编译Javac



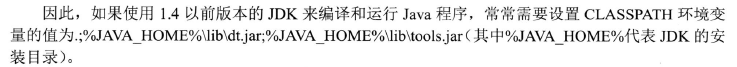
运行java



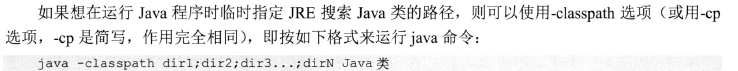
关于CLASSPATH环境变量

JDK1.5及以上：不用设置，包含当前目录

JDK1.4及以下：（.）



临时指定JRE搜索Java类的路径：win(;) Linux(:)隔开

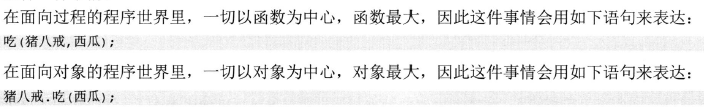


Jshell(JDK9)

面向对象：







UML:



一切都是对象：体会java程序从头到尾的运行流程。

注释：

// /\* \*/ 文档注释(Java9):/\*\* \*/

包和文件夹的关系？

关键字：

Java的所有关键字都是小写的。

Java有指针吗？This？

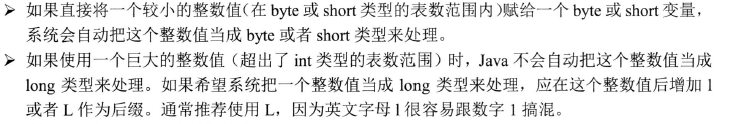
强类型（Strongly typed）语言

Java数据类型：

基本类型，引用类型

Java引用类型？指针？C++引用区别？ 只是一个概念，内存和变量名的关系，没有具体的操作，比如指针加一之类。

整数默认为int(大数加L后缀)



没有signed int之类？没有

Java没有字符串的基本类型，通过String类。

精确保存一个浮点数：BigDecimal类

三个特殊浮点数：

正无穷POSITIVE\_INFINITY

负无穷NEGATIVE\_INFINITY

非数 NaN

当分子、分母都为整型，分母为 0 的时候，会报除0异常

当分子、分母中有一个为浮点型，且分母为 0 或 0.0 的时候，会得到 Infinity (正无穷大) 或 -Infinity（负无穷大）

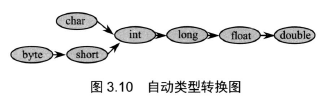
Double和Float的无穷大、无穷小相比，结果为 true

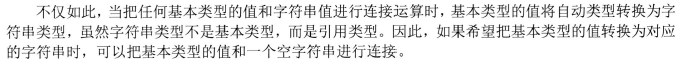


布尔型(boolean)：

java中为true和false，不能用0和非0

自动类型转换和强制类型转换





任何基本类型的数和字符串用+连接，自动转换为字符串，但注意运算顺序的问题。

整数默认int，小数默认double。

有很多转换的方法，比如字符串转各种数字类型。

String类是不可变类

移位：

不改变操作数本身

If else:



数组：



数组初始化new与不new：

1、不同于String类，String由于实现了常量池 所以new 和不new 有区别:new的话，引用变量指向堆区。不new的话，引用变量指向常量池。  
2、对于数组的定义，初始化时用new与不用new 没区别，只是两种方式罢了，因为数组是引用数据类型，建立对象时，无论用不用new，数组实体都是放在堆内存中，引用变量放在栈内存。

这是由于java设计之初为了契合用户习惯做得妥协（也可以说是优化）。

大家都知道java是1995年设计出来的，相比c（1972年）、c++（1983年）这些语言来说是后起之秀。而c、c++定义数组习惯使用 int a[] = {1, 2, 3}; 这样的方式。因此java也顺势继承了下来。

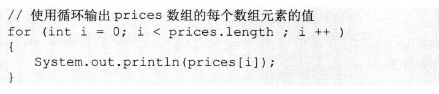
java作为纯面向对象语言，同时数组在java中以对象形式存在。这种写法咋看之下违背了对象创建的方式，而它实际上只是一种语法糖（简化写法）。编译器遇到 int a[] = {1, 2, 3}; 会编译成和 int a[] = new int[] {1, 2, 3}; 完全一样的中间代码。

因此 int a[] = {1, 2, 3}; 和 int a[] = new int[] {1, 2, 3}; 实际上是等价的，任何一种写法都能正确编译，所以当然不会报错了。

数组初始化：(与C不同)

注意：不要同时使用静态初始化和动态初始化，也就是说，不要在进行数组初始化时，既指定数组的长度，也为每个数组元素分配初始值。

数组遍历的循环：(有库可以用就是好)



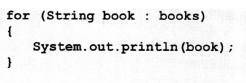
java中length,length(),size()区别

1 java中的length属性是针对数组说的,比如说你声明了一个数组,想知道这个数组的长度则用到了length这个属性.

2 java中的length()方法是针对字符串String说的,如果想看这个字符串的长度则用到length()这个方法.

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!

foreach循环（遍历数组和集合更加简洁）但是不能改变原数组的值（丢掉了索引信息）



//遍历二维数组

int[][] arr2 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}} ;

for(int[] row : arr2)

{

for(int element : row)

{

System.out.println(element);

}

}

Java堆内存栈内存

Java把内存分成两种，一种叫做栈内存，一种叫做堆内存

在函数中定义的一些**基本类型的变量和对象的引用变量都是在函数的栈内存中分配。**当在一段代码块中定义一个变量时，java就在栈中为这个变量分配内存空间，当超过变量的作用域后，java会自动释放掉为该变量分配的内存空间，该内存空间可以立刻被另作他用。

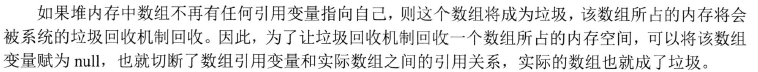
**堆内存用于存放由new创建的对象和数组。**在堆中分配的内存，由java虚拟机自动垃圾回收器来管理。在堆中产生了一个数组或者对象后，还可以在栈中定义一个特殊的变量，这个变量的取值等于数组或者对象在堆内存中的首地址，在栈中的这个特殊的变量就变成了数组或者对象的引用变量，以后就可以在程序中使用栈内存中的引用变量来访问堆中的数组或者对象，引用变量相当于为数组或者对象起的一个别名，或者代号。

引用变量是普通变量，定义时在栈中分配内存，引用变量在程序运行到作用域外释放。而数组＆对象本身在堆中分配，即使程序运行到使用new产生数组和对象的语句所在地代码块之外，数组和对象本身占用的堆内存也不会被释放，**数组和对象在没有引用变量指向它的时候，才变成垃圾，不能再被使用，但是仍然占着内存，在随后的一个不确定的时间被垃圾回收器释放掉。这个也是java比较占内存的主要原因，实际上，栈中的变量指向堆内存中的变量，这就是 Java 中的指针!**

**彻底理解Java中堆和栈的区别**：<https://blog.csdn.net/weixin_37618354/article/details/80239219>

　　栈与堆都是Java用来在Ram中存放数据的地方。与C++不同，Java自动管理栈和堆，程序员不能直接地设置栈或堆。  
  
　　Java的堆是一个运行时数据区,类的(对象从中分配空间。这些对象通过new、newarray、anewarray和multianewarray等指令建立，它们不需要程序代码来显式的释放。堆是由垃圾回收来负责的，堆的优势是可以动态地分配内存大小，生存期也不必事先告诉编译器，因为它是在运行时动态分配内存的，Java的垃圾收集器会自动收走这些不再使用的数据。但缺点是，由于要在运行时动态分配内存，存取速度较慢。  
  
　　栈的优势是，存取速度比堆要快，仅次于寄存器，栈数据可以共享。但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性。栈中主要存放一些基本类型的变量(,int, short, long, byte, float, double, boolean, char)和对象句柄。

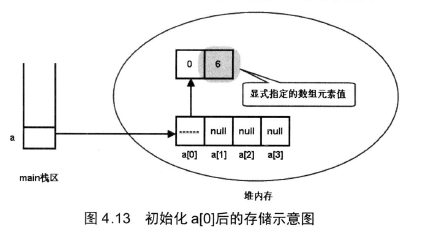
回收一个数组所占的空间：



Java中把一个数组赋值给另一个数组：改变引用。

C/C++呢？没有这样的操作

分清基本类型的数组和引用类型的数组，有些问题可以更清楚。比如多维数组：



五子棋，有时间看看p122

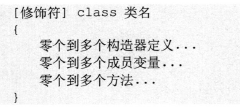
面向对象

访问控制 组合关系 构造器

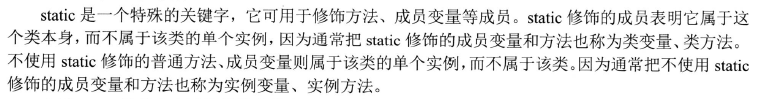
使用类来定义的变量 属于 引用变量

所有引用类型的变量都继承类Object类。基本数据类型不是。

类定义：



Static修饰的成员不能访问没有static修饰的成员（this）



Static：区分成员变量、方法、内部类、初始化块属于类本身还是实例。

Static 静态，并非不能变

Static final 类似于全局常量

构造器即构造函数？

构造器直接使用new关键字来调用。

this

对象的一个成员调用另一个成员，this可以省略。

static修饰的方法可以使用类来直接调用，不能使用this，所以，不能访问没有static修饰的成员。

静态方法访问普通方法：整个对象

方法：

Java里参数传递方式只有一种：值传递

（那如果我的参数是个数组，对象(引用类型)呢？）明明是引用传递

什么时候加static什么时候不加？就方法来说（区分类和对象吗？）

形参个数可变的方法：

在最后一个形参的类型后面加...。所以一个方法最多有一个。传参时可以传递多个参数，也可以传递一个数组。

方法重载：类名相同，形参列表不同。

Java确定一个方法的三个要素：调用者（对象，类）、方法名、形参列表

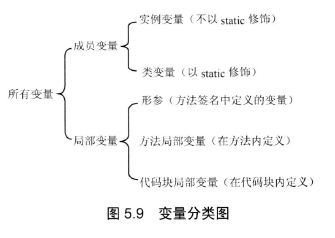
成员变量和局部变量：

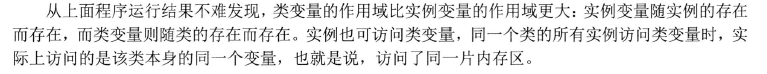
自己想一哈

成员变量：类中的定义的

局部变量：方法中定义的

书上





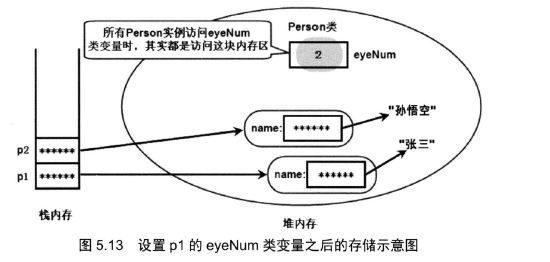
成员变量的初始化和内存中的运行机制

自己：

成员变量——类中定义，初始化： 系统会自己初始化，赋值：类.成员 ~~对象.成员~~

内存：生命周期：类

书上：



当程序需要访问类变量的时候，尽量使用类作为主调，而非对象，这样可以避免产生歧义。

类变量和实例变量 根据什么来设置？

代码块局部变量：定义在代码块内

成员变量：类的固有信息、需要在类的多个方法间共享的信息

实例变量：实例运行时的状态信息

局部变量：作用域和生命周期尽可能小（能用代码块局部变量不用方法局部变量）

隐藏和封装：

封装：

把对象的信息（成员变量和实现细节）隐藏在内部，不允许外部程序访问对象内部信息，而是通过对象提供的方法来实现对内部信息的操作和访问。

实现：怎么限制？（访问控制符）



private:把成员变量隐藏在类的内部

default:相同包的其他类访问

protected:相同包以及不同包的子类：希望子类重写这个方法

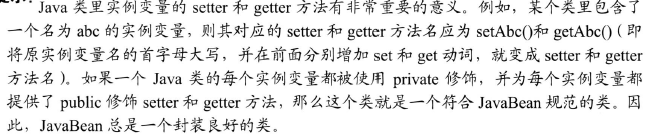
public:所有都能访问

修饰符：



一个文件一个public类

JavaBean:



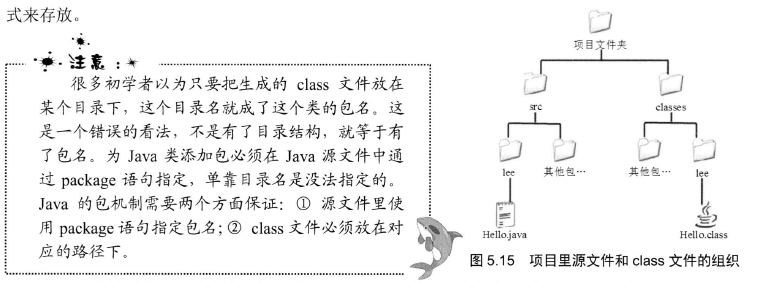
1. 封装 2.对参数边界控制

static对类，什么时候加，什么时候不加

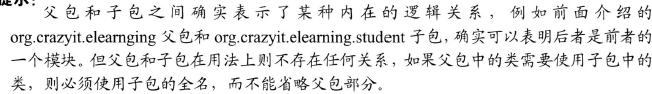
包：

包名应该全部是小写字母，一个或者多个有意义的单词连缀。

package语句作为源文件第一条非注释语句。

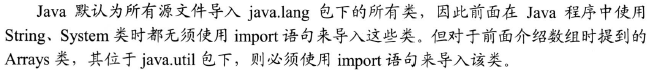


父包和子包：



import:简化编程，可以不使用类的全名，和访问权限没半毛线关系。

一个java文件包含一个package，但可以包含多个import。



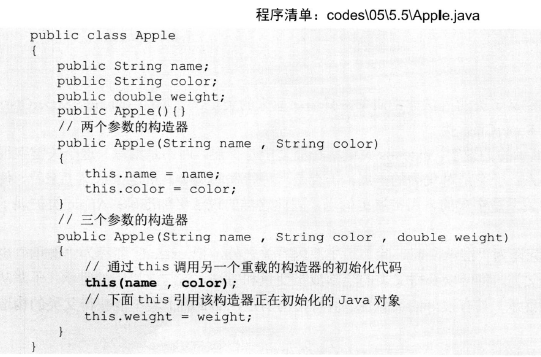
Import static:静态导入



构造器：

因为构造器主要用于被其他方法调用，用以返回该类的实例，通常把构造器设置为public。

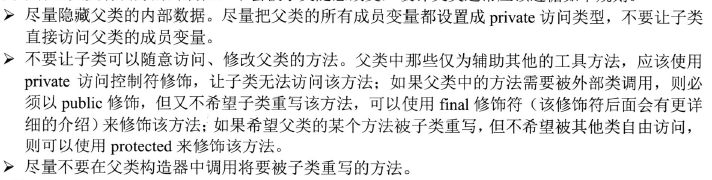
构造器之间的调用 this



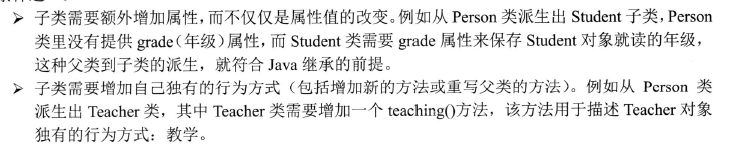
继承：

Java类只有一个直接父类。

java.lang.Object是所有类的父类。



什么时候需要从父类派生子类;



方法重写（方法覆盖）发生在子父类之间，子类的同名方法覆盖类父类方法。

规则：两同两小一大

两同：方法名，形参列表

两小：子类方法返回值类型，子类方法抛出异常类型 都小于等于父类

一大：子类方法访问权限 大于等于父类

子类的同名成员变量会隐藏父类的成员变量。

调用父类：super

把子类对象强制转换为父类，然后调用成员变量。p163

静态方法里不能用super？

this和super都不能出现在static修饰的方法中。

构造器：

使用super调用父类构造器必须出现在子类构造器的第一行。

看p171有点问题

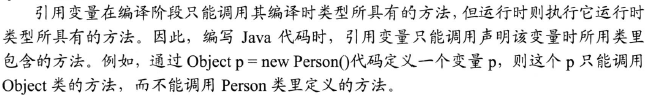
多态：

看完了自己说一说多态

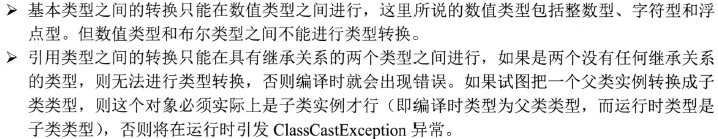
对象的实例变量不具有多态性，方法才具有多态性。

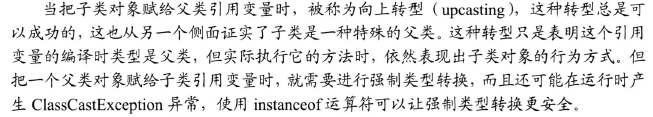
相同类型的变量，调用同一个方法时呈现多种不同的行为特征。

子类对象可以直接赋给父类的引用变量。（向上转型）此时，子类特有的方法调用不了。



引用变量的强制类型转换





Instanceof：

判断前面的对象是否是后面类，子类的实例。（判断是否可以强制类型转换）

前一个操作数：一个引用类型变量，后一个操作数：一个类（也可以是接口）

继承与组合：

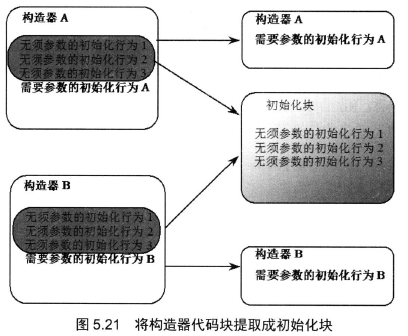
都允许在新类中复用旧类的方法。

继承（is-a） 组合（has-a）

初始化块：

在创建java对象时隐式执行，在构造器之前执行。

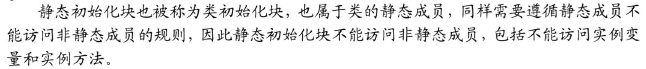
初始化块和构造器选择：初始化块不接收任何参数



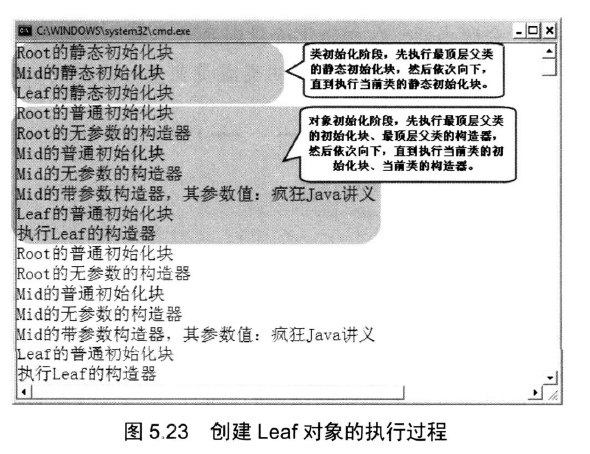
静态初始化块：

初始化块的修饰符只能是static。

（又一次体现出static用来区分类和对象）



类初始化阶段，先执行最顶层父类的静态初始化块，然后依次向下，直到执行到当前类的静态初始化块。



练习：p123 p180

包装类：

所有引用类型都继承类Object类。

为了解决8种基本类型不能当成Object类型变量使用：包装类。

作用：把基本类型近似当成对象 反之亦然

自动装箱 自动拆箱：类型匹配

println 两个数据之间+“”+？？？

基本类型转化为字符串

1.Integer.parseInt()/Integer.valueOf()

2.+来拼接字符串

包装类型的变量和基本类型比较：没有问题。包装类型取出数值来比较

包装类型的变量之间比较：直接比较会出问题。 使用静态的compare方法。

处理对象：

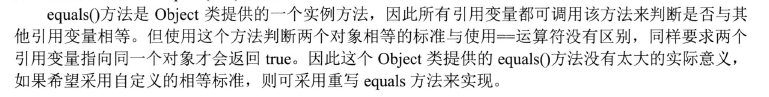
打印对象，toString：

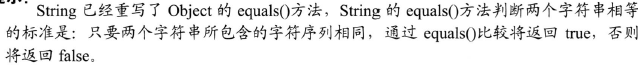
对象打印、对象和字符串连接=p.toString来做

==和equals

== ：数值：ok，引用：指向同一个对象才返回true

equals：







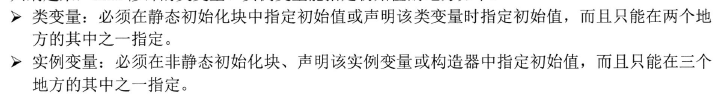


final：

fianl变量：

修饰的类，变量，方法不可改变。一旦获得初值不能改变。

Final修饰的成员变量必须由程序员显式指定初值。



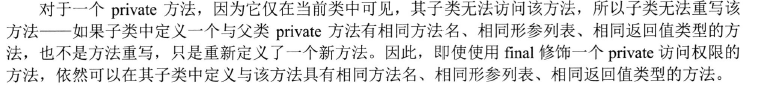
修饰引用类型：只保证引用地址不会改变，即一直引用一个对象，但是对象本身可以改变。

Eg.修饰的变量指向一个数组，不可以指向别的数组，但这个数组的内容可以改变。

用final定义宏变量：



final方法：



final修饰的方法只是不能重写，可以重载。

final类：

不可以有子类

不可变类：

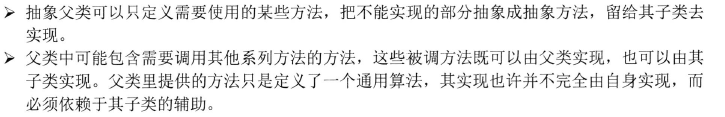
8个包装类和java.lang.String

抽象类：

abstract

不能实例化（new），只能作为父类给子类继承，没有方法体

抽象类体现模板设计模式，抽象类作为多个子类的通用模板。



抽象方法：

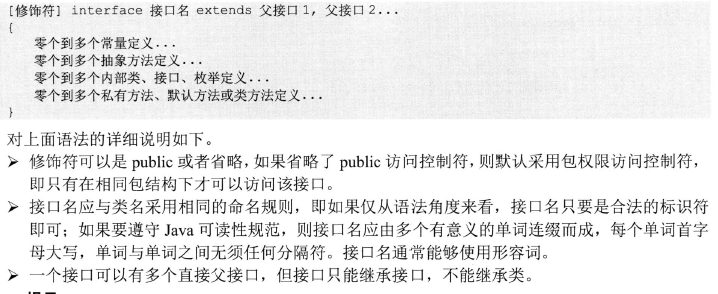
必须由子类提供实现（重写）

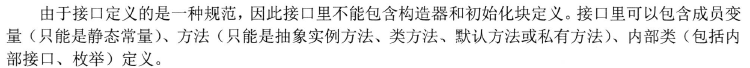
Static和abstract不能同时修饰一个方法。

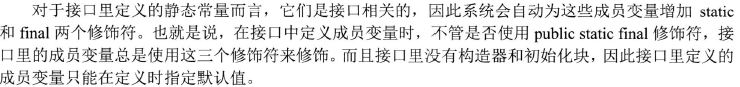
接口：

和抽象类对比

interface





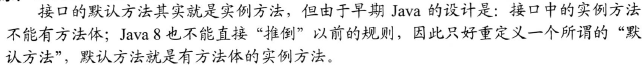


接口中的成员变量：public static final 就是常量的意思

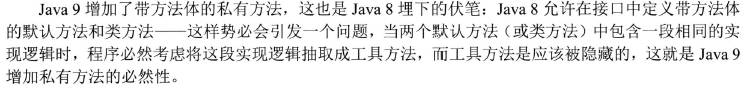
接口中的方法：抽象方法abstract、类方法static、默认方法和私有方法。

接口里的普通方法：总是使用public abstract修饰，且是抽象方法（没有方法体）

默认方法：default



私有方法private



接口里的类：



内部类？

接口可以被看作一个特殊的类，因此一个java源文件里最多一个public接口。

接口里的继承：支持多继承

接口实现：

实现接口的方法必须声明为 public，而且实现方法的类型必须严格与接口定义中指定的类型相匹配。

集合框架：

get(int)方法只有ArrayList的实现是高效的，换成LinkedList后，索引越大，访问速度越慢。所以我们要始终坚持使用迭代器Iterator来访问List。

List 接口

List是有序、可重复的容器。

有序指的是：List中每个元素都有索引标记。可以根据元素的索引标记（在List中的位置）访问元素，从而精确控制这些元素。

可重复指的是：List允许加入重复的元素。更确切地讲，List通常允许满足e1.equals(e2) 的元素重复加入容器；

Map 提供了一个更通用的元素存储方法。Map 集合类用于存储元素对（称作“键”和“值”），其中每个键映射到一个值。

异常：

一旦某个catch捕获到匹配的异常类型，将进入异常处理代码。一经处理结束，就意味着整个try-catch语句结束。其他的catch子句不再有匹配和捕获异常类型的机会。

对于有多个catch子句的异常程序而言，应该尽量将捕获底层异常类的catch子 句放在前面，同时尽量将捕获相对高层的异常类的catch子句放在后面。否则，捕获底层异常类的catch子句将可能会被屏蔽。

final 有必要？

我可以在try catch 后加代码块就行了呀

泛型：

List arrayList = new ArrayList();

List<String> arrayList = new ArrayList<String>();

在编译之后程序会采取去泛型化的措施。也就是说Java中的泛型，只在编译阶段有效。在编译过程中，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，泛型信息不会进入到运行时阶段。

泛型类型在逻辑上看以看成是多个不同的类型，实际上都是相同的基本类型。？？？？

泛型类

容器类

泛型的类型参数只能是类类型（包括自定义类），不能是简单类型

泛型类最常用的使用场景就是“元组”的使用。我们知道方法return返回值只能返回单个对象。如果我们定义一个泛型类，定义2个甚至3个类型参数，这样我们return对象的时候，构建这样一个“元组”数据，通过泛型传入多个对象，这样我们就可以一次性方法多个数据了。

日志：

本文链接：<https://blog.csdn.net/lgh1992314/article/details/70568647>