目录

[一、数据类型和运算符 2](#_Toc500591424)

[0，文档注释？ 2](#_Toc500591425)

[1，Java是强类型语言？ 2](#_Toc500591426)

[2，Java数据类型？ 3](#_Toc500591427)

[3，类型转换？ 3](#_Toc500591428)

[4，计算机表示浮点数的原理？ 3](#_Toc500591429)

[5，怎样比较浮点数的大小？ 3](#_Toc500591430)

[6,无穷大和非数？ 3](#_Toc500591431)

[7.零碎知识点 3](#_Toc500591432)

[8,运算符的注意点 4](#_Toc500591433)

[9，基本数据类型的包装类 5](#_Toc500591434)

[10，== 和 equals 的区别 6](#_Toc500591435)

[12，逻辑运算符 8](#_Toc500591436)

[二、流程控制和数组 8](#_Toc500591437)

[1，分支结构if和switch 8](#_Toc500591438)

[2，break : 结束当前循环 或 当前以及上层循环 9](#_Toc500591439)

[3，continue忽略本次循环剩下的语句 或 结束本循环接着执行上层循环的++i; 9](#_Toc500591440)

[4，区分for和while的continue 9](#_Toc500591441)

[5，定义数组 10](#_Toc500591442)

[6，数组初始化的2种方法 （1）type[] arrname = {1,2,3}; （2）type[] arrname = new arrname[3];         系统初始化成0或null或false         for(int i = 0;i < arrname.length;++i) {         arrname[i] = value; 10](#_Toc500591443)

[7，数组的内存机制 10](#_Toc500591444)

[8，数组常见异常 10](#_Toc500591445)

[9，二维数组 10](#_Toc500591446)

[10，foreach遍历数组和容器 11](#_Toc500591447)

[11，数组常用方法 11](#_Toc500591448)

[三、面向对象<上> 11](#_Toc500591449)

[0，基本概念 11](#_Toc500591450)

[1，类的成员 11](#_Toc500591451)

[2，构造器 11](#_Toc500591452)

[3，关键字this 12](#_Toc500591453)

[4，关键字static 12](#_Toc500591454)

[5，局部变量和内存 12](#_Toc500591455)

[6，使用变量的规则 13](#_Toc500591456)

[7，方法重载 13](#_Toc500591457)

[8，方法的参数传递机制 13](#_Toc500591458)

[9，形参个数可变的方法 13](#_Toc500591459)

[10，封装的概念 13](#_Toc500591460)

[11，详解访问控制符 14](#_Toc500591461)

[12，包机制 14](#_Toc500591462)

[13，决定类能否被继承或创建对象的因素 14](#_Toc500591463)

[四、面向对象<下> 15](#_Toc500591464)

[1，继承 15](#_Toc500591465)

[2，方法重写 15](#_Toc500591466)

[3，多态 15](#_Toc500591467)

[4，instanceof 15](#_Toc500591468)

[5，toString 15](#_Toc500591469)

[6，final 16](#_Toc500591470)

[7，抽象类 16](#_Toc500591471)

[8，接口 16](#_Toc500591472)

[9，抽象类和接口的区别 17](#_Toc500591473)

[10，内部类 18](#_Toc500591474)

[11，lambda表达式 18](#_Toc500591475)

[12，单例类，不可变类，枚举类 18](#_Toc500591476)

[13，JVM常量池 19](#_Toc500591477)

[14，clone 20](#_Toc500591478)

[15，垃圾回收 20](#_Toc500591479)

[五、Java基础类库 20](#_Toc500591480)

[六、异常 20](#_Toc500591481)

[七、Java集合 20](#_Toc500591482)

[八、泛型 20](#_Toc500591483)

[七、流 20](#_Toc500591484)

[八、多线程 20](#_Toc500591485)

[九、网络编程 20](#_Toc500591486)

一、数据类型和运算符

0，文档注释？

使用JDK提供的javadoc工具，把类，接口，成员变量，构造器，方法，内部类前的注释提取成API文档。

javadoc 选项 源文件（支持正则表达式）

javadoc 选项 包名（把包内所有源文件放到相应目录下，添加描述包信息的package.html文件）

注意：

（1）只能是类，接口，成员变量，构造器，方法，内部类前的注释，会忽略其他位置的注释。

（2）注释格式/\*\*

\*......

\*......

\*......

\*/

1，Java是强类型语言？

变量，先定义，初始化，后使用。必须赋予变量相匹配的类型的值。

2，Java数据类型？

基本类型：整型（byte short int long）浮点型（float double）字符型（char）布尔类型（boolean）

byte 1 short 2 int 4 long 8 float 4 double 8 char 2(相当于C语言的无符号2字节，short也是2字节，但是有符号)。

引用类型：包装类，其它类。

3，类型转换？

基本数据类型之间的类型转换，不包括布尔类型。

强类型转换，自动类型转换。把一个表数范围小的数值或变量直接赋给另一个表数范围大的变量时，叫自动类型转换，反之，强制类型转换，需加强制转换符。

byte short char全部转换成int，int->long->float->double

引用变量的类型转换，只能发生在具有继承关系的类中，向下转型的变量，必须先前已经发生向上转型，否则会ClassCastException异常。

4，计算机表示浮点数的原理？

例如单精度：32位，1位符号位，8位指数，23位有效数字。

十进制的9.625，转换成二进制是1001.101 = 1×2^3 +0×2^2+0×2^1+1 ×2^0+1×2^-1+0×2^-2+1×2^-3 = 1.001101×2^3.符号位是0，尾数（有效数字）是0011\_0100\_0000\_0000\_0000\_000，基数是2，指数是3+中间数。第一位肯定是1，所以省略。指数是负数怎么办？8位指数范围是0-255,实际指数表示范围是-127-127,指数是3时，填3+127，指数是-2时，填-3+127.

5，怎样比较浮点数的大小？

double a == double b,错！因为计算机并不能准确的表示每一个小数，比如十进制的0.1，转换成二进制0.000110011001100110011001100110011001100...尾数只有23位，不够用只好舍去近似表示。数都不是准确表示的，计算结果怎么又会是准确的呢？

System.out.print("34.6-34.0=" + (34.6f-34.0f));34.6-34.0=0.5999985。正确方式通过函数表示。

public final double jingdu = 0.001

static boolean equals(double\* a,double\*b) {

return a-b<jingdu || b-a>jingdu?true:false;

}

6,无穷大和非数？

指定指数域全0或全1为无穷大或非数。

正无穷大：double d = Double.POSITIVE\_INFINITY flloat f = Float.POSITIVE\_INFINITY 任何正无穷大都相等

负无穷大：double d = Double.NEGA\_INFINITY flloat f = Float.NEGA\_INFINITY 任何负无穷大都相等

非数： Float.NaN Double.NaN 所有的NaN都不相等

浮点数除以0无穷大 整数除以0产生异常ArithmeticException

7.零碎知识点

重点：在算术表达式中，byte,short,char全部变成int，然后所有变量转成它们中级别最高的类型进行计算。

整数默认int类型long l = 9999999999 编译出错，默认int却超出int范围，long l = 999999999L

浮点数默认double类型，8888.8f表示float类型

只有浮点型数据能表示成科学计数法，5200是整型，52E2是浮点型

java编码方式是16位的Unicode字符集，Unicode的前256个'/u0000'到'/u00FF'和ASCII完全重合

'\n' 换行 '\"'双引号 '\\'反斜杠 '\r' 回车

char ch1 = 'A'; char ch2 = '疯'; char ch3 = '\r'; char ch4 = '\u006F';

System.out.println('a'+1+"string");//98string

System.out.println(1+"string"+'a');//1stringa

1 boolean a = true;

2 double d = 8.2732;

3 System.out.println("haha"+d+a);

4 //输出 haha8.2732true

int i1 = 23；

int i2 = 456;

double d1 = (i1+i2)\*1.2;//系统将转换为double型运算

float f1 = (float)((i1+i2)\*1.2);//需要加强制转换符

byte b1 = 67;

byte b2 = 89;

byte b3 = (byte)(b1+b2);//系统将转换为int型运算，需

//要强制转换符

System.out.println(b3); //溢出,系统自动把超出的字节砍掉

double d2 = 1e200;

float f2 = (float)d2;//会产生溢出,显示无穷大标志符

System.out.println(f2);

float f3 = 1.23f;//必须加f

long l1 = 123;

long l2 = 30000000000L;//必须加L

float f = l1+l2+f3;//系统将转换为float型计算

long l = (long)f;//强制转换会舍去小数部分（不是四舍五入）

8,&与&&

&与&&的区别：&总是先执行完两个操作数后才逻辑，&&先计算左边的操作数，如果是false，不再计算右边的，返回false

I与II的区别参考上。

9，基本数据类型的包装类

|  |  |
| --- | --- |
| 基本数据类型 | 包装类 |
| char | Character |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| boolean | Bollean |

public class TestJava **{**

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

/\*

比较两个整数大小

用 == 比较

基本数据类型 == 基本数据类型 正确比较

引用类型 == 引用类型 可以认为一直是false

基本数据类型 == 引用类型 正确比较

用Integer.compare(a,b)比较

引用类型还是数据类型无所谓，相等就返回0，不相等，就返回非0值。

\*/

Integer inta **=** 234**;**

Integer intb **=** 234**;**

int intc **=** 234**;**

System**.**out**.**println**(**inta **==** intb**);**//false

System**.**out**.**println**(**inta **==** intc**);**//true

System**.**out**.**println**(**Integer**.**compare**(**inta**,**intc**));**//0

System**.**out**.**println**(**Boolean**.**compare**(true,true));**//0

System**.**out**.**println**(**Boolean**.**compare**(true,false));**//1

System**.**out**.**println**(**Boolean**.**compare**(false,true));**//-1

/\*基本数据类型和包装类可以相互直接赋值或比较\*/

char ch1 **=** '$'**;**

Character ch2 **=** '$'**;**

Character ch3 **=** **new** Character**(**'$'**);**

System**.**out**.**println**(**ch1 **==** ch2 **&&** ch1 **==** ch3**);**//true

/\*字符串转成基本数据类型\*/

String str **=** "178"**;**

int a **=** Integer**.**parseInt**(**str**);**

Integer b **=** Integer**.**parseInt**(**str**);**

Integer c **=** **new** Integer**(**str**);**

System**.**out**.**println**(**a**);**

System**.**out**.**println**(**b**);**

System**.**out**.**println**(**c**);**

/\*基本数据类型转成字符串\*/

String strstr **=** **null;**

strstr **=** **new** String**(**String**.**valueOf**(**8E2**));**

System**.**out**.**println**(**strstr**);**//800.0

Float f **=** 0.618F**;**

System**.**out**.**println**(**String**.**valueOf**(**f**));**//0.618

strstr **=** 3.14 **+** ""**;**

System**.**out**.**println**(**strstr**);**//3.14

**}**

**}**

10，== 和 equals 的区别

（1） ==

基本类型和  基本类型或基本类型的包装类，语法正确，且  65 == 65.0 == 'A'  is  true

存在继承关系的两个引用变量   指向同一个实体，返回true。   操作数是不存在继承关系的引用变量，编译错误。！= 注意点参照 ==

（2）equals

默认的equals，两个引用变量指向同一个实体，才返回true，否则返回false。 重写的equals，两个引用变量指向的实体，实体的成员变量都相等就返回true。

equals的标准写法：

class Person {

private String name;

public Person(String name) {

this.name = name;

}

public boolean equals(Object obj) {

/\*如果两个对象是同一个对象\*/

if(obj == this)

return true;

/\*只有obj和自己属于同一种类时\*/

if(obj!=null && obj.getClass()==Person.Class) {

Person objPerson = (Person)obj;

/\*成员变量相等\*/

if(obj.name == this.name)

return true;

}

return false;

}

}

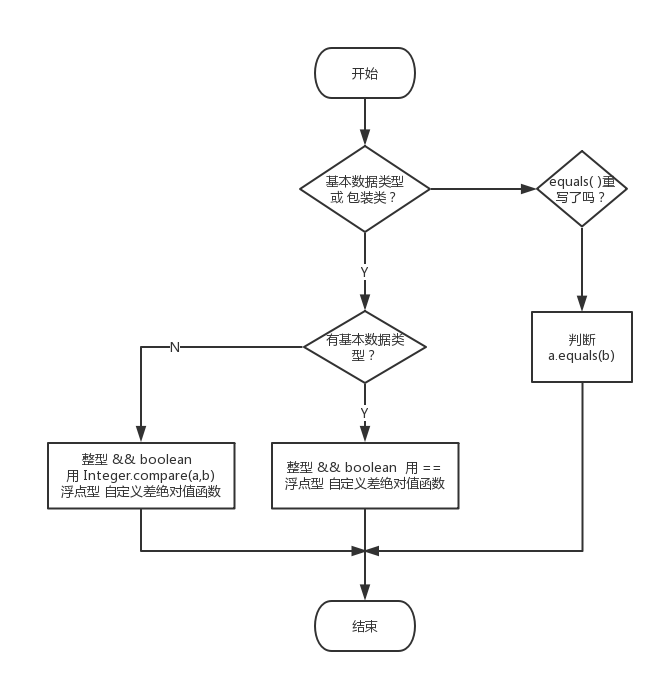
11，判断两个变量相等的标准流程

浮点型用函数

public static boolean compare(double a,double b) {

return if(a-b < 0.000001 || b-a < 0.000001) ? true : false;

}



12，逻辑运算符

<  <=   >   >=

操作数只能是基本类型或包装类，其他引用变量编译出错。

不像判等，它们可以直接使用。

二、流程控制和数组

1，分支结构if和switch

if（boolean） switch（byte   short   int   char    enum    字符串）

string str = "秋天";

switch(str) {//打印出3

case "春天" : System.out.println(1);break;

case "夏天" : System.out.println(2);break;

case "秋天" : System.out.println(3);break;

case "冬天" : System.out.println(4);break;

default : break;

}

2，break : 结束当前循环 或 当前以及上层循环

System.out.println("循环测试开始\n\n");

outer:

for(int i = 0;i < 6; ++i) {

for(int j = 0;j < 4; ++j) {

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

if(j == 2)

break outer;

}

}

打印结果：

循环测试开始

i = 0j = 0

i = 0j = 1

i = 0j = 2

3，continue忽略本次循环剩下的语句 或 结束本循环接着执行上层循环的++i;

System.out.println("循环测试开始\n\n");

outer:

for(int i = 0;i < 5; ++i) {

for(int j = 0;j < 3; ++j) {

if(j == 1)

continue outer;

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

}

}

循环测试开始

i = 0 j = 0

i = 1 j = 0

i = 2 j = 0

i = 3 j = 0

i = 4 j = 0

4，区分for和while的continue

**for的循环增量会执行，但是while的循环增量不会执行！！！**

System.out.println("测试for开始\n\n");

for(int i = 0;i < 4; ++i) {

if(i == 2)

continue;

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

}

System.out.println("测试while开始\n\n");

int k = 0;

while(k < 4) {

if(k == 2)

continue;

System.out.println("k = " + k);

++k;//continue后，++k永远没机会执行了，循环一直执行，没死！只是无法打印k了，就像阻塞了

}

测试for开始

i = 0

i = 1

i = 3

测试while开始

k = 0

k = 1

//程序卡死中...

5，定义数组

int[ ] 是一种数据类型。

定义数组：type[ ] arrayname 定义数组就是定义了一个指向数组的引用，但是还未指向任何内存，定义数组时不能指定数组的长度。

6，数组初始化的2种方法

（1）type[] arrname = {1,2,3};  
（2）type[] arrname = new arrname[3];  
        系统初始化成0或null或false  
        for(int i = 0;i < arrname.length;++i) {  
        arrname[i] = value;

思考引用类型数组的定义，初始化过程。

7，数组的内存机制

数组引用在栈内存中，数组在堆内存中，栈中引用null，堆中数组就变成“野孩子”，垃圾回收机制会在适宜时候，回收该数组，若还有其他同类型的引用指向该堆内存数组，则不会被回收。

8，数组常见异常

数组索引越界异常：java.lang.ArrayIndexOutofBoundsException:N

9，二维数组

//必须指定行数。

/\*a[i][j]表示数组a[i],每个a[i]存放a[j]数组的引用\*/

int [][]a = {{1,2},{3,4,5,6},{7,8,9}};//3行，每一行的列数不同

//共有9个存储空间，而不是12个。

for(int i=0 ; i <a.length ; i++) {

for(int j=0 ; j<a[i].length ; j++) {

System.out.println("a[" + i + "][" + j + "]=" + a[i][j]) ;

}

10，foreach遍历数组和容器

String**[]** books **=** **{**"致青春"**,**"人与自然"**,**"红楼梦"**};**

**for(**String book **:** books**)** **{**

System**.**out**.**print**(**book**);**

**}**

book是临时变量，不是数组元素本身。

11，数组常用方法

Java提供的Arrays类里包含的static方法，import java.util.Arrays;

int binarySearch(type[ ] a,type key) 二分查找

type[ ] copyOf(type[] original,int length),复制数组，length大于数组长度时补null，0，false

Boolean equals(type[ ] a,type[ ] b) 两数组长度相同，元素相同，返回true

void sort(type[ ] a) 排序

三、面向对象<上>

0，基本概念

面向对象的三大特征：封装，继承，多态。

类的各成员可以相互调用，但是类方法不能调用实例成员

类成员用类初始化块初始化，实例成员用实例初始化块或构造器初始化。

1，类的成员

类成员变量 类初始化块 类方法 实例变量 构造器 实例初始化块 实例方法 内部类

成员前加饰static，就属于类成员。static修饰成员变量和方法。

类初始化块只在加载类时，执行一次，从此再也没机会执行了。

构造器不用static修饰，因为构造器是用来初始化实例的。

类成员变量，加载类时创建一份，为所有实体共享，不用创建实体就能使用。通过 类名.变量 使用。

在面向对象的语言中，类是一等公民，方法是二等公民。永远不要把方法当成独立存在的实体。方法必须在类中定义，属于类或实例，通过类名或对象调用。

2，构造器

构造器是一个类创建对象的根本途径。如果程序员没有为一个类编写构造器，系统会为该类提供一个默认构造器，把实例变量初始化成0 false null. 一旦程序员编写了构造器，系统就不再提供默认构造器。

构造器用于对类实例进行初始化的操作，构造器支持重载，如果多个重载的构造器包含了相同的初始化代码，可以把相同代码提取到初始化块中，初始化块先于构造方法执行。

构造器修饰符:public protected private都可以，无返回值，方法名和类名相同。

可以为类编写多个构造方法,增加实体化时,初始化的灵活度。

有多个构造方法时,会根据参数列表,选择适宜的构造方法。

类new出对象时，会自动调用构造方法完成初始化。

public class Apple {

private String color;

private String name;

private int weight;

Apple(String color,String name) {

this.color = color;

this.name = name;

}

Apple(String color,String name,int weight) {

this(color,name);

this.weight = weight;

}

}

3，关键字this

1. 构造器中使用，用于区别局部变量和成员变量。
2. 类成员相互调用时使用this，可省略。
3. 作为对象的默认引用，return。
4. static修饰的方法不能含有this

4，关键字static

static 仅修饰成员变量和方法和初始化类成员变量。

静态变量和静态方法能够通过类名来访问.

[static是为了不实体对象,就能使用变量和方法].

静态成员变量

编写好类后就已经被初始化了,等于直接赋的值,或默认的0，或被静态初始化块初始化。static变量随类而生,未创建对象就已存在,实体化对象时,不会在堆中为静态成员变量分配空间,但是该对象拥有此静态成员变量。static修饰的成员变量,保存在静态区,它只有一份,为所有对象共享,任何对象都可以修改它。

静态方法:

每个对象都拥有,也可独立于实体对象使用,不能调用静态变量和静态方法以及this。实体方法中能调用静态方法和静态变量。

以下情形可以使用静态方法：

一个方法不需要访问对象状态，其所需参数都是通过显式参数提供（例如 Math.pow()）.一个方法只需要访问类的静态变量.

不要通过实例引用调用静态成员。静态方法不能被实体方法重写；

5，局部变量和内存

局部变量是基本数据类型保存在内存中，局部变量是引用类型，引用在内存中。

6，使用变量的规则

内存开销和内聚性（生存时间和作用域尽可能小）。

所有对象相同的固有信息，如人的眼睛数目永远是2，就定义成static。

保存类运行状态的信息的变量，定义成成员变量。

某个信息需要被类中的多个方法共享，定义成成员变量。

变量的作用域应尽可能小，这样在内存中逗留的时间就越短，代码性能就越好。

比如多使用for( int i = 0;i < value; ++i)   而少用 int i; for(i = 0; i < value; ++i)

7，方法重载

为什么要引入方法重载?

答:可以让一个程序段尽量减少方法的种类,更人性化的API.

比如定义一个比较两个变量大小的方法max,变量类型可能是double int char任意一种.

如果没有重载概念,要写3个方法名不同的方法实现,调用时,还要记得函数名与对应参数类型.若有重载,只需记住max(参数列表),系统会根据参数列表选择对应的方法.

还有System.out.print（ ）能输出基本数据类型和字符串。

方法重载的要求就是两同一不同：同一类中方法名相同，参数列表不同.

至于方法的其他部分，如方法返回值类型、修饰符等，与方法重载没有任何关系.

参数列表不同指->个数,类型,顺序至少有一项不同.

调用时,根据方法参数列表选择对应的方法.

同一类的所有方法之间必须满足条件:方法名不同 || 方法名相同&形参列表不同。

8，方法的参数传递机制

值传递：对于swap方法，形参是基本数据类型，不改变实参。形参是引用变量，不会改变引用变量，但会改变堆中的实体。

基本类型 a,b; 引用类型 c,d;

void swap(基本类型a,基本类型b) {

//执行后，成员变量a,b值不变

}

void swap(引用类型c,引用类型d) {

//执行后，成员变量c,d指向的实体成员变量交换了，但是c,d值扔不变。

}

}

9，形参个数可变的方法

void test(int a,String... books)

或

void test(int a,String books[])

可变参数必须位于列表最后。

10，封装的概念

封装是面向对象的三大特征之一，它将对象的成员变量隐藏起来，不允许外部程序直接访问，而是通过对象提供的加方法来访问，可以在访问方法中加入逻辑控制块，限制不合理访问，提高程序的安全性。同时把该暴露的暴露，该隐藏的隐藏，提高代码的内聚性。

11，详解访问控制符

（1）private

private主要修饰成员变量，也可修饰方法。private修饰的成员仅在类内调用。

private修饰的成员变量，通过getter和setter方法访问。可以被继承，但是无法在子类触及，只能通过getter和setter访问。

private修饰的方法，仅在类内被调用，类外无法触及，也无法继承，权当死了。所以除了辅助实现其他方法的工具方法外，方法一般不用private修饰！

（2）default

可修饰外部类，成员变量和方法。包访问权限，同一包的类随便调用。

（3）protected

修饰成员变量和方法。同一包的类随便调用，包外的子类也能（调用），但是一般不是调用，而是重写之。如果某个方法希望被子类重写，又不希望其他类自由访问，就用protected修饰。

（4）public

是否处于同一包中，是否具有父子继承关系，都无所谓。

12，包机制

解决类同名的问题。

Java允许把实现一定功能的类放在同一个包里，从而组成逻辑上的类库单元。

packgae packagename;

class Test1 {

}

则Test1属于package这个包内。

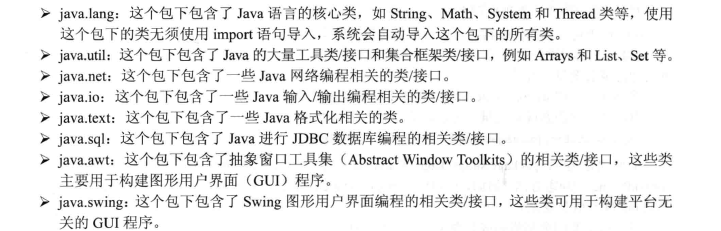
定义每个类时，理论上应该先写package packagename;

包名命名规则：公司域名倒写，如www.sust.edu.cn    则包名是 cn.edu.sust.www；

意味着，硬盘的cn/edu/sust/www/目录下有个Test1.class

编译命令：javac -d  Test1.java   加-d会把生成的class文件自动置于包名的目录下。

使用类时，应该是import cn.edu.sust.www.\*；或，cn.edu.sust.www.Test1



13，决定类能否被继承或创建对象的因素

外部类，public任何位置都可创建对象，default，同一个包的类内可创建。

构造器，public任何位置，default，同一包内的类内，protected，主要被子类调用，private，阻止其他类创建该类的实例。

final 修饰的类不能被继承。

四、面向对象<下>

1，继承

extends，单继承。构造方法和private修饰的方法不能被继承，父类的构造器内必须调用父类构造器super(参数列表)且必须位于第一行。

2，方法重写

两同两小一大原则：方法名相同，形参列表相同，子类方法返回的类型比父类的更小或相等，子类方法抛出的异常比父类更小或相等，子类方法的访问权限应比父类的更大或相等。

重写的方法要么都是类方法，要么都是实例方法。

在子类中，通过super调用父类被重写的方法。

父类被private修饰的方法不被继承，子类中定义同名方法不是重写，而是创建了一个新方法。

重写不会删除父类中的方法，而是对子类的实例隐藏，暂时不使用, 被覆盖的方法只能在子类中只能通过super调用.

3，多态

父类引用 指向了  子类实体。BaseClass bc = new SubClass( );

（1）父类引用.方法

方法没被子类覆盖，调用的是父类方法

方法被子类覆盖，调用的是子类重写的方法

方法不能是子类新增的方法

（2）父类引用.变量，变量不具有多态性，调用的一定是父类的变量。如果子类新增的变量和继承的变量同名，调用的仍旧是父类变量。

需要使用子类新增方法和变量，先把引用变量向下转型，获得转型后的引用，

（1）转型引用.方法：调用的是子类新增方法和子类重写的方法和从父类那里继承来但没重写的方法。

（2）转型引用.变量：使用的是子类的新增变量和从父类那里继承来的且不和子类新增变量重名的变量，如果调用从父类继承来的重名变量，使用super。

4，instanceof

使用语法：引用变量  instanceof  类名

结果：（1）指向子类创建的实体，返回 true （2）指向本类创建的实体，返回 true

（3）指向父类返回false （4）不具有继承关系，编译时异常

5，toString

String toString（）

**打印一个引用类型变量**，会显示什么？  答案不是 打印出实体的所有成员变量。

显示什么，取决于每个类都有的**toString（）**方法。

1 Object c = new Object();

2 System.out.println(c);

3 System.out.println(c.toString());

4 2,3行代码输出结果相同

toString（）是Object 类的 成员方法，所有的类都继承了Object类，所以所有的类都有。

Object类的toString（）是 返回 类名 + @ + hashCode

6，final

final 修饰的变量必须由程序员显示初始化！  
final修饰类变量有两种初始化方法：（1）定义时直接初始化 static final int i = 100; （2）在类初始化块中赋值 static final int i; static {i = 100 }  
final修饰实体变量三种初始化方法：（1）定义时直接初始化final int j = 23;(2)在实体初始化块中初始化 {j = 23} （3）构造方法初始化

final修饰形参，形参被值传入时初始化，在方法内不能再变。final修饰局部变量，定义时可以不直接初始化，但是后面一旦初始化，便不能再修改。  
final修饰引用变量，引用变量的指向谁不能变，但是不影响实体的成员变量改变。

final修饰的方法不能被重写。  
final修饰的类不能被继承。

7，抽象类

抽象类的好处  
多态时，抽象类的引用可以更多的直接调用继承类的方法，不需要向下转型，抽象类能够规定子类必须干什么，有种起到模板的作用，同时，抽象类能更好的复用代码。  
  
抽象类规则  
抽象类用abstract修饰，抽象类的抽象方法用abstract修饰。  
抽象类不能创建实例，被继承且所有的抽象方法被重写才能创建实例。  
抽象类的构造器只是用来被继承类的构造器调用。  
abstract和final，private，static不能同时使用。  
抽象方法声明格式：  
public abstract void f();

8，接口

接口的规则

一个接口能继承多个接口，不能继承类  
成员变量都是 public static final且必须声明时直接赋值  
所有普通方法都是public abstract  
类方法都是public static 要实现  
默认方法都是public default 要实现  
接口没有构造器和初始化块

接口的用途  
接口是一种规范，规定子类必须干什么，而不用知道怎么干。  
接口能使java有多继承的优点，又能避免多继承的缺陷  
接口引用指向子类实例，支持向下转型。  
调用接口的常变量，协议字符

继承接口的类重写接口的抽象方法时,权限只能是public,因为子类重写父类时,只能用相同或者更大的权限。

一个类可以多继承很多接口,但是只能继承一个类.

class extends 类 implements 接口1,接口2... {

}

9，抽象类和接口的区别

接口只能有常变量，但抽象类的成员变量和普通类一样没限制。

接口不能有普通方法，抽象类可以有。

接口没有构造器和初始化块，抽象类可以有

一个类可以继承多个接口，但只能继承一个抽象类

接口是一种规范，它规定了实现者必须提供哪些服务，一个程序使用接口，它是程序多个模块的耦合，多个程序使用接口，它是通信标准，当系统修改了接口，会导致系统的很多类重写，影响很大。

抽象类就像一个中间产品，需要进一步完善而已，修改抽象类对系统的影响较小。

10，内部类

内部类比外部类多了3个修饰符：private，protected，static

非静态内部类不能拥有静态成员：静态成员，静态方法，静态初始化块。

非静态内部类可以直接访问外部类的所有成员，包括private；外部类不能直接访问非静态内部类，除非外部类显示创建一个非静态内部类实例，然后调用非静态内部类成员，注意：外部类的静态方法内不能出现非静态内部类的任何相关代码。

静态内部类只能调用外部类的静态成员，即使内部类的实例方法也不能调用外部类的非静态成员。外部类可以使用静态内部类的类名作为调用者来访问静态内部类的静态成员，也可以使用静态内部类对象来访问静态内部类的实例成员。

在外部类以外使用内部类，定义方法

内部类不要使用private修饰。

静态内部类：Outclass.Inclass in = new Outclass( ).new Inclass( );

非静态内部类：Outclass.Inclass in = new Outclass( ).Inclass( );

匿名内部类用于创建仅使用一次的类。

11，lambda表达式

12，单例类，不可变类

单例类：某个类，仅当系统不存在自身类的实体才创建实体。  即，系统中要么无实体，要么仅有一个实体。

class Singleton

{

//使用一个变量来缓存曾经创建的实例

private static Singleton instance;

//将构造器使用private修饰，隐藏该构造器

private Singleton(){}

//提供一个静态方法，用于返回Singleton实例

//该方法可以加入自定义的控制，保证只产生一个Singleton对象

public static Singleton getInstance()

{

//如果instance为null，表明还不曾创建Singleton对象

//如果instance不为null，则表明已经创建了Singleton对象，将不会执行该方法

if (instance == null)

{

//创建一个Singleton对象，并将其缓存起来

instance = new Singleton();

}

return instance;

}

}

public class TestSingleton

{

public static void main(String[] args)

{

//创建Singleton对象不能通过构造器，只能通过getInstance方法

Singleton s1 = Singleton.getInstance();

Singleton s2 = Singleton.getInstance();

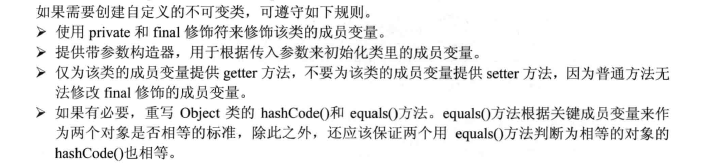
//将输出true

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

}

}

不可变类：不可变类创建的实体，创建后，实体成员变量不再变化。如Java提供的8个包装类和String。



枚举类：

13，JVM常量池

保存类，接口，方法中的常量，还包括字符串常量

public class TestJava {

public static void main(String[] args) {

String str1 = "HelloJava";

String str2 = "Hello";

String str3 = "Java";

String str4 = "HelloJava";

String str5 = str2 + str3;

String str6 = new String("HelloJava");

/\* 测试 == \*/

System.out.println(str1 == str4);//true,字符串在常量池，引用指向同一份

System.out.println(str1 == str5);//false，一个实体在线程池，一个在堆中，引用变量不相等

System.out.println(str5 == str6);//false//两个实体都在堆中，但在不同位置

/\* 测试 equals() \*/

System.out.println(str1.equals(str4));//true

System.out.println(str1.equals(str5));//true

System.out.println(str1.equals(str6));//true

System.out.println(str5.equals(str6));//true

/\*String重写了equals(),只要字符序列相同，就返回true \*/

}

}

14，clone

15，垃圾回收

垃圾回收的是堆内存中的对象，不能回收物理资源（如数据库连接，IO流等）

彻底失去引用的对象才可能被回收

垃圾回收机制回收对象前，会调用它的finalize（）方法，如果finalize（）方法把对象从可恢复状态变成可达状态，则不回收，否则就是不可达状态，回收之

程序不能决定何时垃圾回收，一般内存紧张时，进行垃圾回收

可以调用静态方法System.gc(),Runtime.getRuntime.gc()进行强制垃圾回收，强制垃圾回收并不是立即或一定进行垃圾回收，只是会有点效果。

16、finalize（）

finalize（）方法是Object类的实例方法，在该方法中可清理对象占用的资源，任何类都可以重写。

永远不要主动调用finalize（）方法，应该交给垃圾回收机制调用

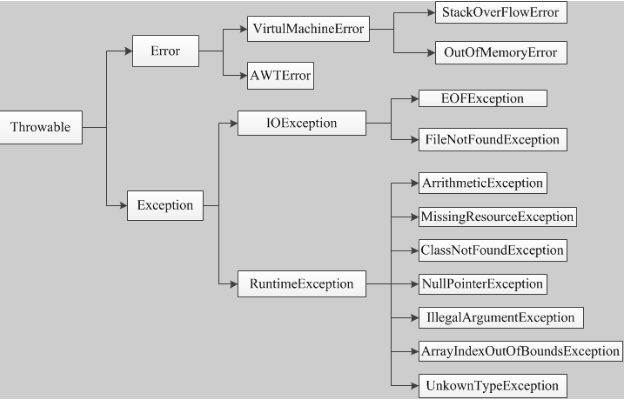
JVM执行可恢复对象的finalize（）方法时，可能使该对象或其他对象重新变成可达状态

JVM执行finalize（）出现异常，垃圾回收机制不会报告异常，继续执行

finalize（）并非一定执行，当程序执行完毕但是没有进行垃圾回收，finalize（）则没执行

五、Java基础类库

六、异常



Error：系统崩溃，动态链接失败，虚拟机错误等程序员无法处理的错误，不需要捕获。

编译时异常和运行时异常

编译时异常必须try，太繁琐，但是它能提醒程序员处理该异常的错误，运行时异常可以在需要的时候捕获，更加灵活。

异常机制可以使程序中的异常处理代码和正常业务代码分离，提高了程序的可独性和健壮性。

java的异常机制主要依赖5个关键字：try catch finally throw throws

代码执行情况：

无异常发生，执行完try块，执行finally块。

发生异常，中断try块的剩下代码，转入执行catch块，再执行finally块。

finally块用于回收物理资源，一定会执行的代码，即使try和catch块中有return语句。当try和catch中有System.exit(-1)语句，退出虚拟机，此时finally块才不会被执行。

try块，catch块，finally块的变量都是代码块的局部变量，三者访问不到彼此的变量。

多重异常捕获，先捕获小异常，再捕获大异常，否则编译时异常。

捕获异常 try catch

import java.io.\*;

public class TestException {

public static void main(String args[]) {

try {

FileInputStream f1 = new FileInputStream("C:/Users/well/Desktop/linux草稿.txt");

}catch(Exception e) {

e.printStackTrace();//打印程序所有出现该异常的位置

System.out.println(e.getMessage()); //打印异常信息

}

}

}

try {

代码块;

}catch(Exception e) {

处理代码;

}

e来接收异常对象.

catch(){}后的代码会继续执行.

多重捕获 try catch catch ...

import java.io.\*;

public class TestException {

public static void main(String args[]) {

try {

FileInputStream f1 = new FileInputStream("C:/Users/well/Desktop/linux草稿..txt");

}catch(IOException e1) {

e1.printStackTrace();//打印程序所有出现该异常的位置

}catch(Exception e2) {

System.out.println(e2.getMessage()); //打印异常信息

}

System.out.println("i am after catch");

}

}

如果没发生异常,try内的代码执行完毕后,执行System.out.println("i am after catch");

如果发生异常,根据多态原则,选择适宜的catch,注意,多重catch,子类异常一定要写在父类异常前面,否则编译错误,因为如果父类写在前面,所有的异常都被父类接收了,写在后面的子类异常无用. System.out.println("i am after catch");一定会被执行.

如果在第一个catch后加入return;不会输出i am after catch,如果在第二个catch后加入return,仍会输出i am after catch.

finally

try {

代码块;

}catch(Exception e) {

异常处理代码块;

}finally {

无论如何都会执行的代码块;

}

import java.io.\*;

public class TestException {

public static void main(String args[]) {

try {

FileInputStream f1 = new FileInputStream("C:/Users/well/Desktop/linux草稿..txt");

}catch(IOException e1) {

e1.printStackTrace();//打印程序所有出现该异常的位置

return;

}catch(Exception e2) {

System.out.println(e2.getMessage()); //打印异常信息

}finally {

System.out.println("i am after catch");

}

}

}

虽然第一个catch有return,但是仍会输出i am after catch.

throws

当方法中有必须catch的编译异常时,但是当前方法没有能力处理该异常或不项catch该异常,可以（方法签名）void f( ) throws Exception1，Exception2...,这样不在方法内try catch 也能编译通过.

调用有throws的方法的方法，必须捕获异常，或继续throws异常，直到main方法中处理异常，如果main方法也不处理，最终抛给虚拟机，虚拟机处理异常的方式是打印异常信息栈，中止程序。

重写throws方法时，抛出的异常必须不能比父类方法多，抛出的异常必须是父类抛出的异常的子类或相同。

g方法调用含throws的方法,必须

try {

f();

}catch(Exception e) {

异常处理代码;

}

或

void g( ) throws Exception {

f( );

}

throw

常用于判断语句,抛出一个异常实体,throw new Exception e;如果条件满足,程序执行了throw new Exception e,有两种处理方式:

(1)try { throw new Exception e } catch(Exception e) {异常处理代码;}

(2)在方法参数后面+ throws Exception;这样编译能通过.但是抛出异常后,后面的代码不能再继续执行了.

throw抛出的编译时异常，则必须显示捕获，或者throws，如果是运行时异常，则可以不理他，或者捕获。

throw与throws的比较

1、throws出现在方法函数头；而throw出现在函数体。

2、throws表示出现异常的一种可能性，并不一定会发生这些异常；throw则是抛出了异常，执行throw则一定抛出了某种异常对象。

3、两者都是消极处理异常的方式（这里的消极并不是说这种方式不好），只是抛出或者可能抛出异常，但是不会由函数去处理异常，真正的处理异常由函数的上层调用处理。

自定义异常

自定义异常,如果是运行型异常,继承runningException,如果是编译时异常,继承Exception.

异常对象是怎样来的?

就是在一个方法中,出现错误,就把错误参数封装成成员变量,做成一个异常类,类中的方法是关于参数要对该错误进行的处理方法.

public class TestBankAccount {

public static void main(String[] args) {

BankAccount b = new BankAccount(9527,0);

b.cunQian(2000.00);

b.seeMoney();

b.quQian(1000.00);

b.seeMoney();

b.quQian(1001.00);

}

}

/\*自定义异常\*/

class QiongException extends Exception {

private double shortMoney;

public QiongException(double shortMoney) {

this.shortMoney = shortMoney;

}

void printShortMoney() {

System.out.println("Sorry,you are short of money is $" + shortMoney);

}

}

class BankAccount {

private int id;//账户

private double yuE;//余额

public BankAccount(int id,int yuE) {

this.id = id;

this.yuE = yuE;

}

/\*查询余额\*/

void seeMoney() {

System.out.println("your total money is $" + yuE);

}

/\*存钱\*/

void cunQian(double inMoney) {

yuE += inMoney;

}

/\*取钱\*/

void quQian(double outMoney) {

try {

if(yuE >= outMoney) {

yuE -= outMoney;

}

else {

throw new QiongException(outMoney - yuE);

}

}catch(QiongException e) {

e.printShortMoney();

}

}

自动关闭资源的try语句

try(声明，定义，初始化IO流等资源类) {使用资源}

能够自动关闭的资源类必须实现Closeable接口的close方法。

七、Java集合

0、Collection

数组和集合的区别：数组能放基本数据类型和对象，集合只能放对象。  数组容量一定，集合可以保存数量不定和具有映射关系的数据。

import java.util.\*;

Collection的主要方法：

boolean add(Object o)

boolean addAll(Collection c)

boolean remove(Object o) 集合中有多个o时，只删除第一个 o

boolean removeAll(Collection c)从集合中删除c里包含的所有元素，删除至少1个返回true

boolean contains(Object o)

boolean retainAll(Collection c) 求与c的交集

void Clear() 请空集合

int size()

boolean isEmpty()判断集合是否为空

Object[] toArray() 把集合中的所有元素变成数组的成员，返回数组的引用

Iterator iterator() 返回一个Iterator对象

public class CollectionTest {

public static void main(String[] args) {

Collection c = new ArrayList();

c.add("西游记");

c.add("红楼梦");

c.add(1);

System.out.println(c);//[西游记, 红楼梦, 1] 重写了toString()

c.remove(1);

Object[] str = c.toArray();

for(int i = 0;i < str.length;++i) {

System.out.print(str[i]);//西游记红楼梦

}

}

}

1、Set

元素不可相等,不维持有序. 和Collection基本相同,没提供额外的方法.

HashSet   LinkedHashSet TreeSet   EnumSet

2、HashSet

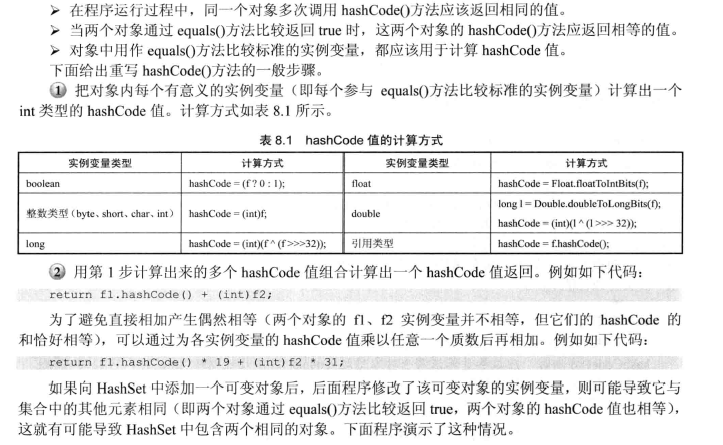
添加新元素 : 先根据元素的hashCode计算出存储位置,如果计算出的存储位置已经有元素,可能已经存入相等元素,也有可能不相等元素但是hashCode相同的元素加入了,如果此时草率的丢弃新元素,可能造成其实元素没加入过,但是永远无法加入容器.  所以再equals一下,如果是true,说明确实是前者原因,就丢弃,如果是后者,HashSet采取的措施是继续加入(链后面). HashSet可以存放null引用.不是线程安全的。

对Hash来说,元素相等意味着:equals,hashCode都要相等.

要想保证Set的性质 : 元素不相等,必须保证:equals相等,hashCode也相等(这条必须要100%保证,所以hashCode的因子要包全所有的equals成员. 而且元素一旦放入HashSet,不准修改成员变量,要是修改了,查找的时候,你就懵逼了:比如,你存的3,改成-2,找3的时候,根据3hashcode找到的存储位置是个-2,你说咋整?或者你干脆把存-2修改成3,容器就有两个3了,与Set性质相悖)

equals不相等,hashCode也不相等(无法避免的相等也可以,但是会造成性能下降,所以应该全力保证不相等) 综上: 含有Hash二字的容器,相等意味着 equals 和 hascode都相等(根据equals生成hascode) , 在其他非hash容器,相等意味着equals就OK, 但是为了通用性,最好equals和hashcode一起重写!

HashCode的计算方法：



LinkedHashSet是HashSet的子类,几乎和HashSet没差别,只是多出:

遍历LinkedHashSet,输出顺序和添加顺序一致,因为LinkedHashSet维护内部顺序,所以性能略逊于HashSet,但是用迭代器遍历,速度比HashSet快.

LinkedHashSet怎样维护内部添加顺序的?

很好办,思考静态链表,哈哈哈. 根据哈希函数找到存储位置后,把这个位置填入上次添加的元素的next位置.

3、TreeSet

TreeSet就是红黑树.TreeSet保证了遍历时的有序性. 自然排序和定制排序。

添加时,从根结点开始比较,确定插入左子树还是右子树,这是个比较的过程. 所以

(1) 添加的对象必须自身继承Comparable接口,实现compareTo(Object obj),比如基本数据类型的包装类,String类,Date类,Time类.

(2)CompareTo比较的原则可以自定义,自定义的叫定制排序,纯粹比较大小的叫自然排序.

import java.util.\*;

public class TestJava {

public static void main(String[] args) {

OkHasComparable a1 = new OkHasComparable(2);

OkHasComparable a2 = new OkHasComparable(1);

OkHasComparable a3 = new OkHasComparable(3);

OkHasComparable a4 = new OkHasComparable(3);

TreeSet c = new TreeSet();

c.add(a2); c.add(a1); c.add(a3); c.add(a4);

Iterator it = c.iterator();

while(it.hasNext()) {

OkHasComparable ohc = (OkHasComparable)it.next();

System.out.println(ohc.i);

}

System.out.print(c); System.out.println();

System.out.println(c.first());

System.out.println(c.last());

}

}

class ErrorNoComparable {

}

/\*

Interface ErrorNoCompareTo implements Comparable {

}\*/

class OkHasComparable implements Comparable {

int i;

public OkHasComparable(int i) {

this.i = i;

}

public int compareTo(Object obj) {

OkHasComparable ahc = (OkHasComparable)obj;

/\*缺少检查类型检查模块,以后会写了再写\*/

if(ahc.i == this.i) return 0;

else if(ahc.i < this.i) return this.i - ahc.i;

else return ahc.i - this.i;

}

public String toString() {

return this.i + "";

}

}

运行结果:

1  
2  
3  
[1, 2, 3]  
1  
3

compareTo方法三要素:(1) 同类型(会发生向下转型) (2) 相等返回0 ,大于返回+ 小于返回 -

TreeSet比较的是关键字,而不是对象整体,只要关键字相同便已经不能添加,即使equals不相等,也不能添加. equals相等,但是compareTo返回0,那它就是无法添加,TreeSet相不相等不看equals.

TreeSet每次插入新元素都会执行compareTo方法,要想安全使用TreeSet,就要遵守: 存放同一类型的对象.

对于TreeSet来说,相等元素不能再存放,相等标准 compareTo返回0.

注意两点: 若equals方法相等,也要保证compareTo方法相等. 若compareTo相等,但是equals不相等,与Set性质相悖.

不要修改已经入了容器的元素.

定制排序: compareTo方法,自定义关键字比较方法,你认为两个关键字相等就想等,谁大谁就大.

4、Set性能分析

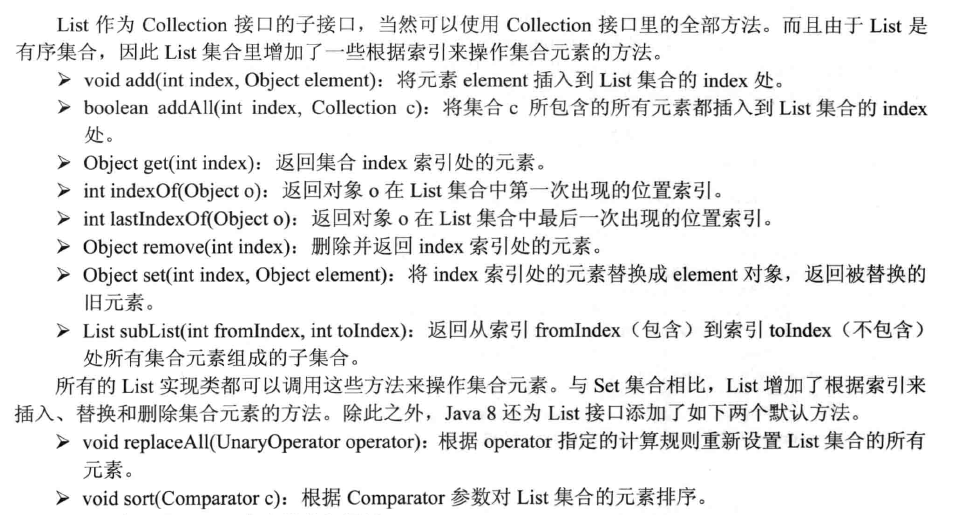
需要一个排序的Set的时候才使用TreeSet，否则使用HashSet。

LinkedHashSet只是遍历时比HashSet快，其他不如HashSet。

所有的Set都不是线程安全的，可使用Collections工具类synchronizedSortedSet方法包装成线程安全的。

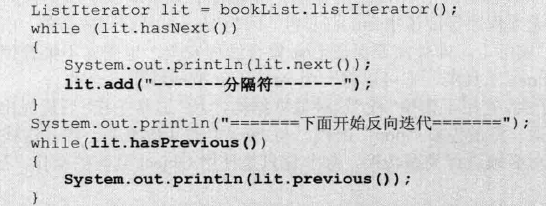
SortedSet s = Collections.synchronizedSortedSet(new TreeSet(…));

4、List





ListIterator能向前和向后迭代，能插入和删除元素，Iterator只能向后迭代和删除元素。



5、Map

HashMap LinkedHashMap TreeMap key-value对 keySet( )提前key为Set集合，

key是Set，value设置成null就变成了Set集合。

Map map **=** **new** HashMap**();**

map**.**put**(**"red"**,**1**);**

map**.**put**(**"yellow"**,**2**);**

map**.**put**(**"black"**,**3**);**

//key重复，会覆盖

map**.**put**(**"black"**,**0**);**

//判断是否包含key

map**.**containsKey**(**"black"**);**

//判断是否包含value

map**.**containsValue**(**1**);**

//根据key删除key-value对

map**.**remove**(**"black"**);**

//keySet()返回key组成的Set集合，get(key)得到key对应的value

**for(**Object key **:** map**.**keySet**()){**

System**.**out**.**println**(**map**.**get**(**key**));**

**}**

System**.**out**.**println**(**map**);**//{red=1, yellow=2}

6、HashMap和HashTable的区别

HashTable是线程安全的，HashMap不是，多线程时使用HashTable性能好。

HashTable的key和value都不允许时null，HashMap的key和value允许是null，所以，key仅能一个null，value不限。

判断两个key相等的标准，equals和hashCode都相等，判断两个value相等的标准，满足equals即可。

LinkedHashMap使用双向链表维护插入顺序，迭代顺序和插入顺序相同。

7、线程安全的容器

可以使用Collections类提供的静态方法把ArrayList,LinkedList,TreeSet,HashSet,TreeMap,

HashMap包装成线程安全的集合。

Collection c **=** Collections**.**synchronizedCollection**(new** ArrayList**());**

List list **=** Collections**.**synchronizedList**(new** LinkedList**());**

Set set **=** Collections**.**synchronizedSet**(new** HashSet**());**

Map map **=** Collections**.**synchronizedMap**(new** TreeMap**());**

java.util.concurrent包下提供了大量支持高效并发访问的集合类和接口。

以Concurrent开头的集合类，支持线程安全的写操作，读操作时，并不锁定集合。

以CopyOnWtite开头的集合类，对CopyOnWriteArrayList写操作时，会在底层拷贝一个数组，写操作对副本操作，读操作和写操作不是同一个数组，因此读操作比较快，写操作需要复制数组，开销大，因此CopyOnWriteArrayList适合频繁读，很少写操作的情况，比如缓存。

八、泛型

七、流

八、多线程

九、网络编程

十、面试补充

1、abstract和static，synchronized，native能否同时使用？

static和private修饰的方法不能被重写或继承，因此abstract和static，private不能同时使用。

java无法实现底层操作，需要依赖其他语言，比如C语言，实现步骤：

java程序声明native方法，无需实现，javac编译生成class文件，javah编译class文件生成.h文件，写一个cpp文件，实现native方法，并包含.h文件，将.cpp文件编译成动态链接库文件，用System或Runtime方法加载上述动态链接库文件，java就可以调用这个native方法了。虽然native方法java无需实现，但是native和abstract不能同时使用。

synchronized方法同步锁对象是this，而抽象方法无法确定this是什么，所以一般也不同时使用。

2、什么是JAVA虚拟机（JVM）？

java虚拟机是用软件模拟，具有完整硬件体系和指令集的抽象计算机，提供JAVA程序的运行环境。

3、JAVA为什么被称为与平台无关的语言？

java源文件被编译成与平台无关的字节码，字节码可以不加修改的在不同平台上的JVM上运行，JVM会把字节码解释成具体平台上的机器指令。

4、JDK和JRE

JRE：java运行时环境，包括java虚拟机，java核心类库和支持文件。

JDK：java开发包，除包含JRE，还包含编译器和其他工具，如javadoc，java调试器等。

5、String、StringBuffer、StringBuilder的区别

String是不可变类，实例创建后不能再修改，频繁修改字符串不应该使用String类，因为每次变化都要新创建一个字符串，内存开销大。StringBuffer和StringBuilder是可以修改的字符串，StringBuilder不是线程安全的，单线程下使用性能比StringBuffer好，StringBuffer是线程安全的。String覆盖了equals方法和hashCode方法，而StringBuffer没有覆盖equals方法和hashCode方法，所以，将StringBuffer对象存储进Java集合类中时会出现问题。

6、String str = new String("xyz")会创建几个对象？

String str = “xyz” 或 String str = new String(“xyz”)

前者先在字符串常量池中，查找xyz，没有的话，在常量池添加xyz，返回一个String引用；如果有，返回已经存在的字符串的引用。若采用new，在常量池查找xyz，若没有，添加一个xyz。并在堆内存中创建对象，返回这个对象的引用，所以会创建两个对象。

String str1 **=** "Hello"**;**

String str2 **=** "Hello"**;**

String str3 **=** **new** String**(**"Hello"**);**

System**.**out**.**println**(**str1 **==** str2**);**//true

System**.**out**.**println**(**str1 **==** str3**);**//false

System**.**out**.**println**(**str1**.**equals**(**str2**));**//true

System**.**out**.**println**(**str1**.**equals**(**str3**));**//true

String str4 **=** "a"**;**

String str5 **=** "b"**;**

String str6 **=** str4 **+** str5**;**

String str7 **=** str4 **+** "b"**;**

String str8 **=** "a" **+** "b"**;**

String str9 **=** "ab"**;**

String str10 **=** **new** String**(**"ab"**);**

System**.**out**.**println**(**str9 **==** str6**);**//false

System**.**out**.**println**(**str9 **==** str7**);**//false

System**.**out**.**println**(**str9 **==** str8**);**//true

System**.**out**.**println**(**str9 **==** str10**);**//false

System**.**out**.**println**(**str9 **==** str10**.**intern**());**//true

String str11 **=** **new** String**(**"ef"**);**

String str12 **=** "ef"**;**

System**.**out**.**println**(**str11 **==** str12**.**intern**());**//fasle

System**.**out**.**println**(**str11**.**intern**()** **==** str12**);**//true

System**.**out**.**println**(**str11**.**intern**()** **==** str12**.**intern**());**//true

7、String s="a"+"b"+"c"+"d";一共创建了几个对象？

String s = “a” + “b” + “c” + “d”; String x = “abcd”; s == x is true

因为编译器会对”a” + “b” 这种表达式优化。所以只创建了一个对象。

8、try {}里有一个return语句，那么紧跟在这个try后的finally {}里的code会不会被执行，什么时候被执行，在return前还是后?

在return中间执行，比如 return a += 1; 执行完a = a + 1后才转入finally块。