编写 Java 程序时，应注意以下几点：

大小写敏感：Java 是大小写敏感的，这就意味着标识符 Hello 与 hello 是不同的。

类名：对于所有的类来说，类名的首字母应该大写。如果类名由若干单词组成，那么每个单词的首字母应该大写，例如 MyFirstJavaClass 。

方法名：所有的方法名都应该以小写字母开头。如果方法名含有若干单词，则后面的每个单词首字母大写。

源文件名：源文件名必须和类名相同。当保存文件的时候，你应该使用类名作为文件名保存（切记 Java 是大小写敏感的），文件名的后缀为 .java。（如果文件名和类名不相同则会导致编译错误）。

主方法入口：所有的 Java 程序由 public static void main(String []args) 方法开始执行。

一、命名规范

1、 项目名全部小写

(1) 标识符要符合语义信息。

(2) 包名所有字母小写。

(3) 类名每个单词首字母大写，其它小写，如：TarenaStudent。

(4) 变量和方法：第一个单词小写，从第二个单词开始首字母大写，如：tarenaStudent。

(5) 常量：所有字母大写，每个单词之间用 \_ 连接。

1、类注释

在每个类前面必须加上类注释，注释模板如下：

/\*\*

\* Copyright (C), 2006-2010, ChengDu Lovo info. Co., Ltd.

\* FileName: Test.java

\* 类的详细说明

\*

\* @author 类创建者姓名

\* @Date 创建日期

\* @version 1.00

\*/

2、属性注释

在每个属性前面必须加上属性注释，注释模板如下：

/\*\* 提示信息 \*/

private String strMsg = null;

3、方法注释

在每个方法前面必须加上方法注释，注释模板如下：

/\*\*

\* 类方法的详细使用说明

\*

\* @param 参数1 参数1的使用说明

\* @return 返回结果的说明

\* @throws 异常类型.错误代码 注明从此类方法中抛出异常的说明

\*/

5、方法内部注释

在方法内部使用单行或者多行注释，该注释根据实际情况添加。

Java编程规范

package的命名: package 的名字由全部小写的字母组成，例如：com.runoob。

class和interface的命名: class和interface的名字由大写字母开头而其他字母都小写的单词组成，例如：Person，RuntimeException。

class变量的命名: 变量的名字用一个小写字母开头，后面的单词用大写字母开头,例如：index，currentImage。

class 方法的命名: 方法的名字用一个小写字母开头，后面的单词用大写字母开头,例如：run()，getBalance()。

staticfinal变量的命名: static final变量的名字所有字母都大写，并且能表示完整含义。例如：PI，PASSWORD。

参数的命名: 参数的名字和变量的命名规范一致。

数组的命名: 数组应该总是用这样的方式来命名：byte[] buffer。

个类可以包含以下类型变量：

局部变量：在方法、构造方法或者语句块中定义的变量被称为局部变量。变量声明和初始化都是在方法中，方法结束后，变量就会自动销毁。

成员变量：成员变量是定义在类中，方法体之外的变量。这种变量在创建对象的时候实例化。成员变量可以被类中方法、构造方法和特定类的语句块访问。

类变量：类变量也声明在类中，方法体之外，但必须声明为static类型。

创建对象

对象是根据类创建的。在Java中，使用关键字new来创建一个新的对象。创建对象需要以下三步：

声明：声明一个对象，包括对象名称和对象类型。

实例化：使用关键字new来创建一个对象。

初始化：使用new创建对象时，会调用构造方法初始化对象。

Java包

包主要用来对类和接口进行分类。当开发Java程序时，可能编写成百上千的类，因此很有必要对类和接口进行分类。

Import语句

在Java中，如果给出一个完整的限定名，包括包名、类名，那么Java编译器就可以很容易地定位到源代码或者类。Import语句就是用来提供一个合理的路径，使得编译器可以找到某个类。

例如，下面的命令行将会命令编译器载入java\_installation/java/io路径下的所有类

成员变量和类变量的区别：

1、两个变量的生命周期不同

成员变量随着对象的创建而存在，随着对象的回收而释放。

静态变量随着类的加载而存在，随着类的消失而消失。

2、调用方式不同

成员变量只能被对象调用。

静态变量可以被对象调用，还可以被类名调用。

3、别名不同

成员变量也称为实例变量。

静态变量也称为类变量。

4、数据存储位置不同

成员变量存储在堆内存的对象中，所以也叫对象的特有数据。

静态变量数据存储在方法区（共享数据区）的静态区，所以也叫对象的共享数据。

弊端：

1、有些数据是对象特有的数据，是不可以被静态修饰的。因为那样的话，特有数据会变成对象的共享数据。这样对事物的描述就出了问题。所以，在定义静态时，必须要明确，这个数据是否是被对象所共享的。

2、静态方法只能访问静态成员，不可以访问非静态成员。

因为静态方法加载时，优先于对象存在，所以没有办法访问对象中的成员。

3、静态方法中不能使用this，super关键字。

因为this代表对象，而静态在时，有可能没有对象，所以this无法使用。

弊端：

1、有些数据是对象特有的数据，是不可以被静态修饰的。因为那样的话，特有数据会变成对象的共享数据。这样对事物的描述就出了问题。所以，在定义静态时，必须要明确，这个数据是否是被对象所共享的。

2、静态方法只能访问静态成员，不可以访问非静态成员。

因为静态方法加载时，优先于对象存在，所以没有办法访问对象中的成员。

3、静态方法中不能使用this，super关键字。

因为this代表对象，而静态在时，有可能没有对象，所以this无法使用。

什么时候定义静态成员呢？或者说：定义成员时，到底需不需要被静态修饰呢？

成员分两种：

1、成员变量。（数据共享时静态化）

该成员变量的数据是否是所有对象都一样：

如果是，那么该变量需要被静态修饰，因为是共享的数据。

如果不是，那么就说这是对象的特有数据，要存储到对象中。

2、成员函数。（方法中没有调用特有数据时就定义成静态）

如果判断成员函数是否需要被静态修饰呢？

只要参考，该函数内是否访问了对象中的特有数据：

如果有访问特有数据，那方法不能被静态修饰。

如果没有访问过特有数据，那么这个方法需要被静态修饰。

成员变量和静态变量的区别：

1、成员变量所属于对象。所以也称为实例变量。

静态变量所属于类。所以也称为类变量。

2、成员变量存在于堆内存中。

静态变量存在于方法区中。

3、成员变量随着对象创建而存在。随着对象被回收而消失。

静态变量随着类的加载而存在。随着类的消失而消失。

4、成员变量只能被对象所调用 。

静态变量可以被对象调用，也可以被类名调用。

所以，成员变量可以称为对象的特有数据，静态变量称为对象的共享数据。

类的构造方法

1、构造方法的名字和类名相同，并且没有返回值。

2、构造方法主要用于为类的对象定义初始化状态。

3、我们不能直接调用构造方法，必须通过new关键字来自动调用，从而创建类的实例。

4、Java的类都要求有构造方法，如果没有定义构造方法，Java编译器会为我们提供一个缺省的构造方法，也就是不带参数的构造方法。

new关键字的作用

1、为对象分配内存空间。

2、引起对象构造方法的调用。

3、为对象返回一个引用。

上是 Oracle 对于静态的定义。大意为，有时候，你想拥有所有对象通用的变量。 这是通过静态修改器完成的。 在其声明中具有静态修饰符的字段称为静态字段或类变量。 他们与类相关，而不是与任何对象相关联

成员变量和局部变量区别

1.声明位置不同

成员变量也就是属性，在类中声明的。

局部变量，在方法中声明或代码块中声明。

2.初始值不同

成员变量如果没有赋值则是有默认值的，数据类型不同则默认值不同。

局部变量是没有默认值，也就是说必须先声明，再赋值，最后才使用。

3.在一个类中，局部变量可以与成员变量同名，但是局部变量优先,如果非要访问成员变量的属性，则必须使用 this.color

this 代表当前这个对象，也就是当前谁调用这个方法则这个对象就是谁。

//在类的内部使用自己的实例变量：要么老老实实new一个出来,就像上面那个

//第二种方法就是在函数里面使用实例变量，注意，如果方法是静态方法参照方法1

访问控制修饰符

Java中，可以使用访问控制符来保护对类、变量、方法和构造方法的访问。Java 支持 4 种不同的访问权限。

default (即缺省，什么也不写）: 在同一包内可见，不使用任何修饰符。使用对象：类、接口、变量、方法。

private : 在同一类内可见。使用对象：变量、方法。 注意：不能修饰类（外部类）

public : 对所有类可见。使用对象：类、接口、变量、方法

protected : 对同一包内的类和所有子类可见。使用对象：变量、方法。 注意：不能修饰类（外部类）。

使用默认访问修饰符声明的变量和方法，对同一个包内的类是可见的。接口里的变量都隐式声明为 public static final,而接口里的方法默认情况下访问权限为 public。

实例中，Logger 类中的 format 变量为私有变量，所以其他类不能直接得到和设置该变量的值。为了使其他类能够操作该变量，定义了两个 public 方法：getFormat() （返回 format的值）和 setFormat(String)（设置 format 的值）

Java 程序的 main() 方法必须设置成公有的，否则，Java 解释器将不能运行该类。

protected 可以修饰数据成员，构造方法，方法成员，不能修饰类（内部类除外）。

接口及接口的成员变量和成员方法不能声明为 protected。

如果我们只想让该方法对其所在类的子类可见，则将该方法声明为 protected。

父类中声明为 protected 的方法在子类中要么声明为 protected，要么声明为 public，不能声明为 private。

非访问修饰符

为了实现一些其他的功能，Java 也提供了许多非访问修饰符。

static 修饰符，用来修饰类方法和类变量。

final 修饰符，用来修饰类、方法和变量，final 修饰的类不能够被继承，修饰的方法不能被继承类重新定义，修饰的变量为常量，是不可修改的。

abstract 修饰符，用来创建抽象类和抽象方法。

synchronized 和 volatile 修饰符，主要用于线程的编程。

final 方法

类中的 final 方法可以被子类继承，但是不能被子类修改。

声明 final 方法的主要目的是防止该方法的内容被修改。

final 类

final 类不能被继承，没有类能够继承 final 类的任何特性。

abstract 修饰符

抽象类：

抽象类不能用来实例化对象，声明抽象类的唯一目的是为了将来对该类进行扩充。

一个类不能同时被 abstract 和 final 修饰。如果一个类包含抽象方法，那么该类一定要声明为抽象类，否则将出现编译错误。

抽象类可以包含抽象方法和非抽象方法。

抽象方法

抽象方法是一种没有任何实现的方法，该方法的的具体实现由子类提供。

抽象方法不能被声明成 final 和 static。

任何继承抽象类的子类必须实现父类的所有抽象方法，除非该子类也是抽象类。

如果一个类包含若干个抽象方法，那么该类必须声明为抽象类。抽象类可以不包含抽象方法。

抽象方法的声明以分号结尾，例如：public abstract sample();。

synchronized 修饰符

synchronized 关键字声明的方法同一时间只能被一个线程访问。

transient 修饰符

序列化的对象包含被 transient 修饰的实例变量时，java 虚拟机(JVM)跳过该特定的变量。

该修饰符包含在定义变量的语句中，用来预处理类和变量的数据类型。

volatile 修饰符

volatile 修饰的成员变量在每次被线程访问时，都强制从共享内存中重新读取该成员变量的值。而且，当成员变量发生变化时，会强制线程将变化值回写到共享内存。这样在任何时刻，两个不同的线程总是看到某个成员变量的同一个值。

一个 volatile 对象引用可能是 null。

static全局变量与普通的全局变量：static全局变量只初使化一次，防止在其他文件单元中被引用;

static局部变量和普通局部变量：static局部变量只被初始化一次，下一次依据上一次结果值；

static函数与普通函数：static函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝。

态变量并不是说其就不能改变值，不能改变值的量叫常量。 其拥有的值是可变的 ，而且它会保持最新的值。说其静态，是因为它不会随着函数的调用和退出而发生变化。即上次调用函数的时候，如果我们给静态变量赋予某个值的话，下次函数调用时，这个值保持不变。

静态方法只能使用类的静态变量。

transient

当对象被序列化时（写入字节序列到目标文件）时，transient阻止实例中那些用此关键字声明的变量持久化；当对象被反序列化时（从源文件读取字节序列进行重构），这样的实例变量值不会被持久化和恢复。

//因此成员变量属于对象独有，每个对象创建时只会调用一次构造函数，

//成员变量可以在构造函数中赋值，但是类变量却不可以。

2：static 和 abstract 也是不能连用的，例子：

abstract static void m(){}

原因：因为 static 是类级别的不能被子类覆盖，而 abstract 需要被继承实现，两者相互矛盾。

instanceof 运算符

该运算符用于操作对象实例，检查该对象是否是一个特定类型（类类型或接口类型）。

如果被比较的对象兼容于右侧类型,该运算符仍然返回true。

在判断一个实例引用的类型时，使用的是实际类型，而不是声明的类型。在下面的代码中，

Vehicle v2 = new Car(); // v2 是 Car 类型

v2 是 Car 类型，而不是 Vehicle 类型。

子类是父类的类型，但父类不是子类的类型。

子类的实例可以声明为父类型，但父类的实例不能声明为子类型。

加法的结合顺序 左->右 所以左侧计算完毕后才会计算右侧 因此结果不同:

public static void main(String[] args) {

int a=2;

System.out.println(--a/2+(++a\*2));

System.out.println(++a\*2+--a/2);

}

结果：

4

7

虽然所有循环结构都可以用 while 或者 do...while表示，但 Java 提供了另一种语句 —— for 循环，使一些循环结构变得更加简单。

for循环执行的次数是在执行前就确定的。语法格式如下：

关于 for 循环有以下几点说明：

最先执行初始化步骤。可以声明一种类型，但可初始化一个或多个循环控制变量，也可以是空语句。

然后，检测布尔表达式的值。如果为 true，循环体被执行。如果为false，循环终止，开始执行循环体后面的语句。

执行一次循环后，更新循环控制变量。

再次检测布尔表达式。循环执行上面的过程。

Java5 引入了一种主要用于数组的增强型 for 循环。

Java 增强 for 循环语法格式如下:

for(声明语句 : 表达式)

{

//代码句子

}

声明语句：声明新的局部变量，该变量的类型必须和数组元素的类型匹配。其作用域限定在循环语句块，其值与此时数组元素的值相等。

表达式：表达式是要访问的数组名，或者是返回值为数组的方法。

实例

Test.java 文件代码：

public class Test {

public static void main(String args[]){

int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ){

System.out.print( x );

System.out.print(",");

}

System.out.print("\n");

String [] names ={"James", "Larry", "Tom", "Lacy"};

for( String name : names ) {

System.out.print( name );

System.out.print(",");

}

}

}

以上实例编译运行结果如下：

10,20,30,40,50,

James,Larry,Tom,Lacy,

跳出循环体的语句 break 只能跳出当前语句的上一层循环。

看输出结果确实达到了只是输出 0-29，但是你会发现和题目要求的输出结果是不一样的,后面多出了很多分隔线条，那是因为跳出循环体的语句 break 只能跳出当前语句的上一层循环。

这时我们需要想办法让他跳出多层循环。

修改后代码如下：

public class Test {

public static void main(String []args) {

lable:

for(int i = 0; i < 10; i++){

for(int j = 0; j < 10; j++){

if(i \* 10 + j > 29){

break lable;

}

System.out.print("" + i + j +" ");

}

System.out.println("\n -------------------------------------- \n");

}

System.out.println("输出完毕！");

}

}

输出：

00 01 02 03 04 05 06 07 08 09

--------------------------------------

10 11 12 13 14 15 16 17 18 19

--------------------------------------

20 21 22 23 24 25 26 27 28 29

--------------------------------------

输出完毕！

我们在第 3 行定义了一个标记 标记名是 lable。

然后在第 7 行的跳出循环语句中使用 break lable 跳出到 lable 标记的循环外面。

if...else if...else 语句

if 语句后面可以跟 else if…else 语句，这种语句可以检测到多种可能的情况。

使用 if，else if，else 语句的时候，需要注意下面几点：

if 语句至多有 1 个 else 语句，else 语句在所有的 else if 语句之后。

if 语句可以有若干个 else if 语句，它们必须在 else 语句之前。

一旦其中一个 else if 语句检测为 true，其他的 else if 以及 else 语句都将跳过执行。

switch 语句可以包含一个 default 分支，该分支一般是 switch 语句的最后一个分支（可以在任何位置，但建议在最后一个）。default 在没有 case 语句的值和变量值相等的时候执行。default 分支不需要 break 语句。

装箱就是自动将基本数据类型转换为包装器类型；拆箱就是自动将包装器类型转换为基本数据类型。

Integer i1 = 128; // 装箱，相当于 Integer.valueOf(128);

int t = i1; //相当于 i1.intValue() 拆箱

注意 == 与 equals的区别

== 它比较的是对象的地址（针对Integer类型，对int类型无效）

equals 比较的是对象的内容

（1）Java 会对 -128 ~ 127 的整数进行缓存，所以当定义两个变量初始化值位于 -128 ~ 127 之间时，两个变量使用了同一地址：

Integer a=123;

Integer b=123;

System.out.println(a==b); // 输出 true

System.out.println(a.equals(b)); // 输出 true

（2）当两个 Integer 变量的数值超出 -128 ~ 127 范围时, 变量使用了不同地址：

Java 中 int 和 Integer 的区别

1. int 是基本数据类型，int 变量存储的是数值。Integer 是引用类型，实际是一个对象，Integer 存储的是引用对象的地址。

int 和 Integer 所占内存比较：

Integer 对象会占用更多的内存。Integer 是一个对象，需要存储对象的元数据。但是 int 是一个原始类型的数据，所以占用的空间更少。

4. 非 new 生成的 Integer 变量与 new Integer() 生成的变量比较，结果为 false。

因为非 new 生成的 Integer 变量指向的是 java 常量池中的对象，而 new Integer() 生成的变量指向堆中新建的对象，两者在内存中的地址不同。所以输出为 false。

5. 两个非 new 生成的 Integer 对象进行比较，如果两个变量的值在区间 [-128,127] 之间，比较结果为 true；否则，结果为 false。

创建格式化字符串

我们知道输出格式化数字可以使用 printf() 和 format() 方法。

String 类使用静态方法 format() 返回一个String 对象而不是 PrintStream 对象。

String 类的静态方法 format() 能用来创建可复用的格式化字符串，而不仅仅是用于一次打印输出。

char[] toCharArray()

将此字符串转换为一个新的字符数组。

实例中的 s 只是一个 String 对象的引用，并不是对象本身，当执行 s = "Runoob"; 创建了一个新的对象 "Runoob"，而原来的 "Google" 还存在于内存中。

length() 方法，length 属性和 size() 方法的区别:

1、length() 方法是针对字符串来说的，要求一个字符串的长度就要用到它的length()方法；

2、length 属性是针对 Java 中的数组来说的，要求数组的长度可以用其 length 属性；

3、Java 中的 size() 方法是针对泛型集合说的, 如果想看这个泛型有多少个元素, 就调用此方法来查看!

关于 String 为啥是不可改变的

这里可以根据 jdk 的源码来分析。

字符串实际上就是一个 char 数组，并且内部就是封装了一个 char 数组。

并且这里 char 数组是被 final 修饰的:

当一个字符串是一个字面量时，它会被放到一个常量池中，等待复用。

String a = "saff";

String b = "saff";

String c = new String("saff");

System.out.println(a.equal(b)); // true

System.out.println(a.equal(c)); // true

这个就是字符串的常量池。

String s1 = "abc"; // 常量池

String s2 = new String("abc"); // 堆内存中

System.out.println(s1==s2); // false两个对象的地址值不一样。

System.out.println(s1.equals(s2)); // true

String s1="a"+"b"+"c";

String s2="abc";

System.out.println(s1==s2);

System.out.println(s1.equals(s2));

java 中常量优化机制，编译时 s1 已经成为 abc 在常量池中查找创建，s2 不需要再创建。

String s1="ab";

String s2="abc";

String s3=s1+"c";

System.out.println(s3==s2); // false

System.out.println(s3.equals(s2)); // true

先在常量池中创建 ab ，地址指向 s1, 再创建 abc ，指向 s2。对于 s3，先创建StringBuilder（或 StringBuffer）对象，通过 append 连接得到 abc ,再调用 toString() 转换得到的地址指向 s3。故 (s3==s2) 为 false。

通过 new 关键字来生成对象是在堆区进行的，而在堆区进行对象生成的过程是不会去检测该对象是否已经存在的。因此通过 new 来创建对象，创建出的一定是不同的对象，即使字符串的内容是相同的。

你可以在一个方法里，不同的非嵌套块中多次声明一个具有相同的名称局部变量，但你不能在嵌套块内两次声明局部变量。

构造方法

当一个对象被创建时候，构造方法用来初始化该对象。构造方法和它所在类的名字相同，但构造方法没有返回值。

通常会使用构造方法给一个类的实例变量赋初值，或者执行其它必要的步骤来创建一个完整的对象。

不管你是否自定义构造方法，所有的类都有构造方法，因为Java自动提供了一个默认构造方法，默认构造方法的访问修改符和类的访问修改符相同(类为 public，构造函数也为 public；类改为 private，构造函数也改为 private)。

一旦你定义了自己的构造方法，默认构造方法就会失效。

大多时候需要一个有参数的构造方法。

finalize() 方法

Java 允许定义这样的方法，它在对象被垃圾收集器析构(回收)之前调用，这个方法叫做 finalize( )，它用来清除回收对象。

例如，你可以使用 finalize() 来确保一个对象打开的文件被关闭了。

在 finalize() 方法里，你必须指定在对象销毁时候要执行的操作。

finalize() 一般格式是：

protected void finalize()

{

// 在这里终结代码

}

关键字 protected 是一个限定符，它确保 finalize() 方法不会被该类以外的代码调用。

当然，Java 的内存回收可以由 JVM 来自动完成。如果你手动使用，则可以使用上面的方法。

一个函数至多只能有一个可变参数，且可变参数为最后一个参数。对于可变参数，编译器会将其转型为一个数组，故在函数内部，可变参数名即可看作数组名。

next():

1、一定要读取到有效字符后才可以结束输入。

2、对输入有效字符之前遇到的空白，next() 方法会自动将其去掉。

3、只有输入有效字符后才将其后面输入的空白作为分隔符或者结束符。

next() 不能得到带有空格的字符串。

nextLine()：

1、以Enter为结束符,也就是说 nextLine()方法返回的是输入回车之前的所有字符。

2、可以获得空白。

如果要输入 int 或 float 类型的数据，在 Scanner 类中也有支持，但是在输入之前最好先使用 hasNextXxx() 方法进行验证，再使用 nextXxx() 来读取：

输入的时候字符都是可见的，所以Scanner类不适合从控制台读取密码。从Java SE 6开始特别引入了Console类来实现这个目的。若要读取一个密码，可以采用下面这段代码：

Console cons = System.console();

String username = cons.readline("User name: ");

char[] passwd = cons.readPassword("Password: ");

Scanner 不仅能从输入流中读取，也能从文件中读取，除了构建 Scanner 对象的方法，其他和上文给出的完全相同，以下案例从一个名为 test.txt 的文件中读取整数。

Scanner sc=new Scanner(new File("test.txt"));

FileOutputStream

该类用来创建一个文件并向文件中写数据。

如果该流在打开文件进行输出前，目标文件不存在，那么该流会创建该文件。

有两个构造方法可以用来创建 FileOutputStream 对象。

Java中的目录

创建目录：

File类中有两个方法可以用来创建文件夹：

mkdir( )方法创建一个文件夹，成功则返回true，失败则返回false。失败表明File对象指定的路径已经存在，或者由于整个路径还不存在，该文件夹不能被创建。

mkdirs()方法创建一个文件夹和它的所有父文件夹。

BufferedReader 是支持同步的，而 Scanner 不支持。如果我们处理多线程程序，BufferedReader 应当使用。

Catch 语句包含要捕获异常类型的声明。当保护代码块中发生一个异常时，try 后面的 catch 块就会被检查。

如果发生的异常包含在 catch 块中，异常会被传递到该 catch 块，这和传递一个参数到方法是一样。

throws/throw 关键字：

如果一个方法没有捕获到一个检查性异常，那么该方法必须使用 throws 关键字来声明。throws 关键字放在方法签名的尾部。

也可以使用 throw 关键字抛出一个异常，无论它是新实例化的还是刚捕获到的。

下面方法的声明抛出一个 RemoteException 异常：

import java.io.\*;

public class className

{

public void deposit(double amount) throws RemoteException

{

// Method implementation

throw new RemoteException();

}

//Remainder of class definition

}

finally关键字

finally 关键字用来创建在 try 代码块后面执行的代码块。

无论是否发生异常，finally 代码块中的代码总会被执行。

在 finally 代码块中，可以运行清理类型等收尾善后性质的语句。

注意下面事项：

catch 不能独立于 try 存在。

在 try/catch 后面添加 finally 块并非强制性要求的。

try 代码后不能既没 catch 块也没 finally 块。

try, catch, finally 块之间不能添加任何代码。

声明自定义异常

在 Java 中你可以自定义异常。编写自己的异常类时需要记住下面的几点。

所有异常都必须是 Throwable 的子类。

如果希望写一个检查性异常类，则需要继承 Exception 类。

如果你想写一个运行时异常类，那么需要继承 RuntimeException 类。

可以像下面这样定义自己的异常类：

class MyException extends Exception{

}

finally不一定被执行，，例如 catch 块中有退出系统的语句 System.exit(-1); finally就不会被执行

throws 用在函数上，声明该函数的功能可能会出现问题。

将异常抛出，是问题暴露出来，用于处理。

可以抛给虚拟机处理，或者使用 try....catch... 进行处理。虚拟机的处理方式，就是将异常打印出来，并且将在异常处的代码终止。

throw 用在代码块中，后面跟着异常的对象，该对象可以是自定义异常，且 throw 使用在方法中。

继承就是子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为。

所以继承需要符合的关系是：is-a，父类更通用，子类更具体。

虽然食草动物和食肉动物都是属于动物，但是两者的属性和行为上有差别，所以子类会具有父类的一般特性也会具有自身的特性。

类的继承格式

在 Java 中通过 extends 关键字可以申明一个类是从另外一个类继承而来的，一般形式如下：

继承的特性

子类拥有父类非 private 的属性、方法。

子类可以拥有自己的属性和方法，即子类可以对父类进行扩展。

子类可以用自己的方式实现父类的方法。

Java 的继承是单继承，但是可以多重继承，单继承就是一个子类只能继承一个父类，多重继承就是，例如 A 类继承 B 类，B 类继承 C 类，所以按照关系就是 C 类是 B 类的父类，B 类是 A 类的父类，这是 Java 继承区别于 C++ 继承的一个特性。

提高了类之间的耦合性（继承的缺点，耦合度高就会造成代码之间的联系越紧密，代码独立性越差）。

继承关键字

继承可以使用 extends 和 implements 这两个关键字来实现继承，而且所有的类都是继承于 java.lang.Object，当一个类没有继承的两个关键字，则默认继承object（这个类在 java.lang 包中，所以不需要 import）祖先类。

implements关键字

使用 implements 关键字可以变相的使java具有多继承的特性，使用范围为类继承接口的情况，可以同时继承多个接口（接口跟接口之间采用逗号分隔）。

super 与 this 关键字

super关键字：我们可以通过super关键字来实现对父类成员的访问，用来引用当前对象的父类。

this关键字：指向自己的引用。

inal关键字

final 关键字声明类可以把类定义为不能继承的，即最终类；或者用于修饰方法，该方法不能被子类重写：

注:实例变量也可以被定义为 final，被定义为 final 的变量不能被修改。被声明为 final 类的方法自动地声明为 final，但是实例变量并不是 final

构造器

子类是不继承父类的构造器（构造方法或者构造函数）的，它只是调用（隐式或显式）。如果父类的构造器带有参数，则必须在子类的构造器中显式地通过 super 关键字调用父类的构造器并配以适当的参数列表。

如果父类构造器没有参数，则在子类的构造器中不需要使用 super 关键字调用父类构造器，系统会自动调用父类的无参构造器。

因此可以定义一个通用类，然后将其扩展为其它多个特定类，这些特定类继承通用类中的特征和动作。

继承的注意点：

子类不是父类的子集，子类一般比父类包含更多的数据域和方法。

父类中的 private 数据域在子类中是不可见的，因此在子类中不能直接使用它们。

3、super 关键字

super 表示使用它的类的父类。super 可用于：

调用父类的构造方法；

调用父类的方法（子类覆盖了父类的方法时）；

访问父类的数据域（可以这样用但没有必要这样用）

super 语句必须是子类构造方法的第一条语句。不能在子类中使用父类构造方法名来调用父类构造方法。 父类的构造方法不被子类继承。调用父类的构造方法的唯一途径是使用 super 关键字，如果子类中没显式调用，则编译器自动将 super(); 作为子类构造方法的第一条语句。这会形成一个构造方法链。

静态方法中不能使用 super 关键字。

调用父类的方法语法：

super.方法名(参数列表);

如果是继承的方法，是没有必要使用 super 来调用，直接即可调用。但如果子类覆盖或重写了父类的方法，则只有使用 super 才能在子类中调用父类中的被重写的方法。

4、this 关键字

this 关键字表示当前对象。可用于：

调用当前类的构造方法，并且必须是方法的第一条语句。如：this(); 调用默认构造方法。this(参数); 调用带参构造方法。

限定当前对象的数据域变量。一般用于方法内的局部变量与对象的数据域变量同名的情况。如 this.num = num。this.num 表示当前对象的数据域变量 num，而 num 表示方法中的局部变量。

final 的作用随着所修饰的类型而不同

1、final 修饰类中的属性或者变量

无论属性是基本类型还是引用类型，final 所起的作用都是变量里面存放的"值"不能变。

这个值，对于基本类型来说，变量里面放的就是实实在在的值，如 1，"abc" 等。

而引用类型变量里面放的是个地址，所以用 final 修饰引用类型变量指的是它里面的地址不能变，并不是说这个地址所指向的对象或数组的内容不可以变，这个一定要注意。

例如：类中有一个属性是 final Person p=new Person("name"); 那么你不能对 p 进行重新赋值，但是可以改变 p 里面属性的值 p.setName('newName');

final 修饰属性，声明变量时可以不赋值，而且一旦赋值就不能被修改了。对 final 属性可以在三个地方赋值：声明时、初始化块中、构造方法中，总之一定要赋值。

2、final修饰类中的方法

作用：可以被继承，但继承后不能被重写。

3、final修饰类

作用：类不可以被继承。

java文件被编译成class文件时，在子类的所有构造函数中的第一行（第一个语句）会默认自动添加 super() 语句，在执行子类的构造函数前，总是会先执行父类中的构造函数。

在编写代码要注意：

1.如果父类中不含 默认构造函数（就是 类名() ），那么子类中的super()语句就会执行失败，系统就会报错。一般 默认构造函数 编译时会自动添加，但如果类中已经有一个构造函数时，就不会添加。

2.执行父类构造函数的语句只能放在函数内语句的首句，不然会报错。

在继承关系中，在调用函数（方法）或者类中的成员变量时，JVM（JAVA虚拟机）会先检测当前的类（也就是子类）是否含有该函数或者成员变量，如果有，就执行子类中的，如果没有才会执行父类中的。如下：

当子类出现与父类一样的函数时，这个被称为 重写 也叫 覆盖

注意：如果父类没有无参构造函数，创建子类时，不能编译，除非在构造函数代码体中的第一行显式调用父类有参构造函数。

子类不能直接继承父类中的 private 属性和方法。

Father f1 = new Son(); // 这就叫 upcasting （向上转型)

// 现在 f1 引用指向一个Son对象

Son s1 = (Son)f1; // 这就叫 downcasting (向下转型)

// 现在f1 还是指向 Son对象

Father f2 = new Father();

Son s2 = (Son)f2; // 出错，子类引用不能指向父类对象

1、父类引用指向子类对象，而子类引用不能指向父类对象。

2、把子类对象直接赋给父类引用叫upcasting向上转型，向上转型不用强制转换吗，如：

在面向对象的概念中，所有的对象都是通过类来描绘的，但是反过来，并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类。

抽象类除了不能实例化对象之外，类的其它功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。

由于抽象类不能实例化对象，所以抽象类必须被继承，才能被使用。也是因为这个原因，通常在设计阶段决定要不要设计抽象类。

父类包含了子类集合的常见的方法，但是由于父类本身是抽象的，所以不能使用这些方法。

在Java中抽象类表示的是一种继承关系，一个类只能继承一个抽象类，而一个类却可以实现多个接口。

抽象方法

如果你想设计这样一个类，该类包含一个特别的成员方法，该方法的具体实现由它的子类确定，那么你可以在父类中声明该方法为抽象方法。

Abstract 关键字同样可以用来声明抽象方法，抽象方法只包含一个方法名，而没有方法体。

抽象方法没有定义，方法名后面直接跟一个分号，而不是花括号。

声明抽象方法会造成以下两个结果：

如果一个类包含抽象方法，那么该类必须是抽象类。

任何子类必须重写父类的抽象方法，或者声明自身为抽象类。

继承抽象方法的子类必须重写该方法。否则，该子类也必须声明为抽象类。最终，必须有子类实现该抽象方法，否则，从最初的父类到最终的子类都不能用来实例化对象。

抽象类总结规定

1. 抽象类不能被实例化(初学者很容易犯的错)，如果被实例化，就会报错，编译无法通过。只有抽象类的非抽象子类可以创建对象。

2. 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。

3. 抽象类中的抽象方法只是声明，不包含方法体，就是不给出方法的具体实现也就是方法的具体功能。

4. 构造方法，类方法（用 static 修饰的方法）不能声明为抽象方法。

5. 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。

子抽象类继承父抽象类，必须显性调用父抽象类的显性构造器

并不是说"一定要调用父类的显性构造器"，而是子类在继承父类时，如果父类的显式构造器中有参数，子类要声明给出这个参数。这是一个关于继承的问题。

2、设计层面上的区别

1）抽象类是对一种事物的抽象，即对类抽象，而接口是对行为的抽象。抽象类是对整个类整体进行抽象，包括属性、行为，但是接口却是对类局部（行为）进行抽象。举个简单的例子，飞机和鸟是不同类的事物，但是它们都有一个共性，就是都会飞。那么在设计的时候，可以将飞机设计为一个类 Airplane，将鸟设计为一个类 Bird，但是不能将 飞行 这个特性也设计为类，因此它只是一个行为特性，并不是对一类事物的抽象描述。此时可以将 飞行 设计为一个接口Fly，包含方法fly( )，然后Airplane和Bird分别根据自己的需要实现Fly这个接口。然后至于有不同种类的飞机，比如战斗机、民用飞机等直接继承Airplane即可，对于鸟也是类似的，不同种类的鸟直接继承Bird类即可。从这里可以看出，继承是一个 "是不是"的关系，而 接口 实现则是 "有没有"的关系。如果一个类继承了某个抽象类，则子类必定是抽象类的种类，而接口实现则是有没有、具备不具备的关系，比如鸟是否能飞（或者是否具备飞行这个特点），能飞行则可以实现这个接口，不能飞行就不实现这个接口。

2）设计层面不同，抽象类作为很多子类的父类，它是一种模板式设计。而接口是一种行为规范，它是一种辐射式设计。什么是模板式设计？最简单例子，大家都用过 ppt 里面的模板，如果用模板 A 设计了 ppt B 和 ppt C，ppt B 和 ppt C 公共的部分就是模板 A 了，如果它们的公共部分需要改动，则只需要改动模板 A 就可以了，不需要重新对 ppt B 和 ppt C 进行改动。而辐射式设计，比如某个电梯都装了某种报警器，一旦要更新报警器，就必须全部更新。也就是说对于抽象类，如果需要添加新的方法，可以直接在抽象类中添加具体的实现，子类可以不进行变更；而对于接口则不行，如果接口进行了变更，则所有实现这个接口的类都必须进行相应的改动。

Door 的 open() 、close() 和 alarm() 根本就属于两个不同范畴内的行为，open() 和 close() 属于门本身固有的行为特性，而 alarm() 属于延伸的附加行为。因此最好的解决办法是单独将报警设计为一个接口，包含 alarm() 行为，Door 设计为单独的一个抽象类，包含 open 和 close 两种行为。再设计一个报警门继承 Door 类和实现 Alarm 接口。

Java 封装

在面向对象程式设计方法中，封装（英语：Encapsulation）是指一种将抽象性函式接口的实现细节部份包装、隐藏起来的方法。

封装可以被认为是一个保护屏障，防止该类的代码和数据被外部类定义的代码随机访问。

要访问该类的代码和数据，必须通过严格的接口控制。

封装最主要的功能在于我们能修改自己的实现代码，而不用修改那些调用我们代码的程序片段。

适当的封装可以让程式码更容易理解与维护，也加强了程式码的安全性。

封装的优点

1. 良好的封装能够减少耦合。

2. 类内部的结构可以自由修改。

3. 可以对成员变量进行更精确的控制。

4. 隐藏信息，实现细节。

实现Java封装的步骤

1. 修改属性的可见性来限制对属性的访问（一般限制为private），

这段代码中，将 name 和 age 属性设置为私有的，只能本类才能访问，其他类都访问不了，如此就对信息进行了隐藏。

2. 对每个值属性提供对外的公共方法访问，也就是创建一对赋取值方法，用于对私有属性的访问，

采用 this 关键字是为了解决实例变量（private String name）和局部变量（setName(String name)中的name变量）之间发生的同名的冲突。

Java 接口

接口（英文：Interface），在JAVA编程语言中是一个抽象类型，是抽象方法的集合，接口通常以interface来声明。一个类通过继承接口的方式，从而来继承接口的抽象方法。

接口并不是类，编写接口的方式和类很相似，但是它们属于不同的概念。类描述对象的属性和方法。接口则包含类要实现的方法。

除非实现接口的类是抽象类，否则该类要定义接口中的所有方法。

接口无法被实例化，但是可以被实现。一个实现接口的类，必须实现接口内所描述的所有方法，否则就必须声明为抽象类。另外，在 Java 中，接口类型可用来声明一个变量，他们可以成为一个空指针，或是被绑定在一个以此接口实现的对象。

接口与类相似点：

一个接口可以有多个方法。

接口文件保存在 .java 结尾的文件中，文件名使用接口名。

接口的字节码文件保存在 .class 结尾的文件中。

接口相应的字节码文件必须在与包名称相匹配的目录结构中。

接口与类的区别：

接口不能用于实例化对象。

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 static 和 final 变量。

接口不是被类继承了，而是要被类实现。

接口支持多继承。

接口特性

接口中每一个方法也是隐式抽象的,接口中的方法会被隐式的指定为 public abstract（只能是 public abstract，其他修饰符都会报错）。

接口中可以含有变量，但是接口中的变量会被隐式的指定为 public static final 变量（并且只能是 public，用 private 修饰会报编译错误）。

接口中的方法是不能在接口中实现的，只能由实现接口的类来实现接口中的方法。

抽象类和接口的区别

1. 抽象类中的方法可以有方法体，就是能实现方法的具体功能，但是接口中的方法不行。

2. 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是 public static final 类型的。

3. 接口中不能含有静态代码块以及静态方法(用 static 修饰的方法)，而抽象类是可以有静态代码块和静态方法。

4. 一个类只能继承一个抽象类，而一个类却可以实现多个接口。

接口的声明语法格式如下：

[可见度] interface 接口名称 [extends 其他的接口名] {

// 声明变量

// 抽象方法

}

接口有以下特性：

接口是隐式抽象的，当声明一个接口的时候，不必使用abstract关键字。

接口中每一个方法也是隐式抽象的，声明时同样不需要abstract关键字。

接口中的方法都是公有的。

接口的实现

当类实现接口的时候，类要实现接口中所有的方法。否则，类必须声明为抽象的类。

类使用implements关键字实现接口。在类声明中，Implements关键字放在class声明后面。

实现一个接口的语法，可以使用这个公式：

Animal.java 文件代码：

...implements 接口名称[, 其他接口名称, 其他接口名称..., ...] ...

重写接口中声明的方法时，需要注意以下规则：

类在实现接口的方法时，不能抛出强制性异常，只能在接口中，或者继承接口的抽象类中抛出该强制性异常。

类在重写方法时要保持一致的方法名，并且应该保持相同或者相兼容的返回值类型。

如果实现接口的类是抽象类，那么就没必要实现该接口的方法。

在实现接口的时候，也要注意一些规则：

一个类可以同时实现多个接口。

一个类只能继承一个类，但是能实现多个接口。

一个接口能继承另一个接口，这和类之间的继承比较相似。

标记接口

最常用的继承接口是没有包含任何方法的接口。

标记接口是没有任何方法和属性的接口.它仅仅表明它的类属于一个特定的类型,供其他代码来测试允许做一些事情。

标记接口作用：简单形象的说就是给某个对象打个标（盖个戳），使对象拥有某个或某些特权。

没有任何方法的接口被称为标记接口。标记接口主要用于以下两种目的：

建立一个公共的父接口：

正如EventListener接口，这是由几十个其他接口扩展的Java API，你可以使用一个标记接口来建立一组接口的父接口。例如：当一个接口继承了EventListener接口，Java虚拟机(JVM)就知道该接口将要被用于一个事件的代理方案。

向一个类添加数据类型：

这种情况是标记接口最初的目的，实现标记接口的类不需要定义任何接口方法(因为标记接口根本就没有方法)，但是该类通过多态性变成一个接口类型。

什么时候使用抽象类和接口

如果你拥有一些方法并且想让它们中的一些有默认实现，那么使用抽象类吧。

如果你想实现多重继承，那么你必须使用接口。由于Java不支持多继承，子类不能够继承多个类，但可以实现多个接口。因此你就可以使用接口来解决它。

如果基本功能在不断改变，那么就需要使用抽象类。如果不断改变基本功能并且使用接口，那么就需要改变所有实现了该接口的类。

在 JDK1.8，允许我们给接口添加两种非抽象的方法实现：

1、默认方法，添加 default 修饰即可；

2、静态方法，使用 static 修饰；示例如下：

interface Test{

//这个是默认方法

default String get(String aa){

System.out.println("我是jdk1.8默认实现方法...");

return "";

}

//这个是静态方法

static void staticmethod(){

System.out.println("我是静态方法");

}

}

调用得话，静态方法只能通过接口名调用，不可以通过实现类的类名或者实现类的对象调用，default 方法只能通过接口实现类的对象来调用。

接口的含义理解

接口可以理解成统一的协议, 而接口中的属性也属于协议中的内容。但是接口的属性都是公共的，静态的，最终的。

接口的成员特点：

1、成员变量只能是常量，默认修饰符 public static final

2、成员方法只能是抽象方法。默认修饰符 public abstract

所以，Java 接口中，使用变量的时候，变量必须被赋值。

使用标记接口的唯一目的是使得可以用 instanceof 进行类型查询，例如：

if(obj instanceof Cloneable) {………}

Java 包(package)

为了更好地组织类，Java 提供了包机制，用于区别类名的命名空间。

包的作用

1、把功能相似或相关的类或接口组织在同一个包中，方便类的查找和使用。

2、如同文件夹一样，包也采用了树形目录的存储方式。同一个包中的类名字是不同的，不同的包中的类的名字是可以相同的，当同时调用两个不同包中相同类名的类时，应该加上包名加以区别。因此，包可以避免名字冲突。

3、包也限定了访问权限，拥有包访问权限的类才能访问某个包中的类。

Java 使用包（package）这种机制是为了防止命名冲突，访问控制，提供搜索和定位类（class）、接口、枚举（enumerations）和注释（annotation）等。

以下是一些 Java 中的包：

java.lang-打包基础的类

java.io-包含输入输出功能的函数

开发者可以自己把一组类和接口等打包，并定义自己的包。而且在实际开发中这样做是值得提倡的，当你自己完成类的实现之后，将相关的类分组，可以让其他的编程者更容易地确定哪些类、接口、枚举和注释等是相关的。

由于包创建了新的命名空间（namespace），所以不会跟其他包中的任何名字产生命名冲突。使用包这种机制，更容易实现访问控制，并且让定位相关类更加简单。

创建包

创建包的时候，你需要为这个包取一个合适的名字。之后，如果其他的一个源文件包含了这个包提供的类、接口、枚举或者注释类型的时候，都必须将这个包的声明放在这个源文件的开头。

包声明应该在源文件的第一行，每个源文件只能有一个包声明，这个文件中的每个类型都应用于它。

如果一个源文件中没有使用包声明，那么其中的类，函数，枚举，注释等将被放在一个无名的包（unnamed package）中。

创建了一个叫做animals的包。通常使用小写的字母来命名避免与类、接口名字的冲突。

import 关键字

为了能够使用某一个包的成员，我们需要在 Java 程序中明确导入该包。使用 "import" 语句可完成此功能。

在 java 源文件中 import 语句应位于 package 语句之后，所有类的定义之前，可以没有，也可以有多条，其语法格式为：

import package1[.package2…].(classname|\*);

如果在一个包中，一个类想要使用本包中的另一个类，那么该包名可以省略。

如果 Boss 类不在 payroll 包中又会怎样？Boss 类必须使用下面几种方法之一来引用其他包中的类。

使用类全名描述，例如：

payroll.Employee

用 import 关键字引入，使用通配符 "\*"

import payroll.\*;

使用 import 关键字引入 Employee 类:

import payroll.Employee;

类文件中可以包含任意数量的 import 声明。import 声明必须在包声明之后，类声明之前。

package 的目录结构

类放在包中会有两种主要的结果：

包名成为类名的一部分，正如我们前面讨论的一样。

包名必须与相应的字节码所在的目录结构相吻合。

通常，一个公司使用它互联网域名的颠倒形式来作为它的包名.例如：互联网域名是 runoob.com，所有的包名都以 com.runoob 开头。包名中的每一个部分对应一个子目录。

并不要求 .class 文件的路径跟相应的 .java 的路径一样。你可以分开来安排源码和类的目录。

设置 CLASSPATH 系统变量

用下面的命令显示当前的CLASSPATH变量：

Windows 平台（DOS 命令行下）：C:\> set CLASSPATH

UNIX 平台（Bourne shell 下）：# echo $CLASSPATH

删除当前CLASSPATH变量内容：

Windows 平台（DOS 命令行下）：C:\> set CLASSPATH=

UNIX 平台（Bourne shell 下）：# unset CLASSPATH; export CLASSPATH

设置CLASSPATH变量:

Windows 平台（DOS 命令行下）： C:\> set CLASSPATH=C:\users\jack\java\classes

UNIX 平台（Bourne shell 下）：# CLASSPATH=/home/jack/java/classes; export CLASSPATH

Enumeration接口中定义了一些方法，通过这些方法可以枚举（一次获得一个）对象集合中的元素。

这种传统接口已被迭代器取代，虽然Enumeration 还未被遗弃，但在现代代码中已经被很少使用了。尽管如此，它还是使用在诸如Vector和Properties这些传统类所定义的方法中，除此之外，还用在一些API类，并且在应用程序中也广泛被使用。

boolean hasMoreElements( )

测试此枚举是否包含更多的元素。

Object nextElement( )

如果此枚举对象至少还有一个可提供的元素，则返回此枚举的下一个元素。

算法：是实现集合接口的对象里的方法执行的一些有用的计算，例如：搜索和排序。这些算法被称为多态，那是因为相同的方法可以在相似的接口上有着不同的实现。

ArrayList

该类也是实现了List的接口，实现了可变大小的数组，随机访问和遍历元素时，提供更好的性能。该类也是非同步的,在多线程的情况下不要使用。ArrayList 增长当前长度的50%，插入删除效率低。

三种方法都是用来遍历ArrayList集合，第三种方法是采用迭代器的方法，该方法可以不用担心在遍历的过程中会超出集合的长度。

ArrayList 和 LinkedList 的区别

ArrayList 是 List 接口的一种实现，它是使用数组来实现的。

LinkedList 是 List 接口的一种实现，它是使用链表来实现的。

ArrayList 遍历和查找元素比较快。LinkedList 遍历和查找元素比较慢。

ArrayList 添加、删除元素比较慢。LinkedList 添加、删除元素比较快。

有关于 map.entrySet() 和 keySet():

1、如果遍历 hashMap() 时 entrySet() 方法是将 key 和 value 全部取出来,所以性能开销是可以预计的, 而 keySet() 方法进行遍历的时候是根据取出的 key 值去查询对应的 value 值, 所以如果 key 值是比较简单的结构(如 1,2,3...)的话性能消耗上是比 entrySet() 方法低, 但随着 key 值得复杂度提高 entrySet() 的优势就会显露出来。

2、综合比较在只遍历 key 的时候使用 keySet(), 在只遍历 value 的是使用 values() 方法, 在遍历 key-value 的时候使用 entrySet() 是比较合理的选择。

3、如果遍历 TreeMap 的时候, 不同于 HashMap 在遍历 ThreeMap 的 key-value 时候务必使用 entrySet() 它要远远高于其他两个的性能, 同样只遍历 key 的时候使用 keySet(), 在只遍历 value 的是使用 values() 方法对于 TreeMap 也同样适用。

算法：是实现集合接口的对象里的方法执行的一些有用的计算，例如：搜索和排序。这些算法被称为多态，那是因为相同的方法可以在相似的接口上有着不同的实现。

1. Set 接口实例存储的是无序的，不重复的数据。List 接口实例存储的是有序的，可以重复的元素。

2. Set检索效率低下，删除和插入效率高，插入和删除不会引起元素位置改变 <实现类有HashSet,TreeSet>。

3. List和数组类似，可以动态增长，根据实际存储的数据的长度自动增长List的长度。查找元素效率高，插入删除效率低，因为会引起其他元素位置改变 <实现类有ArrayList,LinkedList,Vector> 。

要声明一个有界的类型参数，首先列出类型参数的名称，后跟extends关键字，最后紧跟它的上界。

<? extends T>和<? super T>的区别

<? extends T>表示该通配符所代表的类型是T类型的子类。

<? super T>表示该通配符所代表的类型是T类型的父类。

如果你想知道一个 Java 标准类是否是可序列化的，请查看该类的文档。检验一个类的实例是否能序列化十分简单， 只需要查看该类有没有实现 java.io.Serializable接口。

关于 java 中的序列化与反序列化

关于序列化，常又称为持久化，将其写入磁盘中。

进而对于编码规则来说：

任一一个实体类必须要去实现 Serializable 接口，方便以后将该类持久化，或者将其用于转为字节数组，用于网络传输。

对于一个实体类，不想将所有的属性都进行序列化，有专门的关键字 transient：

private transient String name;

当对该类序列化时，会自动忽略被 transient 修饰的属性。

创建一个线程

Java 提供了三种创建线程的方法：

通过实现 Runnable 接口；

通过继承 Thread 类本身；

通过 Callable 和 Future 创建线程。