# 1. 面向对象2

## 1.1 代码块

代码块用于限定变量的范围，在Java中，还有两个特殊的代码块：

（1）构造代码块：在类的成员位置，用{ }括起来的代码。每次执行构造方法前，都会先执行构造代码块中的代码。

可以把多个构造方法中的共同代码写在构造代码块中，用于对创建的每个对象进行统一初始化，而构造方法只是对用此构造创建的对象初始化。

（2）静态代码块：在类的成员位置，用{ }括起来的代码，并且此段代码用static修饰。

静态代码块是对类进行初始化，**只是在加载类时执行一次。**

代码的执行顺序：静态代码块—构造代码块—构造方法。强调：静态代码块只执行一次，构造代码块每次调用构造方法都执行。

（1）如果想使用静态代码块中的变量，必须将变量提升为类的静态成员，然后外部可以调用。本类的静态方法可访问此变量，实例方法不可以。例子：

|  |
| --- |
| **public class** Student {  **public static** String *str*;  **static** {  *// 静态代码块中初始化  str* = **"hello world"**;  }   *// 静态方法中可访问* **public static void** hello() {  System.***out***.println(*str*);  } } |

使用：

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  System.***out***.println(Student.*str*);  Student.*hello*();  } } |

（2）实例成员变量、构造代码块和构造方法操作同一个变量时的先后顺序：

Java编译后，会把成员变量的定义提升到类的最前面，但是对象成员的初始化都是在构造方法中完成的，即成员变量的赋值和构造代码块都会在构造方法中先执行，最后执行构造方法的内容。至于成员变量的赋值和构造代码块谁先执行，就是按照原本代码的顺序来的。例子：

|  |
| --- |
| **public class** Student {  **int age** = 100;  {  **age** = 200;  }   **public** Student() {  **age** = 300;  }   **public void** showAge() {  System.***out***.println(**age**); *// 实例可访问实例变量* } }  如果将构造中的age = 300注释掉，那么输出的就是200，如果再把int age = 100放在构造代码块下面，那么下面输出的就是100了。注意不要重复定义变量，都是在类中定义类变量的，否则外部访问不到。如果在代码块中重新定义了变量，值就不一样了，能访问的只有类中定义的变量。 |

使用：

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  Student stu = **new** Student();  System.***out***.println(stu.**age**);  stu.showAge();  } } |

## 1.2 类之间的关系

（1）整体与部分的关系。现在有两个类，一个是地址类，一个是人类。因为人有住址，所以人类中应该有个“住址”属性，该属性的类型就是地址类。这种关系满足“has a”的关系。

例子：

地址类：

|  |
| --- |
| **public class** Address {  **private** String **province**;  **private** String **city**;  **private** String **street**;   **public** String getProvince() {  **return province**;  }   **public void** setProvince(String province) {  **this**.**province** = province;  }   **public** String getCity() {  **return city**;  }   **public void** setCity(String city) {  **this**.**city** = city;  }   **public** String getStreet() {  **return street**;  }   **public void** setStreet(String street) {  **this**.**street** = street;  } } |

人类：

|  |
| --- |
| **public class** Person {  **private** String **name**;  **private int age**;  **private** Address **addr**; *// 地址信息* **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** Address getAddr() {  **return addr**;  }   **public void** setAddr(Address addr) {  **this**.**addr** = addr;  } } |

（2）继承关系。现在有两个类，一个是动物类，一个是猫类，那么猫类应该继承自动物类，因为猫满足“is a”动物的关系。动物类中是共性，通过继承，有以下好处：

1. 提高代码的复用性，不用写多个类中相同的代码；

2. 提高代码的维护性，关系清晰；

3. 让类与类产生了关系，这是多态的前提。

下面就讲继承。

## 1.2 继承

### 1.2.2 Java中的继承

Java中类的继承只支持单继承，但支持多层继承。即一个类只能继承一个类，允许这个类再被继承，形成继承体系。

继承中，子类只能继承父类非私有的成员；不能继承父类的私有成员和构造方法，但是可以通过super关键字来访问父类的构造方法。

不要为了部分功能去继承。要满足“子类对象 is a 父类对象”的关系。

继承的例子：

（1）Animal类

|  |
| --- |
| **public class** Animal {  *// 有名称属性* **private** String **name**;   *// 动物有吃的行为* **public void** eat() {  System.***out***.println(**this**.**name** + **" is eating"**);  }   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  } } |

（2）Cat类

|  |
| --- |
| **public class** Cat **extends** Animal {  *// name属性和eat方法在动物中有了，继承过来的不用再重复   // 这里写猫特有的叫的行为* **public void** miao() {  System.***out***.println(**this**.getName() + **"喵喵叫"**);  } } |

（3）Demo类中使用

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  Cat cat = **new** Cat();  *// cat中可以调用父类和本类的非私有成员* cat.setName(**"小猫"**);  cat.miao();  cat.eat();  } } |

### 1.2.3 super关键字

super关键字指向父类对象。这个父类对象是从哪里来的。实际上继承中，创建子类对象之前先会创建父类对象，默认子类构造方法的第一行就是super()，就是调用父类的无参构造创建父类对象。

因为默认调用父类的无参构造，所以如果手动写了父类的构造方法（这样父类就没有了默认的无参构造），那么需要在子类的构造方法第一行手动用super关键字调用父类对应的构造方法。

总结this和super的用法：

（1）调用成员变量

this.成员变量 调用本类的成员变量

super.成员变量 调用父类的成员变量

（2）调用构造方法

this(…) 调用本类的构造方法

super(…) 调用父类的构造方法

（3）调用成员方法

this.成员方法() 调用本类的成员方法

super.成员方法() 调用父类的成员方法

### 1.2.4 方法重写(override)

方法重写就是子类重写父类的方法，这样当子类对象调用该方法时，会表现出子类的行为，而父类对象调用该方法时，表现的还是父类的行为。

子类方法与父类方法同名，自动重写。无法重写父类的私有方法

子类重写父类的方法时，访问权限不能更低，最好一致。而且返回值类型必须是父类函数的返回值类型或该返回值类型的子类。不能返回比父类更大的数据类型。

## 1.3 instanceof关键字