的猫狗具备跳高的额外功能

\*

\* 分析：从具体到抽象

\* 猫：姓名，年龄，吃饭(){}

\* 狗：姓名，年龄，吃饭(){}

\* 发现了共性的内容，就提取了一个父类。

\* 抽象动物类：

\* 姓名，年龄，吃饭();

\* 猫：继承动物类

\* 狗：继承动物类

\*

\* 跳高的额外功能是一个扩展功能，所以应该定义接口实现。

\* 跳高接口：

\* 跳高();

\* 猫：继承动物类,实现跳高接口

\* 狗：继承动物类,实现跳高接口

\* 实现：从抽象到具体

\* 使用：使用的是具体的类的对象

\*

\* 作业：具体的狗类，模仿着猫类给出。

\* 狗类的测试，也模仿着猫类的测试给出。

\*

\*/

public class InterfaceTest {

public static void main(String[] args) {

Cat c = new Cat();

c.setName("加菲猫");

c.setAge(3);

System.out.println(c.getName()+"---"+c.getAge());

c.eat();

c.jump();

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

Cat c2 = new Cat("加菲猫",3);

System.out.println(c2.getName()+"---"+c2.getAge());

c2.eat();

c2.jump();

}

}

package com.itheima\_01;

//具体的猫类

public class Cat extends Animal implements Jumpping {

public Cat() {

}

public Cat(String name, int age) {

super(name, age);

}

@Override

public void jump() {

System.out.println("猫可以跳高了");

}

@Override

public void eat() {

System.out.println("猫吃鱼");

}

}

package com.itheima\_01;

//跳高接口

public interface Jumpping {

public abstract void jump();

}

package com.itheima\_01;

//抽象的动物类

public abstract class Animal {

private String name;

private int age;

public Animal() {}

public Animal(String name,int age){

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;

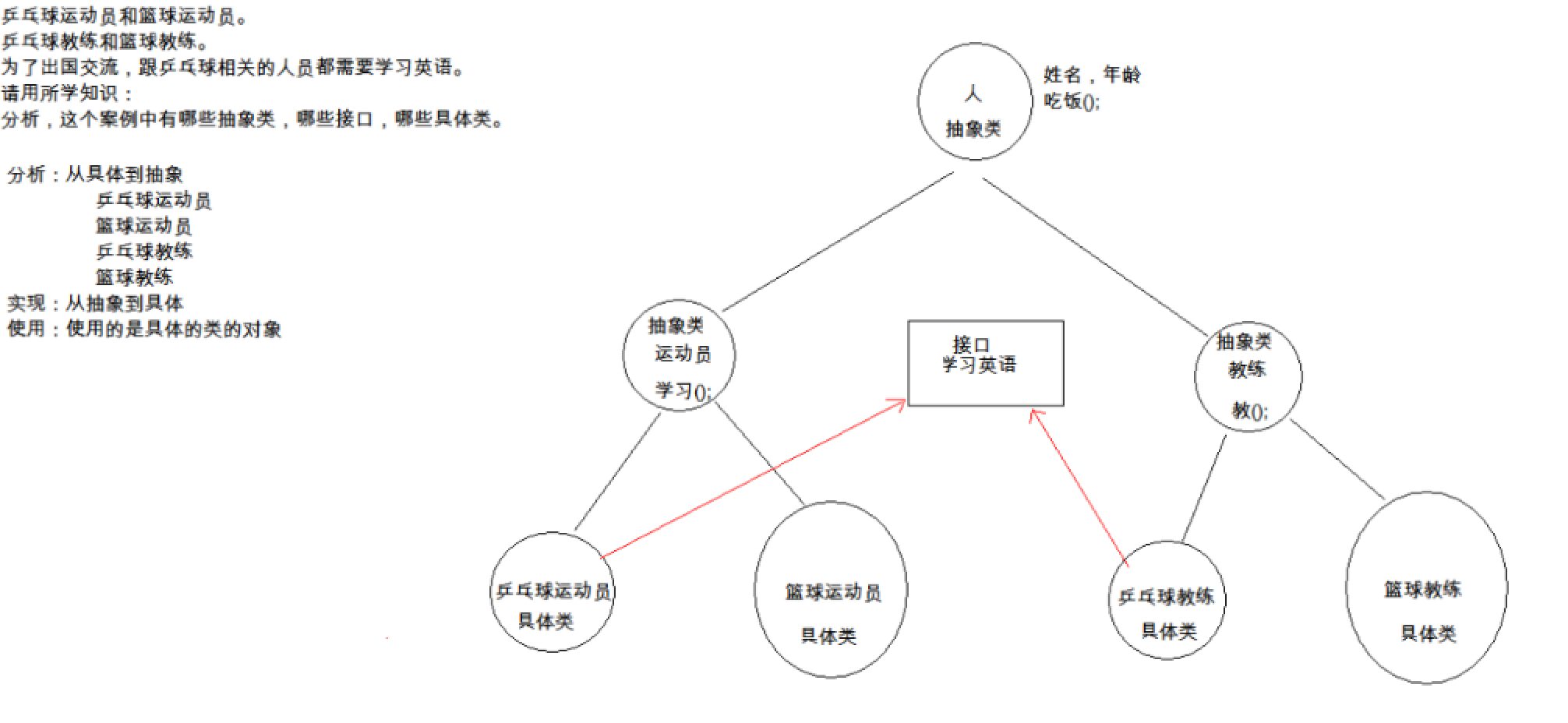
}

public abstract void eat();

}

### 接口的练习之运动员和教练案例

#### 接口的练习之运动员和教练案例分析



#### 接口的练习之运动员和教练案例的代码实现

##### 案例代码十二

package com.itheima\_02;

//乒乓球教练具体类

public class PingPangCoach extends Coach implements SpeakEnglish {

public PingPangCoach() {}

public PingPangCoach(String name,int age) {

super(name,age);

}

@Override

public void speak() {

System.out.println("乒乓球教练说英语");

}

@Override

public void teach() {

System.out.println("乒乓球教练教如何发球和接球");

}

@Override

public void eat() {

System.out.println("乒乓球教练吃小白菜，喝大米粥");

}

}

package com.itheima\_02;

//乒乓球运动员具体类

public class PingPangPlayer extends Player implements SpeakEnglish {

public PingPangPlayer() {

}

public PingPangPlayer(String name, int age) {

super(name, age);

}

@Override

public void speak() {

System.out.println("乒乓球运动员说英语");

}

@Override

public void study() {

System.out.println("乒乓球运动员学习如何发球和接球");

}

@Override

public void eat() {

System.out.println("乒乓球运动员吃大白菜，喝小米粥");

}

}

package com.itheima\_02;

//篮球教练具体类

public class BasketballCoach extends Coach {

public BasketballCoach() {}

public BasketballCoach(String name,int age) {

super(name,age);

}

@Override

public void teach() {

System.out.println("篮球教练教如何运球和投篮");

}

@Override

public void eat() {

System.out.println("篮球教练吃羊肉，喝羊奶");

}

}

package com.itheima\_02;

//篮球运动员具体类

public class BasketballPlayer extends Player {

public BasketballPlayer() {}

public BasketballPlayer(String name,int age) {

super(name,age);

}

@Override

public void study() {

System.out.println("篮球运动员学习如何运球和投篮");

}

@Override

public void eat() {

System.out.println("篮球运动员吃牛肉，喝牛奶");

}

}

package com.itheima\_02;

//抽象的教练类

public abstract class Coach extends Person {

public Coach() {}

public Coach(String name,int age){

super(name,age);

}

public abstract void teach();

}

package com.itheima\_02;

public interface SpeakEnglish {

public abstract void speak();

}

package com.itheima\_02;

//抽象的运动员类

public abstract class Player extends Person {

public Player() {}

public Player(String name,int age) {

super(name,age);

}

public abstract void study();

}

package com.itheima\_02;

//抽象的人类

public 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 void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public abstract void eat();

}

#### 接口的练习之运动员和教练案例的测试

##### 案例代码十三

package com.itheima\_02;

/\*

\* 运动员和教练案例

\*/

public class InterfaceTest {

public static void main(String[] args) {

//测试运动员

//乒乓球运动员

PingPangPlayer ppp = new PingPangPlayer();

ppp.setName("王浩");

ppp.setAge(33);

System.out.println(ppp.getName()+"---"+ppp.getAge());

ppp.eat();

ppp.study();

ppp.speak();

//通过带参构造方法给成员变量赋值的方式(依然留给同学们)

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

//篮球运动员

BasketballPlayer bp = new BasketballPlayer();

bp.setName("姚明");

bp.setAge(35);

System.out.println(bp.getName()+"---"+bp.getAge());

bp.eat();

bp.study();

//测试教练的代码留给同学们实现

}

}

# 包的概述和权限修饰符

## 包的概述和注意事项

### 包的概述

* + 其实就是文件夹
  + 作用：对类进行分类管理
  + 包的划分：
    - 举例：
      * 学生的增加，删除，修改，查询
      * 老师的增加，删除，修改，查询
      * 以及以后可能出现的其他的类的增加，删除，修改，查询
      * 基本的划分：按照模块和功能分。
      * 高级的划分：就业班做项目的时候你就能看到了。

### 定义包的格式

* + 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语句在一个java文件中只能有一个

### 案例代码十四

package com.itheima;

//package com.itheima2;

/\*

\* 包：其实就是文件夹。

\* 作用：就是对类进行分类管理。

\*

\* 举例：

\* 学生：增加，删除，修改，查询

\* 老师：增加，删除，修改，查询

\* ...

\*

\* 方案1：按照功能分

\* com.itheima.add

\* AddStudent

\* AddTeacher

\* com.itheima.delete

\* DeleteStudent

\* DeleteTeacher

\* ...

\* 方案2：按照模块分

\* com.itheima.student

\* AddStudent

\* DeleteStudent

\* ...

\* com.itheima.teacher

\* AddTeacher

\* DeleteTeacher

\* 包的定义格式：

\* package 包名;

\* 如果是多级包用.隔开即可

\* 注意事项：

\* A:package必须是程序的第一条可执行语句

\* B:package语句在一个java文件中只能有一个

\*/

public class PackageDemo {

}

## 导包的格式及使用

### 导包概述

不同包下的类之间的访问，我们发现，每次使用不同包下的类的时候，都需要加包的全路径。比较麻烦。这个时候，java就提供了导包的功能。

### 导包格式

* + import 包名;

### 案例代码十五

package com.itheima;

/\*

\* 导包：

\* import 包名;

\*/

import cn.itcast.Teacher;

public class Test {

public static void main(String[] args) {

Student s = new Student();

s.show();

/\*

//我要使用Teacher下的method()方法

//类不在同一个包下，使用的时候，要加类的全路径名称

cn.itcast.Teacher t = new cn.itcast.Teacher();

t.method();

t.method();

cn.itcast.Teacher t2 = new cn.itcast.Teacher();

t2.method();

//这样做太麻烦了，java就提供了一个导包的功能

\*

\*/

Teacher t = new Teacher();

t.method();

}

}

package com.itheima;

public class Student {

public void show() {

System.out.println("show");

}

}

package cn.itcast;

public class Teacher {

public void method() {

System.out.println("method");

}

}

## 四种权限修饰符的概述和访问特点

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | public | protected | 默认 | private |
| 同一类中 | √ | √ | √ | √ |
| 同一包子类,其他类 | √ | √ | √ |  |
| 不同包  子类 | √ | √ |  |  |
| 不同包  其他类 | √ |  |  |  |

归纳一下：在日常开发过程中，编写的类、方法、成员变量的访问

A:要想仅能在本类中访问使用private修饰

B:要想本包中的类都可以访问除了private修饰符,其它都可以

C:要想本包中的类与其他包中的子类可以访问使用protected修饰

D:要想所有包中的所有类都可以访问使用public修饰。

### 案例代码十六

package com.itheima;

public class Test {

public static void main(String[] args) {

Father f = new Father();

//f.show();

f.show2();

f.show3();

f.show4();

}

}

package com.itheima;

public class Son extends Father {

public static void main(String[] args) {

Father f = new Father();

//f.show();

f.show2();

f.show3();

f.show4();

Son s = new Son();

s.show2();

s.show3();

s.show4();

}

}

package com.itheima;

/\*

\* 权限修饰符：

\* 本类 同一个包下(子类和无关类) 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

\*/

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();

}

}