Static修饰符

武永亮

讲授思路

- 静态方法
- 静态变量
- 静态类

静态方法

- public static void main(String[] args){ }
- 语法定义
  - 权限修饰符 static 返回值类型 方法名（形式参数列表）{
  - …方法体
  - }
- 使用方法
  - 类名. 方法名(实际参数列表)；

静态方法

- 使用规则
  - 在静态方法中，仅能调用其他的static 方法。
  - 在静态方法中，只能访问static数据。
  - 在静态方法中，不能以任何方式引用 this 或 super
- 意义
  - 静态方法常常为应用程序中的其它类提供一些实用工具所用
- Java的类库中大量的静态方法（如：Math）

静态方法

```
class Simple {
      static void go() {
            System.out.println("Welcome");
       }
}
public class SimpleTest {
       public static void main(String[] args) {
            Simple.go();
       }
}
```

讲授思路

- 静态方法
- 静态变量
- 静态类

静态变量

- 声明为static的变量实质上就是全局变量
- 当声明一个对象时，并不产生static变量的拷贝
- 该类所有的实例变量共享同一个static变量
- 在类装载时，只分配一块存储空间，所有此类的对象都可以操控此块存储空间（final修饰的除外）

静态变量

```
class Value {
    static int c = 0;
    static void inc() {
        c++;
    }
}
```

```
class Count2 {
    public static void prt(String s) {
        System.out.print(s);
    }
}
```

```
public class Test{
    public static void main(String[] args) {
        Value v1, v2;
        v1 = new Value();
        v2 = new Value();
        Count count = new Count();
        Count.prt("v1.c=" + v1.c + "  v2.c=" + v2.c);
        v1.inc();
        count.prt(" v1.c=" + v1.c + "  Value.c=" + Value.c);
    }
}
```

```
v1.c=0  v2.c=0 v1.c=1  Value.c=1
```

静态变量

- 静态变量值一旦改变，所有类的对象均共享改变

static代码块

- 初始化你的static变量
- 仅在该类被加载时执行一次

```
class Value3 {
    static int c = 0;
    Value3() {
        c = 15;
    }
    Value3(int i) {
        c = i;
    }
    static void inc() {
        c++;
    }
}
```

static代码块

- 初始化你的static变量
- 仅在该类被加载时执行一次

```
class Count2 {
    static Value3 v1, v2;
    Value3 v = new Value3(10);
    static {// 此即为static块
        prt("v1.c=" + v1.c + "  v2.c=" + v2.c);
        v1 = new Value3(27);
        prt("v1.c=" + v1.c + "  v2.c=" + v2.c);
        v2 = new Value3(15);
        prt("v1.c=" + v1.c + "  v2.c=" + v2.c);
    }
    public static void prt(String s) {
        System.out.println(s);
    }
}
```

static代码块

- 初始化你的static变量
- 仅在该类被加载时执行一次

```
public class Test2 {
    public static void main(String[] args) {
        Count2 ct = new Count2();
        Count2.prt("ct.v.c=" + ct.v.c);
        Count2.prt("Count2.v1.c=" + Count2.v1.c
                + "  Count2.v2.c=" + Count2.v2.c);
        Count2.v1.inc();
        Count2.prt("Count2.v1.c=" + Count2.v1.c
                + "  Count2.v2.c=" + Count2.v2.c);
        Count2.prt("ct.v.c=" + ct.v.c);
    }
}
```

static代码块

- 程序的输出结果

```
v1.c=0  v2.c=0
v1.c=27  v2.c=27
v1.c=15  v2.c=15
ct.v.c=10
Count2.v1.c=10  Count2.v2.c=10
Count2.v1.c=11  Count2.v2.c=11
ct.v.c=11
```

静态变量

- static{后面跟着一段代码，这是用来进行显式的静态变量初始化，这段代码只会初始化一次，且在类被第一次装载时
- 涉及到继承的时候，会先初始化父类的static变量，然后是子类的
- 子类实例化对象时的执行顺序
  - 父类的静态代码块
  - 子类的静态代码块
  - 父类的构造方法
  - 子类的构造方法

静态变量

```
class FatherClass{
    private static int num;
    static{
        num = 10;
        System.out.println("FatherClass static");
    }
    public FatherClass() {
        this.num = 20;
        System.out.println("FatherClass()");
    }
}
```

静态变量

```
class ChildClass extends FatherClass{
    private static int childNum;
    static{
        childNum = 30;
        System.out.println("ChildClass static");
    }
    public ChildClass() {
        this.childNum = 50;
        System.out.println("ChildClass()");
    }
}
```

静态变量

```
public class StaticExtendsCode {
    public static void main(String[] args) {
        new ChildClass();
    }
}
```

```
FatherClass static
ChildClass static
FatherClass()
ChildClass()
```

讲授思路

- 静态方法
- 静态变量
- 静态类

静态类

- 通常一个普通类不允许声明为静态的，只有一个内部类才可以。
- 声明为静态的内部类可以直接作为一个普通类来使用，而不需实例化一个外部类

补充

- static和final一起使用
  - static final用来修饰成员变量和成员方法，可简单理解为“全局常量”
  - 对于变量，表示一旦赋值就不可修改，并且通过类名可以访问
  - 对于方法，表示不可覆盖，并且可以通过类名直接访问

总结

- static表示“全局”或者“静态”的意思，用来修饰成员变量和成员方法，也可以形成静态static代码块，但是Java语言中没有全局变量的概念
- 被static修饰的成员变量和成员方法独立于该类的任何对象，它不依赖类特定的实例，被类的所有实例共享
- static 变量前可以有private修饰，表示这个变量可以在类的静态代码块中，或者类的其他静态成员方法中使用（当然也可以在非静态成员方法中使用），但是不能在其他类中通过类名来直接引用
- 用static修饰的代码块表示静态代码块，当Java虚拟机（JVM）加载类时，就会执行该代码块

总结

- Java中的变量
  - 静态变量（被static修饰的变量，叫静态变量或类变量）
  - 实例变量（没有被static修饰的变量，叫实例变量）
- 对于静态变量在内存中只有一个，JVM只为静态分配一次内存，在加载类的过程中完成静态变量的内存分配，可用类名直接访问（方便），当然也可以通过对象来访问（不推荐）。
- 对于实例变量，没创建一个实例，就会为实例变量分配一次内存，实例变量可以在内存中有多个拷贝，互不影响（灵活）。

总结

- static方法
  - 静态方法可以直接通过类名（或对象，不推荐）调用
  - 静态方法中不能用this和super关键字
  - 不能直接访问所属类的实例变量和实例方法 (就是不带static的成员变量和成员成员方法)
  - static方法独立于任何实例，因此static方法必须被实现，而不能是抽象的abstract。

总结

- 多态的概念
- 多态的实现

Thank You