# Head First Java 读书笔记

[Head First Java 读书笔记 1](#_Toc12006834)

[一：基本概念 2](#_Toc12006835)

[二：类与对象 3](#_Toc12006836)

[三：原始的主数据类型和引用 3](#_Toc12006837)

[四：方法操作实例变量 6](#_Toc12006838)

[五：编写程序 7](#_Toc12006839)

[六：认识Java的API （使用Java库函数） 7](#_Toc12006840)

[ArrayList：长度自动动态改变的集合类（对象） 7](#_Toc12006841)

[比较ArrayList与一般数组： 7](#_Toc12006842)

[库函数： 8](#_Toc12006843)

[第七章：继承与多态。 8](#_Toc12006844)

[多态 9](#_Toc12006845)

[方法的覆盖（重写Overriding）：子类重写父类方法 9](#_Toc12006846)

[方法的重载（overload）：在一个类中讨论 10](#_Toc12006847)

[第八章：接口与抽象类（深入多态） 10](#_Toc12006848)

[抽象类 10](#_Toc12006849)

[Object类（java.lang.Object）有什么？： 10](#_Toc12006850)

[接口： 11](#_Toc12006851)

[Super 12](#_Toc12006852)

[第九章：构造器与垃圾收集器（对象的前世今生） 13](#_Toc12006853)

[堆与栈：生存空间 13](#_Toc12006854)

[成员变量，实例变量，类变量（静态变量）局部变量 13](#_Toc12006855)

[构造方法，类方法，实例方法的区别 13](#_Toc12006856)

[创建对象 13](#_Toc12006857)

[调用父类构造函数 13](#_Toc12006858)

[This() 14](#_Toc12006859)

[数字与静态 14](#_Toc12006860)

[Math 14](#_Toc12006861)

[Static 14](#_Toc12006862)

[Final 15](#_Toc12006863)

[主数据类型的包装类 15](#_Toc12006864)

[数字（日期）的格式化 16](#_Toc12006865)

[操作日期 16](#_Toc12006866)

[有静态的import 17](#_Toc12006867)

[11：异常处理 18](#_Toc12006868)

[14：序列化(serialization)、IO：保存对象 19](#_Toc12006869)

[Java.io.File 20](#_Toc12006870)

[缓冲区（好比超市的推车！！！） 21](#_Toc12006871)

[读取文本文件： 21](#_Toc12006872)

[写入文本文件 22](#_Toc12006873)

[解析字符串（用字符串中的特殊字符） 22](#_Toc12006874)

[15:网络与线程 22](#_Toc12006875)

[建立Socket连接 22](#_Toc12006876)

[端口 22](#_Toc12006877)

[读取Socket 23](#_Toc12006878)

[客户端代码： 23](#_Toc12006879)

[服务器端代码 24](#_Toc12006880)

[线程 25](#_Toc12006881)

[如何启动新的线程。 25](#_Toc12006882)

[并发性问题 26](#_Toc12006883)

[死锁 26](#_Toc12006884)

[16:集合与泛型 27](#_Toc12006885)

[泛型 28](#_Toc12006886)

[Compareable 28](#_Toc12006887)

[集合 28](#_Toc12006888)

[对象的等价 28](#_Toc12006889)

## 一：基本概念

Java的工作方式：

源代码.java——编译器(运行源代码执行javac程序来编译xx.java)——输出（编译器产生.class字节码（平台无关））——JVM（将字节码转换成平台能理解的形式来运行。）

【1】虚拟机启动执行时，自动寻找指定类，锁定执行 public static void main (String[ ] args ) 方法。

【2】java的 Integer与Boolean两种类型并不相容。（ X==4 ）是布尔型。

While（x==1）{ }

√

把测试放到括号里

Int x=1；

While（x）{ }

×

Boolean x=true；

While（x）{ }

√

## 二：类与对象

【1】在设计类时，要记得对象时靠类的模型塑造出来的。

【2】真正的Java程序只会让对象与对象交互，main的用途是测试类和启动Java程序。不要呆在main中！！

【3】对象创建时存储在内存的堆内。当某个对象被Java虚拟机察觉不会再被使用到时，该对象就被标记成可回收。当内存不足，垃圾收集器就会启动来清理垃圾。

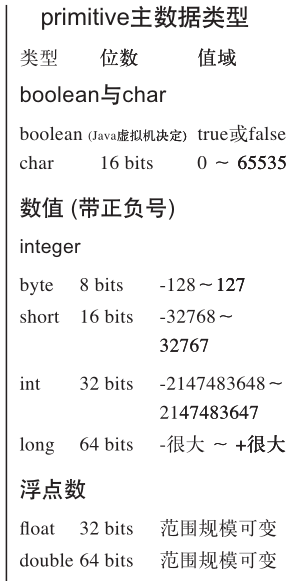
【4】任何变量只要加上 public、 static、 final 基本上都会变成全局变量取用的常数。

【5】java程序在执行期是一组会互相交谈的对象。类可以继承较为抽象的父类。

## 三：原始的主数据类型和引用

【1】变量必须拥有类型variables must have a type，变量必须拥有名称variables must have a name

【2】类型与类是相通的同义字。



【3】float的数必须加上f，否则所有带小数点的数都会被Java当成double处理。

【4】关键字：必须以字母 \_ $ 开头，不可用数字。

【5】对象只会存在可回收垃圾的堆上。

【6】没有对象变量这种东西的存在。只有引用到对象的变量。

如： Dog d = new Dog（）； d就是引用到对象的变量。他不是对象的容器，是类似指向对象的指针。在Java中我们不会也不该知道引用变量中实际装载的是什么，它只是用来代表单一的对象。只有Java虚拟机才会知道如何使用引用来取得该对象。

【7】 Dog myDog = new Dog（）；

Dog myDog 声明了一个引用变量。

New Dog() 创建对象，要求java虚拟机分配堆空间给新建立的Dog对象。（9章详细讨论）

= 将新的Dog对象 赋值给myDog 这个引用变量。

不能对引用变量进行运算（Java不是C）

Dog型的引用变量可以引用某个Dog对象 ，然后一会儿又去引用另外一个Dog 只要是Dog就行。但是被标记成Final的话，一旦被指派给某个Dog之后，就不能赋值给这个特定Dog之外的任何事物，被固定下来了。引用变量可以是空指针，null，但没意义。

【8】Book a = new Book();

Book b = new Book();

Book c = b; c与b引用到了同一个对象。

当某个对象没有了引用，变成无法存取的。被回收。而引用变量不再引用任何对象时，是null 还可以引用其他同类型的对象，不会被回收。

【9】 nums = new int [7];

数组是对象。JAVA的标准库函数包含许多复杂的数据结构，map，tree，set 。快速有序有效率的排列元素的时候，数组是很好的选择。数组可以使用位置索引来快速、随机的取元素。

数组中可以放八大数据类型或者引用。Dog[ ]中的元素是引用变量，只能保存指向对象的引用而不是对象。

记住，数组可以有主数据类型的元素，但数组本身绝对不会是主数据类型。不管数组带有什么，他一定是对象。

【10】**Dog [ ] pets ; 声明数组变量,只有数组变量才能接受数组。**

Pets = new Dog[7]; 创建大小为7的数组，赋值给变量

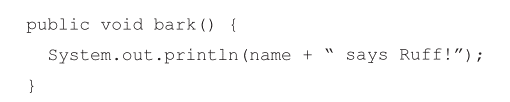
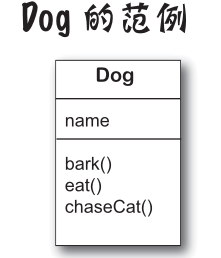
Pets[1]=new Dog();

Pets[2]=new Dog();

对象数组的操作不需要变量名称，数组索引就可以操作特定的数组对象了。

Dog[ ] myDog = new Dog[7];只有数组变量才能接受数组。编程一定要保证对应关系！！！！！

【11】一个对象，由变量和方法组成。



Bark方法中的name变量就是调用bark方法的特定Dog对象拥有的特定的name。

变量有两种，主数据类型和引用

## 四：方法操作实例变量

类时创造对象的蓝图。对象有状态和行为，即 实例变量和方法。方法会用到实例变量的值。



所以即使方法体一样，也可能出现不同的行为结果。

1. 方法只能声明单一的返回值，若要返回3个int 可以把返回值设为int 数组。如果有混合不同类型的值要返回时，可以考虑之后的ArrayList。
2. Setter 是给实例变量赋值，getter是返回实例变量的值。
3. 封装：通过强制其他程序使用setter方法来设定变量而不是直接的用点（.）存取。我们可以编写方法体来保证数据的安全。需要值的时候调用get方法来取值。One.getSize()

将实例变量标记为 private ，将getter与setter方法标记为public。

1. 实例变量永远都会有默认值。整形：0 浮点型0.0 布尔型：false 引用类型：null
2. 实例变量和局部变量的差别：
   1. 实例变量生命在类中，局部变量声明在方法中。
   2. 局部变量没有默认值，在使用前必须初始化，否则报错。
3. 使用==比较两个主数据类型或者判断两个引用是否引用同一个对象

使用equals（）来判断两个对象是否在意义上相等。 如两个String对象是否带有相同的字节组合

## 五：编写程序

要习惯使用 加强for （从Java 5.0开始）

for（int a ： cat）{ } cat 是要被逐个执行的数组或集合

使用 Interger.parseInt( ) 来取得String的整数值。只在String为数字时起作用。

## 六：认识Java的API （使用Java库函数）

### ArrayList：长度自动动态改变的集合类（对象）

add（Object elem）； remove(int index）；remove(Object elem）

contains (Object elem) isEmpty( ) 返回true/false

indexOf( Object elem) 返回对象参数的索引或-1 get(int index) 返回索引参数的对象。

ArrayList<Egg> myList = new ArrayList<Egg>( ); 新的集合对象创建在堆上。

Egg s = new Egg();

myList.add(s);

int theSize = myList.size();

Boolean isIn = myLIst.contians(s);

* ArrayList是对象，但不是数组。他可以确实的将引用删除掉，动态的改变大小。
* ArrayList不能直接保存主数据类型，这点数组更有效率。用包装类把主数据类型包装起来（java5.0之后会自动进行）主数据类型被包装成Object。
* 使用数组时，以特殊的数组的语法来操作。无法调用方法，最多存取他的length实例变量
* 使用ArrayList时，只是在运用ArrayList类型的对象，跟其他对象一样，使用.调用方法。

### 比较ArrayList与一般数组：

1. New String[2] 定要指定大小。New ArrayList<String>() 无需指定大小，会自动调整。
2. Mylist[1]=b 存放时指定索引值。

Mylist.add(b); 不需指定索引值。也可以用Add（int ， Object ）指定索引值。

1. Mylist[1] 方括号[ ] 是只用在数组上的特殊语法。ArrayList是个普通对象，没有特殊语法。
2. ArrayList可以参数化类型.，ArrayList <String> <String>是类型参数。Java5.0才有

### 库函数：

ArrayList是放在java.util包里面，java.util 里面有很多的工具类。

System、String、Math 都是属于 java.lang 包中。

必须指明代码中用到的类的完整名称。1：import java.util.ArrayList; 2：打出全名。

用到java.lang 这个包中的类无需指明完整名称。

* 和C语言的include不同，使用import不会使程序变大，，只是帮我省下每个类前面的包名称而已。
* Java.lang 是个经常会用到的基础包，所以会预先被引用。String和System是独一无二的class，Java知道去哪里找。

## 第七章：继承与多态。

1. 类的成员包括：实例变量和方法。
2. 善于用继承来防止子类中出现重复的代码。所以编程之前的程序设计阶段尤为重要。
3. 当调用对象引用方法时，调用与该对象类型最接近的方法。Java虚拟机会从最低阶的类开始往上找。
4. 要找好 has-a 和 is-a 关系。所有子类应该通过上层父类的is-a测试。使用is-a检测合理性。
5. 父类是不能使用子类特有的方法的，就像遗传不能倒过来。
6. 通过super.方法名 让子类不完全覆盖父类的功能，只是额外加上行为。会先执行super版的方法，然后再回来执行sub版自定义的行为
7. Public 被继承，private不被继承。 Private default protected public 限制程度依次减小。
8. 子类是extends父类出来的。继承下来的方法可以被覆盖，但是实例变量不能被覆盖。
9. 某方法在子类被覆盖过，调用方法时会调用覆盖过的版本。
10. 继承的好处：确保父型下所有类都有父型的全部方法。拟了一份合约。
11. 子类没有层数限制，但一般不会超过一到两层。
12. 除了内部类之外，没有私有类这个概念。三种方法让类不能继承：1：非公有的类只能被同一个包的类继承。2：用final标识类。Final是继承树的末端，不能被继承。3：让类只拥有private的构造方法。
13. 需要确保安全，确保方法都是我写的版本，就需要final。

### 多态

多态下，引用与对象可以是不同的类型。

* 1. Animal myDog = new Dog( );
  2. 运用多态，引用类型可以是实际对象类型的父类。换句话说，任何extends过声明引用变量类型的对象都可以被赋值给这个引用变量。由此可做出多态数组等玩意。
  3. 多态，顾名思义，就是多种样态。即同一个方法，参数是animal 类型，实参可以是dog、cat类型，由此触发不同子类的行为。这就是参数和返回值的多态！！！（将参数声明为父类类型，可以在运行时传入任何子类对象。）
  4. 通过多态，引进新型的子类也不必修改程序。

### 方法的覆盖（重写Overriding）：子类重写父类方法

1. 参数一样，返回类型要兼容（父类返回值类型或者其子类。）
2. 存取权限必须相同，或者更为开放。

### 方法的重载（overload）：在一个类中讨论

两个方法的名称相同，但参数不同。**所以重载与多态毫无关系。**

重载的条件是使用不同的参数，返回值类型自由定义。可以任意更改存取权限。

## 第八章：接口与抽象类（深入多态）

接口是多态和Java的重点。没有接口机制就没法活下去。

Animal a = new Animal（）；让人感到奇怪，animal是什么动物？不应该有animal实例被new出来。

### 抽象类

1. 在类声明前面加上 abstract 就是抽象类了。抽象类不能创建任何类型的实例，但是可以用抽象的类型作为引用类型给多态使用。设计继承结构时，必须决定好哪些类时抽象的，哪些是具体可以被初始化为对象的。
2. 抽象类除了被继承之外，没有用途，没有值，没有目的（除了抽象的 class 可以有static的成员 （10章谈）
3. 抽象方法咩有实体！含有抽象方法的类必须标为抽象。抽象方法存在的意义就是给子类们一个共同的协议！好处就是 多态！所有子类型都会有那些抽象的方法。
4. 抽象的方法必须在具体的类中运行。
5. 不是抽象的类就被称为具体类。第一个具体类必须实现所有的抽象方法。

### Object类（java.lang.Object）有什么？：

* 1. equals（Object o）判断对象是否与其他对象相等。
  2. getClass() 告诉对象是从哪里被初始化的。
  3. hashCode() 列出对象的哈希代码（相当于唯一ID）
  4. toString() 列出类的名称和一个其他的数字。

1. Object不是正式的Java抽象类，因为可以被所有类继承下来的方法都有方法体。所以没有需要必须被重写的方法。
2. 建议重写hashCode() 、equals() 、toString()。不是所有Object方法都可被重写，有的方法有final标记。
3. Object类时具体的，可以创造出Object的实例对象，因为有时会需要一个通用的，轻量化的对象。最常见的用途是在线程的同步化上面（15章）
4. Object的作用：作为多态让方法可以应付多种类型的机制。提供java执行期间对任何对象都需要的方法体。（让所有的类都被继承）。有一部分方法与线程有关。
5. Java是类型检查很强的程序，你只能从确实有该方法的类中调用该方法。类型安全检查。这就是不把所有的参数和返回类型都设置为Object的原因。

Object o = new Dog(); o.eat(); //非法的。

1. 使用Object有代价：ArrayList<Object>
   1. 放进去的对象是猫狗鱼，出来后都变成了Object类的实例。
   2. Object无法赋值给猫狗鱼的引用。无法通过编译，编译器无法确定它是猫狗还是鱼。
   3. 所以一定注意，返回值类型是Object时，不能赋值给其他类型的引用。但是其他任何类型的引用/对象都可以赋值给Object类型的引用。 要通过IS-A测试的。
2. 编译器是根据引用类型来判断有哪些方法可以调用，而不是看对象的类型。
   1. Object o = new Dog(); 引用o不能叫。因为Object类没有叫这个方法。
3. 当把对象装进ArrayList<Object>中时，不管他原来是什么，你只能吧它当成是Object。取出引用时，引用的类型只会是Object。

当你确定时，可以强制转换成原来的类型：

Object o = new Dog();

Dog d = (Dog) o;

d.eat();

如果不确定是个Dog，使用Instanceof运算符来检查，错了的话抛出ClassCastException

if (o instanceof Dog){

Dog d = (Dog) o;

}

1. 把类的公有方法当做是合约的内容，合约是我对其他程序的承诺协议。

### 接口：

1. **接口的所有方法都是抽象的，没有内容，以分号结束。**
   1. **public interface Pet{**

**public abstract void play(); } 抽象方法用abstract修饰。**

* 1. **public class Dog extends Animal implements Pet {…}**

1. **实现接口，必须实现接口中的所有抽象方法（写出方法体），**

**继承抽象类（含有抽象方法的类即抽象类），也必须重写所有的抽象方法，非抽象的方法无所谓。**

**继承非抽象类，重写与否都无所谓。**

**\*\*\*\*所以只要我上面有抽象方法，我就都得重写，不然就不要继承或者实现。**

1. **接口的作用：不管你来自哪里，只要你实现这个接口，别人就会知道你一定会履行某些合约。**

**好的设计不需要在抽象类中定义实现细节（方法体），让细节在具体的子类上实现是合理的。**

1. **类可以实现多个接口，不同继承树上的类可以实现同一个接口！！非常重要。而类当多态来使用的时候，必须来自同一颗继承树。Animal引用不能接受 fire类。**
2. **如何设计：**

**类无法对其他的类通过IS-a测试时，设计不继承其他类的类。**

**在需要类的特殊化版本时，以重写或增加新方法的方式继承现有的类。（写子类）**

**需要定义一群子类的模板，又不想让程序员初始化模板时，设计抽象类。**

**定义出类可以扮演的角色，使用接口。**

**（类来自单亲家庭<只继承一个父类>，但是可以扮演不同的角色）**

### Super

不打算覆盖原来的方法，只是加入额外的动作。

父类有非private方法method()，子类用 super.method()来调用父类的方法。

Super关键字是用来引用父类对象的。

## 第九章：构造器与垃圾收集器（对象的前世今生）

本章讨论对象如何创建，存在于何处以及如何让保存和抛弃更有效率。堆，栈，范围，构造器，空引用。

### 堆与栈：生存空间

当Java虚拟机启动，会从底层操作系统取得一块内存块（我无法控制也不需在乎）

堆<heap>：存对象，栈<stack>：存方法和局部变量（又称栈变量）。

实例变量存在于所属对象中，对象在堆上，实例变量若是个对对象的引用，它是存在于该对象中。

·栈就是一个桶。后进先出。（堆栈，就是栈。是之前的叫法。）读到一个方法（连同其局部变量）就放入栈中，栈顶是目前正在执行的方法。

·堆是啥？就是个存储空间，以完全二叉树存储，方便进行堆排序，查找数据很快。

·为何要知道栈与堆得机制？

可以了解变量的有效范围(scope)，对象的建立，内存管理，线程(thread)和异常处理。

### 成员变量，实例变量，类变量（静态变量）局部变量

·成员变量：类中方法外定义的{

1. 实例成员变量（实例变量）：
   1. 方法外类中正常定义的 int I; 存在于所属对象中。（实例变量可以是对象的引用 … = new… ）
2. 类变量（静态变量）：Static修饰的。一处改变，处处改变。生存期为整个源程序。若在函数中，只能在函数内使用，出了函数，尽管还存在 但不能使用。 }

·局部变量：方法体中定义的变量。

### [构造方法](https://wenwen.sogou.com/s/?w=%E6%9E%84%E9%80%A0%E6%96%B9%E6%B3%95&ch=ww.xqy.chain)，类方法，实例方法的区别

·构造方法和类同名！且没有返回类型。构造方法不会被继承。

·类方法前面有static修饰，而实例方法没有。

·实例方法既能对类变量操作，也能对实例变量操作，而类方法只能对类变量进行操作。

构造函数执行时，第一时间会执行父类的构造函数。（抽象类也有构造函数。每个类都至少有一个）

### 创建对象

New Dog(); 是在调用Dog的构造函数（方法）。构造方法看起来像方法，但不是方法。这段代码会在初始一个对象时使用。唯一能调用构造方法的方式就是新建一个对象。

注意使用含参的构造方法。

写了一个有参的构造方法，且需要无参的时，要自己写一个。 此时编译器不会自动帮我写出。

### 调用父类构造函数

如果任意一个类没有编写构造函数，编译器自动帮写构造函数：

Public ClassName( ) {

super(); }

如果子类有构造函数但是没有调用Super()：

Super(); 注意super的（）内没有参数，所以只能自动调用父类无参的构造方法。

当父类没有无参的构造时，子类只能显式调用父类有参的构造方法。Super( I )

由此保证了，执行构造方法，必先执行其父类的构造方法。栈！

注意，super() 的调用必须放在构造方法的首句。不然ide没等运行直接报错，

### This()

使用this（）从某一构造函数调用同一个类的另外一个构造函数（重载版的构造函数）

This只能用在构造函数中，且必须是第一条，super（）与this（）不兼得。

GC（垃圾收集器） 当最后一个引用消失，便再也无法取得对象的引用。对象就被回收。

## 数字与静态

### Math

最接近全局的方法。这个类中的所有方法都不需要实例变量。方法都是静态的。

这些方法无需实例变量，因此也不需要特定对象来判别行为。直接用类名调用方法！

Math.round(42.2) 返回参数四舍五入之后的整形或者长整形。

Math.random() 返回介于0.0~1.0之间的双精度浮点数。

Math.abs() 返回绝对值。传入整型返回整型，传入双精度返回双精度（此方法有重载）

Math.min(); math.max; 也都有重载版本。可返回整型或双精度。

### Static

关键词标记出不需要类实例（对象）的方法。静态方法时一种不依靠实例变量也就不需要对象的行为。直接用类名来调用，标记构造函数为private 这样就无法创造类的实例（Math就是这样）

静态的方法不能调用非静态的变量/方法。（因为他不知道可以使用哪个实例变量。）

静态变量值对所有的实例共享（对所有的实例都相同）

实例变量：每个实例一个，静态变量：每个类一个。

静态变量会在该类的任何对象创建之前就完成初始化

会在该类的任何静态方法执行前就初始化。

若没有给静态变量赋初值，会自动设置默认值。同实例变量。

### Final

静态的final变量是常数（命名首字母大写）。即类加载之后，静态final变量就会一直维持原值。

Final修饰的变量（实例，局部，方法参数）代表了值不能动。

Final可以阻止方法的覆盖或者类的继承。

类标记final之后，就不必多此一举给方法标记final了。因为不能继承就肯定没有重写这一说。

### 主数据类型的包装类

当需要以对象方式来处理主数据类型，就把他们包装起来。

Int i = 200;

Integer inte =new Integer(i); 包装

Int uninte = inte.intValue();解开包装。

java5.0之后有 autoboxing功能，自动包装。

ArrayList<Integer> list = new ArrayList<Integer>();

List.add(3); 可以直接把整型3 加入ArrayList，没有add(int)，但编译器自动包装成Integer类型。

Int num = list.ger(0) 可以自动解开Integer对象的包装，因此可以直接把Integer赋值给int

ArrayList<int>将无法通过编译。

Autoboxing可以：参数/返回值 是包装类型，可以传入/出相应的主数据类型，反之亦然。

布尔表达式或是boolean包装类型的引用可以放在预期boolean值的位置。

可以对包装类进行数值运算

可以在包装类和主数据之间相互赋值运算。

包装类可以将String 转换成相应的数值。Integer.parseInt(“12313123”);

Double d = 4.25;

String doubleString = “”+ d; //“+”这个操作符是java中唯一有重载过的运算符

### 数字（日期）的格式化

String s = String.format(“%,d”,10000000); 输出的是有逗号的数字格式（美式数字）

完全同C的print

格式化输出日期：

完整的日期与时间：%tc

String .format(“%tc”,new Date() );

只有时间：%tr

String.format(“%tr”,new Date() );

周、月、日: %tA %tB %td

因为没有刚好符合我们要求的输出格式,所以得组合3种形式来产生出所需要的格式:

Date today =new Date ();这样就得把Data时象借进去3次字符串。

String.format (“%tA, %tB, %td”，today, today, today);這里的逗号号是直提給出的,星期天,11月28日

### 操作日期

Java.util.Date 适合用来取得当前的时间，但是很多功能转移到了下面的类

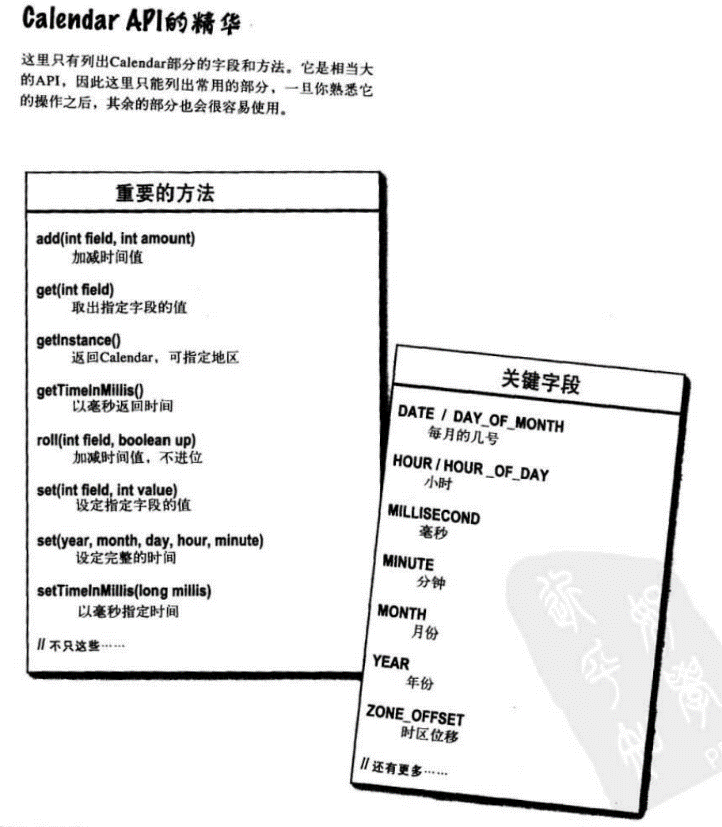
Java.util.Calendar

Calendar是个抽象类，无法取得Calendar的实例，但是可以取得其子类的实例

要这样才能取得Calendar实例：

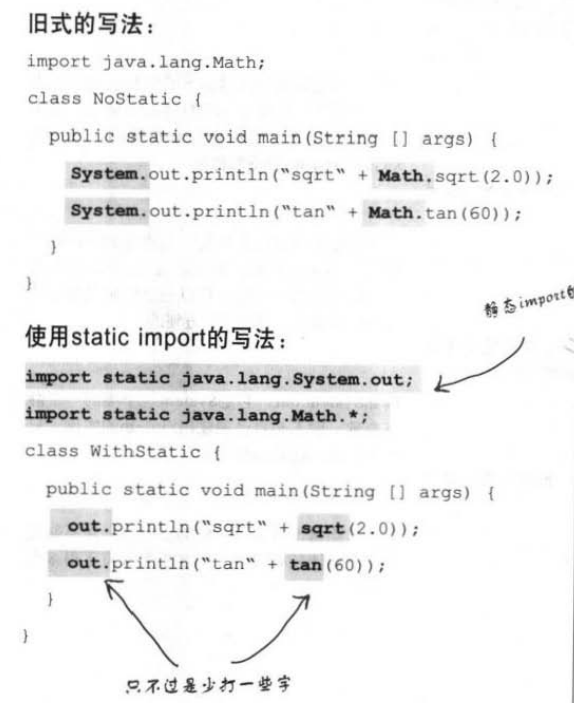
Calendar cal = Calendar.getInstance(); 返回一个具体子类的实例。会依照合约来响应Calendar应有的方法。大部分Java版本会默认返回一个Java.util.GregorianCalendar的实例。

可以不受限制的调用Calendar的静态方法。静态方法时在类上的，而不是具体的实例上的。



Calendar的API用到了要会查，有很多。

### 有静态的import

但是会让程序更容易混淆。让程序少打一些字。

## 11：异常处理

把有风险的代码块包含在try/catch块中。

异常是一种Exception类型的对象。Catch住的也是对象。

这个对象可以调用方法。PrintStackTrace() 列出有用的信息。

如果有抛出异常，必须声明他会抛出什么异常 方法名后面加 throws xxxException

RuntimeException被称为不检查异常，我可以自己抛出与抓住他们，但是没有必要，编译器不管的。

除了RuntimException之外，编译器会关照Exception的所有子类 编译器不管运行时异常。

因为大部分的RuntimeException都是因为程序逻辑问题。

Try/catch 是用来处理真正的异常，而不是逻辑错误。Try/catch块要做的是恢复的尝试，或者至少优雅的列出错误信息。

方法可以用throw抛出异常兑现。

Throw new Exception；

异常处理/恢复程序块放在catch中。

Finally块是用来存放不管有没有异常都得执行的程序。

如果try/catch块中有return指令，还是会执行finally流程会调到finally然后再回到return。

可以处理多重异常，throws 多个Exception。有多个catch语句 不同的Exception子类对象。

异常也是多态的，

throws 父类，便可以抛出其所有子类而不用个别的声明。

Catch父类，可以抛出其所有的子类异常。

最好为需要单独处理的异常编写不同的catch块。  
Catch Exception 会捕获所有异常，所以不知道哪里出错。有多个catch时要从小排到大。

DUCK：

可以不写trycatch语句，但是throws异常，这样谁调用这个方法，谁就自己处理异常，我不管了。

异常处理规格：

Catch与finally不能没有try

Try与catch之间不能有程序

Try一定要有catch或finally

只有finally没有catch的try必须要声明异常 throws

## 14：序列化(serialization)、IO：保存对象

不涉及数据库

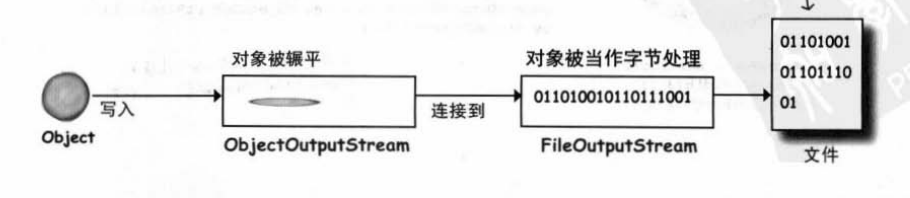
* + - 1. 将对象序列化（存储）
         1. 创建 FileOutputStream对象
         2. 上一步对象能让我写入对象，但无法直接连接文件，需要参数的指引。

创建ObjectOutputStream。 New时的参数是上面的对象。

* + - * 1. 写入对象

Os.writeObject(…..);

* + - * 1. 关闭ObjectOutputStream

Os.close();

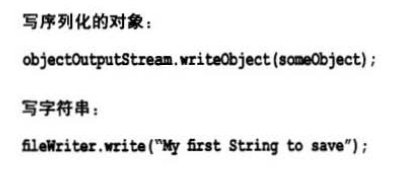
* + - 1. ObjectOutPutStream.writrObject(ss) 任何放在这个括号里面的对象都要序列化否则在执行期会出问题。
         1. 序列化只需要让类 implements Serialization （没有方法需要被实现，只是用来告诉Java虚拟机它可以被序列化。）
         2. 序列化是全有或者全无的，不能部分序列化。
         3. 如果需要序列化的程序能跳过某个实例变量，就标记成transient的变量。
         4. 静态变量不会被序列化。因为static代表每个类一个。而不是每个对象一个。
      2. 通过序列化存储对象的状态。
      3. 使用ObjectOutputStream来序列化对象(java.io)
      4. Stream是链接串流或是链接用的串流。？？？
      5. 链接串流用来衔接连接串流
      6. 用FileOutputStream链接ObjectOutputStream来将对象序列化到文件上。
      7. 调用ObjectOutputStream的writeObject(theObject)来将对象序列化，而不需要调用

FileOutputStream的方法。

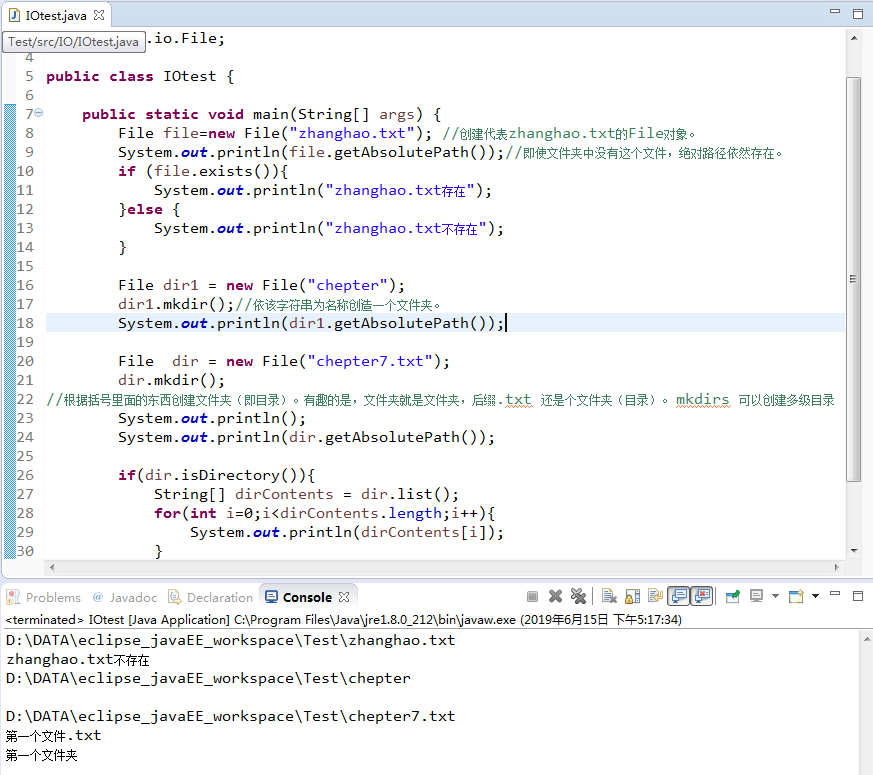
* + - 1. 对象必须实现要序列化的序列化接口。如果父类实现序列化，则不管是否有明确的声明，子类都会自动实现。
      2. 当对象序列化时，将序列化整个对象。这意味着它的实例变量所引用的对象也被序列化。
      3. 如果存在无法序列化的对象，则会在执行期间抛出异常
      4. 除非实例变量被标记为transient，否则在恢复时，它会被赋null或主基本数据类型的默认值。
      5. 解序列化（deserialization）时，所有类必须对Java虚拟机可用
      6. 读取对象的顺序必须与写入对象的顺序相同。
      7. readObject0的返回类型是Object，因此需要将解序列化的对象转换为其原始类型。
      8. 静态变量不会序列化，因为所有对象共享相同的静态变量值。序列化是对对象的序列化。

字符串写入文本文件

Servlet(在Web服务器上执行的Java程序)

 写入文本类似写入对象，使用FileWrite代替FileOutputStream（不会链接到ObjectOutputStream）

### Java.io.File

File类代表磁盘上的文件（并不是文件本身，类似路径。他不能读取或者代表文件中的数据）

实战操作是最好的老师！！！！！！！！

写的操作必须包含在Try/catch里面，否则编译器报错。很牛逼，很智能。

### 缓冲区（好比超市的推车！！！）

每趟磁盘操作都比内存操作要花更多的时间，所以先存进缓冲区，满了之后一并写入目的地。

BufferedWriter writer = new BufferedWriter(new FileWriter(new File(“aFile”)));

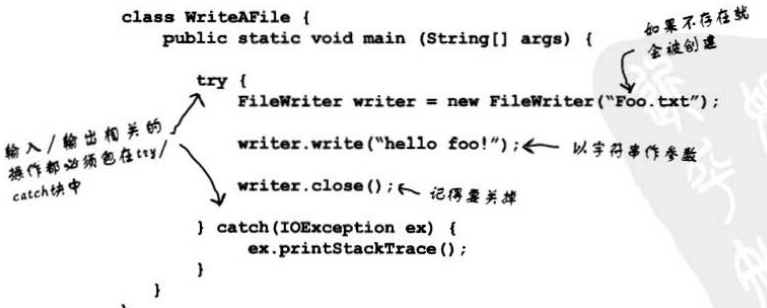
可以直接调用FileWriter的Write（）来写到aField中，但是每次都直接写（拿一件商品付一次账）

强制缓冲区立即写入，调用writer.flush();方法。

### 读取文本文件：

用FileReader来实际读取，用BufferedReader让读取更有效率。

### 写入文本文件



读或者写可以不手动创建File对象。因为File对象就相当于路径。

### 解析字符串（用字符串中的特殊字符）

用String的Split()把字符串拆开，并返回String[ ];

String a=”sdafadf /dsafaf ”;

String res = a.split(“/”);

## 15:网络与线程

一点要再看一下计算机网络的相关知识。

### 建立Socket连接

Ip地址与端口号 他在那里以及用哪个端口与收发数据。

Socket chatSocket = new Socket (“192.16.3.50”,5000);

Socket连接的建立代表两台机器之间存有对方的信息，包括网络地址和Tcp的端口号。

### 端口

一个地址可以有65536个不同的端口号。

0~1023的Tcp端口是保留给已知的特定服务使用，我不能用

不同的程序不能共享一个端口。若使用（绑定）已被占用的端口，会收到 BindException。

IP地址是门牌号码，端口号是该地址不同的工作窗口。

### 读取Socket

### 客户端代码：

### 服务器端代码

上面的程序在没有完成用户的相应程序循环之前无法回到循环的开始来处理下一个请求。（accept）

使用不同的~线程~并让新的客户端取得新的线程就好。

* + 1. 客户端与服务器通过Socket连接来沟通。
    2. Socket表示两个应用程序之间的连接，可能是在不同机器上执行。
    3. 客户端要知道服务器的ip（或网域名称）和端口号
    4. Tcp端口号是个16位的值，用来指定特定的应用程序。
    5. 一旦建立了Socekt连接，客户端可以从socket取得底层串流。Sock.getInputStream();
    6. 建立BufferedReader链接InputStreamReader和来自Socket的输入串流来读取服务器的文本数据。
    7. InputStreamReader是转换字节成字符的桥梁，用来链接BufferedReader与底层的Socket输入串流。
    8. 服务器可以使用ServerSocket来等待用户对特定端口的请求。
    9. 当ServerSocket接到请求时，它会做一个Socket连接（选取另一个端口做）来接受客户端的请求。

### 线程

Thread t = new Thread();

t.start(); Thread类在Java.lang 包中（同System、 String ）

Thread 对象代表线程，当你需要启动新的线程时就建立Thread的实例。

一个java程序代表一个线程（main在开始处），程序员负责启动自己建立的线程。

Thread有启动、连接和让线程闲置等重要方法。

### 如何启动新的线程。

1建立Runnable对象(线程的任务)

Runnable threadJob = new MyRunnable();

2建立Thread对象（执行工人）并复制Runnable（任务）

Thread myThread = new Thread （threadJob）;

告诉Thread对象要执行那个Runnable对象的run（）方法。

3 启动Thread

myThread.start();

没启动的话，什么也不会发生。在此之前myThread只是一个Thread的实例，并不是真正的线程。

Runnable是一个接口！只有一个方法 public void run( );

1. 一旦线程进入可执行状态，它会在可执行与执行中两种状态中来来去去，同时也有另外一种状态：暂时不可执行（又称 堵塞状态）
2. 线程暂时被挡住：调度器会因为（如：执行到等待Socket输入串流的程度，但是没有数据可供读，调度器就会将其移出可执行状态。）线程的Sleep() ，或者线程调用某个被锁住（locked）的对象的方法。要等到锁住该对象的线程放开这个对象才能进行。
3. 没有控制线程调度器的API 但是可以让某个线程Sleep 这可以保证在指定的时间内昏睡的线程一定不会被唤醒。
4. 一旦Thread线程调用过run方法。该线程就不能重新启动。Thread对象可能还呆在堆上，如同活着的对象一样还能执行某些方法的调用，但是已经永远失去了现成的执行性，只剩下对象本身。
5. Java中的每个线程都有独立的执行空间
6. 要把Runnable传给Thread的构造函数才能启动新的线程。
7. Run（）将会是新线程执行的第一项方法。
8. 线程在初始化吼还没有调用Start() 之前处于新建立的状态。
9. 调用Thread对象的Start（）之后，会建立出新的执行空间，他处于可执行状态等待被挑出来执行。
10. 调度不能保证任何的执行时间和顺序！所以你不能期待它会完全的平均分配执行，你最多也只嗯呢该影响Sleep的最小保证时间。
11. Sleep()应包含着 try/catch中。可能抛出InterruptException s
12. 可以创建两个线程，使用同一个任务。
13. 利用setName()来给线程命名，帮助除错。
14. 要判断好哪些指令不能被分割执行！使其同步化。
15. 每个对象都有单一的锁单一的钥匙，只会在对象带有同步化方法时才有实际的用途

### 并发性问题

1.数据安全问题。两个线程存取单一对象数据。2：更新丢失。

使用synchronize（使同步）修饰方法使其每次只能被单一的线程存取。即 锁。

锁不是配在方法上，而是在对象上。保护对象的实例变量。

如果有多个方法可能操作对象的实例变量，则这些方法都应该有同步化的保护。

目的是保护数据，但锁的是方法。

因为同步化是有代价的，程序会变慢。甚至会导致死锁。

被同步化了的代码块形成原子单位（不可分割） 不需要整个方法都同步化时，可同步部分代码块。

可以 Synchronize (this ) {….} 来同步部分代码。只会使用参数所指定的对象的锁来做同步化。

一般以当前对象来同步 this

### 死锁

两个线程互相持有对方正在等待的东西，一直等 等到海枯石烂。

数据库或者其他服务器有事务回滚机制来复原不能全部完成的交易，而Java没有。

Java没有处理死锁的机制，甚至不知道死锁的发生。

## 16:集合与泛型

Java集合框架 能支持绝大多数能用到的数据结构。

ArrayList有一大堆方法，但不能排序！！！

除了ArrayList之外：

TreeSet 以有序状态保持并可以防止重复。

HashMap 可用成对的name/value 来保存与取出。

LinkedList 针对经常插入或删除中间元素所设计的高效率集合 （实际上ArrayList更实用）

HashSet 防止重复的集合，可快速找寻相符的元素

LinkedHashMap 类似HashMap 但可以记住元素插入的顺序，也可设定成依照元素上次存取的先后来排序

把字符串放进TreeSet而不是ArrayList，String会自动按照字母顺序排在正确的位置。每当你想要列出清单，元素总是以字母顺序出现。当需要set集合或者字母排列的清单时，他很好用。

TreeSet的成本会较高，因为每次插入新项目，都会花时间找出适当的位置。

Collection 有一个sort( List list) 方法：把list中的String依照字母排序,原先的list就变成了有序的了。他会用到List 而ArrayList实现了List这个接口 所以ArrayList是个list

因为多态的机制，你可以吧ArrayList传给用List做参数的方法!

LinkedList对于在中间的插入或删除会比较快，但是对于大多数应用程序而言，ArrayList与LinkedList差不多有用。除非元素量真的很大！

对象被System.out.println(对象)列出来时会调用toString() 学会重写toString toString定义在Object中。

### 泛型

泛型基本都与处理集合有关。主要目的是让我能够写出有类型安全性的集合。（让编译器帮忙防止我把dog加到cat中）、

没有泛型，我可以把任何东西放入ArrayList 像是 ArrayList<Object> 取出来的是Object 而不是原先的对象的引用了。

创建被泛型化类的实例： new ArrayList<Song>() 3

声明与指定泛型类型的变量：List<Song> songList = new ArrayList<Song>()

声明与调用取用泛型类型的方法。Void foo(List<Song> list) x.foo(songList)

1. E是集合维护和返回的元素类型。文档上用的是T。因为E清楚地指明是元素。
2. 以泛型的观点,Extends关键词代表“是一个” 使用与类和接口（继承和实现）

### Compareable

### 集合

List 是知道索引位置的集合。可以有多个元素引用相同的对象。

Set 不予许重复的集合！！ 不会有多个元素引用相同的对象（被认为相等的；两个对象也不行）

Map 用键值对来进行存储。用key来搜索的专家 维护与key有关联的值。两个key可以引用相同的对象，值能重复，但key不能重复。典型的key是String 但也可以是任何对象。

Map没有继承Collection接口，但也应该被当做是collection Framework 的一份子。因此他不在Java.util.Collection 的继承层次中，也还是个集合。

### 对象的等价

因为Set不允许重复，所以使用Set时要进行对象的对比~

两对象相等，则对象1.equals(对象2)会返回true 且两者的hashCode()也会放回相同的值。

引用相等性：堆上同一对象的两个引用相等。

对象相等性：堆上两个不同对象在意义上是相同的。要想要吧两个不同的Song对象视为相等的，就必须覆盖从Object继承下来的hashCode()方法与equals()方法。 因为hashCode()默认的行为会返回每个对象特有的序号（大部分的Java版本是依据内存位置计算此序号。）所以必须覆盖过hashCode()才能确保两个对象有相同的hashCode()，也要确保以两一个对象为参数的equals()调用会返回true。

【注】HashSet在对比的时候，同样的hashCode有多个对象，它会使用equals()来判断是否有完全的符合。即：hashCode是用来缩小寻找成本，但最后还是要用equals()才能认定是否真的找到相同的项目。

1. 两对象相等，hashCode()必相等。
2. 两对象相等，互相调用equals()必须返回true
3. 有相同的hashCode值，不一定相等。相等，就一定有相同的hashCode值。
4. 若equals()被覆盖过，则hashCode()也必须被覆盖过。
5. hashCode() 的默认行为是在堆上的对象产生独特的值。如果没有重写过hashCode()，则类的两个对象怎么都不会被认为是相同的。
6. equals()的默认行为是执行==的比较，测试两个引用是否对上同一个对象。如果equals没有被覆盖过，两个对象永远不会被视为相同，因为不同的对象有不同的字节组合。

### TreeSet

TreeSet在防止重复上和HashSet一样，但他还会一直保持集合处于有序状态。



调用没有参数的构造函数意味着以对象的compareTo()方法来进行排序

Comparable 接口要求类必须实现compareTo()方法。