【济南中心】JAVA 编程阶梯：基础篇之第八章

- **代码块**

代码块概述：

* 在 Java 中，使用{}括起来的代码被称为代码块。

代码块分类：

* 根据其位置和声明的不同，可以分为局部代码块，构造代码块，静态代码块，同步代码块(多线程讲解)。

常见代码块：

* a:局部代码块

在方法中出现；限定变量生命周期，及早释放，提高内存利用率

* b:构造代码块 (初始化块)

在类中方法外出现；多个构造方法方法中相同的代码存放到一起，每次调用构造都执行，并且在构造方法前执行

* c:静态代码块

在类中方法外出现，并加上 static 修饰；用于给类进行初始化，在加载的时候就执行，并且只执行一次。

一般用于加载驱动

```
class Text {
		static {
				System.out.println("静态代码块");
		}

		{
				System.out.println("构造代码块");
		}
```

```
                public Text() {
                        System.out.println("构造方法");
                }
}

public class Demo {
                static {
                        System.out.println("Demo 静态代码块");
                }

                public static void main(String[] args) {
                        System.out.println("我是 main 方法");

                        Text s1 = new Text();
                        Text s2 = new Text();
                }
}
```

运行结果：

```
Demo静态代码块
我是main方法
静态代码块
构造代码块
构造方法
构造代码块
构造方法
```

可以看出，执行顺序 静态代码块>构造代码块>构造函数

- **面向对象之继承**

继承(extends)概念：

让类与类之间产生关系,子父类关系

继承的好处：

* a:提高了代码的复用性

* b:提高了代码的维护性

* c:让类与类之间产生了关系，是多态的前提

继承的弊端：

* 类的耦合性增强了。

* 开发的原则：高内聚，低耦合。

* 耦合：类与类的关系

* 内聚：就是自己完成某件事情的能力

继承的特点：

* a:Java 只支持单继承，不支持多继承。(一个儿子只能有一个爹)

* 有些语言是支持多继承，格式：extends 类 1,类 2,...

* b:Java 支持多层继承(继承体系) * 如果想用这个体系的所有功能用最底层的类创建对象

* 如果想看这个体系的共性功能,看最顶层的类

继承的注意事项：

* a:子类只能继承父类所有非私有的成员(成员方法和成员变量)

* b:子类不能继承父类的构造方法，但是可以通过 super(马上讲)关键字去访问父类构造方法。

* c:不要为了部分功能而去继承

什么时候使用继承：

* 继承其实体现的是一种关系："is a"。如果有两个类 A,B。只有他们符合 A 是 B 的一种，或者 B 是 A 的一种，就可以考虑使用继承。

下面举个栗子让大家了解一下继承的使用：

```
class Animal {
        void call() {
                System.out.println("叫...");
        }
```

```
                void run() {
                                System.out.println("跑...");
                }
}

class Felidae extends Animal {
                void shelter() {
                                System.out.println("隐藏....");
                }

}

class Canidae extends Animal {
                void hunting() {
                                System.out.println("捕猎....");
                }
}

class Dog extends Canidae {
                void eat() {
                                System.out.println("吃骨头....");
                }
}

class Cat extends Felidae {
                void eat() {
                                System.out.println("吃鱼....");
                }
}

public class Demo {

                public static void main(String[] args) {
                                Dog dog = new Dog();
                                dog.eat();
                                dog.hunting();
                                dog.run();
                                Cat cat = new Cat();
                                cat.eat();
                                cat.shelter();
                                cat.call();
                }
}
```

运行结果：

```
吃骨头....
捕猎....
跑...
吃鱼....
隐藏....
叫...
```

- **this 和 super 的区别和应用：**

this 和 super 的用法很相似。

this 代表本类对象的引用。

uper 代表父类的内存空间的标识。

this 和 super 的使用区别：

* a:调用成员变量

*this.成员变量 调用本类的成员变量,也可以调用父类的成员变量

*super.成员变量 调用父类的成员变量

* b: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

* c:调用成员方法

*this.成员方法 调用本类的成员方法,也可以调用父类的方法

*super.成员方法 调用父类的成员方法

```
class Fu {
		String name = "老王";
		int age = 40;

		void chageData() {
			name = "老李";
		}
```

```
}

class Zi extends Fu {
		String name = "小王";

		void chageData() {
			age = 20;
		}

		void show1() {
			System.out.println(this.name + "..." +
this.age);// 调用子类的，子类没有的去父类找
			System.out.println(super.name + "..." +
super.age);// 调用父类的
			chageData();// 调用子类的方法
			super.chageData();// 调用父类的方法
			System.out.println(super.name + "..." +
super.age);// 调用父类的

		}
}

public class Demo {
		public static void main(String[] args) {
			Zi zi = new Zi();
			zi.show1();
		}
}
```

运行结果：

```
小王...40
老王...40
老李...20
```

子类中所有的构造方法默认都会访问父类中空参数的构造方法

每一个构造方法的第一条语句默认都是：super() Object 类最顶层的父类。

```
class Fu {
		String name;
		int age;

		public Fu() {

		}

		public Fu(String name, int age) {
			this.name = name;
			this.age = age;
		}

}

class Zi extends Fu {

		public Zi() {
			this("小李");// 调用自己的一个参数的构造方法
		}

		public Zi(String name) {
			super(name, 30);//调用父类有两个参数的构造方法
		}

		void show1() {
			System.out.println(this.name + "..." +
this.age);// 调用子类的，子类没有的去父类找

		}
}

public class Demo {
		public static void main(String[] args) {
			Zi zi = new Zi();
			zi.show1();
		}
}
```

运行结果：

```
小李...30
```

继承中构造方法的注意事项：

如果父类没有无参构造方法,子类需要手动的调用 super 或者 this 调用父类或者自己的有参构造方法

super(...)或者 this(....)必须出现在构造方法的第一条语句上

- **方法重写概述及其应用**

重写:子父类出现了一模一样的方法

方法重写的应用：

* 当子类需要父类的功能，而功能主体子类有自己特有内容时，可以重写父类中的方法。这样，即沿袭了父类的功能，又定义了子类特有的内容。 子类对象调用方法的时候先找子类本身，再找父类。

方法重写注意事项

* a:父类中私有方法不能被重写
    * 因为父类私有方法子类根本就无法继承
* b:子类重写父类方法时，访问权限不能更低
    * 最好就一致
* c:父类静态方法，子类也必须通过静态方法进行重写
    * 其实这个算不上方法重写，但是现象确实如此，至于为什么算不上方法重写，多态中我会讲解(静态只能覆盖静态)
    * 子类重写父类方法的时候，声明一模一样，与返回值类型有关,返回值是

一致(或者是子父类)的

重写和重载的区别？

重载：在同一类中。方法名相同，参数列表不同。重载可以改变返回类型。

重写：在不同类中(子父类中)。方法声明相同(返回类型，方法名，参数列表均相同)。

```
class People {
		void eat() {
			System.out.println("吃饭.....");
		}
}

class Man extends People {
		// 重写
		public void eat() {
			System.out.println("喝酒....");
		}

		// 重载
		void eat(String str) {
			System.out.println("吃" + str + "....");
		}
}

public class Demo {

		public static void main(String[] args) {
			Man man = new Man();
			man.eat();
			man.eat("牛肉");
		}
}
```

运行结果：

```
喝酒....
吃牛肉....
```

- **final 关键字**

final 修饰特点

1，final 修饰的类是一个最终类，不能在派生子类。

2，final 修饰的方法是最终方法，不可以给重写。

3，final 修饰的变量是一个常量，只能被赋值一次。 被 final 修饰的常量名所有的字母都是大写的。如果由多个单词组成单词间通过 _ 连接。

4，可以通过构造代码块或者构造函数赋值一次。

* 基本类型，是值不能被改变

* 引用类型，是地址值不能被改变,对象中的属性可以改变