static关键字

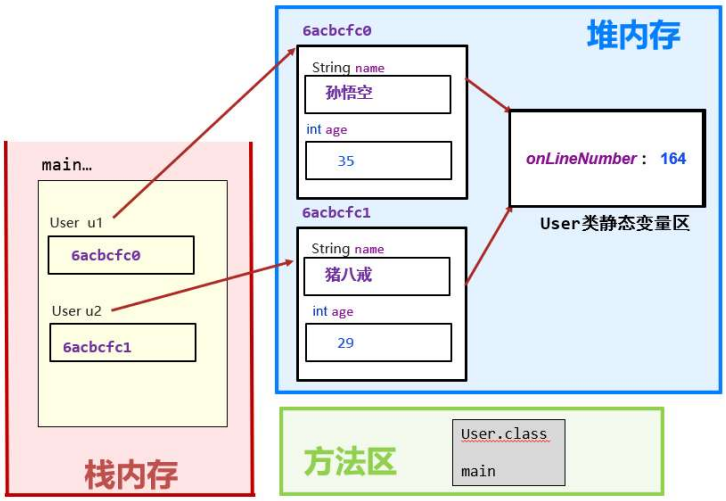
static概述

static是静态的意思，可以修饰成员变量和成员方法，内部类和代码块；
static修饰成员变量表示该成员变量只在内存中只存储一份，可以被共享访问、修改。

成员变量可以分为两种

1. 实例成员变量：类内方法外，无static修饰，属于每一个对象
2. 静态成员变量：类内方法外，有static修饰，属于类（类的每一个对象）需要被共享的信息，可以被共享访问

```
public class User {
    // 静态成员变量：有static修饰，属于类，加载一次，可以被共享访问
    public static int onLineNumber = 161;
    // 实例成员变量：无static修饰，属于每个对象
    private String name;
    private int age;

    public static void main(String[] args) {
        // 1、类名.静态成员变量
        System.out.println(User.onLineNumber);
        System.out.println(onLineNumber);
        User.onLineNumber++;

        // System.out.println(name); // 报错！

        // 2、创建用户对象：对象.实例成员变量
        User u1 = new User();
        u1.name = "孙悟空";
        u1.age = 35;
        System.out.println(u1.name);
        System.out.println(u1.age);
        // 对象名称.静态成员变量（不推荐）
        u1.onLineNumber++;

        User u2 = new User();
        u2.name = "猪八戒";
        u2.age = 29;
        System.out.println(u2.name);
        System.out.println(u2.age);
        // 对象名称.静态成员变量（不推荐）
        u2.onLineNumber++;

        System.out.println(User.onLineNumber); // 164
    }
}
```



1. 加载User.class文件进入方法区内存
2. 将静态成员变量加载进堆内存
3. 将方法区中的main方法加载进栈内存中运行
4. 通过类名操作静态成员变量
5. 在栈内存中开辟一块存储User类型的空间，再到堆内存中开辟一块存放User实例的空间，并为成员变量赋初始值；然后把实例的地址交给栈中的user变量
6. 通过实例的地址操作实例的成员变量
7. 通过实例的地址找到实例再操作静态成员变量

- 两种成员方法

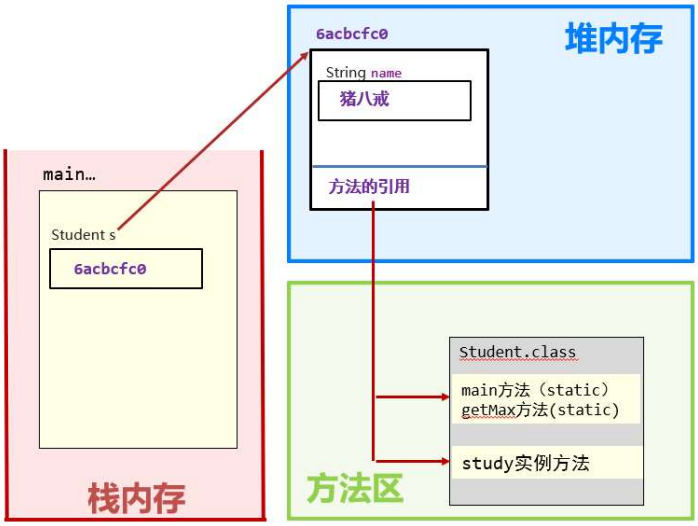
实例成员方法：(无static修饰，属于对象），只能用对象触发访问 表示对象自己的行为的，且方法中需要访问实例成员的，则该方法必须申明成实例方法或者实例一个对象，通过对象访问；
静态成员方法：（有static修饰，属于类），建议用类名访问【子类类名也可以】，也可以用对象访问。如果该方法是以执行一个共用功能为目的，或者想要方便访问则可以申明成静态方法；

```
package static_demo;

/**
 * ClassName: StaticDemo <br/>
 * Description: <br/>
```

```
 * date: 2022/3/6 17:19<br/>
 */
public class StaticDemo {
    private String ip;
    private String time;
    //静态成员变量属于类,可以被所有该类的实例访问
    private static int onlineNumber;

    /**
     * 实例成员方法可以访问本类中全部的成员变量和成员方法
     */
    public void say() {
        ip = "123";
        time = "123";
        onlineNumber = 123;
        test1();
        StaticDemo.test2();
        System.out.println("实例成员方法");
    }

    /**
     * 静态成员方法只能访问静态成员变量和静态成员方法
     */
    public static void funcSay() {
        //ip = "1 你你 23";   //报错
        //time = "123";//报错
        onlineNumber = 123;
        //test1();//报错
        StaticDemo.test2();
        System.out.println("静态成员方法");
    }


    public void test1() {
    }

    public static void test2() {
    }

    public static void main(String[] args) {
        final StaticDemo staticDemo = new StaticDemo();
        System.out.println(StaticDemo.onlineNumber);//访问静态成员变量
        System.out.println(onlineNumber);//访问静态成员变量（在本类中可以这么写）
        StaticDemo.funcSay();//访问静态成员方法
        funcSay();//访问静态方法（在本类中可以这么写）
        staticDemo.onlineNumber = 222;//可以通过实例访问静态成员变量（不推荐）
        staticDemo.funcSay();//可以通过实例访问静态成员方法（不推荐）
    }
}
```

- static成员方法内存原理

```java
public class Student {
    private String name;
    // 1. 实例方法: 无static修饰的，属于对象的
    public void study(){
        System.out.println(name + "在好好学习~~~");
    }
    // 2. 静态方法：有static修饰，属于类和对象共享的
    public static int getMax(int a , int b){
        return a > b ? a : b;
    }
    public static void main(String[] args) {
        // 1. 类名.静态成员方法
        System.out.println(Student.getMax(10 , 2));
        // 注意：同一个类中访问静态成员类名可以不写
        System.out.println(getMax(2 , 10));

        // 2. 对象.实例成员方法
        // study(); // 会报错
        Student s = new  Student();
        s.name = "猪八戒";
        s.study();

        // 3. 对象.静态成员方法。（不推荐）
        System.out.println(s.getMax(20 , 10));
    }
}
```



1. 加载Student.class文件到方法区，此时会暴露静态方法的访问入口
2. 提取main方法到栈内存中运行
3. 通过类名访问该类的静态方法，从方法区中提取到栈中运行
4. 在该类中调用该类的静态方法，可以省略不写 类名. ；从方法区中提取到栈中运行；编译后会不写
5. 在栈中开辟一快存储Student类型的空间，用于存储在堆中开辟Student实例的空间的地址
6. 通过句柄（Student实例的地址）对该实例的实例成员变量进行操作
7. 通过实例地址，找到实例中的方法引用，从方法区中提取到占内存中运行

- 工具类

1. 工具类中定义的都是一些静态方法，每个方法都是以完成一个共用的功能为目的。
2. 工具类既可方便调用，又可提高了代码复用（一次编写，处处可用）
   为什么工具类中的方法不用实例方法做？
   实例方法需要创建对象调用，此时用对象只是为了调用方法，这样只会浪费内存
3. 建议将工具类的构造器进行私有，工具类无需创建对象。
4. 工具类里面都是静态方法，直接用类名访问即可

- 注意点

1. 静态方法只能访问静态的成员，不可以直接访问实例成员。可以间接---》通过构造器获取一个实例，再通过句柄调用实例方法
2. 实例方法可以访问静态的成员，也可以访问实例成员。（属性和方法）
3. 静态方法中是不可以出现this关键字和super

- 代码块

1. 代码块是类的5大成分之一（成员变量、构造器，成员方法，代码块，内部类），定义在类中方法外。
2. 在Java类下，使用 {} 括起来的代码被称为代码块

- 静态代码块

格式：static{}
特点：需要通过static关键字修饰，随着类的加载而加载，并且自动触发、只执行一次；优先执行，比main还早
使用场景：在类加载的时候做一些静态数据初始化的操作，以便后续使用

```
public class StaticDemo2 {
    private static String name;

    public static void main(String[] args) {
        System.out.println("===============main=================");
        System.out.println(name);
    }

    /**
     * 随类一起加载，优先执行 用于对静态变量的初始化
     */
    static {
        System.out.println("静态代码块执行");
        name = "fgcy";
    }
}


静态代码块执行
===============main=================
fgcy
```

- 构造(实例)代码块

格式：{}
特点：每次创建对象，调用构造器执行时，都会执行该代码块中的代码，并且在构造器执行前执行（相当于把实例代码块中的代码放到构造器最开始）
使用场景：初始化实例资源

```
package static_demo;

/**
 * ClassName: StaticDemo3 <br/>
 * Description: <br/>
 * date: 2022/3/6 20:18<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class StaticDemo3 {
    private String name;

    public StaticDemo3() {
        System.out.println("构造器执行");
        System.out.println(name);
    }

    /**
```

```
     * 优先于构造器执行，调用构造器先执行该代码段
     */ {
        System.out.println("构造代码块执行");
        name = "123";
    }

    public static void main(String[] args) {
        new StaticDemo3();
    }
}


构造代码块执行
构造器执行
123
```

- 一副扑克牌

```
package static_demo;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * ClassName: StaticDemo4 <br/>
 * Description: <br/>
 * date: 2022/3/6 20:31<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class StaticDemo4 {
    private static List<String> cards = new ArrayList<>();

    static {
        String[] colors = {"♠", "♥", "♣", "♦"};
        // 5、定义点数
        String[] sizes = {"3", "4", "5", "6", "7", "8", "9", "10", "J", "Q",
"K", "A", "2"};
        for (int i = 0; i < sizes.length; i++) {
            for (int j = 0; j < colors.length; j++) {
                cards.add(sizes[i] + colors[j]);
            }
        }
        cards.add("大王");
        cards.add("小王");

    }

    public static void main(String[] args) {
        System.out.println(cards);
        Collections.shuffle(cards);
        System.out.println(cards);
    }
```

```
}


[3♠, 3♥, 3♣, 3♦, 4♠, 4♥, 4♣, 4♦, 5♠, 5♥, 5♣, 5♦, 6♠, 6♥, 6♣, 6♦, 7♠, 7♥, 7♣, 7♦,
8♠, 8♥, 8♣, 8♦, 9♠, 9♥, 9♣, 9♦, 10♠, 10♥, 10♣, 10♦, J♠, J♥, J♣, J♦, Q♠, Q♥, Q♣,
Q♦, K♠, K♥, K♣, K♦, A♠, A♥, A♣, A♦, 2♠, 2♥, 2♣, 2♦, 大王, 小王]
[小王, 10♣, 3♦, J♦, A♠, 4♥, 6♠, J♥, J♣, 7♦, 3♣, Q♦, 5♥, 6♦, 5♠, A♥, 10♥, K♠, J♠,
Q♣, 大王, Q♠, 3♠, 8♥, 9♦, 9♠, K♣, 8♠, K♥, 2♠, Q♥, 5♣, 4♠, 10♦, 4♦, 2♣, 9♥, 10♠,
6♥, A♦, 7♣, 4♣, 2♥, 8♦, 2♦, 9♣, 3♥, K♦, 7♠, A♣, 7♥, 6♣, 5♦, 8♣]



在启动系统时对数据进行初始化。
使用静态代码块完成数据的初始化操作，代码优雅。
```

设计模式之单例

- 什么是设计模式

```
设计模式是一套被前人反复使用，多数人知晓，经过分类编目的代码设计经验总结，后来者可以直接拿来解决问题
好的设计模式能够提高代码的重用性
```

- 单例模式

```
可以保证系统中，应用该模式的这个类永远只有一个实例，即一个类永远只能创建一个对象。
例如任务管理器对象我们只需要一个就可以解决问题了，这样可以节省内存空间。
```

- 饿汉单例

```
package static_demo;

/**
 * ClassName: SingleInstanceDemo1 <br/>
 * Description: <br/>
 * date: 2022/3/6 20:48<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class SingleInstanceDemo1 {

    public static SingleInstanceDemo1 singleInstanceDemo1 = new
SingleInstanceDemo1();
    private SingleInstanceDemo1() {

    }

    public static void main(String[] args) {
        SingleInstanceDemo1 s1 = singleInstanceDemo1;
        SingleInstanceDemo1 s2 = singleInstanceDemo1;
        SingleInstanceDemo1 s3 = singleInstanceDemo1;
        System.out.println(s1==s2);//true
```

```
        System.out.println(s1==s3);//true
    }
}
```

在用类获取对象的时候，对象已经提前为你创建好了

- 懒汉单例

```
package static_demo;

/**
 * ClassName: SingleInstanceDemo1 <br/>
 * Description: <br/>
 * date: 2022/3/6 20:48<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class SingleInstanceDemo2 {

    private static SingleInstanceDemo2 singleInstanceDemo2;

    private SingleInstanceDemo2() {

    }

    public static SingleInstanceDemo2 getInstance() {
        if (singleInstanceDemo2 == null) {
            singleInstanceDemo2 = new SingleInstanceDemo2();
        }
        return singleInstanceDemo2;
    }

    public static void main(String[] args) {
        SingleInstanceDemo2 s1 = getInstance();
        SingleInstanceDemo2 s2 = getInstance();
        SingleInstanceDemo2 s3 = getInstance();
        System.out.println(s1 == s2);//true
        System.out.println(s1 == s3);//true
    }
}
```

在真正需要该对象的时候，才去创建一个对象(延迟加载对象)。

面向对象三大特征之继承

- 继承概念

1. Java中提供一个关键字extends，用这个关键字，我们可以让一个类和另一个类建立起父子关系。
    public class Student extends People {}
    Student称为子类（派生类），People称为父类(基类 或超类)   子类 extends父类
2. 当子类继承父类后，就可以直接使用父类公共的属性和方法了。因此，用好这个技术可以很好的我们提高代码的复用性
3. 子类 继承父类，子类可以得到父类的公开属性和行为，子类可以使用。
4. Java中子类更强大

- 初始继承

```
package inherit;

/**
 * ClassName: Son <br/>
 * Description: <br/>
 * date: 2022/3/6 21:12<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class Son extends Father {
    public void say() {
        System.out.println(getName() + "张开嘴巴啊啊啊");//调用父类的公共方法
    }

    public static void main(String[] args) {
        final Son son = new Son();
        son.setName("苏未晓");//调用父类的公共方法,初始化父类的成员变量
        son.say();
    }
}
```

- 继承设计规范

子类们相同特征（共性属性，共性方法）放在父类中定义，子类独有的的属性和行为应该定义在子类自己里面

- 继承内存图

```java
public class People {
    private String name;
    private int age;

    /**
     共同行为
     */
    public void queryCourse(){
        System.out.println(name  + ", 您可以开始查看您的课表信息了~~~");
    }

    // getter + setter
}
```

```java
public class Student extends People{
    private String className;
    /**
        独有行为
     */
    public void writeInfo(){
        System.out.println(getName() + "今天自己棒棒的，老师也是666~~~");
    }

    // getter + setter

}
```

```java
public class Test {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("翠花"); // 父类的
        s.setAge(22); // 父类的
        s.setClassName("Java就业999期");// 子类的
        System.out.println(s.getName());
        System.out.println(s.getAge());
        System.out.println(s.getClassName());
        s.queryCourse(); // 父类的
        s.writeInfo(); // 子类的
    }
}
```


```
翠花
22
Java就业999期
翠花，您可以开始查看您的课表信息了~~~
翠花今天自己棒棒的，老师也是666~~~
```

1. 将Test.class文件加载进方法区，并暴露main方法的访问入口
2. 在栈内存中开辟一块Student类型的空间，用于存储在堆内存中开辟的存放Student实例相关信息的空间，（其中包含成员属性和方法的引用）
3. 通过句柄访问实例空间中的父类空间的方法
4. 通过句柄访问实例空间中的子类空间的方法

.....

- 继承的特点

1. 子类可以继承父类的属性和行为，但是子类不能继承父类的构造器。子类有自己的构造器，父类构造器用于初始化父类对象
2. Java是单继承模式：一个类只能继承一个直接父类。当多个父类中有相同名字的方法时引起二义性
3. Java不支持多继承、但是支持多层继承，家族体系（当祖宗们有相同的方法时，使用离自己最近的，可以是自己）
4. Java中所有的类都是Object类的子类
5. 子类可以继承父类私有成员，只是不能直接访问（未证实）
6. 子类可以直接使用父类的静态成员（共享），但子类不能继承父类的静态成员。因为静态成员随类加载一次，其并不存在于子类空间中；

```java
package inherit;

/**
 * ClassName: Father <br/>
 * Description: <br/>
 * date: 2022/3/6 21:12<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class Father {

    public static void test(){
        System.out.println("父类的静态方法");
    }

}
```

```java
package inherit;

/**
 * ClassName: Son <br/>
 * Description: <br/>
 * date: 2022/3/6 21:12<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class Son extends Father {

    public static void main(String[] args) {
        Son.test();//这里可以通过子类类名调父类的静态方法
    }
}
```

- 在子类中访问成员（成员属性，成员方法）满足就近原则

先子类局部范围找
然后子类成员范围找
然后父类成员范围找，如果父类范围还没有找到则报错
如果子父类中，出现了重名的成员，会优先使用子类的，此时如果一定要在子类中使用父类的怎么办？
可以通过super关键字，指定访问父类的成员，this关键字访问当前的成员变量

```java
package inherit;

/**
 * ClassName: Demo <br/>
 * Description: <br/>
 * date: 2022/3/6 22:47<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class Demo {
    public static void main(String[] args) {
        final Student student = new Student();
        student.say();
    }
}

class People {
    public String name = "父类名字";

    public void say() {
        System.out.println(name);
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

class Student extends People {
    public String name = "子类名字";

    @Override
    public void say() {
        String name = "子类局部名字";
        System.out.println(name);//子类局部名字 就近原则
        System.out.println(this.name);//实例成员变量
        System.out.println(super.name);//父类实例成员
    }
}


子类局部名字
子类名字
父类名字
```

```
package inherit;

/**
 * ClassName: Demo <br/>
 * Description: <br/>
 * date: 2022/3/6 22:47<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class Demo {
    public static void main(String[] args) {
        final Student student = new Student();
        student.go();
    }
}

class People {
    public void run() {
        System.out.println("父类跑");
    }
}

class Student extends People {
    public void go() {//中转方法
        run();//就近原则
        super.run();
    }
```

```
    @Override
    public void run() {
        System.out.println("子类跑");
    }
}
子类跑
父类跑
```

- 方法重写

在继承体系中，子类出现了和父类中一模一样的方法声明，我们就称子类这个方法是重写的方法

方法重写的应用场景
    当子类需要父类的功能，但父类的该功能不完全满足自己的需求时。
    子类可以重写父类中的方法

@Override重写注解
    @Override是放在重写后的方法上，作为重写是否正确的校验注解。
    加上该注解后如果重写错误，编译阶段会出现错误提示。
    建议重写方法都加@Override注解，代码安全，优雅！

方法重写注意事项和要求
    重写方法的名称、形参列表必须与被重写方法的名称和参数列表一致。(方法名，形参列表，返回值类型【全部与父类一致】，修饰符【子类大于等于父类】)
    私有方法不能被重写。（子类无法访问）
    子类重写父类方法时，访问权限必须 大于或者等于 父类 （暂时了解 ：缺省 < protected < public）
    子类不能重写父类的静态方法，如果重写会报错的；（子类没有继承父类的静态方法）

- 子类构造器特点

子类中 所有的 构造器默认都会先访问父类中 无参 的构造器，再执行自己

为什么？
    子类在初始化的时候，有可能会使用到父类中的数据，如果父类没有完成初始化，子类将无法使用父类的数据。
    子类初始化之前，一定要调用父类构造器先完成父类数据空间的初始化

子类构造器的第一行语句默认都是：super()，不写也存在

```
package inherit;

/**
 * ClassName: Demo <br/>
 * Description: <br/>
 * date: 2022/3/6 22:47<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
```

```
public class Demo {
    public static void main(String[] args) {
        new Student();
        System.out.println("--------------------------------------------");
        new Student("jjj");
    }
}

class People {
    public People() {
        System.out.println("父类构造器");
    }
}

class Student extends People {
    private String name;

    public Student(String name) {
        this.name = name;
        System.out.println("子类有参构造");
    }

    public Student() {
        System.out.println("子类构造器");
    }
}
```

```
父类构造器
子类构造器
--------------------------------------------
父类构造器
子类有参构造
```

- 子类构造器访问父类有参构造

super调用父类有参数构造器的作用：
    初始化继承自父类的数据。

如果父类中没有无参数构造器，只有有参构造器，会出现什么现象呢？
    会报错。因为子类默认是调用父类无参构造器的。

如何解决？
    子类构造器中可以通过书写 super(…)，手动调用父类的有参数构造器

```
package inherit;

/**
 * ClassName: Demo <br/>
 * Description: <br/>
 * date: 2022/3/6 22:47<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
```

```java
public class Demo {
    public static void main(String[] args) {
        final Student student = new Student("苏未晓", "睡觉", true);
        System.out.println(student.getName());
        System.out.println(student.getHobbies());
        System.out.println(student.isYoungster());
    }
}

class People {
    private String name;
    private String hobbies;

    public People() {//会被子类默认调用
    }

    public People(String name, String hobbies) {
        this.name = name;
        this.hobbies = hobbies;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getHobbies() {
        return hobbies;
    }

    public void setHobbies(String hobbies) {
        this.hobbies = hobbies;
    }
}

class Student extends People {
    private boolean youngster;

    public Student(String name, String hobbies, boolean youngster) {
        super(name, hobbies);//这里用父类的有参构造代替了默认的父类无参构造
        this.youngster = youngster;
    }

    public Student() {//默认在所有子类构造器的第一行调用父类无参构造器
    }

    public boolean isYoungster() {
        return youngster;
    }

    public void setYoungster(boolean youngster) {
        this.youngster = youngster;
    }
}
```

- this与super

- **this**：代表本类对象的引用；**super**：代表父类存储空间的标识。

关键字	访问成员变量	访问成员方法	访问构造方法
this	this.成员变量 访问本类成员变量	this.成员方法(...) 访问本类成员方法	this(...) 访问本类构器
super	super.成员变量 访问父类成员变量	super.成员方法(...) 访问父类成员方法	super(...) 访问父类构造器

- 通过this()调用本类的兄弟构造器

this(...)和super(…)使用注意点：
子类通过 this (...) 去调用本类的其他构造器，本类其他构造器会通过 super 去手动调用父类的构造器，最终还是会调用父类构造器的。
注意：this(…) super(…) 都只能放在构造器的第一行，所以二者不能共存在同一个构造器中

```
package inherit;

/**
 * ClassName: Demo <br/>
 * Description: <br/>
 * date: 2022/3/6 22:47<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class Demo {
    public static void main(String[] args) {
        final People people = new People("苏未晓");
        System.out.println(people.getHobbies());
        System.out.println(people.getName());
    }
}

class People {
    private String name;
    private String hobbies;

    public People() {
    }

    public People(String name) {
        //super();//这里不需要调用父类的构造器，因为兄弟构造器中已经调用了，不需要重复调用
this和super都只能放在构造器的第一行，所以两者不能共存于同一个构造器中
        this(name, "睡觉");//调用本类的其他构造器
    }
```

```
    public People(String name, String hobbies) {
        this.name = name;
        this.hobbies = hobbies;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getHobbies() {
        return hobbies;
    }

    public void setHobbies(String hobbies) {
        this.hobbies = hobbies;
    }
}

睡觉
苏未晓
```