**面向对象设计原则**

面向对象设计原则是学习设计模式的基础，每一种设计模式都符合某一种或多种面向对象设计原则。

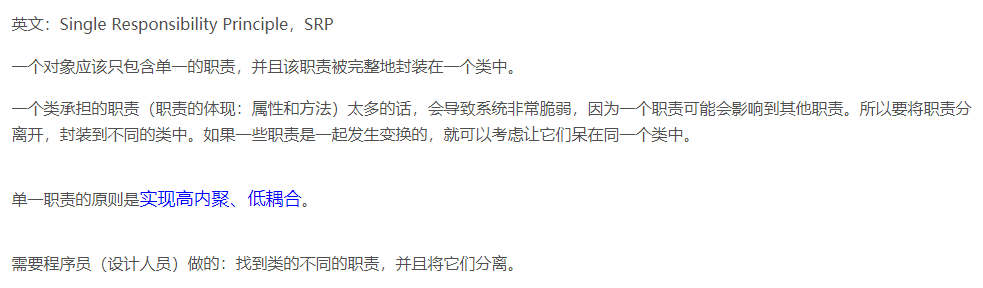
而这些原则是为了在支持可维护性的同时提高系统的可复用性，其中的可维护性的复用是一设计原则为基础的。

对于面向对象的软件系统设计来说，在支持可维护性的同时，提高系统的可复用性是一个核心问题，面向对象设计原则正是为解决这个问题而诞生的。



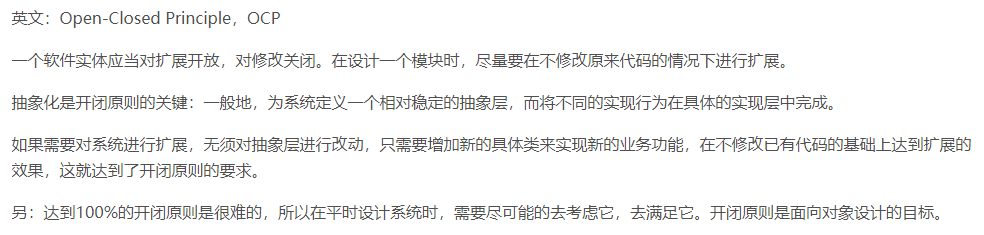
1. **单一职责原则**

**一个类只负责一个功能领域中的相应职责**



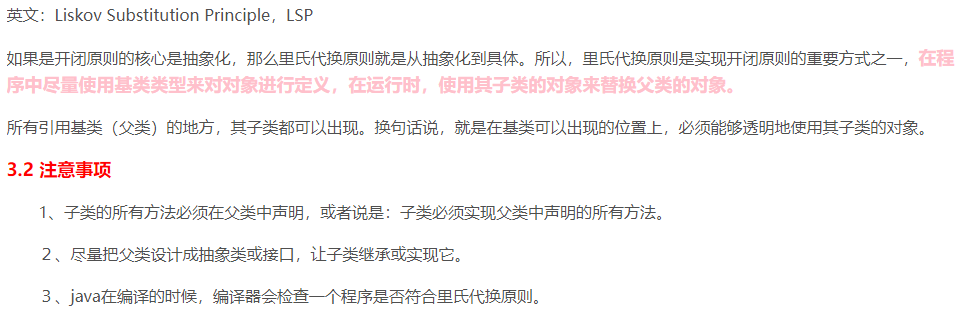
1. **开闭原则**

**一个软件实体应当对扩展开放，对修改关闭。即软件实体应尽量在不修改原有代码的情况下进行扩展**



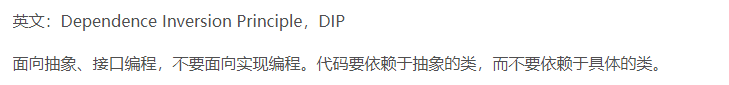
**3、里氏代换原则**

**所有引用基类(父类)的地方必须能透明地使用其子类的对象**



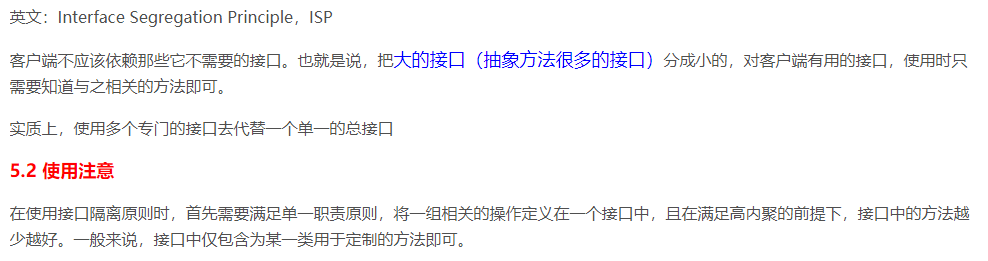
**4、依赖倒转原则**

**抽象不应该依赖于细节，细节应当依赖于抽象。换言之，要针对接口编程，而不是针对实现编程**



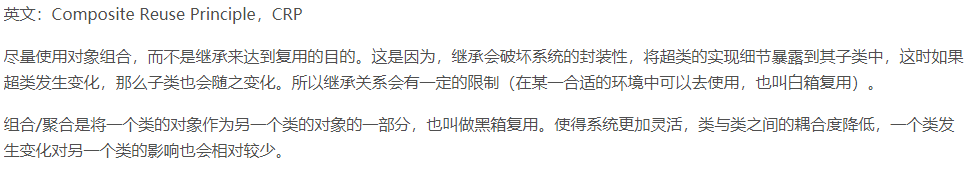
**5、接口隔离原则**

**使用多个专门的接口，而不使用单一的总接口，即客户端不应该依赖那些它不需要的接口**



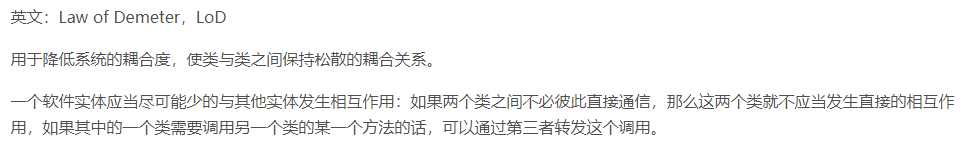
**6、合成复用原则**

**尽量使用对象组合，而不是继承来达到复用的目的**



**7、迪米特法则**

**最少知道原则，一个软件实体应当尽可能少地与其他实体发生相互作用**



**Java跨平台原理**

Java程序不是直接在电脑上运行的，**是在虚拟机上运行的，JVM也是一个软件**，不同的平台有不同的版本。JVM就是负责将字节码文件翻译成特定平台下的机器码然后运行。Java代码首先被编译成字节码文件，再由JVM将字节码文件翻译成机器语言，**字节码不能直接运行，必须通过JVM翻译成机器码才能运行**。不同平台下编译生成的字节码是一样的，但是由JVM翻译成的机器码却不一样。即使将Java程序打包成可执行文件（例如 .exe），仍然需要JVM的支持。注意：**跨平台的是Java程序，不是JVM**。JVM是用C/C++开发的，是编译后的机器码，不能跨平台，不同平台下需要安装不同版本的JVM。

**JVM：**当我们运行一个程序时，JVM负责将字节码转换为特定机器代码，JVM提供了内存管理/垃圾回收和安全机制等。这种独立于硬件和操作系统，正是Java程序可以一次编写多处执行的原因。

**在JVM内存里面主要分布有以下三个区域：**

**栈**：存放基础数据和自定义对象的引用

**堆**：主要存储创建的对象，即new出来的对象。

**方法区**：加载存放class文件（字节码文件）

**Java的加载与执行**

对于Java程序员来说，主要做的事情就是编写以.Java结尾的文件，我们称之为源文件，把在源文件里面所编写的代码叫做源代码。但是这个.Java结尾的文件是不能直接运行的，需要使用Javac.exe命令编译成以.class结尾的文件，这个文件夹叫做字节码文件。要想运行这个字节码文件的话，需要使用Java.exe命令，之后会启动JVM，JVM中的类装载器会去电脑硬盘里面寻找这个class文件，然后将它翻译成2进制文件交由操作系统去运行。

**JDK**：编写Java程序的程序员使用的软件（Java Development Kit）

**JRE**：运行Java程序的用户使用的软件（Java Runtime Environment）

**SE**：用于桌面或简单服务器应用的Java平台（Standard Edition）标准版

**EE**：用于复杂服务器应用的Java平台（Enterprise Edition）企业版

**ME**：用于手机或其他小型设备的Java平台（Micro Edition）小型版

**Java特点**

简单性、面向对象、分布式、健壮性、安全性、体系结构中立、可移植性、解释型、高性能、多线程、动态性

**Java和C++的区别**

都是面向对象的语言，都支持封装、继承和多态

Java 不提供指针来直接访问内存，程序内存更加安全

Java 的类是单继承的，C++ 支持多重继承；虽然 Java 的类不可以多继承，但是接口可以多继承。

Java 有自动内存管理机制，不需要程序员手动释放无用内存

**Java和C区别**

C语言是面向过程的语言，执行效率高；Java是面向对象的语言，执行效率比C语言低；

C语言的安全性不如Java，C语言没有Java的垃圾回收机制，申请的空间要手动释放。

Java的通用性好，可以跨平台直接移植，只要有安装Java虚拟机（JVM）就可以了。

C语言的变量可以不进行初始化，但Java的变量要进行初始化

**创建对象的五种方式**



**两种newInstance方法有没有区别**

Class类位于java的lang包中，而构造器类是java反射机制的一部分。

Class类的newInstance只能触发无参数的构造方法创建对象，而构造器类的newInstance能触发有参数或者任意参数的构造方法来创建对象。

Class类的newInstance需要其构造方法是共有的或者对调用方法可见的，而构造器类的newInstance可以在特定环境下调用私有构造方法来创建对象。

Class类的newInstance抛出类构造函数的异常，而构造器类的newInstance包装了一个InvocationTargetException异常。

Class类本质上调用了反射包构造器类中无参数的newInstance方法，捕获了InvocationTargetException，将构造器本身的异常抛出。

**为什么有数据类型**

Java语言是强类型语言，对于每一种数据都定义了明确的具体数据类型，在内存中分配了不同大小的内存空间，可以合理的分配内存空间。在定义变量的时候，声明这个变量的数据类型，这样就会给这个变量开辟一块合理大小的内存区域，从而避免空间浪费。

**八种基本数据类型**



**8种基本数据类型除boolean类型之外都可以相互转换。byte,short,char做混合运算的时候，各自都先转换成int在做运算。多种数据类型做混合运算，先转换成容量最大的那种再做运算。在Java语言里面除了基本数据类型，其他的都是引用数据类型。**

**char型变量中能不能存储一个中文汉字? 为什么?**

char型变量是用来存储Unicode编码的字符的，unicode编码字符集中包含了汉字。不过，如果某个特殊的汉字没有被包含在 unicode 编码字符集中，那么，这个char型变量中就不能存储这个特殊汉字。unicode编码占用两个字节，所以，char类型的变量也是占用两个字节。

**short s1 = 1; s1 = s1 + 1; 有什么错? short s1 = 1; s1 += 1; 有什么错?**

对于short s1 = 1；s1 = s1 + 1；由于s1+1运算时**会自动提升表达式的类型**，所以结果是int型，再赋值给short类型s1时，编译器将报告需要强制转换类型的错误。

对于 short s1 = 1；s1 += 1；**由于+=是 java 语言规定的运算符，java 编译器会对它进行特殊处理，因此可以正确编译**。

**五种引用数据类型**

类、接口类型、数组类型、枚举类型、注解类型

**基本类型与引用类型**

基本类型值指的是那些**保存在栈内存中的简单数据段**，即这种值完全保存在内存中的一个位置。

引用类型值则是指那些**保存在堆内存中的对象**，意思是变量中保存的实际上只是一个指针，这个指针指向内存中的另一个位置，该位置保存对象。

**参数传递**

基本数据类型的值传递,不改变其值

引用数据类型的值传递,改变其值

String类虽然是引用数据类型,但它当做参数传递时和基本数据类型是一样的

**当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递?**

是值传递。Java 语言的方法调用只支持参数的值传递。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的属性可以在被调用过程中被改变，但对对象引用的改变是不会影响到调用者的。C++和 C#中可以通过传引用或传输出参数来改变传入的参数的值。说明：Java 中没有传引用实在是非常的不方便，这一点在 Java 8 中仍然没有得到改进，正是如此在 Java 编写的代码中才会出现大量的 Wrapper 类（将需要通过方法调用修改的引用置于一个 Wrapper 类中，再将 Wrapper 对象传入方法），这样的做法只会让代码变得臃肿，尤其是让从 C 和 C++转型为 Java 程序员的开发者无法容忍。

**^：逻辑异或，两边只要是不一致就是true**

**移位运算符**

<<：左移运算符，num << 1，相当于num乘以2

>>：右移运算符，num >> 1，相当于num除以2

>>>：无符号右移，忽略符号位，空位都以0补齐

**&&和&的区别**

&&具有短路效果。如果左边结果是false，则右边不执行。

&是无论左边是false还是true,右边都会执行。

**分支语句中，如果只有一条Java语句，大括号可以省略，但是不推荐这样使用，建议加上大括号**

**switch后面的括号中可以填写byte、short、char、int类型，即能够自动转换为int类型的都可以，在jdk7版本之后可以填写String类型。case后面只能是常量，不能是变量，而且，多个case后面的值不能出现相同的**

**while循环本质上和for循环是一样的，可以相互替换，作用相同，格式不同，实际工作中建议使用for循环，因为变量会及早的从内存中消失，可以提高内存的使用效率。**

**break与continue,return结束循环区别**

break：跳出一层循环，可以控制结束嵌套循环

continue：结束一趟循环，也可以像break那样加上循环的名字

return：它的作用不是结束循环的，而是结束方法的

**什么是面向对象**  
面向对象是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述某个事物在整个解决问题的步骤中的行为

**面向过程和面向对象的区别**

**面向过程**注重过程，把问题分解成多个不同的步骤，然后把各个步骤变成方法，使用的时候依次调用

**面向对象**注重对象之间的交互，将复杂的事情简单化，把问题分解成各个对象，然后各个对象之间进行交互，每个对象内部都进行了封装

**在工作当中尽量面向抽象编程，不要面向具体编程。**

### 面向过程

**优点：** 性能比面向对象高，类调用时需要实例化，开销比较大，比较消耗资源

**缺点：** 没有面向对象易维护、易复用、易扩展

### 面向对象

**优点：** 易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护

**缺点：** 性能比面向过程低

**类是具有相同属性和方法的一组对象的集合**

**对象是系统中描述客观事物的一个实体**

**方法是指为了完成某个任务，对象所能执行的操作**

**方法的优点就是可以提高代码的复用性**。

**不要在两个方法里面互相调用，程序会报出StackOverflowError错误。**

**main方法是所有程序的入口，这是sun公司规定的，程序都是从main方法开始执行的，如果将一个程序比喻成一座大楼，那么main方法就相当于是这个楼最外面的大门。**

**(String[] args)是形式参数列表，该参数表示是一个String类型的数组，形参的名字是随意的，只不过通常都会将参数名字写成args。**

**什么是递归**

递归就是方法自己调用自己，在编写的时候一定要有一个结束条件，否则将会造成StackOverflowError的错误。

**什么是栈**

栈的英文是stack，特点是先进后出，可以把栈想象成一个桶，向里面放置元素的时候叫做压栈（push），向外取出元素的时候叫做弹栈（pop）。在JVM当中有一块内存空间，就是栈区，专门为方法准备的。方法执行所需要的内存空间都在栈当中分配。

方法调用之后给该方法分配内存，在JVM的“栈区”当中分配空间，

方法调用分配空间的过程就是：压栈（push）的过程。

方法执行结束之后，给这个方法分配的所有空间全部释放，此时发生弹栈（pop）的动作。

如果持续的向栈里面进行压栈，超出栈的存储空间的话，就会出现StackOverflowError的错误。

**成员变量与局部变量的区别**

**成员变量**：写在类里面，方法体的外面，声明时可以不进行初始化值，会自动以类型的默认值而赋值，可以被本类或其他类的方法进行调用，可以被访问控制修饰符及 static 所修饰，存在于堆内存，随着对象的创建而存在  
**局部变量**：写在方法体的里面，声明时必须进行初始化，只能在声明局部变量的方法内进行调用，不能被访问控制修饰符及 static 所修饰，存在于栈内存，随着方法的调用而自动消失

**什么是封装**  
封装是指隐藏对象的属性和实现细节，仅对外提供公共访问方式

**封装的优点**  
不用关心具体实现，提高安全性，提高了代码的复用性，减少耦合。

**封装的使用**

将成员变量用private修饰

提供对应的getXxx()和setXxx()方法

private仅仅是封装的一种体现形式,封装不是私有

### 什么是继承？

继承是子类自动共享父类数据结构和方法的机制，是类之间的一种关系。在定义和实现一个类的时候，可以在一个已经存在的类的基础之上来进行，把这个已经存在的类所定义的内容作为自己的内容，并加入若干新的内容。在**Java语言里面只支持单继承**，支持**多层继承，**子类可以继承父类中的非private修饰的成员方法和成员变量，构造方法不能被继承。**如果一个类没有显示的继承其他类，那么这个类会默认继承Object类，Object是SUN公司提供的Java中的根类**。

**继承的优点**

提高了代码的复用性

提高了代码的维护性

让类与类之间产生了关系，是多态的前提

**继承的缺点**  
增强了类之间的耦合。  
软件开发的一个原则是**高内聚，低耦合。**

**内聚是一个模块内各个元素彼此结合的紧密程度  
耦合是一个软件里面不同模块之间相互连接的数量**

**什么是多态？**

一种行为，多种状态。同一个接口，不同的实例执行不同的操作，在面向对象语言中，接口的多种不同的实现方式即为多态。

**如何实现多态**

有继承

有方法重写

有父类引用指向子类对象

**静态绑定：**在程序执行前已经被绑定，也就是说在编译过程中就已经知道这个方法是哪个类的方法，此时由编译器获取其他连接程序实现。

**动态绑定：**在运行根据具体对象的类型进行绑定。

子类向父类型进行转换，是**自动类型转换**，也叫做向上转型。还有一种情况是父类向子类型转换，是**强制类型转换**，也叫向下转型。

**多态的优点**

提高了程序的扩展性

降低了代码之间的耦合

**方法重载的条件**

发生在同一个类中

方法名相同，参数列表不同(类型，个数，顺序)

方法重载和返回值类型、修饰符无关

### 方法重写

重写，也叫做覆盖，当父类中的方法无法满足子类需求时，子类可以将父类的方法进行重写编写来满足需求。

**方法重写的条件**

必须是继承关系的两个类

必须具有**相同的方法名，相同的返回值类型，相同的参数列表**

重写的方法**不能比被重写的方法拥有更低的访问权限，抛出更宽泛的异常。**

私有方法、构造方法不能被重写。

静态的方法不存在重写。

重写指的是成员方法，和成员变量无关。

**为什么函数不能根据返回类型来区分重载？**

因为调用时不能指定类型信息，编译器不知道你要调用哪个函数。例如：

1．float max(int a, int b);

2．int max(int a, int b);

当调用max(1, 2);时无法确定调用的是哪个，单从这一点上来说，仅返回值类型不同的重载是不应该允许的。

再比如对下面这两个方法来说，虽然它们有同样的名字和自变量，但其实是很容易区分的：

1．void f() {}

2．int f() {}

若编译器可根据上下文（语境）明确判断出含义，比如在int x=f()中，那么这样做完全没有问题。然而，我们也可能调用一个方法，同时忽略返回值；我们通常把这称为“为它的副作用去调用一个方法”，因为我们关心的不是返回值，而是方法调用的其他效果。所以假如我们像下面这样调用方法：f(); Java 怎样判断f()的具体调用方式呢？而且别人如何识别并理解代码呢？由于存在这一类的问题，所以不能。

函数的返回值只是作为函数运行之后的一个“状态”，他是保持方法的调用者与被调用者进行通信的关键。并不能作为某个方法的“标识”。

**构造方法的作用**

给对象数据进行初始化。

**构造方法是否可被重写**

构造方法不能被继承，因此不能被重写，但可以被重载。

**构造方法格式特点**

方法名与类名相同

没有返回值类型

没有void修饰

没有具体的返回值return;

如果一个类没有提供任何构造方法，系统默认提供无参数构造方法

如果一个类已经手动的提供了构造方法，那么系统不会再提供任何构造方法。

在创建对象的时候会默认执行**无参构造方法**。**构造方法不能手动调用。**

如果我们还想使用无参构造方法，就必须自己写出。**建议自己写出无参构造方法**

**This**

**代表着对当前对象的引用，**在堆内存中的每个Java对象上都有一个this指向自己

**This的作用？**

可以使用this.方法名的方式调用当前对象的方法

可以区分成员变量和局部变量。

如果要是使用this调用构造方法的话，那么this**必须出现在构造方法的第一行**。

在Java里面，局部变量和成员变量的标识符是可以相同的，如果有相同的标识符，则可以通过this调用成员变量

### Super

super代表的是当前子类对象中的父类型特征。

**Super的作用**  
子类和父类中都有某个数据，例如，子类和父类中都有name这个属性。如果要再子类中访问父类中的name属性，需要使用super。

子类重写了父类的某个方法（假设这个方法名叫m1），如果在子类中需要调用父类中的m1方法时，需要使用super。

子类调用父类中的构造方法时，需要使用super，**一个构造方法第一行如果没有this(…);也没有显示的去调用super(…);系统会默认调用super();如果已经有this了，那么就不会调用super了**，super(…);的调用只能放在构造方法的第一行，只是调用了父类中的构造方法，但是并不会创建父类的对象。

### super和this的区别

**调用成员变量**

this.成员变量： 调用本类的成员变量

super.成员变量： 调用父类的成员变量

**调用构造方法**

this(…) ：调用本类的构造方法

super(…)：调用父类的构造方法

**调用成员方法**

this.成员方法：调用本类的成员方法

super.成员方法：调用父类的成员方法

### static的作用

**可以修饰变量**

被static修饰的变量叫做静态变量，程序运行时静态变量存放在方法区里面。

**可以修饰方法**

被static修饰的方法叫做静态方法，**不用创建对象就能直接访问该方法**。静态方法不能直接访问非静态的数据，静态方法不能使用this和super。

**可以定义静态语句块**

**静态语句块在类加载阶段执行，并且只执行一次，并且是自上而下的顺序执行，在构造方法之前执行。**

static修饰的变量、方法、代码块都是隶属于**类(class)**级别的，跟对象无关。某一类物体如果可以被多个其他物体所共享，那么可以将这类物体使用static修饰。可以通过类名.静态方法名的方式访问，工作中使用这种写法

**静态变量和实例变量的区别**

**在语法定义上的区别**

静态变量前要加static关键字，而实例变量前则不加。

**在程序运行时的区别**

**实例变量属于某个对象的属性，必须创建了实例对象**，其中的实例变量才会被分配空间，才能使用这个实例变量。

**静态变量不属于某个实例对象，而是属于类**，所以也称为类变量，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用。

**是否可以从一个statc方法内部发出对非 static 方法的调用？**

不可以。因为非static方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，而 static 方法调用时不需要创建对象，可以直接调用。也就是说，当一个static方法被调用时，可能还没有创建任何实例对象，如果从一个static方法中发出对非static方法的调用，那个非static方法是关联到哪个对象上的呢？这个逻辑无法成立，所以，一个static方法内部发出对非static方法的调用。

**Java中是否可以覆盖(override)一个 private 或者是 static 的方法？**

Java 中 static 方法不能被覆盖，因为方法覆盖是基于运行时动态绑定的，而 static 方法

是编译时静态绑定的。static 方法跟类的任何实例都不相关，所以概念上不适用。

java 中也不可以覆盖 private 的方法，因为 private 修饰的变量和方法只能在当前类中使用，

如果是其他的类继承当前类是不能访问到 private 变量或方法的，当然也不能覆盖。

### final的作用

final修饰的**类无法被继承**。

final修饰的**方法无法被重写**

final修饰的**局部变量，一旦赋值，不可再改变**。

final修饰的**成员变量必须初始化值**。

final修饰的引用类型，该引用不可再重新指向其他的Java对象。但是**fianl修饰的引用，地址值不能被改变，该引用指向的对象的属性值是可以修改的**。

### 什么是代码块？

使用{}括起来的代码被称为代码块

**局部代码块**，在方法中出现，限定变量生命周期，及早释放，提高内存利用率

**构造代码块**，在类中方法外出现；多个构造方法方法中相同的代码存放到一起，每次调用构造都执行，并且在构造方法前执行

**静态代码块**，在类中方法外出现，并加上static修饰；用于给类进行初始化，在加载的时候就执行，并且只执行一次。一般用于加载驱动。

**同步代码块**（多线程）

方法执行顺序  
1、**静态代码块**，随着类加载而加载,且只执行一次  
2、**构造代码块**，每创建一个对象就会执行一次，优先于构造方法执行  
3、**构造方法**，每创建一个对象就会执行一次

**抽象类的特点**

抽象类无法被实例化，无法创建抽象类的对象，但是**抽象类也有构造方法**，该构造方法是给子类创建对象用的。

抽象类中不一定有抽象方法，但抽象方法必须出现在抽象类中。

抽象类中的子类可以是抽象类，如果不是抽象类的话必须对抽象类中的抽象方法进行重写。

抽象类和抽象方法不能被 final修饰

**接口的特点**

在Java语言里面使用interface来声明一个接口，**接口其实是一个特殊的抽象类，在接口里面的方法全部都是抽象的。**

接口中只能出现常量和抽象方法

接口里面没有构造方法，无法创建接口的对象

接口和接口之间支持多继承，即一个接口可以有多个父接口

一个类可以实现多个接口，即一个类可以有多个父接口

一个类如果实现了接口，那么这个类需要重写接口中所有的抽象方法（建议），如果不重写则这个类需要声明为抽象类（不建议）

接口中方法默认用public abstract，可以省略

接口中常量必须用public static final修饰，可以省略

### 接口的作用

可以使项目分层，都面向接口开发，提高开发效率

降低了代码之间的耦合度，提高了代码的可插拔性

开发中尽量使用接口，少用抽象类，一个类可以实现多个接口，却只能继承一个父类

**接口和抽象类的区别是什么**

**抽象类**

抽象类不能被实例化，只有抽象类的非抽象子类可以创建对象。

抽象类中不一定包含抽象方法，但是有抽象方法的类必定是抽象类。

抽象类可以提供某些方法的部分实现

构造方法，类方法（用static修饰的方法）不能声明为抽象方法。

抽象类的子类必须给出抽象类中的抽象方法的具体实现，除非该子类也是抽象类。

抽象是对类的抽象，是一种模板设计

**接口**

接口不能用于实例化对象，但可以声明，但是必须引用一个实现该接口的对象。

接口没有构造方法。

接口中所有的方法必须是抽象方法。

接口不能包含成员变量，除了常量。

接口不是被类继承了，而是要被类实现。

接口是行为的抽象，是一种行为的规范。

**API是Application Programming Interface的缩写,中文意思是应用程序编程接口。sun公司提供了一些预先定义好的类和方法以供Java开发者使用，有了API，开发者可以在不查看源代码的情况下了解某个类或者某个方法的使用方式。**

SUN在Object类中设计toString方法的目的：返回Java对象的字符串表示形式。

在现实的开发过程中，Object中的toString方法就是要被重写的。

如果直接打印一个引用数据类型的对象，系统会默认调用其toString方法。

### finalize方法

finalize方法不需要程序员去调用，由**系统自动调用**。Java对象如果没有更多的引用指向它，则该Java对象成为垃圾数据，等待垃圾回收器的回收，垃圾回收器在回收这个Java对象之前会自动调用该对象的finalize方法。finalize方法是该对象马上就要被回收了，例如：需要释放资源，则可以在该方法中释放。程序员只能“建议”垃圾回收器回收垃圾：**System.gc()**;

重写Object中的finalize方法.，不是给程序员调用，垃圾回收器在回收之前会自动调用

**包**

在类名前面使用关键字package加入包名来避免命名冲突问题，因为域名是世界上唯一的，所以建议使用公司倒写的域名来命名包名,通常是**小写**的

**package语句只能出现在.Java源文件的第一行**

**package语句在一个Java文件中只能有一个**

**如果没有package，默认表示无包名**

**Java.lang软件包下所有类不需要手动导入，系统自动导入，Object类，String类都在这个包里面**

**访问级别修饰符**

private、protected、public和default（默认），可以限定其他类对该类、属性和方法的使用权限。

**对类的修饰只有：public和default**

**private：本类**

**default：本类、同包类**

**protected：本类、同包类、子类**

**public：所有类**

### 内部类

内部类，顾名思义就是在一个类的内部声明一个类。

**内部类不能定义静态成员，可以直接访问外部类的成员，包括private修饰的变量和方法**

内部类主要分为：

静态内部类

匿名内部类

成员内部类

局部内部类

### 什么是数组？

数组是存储同一种数据类型多个元素的集合。也可以看成是一个容器。

数组里面既可以存储基本数据类型，也可以存储引用数据类型，但是**只能存储同一种数据类型**。

数组是一种简单的数据结构，线性的结构

**数组一旦创建其长度是不可变的**

数组是引用数据类型

### 静态初始化数组

数组初始化就是为数组开辟连续的内存空间，并为每个数组元素赋予值。知道数组的首元素的内存地址，要查找的元素只要知道下标就可以快速的计算出偏移量，通过首元素内存地址加上偏移量可以快速计算出要查找元素的内存地址，通过内存地址快速定位该元素， 所以数组查找元素的效率较高。  
**注意：数组下标是从0开始的**

### 动态初始化数组

动态初始化数组,会先在堆内存中分配这个数组，并且数组中每一个元素都采用默认值。

main方法中的String[] args是专门用来接收命令行参数的。

### 异常的分类

异常主要分为：Error、一般性异常、RuntimeException

**Error：**如果程序出现了Error，那么将无法恢复，只能重新启动程序，最典型的Error的异常是：OutOfMemoryError

**一般性异常：**出现了这种异常必须在程序里面显示的处理，否则程序无法编译通过

**RuntimeException：**此种异常可以不用显示的处理，例如被0除异常，java没有要求我们一定要处理。

**为何要用异常**

JAVA号称安全性，异常考虑到可能出现的错误，

使用异常处理后

把错误处理和真正的工作分开来；

代码更易组织，更清晰，复杂的工作任务更容易实现；

更安全，不至于由于一些小的疏忽而使程序意外崩溃了；

**Java中的异常处理机制的简单原理和应用。**

Java 对异常进行了分类，不同类型的异常分别用不同的 Java 类表示，所有异常的根类为 java.lang.Throwable，Throwable 下面又派生了两个子类：Error 和 Exception，Error 表示应用程序本身无法克服和恢复的一种严重问题。Exception 表示程序还能够克服和恢复的问题，其中又分为系统异常和普通异常，系统异常是软件本身缺陷所导致的问题，也就是软件开发人员考虑不周所导致的问题，软件使用者无法克服和恢复这种问题，但在这种问题下还可以让软件系统继续运行或者让软件死掉，例如，数组脚本越界（ArrayIndexOutOfBoundsException），空指针异常（NullPointerException）、类转换异常（ClassCastException）；普通异常是运行环境的变化或异常所导致的问题，是用户能够克服的问题，例如，网络断线，硬盘空间不够，发生这样的异常后，程序不应该死掉。java为系统异常和普通异常提供了不同的解决方案，编译器强制普通异常必须try..catch处理或用throws声明继续抛给上层调用方法处理，所以普通异常也称为 checked 异常，而系统异常可以处理也可以不处理，所以，编译器不

强制用 try..catch 处理或用 throws 声明，所以系统异常也称为 unchecked 异常。

**常见的runtimeException**

ArithmeticException、ArrayIndexOutOfBoundsException、ClassCastException、IllegalArgumentException、IllegalStateException、IndexOutOfBoundsException、NullPointerException、NumberFormatException、

### throws

throws的作用是声明抛出异常，在方法声明的位置上使用throws关键字向上抛出异常。

**被finally控制的语句体一定会执行**，除非在执行finally语句体之前JVM退出(比如System.exit(0))，一般用于关闭资源

**final,finally和finalize的区别？**

**final**可以修饰类,不能被继承，修饰方法,不能被重写，修饰变量,只能赋值一次

**finally**是try语句中的一个语句体,不能单独使用,语句体中的语句一定会执行

**finalize**是Object中的一个方法,当没有引用指向这个对象时，由对象的垃圾回收器在回收之前调用此方法。

**finally和return的执行顺序**

如果程序是从try代码块或者catch代码块中返回时，finally中的代码总会执行。而且finally语句在return语句执行之后return返回之前执行的。

当finally有返回值时，会直接返回，不会再去返回try或者catch中的返回值。

如果try和catch的return是一个变量时且函数的是从其中一个返回时，后面finally中语句即使有对返回的变量进行赋值的操作时，也不会影响返回的值。

### throw和throws的区别

**throws**

用在方法声明后面，跟的是异常类名

可以跟多个异常类名，用逗号隔开

表示抛出异常，由该方法的调用者来处理

**throw**

用在方法体内，跟的是异常对象名

只能抛出一个异常对象名

表示抛出异常，由方法体内的语句处理

**Throwable类常用方法**

getMessage()返回异常发生时的详细信息

string toString()返回异常发生时的简要描述

string getLocalizedMessage()返回异常对象的本地化信息。使用Throwable的子类覆盖这个方法，可以声称本地化信息。如果子类没有覆盖该方法，则该方法返回的信息与getMessage（）返回的结果相同

void printStackTrace(:在控制台上打印Throwable对象封装的异常信息

**equals跟==区别**

== : 它的作用是**判断两个对象的地址是不是相等**。即判断两个对象是不是同一个对象。**基本数据类型比较的是值，引用数据类型比较的是内存地址。**

equals() : 它的作用也是**判断两个对象是否相等**。但它一般有两种使用情况：

**类没有覆盖equals()方法。**则通过equals()比较该类的两个对象时，等价于通过“==”比较这两个对象。

**类覆盖了equals()方法。**一般，我们都覆盖equals()方法来两个对象的内容相等；若它们的内容相等，则返回true，即认为这两个对象相等。

String中的equals方法是被重写过的，因为object的equals方法是比较的对象的内存地址，而String的equals方法比较的是对象的值。

**hashCode()**

hashCode() 的作用是获取哈希码，也称为散列码；它实际上是返回一个int整数。这个哈希码的作用是确定该对象在哈希表中的索引位置。hashCode() 定义在JDK的Object.java中，这就意味着Java中的任何类都包含有hashCode() 函数。

**散列表存储的是键值对(key-value)，它的特点是：能根据“键”快速的检索出对应的“值”。这其中就利用到了散列码！（可以快速找到所需要的对象）**

**为什么要有hashCode**

我们以“HashSet 如何检查重复”为例子来说明为什么要有 hashCode：

当你把对象加入 HashSet 时，HashSet 会先计算对象的 hashcode 值来判断对象加入的位置，同时也会与其他已经加入的对象的 hashcode 值作比较，如果没有相符的hashcode，HashSet会假设对象没有重复出现。但是如果发现有相同 hashcode 值的对象，这时会调用 equals（）方法来检查 hashcode 相等的对象是否真的相同。如果两者相同，HashSet 就不会让其加入操作成功。如果不同的话，就会重新散列到其他位置。这样我们就大大减少了 equals 的次数，相应就大大提高了执行速度。

**hashCode（）与equals（）的相关规定**

如果两个对象相等，则hashcode一定也是相同的

两个对象相等,对两个对象分别调用equals方法都返回true

两个对象有相同的hashcode值，它们也不一定是相等的

因此，equals 方法被覆盖过，则 hashCode 方法也必须被覆盖

hashCode() 的默认行为是对堆上的对象产生独特值。如果没有重写 hashCode()，则该 class 的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）



**String**

String类在java.lang包下面，是Object类的直接子类，通过API或者源码可以看到，String类是final修饰的，这说明**String类不能被继承**。String 被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。

字符串一旦创建好之后，里面的内容是不能被修改的，jvm会将双引号””中的内容存放在字符串常量池里面，常量池中的对象内容是不可修改的。创建字符串时，jvm会在常量池中创建一个字符串对象，创建另一个字符串时，jvm会去常量池中搜索，如果此时常量池中有相同的字符串对象，就不创建了，直接指向之前创建的字符串对象。

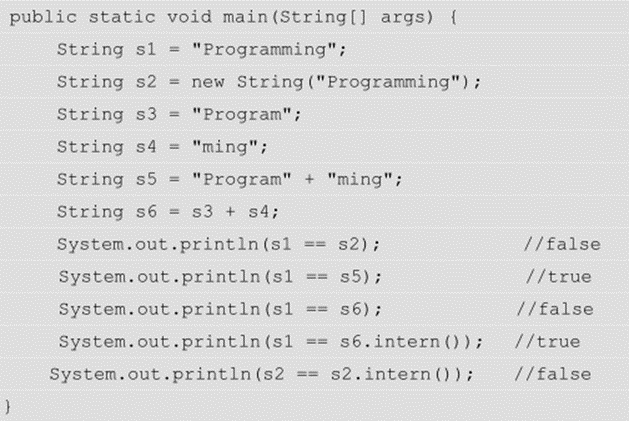
当创建String类型的对象时，虚拟机会在常量池中查找有没有已经存在的值和要创建的值相同的对象，如果有就把它赋给当前引用。如果没有就在常量池中重新创建一个String对象。

在Java中无论使用何种方式进行字符串连接，实际上都使用的是StringBuilder

String 对象的 intern（）方法会得到字符串对象在常量池中对应的版本的引用（如果常量池中有一个字符串与String 对象的 equals 结果是 true），如果常量池中没有对应的字符串，则该字符串将被添加到常量池中，然后返回常量池中字符串的引用

**String类常用方法**





**StringBuffer**是一个字符串缓冲区，如果需要频繁的对字符串进行拼接时，建议使用StringBuffer。

**StringBuffer工作原理**  
**StringBuffer的底层是char数组，如果没有明确设定，则系统会默认创建一个长度为16的char类型数组，在使用时如果数组容量不够了，则会通过数组的拷贝对数组进行扩容，所以在使用StringBuffer时最好预测并手动初始化长度，这样能够减少数组的拷贝，从而提高效率。**

**String与StringBuffer的区别**  
String是不可变字符序列，存储在字符串常量池中  
StringBuffer的底层是char数组，系统会对该数组进行扩容

### StringBuilder和StringBuffer的区别

通过API可以看到StringBuilder和StringBuffer里面的方法是一样的，那他们有什么区别呢？  
StringBuffer是jdk1.0版本中加入的，是线程安全的，效率低  
StringBuilder是jdk5版本加入的，是线程不安全的，效率高

**String s="a"+"b"+"c"+"d"一共创建了多少个对象**

代码被编译器在编译时优化后，相当于直接定义了一个”abcd”的字符串，所以上面的代码应该只创建了一个String对象。

**为什么String是不可变的**

String所有属性都被final修饰、私有的并且没有提供修改方法。（主要字段是char数组，虽然被final修饰但数组是可变的，私有保证了不被修改但还是可以通过反射来改变String）

为什么String设计成final

避免被继承后破坏，防止方法被重写

**为什么String设计成不可变的**

可以缓存 hash 值，因为 String 的 hash 值经常被使用，例如 String 用做 HashMap 的 key。不可变的特性可以使得 hash 值也不可变，因此只需要进行一次计算。

字符串常量池的需要，提升效率和减少内存分配

安全性

String 经常作为参数，String 不可变性可以保证参数不可变。例如在作为网络连接参数的情况下如果 String 是可变的，那么在网络连接过程中，String 被改变，改变 String 对象的那一方以为现在连接的是其它主机，而实际情况却不一定是。

线程安全

String 不可变性天生具备线程安全，可以在多个线程中安全地使用。

作为HashMap、HashTable等hash型数据key的必要。因为不可变的设计，jvm底层很容易在缓存String对象的时候缓存其hashcode，这样在执行效率上会大大提升。

**String有重写Object的hashcode和toString吗？如果重写equals不重写hashcode会出现什么问题**

String重写了Object类的hashcode和toString方法。当equals方法被重写时，通常有必要重写hashcode方法，以维护hashCode方法的常规协定，该协定声明相对等的两个必须有相同的hashcode。

重写equals不重写hashcode会出现什么问题

在存储散列集合时(如Set类)，如果原对象.equals(新对象)，但没有对hashCode重写，即两个对象拥有不同的hashcode，则在集合中将会存储两个值相同的对象，从而导致混淆。因此在重写equals方法时，必须重写hashcode方法。



**包装类**

Java里面8个基本数据类型都有相应的类，这些类叫做包装类，包装类都在java.lang包里面。

**包装类的优点**

可以在对象中定义更多的功能方法操作该数据，方便开发者操作数据，例如基本数据类型和字符串之间的转换。

**基本类型与包装类型的异同：**

在Java中，一切皆对象，但八大基本类型却不是对象。

包装类是对象，拥有方法和字段，对象的调用都是通过引用对象的地址，基本类型不是。集合类使用时只能用包装类型。

**包装类型是引用的传递，基本类型是值的传递。**

**声明方式的不同**

基本类型无需通过new关键字实例化来创建，而封装类型需new关键字实例化。

**存储方式及位置的不同**

基本类型是直接存储变量的值保存在堆栈中能高效的存取，包装类型是把对象放在堆中，然后通过对象的引用来调用他们。

**初始值不同**

封装类型的初始值为null，基本类型的的初始值视具体的类型而定，例如int的初始值为0 、boolean的初始值为false，而包装类型的初始值为null

**延伸**



**int,Integer,String三种类型之间的转换**

**int–>Integer**

Integer i1 = Integer.valueOf(10);

**Integer–>int**

int i2 = i1.intValue();

**String–>Integer**

Integer i3 = Integer.valueOf("10");

**Integer–>String**

String s1 = i3.toString();

**String–>int**

int i4 = Integer.parseInt("123");

**int–>String**

String s2= 10 + "";

**自动装箱与拆箱**

**装箱**：将基本类型用它们对应的引用类型包装起来；

**拆箱**：将包装类型转换为基本数据类型；

**数组有没有length()这个方法? String有没有length()这个方法？**

数组没有length()这个方法，有length的属性。String有有length()这个方法

**日期/时间**

**y表示年**

**M表示月**

**d表示日**

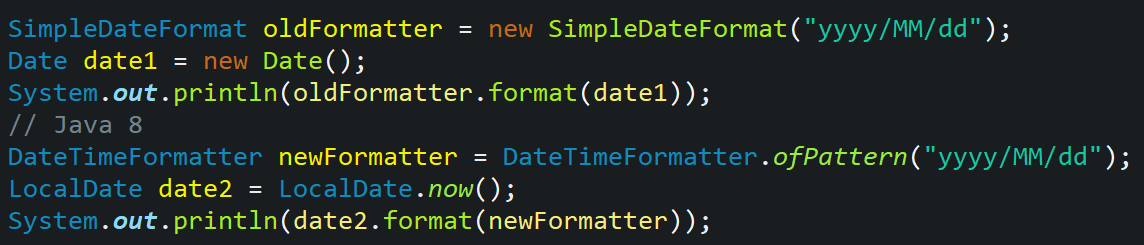
**H表示小时**

**m表示分钟**

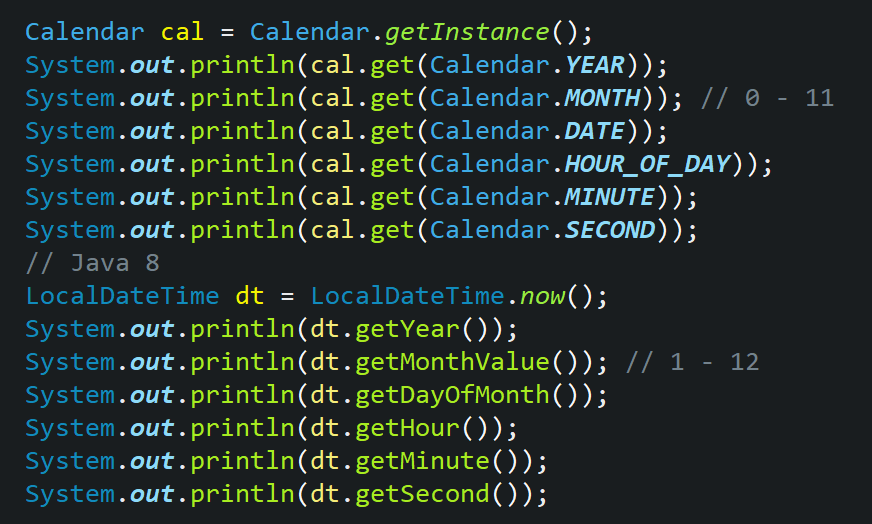
**s表示秒**

**S表示毫秒**

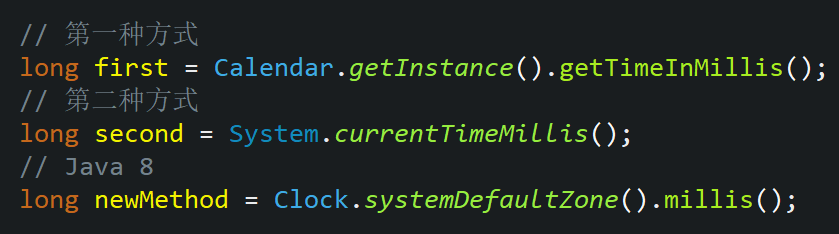
**格式化日期**



**如何取得年月日时分秒**



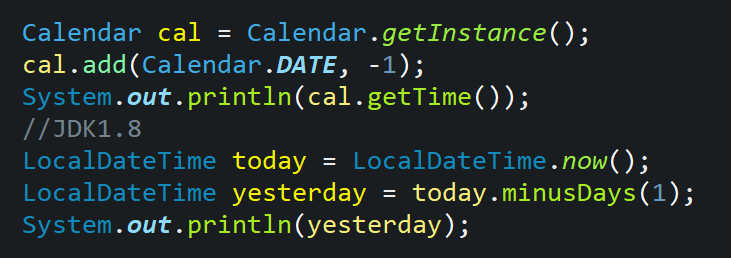
**如何取得从1970年1月1日0时0分0秒到现在的毫秒数**



**如何取得某月的最后一天**



**打印昨天的当前时刻**



**Math常用方法**

Math.abs()绝对值

Math.pow(a,b)计算a的b次方

Math.random()随机数

Math.floor()向下取整

Math.ceil()向上取整

Math.round()四舍五入

parseInt()字符串转换数字

**BigInteger类可以让超过Integer范围的数据进行运算，通常在对数字计算比较大的行业中应用的多一些**

由于在运算的时候，float类型和double很容易丢失精度，在金融、银行等对数值精度要求非常高的领域里面，就不能使用float或double了，为了能精确的表示、计算浮点数，Java提供了BigDecimal。

注意：**如果对计算的数据要求高精度时，必须使用BigDecimal类**

**DecimalFormat类**

DecimalFormat(",###").format(211321123)->211,321,123

DecimalFormat("00.000").format(3.1415926)->03.142

**泛型的优点**

可以统一集合中的数据类型，提高安全性

可以减少强制类型转换

**什么时候用assert**

assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。在实现中，assertion就是在程序中的一条语句，它对一个boolean表达式进行检查，一个正确程序必须保证这个boolean表达式的值为true；如果该值为false，说明程序已经处于不正确的状态下，系统将给出警告或退出。一般来说，assertion用于保证程序最基本、关键的正确性。assertion检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion检查通常是关闭的

**为什么要用clone**

在实际编程过程中，我们常常要遇到这种情况：有一个对象 A，在某一时刻 A 中已经包含了一些有效值，此时可能会需要一个和 A 完全相同新对象 B，并且此后对 B 任何改动都不会影响到 A 中的值，也就是说，A 与 B 是两个独立的对象，但 B 的初始值是由 A 对象确定的。在 Java 语言中，用简单的赋值语句是不能满足这种需求的。要满足这种需求虽然有很多途径，但实现 clone（）方法是其中最简单，也是最高效的手段。使用 clone() 方法来拷贝一个对象即复杂又有风险，它会抛出异常，并且还需要类型转换。Effective Java 书上讲到，最好不要去使用 clone()，可以使用拷贝构造函数或者拷贝工厂来拷贝一个对象。

**new一个对象的过程和clone一个对象的过程区别**

new 操作符的本意是分配内存。程序执行到 new 操作符时，首先去看 new 操作符后面的类型，因为知道了类型，才能知道要分配多大的内存空间。分配完内存之后，再调用构造函数，填充对象的各个域，这一步叫做对象的初始化，

构造方法返回后，一个对象创建完毕，可以把他的引用（地址）发布到外部，在外部就可以使用这个引用操纵这个对象。

clone 在第一步是和 new 相似的，都是分配内存，调用 clone 方法时，分配的内存和原对象（即调用 clone 方法的对象）相同，然后再使用原对象中对应的各个域，填充新对象的域，填充完成之后，clone 方法返回，一个新的相同的对象被创建，同样可以把这个新对象的引用发布到外部。

**深拷贝与浅拷贝**

浅拷贝：拷贝实例和原始实例的引用类型引用同一个对象；

深拷贝：拷贝实例和原始实例的引用类型引用不同对象。