1 System.out.println(“Introduction to Java Programming,

By Y.Daniel Liang”);

——>System.out.println(“Introduction to Java Programming,”+

“By Y.Daniel Liang”);

2 Java 区分大小写 eg :Main代替main是错误的

2.1 public static void main(String[] args)

Args用来存储命令行的输入

例如：java Ex 1 2 3

3 源文件的扩展名必须是Java，而且文件名必须与公用类名完全相同。Eg：Welcome.java

4 在执行一个Java程序时，不要在命令行使用扩展名.Class。要使用java Classname。

5 print方法& println方法类似，不同点是当显示完字符串之后，println会将光标移动到下一行，而print不会。

6 使用变量的时候需要声明（variable declaration）告知编译器根据数据类型来为变量分配合适的存储空间。变量名如果由多的词组组成，除第一个词以外，其他词的第一个字母都要大写。Eg：int a; \int interestRate,b; \int a=4;

7 赋值语句中，左边变量的数据类型必须和右边值的数据类型兼容。~~Int x = 1.0 ;~~

8 定名常量（named constant）在同一个语句中声明与赋值(通常用大写字母命名)。Eg： datatype CONSTANTNAME = value;

8.0 final 使用通常出于两个理由：设计和效率

**Chapter7 FinalDATA**

8.1 final 使用的三种情况:数据 、方法、类。

一个既是static（只有一份）又是final（常量）的域只占据一段不能改变的存储空间。（常量池）一般都要大写，字与字之间用\_隔开，一般常用public修饰。（关键字顺序好像没什么要求）

当基本类型运用final时，使数值恒定不变；

比如：一个永不改变的编译时常量；

一个在运行时被初始化的值，而你不希望它被改变。

当对象引用运用final时，使引用恒定不变，一旦引用被初始化指向一个对象，就无法让他指向另一个对象，然而，对象其本身是可以修改的。

8.2 空白final：声明final但未给定初始值。在构造器中给定其值（一个类中的final域就可以做到根据对象而有所不同，却又保证其恒定不变的特性）。

其实就是一定要保证编译器在使用前初始化。

8.3 final参数：void without(final Gizmo g){

//! G = new Gizmo()};

你可以读参数，但不能改变他的值。（用来向匿名内部类传递数据,避免不了解的方法修改变量）

8.4 final方法：两个原因，第一个是方法锁定，防治任何继承类修改它的含义，出于设计的考虑，防治方法覆盖；

“方法覆盖”只有在某个方法是基类的接口的一部分时才会出现。即必须能将一个对象向上转型为它的基本类型并调用相同的方法。如果方法是private，它就不是基本类的接口部分，它仅是一些隐藏类中的程序代码，只不过具有相同的名字而已。此时你并没有覆盖方法，只是生成了一个新的方法。但如果导出类中以相同的名称生成一个public、protected或者包访问权限方法的话，该方法就不会产生在基类中出现的“仅具有相同名称”的情况。

Private和final的区别，private强于final，final只是不能覆盖

第二个是效率（内嵌），就是同意编译器将针对该方法的所有调用都转为内嵌调用。会跳过插入程序代码这种正常方式而执行方法调用机制，并且以方法体中的实际代码的副本来代替方法调用。这将消除方法调用的开销。（这种做法正被劝阻，让编译器和JVM去处理效率上的事）

8.5 类中的所有private方法都隐式的指定为final。即无法覆盖。

（protected）子类继承父类的方法，如果子类没有该方法，直接继承父类方法；如果子类有该方法则覆盖父类方法。

8.6 final类：当将整个类定义为final时，就表明了你不打算继承该类，而且也不允许别人这么做。这个已经很完美了，不需要修改。

9 %可以用于正整数，只用当被除数是负数时余数才是负的。

10 涉及浮点数的计算都是近似的，因为这些数没有准确的精度来存储。整数可以精确计算。Eg：System.out.println(1.0-0.1-0.1-0.1-0.1-0.1);显示为0.5000000001.

11副作用与顺序点

Eg：int a=3,b;  
b=(++a)+(++a);  
在电脑上面运行的结果是：b=9

Eg: double x=1.0, y=5.0, z=x-- + (++y);

Y=6, z=7, x=0

12 数据类型 ：

System.out.println( (int)1.7);

13 字符数据类型char表示单个字符：

Char letter = ‘A’;

Char numChar = ‘4’;

字符串直接量必须括在双引号中，而字符直接量是括在单引号中的单个字符。”A”, ‘A’

14 java支持统一码。ASCII码：char ch = ‘a’ 🡸=🡺 char ch = ‘ \u0041’;

15 单个字符不需要转译成其他数据类型。Eg：int a = ‘a’; (a=97)；

15.1 字符的运算所有的运算符都能用在char型操作数上，如果另一个操作数是一个数字或字符，那么char型操作数会被自动转换成一个数字；如果另一个操作数是一个字符串，字符会和字符串相连。

15.2 字符串操作，如果表达式以一个字符串起头，那么后续所有操作数都必须是字符串型（编译器会把双引号内的字符序列自动转成字符串）

Int a = 1;

Int b = 2;

String c = “total”;

Syso(c+a+b); ------total12

Syso(c+(a+b));------total3

16 关于数据溢出：

import java.util.Scanner;

public class Change{

public static void main(String[] args){

Scanner input = new Scanner(System.in);

System.out.print("Please enter the Change:");

double amount = input.nextDouble();

double remainingAmount = amount\*100;

System.out.println("the amount is :"+ remainingAmount+"the true Change is :"+ amount);

}

}

17 String 实际上与System类一样，都是一个java库中预定义的类。String类型不是基本类型而是引用类型（reference type）.

18 使用/\*\* \*/可以被jdk的javadoc命令提取出来放在HTML文档中，一般用于类或方法的前头。

19 double a = 4.87;

a = (int)(a\*100)/100 ;

20 boolean 数据类型：boolean lightsOn = true;

21 注意，同一个变量不要重复定义！

22 if (number % 2 == 0)

Even= true;

Else

Even=false;

23 如果有if else 语句时，要注意变量的赋值问题，如果变量a的值只有在选择块中给定，那么应该在程序之前给定a的值。（会有else语句没给a赋值的情况）（作用域空间问题）

24 switch(switch表达式){

Case value : 语句;

Break;

Default : 默认语句

}

25 三目运算符：boolean-expression ? expression1 : expression2;

26 赋值运算符是右结合：

Eg: a = b += c = 5 🡪 a = ( b += ( c = 5 ))

27 continue 只是跳出一次迭代，break是跳出整个循环；

27.1 return 也是仅仅跳出一层循环

注：当需要内层的return来传到外层时，

。。。。。**if**(map.containsKey(value)) {

**int** index = map.get(value) ;

**return** **new** **int**[]{i+1, index};

}

}

//这是一种解决return的数组不在最外层的方法

**return** **new** **int**[0];

28 在方法头中，需要对每一个参数进行独立的数据类型声明。

29 在其他语言中方法称为过程或函数，带返回值的方法称为函数，不带返回值类型的void方法叫过程。Java中的方法只能作为类的一部分来创建。方法只能通过对象来调用(静态方法是针对类调用的)

ReturnType methodName(/\*Argument list\*/){

/\*Method body\*/

}

29.1 声明一个静态方法，就是类的方法，可以不用实例的调用。（有点像工具方法）

29.2 在静态方法中，不能直接访问非静态成员(包括方法和变量)

30定义方法是指被定义的条目是什么，声明变量是指被声明的条目分配内存来存储数据。

31在Java中带返回值的方法也可以当作语句调用，这种情况下，函数调用者只需忽略返回值即可。

32 main方法与其他方法的唯一区别在于它是由Java虚拟机调用的。

33对于带返回值的方法而言，必须有return语句。

34每当调用一个方法时，系统会将参数，局部变量存储在一个栈中，当调用另一个方法时。调用者的堆栈空间保持不变，开辟新的空间处理新方法的调用。

35当void方法需要非正常终止时，可以用return 语句。

36 被重载的函数必须有不同的形参列表，不能通过返回值或修饰符的不同类别来区分。

36.1重载如果传入的数据类型小于方法中声明的形式参数类型，实际数据类型就会得到提升。Char型略有不同，如果无法找到恰好接受char参数的方法，就会把char直接提升至int型。

如果实际参数比形式参数大，必须通过类型转换来执行窄化转换。

37 声明引用数组的变量：

ElementType [ ] arrayRefVar ; eg: double [ ] mylist;

创建数组：

arrayRefVar = new elementType[arraySize];

eg: double[] myList = new double[10];

赋值：

arrayRefVar[index] = value ;

38 一个数组变量实际上存储的是指向数组的引用。

39创建数组后，数值型的基本数据类型默认值是0 ，char型的默认值为’\u0000’， boolean型的默认值为false。

40 arrayRefVar.length 可以得到数组的长度。

41数组初始化：

ElementType[] arrayRefVar = {value0 ,value1 ,…,valuek};

或者：ElementType[] arrayRefVar = new ElementType[]{value0 ,value1 ,…,valuek};

42对于char[]类型的数组，可以使用一条打印语句打印：

Char[] city = {‘d’ ,’a’ ,’l’ ,’l’ , ‘a’ ,’s’};

System.out.println(city);

43 for-each循环或增强循环：

For (elementType element: arrayRefVar){

//process the element

} 对List中的每个元素进行以下操作。不使用下标变量就可以顺序的遍历整个数组。

44 访问越界是数组程序中经常出现的问题，抛出的运行错误是ArrayIndexOutOfBoundsException。设计时，应避免所使用的下标不超过arrayRefVar.length-1.

45 数组的复制：数组的复制list1= list2 实际上是将数组的引用复制给了另一个变量，使得两个变量指向相同的内存地址空间。

复制数组有三种办法:1）使用循环语句逐个复制：

Int[] sourceArray = {2,3,1,5,10};

Int[] targetArray = new int[sourceArray.length];

For (int i= 0; i<sourceArray.length;i++){

targetArray[i] = sourceArray[i];}

2)使用java.lang.System类的arraycopy方法复制数组： System.arraycopy(sourceArray,0,targetArray,0,sourceArray.length);

45.1 把握住什么是不变的，对于一个链表来说，如果名字一直在覆盖，会找不到头指针，但是空间上他还是存在的，可以通过数组复制本质上是对于空间的复制这一特性，来固定头指针。

46 Java使用的是值传递的方法将实参传递给方法。传递基本数据类型变量的值与传递

数组值会有很大的不同。

Eg public class test{

Public static void main(String[] args){

Int x = 1;

Int[] y = new int[10];

m (x , y);

System.out.println(“x is ” + x);

System.out.println(“y[0] is ” + y);

}

Public static void m(int number , int[] number){

Number = 1001;

Numbers[0] = 5555;

}

}

47 从方法中返回数组：

Public static int[] reverse(int[] list){

Int[] result = new int[list.length];

For(int i = 0,j = result.length-1; i<list.length; i++,j--)

Result[j] = list[i];

Return result;

}

Int[] list1 = {1,2,3,4,5,6};

Int[] list2 = reverse(list1);

48 可变长度参数列表：

方法中声明如下：typeName…parameterName(类型名…参数名)

49 构造方法：Classname() 一般用大写

三个特殊性：

1. 构造方法必须具备和所在类相同的名字
2. 构造方法没有返回类型，甚至连void也没有。
3. 构造方法是在创建一个对象使用new操作符时调用的。构造方法作用是初始化对象

同样，构造方法也可以重载

一个类可以不定义构造方法，这种情况下，类中隐含定义一个方法体为空的无参构造方法。这个构造方法称为默认构造方法。

49.1 构造器确保初始化。创建对象时，为对象分配存储空间，并调用相应的构造器。

50 对象是通过对象引用变量来访问的:

Classname objectRefVar = new ClassName();

50.1 对象是一种引用变量，倘若“将一个对象赋值给另一个对象”，实际上是将“引用”从一个地方复制到另一个地方。改变一个将都发生改变。这就是“别名现象”

可以通过对象属性间的赋值避免：t1.value =t2.value

51 访问对象的数据和方法

通过圆点运算符（.）

objectRefVar.dataField引用对象的数据域

objectRefVar.method(参数)调用对象的方法

对于实例方法和数据，必须通过某个具体的实例，调用对象上的实例方法的过程称为调用对象

52 如果一个引用类型的数据域没有引用任何对象，那么这个数据域就有一个特殊的java值null (引用类型的数据域的默认值是null，数值类型的数据域的默认值是0，boolean类型的数据域的默认值是false，char类型的数据域的默认值是’\u0000’)

应该是静态变量吧？

注意：java没有给方法中的局部变量赋默认值：

Class Test {

Public static void main(String[] args){

Int x;

String y;

Sytem.out.println(“x is :” + x + ” y is ” + y);

}

}是错误的

53 每个变量赋值都代表一个储存值得内存位置。

对于基本类型变量来说，对应内存所存储的值是基本类型值。

对于引用类型变量来说，对应内存所存储的值是一个引用，是对象的存储地址。

对于基本类型变量来说，赋值以后就是变量的实际值赋给另一个变量，与原变量无关

对于引用变量来说，赋值以后只是指向了同一个对象。原对象会成为garbage，Java会自动检测并收回（garbage collection）

54 实例变量是绑定到类的某个特定实例的，他是不能被同一个类的不同对象所共享的

如果想要一个类的所有实例共享数据，就要使用静态变量（static variable），也称为类变量（class variable）在uml中静态方法和变量都是以下划线标注

E.g. static int numberOfObjects;

E.g. static int getNumberObjects(){return numberOfObjects;}

54.1 Static关键字：

当创建类的时候是在描述那个类的对象的外观与行为。除非用new创建那个类的对象，数据存储空间才会被分配，其方法才会被外界调用。

有时候只想为特定域分配单一存储空间，而不去考虑究竟要创建多少对象，甚至根本不创建任何对象。

有时候希望某个方法不与包含它的类的任何对象关联在一起。也就是说，即使没有对象，也能调用这个方法。

当声明了一个物体是Static时，就意味着这个域不会与它包含的那个类的任何对象实例关联在一起。即不同对象引用的Static值是一个存储空间中的，改变一个其他也会改变。

对于一个Static 的数据，可以通过一个对象去访问它，如st2.i；也可以通过类名直接引用，如StaticTest.i这对于非静态成员则不行。

54.2 static方法就是没有this的方法。在static方法的内部不能调用非静态方法，反过来是可以得（想全局方法）

55 变量名，方法使用小写，其他词的首字母大写；定名常量用大写字母命名；类名首字母大写，如果类名是由多个词构成，那么把他们并在一起，其中每个首字母采用大写形式。

56 如何判断一个变量是否应该是静态或实例：

如果一个变量与方法依赖于某个具体实例，那就应该将他定义为实例变量，方法

反正，应该定义为静态变量

任何实例的方法都能通过参数传递改成静态方法，有什么好坏呢

1. 有静态属性的类，一般会定义静态方法
2. 没有属性的类，一般会定义静态方法，这样可以通过类名.方法名直接调用，不用定义对象
3. 用于对静态字段、只读字段等的初始化
4. 添加static 关键字，不能添加访问修饰符，因为静态构造函数都是私有的
5. 静态构造函数是不可继承的，而且不能被直接调用
6. 如果类中包含用来开始执行的main方法，则该类的静态构造函数将在调用main方法之前执行。
7. 静态方法效率上要比实例化高，静态方法的缺点是不能自动销毁，而实例的可以做销毁

静态方法的弊端

1. 一些非常频繁的方法都使用静态方法可以提高系统性能。
2. 为了方便，不用实例化，但这样程序编译运行时就占用了系统资源，静态的属性和方法在程序启动时后，就全部装入内存的，而且不管这些方法属性以后是否应用
3. 所有访问者看到的静态属性的数据几乎是一样的
4. 静态方法/数据成员是属于类的，不是某个对象的，相当于共享变量

57 可见性修饰符

可以在类，方法和数据域前使用public修饰符。表示他们可以被任何其他的类访问。

如果没有使用可见性修饰符，那么默认为类、方法和数据域是可以被同一个包的任何一个类访问的。也称为包私有或包内访问

Private修饰符限定方法和数据域只能在它自己的类中被访问。

57.1 private可以控制访问构造器，通过调用特定方法来创建对象。任何可以肯定只是该类的一个“助手”方法的方法，都可以把它指定为private，以确保不会有包内的其他方法误用到它，也防治了你会去改变或删除这个方法。

58 在大多数情况下，构造方法应该是公共的。但是如果想防止用户创建类的实例，就应该使用私有构造方法：private Math(){},用户无法自己创建对象。

59 在定义私有数据域的类外的对象是不能访问这个数据域，但是经常有客户端需要存储、修改数据域的情况。为了能访问私有数据域，可以提供一个get方法返回数据域的值，为了能更新一个数据域，可以提供一个set方法给数据域设置新值。

注意:将数据域定义为私有数据域后，用方法去访问。

60 给方法传递对象参数/对象数组

Public static void printCircle(Circle3 c);

Circle[] circleArray = new Circle[10];

61 不可变对象和类，通常创建一个对象以后，他的内容是允许改变的。有时候，我们也需要一个对象，创建以后其内容是不能改变的。我们称作是不可变对象。

要使一个类是不可变的，它必须满足：

1. 所有数据域都是私有的
2. 没有修改器方法
3. 没有一个访问器方法，它会返回一个指向可变数据域的引用

Public class Student{

Private int id;

Private String name;

Private java.util.Date dateCreated;

Public Student(int ssn, String newName){

Id = ssn;

Name = newName;

dateCreated = new java.util.Date();

}

Public int getId(){

Return id;

}

Public String getName(){

Return name;

}

Public java.util.Date getDateCreate(){

Return dateCreated;

}

}

62 this引用，关键字this是指向调用对象本身的引用名。

构造函数名字跟类名一致，首字母要大写

一种常见的用法就是引用类的隐藏数据域，关键字this给出一种指代对象的方法，这样就可以在实例方法代码中调用实例方法。

关键字this的另一个常用方法是让构造函数方法调用同一个类的另一个构造方法。

构造器和方法使用关键字this有很大的区别。方法引用this指向正在执行方法的类的实例。静态方法不能使用this关键字，因为静态方法不属于类的实例，所以this也就没有什么东西去指向。构造器的this指向同一个类中，不同参数列表的另外一个构造器

62.1 对象调用方法的时候：a.peel(1); (Banana a = new Banana())

内部的表示形式为：banana.peel(a,1);他暗自把“所操作对象的引用”作为第一个参数传递给peel();

假如你希望在方法内部获得对当前对象的引用，需要用this。This只能在方法内部使用，表示“调用方法的那个对象”的引用。

注意：如果在方法内部调用同一个类的其他方法可以直接调用，不用用this。

62.2

**public** **class** Leaf {

**int** i = 0;

Leaf increase(){

i++;

**return** **this**;

}

**void** print(){

System.***out***.println("i = " + i);

}

**public** **static** **void** main(String[] args){

**new** Leaf().increase().increase().increase().increase().print();

}

}

63 方法抽象是通过将方法的使用和它的实现分离出来实现的。方法的实现细节封装在方法内，对使用方法的用户是隐藏的。这就称为信息隐藏（information hiding）或封装（encapsulation）。

64 类抽象，是将类的实现和使用分离。类的使用者不需要知道类是如何实现的，实现的细节经过封装，对用户隐藏起来，这称为类的封装。

65 对象的组合，一个对象可以包含一个对象，这两个对象之间的关系称为组合。

Public class Name{….}

Public class Address{….}

Public class Student{

Private Name name;

Private Address address;

}

如果一个对象只归属于一个聚集对象，那么它和聚集对象的关系就称为组合（实菱形）

如果一个对象可以被多个聚集对象所拥有，那么称之为聚集。（空菱形）

66 类的设计原则： 内聚性，一致性， 封装性， 清晰性， 完整性

67 在面向对象的程序设计中，可以从已有的类派生出新类，这叫做继承。

68 super关键字可以用于两种途径，调用父类的构造方法，调用父类的方法。

构造方法可以用来构造一个类的实例。不像属性和方法，父类的构造方法是不被子类继承的。它们只能从子类的构造方法中用关键字super调用。

68.1 当重写子类的构造方法时，一定要注意是否需要重写所有的内容，可以通过添加Super语句来确保原来的方法得到进行

69 构造一个类的实例时，将会调用沿着继承链的所有父类的构造方法。当构造一个子类的对象时，子类构造方法会在完成自己的任务之前，首先调用它的父类的构造方法。这个过程持续到沿着这个继承体系结构的最后一个构造方法被调用为止。

这就是构造方法链。

70 如果一个类要设计为扩展的，最好提供一个无参的构造方法以避免程序设计错误。最好能为每一个类提供一个无参构造方法，以便于对该类进行扩展时避免错误。

71 子类从父类中继承方法，有时子类需要修改父类中定义的方法的实现，这称做方法覆盖

72 Java中的每一个类都源于java.lang.Object类。Object 类中有toString()方法。对于元素是对象的话：toString会返回一个描述该对象的字符串。默认情况下，它返回一个由该对象所属的包名+类名，@，以及该对象十六制形式的内存地址组成的字符串。

对于元素是基本类型的话：会输出其值，例如在arraryList中存储的String会显示字符串。

72.01 这个是该对象的散列码，通过hashCode()方法产生的。

72.1 当编译器需要一个String类型而你却只有一个对象时，编译器会先找你重写的toString方法，如果没有则返回上述内容

72.2 syso(object.toString()); ==syso(object);

73 使用父类对象的地方都可以使用子类的对象，这就是多态。

多态就意味着父类型的变量可以引用子类型的变量

74 动态绑定 声明类型和实际类型

Object o = new GeometricObject();

System.out.println(o.toString);

一个变量必须被声明为某种类型，变量的这个类型称为它的声明类型。这里o的声明类型是Object。一个引用类型变量可以是一个null值或者是一个对声明实例的引用，实例可以是声明类型或它的子类型的构造方法创建。

变量的实际类型是被变量引用的对象的实际类，这里o的实际类型是GeometricObject。O调用的到底是哪个toString（）方法是由o的实际类型所决定的，这就是动态绑定。

75 对象转换

使用转换运算符把一种基本类型变量转换为另一种基本类型

也可以把一种类类型的对象转换为继承体系结构中的另一种类类型对象

Object o = new Student();----隐式转换

Object myObject = new Circle();

If(myObject instanceof Circle){

System.out.println(“The circle diameter is ” + ((Circle)myObject).getDiameter());

}

76 例子：

Public static void displayObject(Object object){

If(object instanceof Circle4){

System.out.println(“The circle area is “ + ((Circle4)object).getArea());

}

Else if(object instanceof Rectangle1){

System.out.println(“The circle area is “ + ((Rectangle1)object).getArea());

}

}

77 Object类中的equals方法。（不适用于基本类型(==；!=)）

签名：public Boolean equals(Object o)

测试两个对象是否相等：object1.equals(object2);

注意：使用签名equals（someClassName obj）是一个常见错误，应该用

equals（Object obj）

注Java中equals和==的区别

java中的数据类型，可分为两类：

1.基本数据类型，也称原始数据类型。byte,short,char,int,long,float,double,boolean

他们之间的比较，应用双等号（==）,比较的是他们的值。

2.复合数据类型(类)

当他们用（==）进行比较的时候，比较的是他们在内存中的存放地址，所以，除非是同一个new出来的对象，他们的比较后的结果为true，否则比较后结果为false。

3. JAVA当中所有的类都是继承于Object这个基类的，在Object中的基类中定义了一个equals的方法，这个方法的初始行为是比较对象的内存地址，但在一些类库当中这个方法被覆盖掉了，如String,Integer,Date在这些类当中equals有其自身的实现，而不再是比较类在堆内存中的存放地址了。

77.1 String 对象equals() 是比较两个对象的内容而不是引用（重写后）

将此字符串与指定的对象比较。当且仅当该参数不为 null，并且是与此对象表示相同字符序列的 String 对象时，结果才为 true。

77.2 public static void main(String[] args){

Integer a = new Integer(47);

Integer b = new Integer(47);

int c = 47;

int d = 47;

System.out.println(c==d); T

System.out.println(a==c); T

System.out.println(b==c); T

System.out.println(a==b); F(因为比较的是a和b的引用)

}

78 数组线性表ArrayList类 java.util.ArrayList

方法介绍：

ArraryList()

Add(o:object);void

Add(index:int, o:object);void 在特定位置上加入元素

Clear();void 删除表中所有元素

Contains(o:object);Boolean 查找是否包含一个元素

Get(index:int);object

indexOf(o:object);int 返回线性表中第一个匹配元素的下标

isEmpty();Boolean

lastIndex(o:object);int 返回线性表中最后一个匹配元素的下标

remove(o:object);boolean 删除元素o

size();int 已插入多少对象

remove(index:int)boolean 删除指定下标

set(index:int, o:object);object

77 protected 数据和方法。经常需要允许子类访问定义在父类中的数据域或方法，但是不允许非子类访问这些数据域和方法。可以使用关键字protected完成该功能。父类中被保护的数据域或方法可以在它的子类中访问。

77.1 最好的方式还是将域保持为private，而且你应该一直保留“更改底层实现”的权利，然后通过规定protected方法来控制类的继承者的访问权限。

77.2 protected 数据域同包可访问。

77.3 public 完全访问权限，包外包内

Protected 不完全访问权限，包内，包外的子类(子类访问的时候是通过新建子类对象，来访问父类方法)

Private 限制访问权限 包内的该类自己

78 终极类或方法防止扩展或覆盖

Public final class C{

}

Public class Test{

Public final void m(){

}

}

79 异常的创建，抛出，捕获和处理异常

Try{

Code to try

Throw an exception with a throw statement or from method if necessary;

More code to try

}

Catch(type ex){

Code to process the exceprion;

}

80 nextLine() and nextInt()…nextLine()会丢弃当前行，只需重新输入

81 异常处理的好处。它能使方法抛出一个异常给它的调用者。这个调用者可以处理该异常，而被调用的方法通常不知道在出错情况下该做些什么。

最根本的优势就是将检测错误（由调用的方法完成）从错误处理中分离出来。

82 异常种类：

Error：有java虚拟机抛出

Exception：由程序和外部环境所引起的错误

RuntimeException：运行时异常

83 接口是为了定义多个类的共同行为。

84 抽象类，一个父类设计的非常抽象，以至于它都没有任何具体的实例。这样的类叫抽象类。

同理，在类中可以定义一些方法，但是它们的实现取决于几何对象的具体类型，这样的方法称为抽象方法，在方法头中使用abstract修饰符表示。在类头使用abstract修饰符表示该类为抽象类。

抽象类的好处，父类对象声明的子类实例可以直接调用父类的抽象类而不用转换为子类类型。

**abstract** **class** Cup{

}

**class** TeaCup **extends** Cup{

**void** addTea(){

System.***out***.println("add Tea to the Cup");

}

}

**abstract** **class** SecondCup{

**abstract** **void** addTea();

}

**class** SecondTeaCup **extends** SecondCup{

**void** addTea(){

System.***out***.println("add second Tea to the Cup");

}

}

**public** **class** Test4 {

**static** **void** trans(Cup var){

((TeaCup) var).addTea();

}

**static** **void** trans(SecondCup var){

var.addTea();

}

**public** **static** **void** main(String[] args){

Cup x = **new** TeaCup();

// ((TeaCup) x).addTea();

*trans*(x);

SecondCup y = **new** SecondTeaCup();

// y.addTea();

*trans*(y);

}

}

注意：

1. 抽象方法不能包含在非抽象类中，如果继承抽象父类的子类不能实现父类所有的抽象方法，那么子类也必须定义为抽象的。而且抽象方法是非静态的。
2. 抽象类不能用new操作来初始化。但是仍然可以定义它的构造方法，这个构造方法在子类中调用，
3. 包含抽象对象的类必须是抽象的。但是可以定义一个不包含抽象方法的抽象类，这种情况下，不能使用new来创建该类的实例，这种类是用来定义新子类的基类的。
4. 即使父类是具体的，这个子类可以是抽象的。
5. 子类可以覆盖父类的方法并将他定义为abstract。当父类的方法实现在子类中变的不合法时是很有用的。
6. 不能用new创建抽象类实例，但是抽象类可以作为一种数据类型

GeometricObject[] object = new GometricObject[10];

85 接口是一种与类相似的结构，只包含常量和抽象方法。它的目的是指明多个对象的共同行为

修饰符 interface 接口名{

//常量声明

//方法签名

}

85.1

一个class可以包括多个接口

86 使用implements关键字让对象的类实现这个接口来完成，类和接口之间的关系称为接口继承：

Public interface Edible{

Public abstract String howToEat();

}

Class Chicken extends Animal implements Edible{

Public String howToEat(){

Return “fry it”;

}

}

Abstract class Fruit implements Edible{

}

Class Apple extends Fruit{

Public String howToEat(){

Return “make a juice ”;

}

}

87 ActionListener接口，按钮是动作的源对象，需要创建一个对象来处理按钮上的动作事件。这个对象称为监听器。

一个对象要成为源对象上动作事件的监听器，需要满足两个条件：

1. 这个对象必须是ActionListener接口的一个实例。该接口定义了监听器的所有共有动作
2. ActionListener对象listener必须使用方法source.addActionListener(listener)注册给源对象。

ActionListener接口包含处理事件的actionPerformed方法。监听器必须覆盖该方法来响应事件。

89 Comparable界面：比较

90 Cloneable接口：

91 接口和抽象类

Java只允许为类做单一继承，但是允许使用接口做多重扩展

Public class NewClass extends BaseClass implements Interface1, …., InterfaceN{

}

接口可以继承其他接口，这样的接口称为子接口

Public interface New Interface extends Interface1, Interface2, …{

}

所有的类共享一个根类Object，但是接口没有共同的根。

与类相似，接口也可以定义一种类型，一个接口类型的变量可以引用任何实现该接口的类的实例。

注意：

抽象类和接口都是用来明确多个对象的共同特征的，

一般来说，详细描述父子关系的强是关系（strong is-a relationship）应该用类建模。例如，公历是一种日历，所以应该用类建模

若是弱关系（weak is-a relationship）也称为类属关系，他表明对象拥有某种属性，可以用接口来建模。例如，所有的字符串都可以比较，String类实现Comparable接口。

通常推荐使用接口而非抽象类是因为接口可以定义不相关类共有的父类型，接口比类更加灵活。

91.1 interface 使抽象的概念更近了一步。Abstract允许人们在类中创建一个或多个没有任何定义的方法——提供了接口部分，但是没有具体实现，由此类的继承者创建。Interface产生一个完全抽象的类，它根本没有提供任何具体的实现，它允许创建者确定方法名，参数列表和返回类型。接口只提供了形式，而未提供任何具体

实现。

91.2 一个接口表示：所有实现了该特定接口的类看起来都像这样。（接口可以用来建立类与类的协议。

91.3 类和接口很像，一个class中只能有一个public

91.4 接口的方法隐式的默认为public，所有实现它的类的方法都应该定义的public。

在接口中定义的方法必须显式的定义为public

91.5 接口可以提供多种基类分类，将一个对象在不同需求下分配为不同基类的子类型。

91.6 接口可以完全解耦：对于类和方法之间耦合过紧，通过将类设计为接口。复用代码的第一种方式是客户端程序员遵循该接口来编写他们的类。

当你无法修改你想要使用的类的时候：第二种方式是通过适配器模式。适配器中的代码将拥有你所拥有的所有接口，并产生你所需要的接口。（包装了一下）

91.7 抽象工厂，抽象类和接口。接口的实现类必须完全满足接口的声明，如果实现接口的类的返回类型是接口的子类或没有完全实现接口声明的方法时，必须用抽象类（抽象类可以实现接口的几个功能，然后在用子类继承抽象类，创建对象）

**class** Change{

**public** String name(){

**return** getClass().getName();

}

**public** String chang(String s){

**return** "this is change " + s;

}

}

**class** LowChange **extends** Change{

**public** String chang(String s){

**return** "this is Lowchange" + s;

}

}

**class** HighChange **extends** Change{

**public** String chang(String s){

**return** "this is Highchange" + s;

}

}

**class** FilterChange **implements** Porcessor{

Change change;

**public** FilterChange(Change change){

**this**.change = change;

}

**public** String name(){

**return** change.name();

}

**public** String chang(Object o){

**return** change.chang("Filter");

}

}

**interface** Porcessor {

String name();

Object chang(Object o);

}

**class** GetPorcessor **implements** Porcessor{

**public** String name(){

**return** "lee";

}

**public** String chang(Object s){

**return** "this is Interface";

}

}

**class** TakePorcessor **extends** GetPorcessor{

**public** String chang(Object s){

**return** "this is TakePorcess and Interface";

}

}

**public** **class** Porcess{

**static** **void** proc(Porcessor p){

System.***out***.println(p.chang("lee"));

}

**public** **static** **void** main(String[] args){

GetPorcessor g = **new** GetPorcessor();

TakePorcessor t = **new** TakePorcessor();

Change c = **new** Change();

HighChange h = **new** HighChange();

LowChange l = **new** LowChange();

*proc*(g);

*proc*(t);

*proc*(**new** FilterChange(c));

*proc*(**new** FilterChange(h));

*proc*(**new** FilterChange(l));

}

}

91.8 使用接口的核心原因：为了能够向上转型为多个基类型（以及由此而带来的灵活性）

第二个使用接口的原因却是与使用抽象基类相同：防止客户端程序员创建该类的对象，并确保这仅仅是一个接口。

91.9 一个类遵循一个接口时使用implements，一个接口扩展一个接口用extends

91.10 避免接口使用相同的名字，组合接口的时候出现覆盖。

91.11 放入接口的任何域都自动是static和final的，在接口定义的域不能是空final

91.12 接口实现了多重继承的途径，而生成遵循某个接口的对象的典型方式就是工厂模式。

92 事件：能创建一个事件并触发该事件的组件称为源对象或源组件

事件类的根类是Java.util.EventObject

EventObject :

1. AWTEvent
   1. ActionEvent
   2. AdjustmentEvent
   3. ComponentEvent
      * 1. ContainerEvent
        2. FocusEvent
        3. InputEvent

* MouseEvent
* KeyEvent
  + - 1. PaintEvent
      2. WindowEvent
  1. ItemEvent
  2. TextEvent

1. ListSeletionEvent
2. ChangeEvent

如果以个组件可以触发某个事件，那么这个组件的任意子类都可以触发同类型的事件

93 Java使用一种基于委托的模型来处理事件:源对象触发一个事件，对此事件感兴趣的对象会处理它，将对此事件感兴趣的对象称为监听。两个要素：

1. 监听器对象的类必须是相应的事件监听器接口的实例，以确保监听器有处理这个事件的正确方法。Java为每一种类型的事件都提供了监听器接口。通常，事件XEvent的监听器接口命名为XListener。包含事件处理的监听器接口称为处理器（handler）的方法。
2. 监听器对象必须由源对象注册。一般来说，XEvent的注册方法命名为addXListener。一个源对象可以触发几种类型的事件。

94 字段和方法：

字段可以是任何基本类型的对象，也可以通过其引用与其通信，如果字段是对某个对象的引用，那么必须初始化该引用，以便使其与一个实际对象相关联。

对于基本数据类型，即使没有Java也会确保它获得一个默认值。

94.1 对于局部变量，必须初始化，否则报错；对于类的数据成员（即字段）系统会初始化，类的每个基本数据成员保证有一个初始值。

在类里定义一个对象引用时，如果不将其初始化，此引用会获得一个特殊值null。

95 C++ 中Struct和Class是一样的，区别是Struct默认是公有的，Class默认是私有的。

96 方法一般只能通过对象才能调用，但是static方法是针对类调用的，不依赖于对象的存在。

97 发送消息给对象

Int x = a.f()

返回值类型必须兼容

98 javadoc便于提取注释的工具/\*\* \*/

@独立文档标签

99 自动装箱和拆箱：（包装器）

装箱： Integer a = new Integer() ;

a = 100 ; //1.5以前不支持为对象如此赋值

拆箱： int b = new Integer(100) ;

基本数据类型与包装类对应关系如下：

short Short；

int Integer；

long Long；

char Char；

float Float；

double Double；

boolean Boolean；

100 与或非（&&；||；！）操作只可以应用于布尔值。与C++和C不同的是：不可将一个非布尔值当作布尔值在逻辑表达式中使用。

101 foreach 语法：

F[]

For (float x : f)

Syso(x);

这条语句定义了一个float类型的变量x，继而将每一个f中的元素赋值给x。

102 finalize() ；终结处理和垃圾回收

在C++中，对象一定被销毁；而在Java中对象并非总是被垃圾回收

1：对象可能不被垃圾回收

2：垃圾回收并不等于“析构”

3：垃圾回收只与内存有关。

Finalize()可以发现程序中哪些对象存在没有被适当清理的部分（程序存在隐晦的缺陷）——发现缺陷

**class** Book{

**boolean** checkedOut = **false**;

Book(**boolean** check){

checkedOut = check;

}

**void** checkIn(){

checkedOut = **false**;

}

**protected** **void** finalize(){

**if**(checkedOut)

System.***out***.println("Error : checked out");

}

}

**public** **class** TerminationCondition {

**public** **static** **void** main(String[] args){

Book novel = **new** Book(**true**);

novel.checkIn();

//forget to cleanup

**new** Book(**true**);

System.*gc*();

}

}

102.1 java提供finalize()方法，垃圾回收器准备释放内存的时候，会先调用finalize()。

(1).对象不一定会被回收。

(2).垃圾回收不是析构函数。

(3).垃圾回收只与内存有关。

(4).垃圾回收和finalize()都是靠不住的，只要JVM还没有快到耗尽内存的地步，它是不会浪费时间进行垃圾回收的。

102.3

1 java的GC只负责内存相关的清理，所有其它资源的清理必须由程序员手工完成。要不然会引起资源泄露，有可能导致程序崩溃。

2 调用GC并不保证GC实际执行。

3 finalize抛出的未捕获异常只会导致该对象的finalize执行退出。

4 用户可以自己调用对象的finalize方法，但是这种调用是正常的方法调用，和对象的销毁过程无关。

5 JVM保证在一个对象所占用的内存被回收之前，如果它实现了finalize方法，则该方法一定会被调用。Object的默认finalize什么都不做，为了效率，GC可以认为一个什么都不做的finalize不存在。

6 对象的finalize调用链和clone调用链一样，必须手工构造。

103 指定初始化： 方法可以向前调用，但是变量的初始化必须在调用它之前。

对于通过构造器初始化，是无法阻止自动初始化的进行的，它将在构造器被调用之前发生。

104 初始化的顺序问题，当创建了一个对象以后，首先会初始化该类的所有成员变量，然后再执行构造器，成员的初始化顺序依赖于其定义的先后顺序。先静态，后非静态。对于非静态变量，每次new新的对象都要重新初始化，静态变量不用。

105 A class file was not written. The project may be inconsistent, if so try refreshing this project and building it

eclipse中编译出现该问题

检查自己的类名...

我的问题是使用了 类名: con

con是操作系统保留的一个设备名字，不可以使用该类名..

其他的比如 CON, PRN, AUX, CLOCK$, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9都不可以

106 一个文件中只能有一个类是public的，除了内部类

107 可变参数列表（数组）：

旧：通过void print(Object[] args)创建数组

新：通过void print(Object… args)创建数组

108 定制工具类：

Package tool;

Class Print{

Public static void print(Object obj){

Syso(obj);}

Package Test;

Import static tool.Print.\*;

Class test{

Print(“lee”);

}

108.1 静态导入：导入静态变量，可以在其他包中使用另一个包的静态变量。

109 Java 访问权限修饰符P116

110 组合语法：将对象引用置于新的类中即可。

定义的地方；类的构造器；就在正要使用之前（惰性初始化）；使用实例化。

继承语法：除非已明确的指出要从其他类中继承，否则就是在隐式的继承JAVA的标准根类Object。

Extends 自动获得基类中所有的域和方法

110.1 组合和继承的应用场景区别：

组合和继承都允许在新的类中放置子对象，组合是显式这么做，继承是隐式。

组合技术通常用于想在新类中使用现有类的功能而非它的接口这种情况。即在 新类中嵌入某个对象，让其实现所需要的功能。Has-a

继承的时候，是使用某个现有类，并开发一个它的特殊版本。意味着你在使用一个通用类，并为了某种特殊需要把他特殊化。Is –a

慎用继承，除非是需要向上转型。

111 关于基类的初始化:在构造器中调用基类构造器来执行初始化。基类构造器具有执行基类初始化所需的所有知识和能力，Java会自动在导出类的构造器中插入基类构造器的调用。

构造过程是从基类“向外扩散的”。所以基类在导出类构造器可以访问它之前就已经完成了初始化。

注：使用导出类时，必须初始化基类。而且在导出类的构造器中，第一件事必须是初始化基类。

111.1 类的3种关系，组合；继承；代理。

代理：介于组合和继承之间。我们将一个成员对象置于所要构造的类中（就像组合），但与此同时我们在新的类中暴露了该成员对象的所有方法（就像继承）

不同于继承将所有的方法导出，代理创建一个对象后通过对该对象封装改写基类的方法。从而得到继承基类方法的效果。只得到想要的方法。

112 try{}finally{}：关键字try表示，下面的块（用大括号括起来的范围）是所谓的保护区（guarded region），这意味着它需要被特殊处理。其中一项特殊处理就是无论try块是怎样退出的，保护区后的finally子句的代码总是要执行。

113 名称屏蔽：如果java的基类拥有某个已被多次重载的方法名称，那么在导出类中重新定义该方法名称并不会屏蔽其在基类中的任何版本(与C++不同)。

可以使用与基类完全相同的特征签名及返回类型来覆盖具有相同名称的方法。（可以添加@Override注释来注解）。

114 初始化顺序：加载问题，类的代码在初次使用它的时候才载。

先static ——对象（如果有）——域的初始化

(父静态，子静态，父成员，父构造，子成员，子构造)

115 容器（泛型）和数组的区别：数组采用[]方式来访问元素，而List使用add()和get()这样的方法

随着自动包装机制的出现，容器已经可以与数组几乎一样方便地用于基本类型中了，数组仅存的优点就是效率。然而如果要解决更一般化的问题，那数组就可能会受到过多的限制，此时在这些情况下还是会使用容器。

116 数组的length方法表示数组对象可以存储多少对象，而不是实际存储元素个数，并且“[]”语法是访问数组对象的唯一方式。

117 容器类: list、set、queue、map 存储对象（也称为集合类）

容器还有一些其他的特性。例如，Set对于每个值都只保存一个对象、Map是允许你将某些对象与其他一些对象关联起来的关联数组，java容器类可以自动的调整自己的大小。

ArrayList保存的是Object，所以不同的类的对象都能add进去，当你使用get方法取出对象的时候，取到的只是Object引用，必须把他转型为你要的类型，比如Apple。((Apple)apples.get(i));

Java使用泛型来定义类太复杂，可以运用预定义的泛型，例如如果你要用ArrayList保存Apple对象，可以声明ArrayList<Apple>，而不是仅仅用ArrayList。

对于容器，一定要注意不要把错误的对象也放进去。

对于存储相同对象的容器，不需要使用索引，可以用foreach语法来选择List中的每个元素。

118 容器的基本概念：

用途是保存对象。Collection。一个独立元素的序列，这些元素都服从一条或多条规则。List必须按照插入的顺序保存元素。Set不能有重复元素。Queue按照排队规则来确定产生的顺序（通常和他们的插入顺序相同）。（Collection是一个槽一对象）

Map。一组成对额“键值对”对象，允许你使用键来查找值。ArrayList允许你使用数字来查找值（将数字与对象关联在了一起）。映射表允许我们用另一个对象来查找某个对象。（Map是一个槽两个对象

）

119 添加一组元素。Collection.addAll()成员方法的对象只能是另一个Collection对象作为参数。Arrays.asList()或者Collections.addAll()更加灵活，这两个方法使用的都是可变参数列表。

120 Arrays.toString()打印数组，输出数组的2个方式，循环和这个数组输出

Arrays.asList()接受一个数组或是一个逗号分隔的元素。

可以通过toArray()方法，将任意的Collection转换为一个数组：

Object[] o = pets.toArray();

121 list接口在collection的基础上又添加了大量的方法。有两种类型的List：

ArrayList基本的。它长于随机访问元素，但是在List的中间插入和移除元素时比较慢。

(方法：list.sublist(index1,index2))包括1，不包括2

List.removeall(sublist) 移除list中的sublist

LinkedList。它通过代价较低的在List中间进行的插入和删除操作，提供了优化的顺序访问。LinkedList在随机访问方面相对比较慢，但是他的特性集较ArrayList更大。

122 迭代器，轻量级对象，只能单向移动。只关注于功能，不用为容器中的对象确切类型操心。

122.1 迭代器listlterator只能用于各种List类的访问，可以双向移动。有set()方法。

122.2 迭代器相当于一个指针列表，是辅助在容器之上。在容器中修改。

122.3 next方法自动取到下一个数并往后移动一位

123 TreeSet有一个问题，用TreeSet.add()方法的时候遇到Source not found。

124 容器添加一组对象

1： Collection<Integer> colletion = new ArrayLisy<Integer>(Arrays.asList(1,2,3));

2: Integer[] moreInts = {1,2,3};

Collection.addAll(Arrays.asList(moreInts));

3: Collections.addAll(collection, 11,12,13);

4: Collections.addAll(collection, moreInts);

5: List<Integer> list = Arrays.asList(11,12,13);

125 栈(后进先出)；栈在Java中经常用来对表达式求值。Stack

126 假设编译通过，从java生命周期中的装载——连接——初始化分为3个部分：验证、准备、解析，

装载：查找和导入类或接口的二进制数据；

　　链接：执行下面的校验、准备和解析步骤，其中解析步骤是可以选择的；

　　校验：检查导入类或接口的二进制数据的正确性；

　　准备：给类的静态变量分配并初始化存储空间；

　　解析：将符号引用转成直接引用；

初始化：激活类的静态变量的初始化Java代码和静态Java代码块。

127 Set中最常用的就是测试归属性，正因如此，查找就成了Set中最常用的操作，通常选择HashSet实现。它专门对快速查找进行了优化。

Set具有Collection完全一样的接口，因此没有额外的功能。实际上Set就是Collection，只是行为不同：（这是继承和多态思想的典型应用：表现不同的行为）

Set是通过对象的值来确定归属性。

HashSet使用的是散列函数，TreeSet将元素存储在红黑树数据结构中，LinkedHashSet因为查询速度原因也使用了散列，但是使用链表来维护元素插入顺序。

128 HashSet : addAll；containAll；

129 Map:

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

m.put<>放入元素；m.get(键)得到键的值；m.keySet返回所有key视图；m.value返回所有value

130 Queue通过LinkedList实现，Queue窄化了LinkedList的方法的访问权限。

131 对于%来说，速度比减法差很多

132 对象的创建过程：（Dog类的对象）

1. 即使没有显式的使用static关键字，构造器实际上也是静态方法。因此当首次创建类型为Dog的对象时，或者Dog类的静态方法/静态域首次被访问时，java解释器必须查找类路径，以定位Dog.class
2. 然后载入Dog.class（创建一个Class对象），有关静态初始化的所有动作都会执行。因此，静态初始化只在Class对象首次加载的时候进行一次。
3. 当用new Dog()创建对象时，首先将在堆上的Dog对象分配足够的存储空间。
4. 这块存储空间会被清零，这就自动的将Dog对象中的所有基本类型数据都设置成了默认值，而引用则被设置成了null。
5. 执行所有出现于字段定义处的初始化动作。
6. 执行构造器。尤其涉及继承的时候

132.1 编译器会在每个构造方法的第一行隐式添加父类的默认无参构造器，即添加super()。

Super关键字可以用来选择调用父类的那个构造函数。

133 判断是不是一个方法只能从俩个方面:一个是方法名；第二个是形参是否一样。不能从返回值判断。

134 适配接口：接口最吸引人的原因就是允许一个接口有多个不同的具体实现。

135 对于接口的选择：任何抽象都应该是应真正的需求而产生的。当必须时，你应该重构接口而不是到处添加额外级别的间接性，并由此带来复杂性。

恰当的原则应该是优先选择类而不是接口。从类开始，如果接口的必须性变得非常明确，那么就进行重构。接口是一种工具，但它很容易被滥用。

136 通过异常处理错误：

JAVA的基本理念是：结构不佳的代码不能运行。

把“描述正常代码执行过程中做什么事”的代码和“出了问题怎么办”的代码相分离。

136.1 异常执行流程：

首先，同Java中的其他对象的创建一样，将使用new在堆上创建异常对象。然后，当前的执行路径（它不能在执行下去）被终止，并且从当前环境中弹出对异常对象的引用。此时，异常处理机制接管程序，并开始寻找一个恰当的地方来继续执行程序。这个恰当的地方就是异常处理程序，它的任务就是将程序从错误的状态中恢复，以使程序要么换一种方式运行，要么继续运行下去。

136.2 通常，异常对象中仅有的信息就是异常类型，除此之外不应该包含任何其他意义的内容。

137 异常处理有两种基本模型，终止模型和恢复模型。

终止模型将假设错误非常关键，以至于程序无法返回到异常发生的地方继续执行。

恢复模型意思是异常处理程序的工作是修正错误，然后重新尝试调用出错的方法。

恢复模型不是很实用

138 异常说明：关键字为：throws。如果在方法中要throw 一个异常，必须在方法名字声明：

**public** **void** f() **throws** SimpleException{

System.***out***.println("throws SimpleException from f()");

**throw** **new** SimpleException();

}

在监控区域（guarded region）可以直接throw：

**try** {

**throw** **new** SimpleException();

} **catch** (SimpleException e) {

System.***out***.println("Caught it");

}

139 常见的Exception

NullPointerException: Thrown when an application attempts to use null in a case where an object is required.

ArrayIndexOutOfBoundsException

140 异常链 （cause对象作为参数）

常常会想要在捕获一个异常后抛出另一个异常，并且希望把原始的异常信息保存下来，这个被称为异常链。

在Throwable的子类中，只有Error、Exception、RuntimeException提供了带cause参数的构造器。

141 重新抛出异常，如果在catch中抛出了与捕获exception类型相同的异常，printStackTrace()获得的是上一个异常，fillInStackTrace()获得的是当前的异常。这种方法可以保存住上一个异常。

如果在catch中抛出了与捕获exception类型不相同的异常，有关原来异常发生点的信息会丢失。

142 i++会先用i参与运算，然后再在i上加1。

**public** **static** **void** main(String[] args) {

**int** count = 0;

**while** (**true**) {

**try** {

**if** (count++ == 0)

**throw** **new** ThreeException();

System.***out***.println("No exception");

} **catch** (ThreeException e) {

// **TODO**: handle exception

System.***out***.println("ThreeException");

}**finally**{

System.***out***.println("In finally clause");

**if**(count == 2)

**break**;

}

}

}

注：当java中的异常不允许我们回到异常的抛出点时，可以把try放入循环里，建立一个“程序继续执行前必须要达到”的条件，还可以加一个static类型的计数器或者别的什么装置，使循环在放弃之前能尝试一定的次数。

143 对于没有析构函数和垃圾回收自动调用机制的语言来说，finally非常重要。

Java中，当要把除内存之外的资源恢复到他们的初始状态时，就要用到finally子句。这种需要清理的资源包括：已经打开的文件或网络连接，在屏幕上显示的图形，甚至可以是外部世界的某个开关。

143.1 如果扔出一个没有具体内容的Exception,如果try中调用了这个方法，必须分别补全2个catch，分别说明当出现OnOffException1，和OnOffException2的情况。

144 构造器的清理问题：基本原则是在创建需要清理的对象之后，立即进入一个try-finally语句块。

145 File类：list()方法，返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中的文件和目录。

list(FilenameFilter filter)：返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中满足指定过滤器的文件和目录。

构建File()类的时候：

对于 Microsoft Windows 平台，包含盘符的路径名前缀由驱动器号和一个 ":" 组成。如果路径名是绝对路径名，还可能后跟 "\\"。UNC 路径名的前缀是 "\\\\"；主机名和共享名是名称序列中的前两个名称。没有指定驱动器的相对路径名没有前缀。

145.1 两种方法：”/”或者“\\”

146 找不到或无法加载主类：如果命令行不能加载java类，一个原因是class配置没好，另一个就是可能java文档中保留了package信息，应该删除。

147 BufferedReader类的readline()方法如果已经达到流的末尾，就会返回null。

这就意味着readline()只能给一个String赋值一次

148 String 和StringBuffer，String是一个常量，一个对象（整体感）；StringBuffer更多的像一个容器，有append()的感觉。

String的内部结构是通过StringBuilder实现的

String 是值传递，StringBuffer 是引用传递

148.1 而在某些特别情况下， String 对象的字符串拼接其实是被 JVM 解释成了 StringBuffer 对象的拼接，所以这些时候 String 对象的速度并不会比 StringBuffer 对象慢，而特别是以下的字符串对象生成中， String 效率是远要比 StringBuffer 快的：

String S1 = “This is only a” + “ simple” + “ test”;

StringBuffer Sb = new StringBuilder(“This is only a”).append(“ simple”).append(“ test”);

你会很惊讶的发现，生成 String S1 对象的速度简直太快了，而这个时候 StringBuffer 居然速度上根本一点都不占优势。其实这是 JVM 的一个把戏，在 JVM 眼里，这个

String S1 = “This is only a” + “ simple” + “test”; 其实就是：

String S1 = “This is only a simple test”; 所以当然不需要太多的时间了。但大家这里要注意的是，如果你的字符串是来自另外的 String 对象的话，速度就没那么快了，譬如：

String S2 = “This is only a”;

String S3 = “ simple”;

String S4 = “ test”;

String S1 = S2 +S3 + S4;

这时候 JVM 会规规矩矩的按照原来的方式去做

148.2 在大部分情况下 StringBuffer > String

StringBuffer

Java.lang.StringBuffer线程安全的可变字符序列。一个类似于 String 的字符串缓冲区，但不能修改。虽然在任意时间点上它都包含某种特定的字符序列，但通过某些方法调用可以改变该序列的长度和内容。

可将字符串缓冲区安全地用于多个线程。可以在必要时对这些方法进行同步，因此任意特定实例上的所有操作就好像是以串行顺序发生的，该顺序与所涉及的每个线程进行的方法调用顺序一致。

StringBuffer 上的主要操作是 append 和 insert 方法，可重载这些方法，以接受任意类型的数据。每个方法都能有效地将给定的数据转换成字符串，然后将该字符串的字符追加或插入到字符串缓冲区中。append 方法始终将这些字符添加到缓冲区的末端；而 insert 方法则在指定的点添加字符。

例如，如果 z 引用一个当前内容是“start”的字符串缓冲区对象，则此方法调用 z.append("le") 会使字符串缓冲区包含“startle”，而 z.insert(4, "le") 将更改字符串缓冲区，使之包含“starlet”。

在大部分情况下 StringBuilder > StringBuffer

java.lang.StringBuilde

java.lang.StringBuilder一个可变的字符序列是5.0新增的。此类提供一个与 StringBuffer 兼容的 API，但不保证同步。该类被设计用作 StringBuffer 的一个简易替换，用在字符串缓冲区被单个线程使用的时候（这种情况很普遍）。如果可能，建议优先采用该类，因为在大多数实现中，它比 StringBuffer 要快。两者的方法基本相同

149 I/O重定向操纵的是字节流，而不是字符流：

SetIn(InputStream);SetOut(PrintStream);SetErr(PrintStream)

150 System.arraycopy()，当复制一维数组时，他是值传递，即改变copy副本不影响原始数组；当复制的是二维数组时，复制传递的是一维数组的一个引用，改变副本影响原始函数

151封装的层数越多，传入的参数应该就越少。得到的实际函数越贴近现实需求

152 StringBuffer和StringBuilder的区别：

1. 在执行速度方面的比较：StringBuilder > StringBuffer

2. StringBuffer与StringBuilder，他们是字符串变量，是可改变的对象，每当我们用它们对字符串做操作时，实际上是在一个对象上操作的，不像String一样创建一些对象进行操作，所以速度就快了。

3. StringBuilder：线程非安全的

StringBuffer：线程安全的

当我们在字符串缓冲去被多个线程使用是，JVM不能保证StringBuilder的操作是安全的，虽然他的速度最快，但是可以保证StringBuffer是可以正确操作的。当然大多数情况下就是我们是在单线程下进行的操作，所以大多数情况下是建议用StringBuilder而不用StringBuffer的，就是速度的原因。

对于三者使用的总结：

1.如果要操作少量的数据用 = String

2.单线程操作字符串缓冲区 下操作大量数据 = StringBuilder

3.多线程操作字符串缓冲区 下操作大量数据 = StringBuffer

153 用并发解决的问题通常分为“速度”和“设计可管理性”两种

154 将线程Runnable对象转变为工作任务的传统方式是把它提交给一个Thread构造器。Thread构造器只需要一个Runnable对象，调用start():完成初始化

154.1注意，Runnable必须重写run（）方法

155 float 是区分大小写的，注意哪些是大写哪些是小写

156 内部类：当生成一个内部类对象时， 此对象与制造它的外围对象之间就有了一种联系，所以它可以访问其外围对象的所有成员。

157 .this和.new:通过this生成外部对象的引用；如果要创建内部类对象，必须使用外部类对象来创建内部类对象

158 字符串是不可变对象；它们的值在创建之后不能更改。所有String类中看起来会改变String值得方法，实际上都是创建了一个全新的String。字符串缓冲区支持可变的字符串。因为 String 对象是不可变的，所以可以共享。

158.1 String对象一旦声明则不能轻易改变，如果要改变则需要先断开原有的对象引用，再开辟新的对象，之后再指向新的对象空间。

158.2 String a = “1”;//会创建一个String的对象

String b = a; //不会创建一个String的对象，对象b直接指向a数据“1”的首地址。

a = “2”; //创建一个新的数据空间“2”然后让a指向其首地址。“1”没变

159 String str = "abc"; 等效于：char data[] = {'a', 'b', 'c'};

String str = new String(data);

160 字符串串联是通过 StringBuilder（或 StringBuffer）类及其 append 方法实现的。字符串转换是通过 toString 方法实现的，该方法由 Object 类定义，并可被 Java 中的所有类继承。

161 javap –c Test 反编译Test.class 文件。-c表示将生成JVM字节码

162 String 的操作应该用StringBuilder通过append（）一步步拼接起来。如果走捷径，例如：append(a +＂sdf＂ + b)那么编译器会掉入另一个陷阱，从而为你另外创建一个StringBuilder对象处理括号内的字符串

163 StringBuilder不是常量，必须用new声明

164 输出PrintWriter （public class PrintWriterextends Writer）

向文本输出流打印对象的格式化表示形式。此类实现在 PrintStream 中的所有 print 方法。它不包含用于写入原始字节的方法，对于这些字节，程序应该使用未编码的字节流进行写入。

此类中的方法不会抛出 I/O 异常，尽管其某些构造方法可能抛出异常。客户端可能会查询调用 checkError() 是否出现错误。

165 compareTo(Object o)方法是java.lang.Comparable<T>接口中的方法，当需要对某个类的对象进行排序时，该类需要实现Comparable<T>接口的，必须重写public int compareTo(T o)方法

165.1 当且仅当equals(object)为true的时候，返回0

166 初始化和类的加载：java采用了加载，在java中，一切都是对象，每个类的编译代码都存在于他自己的独立的文件中。该文件只在需要使用程序代码时才会被加载。一般来说，可以说：“类的代码在初次使用时才加载。”这通常是指加载发生于创建类的第一个对象之时，但是当访问Static域或Static方法时，也会加载。所有Static按定义顺序初始化，当然定义为Static的东西只会初始化一次。

当Be.java开始运行时，首先是试图访问main()，于是加载器开始启动并找出Be类的编译代码(class文件中)。在对他加载的过程中，编译器注意到它有一个基类，于是他会先加载基类，无论你是否打算创建一个该基类的对象。

基类加载会一层一层递推，接下来基类中的static初始化立即被执行，然后是他导出类的，至此为止，必要的类都加载完毕了，对象可以被创建了。

首先，对象中的所有基本类型都会被设置为默认值，对象的引用会被设置为null，然后基类的构造器会被调用。（一般会自动调用，也可以用super指定对基类构造器的调用）。构造器完成以后，实例变量按其次序被初始化。

167 protected提供包访问权限

168 类访问权限，每个编译单元（文件）都只能有一个public类，这表示每个编译单元都有单一的公共接口，用public类来表现。如果在某个编译单元内有一个以上的public类，编译器会给出出错信息。

Public类的名称必须完全与该编译单元的文件名相匹配

虽然不是很常用，但是编译单元内完全不带public类也是可能的。在这种情况下，可以随意对文件命名。

169 内存溢出 out of memory，是指程序在申请内存时，没有足够的内存空间供其使用，出现out of memory；比如申请了一个integer,但给它存了long才能存下的数，那就是内存溢出。

内存泄露 memory leak，是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重，无论多少内存,迟早会被占光。

170 虚拟机类的加载:虚拟机把描述类的数据从Class文件加载到内存，并对数据进行校验、转换解析和初始化，最终形成可以被虚拟机直接使用的Java类型。

171 ClassLoader开始的5种情况：

1：遇到new、getstatic、putstatic、invokestatic指令时，如果类没有进行过初始化，则需要触发初始化。生成这4条指令最常见的Java场景是：使用new实例化对象、读取或设置一个类的静态字段（除final外，已在编译器将结果放置在常量池）、调用一个类的静态方法

2：使用reflect包的方法对类进行反射调用的时候，如果类没有进行初始化的话。

3：当初始化一个类发现他的父类没有初始化（接口初始化的时候不需要父类也初始化）

4：当虚拟机启动，用户需要指定一个要执行的主类（main）

172 如果父类和子类定义了一个相同的value，构造函数中this.value 会先指向子类的value，如果子类没有定义这个value会指向父类的value。

声明的时候，value跟声明的类的默认值走Sup s = new Sub();value 就是Sup的（没有this改变的话）

173 Human man = new Man()

Human称为变量的静态类型，Man称为变量的实际类型，区别是静态类型的变化仅仅在使用时发生，变量本身的静态类型不会被改变，并且最终的静态类型是在编译期可知的；而实际类型变化的结果在运行期才可确定

当重载时void say(Human()); void say(Man()) 调用say（man）时，使用哪个版本完全取决于传入参数的数量和数据类型。编译器在重载时是通过参数的静态类型而不是实际类型作为判断的，并且静态类型是编译期可知的。

重写的话是根据对象的实际类型来选择方法。

174 vector的许多方法是final的，所有如add，get，size等都是同步的，这导致了其效率很低，但是线程确是安全的。现在用的arraylist代替。在java语言中大部分线程安全类型都属于相对线程安全类型：它保证对这个对象单独的操作是线程安全的，我们在调用的时候不需要做额外的保护措施，但是对于一些特定顺序的连续调用需要在调用端使用额外的同步手段来保证调用的正确性(synchronized(…))例如Vector、hashtable、Collections的synchronizedCollection()方法包装等。

174.1 Stack继承vector

关于verctor：插入功能：

（1）public final synchronized void adddElement(Object obj)

将obj插入向量的尾部。obj可以是任何类型的对象。对同一个向量对象，亦可以在其中插入不同类的对象。但插入的应是对象而不是数值，所以插入数值时要注意将数组转换成相应的对象。

例如：要插入整数1时，不要直接调用v1.addElement(1),正确的方法为：

Vector v1 = new Vector();

Integer integer1 = new Integer(1);

v1.addElement(integer1);

174.2 list接口实现的3个类，vector arraylist linklist

1.ArrayList是最常用的List实现类，内部是通过数组实现的，它允许对元素进行快速随机访问。数组的缺点是每个元素之间不能有间隔，当数组大小不满足时需要增加存储能力，就要讲已经有数组的数据复制到新的存储空间中。当从ArrayList的中间位置插入或者删除元素时，需要对数组进行复制、移动、代价比较高。因此，它适合随机查找和遍历，不适合插入和删除。

2.Vector与ArrayList一样，也是通过数组实现的，不同的是它支持线程的同步，即某一时刻只有一个线程能够写Vector，避免多线程同时写而引起的不一致性，但实现同步需要很高的花费，因此，访问它比访问ArrayList慢。

3.LinkedList是用链表结构存储数据的，很适合数据的动态插入和删除，随机访问和遍历速度比较慢。另外，他还提供了List接口中没有定义的方法，专门用于操作表头和表尾元素，可以当作堆栈、队列和双向队列使用。

175 hashmap和hashtable（hashtable的线程安全不是通过final实现的）

也许最重要的不同是Hashtable的方法是同步的，而HashMap的方法不是。这就意味着，虽然你可以不用采取任何特殊的行为就可以在一个多线程的应4.2 用程序中用一个Hashtable，但你必须同样地为一个HashMap提供外同步。一个方便的方法就是利用Collections类的静态的synchronizedMap()方法，它创建一个线程安全的Map对象，并把它作为一个封装的对象来返回。这个对象的方法可以让你同步访问潜在的HashMap。这么做的结果就是当你不需要同步时，你不能切断Hashtable中的同步（比如在一个单线程的应用程序中），而且同步增加了很多处理费用。

第三点不同是，只有HashMap可以让你将空值作为一个表的条目搜索的key或value。HashMap中只有一条记录可以是一个空的key，但任意数量的条目可以是空的value。这就是说，如果在表中没有发现搜索键，或者如果发现了搜索键，但它是一个空的值，那么get()将返回null。如果有必要，用containKey()方法来区别这两种情况。

176 c/c++ 有类型提升的概念 也就是说char型实际上是int型 你还记得ASCII码吧 就是把数字映射到码表中找到对应字符 再输出 当你做把char型做运算的时候 会提升为int再做运算

177 int 最大为0x 7fffffff\最小为0x80000000

178 int long

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 16位系统/字节 | 32位系统/字节 | 64位系统/字节 |
| char | 1 | 1 | 1 |
| char\* | 2 | 4 | 8 |
| short | 2 | 2 | 2 |
| int | 2 | 4 | 4 |
| long | 4 | 4 | 8 |
| long long | 8 | 8 | 8 |

1、64位系统下，编译器为了向前兼容，将int类型自动编译为4个字节的了；

2、这些个玩意儿就是编译器来控制的，不同的编译器将int类型编译成不同长度的，所以，使用什么编译器才是重要的，比如：32位系统，tc下的int就是16位长度的，VC就是32位长度的。

3、long long是标准的8 bytes，不受编译器限制，所以，16位的依然也是8 bytes。（多谢博友0\_0i\_i 的指正）

179对于递归来说本质就是一个栈结构，意味着能用栈的问题肯定能用递归，但是，当用递归解决问题很大的话，会导致函数调用的层次很深，从而有可能导致函数调用栈溢出。显式的用栈基于循环实现的代码鲁棒性要好一些。

递归的方法代码比较整洁，但是性能不如基于循环的实现方法

180 位运算的对象是二进制数字，所以不是很直观，但也很简单，总共只有与、或、异或、左移<<、右移>>（在数学上与除以2是等价的，但是除法的效率比位运算要低很多，在实际编程中应该尽可能用位运算代替除法）。效率不一定高，会导致栈溢出

注意，对于右移来说，如果是正数的话是在右边补0，如果是负数的话是在右边补1。

181 c<<1 做32次不等于c<<32 C语言中 << 是逻辑移位，不是循环移位。

182 把一个整数减去1后再与原来的整数做位与运算，得到的结果相当于是把整数的二进制表示中的最右边一个1变为0.思路

位运算的时候一定要注意是否负数是合法的

183 由于精度原因，不能用等号判断两个小数是否相等。Double 用equal

Boolean equal(double a, double b){

If((a-b>-0.0000001)&&(a-b<0.0000001))

Return true;

Else

Return false;

}

184 三种错误的处理方法：

1. 函数用返回值来告知调用者是否出错（但是不能方便的使用计算结果）
2. 当发生错误时设置一个全局变量（用户可能会忘了检查全局变量）
3. 利用异常（抛出异常时对性能有负面影响）

185 sleep和wait区别： sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。**调用sleep不会释放对象锁**。wait是Object类的方法，**对此对象调用wait方法导致本线程放弃对象锁**，进入等待此对象的等待锁定池，只有针对此对象发出notify方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。

sleep是线程被调用时,占着cpu去睡觉,其他线程不能占用cpu,os认为该线程正在工作,不会让出系统资源,wait是进入等待池等待,让出系统资源,其他线程可以占用cpu,一般wait不会加时间限制,因为如果wait的线程运行资源不够,再出来也没用,要等待其他线程调用notifyall方法唤醒等待池中的所有线程,才会在进入就绪序列等待os分配系统资源,

sleep是静态方法,是谁掉的谁去睡觉,就算是在main线程里调用了线程b的sleep方法,实际上还是main去睡觉,想让线程b去睡觉要在b的代码中掉sleep

sleep(100L)是占用cpu,线程休眠100毫秒,其他进程不能再占用cpu资源,wait（100L）是进入等待池中等待,交出cpu等系统资源供其他进程使用,在这100毫秒中,该线程可以被其他线程notify,但不同的是其他在等待池中的线程不被notify不会出来,但这个线程在等待100毫秒后会自动进入就绪队列等待系统分配资源,换句话说,sleep（100）在100毫秒后肯定会运行,但wait在100毫秒后还有等待os调用分配资源,所以wait100的停止运行时间是不确定的,但至少是100毫秒.

总结来说：

在Java对象中，有两种池

琐池-----------------------synchronized

等待池---------------------wait(),notify(),notifyAll()

如果一个线程调用了某个对象的wait方法，那么该线程进入到该对象的等待池中(并且已经将锁释放)，

如果未来的某一时刻，另外一个线程调用了相同对象的notify方法或者notifyAll方法，

那么该等待池中的线程就会被唤起，然后进入到对象的锁池里面去获得该对象的锁，

如果获得锁成功后，那么该线程就会沿着wait方法之后的路径继续执行。注意是沿着wait方法之后

186 成员变量会被系统默认初始化，局部变量没这功能，所以必须自己初始化。

还要注意静态成员变量也没系统默认初始化，必须在初始化块或者定义时或者构造函数里进行手动初始化。

187 静态变量，成员变量与局部变量:

1.成员变量

属性：对应类中的成员变量

行为：对应类中的成员函数

定义类其实就是在定义类中的成员（成员变量和成员函数）

2. 静态变量

由static修饰的变量称为静态变量，其实质上就是一个全局变量。如果某个内容是被所有对象所共享，那么该内容就应该用静态修饰；没有被静态修饰的内容，其实是属于对象的特殊描述。

3. 成员变量和局部变量的区别

成员变量：

①成员变量定义在类中，在整个类中都可以被访问。

②成员变量随着对象的建立而建立，随着对象的消失而消失，存在于对象所在的堆内存中。

③成员变量有默认初始化值。

局部变量：

①局部变量只定义在局部范围内，如：函数内，语句内等，只在所属的区域有效。

②局部变量存在于栈内存中，作用的范围结束，变量空间会自动释放。

③局部变量没有默认初始化值

4.在使用变量时需要遵循的原则为：就近原则

首先在局部范围找，有就使用；接着在成员位置找。

5. 成员变量和静态变量的区别

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，随着对象被回收而释放。

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。

6.  成员变量、局部变量、静态变量的区别

|  |  |  |  |
| --- | --- | --- | --- |
|  | **成员变量** | **局部变量** | **静态变量** |
| 定义位置 | 在类中,方法外 | 方法中,或者方法的形式参数 | 在类中,方法外 |
| 初始化值 | 有默认初始化值 | 无,先定义,赋值后才能使用 | 有默认初始化值 |
| 调用方式 | 对象调用 | --- | 对象调用，类名调用 |
| 存储位置 | 堆中 | 栈中 | 方法区 |
| 生命周期 | 与对象共存亡 | 与方法共存亡 | 与类共存亡 |
| 别名 | 实例变量 | --- | 类变量 |

7．小结：

尽管三者的本质都是变量，可是使用时却有相当大的区别，稍不留神就可能陷入陷阱。且先记住：在一个类中，如果一个变量能够用来描述一个类的属性，那就定义为成员变量，否则，它就应该定义为局部变量。而如果一个变量在全局中都能使用（某个内容是被所有对象所共享），那么我们就可以把这个变量用static来修饰，即为静态变量。（还要注意它与静态方法的紧密联系，这里且不细述了）

188 ClassLoader

C++和Java区别: Java并不是一个可执行文件，而是由许多独立的类文件组成的，每一个文件对应一个Java类。此外这些类文件并不是全部装入内存，而是根据程序需要逐渐载入。

ClassLoader包括bootstrap classloader，ExtClassLoader，AppClassLoader

ClassLoader加载流程：当运行一个程序的时候，JVM启动，运行bootstrap ClassLoader，该ClassLoader加载Java核心API（ExtClassloader和AppClassLoader也在此时被加载），然后调用ExtClassloader加载扩展API，最后AppClassLoader加载ClassPath目录下定义的Class，这就是一个程序最基本的加载流程。

188.1 加载类的两个方法：classloader中的loadclass（第二个参数是false，加载的时候不对该类进行解释，因此不会初始化该类）；class中的forname()（第二个参数是true，会对该类进行解释和初始化）

188.2 加载类的三个步骤：装载，链接，初始化

装载就是找到相应的class文件，读入JVM；初始化就是class文件初始化；链接分为三步：第一步是验证class是否符合规格；第二步是准备，就是为类变量分配内存的同时设置默认初始值；第三步就是解释，这步是可选的，根据上面loadclass方法的第二个参数来判定是否需要解释，这里的解释是指根据类中的符合引用查找相应的实体，再把符号引用替换成一个直接引用的过程。

189 JVM的编码用Unicode characters

Java开发者必须牢记，在java中（在JVM，内存中，代码里声明的每一个char、string类型的变量中）字符只以一种形式存在，那就是Unicode（16位，也就是2两个字节代表一个字符）

189.1 所有的编码转换只发生在边界的地方，JVM内部和OS的文件系统，在JVM和OS的交界处，也就是各种输入/输出流（或者Reader，Writer类）起作用的地方

如果面向字节，那么这类工作要保证系统中的文件二进制内容和读入JVM内部的二进制内容一致，不能变换任何0，1的顺序，这种输入/输出方式很适合读入视频文件或者音频文件，或者任何不需要走变换的文件内容

面向字符的I/O是指希望系统中的文件的字符和读入内存的字符要一致。

从这个意义上可以看出，面向字符（Writer/Reader类）实际上隐式的做了编码转换，傻瓜式的，不能指定编码。Reader只会用GBK，不管是不是

当需要编码转换时，只能用字节（InputStreamReader/OutputStreamWriter）。