Java se 是标准版（standard enversion）

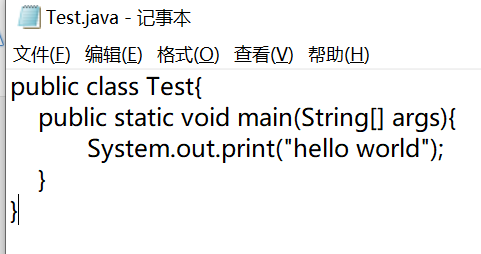
Java ee 是企业版（）

Java两种核心机制：

Java虚拟机（JVM）：程序依赖对应不同系统的虚拟机，可以在各种系统上运行。

垃圾回收（GC）：提供系统级线程自动回收内存（而c系列需要手动回收）

* 安装
* 官网下载jdk并安装；
* 使用压缩版的jdk，根据情况解压不同版本使用解压（推荐，便于不同项目下切换环境）（解压后配置环境变量）
* 配置环境变量
* 在系统变量新建JAVA\_HOME，设置JAVA\_HOME为解压jdk的目录（可能有多个jdk解压文件夹，只需要通过调整JAVA\_HOME的路径就可以更改使用的java jdk版本）
* 在系统变量新建CLASSPATH： .;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar
* Path中新建两个：
* %JAVA\_HOME%\bin
* %JAVA\_HOME%\jre\bin
* 验证是否成功 在命令行输入java-version

1. Jdk jre JVM的关系（）
2. JDK（development kit）： java开发工具包，包含了jre，以及编译工具（javac.exe） 打包工具（jar.exe）
3. JRE（run environment）运行环境，包含JVM和java程序所需的核心库等（运行一个开发好的程序，只需jre）
4. 使用jdk的开发工具完成java程序交给jre运行。
5. 步骤
6. **.java文件(源文件)经过javac.exe编译成.class文件（字节码文件），再使用java.exe运行。**
7. Java源文件以java为扩展名，基本组成是类，执行入口是main方法，有固定书写格式。语句以英文分号结束。
8. 
9. Java基本语法
10. 关键字（均为小写）

class interface enum byte short int long float double char boolean void

true false null

switch case

1. 标识符 自己起名的地方 可以字母 数字 \_ 或$组成。数字不能开头。
2. 数据类型

整型数值型： byte(1字节=8bit -128-127，-2^7 - 2^7-1) short（2字节 -2^15 2^15-1） int（4字节 -2^31） long（8字节 -2^63）

Long类型赋值 时要在值后面跟一个字母l : long l = 98l ;

浮点数值型： float(4字节 -2^128 到 2^128 7位有效数字) double（8字节 -2^1024 到 2^1024 16位有效数字）

Float类型赋值时，值后面跟字母f。float aa = 1.22f ;

字符型：char (用单引号括起来的单个字母 数字或符号) 布尔型：boolean （true或false）

除了以上其他的都是引用类型。

引用类型可以在初始化时赋值为null,其他类型的初始值为0，布尔型初始值为false（但是只声明不赋值，是无法打印的，因为无法编译。但是数组可以不赋初值不赋初值就打印，打印出来是乱码）

String是引用类型 字符串 String stt = “hello” ；

引用类型：类（class） 接口（interface）数组（[ ]）

1. 可变 不可变数据类型

int i1 = 1;

int i2 = 1;

内存产生两个1

String s1 = “hello”;

String s2 = “hello”;

此时内存只有一个“hello”。

其他不可变类型有哪些？单双引号有区别吗（有区别） float可以赋值为整型数吗（可以）

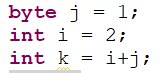
单双引号有区别：单引号只能包裹单个字符，不能包裹字符串；双引号只能用来包裹字符串，单一字符也看作字符串。

1. 基本数据类型转换
2. 自动类型转换：

容量小的类型自动转换为容量大的。Byte char short 可以转为int

int可以继续转换int->long->float->double，例如

容量大的不能转为容量小的，会报错。

1. 多种类型混合运算时，自动将所有数据转为容量最大的类型再计算。
2. Byte char short 不会互相转换；**这三种类型计算时先转换为int型，得到的结果也是int。**（也就是说这三种类型一旦参与了运算，就不再是原类型了，结果是int型（或更大容量的类型））
3. Char类型可以直接参与计算，使用ascii码（规定了128个字符）。
4. 将任何类型的值与字符串类型的值进行连接时（+），基本类型值自动转为字符串类型。
5. 一些练习：

System.out.print(3+4+”aa”) 输出是 7aa

System.out.print(“aa”+3+4) 输出是 aa34

System.out.print(‘a’+1+”aa”) 输出是98aa（’a’的ascii码为97）

System.out.print(”aa”+’a’+1) 输出是 aaa1

String str1 = 1+ 3+2+”a”+2+4 则str1为字符串“6a24”

1. 强制类型转换：
2. 自动类型转换的逆过程，将容量大的转为小的，使用时加上强制转换符（），可能导致溢出或精度降低。
3. 通过基本类型对应的包装类可以把字符串转为基本类型

String a = “34”; int I = Integer.parseInt(a);

1. Boolean型无法转为其他。
2. 运算符
3. 算术

/ 当整数相除时得到的结果也是整数，将小数舍弃。

% 取余数

++ -- 自增,自减， 在数字之前的话则先运算后取值；否则先取值后运算。

+ 可以用于字符串拼接。

字符串与其他类型相加时，还可以把用加号连接的非字符串变字符串。

注意：

1. 对负数取模，符号忽略 如5%-2=1，若被模数为负，不可忽略。此外取模运算结果不一定是整数。
2. Char类型数据可以做数学运算，结果得到int型
3. 赋值
4. =号赋值时的类型转换
5. 连续赋值 i1=i2=i3=0;
6. 扩展赋值运算符：+= -= \*= /= %= ；

使用时会把结果强制转换为当前变量的类型（要知道对于 short byte char 型，一旦参与运算就不再是原类型了，会自动转成int型）

比如 int i =1; i \*= 0.1 结果为0。相乘后的结果是0.1，被强制转为回了int型。

1. 比较运算符
2. 与python相同：== != < > <= >= 返回bool型
3. 逻辑运算符

且 或 异或 非

在python是 and or ? not

& | ^ !

**&&：短路与， ||：短路或。 尽量使用这个**

短路与的意义：只有左边为真时，右边才参与运算。

短路或的意义：只有左边为假时，右边才参与运算。

如果右边有自增 自减符号时，短路与否产生的结果就不同了）

1. 位

与python相同：与& 或| 异或^ 反码~ 左移<< 右移>>

无符号右移：>>> 对于正数来说无影响

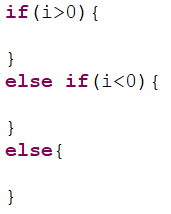
左移<< 空缺位补0.

有符号右移>> 看被移动的数字的二进制最高位，是1移动后就补1 是0移动后就补0

无符号右移>> 空位补0.

1. 三元运算符

Int k = i > 1 ? 1 : 0

1. 运算符优先级
2. 乘除高于取模
3. [] 低于 ()
4. ()优先级最高
5. 程序流程控制
6. 顺序
7. 分支
8. 
9. swith（表达式）{

case 常量1：

语句;

break;

default:

语句;

break;

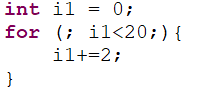
}

表达式的返回值必须是：**byte short char int String 枚举；**

**Case子句值必须是常量且互不相同；**

**Break用来跳出swith，否则顺序执行到swith结尾。**

1. 循环
2. for：注意 写法并不是固定死的。



1. while(条件){循环体}： 与python类似
2. do while
3. 数组

* 一维数组初始化：

int[] arr = new int[3]; 生成一个长为3的数组，这是动态初始化，数组内是有一个初始值的，数字类型的默认值是0，对象类型的默认值是null。

int[] arr = new int[] {1, 2, 3}; 静态初始化，初始化同时还赋值。

数组有一个length属性表长度，**数组初始化后长度不可变。**

**arr.length 表数组arr的长度。很常用**

* 二维数组初始化：

int [][] arr = new int [2][3];

int [][] arr = new int [][]{{1,2,3}，{1,2}，{1,2,3,4}}；数组长度可以不相同。

int [][] arr = new int [2][]; 这个是只制定了第一维的长度，第二维不定义。

**注意：一维数组声明时必须声明长度，二维数组同样，但是二维数组是必须声明第一维的长度，第二维的长度无要求。如果是定义为{},就可以不声明长度，比如：**

**int[] arr = {}; 是没有问题的。**

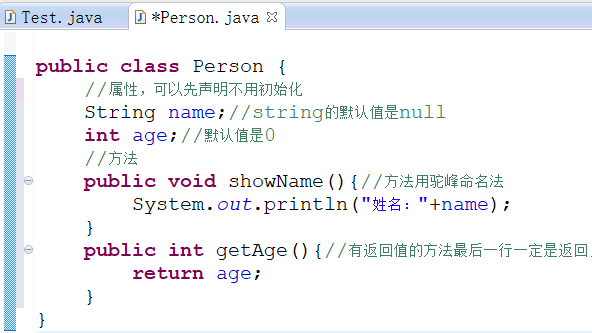
**int[] x,y[];** 表示定义了一个一维数组x，和一个二维数组y。二维数组y里每一个元素都是一个一维数组而不是一个数，x里每个元素都是一个数而不是数组。

* 数组算法
* 最值，总和，平均数：用for循环解决
* 复制：定义空数组，再for循环赋值
* 反转：与复制相同，倒着进行for循环赋值即可。
* 数组是个引用类型的变量，无法直接打印（比python麻烦多了）

1. 面向对象

* 面向对象的三大特征：封装 继承 多态
* 类及类成员

类成员就是属性和方法 public static void main(String[] args){}是函数入口



* 实例化：类名 实例名 = new 类名（）；如Person xiaoming = new Person();
* **类的属性**

修饰符 类型 属性名 = 处值；也可以只声明，不赋初值。

修饰符 private 该属性只能被该类的方法访问（不能通过对象.属性调用），可以被子类继承，但子类无法访问。Public 可以被其他方法访问。

* 变量分类

1. 成员变量：方法体外，类体内定义；在整个类中都可以访问；**有默认初始化值**
2. 实例变量（不以static修饰），这种变量**只能实例化之后才能使用：实例名.实例变量**
3. 类变量（以static修饰），静态，**不需要类实例化成对象就可以用：类名.类变量，实例化后也可以用实例名.类变量调用。**
4. 局部变量：方法体内定义；存在于栈内存中；**没有默认初始化值，**必须显式初始化；不需要权限修饰符；作用范围结束就释放自动变量空间。
5. 形参：传入函数的参数；
6. 方法局部变量：在方法内定义 int i = 0;
7. 代码块局部变量（在代码块中定义的）：代码块就是在类内部，与各方法平级，写一个大括号，里面的就是代码块（怎么用？）

* **不可以在方法内部定义方法（python可以）。同一个类中所有方法可以互相调用，不用实例化对象。**
* **关于默认值问题：**

**布尔型默认值是false 数字类型默认值为0，引用类型默认值为null char类型默认值为‘\u0000’(表示为空)；当一个变量只声明而没有初始化赋值时，就自动使用默认值。**

**因此，当声明一个String而不初始化时，该String为null**

**当声明一个数组但没有标志长度时，该数组为null。**

* 匿名对象：new Student().showInfo(); 对一个对象只进行一次方法调用时可以使用。经常将匿名对象作为实参传递给一个方法调用。

1. 如何像python一样传入参数进行实例化呢

答：利用构造方法

1. 类中的方法可以直接访问成员变量但是有**例外：**Static方法不能访问非static的成员变量!!!只能访问static的成员变量！！！！
2. 方法的重载：

同一个类中允许有多个同名方法，但这些同名方法的参数在数量或类型或顺序上必须有所不同，足以以此来区别同名方法。

1. 什么是形参 什么是实参？

答：形参是方法定义时的参数，实参是方法调用时传入的参数。

* Java中是值传递，
* 对于基本数据类型，进行值传递，将实际参数的值的在内存中复制了一份后传入方法，方法操作的是这个复制值，函数外部的参数本身不受影响。
* 对于引用类型（引用类型存的是对象的地址，数组、String也属于对象），进行值传递，引用类型的值就是对象的地址，所以是将对象地址复制了一份传入函数。函数中操作的是同一个对象，因此对象在函数中改变会受影响的。
* 在python中是引用传递，（引用传递：例如有一个变量a，一个函数f(x),将a传入函数f(x)后，x和a都指向同一个对象，如果用=对x重新赋值，结果只是让x指向了其他对象，a不会受影响；如果x和a指向的对象是可变对象，在函数中对该对象做改变的话，由于x和a指向同一对象，a也就同时被改变了）

在python和java中，字符串都是不可变对象。可以通过下标查看但不可改变。

Java虚拟机中有堆内存 栈内存，栈内存存放变量的值（如果是基本数据类型的变量，直接存放该数据值，如果是引用类型的变量，存放变量的引用对象在堆内存中的地址，因为变量的值就是引用对象的地址）；堆内存存放的是实例化对象。

1. 如何进行平方运算？Math.pow(1,2) = 1
2. 可变参数 ，传入不定数量的参数：

String[] args 这个会在函数内部生成一个名为args的字符串构成的数组；如果没有参数，就必须要传入一个空数组或null。

String… args 同样会生成一个名为args的数组；如果没有传入参数，就可以不填。Java特有的的写法。

注意：两种用法没有区别；也可以直接传入一个包含多个字符串的数组。

问题：只能传入字符串类型的值吗？

答： int… args 表示传入若干个int型变量,其他同理。

在python中，可变参数这样传递 def test( \*nums ): 在函数内部生成一个名为nums的元组。

如果函数接受多个参数，可变参数一定要放在最后。例如 public void test(String s, int i, String… args){}

1. **包（package）和import的问题**
2. **相当于文件夹，包下还可以有包**
3. **.java文件的第一句，指明该文件属于哪个package（缺省则指定为无名包）**
4. **包名小写**
5. **格式 package 包名.子包名.子子包名;创建java文件时会自动添加**
6. **引用 import 包名.子包名.java文件名;(引用某个类) import 包名.子包名.\*;（引用所有类）**
7. **使用同一包下的类不用import。**
8. 一个包package里面有多个java文件，每个类都是写在一个.java文件中，函数执行时会找到某一个类中的 public static void main(String[] args){}方法作为入口运行。

同处于一个package里的类（.java文件）不用导入即可使用。

问题：入口方法可以同时写在多个类（.java文件）中吗？

答：可以，相当于python中的if \_\_name\_\_ == ”\_\_main\_\_”: 。

一个.java文件可以写多个类吗？

答：可以 但只有一个public，其他都是缺省的修饰符,即default。

1. 面向对象特征：封装和隐藏（**private public**）（四个级别的修饰符）

隐藏就是加private，这样的方法或属性只能在自己的类方法中调用。再提供public方法实现对该属性的操作。

* 隐藏不必对外提供的细节
* 使用者只能通过定制好的方法访问数据，可以加入控制逻辑避免不合理操作
* 便于修改，增强代码可维护性。

1. 四种访问权限修饰符

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修饰符 | 类内部 | 同一个包 | 子类 | 任何地方 |
| Private | Yes |  |  |  |
| 缺省 | Yes | Yes |  |  |
| Protected | Yes | Yes | Yes |  |
| Public | Yes | Yes | Yes | Yes |

对于class只有public和default（缺省）两种修饰符，前者表示可以在任何地方被访问，后者表示只允许在同一个包内的类访问。其他包内的类根本无法导入default的class。

1. 类的成员之三：构造器（构造方法）（前两个成员是属性和方法）

（相当于python中的\_\_new\_\_方法，但是构造方法不必返回创建的实例，不必有返回值，反而可以给类属性赋初始值，功能上更像python类中的\_\_init\_\_方法）

New出一个对象时，调用的是class中的构造方法（是一个类方法，称为构造器）

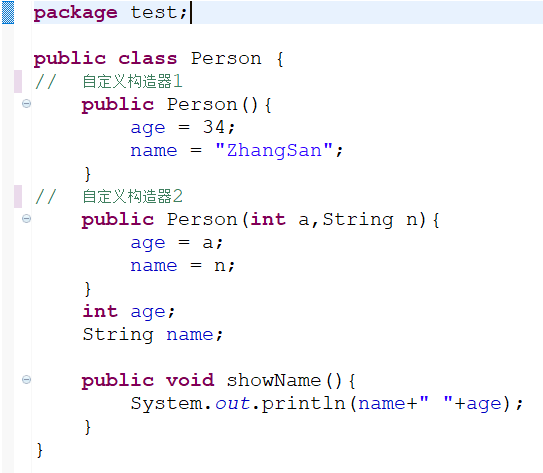
构造器有以下两类：

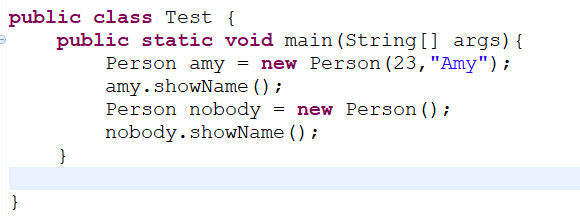
* 隐式无参构造器（系统默认提供）
* 显示定义一个或多个构造器（有参或无参）

注意

* 每个类至少有一个构造器
* 默认构造器的修饰符与所属类的修饰符一致（要么public 要么缺省）
* 一旦显示式定义了构造器，隐式失效（系统不再提供默认构造器）
* 一个类可以创建多个重载构造器
* 父类构造器不可被子类继承

构造器使用举例，和python的init太像了：





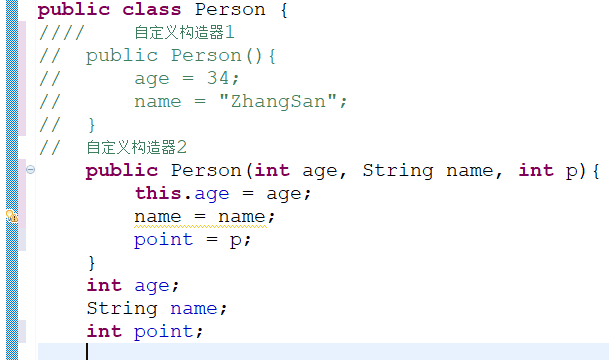
根据传入的参数，会自动选择不同的构造器并执行初始化。（构造器可以**重载**）

1. this关键字
2. 在方法内部使用代表这个方法所属对象的引用
3. 在构造器中使用，代表构造器正在初始化的对象；
4. 通过以上两条可以看出相当于python中的 self。
5. 在java中这个this可以不使用，而python中必须加self指明对象的属性或方法。
6. 使用this可以增加阅读性；可以防止方法的形参与成员变量重名而爆出警告；

this()；代表本类的构造方法。使用this()必须放在构造方法的首行！不能自己调用自己！

1. 何时使用？在方法内部，需要调用该方法的对象时，就用this。为什么

使用例子：



上图纸，this.age 代表当前对象实例的age属性，形参age尽管重名了但是没有影响；point的属性名和形参不重名，也没有影响；但是name的属性和形参重名了，所以有黄色警告。因此，this并不是一定要用。只有在需要区分的时候才需要用this声明。

1. JavaBean

Java语言写的可重用对象。

自动为private属性创建get set方法：右键-Source-Generate Getters and Setters

1. 调用一个包时 import 大包.子包.\*（\*代表这个包下的所有类） 如import package.subpackage.\*

每个包实际上是个文件夹，如果有个包叫java 有个包叫java.test 那么会发现文件管理器中java文件夹下有个test文件夹，但是在IDE里， java包和java.test包是并列、同级显示的。

1. 高级类
2. 面向对象特征之二：继承

格式：class 子类名 extends 父类名{

}

子类含有父类的所有方法和属性（包括private属性和private方法，只是这些属性或方法无法直接访问）。

只支持单继承，不支持多继承。

1. 方法的重写

子类根据需要把父类的方法进行改造。快捷键Alt+/

* 要求：
* 重写方法与被重写方法有同名 同参数类别 同返回类型（也就是**只能改方法体）**
* 重写方法不能使用比被重写方法更严格的访问权限（IDE会标红）
* 二者必须同为static或非static的
* 重写方法抛出的异常不能大于被重写方法抛出的异常

1. 四种访问权限修饰符

注意：**缺省default权限 允许同一个包内访问，子类无法继承（除非与父类在同一个包下）**

子类一定可以继承父类的protected、public属性和方法。

1. Supper

* 用来访问父类的属性和方法（除了private类型的）；在子类构造方法下调用父类的构造方法。
* 子类有与父类同名的成员时，可以用this和super区分。（**可以增强代码可读性，但是有必要区分吗？同名的不都是继承过来的吗**）
* 子类可以调用父类的父类 例子：super.super.xxx
* 调用父类的构造器
* **子类的构造器必须先调用父类的构造器**
* **子类所有构造器默认自动调用父类空参数的构造器（所以，如果父类因为自定义了构造器而没有无参构造器时，子类就需要自己写构造器，并在构造器的首部调用父类的有参构造）**
* 由以上两点可知，若父类没有空参数构造器(可能由于自定义构造器使得无参构造器失效)，子类构造器必须显式的写至少一个构造器，且必须通过**this(参数列表)或者super(参数列表)语句**指定调用本类或父类的相应的构造器，并放在构造器第一行，否则编译出错。（当然，不能本构造器自己调用自己，可以调用本类下的其他构造器）

1. 子类对象实例化过程

栈内存：存基本数据类型变量的值 和引用类型变量的值（对象在堆内存中的地址）

堆内存：存放实例化的对象

方法区：

过程：从方法区加载class（先加载父类后加载子类）；在栈内存开辟空间声明变量；在堆内存开辟空间，存储地址；在堆内存中，在对象空间中，将对象属性初始化为默认值（先父类后子类）；调用构造方法进行初始化，子类构造方法入栈内存执行，父类构造方法入栈执行并出栈；初始化结束后，将堆内存地址赋值给引用变量，子类构造方法出栈。

1. 面向对象特征之三：多态

* 两个体现：方法的重载和重写；对象的多态性：可以直接应用在抽象类和接口上（稍后再讲）。
* Java引用类型的变量有两个类型：编译时类型和运行时类型（编译时类型是由声明该变量时使用的类型决定，运行时类型由实际赋给该变量的对象决定，比如程序声明了一个字符串，编译类型就是字符串，而运行时的赋给变量的是该字符串对象的地址，这就是运行时类型）；编译时类型与运行时类型不一致，就出现多态。
* Person p = new Student(); 父类变量可以指向子类的实例，因为子类可以看作特殊的父类，这叫**向上转型**。，把子类的对象可以给父类的类型变量。

Person p = new Person();

p = new Student(); 这也是向上转型的写法

* 但是：
* 如果一个父类的变量指向子类的对象，这个变量不会访问子类重写后的属性，只会访问父类的。（成员变量不具备多态性）
* 但这个变量会调用子类的重写方法（如果子类重写了父类的方法），而不会使用父类的方法，因为：编译时p为Person类型，而方法调用是在运行时确定的，这叫虚拟方法调用，也叫动态绑定。
* 多态的前提是：要有继承或实现关系；要有覆盖操作。
* 多态总结：Person p = new Student(); 父类变量可以接收子类实例对象，引用变量p**可以访问子类Studennt中的重写方法，但无法访问子类Student中的重写的属性。方法声明时形参为父类类型，可以将子类实例作为实参传入。**
* 多态性举例：方法声明时形参类型为父类类型，调用时可以将子类类型作为实参传递进去（实际上就是把子类看作特殊的父类）
* instanceof

Person p = new Person();

Student s = new Student();

Person e =new Student();

Print(p instanceof Person) True

Print(s instanceof Student) True

Print(s instanceof Person) True

Print(p instanceof Student) False

Print(e instanceof Student) True

1. Object类、包装类

根父类，处于最高处的父类

如果类定义没有extends 默认继承Object类

如果定义方法时的形参只能确定是个类,但不确定将来传入的是什么类，形参可以设置为Object类型。

Object类的一些方法（由于所有类都继承自Object，所以所有类都有Object的方法）

对象.equeals(另一个对象) 返回是否是同一个对象

对象.hashCOde() 返回对象的哈希值

1. 对象类型转换

* 基本数据类型转换：自动或强制
* 对象类型转换：也有自动和强制
* 子类到父类可以自动进行

Student s = new Student();

Person p = s;

* 父类到子类必须强制转换（称为造型）（转换后的s有父类属性也有子类属性）

Person p = new Student(); 只有new出一个子类，将来才能转回子类对象。

Student s = (Student) p;

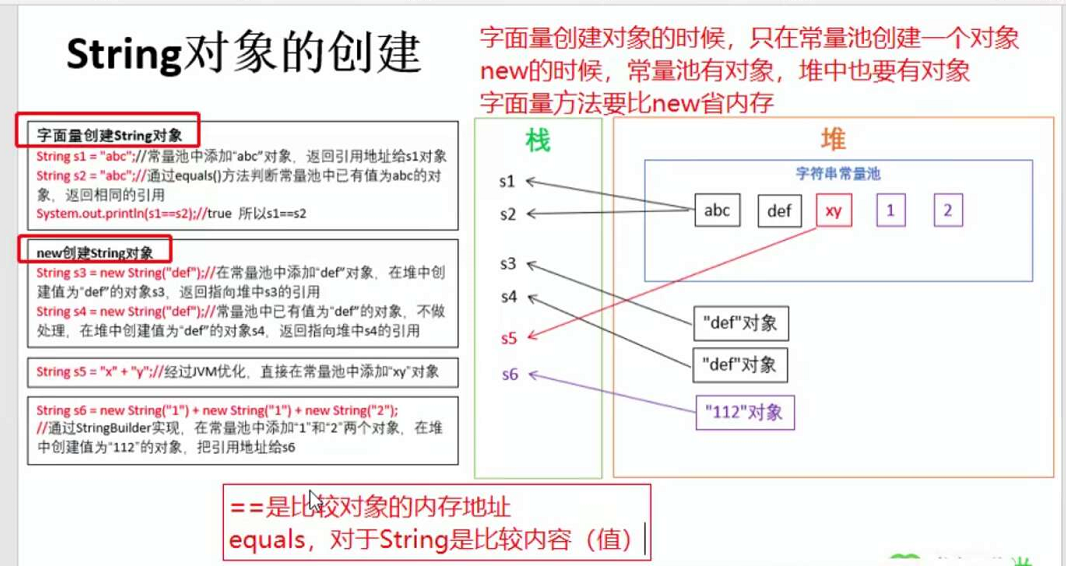
* 无继承关系的无法转换。
* 对象类型转换后里面的属性、方法、重写方法会怎么样呢？
* 只有我们的父类对象本身就是**用子类new出来的时候, 才可以在将来被强制转换为子类对象.**
* 子类继承父类的所有(包括属性以及方法), 也就是说子类包含的内容应该是比父类要大的. 如果正常环境下, 把一个父类转换成子类一定会损失精度(因此不允许强制转型).  但是如果父类对象本身就是用子类去new出来的, 就类似于一个本来就是儿子角色的人, 去装了一会儿父亲, 这个角色是可以在未来转换回去的, 但是一个父亲, 永远不可能去装儿子的.

1. **==和equal**

==**用来比较值是否相等**（int也可以和float比较），对于引用类型，则是比较是否指向同一个对象。

euqals是来自Object类的方法，格式 obj1.equals(obj2) 返回布尔型。 **只能用与引用类型的比较，判断是否指向同一对象**。（若比较的对象不属于同一类则会编译出错，不兼容）

特例：当比较 File 、String、 Date 及包装类（Wrapper Class）来说，equals是比较类型及内容而不是比较是否指向同一个对象。原因：这些类中重写了equals方法。



**字面量创建的字符串，==和equals结果相同，因为相同值的字符串在字符串常量池中一定指向同一个对象。**

**New 两个相同值的包装类（或者new两个相同值的String类），他们用==比较结果是false，因为指向不同的堆内，但equals比较结果是true，因为equals在包装类（String类）被重写了，比较的是指向的具体值。**

**常见用法：判断一个字符串是否为空**

String s = “aa”;

if (s != null && !s.equals(“”) ){

}

类的强制类型转换用法：

Public void method (Object obj){

If (obj instancdof Person){

Person p = (Person) obj;

}else{

}

}

1. 包装类（Wrapper）

针对八中基本数据类型定义的相应的引用类型-包装类

|  |  |
| --- | --- |
| 基本数据类型 | 包装类 |
| **int** | **Integer** |
| **char** | **Character** |
| boolean | Boolean |
| byte | Byte |
| short | 首字母大写 |
| long |  |
| float |  |
| double |  |

用法：

* 基本数据类型包装成实例：装箱

int i = 500; Integer i = new Integer(500); Integer k = new Integer(“123”);

boolean b = new Boolean(“false”).booleanValue();

Float f = new Float(“4.56”); 可以将字符串参数构造成包装类对象,**如果字符串参数不是数字，编译不报错，但运行报错。**

* 获得包装类对象中的基本数据类型：拆箱

调用包装类的 **.xxxValue()**

**例如：boolean b = bObj.booleanValue();**

* **JDK1.5之后支持自动装箱拆箱，但类型必须匹配。**

**Integer k = 112; 自动装箱 Boolean bobj = true;**

**int j = k; 自动拆箱**

**包装类的主要用途**

* **字符串转换数据类型转换：#############**

int i = Interger.parseInt(“123”);

float f = Float.parseFloat(“0.2”);

boolean b = Boolean.parseBoolean(“true”);

* **数据类型转字符串：##################**

String istr = String.valueOf(i);

String fstr = String.valueOf(f);

String bstr = String.valueOf(b);

1. toString()方法

Object类有个这个方法，输出当前对象的内存地址（返回一个字符串，可以直接print），如果需要输出其他信息就重写这个方法。

这个相当于python中的\_\_str\_\_

直接print一个对象，输出的和toString方法返回值一样，也是对象的内存地址。

重写：

public String toString(){

String str = “aaaa”;

return str;

}

1. static关键字  **做工具类用的最多**

比如写一个Chinese类，里面有个属性country，对于任何实例来说，这个属性的值都是String country = “China”；那就可以将该属性加static，即static String country；初始化时直接写：Chinese.country = “China”;即可，不必实例化就可以初始化该值。访问时也是通过 **类名.属性** 来访问，用不到实例对象名，被所有该类的实例化对象所共享。

不加static的是实例变量，只有实例化后才能使用，是某个实例所私有的，调用时也要：**实例名.属性**。

也就是说，如果让一个类的所有实例共享一个数据，就用static. 同样的，方法也可以用static修饰，调用时可以直接：**类名.方法**。

当然，也可以通过 实例对象名.属性、实例对象名.方法 调用static修饰的。

**被修饰成员特点：**

* 优先于对象存在
* 随着类加载而加载
* 修饰成员被所有实例对象共享；（可以用来计算new出来的该类的实例个数，用static修饰一个count类变量，在构造方法中将该类变量加1，）
* 访问权限（private、缺省、protected、public（常用））允许时，可不创建对象，直接被类调用。

**注意点：**

1. **因为static方法不用实例化就能调用，因此static方法内部不能有this、super。也就是不能涉及实例变量**
2. **重载的方法必须同时为static或非static。**
3. 设计模式

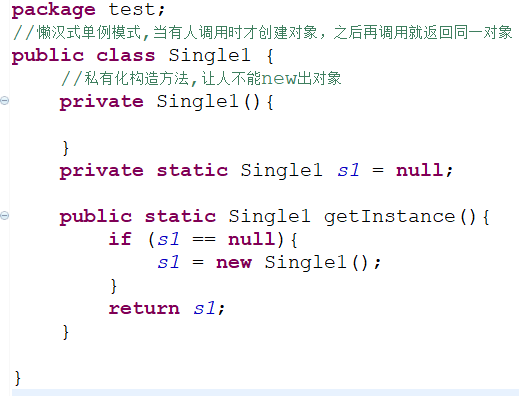
设计模式就是在实际编程过程中主键总结出的解决问题的套路。

单例模式：只有一个实例

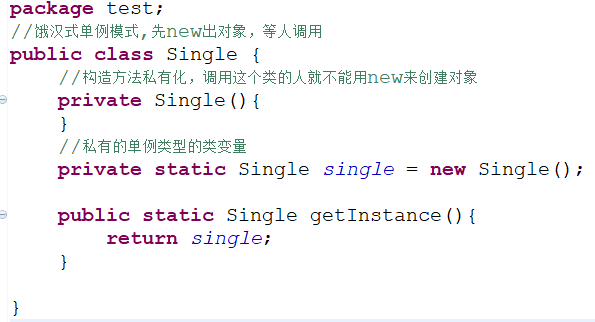
在整个软件系统运行中这个类只被实例化一次，只能调用这一个实例。

适用场景： 实例的创建要消耗大量时间和资源时。（比如 构造方法要执行很多代码，耗时很长）

懒汉式单例设计模式



饿汉式单例模式：



**当new一个对象时，实际上在调用该对象的构造方法，如果构造方法私有化，就无法new出对象。**

1. 再谈main方法

Java虚拟机需要调用类的main方法，因此是public的，由于不需要创建对象就调用，因此是static的，该方法接受一个String类型的数组参数，该数组中保存执行java命令时所有传递给该运行类的参数。

1. 类的成员之四：**初始化块（前三个是属性 方法 构造器）**

作用就是**先于构造方法的初始化。**

* New一个对象时，先执行类属性的默认初始化和显式初始化，后执行static静态代码块的代码，后执行无static的非静态代码块，后执行构造方法。
* 有多个代码块就按程序的顺序执行。
* static修饰的静态代码块里只能使用static修饰的类属性和类方法。
* **Static修饰的初始化块只在第一次new对象时执行一次，而其他非static修饰的代码块和构造方法都是new一次就执行一次。**
* 在实际开发中，static代码块用的多，用于初始化类的static属性。
* 代码块可以用来进行匿名内部类的初始化（由于是匿名内部类，无法调用构造方法，只能通过代码块初始化）所以，java现存的这些东西都是有作用的，而且在某个方向上有不可替代的作用。

1. 关键字final：修饰类、属性、方法

修饰的类无法被继承；

修饰的方法无法被重写；

修饰的变量（成员变量或局部变量）为常量，名称大写（如果多个单词构成名称，用下划线连接），只赋值一次，不能被改变

* Final标记的成员变量必须在声明时或在每个构造方法或代码块中显式的赋值，然后才能使用
* 例如：final double PI = 3.14;

1. 抽象类（abstract class）

* 子类更具体，父类更通用。有时将一个类设计的很抽象以至于没有具体的实例，这就是抽象类。

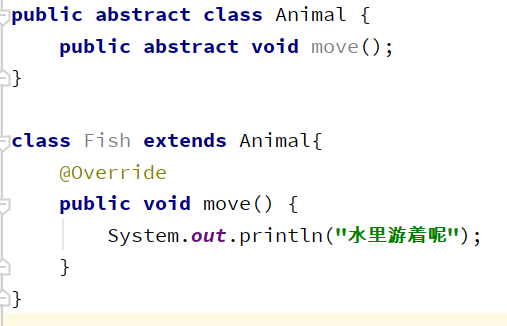
比如：动物是抽象类，狗、鱼、鸟都是更具体的类。动物类中有个move方法，但这个方法没法在动物类中去写，所以动物类和它的方法都是抽象化处理：不写具体的实现（不写方法体），当被继承时才实现方法体。

* 实现： 用abstract修饰一个类，称为抽象类；public abstract class Animal{}

用abstract修饰一个方法，称为抽象方法，没有实现，以分号结束：例如

abstract int abMethod(int a);

* 含有抽象方法的必须被声明抽象类
* 抽象类不能实例化，只能被继承，子类必须重写所有抽象方法，否则仍是抽象类。
* Abstract不能修饰属性、私有方法、构造器、静态方法、final方法。



抽象类可以有构造器，但是抽象类不能实例化（new一个抽象类是非法的），没法直接使用构造器，只能在继承的子类里使用构造器。

1. 抽象类有个说法：模板方法设计模式（TemplateMethod）

* 抽象类就像一个大纲，每个抽象方法就是每个章节的标题；子类根据标题将每个章节写出来。
* 相当于每个子类的通用模板
* 解决的问题：
* 当功能一部分实现是确定，一部分不确定，可以把不确定的部分暴露出去让子类去实现
* 编写一个抽象父类，提供多个子类的通用方法，并把方法留给子类实现，就是一种模板模式。

1. 接口（Interface）

Java不支持多重继承，使用接口达到多重继承的效果（一个子类同时得到多个父类的属性和方法）

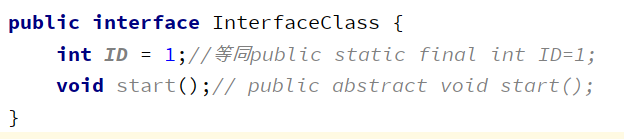
本质上是一种特殊抽象类，只有常量和方法的定义，没有变量和方法的实现。

一个类可以实现（implements）多个接口，一个接口可以继承（extends）多个其他接口。

特点：

* 用interface定义
* 所有成员变量默认是public static final修饰
* 所有方法默认由public anstract修饰
* 接口没有构造器
* 接口采用多层继承机制

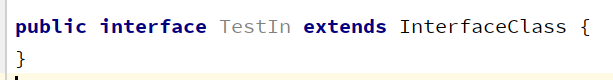
**定义接口：**



**继承多个接口：**



**接口继承接口：**



**一个类实现了接口之后，要么实现所有的接口方法，否则需要定义为抽象类。**

如果一个类既有继承extends又有实现implements，那么先写extends关键字再写implements关键字。

1. 接口和抽象类很相似，好像接口能做的用抽象类也能做，那么**接口的作用何在？**

抽象类新增抽象方法带来的问题：当抽象类新增了一个抽象方法，子类就必须实现抽象父类的抽象方法，否则必须改为抽象类。

实际应用中，会把新增的抽象方法放到一个接口里，子类如果需要就去实现这个接口，不需要就不必管这个接口。

举例：人类由猴子进化来，如果穿越过去改变了猴子，那么对现在的人类会造成什么影响？人类的生活习惯就会很大改变。所以**父类需要稳定的抽象**。如果父类一直在变，那么子类和子类的子类都需要维护，**有时又需要给父类增加一些方法，而不能直接改变父类，就需要新建接口**，在接口上扩展方法，由子类自行决定是否实现接口。

**举例2：用java描述会唱歌的厨师是个老师。如果用继承，需要Singer继承Person，Cooker继承Singer，Teacher继承Cooker，但这种继承没有逻辑关系，Cooker继承自Singer实际上污染了Cooker类，因为并不是所有Cooker都是Singer。**

**正确的做法是：Teacher继承Person，再进一步实现Singing接口和Cooking接口。表示这个Teacher有Sing和Cook技能。**

**可以用接口去接收一个实现了该接口的实例，但该实例只能调用本接口的重写方法，无法调用其他接口的方法和继承的抽象类的方法，这体现了对象的多态：**

**Cooking c = new SCTeacher();**

**c.fry();**

**Singing s = new SCTeacher();**

**s.sing();**

**接口可以用extends关键字继承另一个接口。实现类必须实现接口和所继承抽象类的所有方法，否则必须设为抽象类。**

1. 抽象类和接口的**总结**

抽象类是对于一类事物的高度抽象，其中既有属性也有方法。

接口是对方法的抽象，也就是对一系列动作的抽象。

当需要对一类事物抽象时使用抽象类，好形成一个父类。

当需要对一系列动作进行抽象，使用接口，需要使用这些动作的类去实现相应的接口即可。

1. 工厂方法（工厂模式）我没理解 <https://www.runoob.com/design-pattern/factory-pattern.html>

举例：

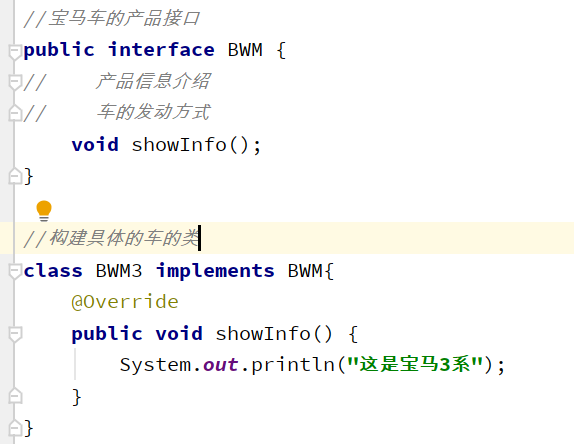
真正开发都是合作开发，每人写一部分代码，集合在一起就是一个项目。

开发人A写类，开发人B实例化使用这个类，产生的问题就是A改代码的话，B也要跟着改。

解决方法就是，A写一个产品接口，让不同的产品类去implements这个产品接口；写一个产品工厂接口，让不同的产品的产品工厂类去implements这个产品工厂接口。

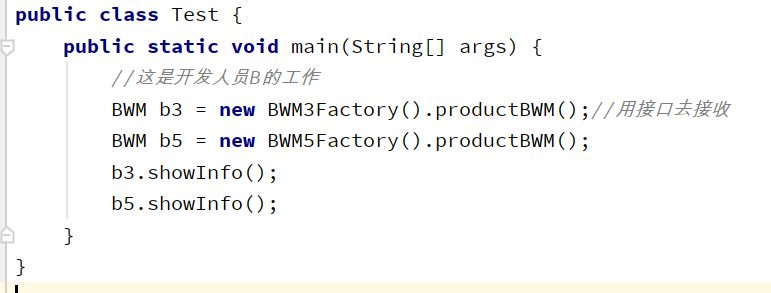
通过工厂把new对象给隔离，通过产品的接口可以接受不同实际产品的实现类。这样A的代码如果类名改变，不会影响B的代码。

A的代码如下：





B的代码如下：

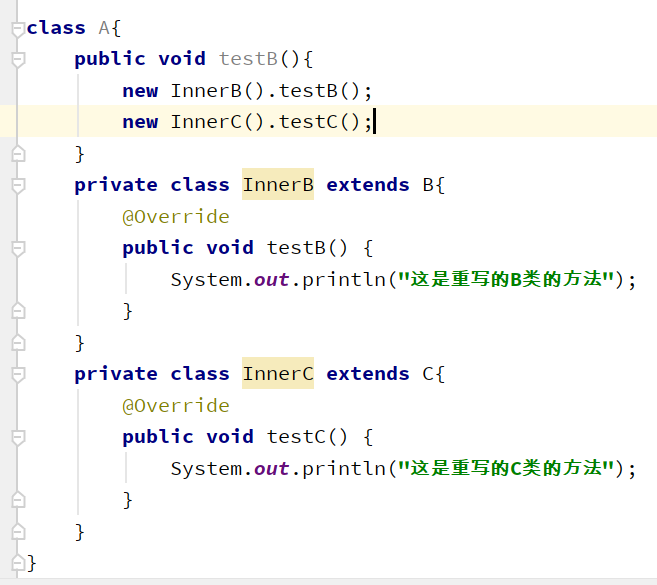


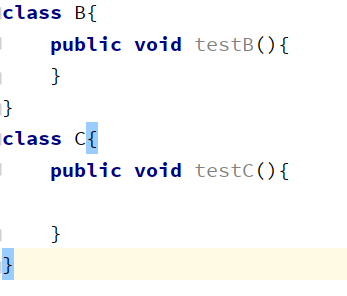
1. 类的成员之五：**内部类（属性，方法，构造器，代码块是其他四个成员）**

* 一个类的内部定义一个类就是内部类，不能与包含他的类名相同。
* 内部类可以使用外部类的私有数据，因为他是外部类的成员（同一类的成员之间可以互相访问），而外部类要访问内部类成员则需要：内部类.成员 或者 内部类对象.成员
* 内部类的分类
* 成员内部类（static成员内部类和非static成员内部类）（写在类里面的）
* 局部内部类（不谈修饰符）、匿名内部类（写在方法里面的）
* 内部类特性
* 可以声明为final
* 可以定义多个内部类
* 与外部类不同的是，可以声明为private和protected、可以声明为static（看作外部类的成员）
* 如果内部类是static就不能使用外部类的非static成员。
* 可以声明为abstract，可以被其他内部类继承
* 问题：**内部类有什么用？**

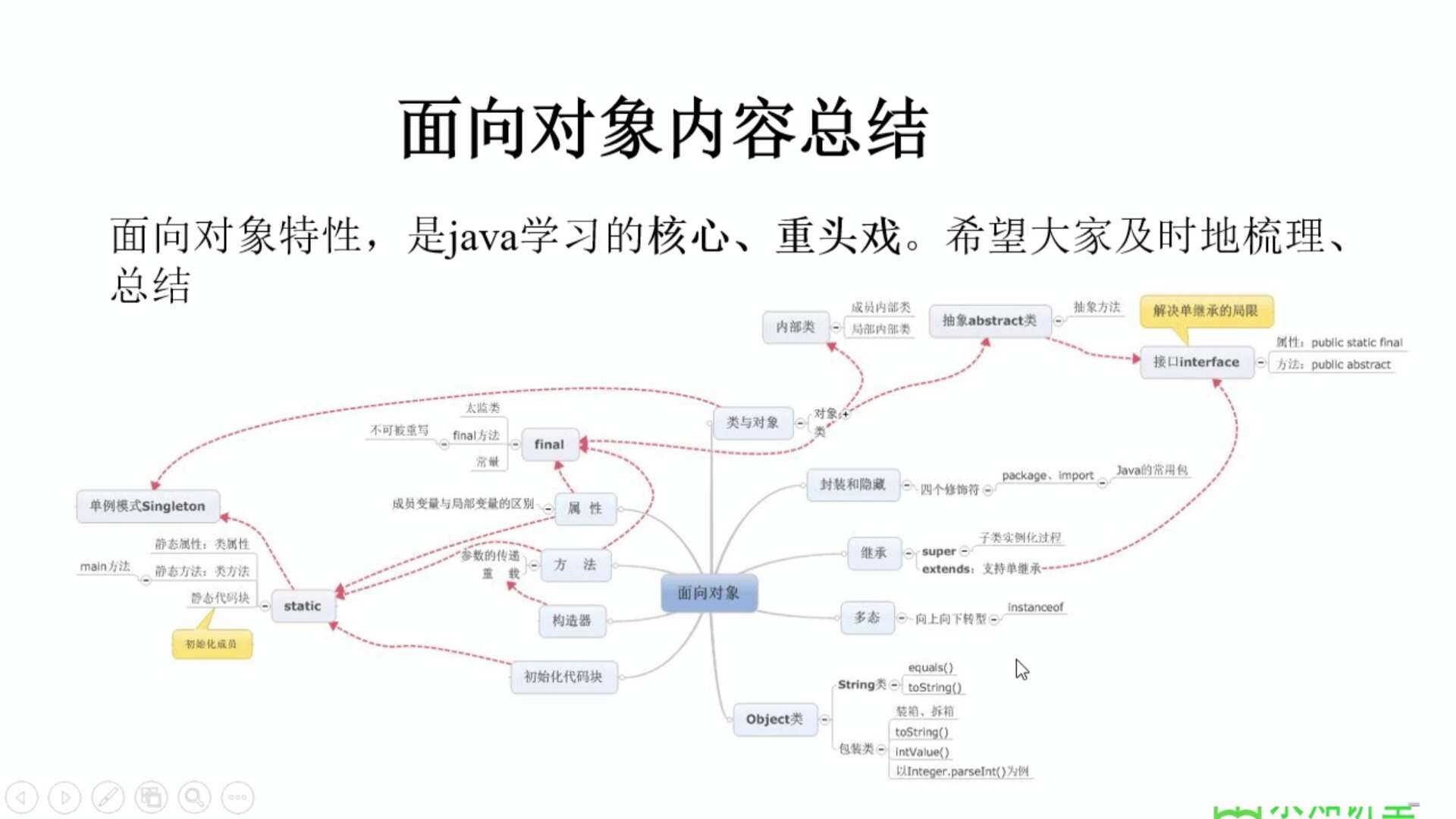
**解决java不能多重继承的问题**

**如下图所示，写多个内部类，让他们分别继承不同的类并重写方法，外部类可以通过内部类变相的使用类的多重继承，可以同时继承多个类。**





1. 面向对象总结



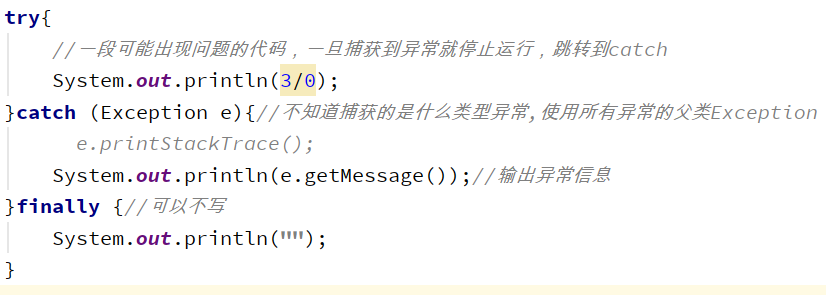
1. **面向对象已经结束，接下来高级内容**
2. 异常处理

常有：数组下标越界异常 空指针异常

Person p = null;

System.out.print(p.name); 这就会出现空指针异常。

* 异常
* Error
* 常见的有堆栈溢出、内存溢出
* Exception
* IOException
* RuntimeException（常见）

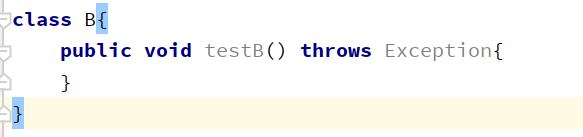


跟python很像

**注意：可以写多个catch分别用来捕获不同类型的异常，但是try代码块中如果有多个异常，只会捕获第一个出现的异常，然后跳转到对应类型的catch继续执行。**

**抛出异常：throws，**

* **一个方法的定义上加throws Exception(如下图)，可以将本方法中捕获的异常抛给调用该方法的上一层代码块。也可以在main方法上抛出异常，但是是没有人处理的，是直接抛到虚拟机上。**



**注意：子类继承父类，父类有个方法会抛出异常，那么子类重写该方法也要抛出异常，且子类方法抛出的异常类型范围不能大于父类。**

* **throws可以直接写在代码里，作为语句使用，比如：throws new Exception(“年龄不对！”);**
* **创建用户自定义异常类**

**Java提供的异常类一般够用，只有特殊情况需要自己写异常类，很少见。**

* **Try catch代码块中finally和catch至少有一个。**
* **Finally中的代码一定会执行，即使try catch里有return，也会在return之前执行，如果finally里也有return，那么会覆盖其他的return结果。**

1. **集合##################**

集合存放于java.util包中，用来存放对象的容器

* 集合只能存对象，存基本数据类型会自动转成相应对象
* 存放的是对象的引用，对象还在堆内存中
* 可以存放不同类型，不限数量的数据类型。

可分为Set List Map三大体系

* Set 无序 不可重复的集合
* List有序 可重复的集合
* Map 具有映射关系的集合

Jdk5之后增加了泛型，java集合可以记住容器中对象的数据类型

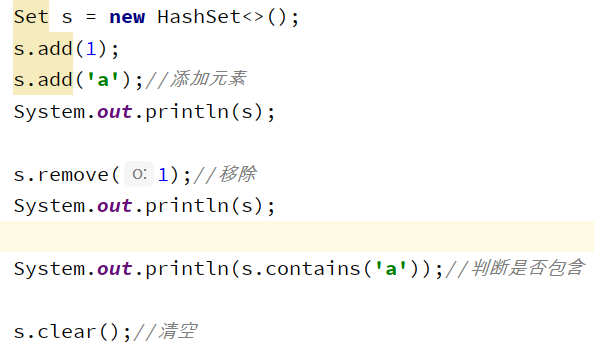
HashSet

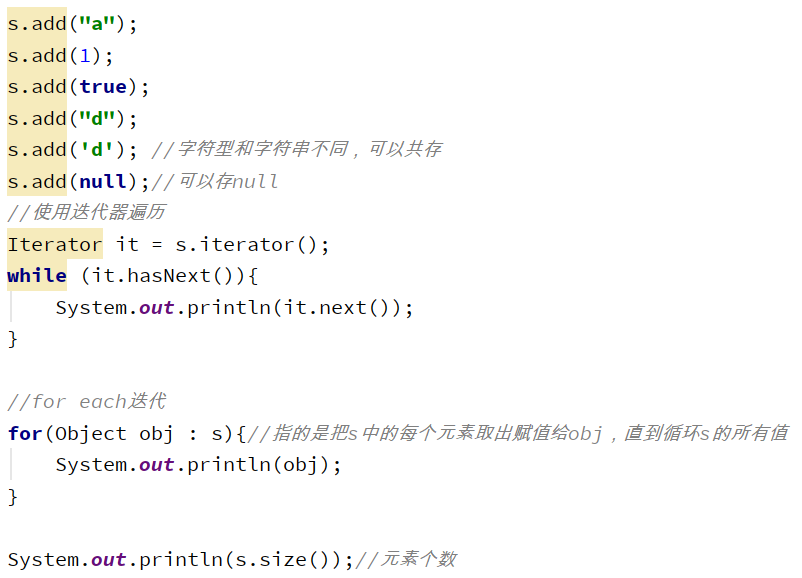
* 无序： 会调用对象的hashCode()方法得到对象的hashCode值进而决定对象的存储位置
* 不可重复：equals是否为true不重要，指的是hashCode不相同（但是一般情况下equals与hashCode是同源的）
* 线程不安全
* 集合元素可以存null

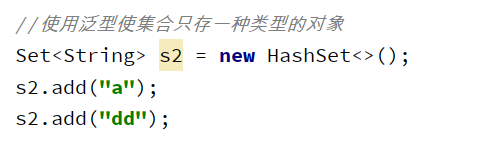
HashSet实现了Set接口，Set接口继承Collection接口

HashSet中不允许出现重复元素是指，重复元素是指equals相同 且hashCode也相同的对象（理论上二者是同源的）

**HashSet使用方法：**





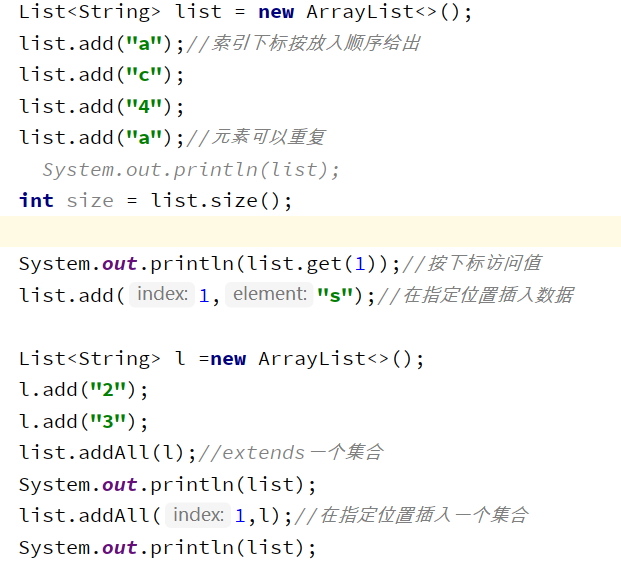


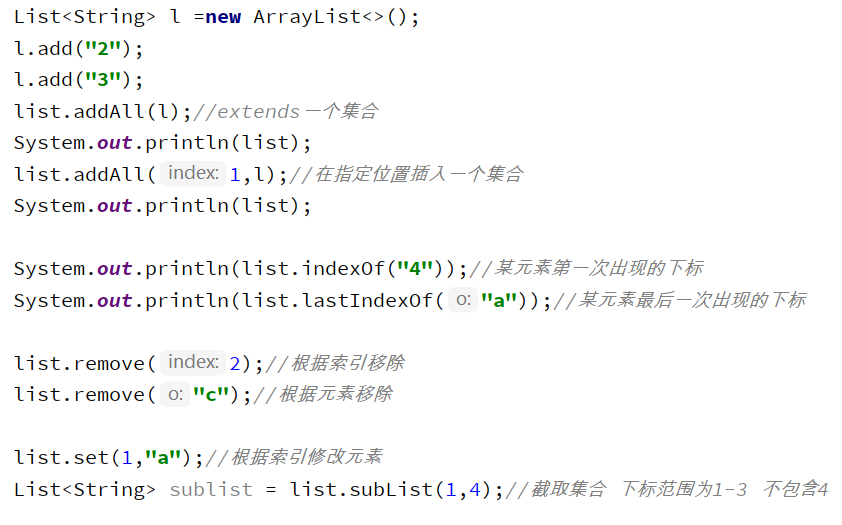
1. TreeSet



1. ArrayList

ArrayList和Vector是List接口的两个实现，推荐ArrayLitst，尽管ArrayList是线程不安全的，而Vector是线程安全的。二者用法基本一样，因为继承同一个接口。





1. Map

Map里有key和value，都可以是任意类型，key不许重复。Map是接口，HashMap是实现类。

Map<String, Integer> m = new HashMap<>();

m.put(key, value);

m.remove(key);

m.get(key);

m.size()

m.containsKey(key);

m.containsValue(value);

遍历：

Set <String> keys = m.keySet();获取集合的Key的集合

m.values(); value的集合

for ( String key : keys){

System.out.println(“key:”+key +”,value:” + m.get(key) );

}

另一种遍历方式：

Set< Entry<String, Integer>> entrys = m.entrySet();

For (Entry<String, Integer> en : entrys){

System.out.println(“key:”+en.getKey() + “value:” + en.getValue());

}

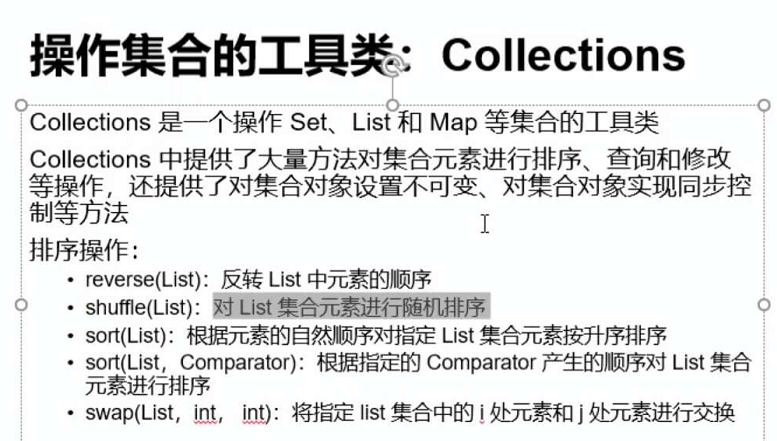
HashMap和HashTable都是Map接口的典型实现

HashTable古老的Map实现类，不建议使用，但是线程安全，（HashMap线程不安全）不允许null作为key和value，但HashMap可以。

二者判断key是否相等，必须equals为True且hashCode相同

TreeMap与TreeSet差不多，可以保证key是有序的。

1. Collections



其他操作：

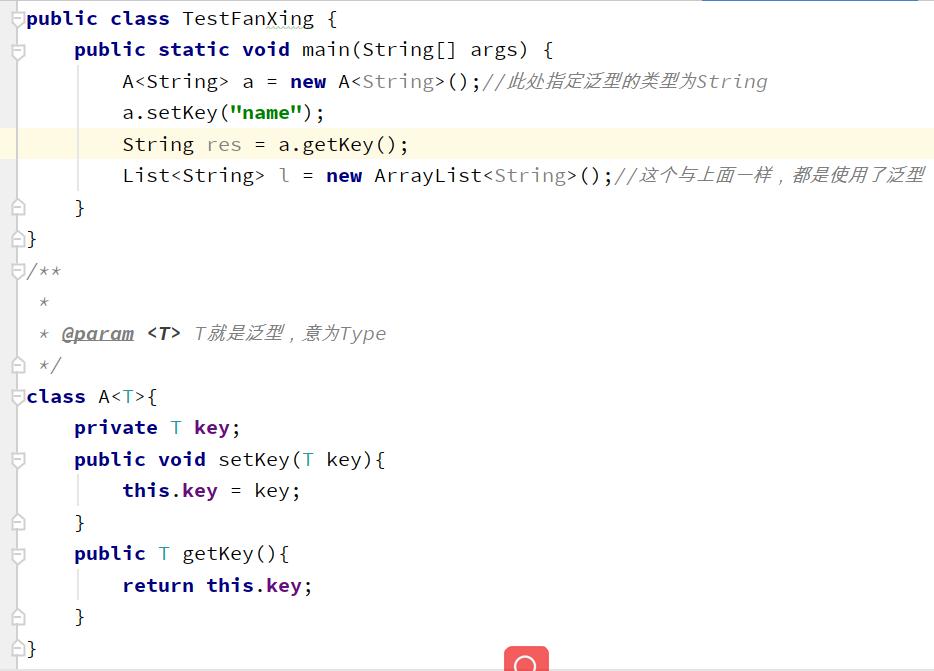
max(List); min(List); frequency（集合,元素);返回集合中指定元素的出现次数

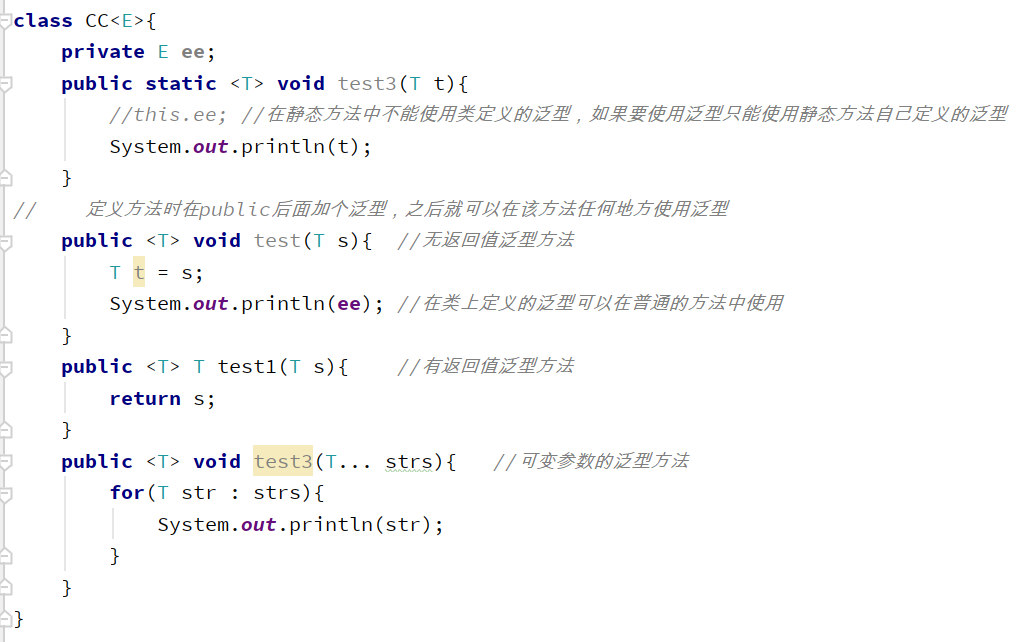
repleaceAll(集合，旧值，新值); 使用新值替换集合中所有旧值。



1. 泛型
2. 不使用泛型的话，就可以往集合里放各种数据类型
3. Java中的泛型只在编译过程生效：编译过程在正确检验泛型结果后，会将泛型的相关信息擦出，并且在对象进入和离开方法的边界处添加类型检查和类型转换的方法，也就是说，泛型信息不会进入运行阶段。

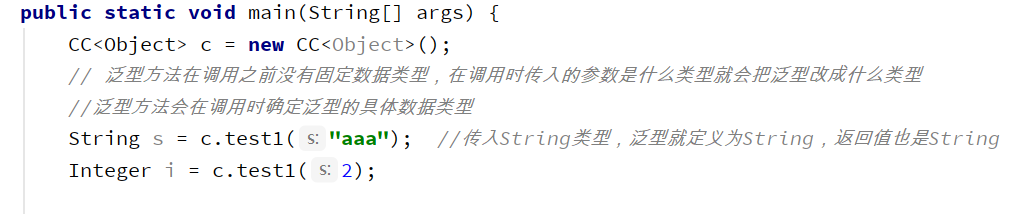
泛型的使用：

1. 泛型类
2. 
3. 如果在创建一个泛型类的实例时没有指明泛型，那么就默认指明泛型为object类型。
4. 同样的类，如果在new对象时指明的泛型不一样，那么这些对象的引用类型变量不能互相赋值。
5. 泛型方法

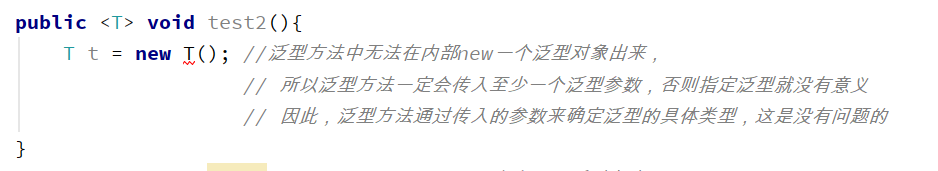


可以看出，泛型方法在定义时，前面的<T>在调用时用不到，它的作用只是用来说明形参中的T不是一个具体的类名，而是一个泛型。

泛型方法的调用：

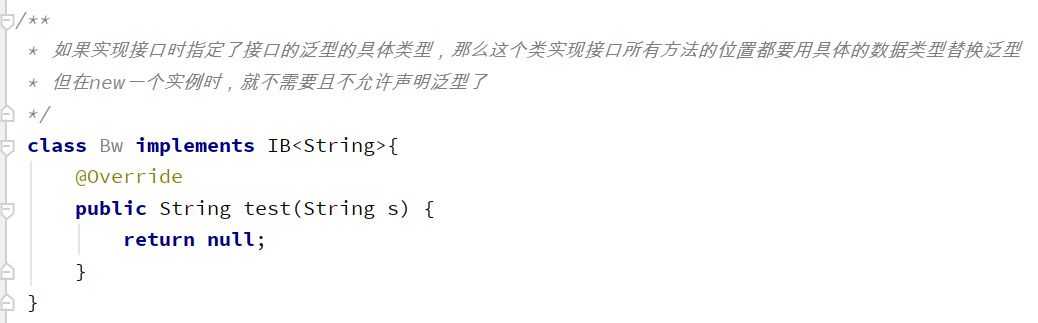


泛型方法如果是无参的，如何确定泛型的具体类型呢？



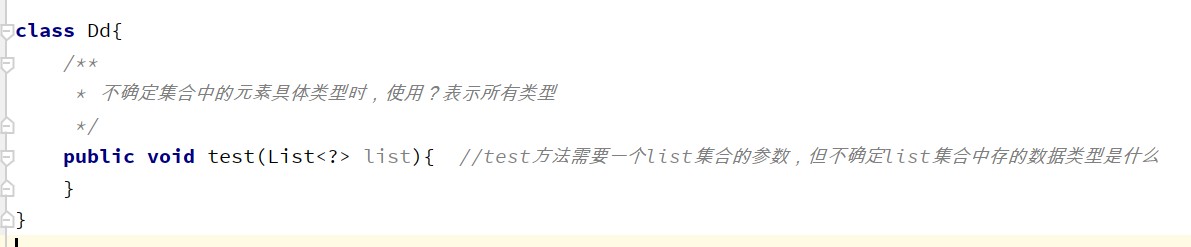
1. 泛型接口





1. 泛型通配符

什么是通配符，通配符就是一个问号，通配符用于方法的形参上。方法需要一个集合做参数，但不确定集合的泛型是什么时使用通配符。



有限制的通配符

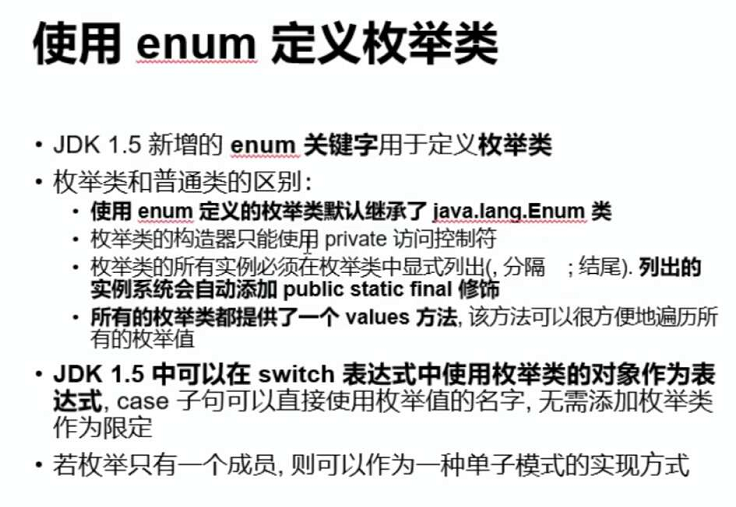


1. 枚举类

**某些类只有有限的实例，比如四季类，只有春夏秋冬四个实例。如果用单利模式实现，如下：**



**用枚举类实现：**



**代码示例：**



**枚举类常用方法：**

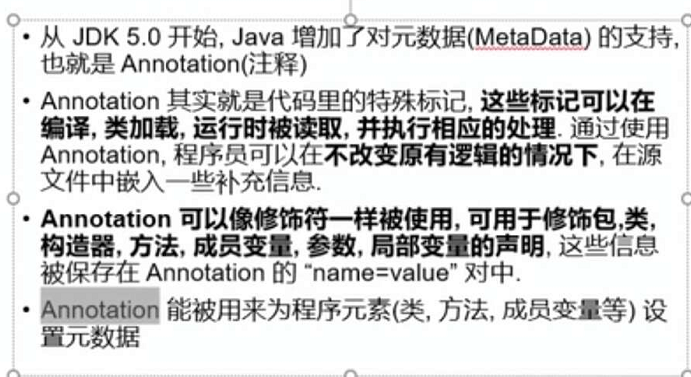
**compareTo比较常用。**



1. 注解
2. 概念：说明程序的，但它是给计算机看的。类似于给人看的注释。
3. 定义：也叫元数据，一种代码级别的说明，**1.5**之后提供的新特性，与类，接口，枚举是同一层次，。。。。
4. 概念描述：

1）1.5 之后出现

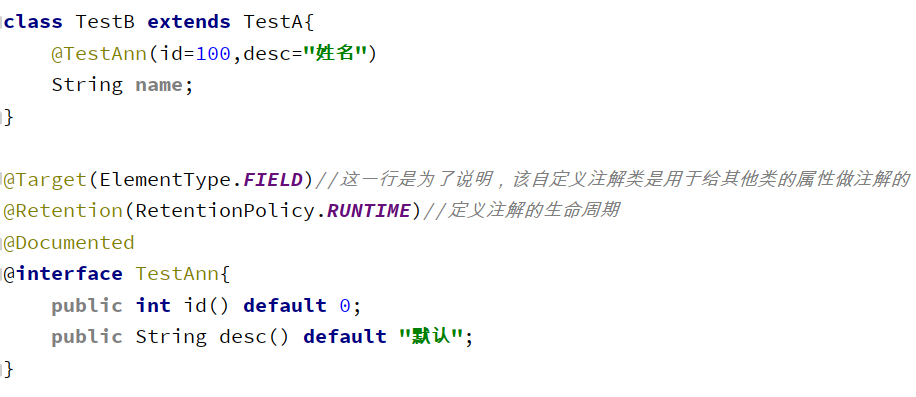
2） 说明程序的



1. 自定义注解



代码演示：



1. IO流

主要内容：

1. Java.io.File类的使用
2. IO原理及流的分类
3. 文件流(操作文件)： FileInputStream / FileOutputStream / FileReader / FileWriter /
4. 缓冲流(操作内存)： BufferedInputStream / BuffferedOutputStream / BufferedReader / BufferedWriter
5. 线程

几核CPU即代表能同一时间处理的任务数。

进程是电脑中正在运行的一个程序比如QQ、网易云。

CPU主频：看起来电脑运行了好多软件，但实际上CPU的多核在进程间频繁切换，而人感觉不到。（核数和主频越高越好）

一个进程可以同时执行多个线程。

何时需要多线程？

1程序需要同时执行多个任务。

2程序需要实现一些需要等待的任务时，如用户输入，文件读写，网络操作。

3需要一些后台运行的程序时。

浏览器进程在等待服务器响应时，可以让这个进程占用的CPU干点别的事，等响应来了再继续使用。

1. 多线程创建与启动

通过Java.long.Thread类实现

常用方法：这个类的run()方法是线程体，start()方法调用线程。

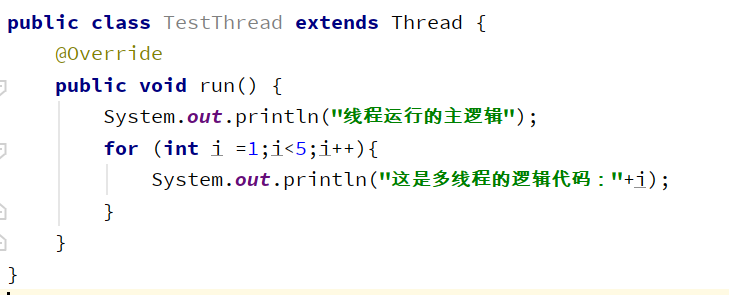
* 方式1

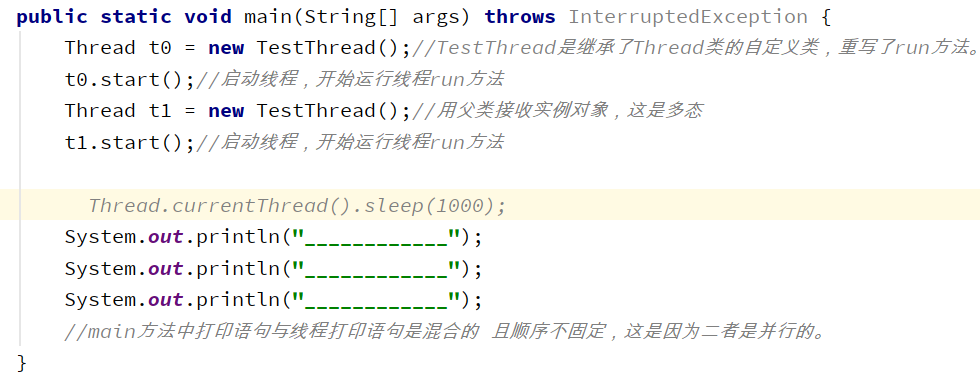
定义子类继承Thread类

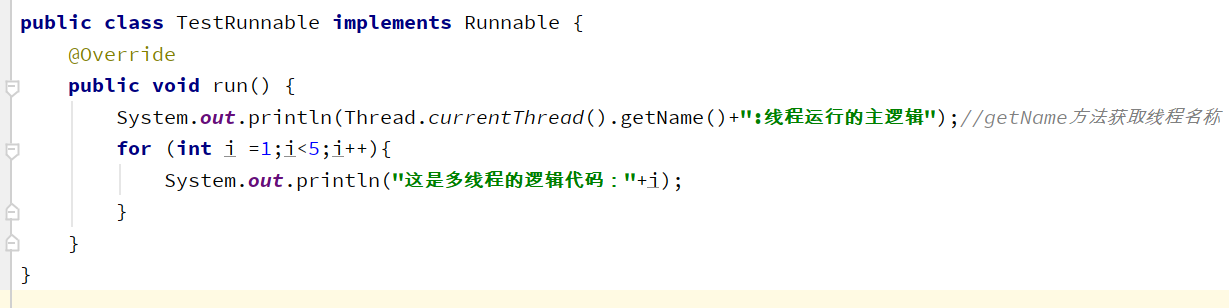
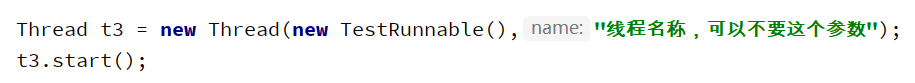
重写run方法

创建对象

调用线程对象的start方法，启动线程。（之后线程的run方法和主程序main之后的代码共同运行）





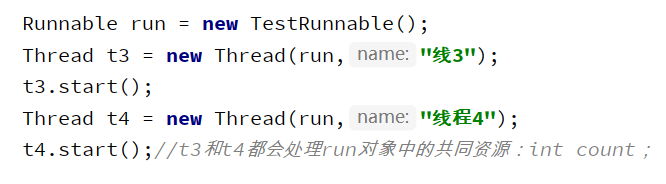
* 实现Runnable接口
* 继承方式与实现方式的联系与区别

二者的代码都放在子类的run方法中。一个是重写run方法，一个是实现run方法。

**一般用实现接口的方式，原因：**

* 避免单继承的局限性
* 多线程可以共享同一个接口实现类的对象，非常适合多个相同线程处理同一资源。





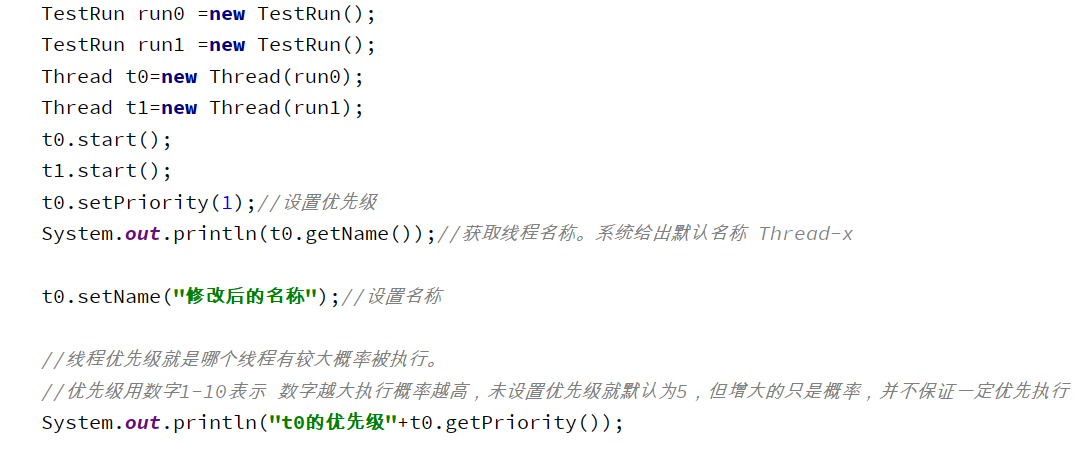
* 多线程优点

提高应用程序的响应

提高CPU利用率（一个线程遇到IO操作等待时可以将CPU给其他线程使用）

改善程序结构。（将没有关联的代码写进不同线程并行执行）

1. Thread常用方法



1. 其他常用方法

static void yield(); 线程让步让给其他线程优先执行，不是很明显，如果没有其他线程，该方法无效。

join(); 相当于把run方法放到join方法位置处执行，而且是同步执行：执行run方法前，join前面的代码必须先执行，执行run方法后，join后面的语句才会继续执行。

static void sleep(毫秒); 线程睡眠；

stop(); 强制结束

boolean isAlive(); 判断是否存活

1. 线程生命周期

线程从生到死的

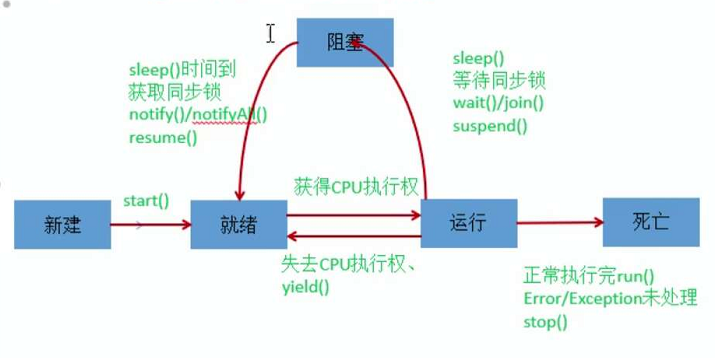
新建：new之后

就绪：执行start方法后，获得cpu执行权之前

运行：run方法代码开始执行

阻塞：类似堵车，run方法代码暂停执行，比如IO操作

死亡：完成了全部工作或被强制终止（执行stop方法、断电、杀掉进程）

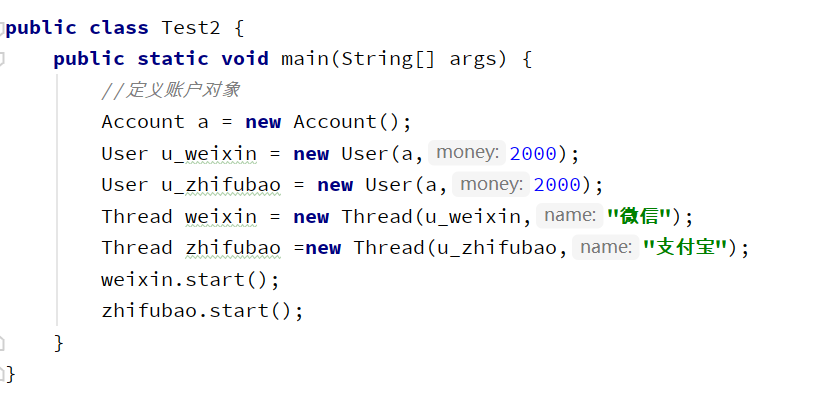


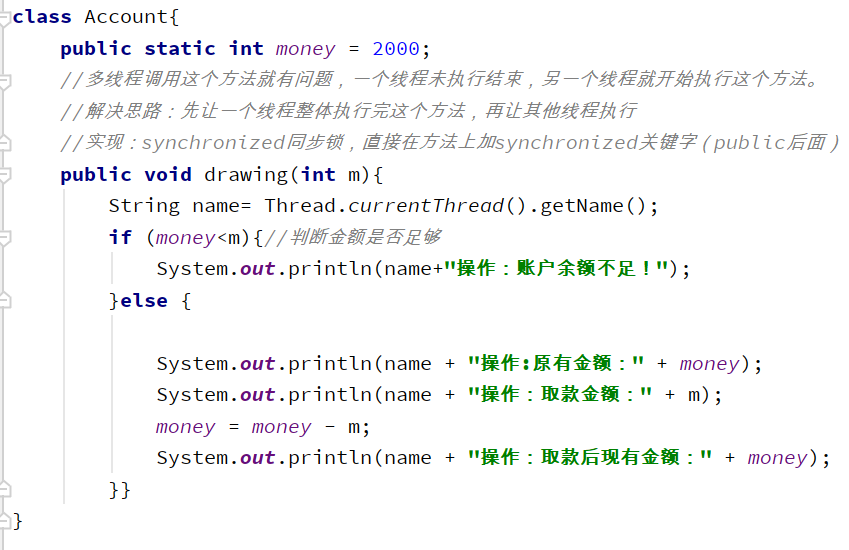
1. 线程同步

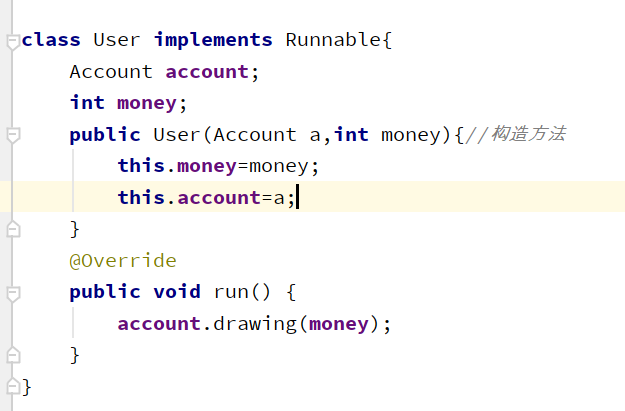
问题：多线程执行的不确定性引起执行结果的不稳定；多个线程对账本的共享，造成操作的不完整，会破坏数据。（比如两个线程同时取款）

原因：一个线程未执行完时，另一个线程也参与执行，导致共享数据错误。

解决：一个线程先执行完再让另一个线程执行。







1. 加synchronized关键字解决线程同步

解决线程安全问题：同步机制

1. 在一个方法上加synchronized关键字，可以让多个线程在同时执行这个方法时，避免并行。



在普通方法上加同步锁，锁的是方法对应的当前对象，而不是某一个方法。当前对象中所有加了锁的方法公用一个锁。

不同的对象使用不同的锁：给普通方法加同步锁，然后不同线程使用不同的对象，对象的类如果有static资源的话，由于static资源在对象之间共享，还是线程不安全的。

静态方法加同步锁，那么对于所有对象都是同一个锁。

1. Synchronized(对象){

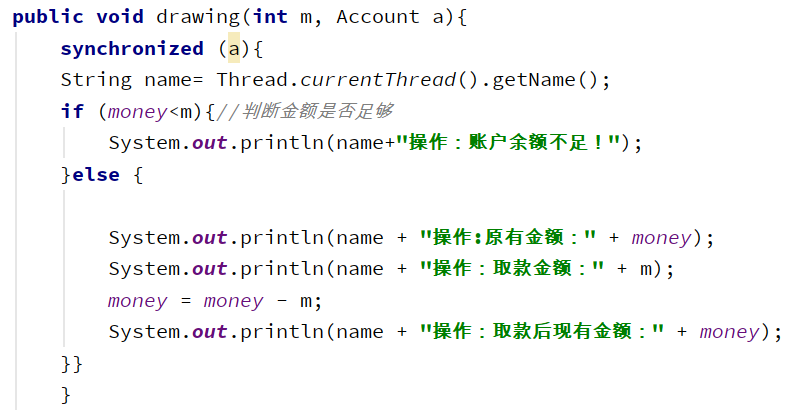
需要同步的代码

}



用synchronized（this）{}括起来的代码块，不管存在于多少个方法里，用的都是同一把锁。

synchronized（this）{}如何根据不同对象使用使用不同的锁：在方法里传入要锁的对象即可：



总结：如果针对对象加同步锁，那就加在方法上，如果针对某一段代码加锁，就加在代码块上。

1. 线程通信
2. wait() 2. notify() 3. notifyAll()

java.long.Object提供的这三个方法只能在synchronized代码中使用。

1：当前线程挂起并释放CPU、同步资源，当前线程排队等待再次对资源的访问

2：唤醒正在排队等待资源的优先级最高的线程 结束等待

3：唤醒所有排队等待资源的线程结束等待。

使用：对象名.wait(); 使当前线程进入等待状态，直到另一个线程对该对象发出: 对象名.notify(); 为止。

对象名.notify() 唤醒等待该对象监控权的一个线程。 调用该方法的当前线程必须具有该对象的监控权（同步锁）。