面向对象基础：

·类与对象

·封装

·构造方法

·方法

·重载

·对象数组

·this关键字

·static关键字

1. 类与对象

类：抽象的，概念上的，共性的，是对象的模板

对象：实际存在的，是类的每个个体，也称为类的实例，个性的，需要依靠类。

类是由什么组成的？

·属性

·方法

在面向对象中，是类还是对象更为重要？重点应该放在类的设计上。

结论：类是对象的模板，对象是类的具体体现，每个个体。

类的格式：

class 类名{

属性;

方法。

}

对象的创建：

类名 对象名 = new 类名（）；

|  |
| --- |
| **package** com.wanczy.OopDemo01;  **public** **class** Person {  String name;  **int** age;  **double** hight;    **public** **void** run(){  System.*out*.println(name + "is running");  }  **public** **void** eat(){  System.*out*.println(name + "is eating");  }  } |
| **package** com.wanczy.OopDemo01;  **public** **class** OopDemo01 {  **public** **static** **void** main(String[] args) {  Person p = **new** Person();  p.age = 10;  p.name = "张三";  p.hight = 184.5;    p.run();  p.eat();  }  } |

通过以上的例子可以发现，类的对象拥有类所定义的所有的属性和方法。

对象的引用传递：

|  |
| --- |
| **package** com.wanczy.OopDemo01;  **public** **class** OopDemo02 {  **public** **static** **void** main(String[] args) {  Person p = **new** Person();  p.age = 10;  p.name = "张三";  p.hight = 184.5;    Person p2 = p;  p2.name = "李四";  System.*out*.println(p.name);  }  } |

面向对象的特征：封装、继承、多态

二、封装：在类中使用private实现封装，用private修饰的属性不能被外部直接访问，private修饰的方法也不能直接被外部访问。

|  |
| --- |
| **package** com.wanczy.OopDemo02;  **public** **class** Person {  **private** String name;  **private** **int** age;  **private** **double** hight;    **private** **void** run(){  System.*out*.println(name + "is running");  }  **private** **void** eat(){  System.*out*.println(name + "is eating");  }  } |

现在发现使用private封装之后，对象无法访问类的属性和方法，那么对象怎么去访问类的封装的属性和方法呢？这个时候我们需要使用到getter和setter方法，这个方法是标准的取得属性和设置属性的方法，以后的开发都需要这样来写。

|  |
| --- |
| **package** com.wanczy.OopDemo02;  **public** **class** Person {  **private** String name;  **private** **int** age;  **private** **double** hight;  **public** **void** setName(String name){  **this**.name = name;  }  **public** **void** setAge(**int** age){  **this**.age = age;  }  **public** **void** setHight(**double** hight){  **this**.hight = hight;  }  **public** String getName(){  **return** **this**.name;  }  **public** **int** getAge(){  **return** **this**.age;  }  **public** **double** getHight(){  **return** **this**.hight;  }      **public** **void** run1(){  **this**.run();  }  **public** **void** eat1(){  **this**.eat();  }  **private** **void** run(){  System.*out*.println(name + "is running");  }  **private** **void** eat(){  System.*out*.println(name + "is eating");  }  } |
| **package** com.wanczy.OopDemo02;  **public** **class** OopDemo01 {  **public** **static** **void** main(String[] args) {  Person p = **new** Person();  p.setAge(10);  p.setHight(184.5);  p.setName("张三");  // p.age = 10;  // p.name = "张三";  // p.hight = 184.5;    p.run1();  p.eat1();  }  } |

封装和不封转我们没有发现有任何的好处，而是无缘无故的多了非常多的代码，那么我们怎么去看待封装的好处呢？下面我们举个例子来说明：比如人类的年龄一般在0-120岁之前，如果说不再这个范围就提示年龄输入不合法。

|  |
| --- |
| **package** com.wanczy.OopDemo02;  **public** **class** Person {  **private** String name;  **private** **int** age;  **private** **double** hight;      **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **if**(age >=120 || age<0){  System.*out*.println("年龄不合法");  }**else**{  **this**.age = age;  }  }  **public** **double** getHight() {  **return** hight;  }  **public** **void** setHight(**double** hight) {  **this**.hight = hight;  }  **public** **void** run1(){  **this**.run();  }  **public** **void** eat1(){  **this**.eat();  }  **private** **void** run(){  System.*out*.println(name + "is running");  }  **private** **void** eat(){  System.*out*.println(name + "is eating");  }  } |

匿名对象：只使用一次的对象称之为匿名对象，使用一次之后，这个对象无法再次使用。此对象没有名称。

|  |
| --- |
| new Person();//这个就是匿名对象 |

构造方法：用于构造对象的方法，实例化对象的时候可以给对象的属性赋值。

|  |
| --- |
| **public** Person(){  System.*out*.println("实例化Person的对象");  } |

·无任何返回值类型

·方法名必须和类名相同

|  |
| --- |
| **package** com.wanczy.OopDemo02;  **public** **class** Person {  **private** String name;  **private** **int** age;  **private** **double** hight;      **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **if**(age >=120 || age<0){  System.*out*.println("年龄不合法");  }**else**{  **this**.age = age;  }  }  **public** **double** getHight() {  **return** hight;  }  **public** **void** setHight(**double** hight) {  **this**.hight = hight;  }  **public** **void** run1(){  **this**.run();  }  **public** **void** eat1(){  **this**.eat();  }  **private** **void** run(){  System.*out*.println(name + "is running");  }  **private** **void** eat(){  System.*out*.println(name + "is eating");  }  **public** Person(){  System.*out*.println("实例化Person的对象");  }  } |

通过以上的程序运行结果发现，对象实例化的时候都会去调用类的构造方法，如果说一个类没有任何的构造方法， 则默认调用一个空的，无参数的什么都不做的构造方法。

|  |
| --- |
| **public** Person(String name,**int** age ,**double** hight){  System.*out*.println("这是一个有参数的构造方法");  **this**.name = name;  **this**.age = age;  **this**.hight = hight;  } |
| **package** com.wanczy.OopDemo03;  **public** **class** OopDemo01 {  **public** **static** **void** main(String[] args) {  Person p = **new** Person("张三",10,184.5);//调用有三个参数的构造方法  System.*out*.println(p.getName() +"\t"+ p.getAge() +"\t"+ p.getHight());  }  } |

以上的程序运行结果发现，对象实例化的时候可以通过类的构造方法进行属性的赋值。

方法：一段可以反复调用的特定的代码

方法的结构：public|private|default|protected static|final|synchronized|native void|返回类型 方法名称（参数）|throws Excpetion

public|private|default|protected:方法的访问权限

static|final|synchronized|native：方法的修饰，static表示静态方法，final表示最终方法，synchronized表示同步方法，native表示系统会自动调用的方法。

void|返回类型：返回值类型，void表示方法没有返回值

throws Excpetion：方法本身不处理异常，而是向上一级抛出异常。

|  |
| --- |
| **public** **void** eat(){  System.*out*.println(name+"要吃饭" );  }  **public** **void** eat(**int** x){  System.*out*.println(name+"要吃" +x +"碗饭");  } |

发现两个方法名称一致，参数不同，实现不同的功能，这就是一个方法重载的例子。

方法的重载：方法名称相同，根据参数的个数、类型、顺序不同而实现不同的功能，我们称之为方法的重载。在调用重载方法的时候会根据传入的参数不同而调用相应的方法。

构造方法可以重载吗？当然是可以的，一个类可以有多个构造方法，正常情况下，在开发的时候一个类中一般会有两个构造方法，一个 是空的无参数的，什么都不做的构造方法，另一个是可以为所有属性赋值的构造方法。在实例化对象的时候会根据传入参数的不同而去调用相应的构造方法。

|  |
| --- |
| **public** Person (){  System.*out*.println("这是一个无参数的什么都不做的构造方法");  }  **public** Person(String name,**int** age ,**double** hight){  System.*out*.println("这是一个有参数的构造方法");  **this**.name = name;  **this**.age = age;  **this**.hight = hight;  } |
| **package** com.wanczy.OopDemo03;  **public** **class** OopDemo01 {  **public** **static** **void** main(String[] args) {  Person p = **new** Person("张三",10,184.5);//调用有三个参数的构造方法  Person p1 = **new** Person();    }  } |

this关键字的使用：

1. this关键字可以调用当前类的属性
2. this关键字可以调用当前类的方法
3. this关键字是调用当前类的构造方法的

但是，调用当前类的构造方法的代码，必须写在代码块的第一行

4. this关键字他可以指代当前的对象

this调用当前类的属性以及方法

|  |
| --- |
| **package** com.wanczy.OopDemo03;  **public** **class** Person {  **private** String name;  **private** **int** age;  **private** **double** hight;      **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **if**(age >=120 || age<0){  System.*out*.println("年龄不合法");  }**else**{  **this**.age = age;  }  }  **public** **double** getHight() {  **return** hight;  }  **public** **void** setHight(**double** hight) {  **this**.hight = hight;  }    **public** **void** eat(){  System.*out*.println(name+"要吃饭" );  }  **public** **void** eat(**int** x){  System.*out*.println(name+"要吃" +x +"碗饭");  }  **public** Person(String name, **int** age, **double** hight) {  name = name;  **this**.age = age;  **this**.hight = hight;  }  **public** Person() {  }      } |

this调用当前类的构造方法

|  |
| --- |
| **package** com.wanczy.OopDemo03;  **public** **class** Person {  **private** String name;  **private** **int** age;  **private** **double** hight;      **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **if**(age >=120 || age<0){  System.*out*.println("年龄不合法");  }**else**{  **this**.age = age;  }  }  **public** **double** getHight() {  **return** hight;  }  **public** **void** setHight(**double** hight) {  **this**.hight = hight;  }    **public** **void** eat(){  System.*out*.println(name+"要吃饭" );  }  **public** **void** eat(**int** x){  System.*out*.println(name+"要吃" +x +"碗饭");  }  **public** Person(String name, **int** age, **double** hight) {  **this**();  **this**.setName(name);  **this**.age = age;  **this**.hight = hight;  }  **public** Person() {  System.*out*.println("生成了一个Person类");    }      } |

this指代当前类的对象

|  |
| --- |
| **package** com.wanczy.OopDemo03;  **public** **class** Person {  **private** String name;  **private** **int** age;  **private** **double** hight;      **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **if**(age >=120 || age<0){  System.*out*.println("年龄不合法");  }**else**{  **this**.age = age;  }  }  **public** **double** getHight() {  **return** hight;  }  **public** **void** setHight(**double** hight) {  **this**.hight = hight;  }    **public** **void** eat(){  System.*out*.println(name+"要吃饭" );  }  **public** **void** eat(**int** x){  System.*out*.println(name+"要吃" +x +"碗饭");  }  **public** Person(String name, **int** age, **double** hight) {  **this**.setName(name);  **this**.age = age;  **this**.hight = hight;  }  **public** Person() {  System.*out*.println("生成了一个Person类");    }  **public** **boolean** compare(Person p){  **boolean** bool=**false**;  **if**(**this**.name.equals(p.getName())&&**this**.age==p.getAge()){  System.*out*.println("恭喜跟小明一样了");  bool=**true**;  }**else**{  System.*out*.println("你还差的远呢");  }  **return** bool;  }            } |

this指代当前类的对象的时候，一般他是用于类的比较

static关键字：

·static变量的声明和使用

·static方法的声明和使用

·static代码块的使用

static变量的声明和使用：非static修饰的变量不可以被static方法调用，非static方法可以调用static变量。

|  |
| --- |
| **package** com.wanczy.OopDemo04;  **public** **class** OopDemo01 {  **int** i = 10;  **static** **int** *j* = 20;  **public** **static** **void** main(String[] args) {  System.*out*.println(i);//静态方法不能调用非静态变量  }  **public** **void** add(){  System.*out*.println(i);  System.*out*.println(*j*);//非静态方法可以调用静态变量  }  } |

static方法的声明和使用：非static方法不能被static方法调用，非static方法可以调用static方法。

|  |
| --- |
| **package** com.wanczy.OopDemo04;  **public** **class** OopDemo01 {  **int** i = 10;  **static** **int** *j* = 20;  **public** **static** **void** main(String[] args) {  System.*out*.println(i);//静态方法不能调用非静态变量  add();//静态方法不能访问非静态方法  }  **public** **void** add(){  System.*out*.println(i);  System.*out*.println(*j*);//非静态方法可以调用静态变量  *del*();//非静态方法可以调用静态方法  }    **public** **static** **void** del(){  }    } |

变量和方法的访问：

·普通变量（属性）和方法：通过对象访问，对象.属性；对象.方法

·静态变量（属性）和方法：可以直接通过类名.属性，类名.方法

|  |
| --- |
| **package** com.wanczy.OopDemo03;  **public** **class** OopDemo01 {  **public** **static** **void** main(String[] args) {  Person p = **new** Person("小明",12,134.5);//调用有三个参数的构造方法  p.compare(**new** Person("小明",12,130));  // System.out.println(p.getName() +"\t"+ p.getAge() +"\t"+ p.getHight());  // p.eat();  // p.eat(6);    // Person p1 = new Person();  Person.*eat*(10);//通过类名直接访问静态方法  Person.*weight* = 20;//通过类名直接访问静态属性    }  } |

、

栈：保存变量（对象）的名称

堆：保存的是对象的属性值或者说是变量的值

全局代码区：所有使用static修饰的方法

全局数据区：所有使用static修饰的变量（属性）

代码块：使用一对{}包围起来的代码就是代码块

·普通代码块：直接写在方法中的代码

·构造块（实例块）：直接写在一个类中，使用一对{}包围起来，在实例化对象的时候优于构造方法先执行，在实例化多个对象的时候构造块执行多次

·静态块：直接写在方法中，但是用static修饰，在实例化对象的时候优于构造块先执行，在实例化多个对象的时候，静态块只执行一次

·同步代码块：使用synchronized修饰，直接写在方法中（到多线程时再讲）

|  |
| --- |
| **package** com.wanczy.OopDemo04;  **public** **class** OopDemo02 {  **public** **void** add(){  System.*out*.println("普通代码块");  **synchronized**(**this**){  System.*out*.println("这里是同步快");  }  }  {  System.*out*.println("这里是构造块");  }  **static**{  System.*out*.println("这里是静态块");  }  **public** OopDemo02(){  System.*out*.println("这是构造函数");  }    } |

静态变量或者静态方法还有静态代码块的好处在于用在特定的业务逻辑中，比如说买票程序，可以实现票数的共享。

|  |
| --- |
| **package** com.wanczy.OopDemo04;  **public** **class** OopDemo02 {  **static** **int** *ticket*;  **public** **static** **void** saleTicket(){  --*ticket*;//每次售票票数减1  }  **public** **void** add(){  System.*out*.println("普通代码块");  **synchronized**(**this**){  System.*out*.println("这里是同步快");  }  }  {  System.*out*.println("这里是构造块");  }  **static**{  *ticket* = 100;//只能给总票数赋值一次  System.*out*.println("这里是静态块");  }  **public** OopDemo02(){  System.*out*.println("这是构造函数");  }    } |

对象数组：数组中存放的是对象，大家去完成一个练习，在数组中存放对象，进行增删改查。

|  |
| --- |
| **package** com.wanczy.OopDemo05;  **public** **class** Student {  **private** **int** age ;  **private** String name;  **private** String stu\_num;  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** String getStu\_num() {  **return** stu\_num;  }  **public** **void** setStu\_num(String stuNum) {  stu\_num = stuNum;  }  **public** Student(**int** age, String name, String stuNum) {  **super**();  **this**.age = age;  **this**.name = name;  stu\_num = stuNum;  }  **public** Student() {  **super**();  // **TODO** Auto-generated constructor stub  }    } |
| **package** com.wanczy.OopDemo05;  **public** **class** OopDemo01 {  Student s[] = **new** Student[10];  **public** **void** add(Student stu){  **for** (**int** i = 0; i < s.length; i++) {  **if**(**null** == s[i]){  s[i] = stu;  System.*out*.println("新增成功");  **return**;  }  }  System.*out*.println("新增失敗");  }  **public** **void** upd(Student stu\_old,Student stu\_new){  **int** i = **this**.find(stu\_old);  **if**(i!=-1){  s[i] = stu\_new;  System.*out*.println("修改成功");  **return**;  }  System.*out*.println("修改失败");  }  **public** **void** del(Student stu){  **int** i = **this**.find(stu);  **if**(i!=-1){  **if**(i == s.length-1){  s[s.length-1] = **null**;  }**else**{  **for**(**int** x = i;x<s.length-1;x++){  **if**(**null**!=s[x]){  s[x-1] = s[x];  }**else**{  s[x-1] = **null**;  }  }  }  System.*out*.println("删除成功");  **return**;  }  System.*out*.println("删除失败");  }  **public** **void** print(){  **for** (**int** i = 0; i < s.length; i++) {  **if**(**null** != s[i]){  System.*out*.println(s[i].getStu\_num()+"\t" + s[i].getName()+"\t" + s[i].getAge());    }  }  }  **public** **int** find(Student stu){  **for** (**int** i = 0; i < s.length; i++) {  **if**(**null** != s[i] && s[i].getName().equals(stu.getName())  && s[i].getStu\_num().equals(stu.getStu\_num())&& s[i].getAge()==(stu.getAge())){  **return** i;  }  }  **return** -1;  }    } |

·以对象数组作为方法的参数

·以对象数组作为方法的返回值

·在一个方法中，参数和返回值同时都是对象数组

main方法：main是java程序运行的统一入口

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

public :访问权限最高

static:表示此方法是静态的

void：表示此方法没有返回值

main:方法名称

String args[]：此方法运行时传入的参数

|  |
| --- |
| public class MainDemo  {  public static void main(String args[]){  for(int i=0;i<args.length;i++){  System.out.println(args[i]);  }    }  } |

main方法参数的输入：java 类名 参数1 参数2 参数3

讲方法的时候漏了的内容：

形式参数：形参，是方法定义的时候的参数

实际参数：实参，调用方法的时候传入的参数

|  |
| --- |
| **package** com.wanczy.OopDemo05;  **public** **class** OopDemo02 {  **public** **static** **void** main(String[] args) {  **int** i = 10;  **int** j = 20;  System.out.println(add(i,j));//这边的i和j就是实际参数  }  **public** **static** **int** add(**int** x,**int** y){//这边的x和y就是形式参数  **return** x+y;  }  } |