**java面向对象**

# 一.面向对象的基本概述

## 1、java面向对象语言

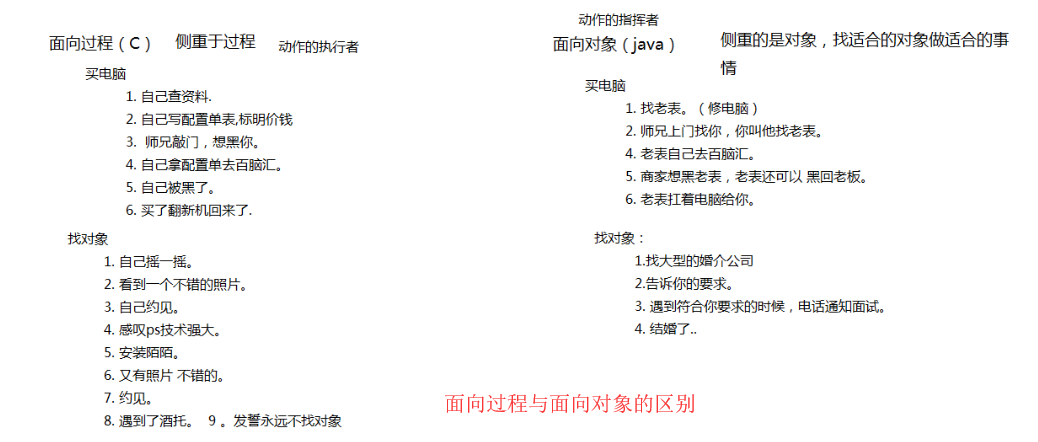
（1）对象：真是存在的唯一事务

（2）类：对某中类型的事务的共同属性和行为的抽取，是一种抽象概念……

## 2、面向对象（java）与面向过程（c语言）的区别

（1）面向过程（c语言）侧重于过程，动作的执行者。

（2）面向对象（java）侧重于对象，找适合的对象做适合的事，动作的指挥者。



## 3、面向对象的核心思想：

* 找合适的对象做适合的事。
* 如何找到适合的对象？
  + sun已经定义好了很多类，我们只需要认识这些类，就可以通过类来创建对象
  + 自定义类，通过自定义类来创建我们需要的对象使用。

## 4、自定义类创建对象*// 车类*

1. 自定义类。

格式：

class 类名{

事物的公共属性使用成员变量描述。

事物的公共行为使用函数描述。

}

2. 通过自定义类创建对象。

格式：

类名 变量名 = new 类名();

3. 访问(设置)对象的属性或者调用对象的功能。

1.访问对象属性的格式：

对象.属性名.

2.设置对象的属性：

对象.属性名 = 数据。

3. 调用对象的功能

对象.函数名();

|  |
| --- |
| **public static void** main(String[] args) {  *//使用了Car类声明了一c变量， c变量指向了一个车对象。* Car c = **new** Car();  *//设置车对象的属性值。* c.**name** = **"BMW"**;  c.**color** = **"白色"**;  c.**wheel** = 4;  *//访问车对象的属性值* System.***out***.println(**"名字："** + c.**name** + **" 颜色:"** + c.**color** + **" 轮子数："** + c.**wheel**);  c.run(); } |

## 5、局部变量与成员变量

（1）位置上区别

* 成员变量：定义在方法之外，类之内的
* 局部变量：定义在方法之内的

（2）作用上的区别

* 成员变量：是用于描述一类事物的公共属性
* 局部变量：提供一个变量给方法内部使用

（3）生命周期的区别

* 成员变量：随着对象的创建而创建，随着对象的消失而消失
* 局部变量：在调用了对应的方法执行到创建该变量的语句时存在，局部变量一旦出了自己的作用域马上消失

（4）初始值的区别

* 成员变量：有默认的初始值
* 局部变量：没有默认的初始值，必须先初始化才能使用。

**注：成员变量的初始值**

**int 0 boolean false**

**float 0.0f char ‘ ’**

**double 0.0 String null**

# 二、匿名对象

1、什么是匿名对象？

* 没有引用类型变量指向的对象称作匿名对象
* 例如 new Student();

2、匿名对象注意事项

* 一般不会给匿名对象赋予属性值，因为我们无法获取到
* 两个匿名对象永远不可能是同一个对象

3、匿名对象的使用场景

* 如果一个对象需要调用一个方法，而调用完这个方法后，该对象就不再使用了，这个时候可以使用匿名对象。
* 可以作为实参调用一个函数

# 三、封装

## 1、面向对象三大基本特征（封装、继承、多态）

## 2、权限修饰符

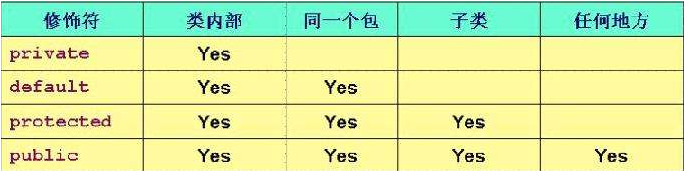
public 公有的，任何人都能访问

protected 包

default 默认

private 私有的

* private（私有的）：可以修饰成员变量、成员方法、构造方法，不能修饰类（外部类，内部类不加考虑）、不能修饰局部变量。只能在本类中被访问，不能被其他类访问，但是可以通过get、set方法访问。
* default（默认的）：可以修饰成员变量、成员方法、构造方法、局部方法、类，默认修饰后，只能被同包下的类访问
* protected（受保护的）：可以修饰成员变量、成员方法、构造方法，不能修饰类（外部类，内部类不加考虑），不能修饰局部变量，只能被同包或其子类访问。
* public（公共的）：最大权限修饰符，可以修饰类、成员变量、成员方法、构造方法、可以在任何类中被使用。



## 3、封装的步骤

（1）使用private修饰需要被封装的属性

（2）提供一个public 方法设置或获取该私有的成员属性

* 命名规范：

set() 属性名

get() 属性名

## 4、封装的好处

* 提高数据安全性
* 操作简单
* 隐藏了实现

## 4、疑问？封装一定要提供get或set方法吗？

* 不一定，根据需求而定

## 5、封装的规范

* 在现实开发中，一般实体类的所有成员属性都要封装起来。

## 6、实体类与工具类

* 实体类：用来描述一类事物的就称作实体类
* 工具类：Arrays数组工具类等

# 四、构造函数

## 1、要注意的细节

（1）构造函数 没有返回值

（2）构造函数 函数名必须与类名一致

（3）构造函数 是在创建对象时由jvm虚拟机主动调用对应的构造函数

（4）如果一个类没有显示的构造函数，那么java编译器会为该类添加一个无参的构造函数。

（5）如果一个类已经存在显示的构造函数，那么java编译器不会为该类添加一个无参的构造函数

（6）构造函数 是可以在一个类中以 函数重载的 形式存在多个的。

## 2、构造函数、代码块、静态代码块的执行顺序

（1）构造函数

* + 对象一建立，jvm虚拟机就会立即调用构造函数，每创建一个对象，构造函数就会被调用一次。
  + 构造函数是用于给指定对象进行初始化的（为当前构建的对象进行初始化），**因为构造函数可以有多个，也就是可以创建多个不一样的对象。**

（2）代码块

* 对象一建立就运行构造代码块，并且优先于构造函数。
* **构造代码块是对所有对象进行统一的初始化（通过该类创建的所有对象），构造函数是对当前对象就行初始化。**

（3）静态代码块

* **静态代码块是被类调用的，一旦类被加载，静态代码块就被执行，并且只被执行一次。**
* **静态代码块优于主函数，静态代码块是给类初始化的，而构造代码块是给对象初始化的**。
* 一个类中可以有多个静态代码块。按代码顺序从上到下顺序执行。

## 3、疑问？

### （1）创建对象时，jvm调用的构造方法从何而来？

* 每个类都有默认的无参构造函数，创建对象时，Java编译器会在编译的时候给加上。

### （2）java编译器添加的无参构造方法的 权限修饰符 是什么？

* 与 类 的权限修饰符是 一致 的

## 3、反编译

* + Jdk给我们提供了一个开发工具（javap.exe），给我们进行反编译的。

### （1）javap.exe反编译工具的使用格式

* + javap –c –l –private 类名

## 4、构造函数与普通函数的区别

1. 返回值类型
   * 构造函数没有返回值类型
   * 普通函数有返回值类型，即使没有也要写上void
2. 函数名

* 构造函数的函数名要与类名一致。
* 普通函数的函数名只需要符合标识符命名规则即可

1. 调用方式

* 构造函数在创建对象的时候，由jvm主动调用一次
* 普通函数需要我们通过对象手动调用，一个对象可以多次调用一个或多个普通的函数。

1. 作用

* 构造函数用来初始化一个对象
* 普通函数用来描述一类事物的公共行为。

# 五、this关键字

## 1、this关键字

* this关键字代表了所属函数的调用者对象（调用该方法的是哪个对象，this就代表了这个对象）

## 2、this关键字的作用

* 如果存在同名的成员变量与局部变量时，方法内部默认调用的是局部变量数据（java就近原则）。可以通过this关键字指定访问成员变量数据。
* 在一个构造函数中，可以调用本类另一个构造函数初始化对象
  + - this(name) 表示调用本类中一个参数的构造函数

## 3、this关键字注意事项

* 如果在一个方法中访问一个变量，该变量只存在成员变量的情况下（不存在同名的局部变量），那么java编译器会在该变量的 前面添加this关键字

## 4、this关键字调用其他构造函数时要注意的事项

* this关键字调用其他构造函数时，必须位于构造函数的第一个语句
* this关键字在构造函数中不能出现互相调用，因为这是一个死循环。

# 六、static关键字

## 1、static关键字（公共的、共享的）

* static（静态\修饰符）
* 如果有数据需要共享给所有对象使用时，那么就可以使用static修饰

## 2、static修饰成员变量

### （1）静态成员变量的访问方式

* 方式一：可以使用对象进行访问

格式：对象.变量名

* 方式二：可以使用类名进行访问

格式：类名.变量名

### （2）注意：

* 非静态成员变量只能使用对象进行访问，静态成员变量可以使用 对象 和 类名 两种方式进行访问。
* 只有成员变量的数据需要共享的时候，才使用static关键字
* 共享的（static）成员变量，被任意修改一次的话，以后的数据都是被修改后的数据。

### （3）static修饰成员变量的应用场景

* 如果一个变量需要被所有对象共享使用时，这个时候即可使用static关键字

## 3、static修饰成员函数

### （1）访问方式：

* 方式一：可以使用对象进行访问。

格式：对象.静态的函数名();

* 方式二：可以使用类名进行访问。

类名.静态函数名字。

* **推荐使用是类名直接访问静态的成员。**

### （2）静态的成员变量与非静态的成员变量的区别：

1. 作用上的区别：

* + 静态的成员变量的作用共享一个 数据给所有的对象使用。
  + 非 静态的成员变量的作用是描述一类事物的公共属性。

2. 数量与存储位置上的区别：

* 静态成员变量是存储方法 区内存中，而且只会存在一份数据。
* 非静态的成员变量是存储在堆内存中，有n个对象就有n份数据。

3. 生命周期的区别：

* 静态的成员变量数据是随着类的加载而存在，随着类文件的消失而消失。
* 非静态的成员数据是随着对象的创建而存在，随着 对象被垃圾回收器回收而消失。

### （3）静态函数要注意的事项：

1. 静态函数是可以使用类名或者对象进行调用的，而非静态函数只能使用对象进行调用。

2. 静态的函数可以直接访问静态的成员，但是不能直接访问非静态的成员。

* 原因：静态函数是可以使用类名直接调用的，这时候可能还没有存在对象，而非静态的 成员数据是随着对象 的存在而存在的。

3. 非静态的函数是可以直接访问静态与非静态的成员。

* 原因：非静态函数只能由对象调用，当对象存在的时候，静态数据老早就已经存在了，而非静态数据也随着对象的创建而存在了。

4. 静态函数不能出现this或者super关键字。

* 原因：因为静态的函数是可以使用类名调用的，一旦使用类名调用这时候不存在对象，而this关键字是代表了一个函数 的调用者对象，这时候产生了冲突。

### （4）静态的数据的生命周期：

* 静态的成员变量数据是优先于对象存在的。

### （5）static什么时候修饰一个函数？

* 如果一个函数没有直接访问到非静态的成员时，那么就可以使用static修饰了。 **一般用于工具类型的方法**

### （6）静态函数不能访问非静态的成员？

* 静态函数只要存在有对象，那么也可以访问非 静态的数据。只是不能直接访问而已。

# 七、main方法详解（主函数）

public ： 公共的。 权限是最大，在任何情况下都可以访问。 private

原因： 为了保证让jvm在任何情况下都可以访问到main方法。

static: 静态。

静态可以让jvm调用main函数的时候更加的方便。不需要通过对象调用。

void: 没有返回值。

因为返回的数据是 给 jvm，而jvm使用这个数据是没有意义的。所以就不要了。

main: 函数名。

注意： main并不是关键字,只不过是jvm能识别的一个特殊的函数名而已。

arguments ：担心某些程序在启动需要参数。

# 八、单利模式

## 1、单例设计模式：

* 保证一个类在内存中只有一个对象。

## 2、单例设计模式的步骤：

### （1）饿汉单例设计模式

1. 私有化构造函数。

2. 声明本类的引用类型变量，并且使用该变量指向本类对象。

3. 提供一个公共静态的方法获取本类的对象。

|  |
| --- |
| **// 确保Single在内存中只有一个对象（饿汉单例模式） public class** Single {  // 私有化构造函数，确保外部 不能创建Single对象  **private** Single() {  }   //初始化对象；static静态的，保证类加载后就创建该对象；private私有的，确保外部不能访问该对象  **private static** Single single = **new** Single();   // 对外提供获取该对象实例的方法  **public static** Single getInstance() {  **return** single;  } }  **// 单例模式测试类**  **class** SingleTest {  **public static void** main(String[] args) {  Single single1 = Single.getInstance();  Single single2 = Single.getInstance();  System.**out**.println(**"Single1 和 Single2是否是同一个对象？"** + (single1 == single2));  } } |

### （2）懒汉单例设计模式：

1. 私有化构造函数。

2. 声明本类的引用类型变量，但是不要创建对象，

3. 提供公共静态 的方法获取本类 的对象，获取之前先判断是否已经创建了本类 对象

，如果已经创建了，那么直接返回对象即可，如果还没有创建，那么先创建本类的对象，

然后再返回。

|  |
| --- |
| **// 确保Single在内存中只有一个对象（懒汉单例模式）** **public class** Single {  // 声明对象，但不创建对象  **private static** Single single;   // 私有化构造函数  **private** Single() {   }   // 提供获取Single对象的方法  **public static** Single getInstance() {  // 判断Single是否已被创建，如果没有被创建，那么创建single对象；如果已经存在，直接返回single对象  **if** (single == **null**) {  single = **new** Single();  }  **return** single;  } }  **// 单例模式测试类**  **class** SingleTest {  **public static void** main(String[] args) {  Single single1 = Single.getInstance();  Single single2 = Single.getInstance();  System.**out**.println(**"Single1 和 Single2是否是同一个对象？"** + (single1 == single2));  } } |

**推荐使用： 饿汉单例设计模式。 因为懒汉单例设计模式会存在线程安全问题，目前还不能保证一类在内存中只有一个对象。**

# 九、继承（extends）

面向对象的三大特征：

1. 封装

2. 继承

3. 多态.

## 1、继承：

* + 继承是通过关键字extends体现的。

2、继承的格式：

class 类名1 extends 类名2{ }

3、继承要注意的事项：

1. 千万不要为了减少重复代码而去继承，只有真正存在着继承关系的时候才去继承。

2. 父类私有的成员不能被继承。

3. 父类的构造函数不能被继承。

4. 创建子类对象时默认会先调用父类无参的构造函数。

4、疑问： 为什么要调用父类的构造方法啊？这样子做的意义在那？

* 调用父类 的构造方法是可以初始化从父类继承下去的属性的。

## 2、方法重写

（1）方法重写的前提： 必须要存在继承的关系。

（2）什么是方法的重写？

* 子父类出了同名的函数，这个我们就称作为方法的重写。

（3）什么是时候要使用方法的重写？

* 父类的功能无法满足子类的需求时。

（4）方法重写要注意的事项：

1.方法重写时， 方法名与形参列表必须一致。

2.方法重写时，子类的权限修饰符必须要大于或者等于父类的权限修饰符。

3.方法重写时，子类的返回值类型必须要小于或者 等于父类的返回值类型。

4.方法重写时， 子类抛出的异常类型要小于或者等于父类抛出的异常类型。

Exception(最坏)

RuntimeException(小坏)

## 3、方法重载

* 方法的重载：在一个类中 存在两个或者两个 以上的同名函数,称作为方法重载。
* 方法重载的要求

1. 函数名要一致。

2. 形参列表不一致（形参的个数或形参 的类型不一致）

3. 与返回值类型无关。

# 十、抽象类（abstract）

## 1、抽象类的应用场景：

我们在描述一类事物的时候，发现该种事物确实存在着某种行为，但是这种行为目前是不具体的，那么我们可以抽取这种行为 的声明，但是不去实现该种行为，这时候这种行为我们称作为抽象的行为，我们就需要使用抽象类。

## 2、抽象类的好处

* 强制要求子类一定要实现指定的方法。

## 3、抽象类要注意的细节：

1. 如果一个函数没有方法体，那么该函数必须要使用abstract修饰,把该函数修饰成抽象 的函数。。

2. 如果一个类出现了抽象的函数，那么该类也必须 使用abstract修饰。

3. 如果一个非抽象类继承了抽象类，那么必须要把抽象类的所有抽象方法全部实现。

4. 抽象类可以存在非抽象方法，也可以存在抽象的方法.

5. 抽象类可以不存在抽象方法的。

5. 抽象类是不能创建对象的。

* + **疑问：为什么抽象类不能创建对象呢？**
    - **因为抽象类是存在抽象方法的，如果能让抽象类创建对象的话，那么使用抽象的对象调用抽象方法是没有任何意义的。**

6. 抽象类是存在构造函数的，其构造函数是提供给子类创建对象的时候初始化父类的属性的。

## 4、abstract不能与以下关键字共同修饰一个方法：

* **abstract不能与private、static、final共同修饰一个方法。**

1. abstract与private：

* abstract抽象类是为了让别的类重写其方法的，private只能当前类可用，冲突

（2）abstract与static：

* static修饰的方法可以用 类.方法名 的方式来调用，但是abstract修饰的方法没有方法体，被调用没有意义。

（3）abstract与final：

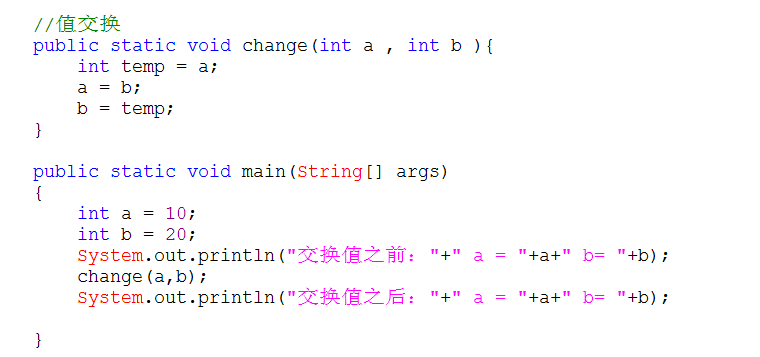
* final修饰的方法是最终方法，不能被重写，abstract修饰的方法是用来被重写的，冲突。

# 十一、值交换

* 值传递：调用一个方法的时候，传递给方法的参数，实际上是传递变量所存储的值。

**案例： 定义交换数值的功能函数，基本类型数据, 数组,实例对象, String。**

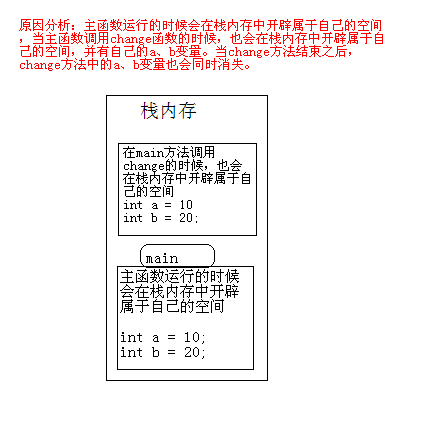
## 1、基本数据类型交换



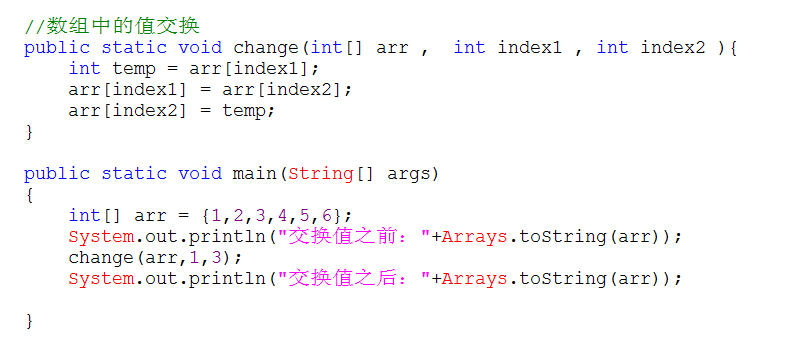


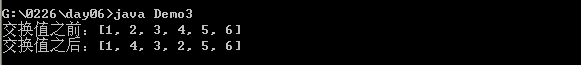
结果：发现交换值前后没有变量的值发生变化。

原因分析：

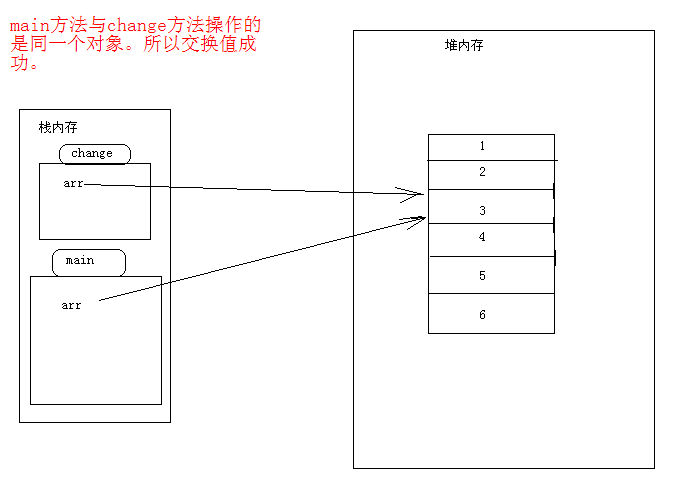


## 2、数组类型交换



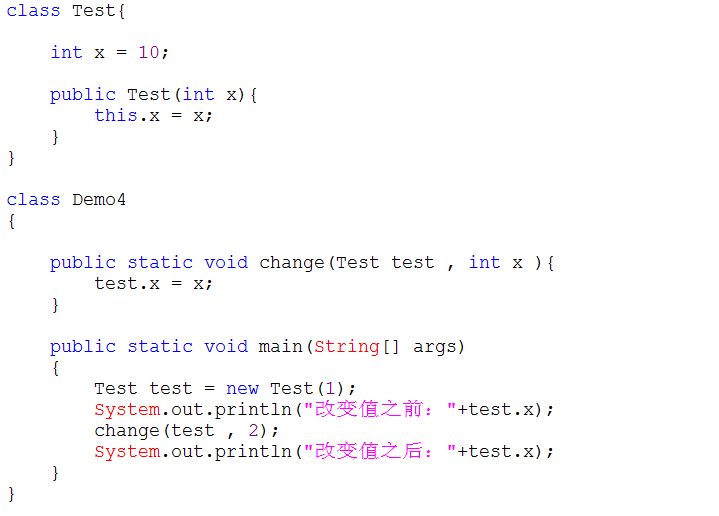


**结果：交换值成功。**



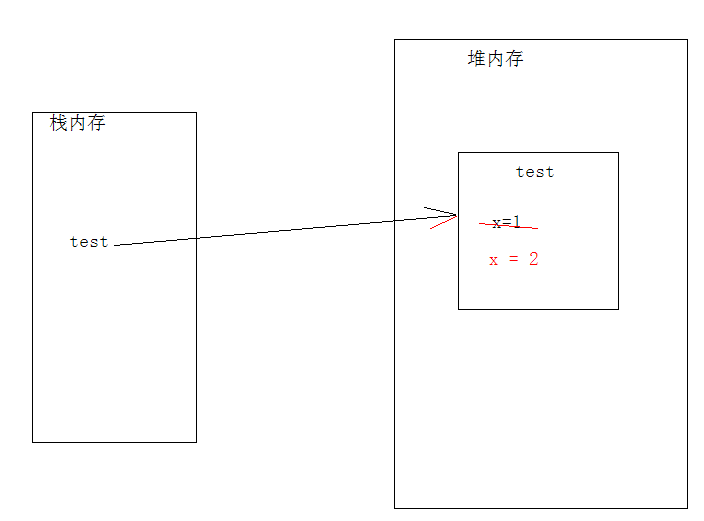
原因分析：操作的是同一个数组对象。

## 3、对象的值交换：

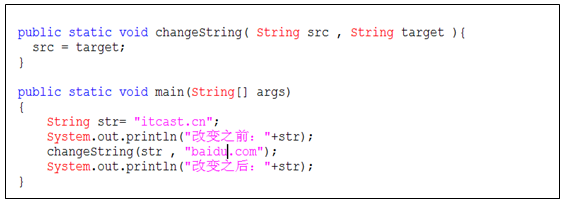




结果：交换值成功。

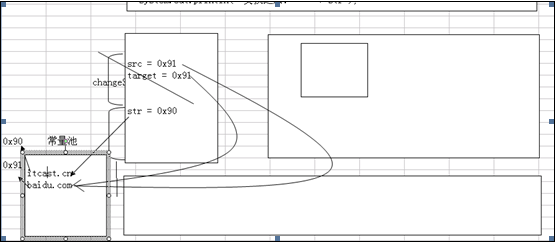


## 4、字符串的值交换





结果：交换失败



# 十二、接口（interface）

## 1、接口格式

* interface 接口名{

public static final int aa = 123;

public static final String bb = “字符串”;

public abstract void aa();

public abstract void bb();

}

## 2、接口的实现方式

* class 类名 implement 接口1，接口2…{

@Override

类 所实现的 接口 的所有 方法(){

}

}

|  |
| --- |
| public interface A {  int a = 1;  String b = "aa";  void test();  void aa();  }  interface B {  void bb();  }  class C implements A,B{  @Override  public void test() {  }  @Override  public void aa() {  }  @Override  public void bb() {    }  } |

## 3、接口要注意的事项

**1、接口是一个特殊的类**

**2、接口的所有成员变量都是 常量（public static final 修饰）**

**3、接口中的所有方法都是 抽象方法（public abstract）**

**4、接口不能创建对象，因为接口没有构造方法**

**5、接口是给类去实现的使用的，非抽象类（非abstract）实现一个接口，要实现这个接口的所有方法（因为接口的所有方法都是抽象方法）**

## 4、接口的作用

1、解耦

2、定义规范

3、拓展功能

## 5、接口与类之间的关系

* 类与接口之间关系： 实现关系。
* 类与接口要注意的事项：

1. 非抽象类实现一个接口时，必须要把接口中所有方法全部实现。

2. 抽象类实现一个接口时，可以实现也可以不实现接口中的 方法。

3. 一个类可以实现多个接口 。

## 6、接口与接口之间的关系

* 接口与接口之间关系： 继承关系。
* 接口与接口之间要注意事项：

1. 一个接口是可以继承多个接口的。

## 7、疑问： java为什么不支持多继承，而支持了多实现呢？

* 如果是多继承的话，如果A继承了B和C，B和C存在相同名称的成员变量和方法，那么当A创建对象并调用这个方法时，调用的是B和C哪个类的方法呢？这里存在问题。
* 而实现多个接口，可以避免这个问题，因为接口没有方法体，没有具体的实现。

# 十三、多态

## 1、什么是多态？

* 一个对象具备多种形态
  + 父类的引用类型变量指向了子类对象
  + 接口的引用类型变量指向了接口实现类的对象

## 2、多态的前提

* 存在继承或实现关系

## 3、多态要注意的细节

（1）多态情况下，子父类存在同名成员变量时，对象访问的是父类的成员变量。

（2）多态情况下，子父类存在同名静态（static）函数时，对象访问的是父类的静态函数。

（3）多态情况下，子父类存在同名非静态函数（普通函数）时，对象访问的是子类的非静态函

（4）多态情况下，不能访问子类特有的成员

**总结：**

* + **多态情况下，子父类存在同名成员时，对象只能访问子类的非静态函数，其他的都是访问父类的成员（包括静态函数、成员变量、常量等）**
  + **多态情况下，对象不能访问子类特有的成员（成员变量、静态函数、非静态函数等，都不能访问，只能访问与父类同名的非静态函数）**

4、编译看左边，运行不一定看右边

* + Java编译器在编译的时候，会检查引用类型变量所属的类是否具备指定的成员，如果不具备马上报错。

## 5、多态的应用场景

（1）多态用于形式参数的时候，可以接受更多类型的数据

（2）多态用于返回值类型的时候，可以返回更多类型的数据

|  |
| --- |
| //需求： 定义一个函数可以接收任意类型的动物对象，在函数内部要调用到动物特有的方法。  public static void print(Animal a){ // Animal a = new Mouse("米老鼠");  if(a instanceof Fish){  Fish f = (Fish)a;  f.bubble();  }else if(a instanceof Mouse){  Mouse m = (Mouse)a;  m.dig();  }  } |

## 6、多态的好处

* + 提高了代码的拓展性

## 7、实现关系下的多态

格式：

**接口 变量名 = new 实现类();**

# 十四、内部类

* 什么是内部类？
* 一个类定义在另一个类的内部，那么该类就称作内部类
* 内部类的class文件命名方式？
* 外部类$内部类.class
* 好处：便于区分该class文件是属于哪个外部类

## 1、成员内部类

* 成员内部类的访问方式
  + 方式一：在外部类提供一个方法，创建内部类的对象进行访问
  + 方式二：在其他类直接创建内部类的对象：
    - 格式：

外部类.内部类 变量名 = new 外部类().new 内部类();

**注意：**

**如果是一个静态内部类，那么其创建格式是：**

**外部类.内部类 变量名 = new 外部类.内部类();**

4、内部类的应用场景：

我们在描述A事物的时候，发现描述的A事物内部还存在另外一个比较复杂的事物B时候，而且这个比较复杂事物B还需要访问A事物的属性等数据，那么这时候我们就可以使用内部类描述B事物。

5、内部类的好处？

* 内部类可以直接访问外部类的所有成员

6、内部类要注意的细节：

* 如果外部类与内部类存在同名的成员变量时，在内部类中默认情况下是访问内部类的成员变量。可以通过"外部类.this.成员变量名" 指定访问外部类的 成员。
* **私有的（private）**成员内部类只能在外部类提供一个方法创建内部类的对象进行访问，不能在其他类创建对象了。
* 成员内部类一旦出现了静态的成员，那么该类也必须 使用static修饰。

原因：静态数据不需要依赖对象而存在，而内部类需要依赖外部类的对象才能被访问，这就违背了静态（static）的原则。

## 2、局部内部类

（1）局部内部类：

* 在一个类 的方法内部定义另外一个类，那么另外一个类就称作为局部内部类。

（2）局部内部类要注意的细节：

* 如果局部 内部类访问了一个局部变量，那么该局部变量必须使用final修饰、
  + 原因：
    - 局部变量在方法执行完毕后就会被销毁
    - 而内部类的对象是在不被使用后，被jvm垃圾回收器不定时销毁，这就表示内部类的对象比局部变量的声明周期长。
    - 当局部变量被销毁但内部类对象还在访问局部变量，这就给人感觉局部变量生命周期延长了
  + 解决方案：
    - 如果一个局部内部类访问一个局部变量的时候，那么就让这个局部内部类访问这个局部变量的复制品。
    - 复制品，就是与原品一模一样的存在，不能被修改，那么使用final就能保证这个复制品与原品一模一样，不会被修改。

## 3、匿名内部类

1、什么是匿名内部类？

* 没有类名的类就称作为匿名内部类。

2、匿名内部类的好处：

* 简化书写。

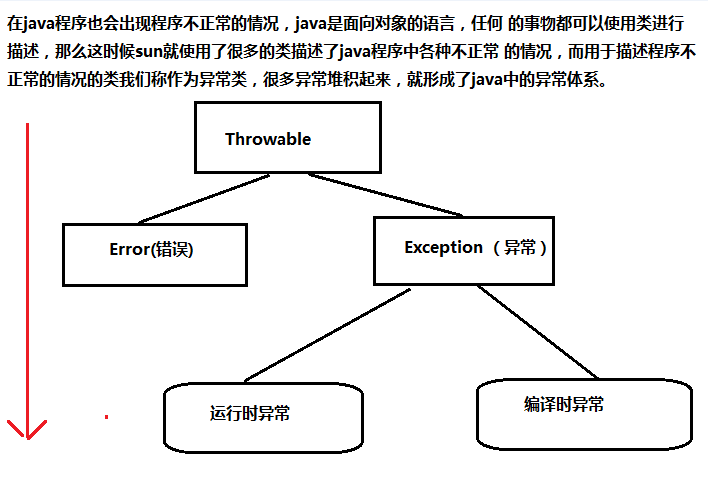
1. 匿名内部类的使用前提：
   * 必须存在继承或者实现关系才能使用。

**4、匿名内部类一般是用于 实参**

# 十五、异常

1. 异常体系
   * Throwable：所以异常或者错误类的超类
     + Error（错误）：错误一般是用于jvm或者是硬件引发的问题，所以我们一般不会通过代码去处理错误的
     + Exception（异常）：是需要通过代码去处理的。
2. 如何区分错误与异常呢：

* 如果程序出现了不正常的信息，如果不正常的信息的类名是以Error结尾的，那么肯定是一个错误。
* 如果是以Exception结尾的，那么肯定就是一个异常。



2、异常：

我们的java程序也是会存在某些不正常 的情况的，这些不正常的 情况我们就统称异常。

## 3、Throwable常用的方法：

* 1. toString() 返回当前异常对象的完整类名+病态信息。
  2. getMessage() 返回的是创建Throwable传入的字符串信息。
  3. printStackTrace() 打印异常的栈信息。

**4、java虚拟机在默认的情况下只能管理64m内存。**

## 5、疑问： 下面的信息是通过printStackTrace方法打印出来，那么异常对象从何而来呢？

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Demo10.div(Demo10.java:10)

at Demo10.main(Demo10.java:5)

* jvm运行到a/b这个语句的时候，发现b为0，除数为0在我们现实生活中是属于不正常的情况，jvm一旦发现了这种不正常的情况时候，那么jvm就会马上创建一个对应的异常对象，并且会调用这个异常对象 的printStackTrace的方法来处理。

## 6、异常的处理方式：

### （1）方式一：捕获处理

* 捕获处理的格式：

**try{**

**可能发生异常的代码；**

**}catch(捕获的异常类型 变量名){**

**处理异常的代码....**

**}**

1）捕获处理要注意的细节：

* + 如果try块中代码出了异常经过了处理之后，那么try-catch块外面的代码可以正常执行。
  + 如果try块中出了异常的代码，那么在try块中出现异常代码后面的代码是不会执行了。
  + 一个try块后面是可以跟有多个catch块的，也就是一个try块可以捕获多种异常的类型。
  + 一个try块可以捕获多种异常的类型,但是捕获的异常类型必须从小到大进行捕获，否则编译报错。

2）异常疑问：

疑问一 : 异常的处理感觉没有多大作用，因为都是输出一句话而已?

异常处理非常有用，只不过是由于我们目前所接触的知识点太过于局限而已。

疑问二： 以后捕获处理 的时候是否就是捕获Exception即可？

错的，因为我们在现实开发中遇到不同的异常类型的时候，我往往会有不同 的处理方式。所以要分开不同的异常类型处理。

### （2）异常的处理方式二：抛出处理

* 抛出处理（throw throws）
* 抛出处理要注意的细节：
  + 如果一个方法的内部抛出了一个异常 对象，那么必须要在方法上声明抛出。
  + 如果调用了一个声明抛出异常 的方法，那么调用者必须要处理异常。
  + 如果一个方法内部抛出了一个异常对象，那么throw语句后面的代码都不会再执行了（一个方法遇到了throw关键字，该方法也会马上停止执行的）。
  + 在一种情况下，只能抛出一种类型异常对象。
* throw 与throws两个关键字：
  + throw关键字是用于方法内部的，throws是用于方法声明上的。
  + throw关键字是用于方法内部抛出一个异常对象的，throws关键字是用于在方法声明上声明抛出异常类型的。
  + throw关键字后面只能有一个异常对象，throws后面一次可以声明抛出多种类型的 异常。
* 疑问：何时使用抛出处理？何时捕获处理？原则是如何？
  + 如果你需要通知到调用者，你代码出了问题，那么这时候就使用抛出处理.
  + 如果代码是直接与用户打交道遇到了异常千万不要再抛，再抛的话，就给了用户了。这时候就应该使用捕获处理。

## 7、自定义异常

（1）自定义异常类，格式为 \*\*Exception 结尾，继承超类Exception

|  |
| --- |
| *// 自定义异常类，继承超类 Exception* **class** NotMoneyException **extends** Exception {  *// 调用父类Exception 的带参构造* **public** NoMoneyException(String message) {  **super**(message);  } } |

1. 在需要抛出该异常的代码中，抛出异常对象，并在方法上声明抛出异常的类型

|  |
| --- |
| *// 将方法内抛出的异常进行声明*  public static void eat(double maney) throws NoMoneyException {if (maney < 10) {  *//抛出异常对象*  throw new NoMoneyException("余额不足，无法想用香喷喷的地沟油木桶饭");}  System.*out*.println("余额充足，请享受香喷喷的地沟油木桶饭"); } |

1. 在调用该方法的位置，对方法进行异常捕获并处理（try-catch-finally）

|  |
| --- |
| *// 将带有异常声明的方法进行 try-catch-finally异常捕捉，然后进行处理* **try** {  eat(10); // 捕捉并处理异常 } **catch** (NotSufficientFundsException e) {  e.printStackTrace();  System.***out***.println(e.getMessage()); } **finally** {  System.***out***.println(**"不管是否发生异常，都会执行finally块的代码"**); } |

## 8、运行时异常及编译时异常

异常体系：

* Throwable 所有错误或者异常的父类
  + Error（错误）：
  + Exception(异常) ：异常一般都通过代码处理
    - RuntimeException（运行时异常）：如果一个方法内部抛出了一个运行时异常，那么方法上 可以声明也可以不 声明，调用者可以以处理也可以不处理。
    - 编译时异常(非运行时异常、受检异常): 如果一个方法内部抛出了一个编译时异常对象，那么方法上就必须要声明，而且调用者也必须要处理。
* 运行时异常： RuntimeException以及RuntimeException子类 都是属于运行时异常。
* 编译时异常： 除了运行时异常就是编译异常。

疑问： 为什么java编译器会如此严格要求编译时异常，对运行时异常如此宽松？

* 运行时异常都是可以通过程序员良好的编程习惯去避免，所以java编译器就没有严格要求处理运行时异常。

# 十六、try-catch-finally异常捕获处理

## 1、try-catch-finally的三种组合方式

* 第一种： 比较适用于有异常要处理，但是没有资源要释放的。

|  |
| --- |
| try{  可能发生异常的代码  }catch(捕获的异常类型 变量名){  处理异常的代码  } |

* 第二种：比较适用于既有异常要处理又要释放资源的代码。

|  |
| --- |
| try{  可能发生异常的代码  }catch(捕获的异常类型 变量名){  处理异常的代码  }finally{  释放资源的代码;  } |

* 第三种： 比较适用于内部抛出的是 运行时异常（RuntimeException），并且有资源要被释放。

|  |
| --- |
| try{  可能发生异常的代码  }finally{  释放资源的代码;  } |

## 2、finally 块

* finally块的使用前提：是必须要存在try块才能使用。
* finally块的代码在任何情况下都会执行的，除了jvm退出的情况。
* finally非常适合做 资源释放的工作，这样子可以保证资源文件在任何情况下都 会被释放。

# 十七、导包

## 1、包（package）

* java中的包就相当于windows文件夹。

### （1）包的作用：

* + 解决类名重复产生冲突的问题。
  + 便于软件版本的发布。

### （2）定义包的格式：

* Package 包名

### （3）包名命名规范：

* 包名全部小写。

### （4）包语句要注意的事项：

* package语句必须位于java文件中中第一个语句。
* 如果一个类加上了包语句，那么该类的完整类名就是: 包名.类名
* 一个java文件只能有一个包语句。

### （5）问题：

* 每次编译的时候都需要自己创建一个文件夹，把对应 的class文件存储 到文件夹中。烦！！！
  + 解决方案：javac -d 指定类文件的存放路径 java源文件

## 2、导包（import）

### （1）导包语句的格式：

* import 包名.类名; （导入xxx包中某个类）

### （2）导包语句要注意的细节：

* 一个java文件中可以出现多句导包语句。
* "\*"是 导包语句的通配符。可以匹配任何 的类名。
* import aa.\*; 是不会作用于aa包下面的子包的。

推荐使用：import 包名.类名; 因为使用\*通配符会导致结构不清晰。

### （3）什么时候使用import语句:

* + - 相互访问的两个类不是在同一个包下面，这时候就需要使用到导包语句。
    - java.lang 是默认导入 的，不需要我们自己导入。

# 十八、权限修饰符

* 权限修饰符： 权限修饰符就是控制被修饰的成员的范围可见性。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | public（公共的） | protected（受保护） | default（缺省） | private（私有） |
| 同类 | true | true | true | true |
| 同包 | true | true | true | false |
| 子父类（继承） | true | true | false | false |
| 不同包 | true | false | false | false |

注意： 在不同包下面只有public 与 protected 可以访问，而且protected必须是在继承关系下才能够访问。

# 十九、jar包

1、打jar包: 需要使用到jdk的开发工具（jar.exe）.

## 2、jar的用法：

（1）使用格式：

* + - jar cvf jar文件的名字 class文件或者是文件夹

（2）打jar包要注意的事项：

* + - 一个程序打完了jar之后 必须要在清单文件上指定入口类： 格式 Main-Class: 包名.类名
    - jar包双击运行仅对于图形化界面的程序起作用，对控制台的程序不起作用。

## 3、jar文件的作用：

* + - 方便用户快速运行一个项目。
    - 提供工具类以jar包的形式给别人使用。

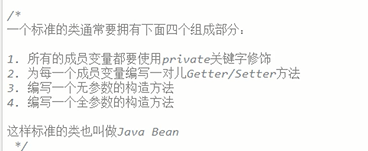
4、如果使用jar包里面的类必须要先设置classpath路径。

5、jre = jvm+ 核心类库

# 二十、定义一个标准的类

一个标准的类通常要用有下面四个组成部分：

1. 所有成员变量都要使用 private关键字修饰
2. 为所有成员变量编写一对 getter/setter方法
3. 编写一个无参构造方法
4. 编写一个全参构造方法



# 二十一、包、权限修饰符、打包

## 1 包机制

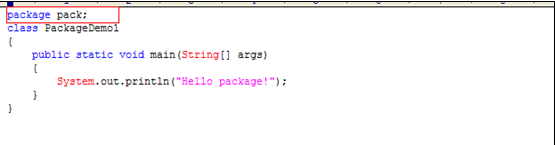
问题：当定义了多个类的时候，可能会发生类名的重复问题。

在java中采用包机制处理开发者定义的类名冲突问题。

怎么使用java的包机制呢？

1. 使用package 关键字。

2. package 包名。



问题：

1. javac PackDemo1.java编译没有问题。

2. java PackDemo1 运行出错。



错误原因分析：

在当前目录下找不到有pack目录，更加找不到pack目录下面的PackageDemo1.java文件。

解决办法：

1. 自己在当前目录下新建一个pack目录。

2. 执行Java pagk.PackageDemo1命令。(包其实就是文件夹).

存在的问题：使用包机制的话，我们是否每次都要自己创建一个文件夹呢？

解决：

在编译的时候则可以指定类文件存放的文件夹了。

javac -d . PackageDemo1.java -d 后面跟着就是包名，指定包存放的路径。

包的优点

1. 防止类文件冲突。

2. 使源文件与类文件分离，便已软件最终发布。

注意细节

1. 一个java类只能定义在一个包中。

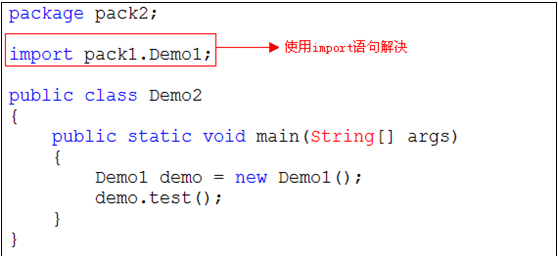
2. 包语句肯定是描述类的第一条语句。

包机制引发的问题

有了包之后访问类每次都需要把包名和类名写全。

解决:使用import语句。

格式： import 包名.类名 ;



### 注意细节：

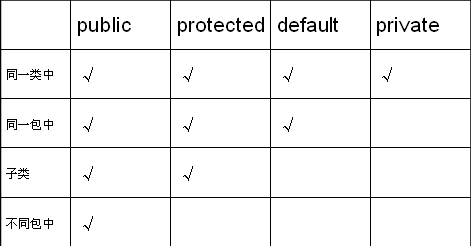
1. 如果想使用一个包中的许多类时，这时不需要多条的导入语句，使用 “\*”号通配符代表所有的类。

2. 使用\*时不能导入包中的子类包的class文件。

3. import语句可以是多条.

## 2 访问修饰符

访问修饰符是用来控制类、属性、方法的可见性的关键字称之为访问修饰符。



1. public 一个类中，同一包中，子类中，不同包中

2. protected 一个类中，同一包中，子类中

3. default 一个类中，同一包中

4. private 一个类中

### 1. (修饰类成员)类成员

1. 成员使用private修饰只在本类中使用。

2. 如果一个成员没有使用任何修饰符，就是default，该成员可以被包中的其他类访问。

3. protected成员被protected修饰可以被包中其他类访问，并且位于不同包中的子类也可以访问。

4. public修饰的成员可以被所有类访问。

### 2. (修饰类)类

1. 类只有两种public和默认(成员内部类可以使用private))

2. 父类不可以是private和protected，子类无法继承

3. public类可以被所有类访问

4. 默认类只能被同一个包中的类访问

## 3 Jar包

### 1：jar就是打包文件

jar文件时一种打包文件java active File,与zip兼容，称之为jar包

开发了很多类，需要将类提供给别人使用，通常以jar包形式提供.当项目写完之后，需要及将class字节码文件打包部署给客户。如何打包？可以使用jar命令.

### 2：jar命令

1：jar工具存放于jdk的bin目录中(jar.exe)

2：jar工具：主要用于对class文件进行打包(压缩)

3：dos中输入jar查看帮助

### 3：案例使用jar命令

问题：将day10中的cn文件打包为名字为test.jar文件(cn 文件是使用javac -d 编译带包的class文件夹)

答：jar cvf test.jar cn

详细命令:

1：jar cf test.jar cn 在当前目录生成test.jar 文件，没有显示执行过程

2：jar cvf test.jar cn 显示打包中的详细信息

3：jar tf test.jar 显示jar文件中包含的所有目录和文件名

4：jar tvf test.jar 显示jar文件中包含的所有目录和文件名大小，创建时间详细信息

5：jar xf test.jar 解压test.jar到当前目录，不显示信息

6：jar xvf test.jar 解压test.jar到当前目录，显示详细信息

7：可以使用WinRaR进行jar解压

8；将两个类文件归档到一个名为 test2.jar 的归档文件中：

jar cvf test2.jar Demo3.class Demo4.class

9：重定向

1：tvf可以查看jar文件内容，jar文件大，包含内容多，dos看不全。

2：查看jdk中的rt.jar 文件 jar tvf rt.jar

3：jar tvf rt.jar>d:\rt.txt

## 4 模板设计.

设计模式就是为了解决某类事情提出的解决方法。

案例：计算一段程序的执行时间



存在问题：

1. 计算的程序的可变的。

2. 把会改变的程序抽取出来单独做一个方法。

3. 但是该方法不能确定运行的代码，声明为抽象的方法。

4. 创建实现类继承并实现父类的未实现的函数。

5. 为了避免子类重写父类的模版代码，需要将模版代码修饰为final

案例二：炒菜做饭

