静态static

入门

```java

public class Person { int age = 0; String name; // 共享 static String country = "中国";

```
public void show() {
    System.out.println("show");
}

static public void a() {
    System.out.println("a.");
}
```

}

public class DemoStatic {

```
public static void main(String[] args) {
    // 1.static 静态
    // 1.1 由来 :只要点调用，不想创建对象。
    // Person p=new Person(); p.a p.getName();
    // Person.a Person.getName();
    // 1.2概念：不需要创建对象可以直接使用的成员
    // 1.3使用：设置变量被所有的对象共享，如果被修改能否影响所有的对象。
    System.out.println(Person.country);
    Person p1 = new Person();
    p1.age = 11;
    p1.name = "凤姐";
    Person p2 = new Person();
    p2.age = 12;
    p2.name = "富帅";
    p1.country = "美国";
    System.out.println(p1.country);
    System.out.println(p2.country);
    // 1.4注意事项:共享
    // 类成员:有static修饰
    // 对象的成员:没有static
    // 类成优先于对象的成员初始化，类成员跟对象没有关系。
    //  点调用.  类.成员变量 类.成员方法
    System.out.println(p1.country);
    p1.a();
    System.out.println(Person.country);
    Person.a();
}
```

}

```

## 练习

```java public class Demo {

```
int number1 = 11;
static int number2 = 12;

public void showNumber1() {
    System.out.println("showNumber1");
}
```
```

```
public static void showNumber2() {
    System.out.println("showNumber2");
}

public static void main(String[] args) {
    // 对象成员
    Demo demo = new Demo();
    System.out.println(demo.number1);
    demo.showNumber1();
    // 类成员
    System.out.println(Demo.number2);
    Demo.showNumber2();
    // 在类的内存调用。直接写类成员
    System.out.println(number2);
    showNumber2();
}
```

}

```

## 静态方法不能使用成员

```java public class DemoStaticMember {

```
int number1 = 11;
static int number2 = 12;

public static void showNumber2() {
    System.out.println("number2 " + number2);
}

// DemoStaticMember obj=new DemoStaticMember();
public void show() {
    System.out.println("number2 " + number2);
}

// 错误:
// 静态加载 完毕对象还没有创建，所以不能使用。
// public static void showNumber() {
// System.out.println("number1 " + number1);
// }
// 静态的优先加载 。
```

}

```

## 静态代码块

```java public class DemoStatic {

```
public static void main(String[] args) {
    //
    DemoStatic s = new DemoStatic();
    DemoStatic s1 = new DemoStatic();
    s1 = new DemoStatic();
    s1 = new DemoStatic();
    s1 = new DemoStatic();
    s1 = new DemoStatic();
    s1 = new DemoStatic();
    //企业应用:
}
```

```
// static 优先加载，整个过程仅加载一次
static {
    System.out.println("我是静态代码块");
}
```

}

```

# Final

```java public class DemoFinal {

```
public static void main(String[] args) {
    // final
    // 由来:不想类有子类。
    // :父类方法不想给子类覆盖。
    // 面试:常用String 与基本数据类型的包装类 int Integer
    // 概念:一经赋值不可修改。
    // 使用:修饰类 public final class Fu
    // 修饰方法: public final void show()
    // 修饰变量
    // 成员变量 final+构造函数
    // 企业:身份证系统 银行 = 姓名 与身份证号不可改变
    // 局部变量
    // final float PI = 3.141592F;
    final float PI;
    PI = 3.141592F;
    final String country = "中国";
    // 数组加final修饰 内存地址与内容 final int[] numbers 地址一经初始化就不可修改
    final int[] numbers = new int[] { 1, 2, 3, 4, 5 };
    numbers[0] = 14;// 可行
    // numbers=new int[]{1,2,3};不行。
    // 引用类型: 对象 有地址与内容
    final Person p = new Person();// 地址一经赋值不可修改
    p.age = 12;// 内容
    p.name = "中国";
    // p=new Person(); 不行。
    // 注意事项:

}
```

}

public class Fu { public void show() { System.out.println("Fu.show"); } }

public class Zi extends Fu{

```
@Override
public void show() {
    System.out.println("覆盖重写父类方法show");
}
```

}

```

# final修饰成员变量

```java

public class Student { // final int age=1; // final String name="中国"; final int id; final String name;

```
public Student(int id, String name) {
```

```
    this.id = id;
    this.name = name;
}
```

}

```

# 包

```java public class DemoPackage {

```
public static void main(String[] args) {
    // 包
    // 由来：类多了，可能出现混乱冲突，所以必须进行管理类。 分包管理。
    // 概念：管理类的单元 本质分好文件夹 放置类
    // 定义：package cn.itcast.demo04
    // 1.使用点隔开
    // 2.命名一般在项目中 是用公司的网址倒写 。com.itcast cn.itcast
    // 在类的第一行 使用 package cn.itcast.demo05;
    // 使用 import关键字:导入当前类的包名 简化代码
    cn.itcast.demo01.Person person = new cn.itcast.demo01.Person();
    Person p = new Person();
    // 注意事项 java.lang类不需要导包，默认当前类已经导入java.lang包
    String name = "中国";

}
```

}

```

# 权限

```java package cn.itcast.demo6; public class DemoProtected {

```
public static void main(String[] args) {
    // 访问权限
    // java public > protected > 默认 default > private

    // new Person().num1; 不可访问
    System.out.println(new Person().num4);
    //System.out.println(new Person().num3);
}
```

}

public class Person { private int num1 = 1; //default 本包内就可以访问 int num2 = 2; //protected 子类可以使用。 protected int num3 = 3; //public 所有地方都可访问 public int num4 = 4;

```
//private 不超本类范围都可以访问
public void show() {
    System.out.println(num1);
}
```

}

public class Student extends Person {

```
//子类方部可以访问父类的protected资源
public void show2() {
    System.out.println(num3);
}
```

}

另外一个包 package cn.itcast.demo7; public class DemoDefault {

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    System.out.println(new Person().num2);
}
```

}

```

# 内部类

## 成员内部类

```java public class BodyInner { //包含 class Heart {

```
}
```

}

public class DemoInnerClass {

```
public static void main(String[] args) {
    // 内部类
    // 由来：表示包含关系
    // 概念:类中类
    // 定义
    // 一。成员内部类 -成员       再敲一个正常的类
    BodyInner.Heart heart = new BodyInner().new Heart();
    // 外部类.内部类 obj=new 外部类().new 内部类();

    // 二。匿名内部类 -方法内部

    // 使用

}
```

}

```

## 访问外部类变量

```java

public class OuterClass { int a = 1; class InnerClass { int a = 2;

```
    public void show() {
        int a = 3;// 就近原则
        System.out.println(a);
        System.out.println(this.a);// this 区别局部变量与成员变量
        //外部类.this.成员
        System.out.println(OuterClass.this.a);
    }
}
```

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

```
    //调用内部类成员方法 要求先创建内部类对象
    OuterClass.InnerClass obj=new OuterClass().new InnerClass();
    obj.show();
}
```

}

```

# 练习

```java public class Body { public boolean isLive = true;

```
class Heart {
    public void jump() {
        // if(Body.this.isLive)
        // {
        // System.out.println("心跳.....");
        // }

        if (isLive) {
            System.out.println("心跳.....");
        }
    }
}
```

} public class ExInnerClass {

```
public static void main(String[] args) {
    Body.Heart heart=new Body().new Heart();
    heart.jump();
}
```

}

```

# 匿名内部类

### 情况一.接口

```java public interface MyInterface { public abstract void method();

} MyInterface obj = new MyInterface() { @Override public void method() { System.out.println("No Name impl");

```
        }
    };
    obj.method();
```

```

### 情况二.抽象类

```java

public abstract class Person { public abstract void walk();

} // 使用匿名内部类 一步到位。 Person p = new Person() { @Override public void walk() { System.out.println("匿名内部类。。walk实现"); } };// (){} 没有class p.walk(); ```

### 建议编写方式

```java public interface MyInterface2 { public abstract void add(); public abstract void delete(); } //不建议使用匿名对象 MyInterface2 obj2=new
```

MyInterface2() { @Override public void add() { System.out.println("添加");

```
        }

        @Override
        public void delete() {
            System.out.println("删除");

        }

    };

    obj2.add();
    obj2.delete();
```

```
```