# 总结：

public static 包装类 valueOf**(**相应基本数据类型**);**

int intValue**(**Integer i**);**

static int Integer**.**parseInt**(**String**);**

String**:**

public char charAt**(**int index**);**

public int compareTo**(**String str**);**

public String**(**byte**[]** bytes**,** int offset**,** int length**);**

public String**(**char**[]** value**,** int offset**,** int count**);**

public boolean endsWith**(**String suffix**);**

public int indexOf**(**int ch**);**

public int indexOf**(**int ch**,** int fromIndex**);**

public int indexOf**(**String str**);**

public int indexOf**(**String str**,** int fromIndex**);**

public int lastIndexOf**(**int ch**)**

public boolean isEmpty**();**

public String substring**(**int beginIndex**);**

public String substring**(**int beginIndex**,** int endIndex**);**

public char**[]** toCharArray**();**

length**();**

toLowerCase**();**

toUpperCase**();**

Arrays**.**toString**(**arr**);**

Arrays**.**sort**(**arr**);**

Arrays**.**copyOf**(**int**[]** original**,** int newLength**);**

正则：

**[]-------**表示一个位置，如果该位置上只有一种可能，可以忽略不写

**()-------**表示一个组合（至少是两位，并且是连续的）

**{}-------**用来定义字符串长度的范围

public String**[]** split**(**regex**);**

public boolean matches**(**regex**);**

StringBuffer**/**StringBuilder：

append**(**Object obj**);**

insert**(**int index**,** Object obj**);**

delete**(**int beginIndex**,** int endIndex**);**

deleteCharAt**(**int index**);**

Collection**:**

boolean add**(**Object obj**);**

void add**(**int index**,** Object obj**);**

int size**();**

void clear**();**

boolean isEmpty**();**

boolean contains**();**

E get**(**int index**);**

E set**(**int index**,** E element**);**

E remove**(**int index**);**

boolean remove**(**Object obj**);**

void addFirst**(**E e**);** //only used in LinkedList

void addLast**(**E e**);** //only used in LinkedList

public static List Arrays**.**asList**(**Array arr**);**

list**.**toArray**(**arr**);**

Collections**.**sort**(**list**);**

Map**:**

V put**(**K k**,** V v**);**

Set**<**K**>** keySet**();**

int size**();**

V remove**(**K k**);**

boolean remove**(**K k**,** V v**);**

public boolean containsKey**(**Object key**);**

public boolean containsValue**(**Object value**);**

V get**(**Object key**);**

boolean isEmpty**();**

public void clear**();**

Iterator**;**

Iterator it **=** list**/**set**.**iterator**();**

boolean hasNext**();**

E next**();**

File**:**

public static String separator**;**

public boolean createNewFile**()** **throws** IOException**;**

public boolean mkdir**();**

public boolean delete**();**

public File getAbsoluteFile**();**

public String getAbsolutePath**();**

public String getName**();**

public int compareTo**(**File f**);**

public long length**();**获取文件中符号的个数

public File**[]** listFiles**();**

public String**[]** list**();**

public boolean exists**();**

public boolean isAbsolute**();**

public boolean isDirectory**();**

public boolean isFile**();**

public boolean isHidden**();**

Reader：//字符

FileReader**:** public int read**(),**

public int read**(**char**[]** buf**)**

BufferedReader**:** public int read**(),**

public int read**(**char**[]** buf**,** int off**,** int len**),**

String readLine**()**

Writer**:** //字符

FileWriter**:** public void write**(**char**[]** buf**),**

public void write**(**char**[]** buf**,** int beginIndex**,** int len**),**

public void write**(**String str**),**

public void write**(**String str**,** int beginIndex**,** int len**)**

BufferedWriter**:** public void write**(**int ch**),**

public void write**(**char**[]** buf**,** int off**,** int len**),**

public void write**(**String str**,** int off**,** int len**),**

public void newLine**()**

InputStream**:** //字节

FileInputStream**:** public int read**(),**

public int read**(**byte**[]** buf**),**

public int read**(**byte**[]** buf**,**int beginIndex**,**int len**)**

BufferedInputStream**:** public int read**(),**

public int read**(**byte**[]** buf**,**int beginIndex**,**int len**)**

OutputStream**:** //字节

FileOutputStream**:** public void write**(**int b**),**

public void write**(**byte**[]** buf**),**

public void write**(**byte**[]** buf**,**int beginIndex**,**int len**)**

BufferedOutputStream**:** public void write**(**int b**),**

public void write**(**byte**[]** buf**,**int beginIndex**,**int len**)**

InputStreamReader**/**OutputStreamWriter //转换流

**for(**int i**=**1**;**i**<**arr**.**length**;**i**++){**//冒泡算法

**for(**int j**=**1**;**j**<**arr**.**length**-**i**;**j**++){**

//交换位置

**}**

折半查找：查找指定元素在数组中的位置，如果不存在返回**-**1

public static int zheBan**(**int**[]** ar**,**int num**){**

int begin**=**0**;**//查找起始位置

int end**=**ar**.**length**-**1**;**//查找结束位置

/\* 1.如果结束位置小于起始位置，这样的数组不存在

\* 2.如果结束位置大于起始位置，数组中有多个元素

\* 3.如果结束位置等于起始位置，数组中只有一个元素

\* \*/

**while(**begin**<=**end**){**

int middle**=(**begin**+**end**)/**2**;**//中间位置

**if(**num**==**ar**[**middle**]){**

**return** middle**;**

**}else** **if(**num**>**ar**[**middle**]){**

begin**=**middle**+**1**;**

**}else{**

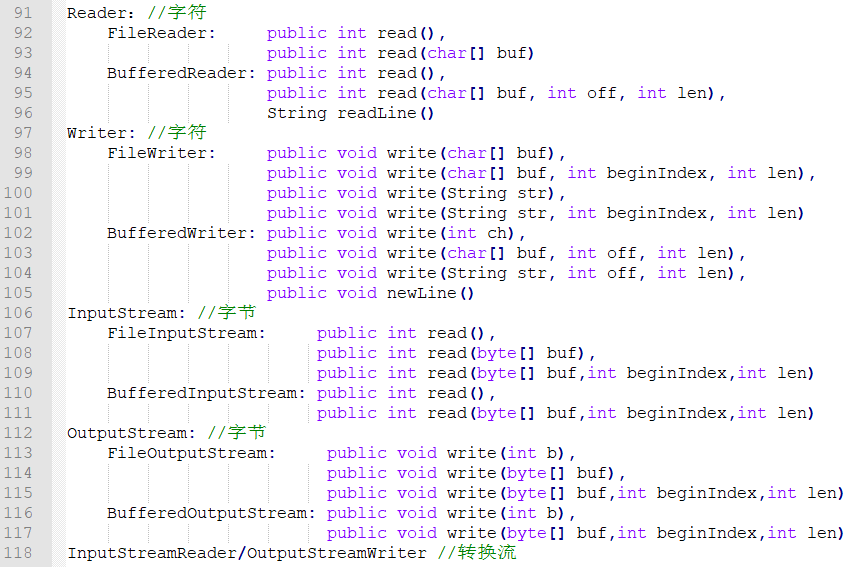
end**=**middle**-**1**;**

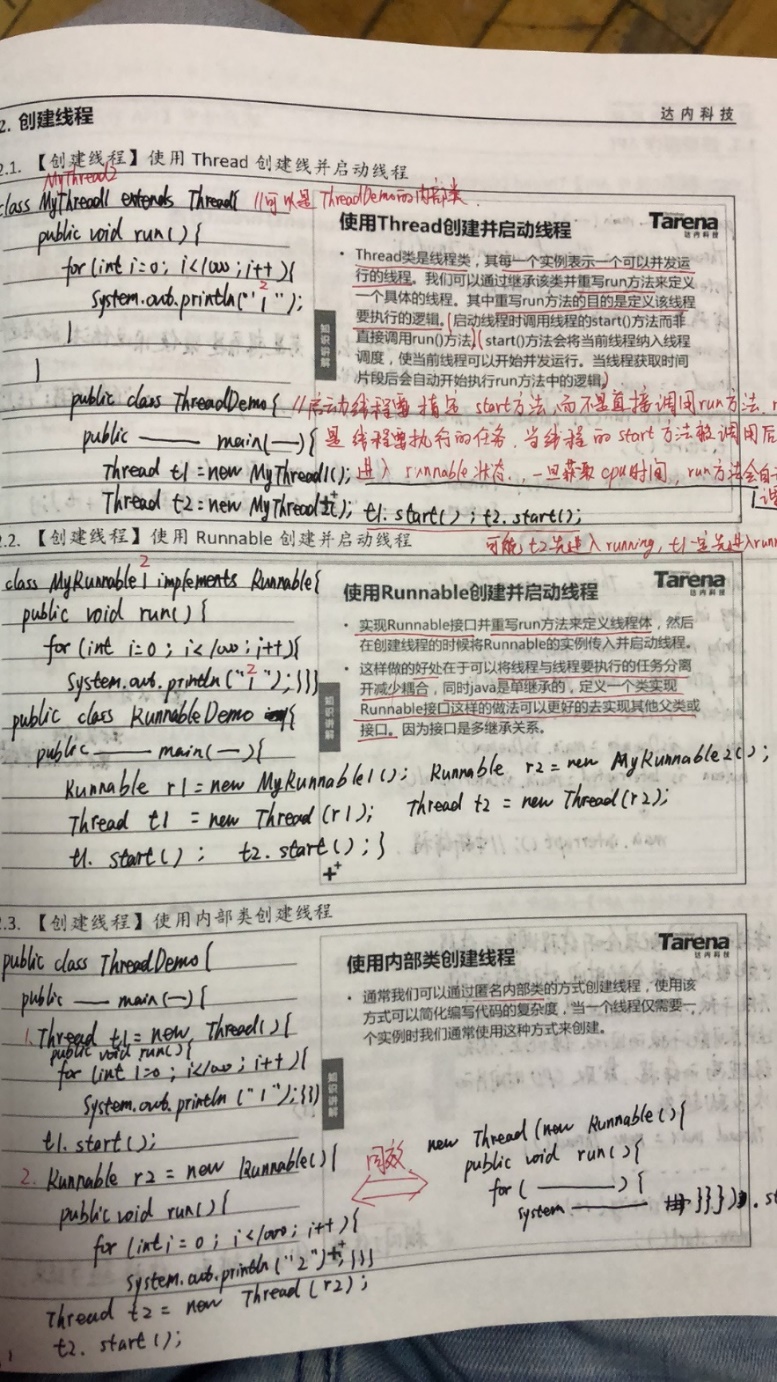
**}**

**}**

**return** **-**1**;**

**}**





# 变量与数据类型

## 含义：

生活中：变化的数量-----》只能通过数字的变化体现

Java中：一个数据存储空间的表示------》空间

## 定义：

格式：数据类型 变量名 = 值;

## 变量名：

定义规则（只能出现3种特殊符号\_ $ .（例如com.））

1. 只能用字母 \_ $开头
2. 余下部分可以是字母 \_ $ 数字，长度不宜过长，且有意义，注意千万不能出现特殊符号（如空格）
3. 不能使用Java中的关键字和保留字
4. 驼峰命名法
5. 如果变量名是一个单词，全部小写
6. 如果是两个或两个以上，从第二个开始首字母大写

## 数据类型（两大类）

1. 引用数据类型：类 类型

String：字符串 存放多个字符 值在””中

1. 基本数据类型：
2. 数值型：占内存空间的大小，有小到大
   1. 整数：

byte---字节型整数

short----短整型

int----整型 Java默认的整数类型

long---长整型 值的后边可以加上L或l

* 1. 浮点型（小数型）：

float---单精度浮点型 值的后边必须加上F或f

double--双精度浮点型 Java默认的浮点型

1. 字符型：char 存放单个字符 值在’’中

单个字符：一个数字，一个字母，一个汉字，一个转义字符

转义字符：

含义：将Java中与计算机所冲突的快捷符号进行转换

格式：\任意符号

常用：\t 多个空格----Tab 制表符

\n 回车

\\ \” \’

1. 布尔型：boolean
2. 值只有两个true/false
3. 任意关系表达式或逻辑表达式的结果
4. 基本数据类型的转换
5. 注意：布尔型不可以和任意的基本数据类型进行转换
6. 前提：两种数据类型相互兼容
7. 分类：
8. 自动数据类型转换：自转--》由小到大

byte

short--int--long--float--double

char

常用阿斯克码值

A~Z 65~90

a~z 97~122

0~9 48~57

b)强制数据类型转换：强转--》由大到小

格式：小的数据类型 变量名 = (小的数据类型)大的数据类型的值;

1. 如何定义有效变量？
2. 根据值的数据类型去定义变量的数据类型
3. 看变量接下来做的操作

例如：**int** age = 18;

String nianLing = "18";

用int定义的年龄可以进行数学运算

# 运算符（5类）

## 赋值运算符（5个）：= \*= /= += -=

=：将符号右侧的值赋给左侧的变量

## 算数运算符（7个）：+ - \* / % ++ --

+：

如果操作的数据都是基本数据类型（除布尔型以外），做和的运算

如果操作的数据中包含引用数据类型，则从该引用数据类型开始，后边做的都是连接的操作

/：

如果两个整数相除，结果一定是整数

如果想要得到与现实生活中一致的数据，至少保证除数或被除数有一个是浮点型

%：余数的正负只与被除数有关

注意：什么时候使用/？什么时候使用%？

相同点：都会做除法的操作

不同点：

判断数字之间的关系时，使用%

++：在自身原有值的基础上+1

如果作为单条语句操作，则x++与++x没有任何区别

如果作为组合语句操作：

x++:先执行操作，再自身+1

++x:先+1，再执行操作

--：在自身原有值的基础上-1

用法同++

## 逻辑运算符（3个）

!逻辑非-----取反

&&逻辑与（短路与）

&按位与

(4>5)&&(4==4)---false（如果第一个是假，第二个就不用判断了）

(4>5)&(4==4)---false如果第一个是假，第二个继续判断）

||逻辑或

|按位或

结果一定是布尔型

## 关系运算符（6个）：== != > < >= <=

> < >= <=只用于数字上的比较

==：数学上的等于

如果比较的是基本数据类型（除布尔型以外），是值是否相等

如果比较的是引用数据类型，在内存空间中的地址是否相同

结果一定是布尔型

## 条件运算符：

等同于if...else...语句

格式：(任意关系表达式或逻辑表达式)?结果1:结果2

## 运算符的优先级（从高到低）

()

++ -- !

算数运算符

关系运算符

逻辑运算符

条件运算符

赋值运算符

例：

已知：共用条件

int x = 4;

int y = 5;

分别输出

System.out.print((++x==y) && (++x!=y));

T((5==5)&&(6!=5))

System.out.print((x++==y) && (x++!=y));

F

System.out.print((++x==y) && (x++!=y));

F

System.out.print((x++==y) && (++x!=y));

F

# 从控制台录入数据-----java.util.Scanner;

1. Scanner sc = new Scanner(System.in);

导包：java.util

1. 通过输出语句进行操作提示
2. 接收数据 格式
3. 整数型：int 变量名 = sc.nextInt();
4. 浮点型：double 变量名 = sc.nextDouble();
5. 字符型：String 变量名 = sc.next();

# 流程控制语句

## 条件分支语句

if...else if...else与switch的异同点

相同点：都是进行多个条件判断

不同点：

条件：

if...else if...else条件是任意关系表达式逻辑表达式，结果只能是布尔型（等值表达式或不等值表达式）

switch条件是一个确定的数据，并且该数据必须满足byte/short/int/char/String（jdk1.7版本后）（等值表达式）

如何使用？

如果条件选择较少（一般在5个一下），并且满足以上五种数据类型，建议使用switch，效率稍高

实际开发过程中，建议使用if...else if...else

## 循环控制语句

循环四要素：四个什么

1. 循环初始化条件：什么时候开始循环
2. 循环判定条件：满足什么才循环
3. 循环体：循环的是什么
4. 循环结束条件：什么时候停止循环

while与do...while的异同点？

相同点：都是用来进行重复操作的

不同点：while先判断，再执行

do...while先执行一次，再判断

for与while的异同点？

相同点：都是用来进行重复操作的

不同点：在初始化条件定义位置；for定义在循环中；while定义在循环外。简而言之，for比while更节省内存空间

如何使用for与while？

知道循环次数使用for

不知道循环次数使用while，并且条件一般情况下是true

双重循环

for：

外循环控制行，内循环控制列

外循环执行一次，内循环执行一遍

结束语句

break：终止语句，用来结束整体流程控制的，能够作用于所有的循环语句和switch语句中

continue：中断语句，用来结束本次循环的，

只能作用于所有的循环语句

return：终止语句，用来结束方法的

break与continue的异同点？

相同点：都可以结束循环操作，可以定义在任意的流程控制语句中

不同点：continue只能作用于循环语句，且只结束本次操作

如何结束指定层流程控制语句？

结束语句只结束与之定义最近的流程控制语句

想结束指定层，需在指定层流程控制语句前，起别名。



# 数组

## 含义：存放一组相同数据类型的集合

## 定义：

1. int[] ar = {4,80,8,5,2};

已知元素的值时，使用

1. int[] ar = new int[2];

已知元素个数时，使用

1. int[] ar = new int[]{4,80,8,5,2};

在集合中使用

## 常识：

1. 元素下标从0开始
2. 元素表示格式：ar[i]
3. 元素最大下标数永远比数组长度小1
4. 常用数据类型数组的默认值

int----0

double---0.0

String---null

char-----空格

## 向数组中存取数据

存数据：for

取数据：for与增强for(增强for不能取某一个，只能全部取出)

## 最值问题、冒泡排序

# 随机数

## Math.random()

取一个随机数。

返回值double类型，范围[0.0,1.0)

格式：(int)(Math.random()\*10);

## Random类

取多个随机数

nextInt()---int类型范围内的任意一个整数

nextInt(int num)---[0,num)之间的任意一个正整数

# 面向对象：

三大特征：封装、继承、多态

一个特殊使用：抽象

## 如何理解面向对象？

万事万物皆对象

名词提炼法：“笔”

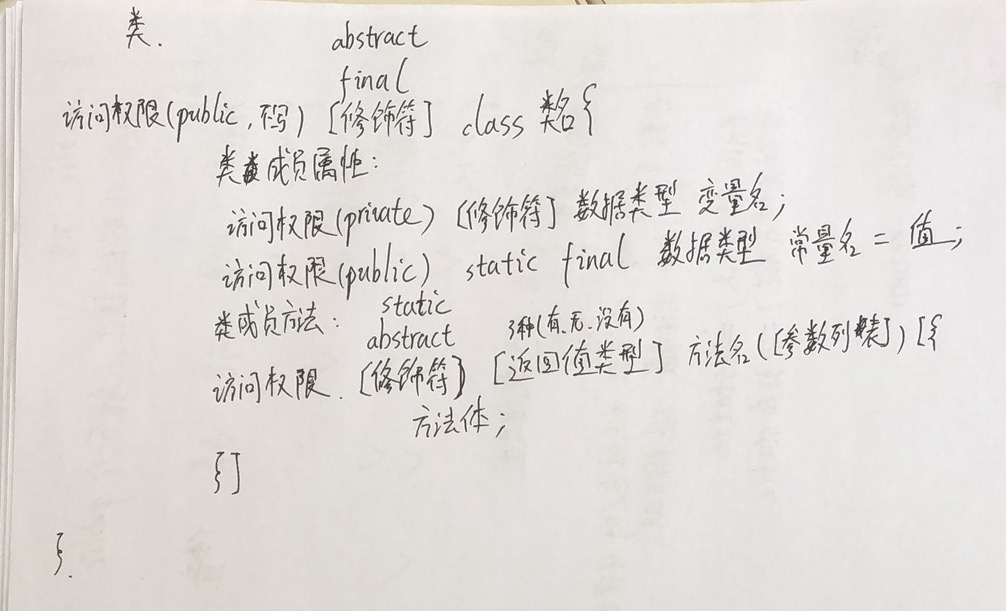
1. 含义：能够书写内容的物体---类
2. 具体：水性笔、油笔、铅笔---对象

类：对具体事物的抽象概念------抽象

对象：是具体事物的一个实例----具象

类与对象相互依存，通过类创建对象，先有类才能有对象；通过对象可以归纳出一个类

## 如何定义类和对象？



## 类----名词提炼法：

人解锁手机

名词：人、手机-------类

动词：解锁-------方法

锁在手机上，最了解锁的是手机类中，人类去调用手机

人{

手机 s = new 手机（）;

s.解锁（）;

}

手机{

解锁（）{

........;

}

}

类的格式：

[访问权限] [修饰符] class 名{

}

## 两图一表：

流程图、类图、数据表（针对数据库）

类图：

|  |
| --- |
| 类名 |
| 属性 |
| 方法 |

属性：变量（全局变量）、成员变量

格式：变量名：数据类型

方法：成员方法

格式：名：返回值

## 如何定义类成员（成员变量、成员方法）

### 成员变量：

格式：[访问权限] [修饰符] 数据类型 名[=值];

### 成员方法：

格式：[访问权限] [修饰符] [返回值] 名 ([参数列表])[{

方法体;

}]

## 访问权限：

### 类的访问权限：public/不写

在同一代码编辑区，可以定义多个类（但不建议），但只能有一个被public修饰。

### 类成员的访问权限

类成员 作用域：同类 同包 子类 任意

private √

不写 √ √

protected √ √ √

public √ √ √ √

注意：所有成员属性一律私有化

所有成员方法一律公有化

如果问java中的访问权限：类的访问权限和类成员的访问权限

#### 修饰符：

1. static:静态的，只可以修饰类成员

代码块：优先加载（无论主函数是否执行，static代码块都执行）（数据库）

类成员：可以直接通过类名去调用（成员：常量）

格式：[访问权限] static final 数据类型 名（全部大写）= 值；

静态方法中不能使用this，super关键字。

1. abstract:抽象的，可以修饰类和成员方法（特点：没有方法体）

类：不能实例化对象

1. synchronized:代码块、方法

相同点：可以解决多线程安全隐患问题

不同点：代码块：任意对象

方法：this

1. final:最终的，可以修饰类成员和类

类：不能被继承

属性：必须是常量public static final 数据类型 常量名 = 值；

注意：常量名全部大写，值一定不能忽略

方法：

#### 分类：

A:返回值：

没有------不写（省略）

无-------写，返回值类型：void

有-------写，返回值类型为任意数据类型

B:参数：

有：多个参数，之间用逗号隔开

无

#### 注意：

A:有返回值：return：

可以定义多次，但只返回一个结果

一般定义在方法结尾，其后的代码不能执行

B:有参数

形参：方法被定义时，传递的参数必须写数据类型

实参：方法被调用时，传递的参数不需要写数据类型

C:本身：

方法不能嵌套

主函数不能外部调用（没有意义）

1. 自定义类{

成员变量；

构造函数（有参、无参）

get/set(有参、无参) 成员方法

自定义成员方法

}

1. 值传递：通过不同方式给成员属性赋值
2. 在自定义类中，在定义成员属性的同时，直接赋值

private String name = "";

1. 通过对象名打点调用（除私有化外，不符合封装思想）

不建议使用1）和2）

1. 构造函数

A:作用：a实例化对象

b给成员属性赋值

B:格式：

public/protected 方法名（类名）（[参数]）{

}

给几个成员变量赋初值，参数就有几个

C:分类：

有参：外部给成员属性赋值

无参：给成员属性赋初值

D:特点：a没有返回值

b方法名与类名相同

c如果没有明确定义构造函数，Java会提供一个默认的无参构造，如果明确定义，则不会提供

E:get/set

get:外部获取

set:外部赋值

## 封装-------面向对象特征之一

### 含义:

对不想让外部访问的代码进行“相对”的保护

### 封装什么？

对类的内部进行封装，只对类成员起作用

成员属性：全部私有化，即被private修饰

成员方法：尽可能多的将功能代码定义自定义类中的自定义方法中，简化测试类

### 优点：

降低耦合，使外部不能任意调用成员属性。若想调用私有方法，需要使用get/set

在一定程度上提高了代码的安全性

## 继承

面向对象特征之一

1. 含义：向上找共性，将多个类中相同的代码提取出来，定义在一个类中，该类就是父类，其它为子类
2. 前提：多个类之间存在着横向的逻辑
3. 父类与子类的关系：

父类：更具有共性、通用性

子类：更具有特性

1. 特性：单继承

一个父类可以有多个子类

一个子类只能有一个父类

1. 一定继承：父类中所有public/protected修饰的类成员

一定不能继承：父类中所有private修饰的类成员；构造函数不被继承。

选择继承：如果父类、子类在同一个包中，也可以继承被不写修饰的类成员

1. 优点:提高代码重用性
2. 使用：父类与子类共有的类成员，子类可以直接调用，子类中特有的类成员父类不能调用

## 多态

面向对象特征之一

1. 含义：

生活中：同一事物，在不同条件下的不同状态

Java中：同一对象，指向不同的引用

对象声明(创建)：Pet p;

对象实例化：Pet p=new Pet();

Pet p=null;

1. 如何判定对象的本质和引用？

public class Dog extends Pets{}

public class Cat extends Pets{}

Pet p1=new Dog();

Pet p2=new Cat();

本质（判定一个对象是哪个类）：看对象前

引用（判定一个对象是哪个类对象）：看new之后

1. 引用数据类型转换

自转：格式：父类 对象名=new 子类（）；

强转：格式：子类 对象名=（子类）父类对象；

1. instanceof运算符：

A含义：判定指定对象是否是指定类或其子类对象的运算符，结果一定是布尔型

B格式：boolean flag = 对象名 instanceof 类名；

1. 优点：A:降低耦合

B:提高重用性

1. super与this

super：父类对象，只能定义在非静态方法中，永远在前

this：本类对象，只能定义在非静态方法中

1. 重载Overload
2. 含义：在同一个类中，出现一次以上名字相同的方法
3. 构成要素：

A:与返回值无关（与想重载的方法一致，若想重载的方法返回值类型是void或任意数据类型，重载的方法都可以。）

B:方法名相同

C:参数不同（参数个数，数据类型，顺序）

1. 重写Override
2. 含义：在子类中对父类已有方法进行功能性的增强
3. 构成要素：

A:方法名相同

B:参数相同

C:返回值一致或是子类

D:访问权限不严于父类

注意：构造函数可以重载，不可以重写；构造函数只能被调用，不能被继承。

重载与重写比较

位置 方法名 返回值 参数列表 访问权限

重载 同类 相同 无关 不同 无关

重写 子类 相同 一致或是子类 相同 不严于父类

1. 抽象类VS普通类VS接口

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 定义关键字 | 继承或实现的类 | 属性 | 构造 | 方法 |
| 普通类 | class | extends | 变量、常量 | 既能定义，也能实例化 | 普通 |
| 抽象类 | abstract class | extends | 变量、常量 | 只能定义，不能实例化 | 普通，抽象 |
| 接口 | interface | implements | 常量 | 既不能定义，也不能实例化 | 抽象 |

接口：只能定义常量，只能用public修饰

不能定义变量

不能定义构造函数，不能定义get/set方法

只有抽象方法，abstract关键字可写可不写

抽象类与接口

相同点：都能定义常量，构造方法都不能实例化，都能定义抽象方法

区别：关键字：定义和继承（实现）

属性：抽象类不能定义变量。

构造：抽象类可以定义构造

方法：抽象类可以有普通方法

什么时候用接口：只定义抽象方法、常量

接口作用：避免Java中单继承的局限性，实现“类似”多继承的这种形式

三个关键字：interface:定义接口的关键字

implements:实现接口的关键字

instanceof:运算符，返回布尔类型

15.面向对象编程思想编写代码“套路”

1. 读：需求------读明白
2. 画：“两图一表”-------流程图，类图；数据结构表
3. 编（根据类图编写）

A:成员属性

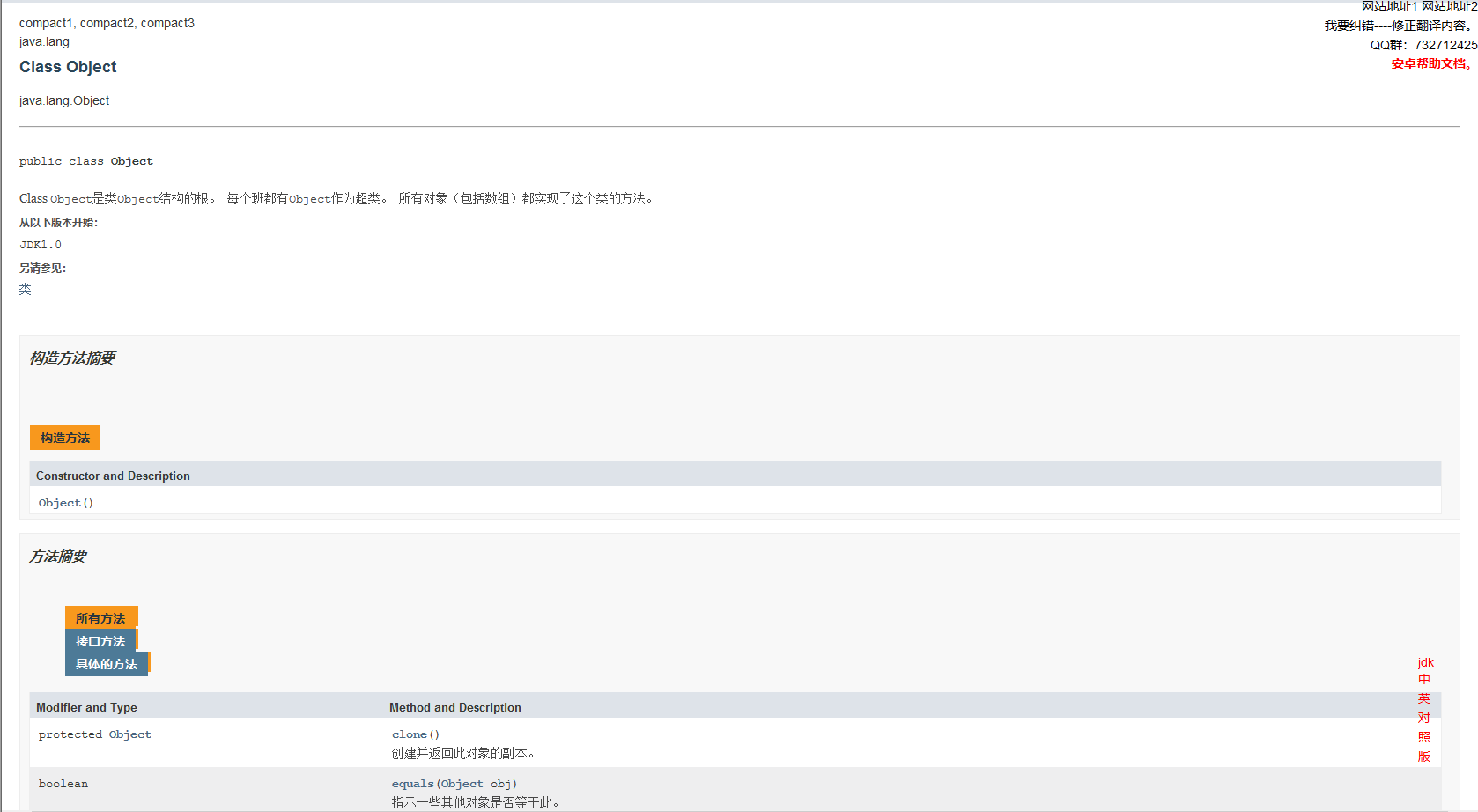
B:构造函数（有参、无参）

C:get/set方法

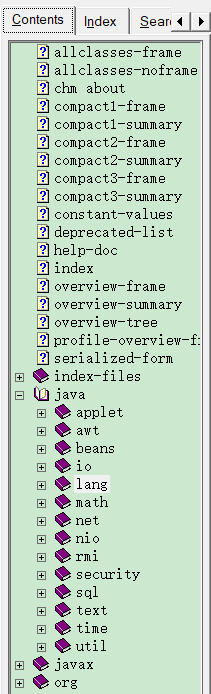
D:成员方法

E:测试类（带主函数）去测试

# Java高级



* + - 1. 是否是要找的，类名，类所属的位置，类结构
      2. 描述（前一两句话）
      3. 字段摘要（用的话再看），构造摘要（先看），方法摘要（方法的封装）

包的层次结构（中间还有一个）

# 工具类

1. 、Java常用包文件----5个
2. java.lang包------语言包，包含类：System类、Object类、String类包装类（将所有的基本数据类型包装成引用数据类型）、Math类

Java中唯一一个只要创建Java项目就会自动导入的包

1. java.util包------工具包，包含类：Scanner类、Random类、Arrays类、集合的所有类、Date类
2. java.io包-------文件与流包，包含类：文件类，所有的流资源类
3. java.net包-------网络编程包
4. java.sql包-------操作数据库包（JDBC）

# 熟练使用Java帮助文档-----Java API

1、如果知道包------目录

2、如果知道类或方法-------索引

Java中的导包

关键字：import java.包名.

如果第二个点后首字母大写，那么导入的是该包中的类

如果第二个点后全部小写，那么导入的是该包中的子包

\*代表当前包中的所有类（点前面）

用指定字符，去替换指定字符串的指定位置的符号

1、字符串对象使用

2、public char set(int index,char ch){}

# Object类

1、Java中所有类的父类（或者叫超类、基类）

2、只有无参构造

3、是Java中唯一一个没有父类的类

4、只要是Java中的类，如果没有明确继承哪个类，那么默认继承Object类；Java中的类不是Object类的直接子类，就是它的间接子类

5、常用方法

public boolean equals(Object obj)用来判定内存地址是否相同，等效于==（判定引用数据类型）

==：用来判定基本数据类型值是否相等；如果是引用数据类型，判定的是内存空间地址值是否相等

public int hashCode()返回已生成对象的字符串表现形式

# Java小常识

1、下标永远从0开始

2、取值范围[begin,end)-----左闭右开（元素的下标）（正则表达式{}左右均闭）

3、大部分将一个对象转换成另一种对象的方法，都用to开头

4、外部获取元素-----方法名一定有get

外部给元素修改或赋值-----方法名一定有set

5、如果方法的返回值或参数列表是任意的数据类型，那么返回值可以定义为Object，参数列表可以定义为Object obj

6、只要方法中提到位置，那么参数列表中必然会定义一个int index

7、Java中一个特殊的取值范围[0,65535]（阿斯克码值，哈希码值，逻辑端口号）

8、两集，两表

“两集”

UTF-8：国际通用编码表

GBK：中国编码集

“三码”

阿斯克码表：包含所有已知符号和常用简体汉字

哈希码表：作为已生成（必须有new）对象的唯一标识

unicode码：用来判定是否是汉字[\u4e00 - \u9fa5]

9、final可以修饰方法（jdk1.8开始）：方法不能被重写（比如getClass()）

10、 private final是绝对不可以重写的，其他的是可以有条件被重写的。

# Math类

最终类，所有的方法都是静态的

常用类成员：public static double PI；数学上关于计算圆的∏

public static double random()返回一个在[0.0,1.0)之间的任意无限不循环小数

# Random类

获取任意基本数据类型（数值型）的随机数类

public int nextInt()返回一个在int取值范围（-2的31次方～2的31次方-1）内的任意随机数类

public int nextInt(int num)返回一个在[0,num)范围内的任意随机数

num必须大于0

ps:

什么时候使用random（）？什么时候使用Random类？

1. 如果获取的随机数在[0,10)之间并且是一个，建议使用random（）

格式：int r=(int)(Math.random()\*10);

1. 如果获取的随机数范围不确定，建议使用Random类

# Date类

yyyy-----四位的年

MM-----两位的月

dd-----两位的天（DD按年来算的天）

HH-----24进制的小时

mm-----分

ss-----秒

aa-----上下午

zz-----所在时区

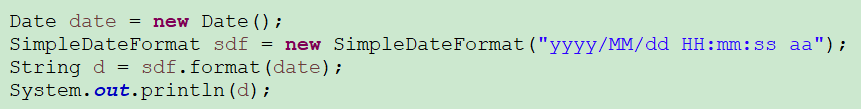
1. 如果获取当前系统时间------java.util.Date;

格式：Date d=new Date()

1. 如果获取的是数据库中的时间------java.sql.Date;

格式：Date d=new Date(long s)参数为过了多少毫秒

数据库默认时间为1970-01-01 00:00:00



# 包装类

## 包装的是什么？

是所有基本数据类型

## 为什么使用包装类：

由于Java是面向对象的编程语言，有时，需要操作对象来完成对基本数据类型的操作

## 包装类的名字

除int，char以外，所有的基本数据类型包装类都是将基本数据类型的关键字首字母大写

int------Integer

char-----Character

## 基本数据类型转换成包装类

1. 通过对应的基本数据类型的包装类的构造函数-----已过时
2. 通过对应的基本数据类型的包装类的------\*\*\*\*\*\*\*

public static 包装类 valueOf(相应基本数据类型)；

## 包装类转换成基本数据类型

public 基本数据类型 基本数据类型Value()比如：[intValue](mk:@MSITStore:C:\Users\Zicheng%20Qu\Desktop\Programming%20Software\乾包\jdkapi18chm\jdk%20api%201.8.CHM::/java/lang/Integer.html#intValue--)();

## 包装类（基本数据类型）转换String类

public String toString()

## String转换成包装类（基本数据类型）

1. 通过包装类的构造函数-----已过时
2. public static 包装类 valueOf(String str);

前提：字符串的值必须是相对应的基本数据类型时，才可以转化成对应基本数据类型的包装类

注意：字符串不能转换成char的包装类Character

## 自动拆、装箱-----jdk1.5版本后

装箱：将基本数据数据类型转换成包装类

拆箱：将包装类转换成基本数据类型

注意：仅指基本数据类型与对应包装类之间的相互转换

在[-128,127]之间，通过int直接转换成integer类型；在范围之外，通过new Integer()去转换。

# String类:

### public char charAt(int index)

普通方法：返回指定字符串的指定索引位置的字符

普通方法：第一个不同的字符的ASCII码的差值。。如果调用字符串与参数字符串长度相同并且值相等，则返回0；如果值不相等，则返回第一次出现不同字母的ASCII码的差值。如果（短的字符串是长的开头的一部分）长度不同，但大部分的值相同，则返回长度差。

### public int compareTo(String str)

构造方法：通过使用平台的默认字符集解码指定的字节子阵列来构造新的 String:public [String](mk:@MSITStore:C:\Users\Zicheng%20Qu\Desktop\Programming%20Software\乾包\jdkapi18chm\jdk%20api%201.8.CHM::/java/lang/String.html#String-byte:A-int-int-)(byte[] bytes, int offset, int length)

构造方法：分配一个新的 String ，其中包含字符数组参数的子阵列中的字符: public [String](mk:@MSITStore:C:\Users\Zicheng%20Qu\Desktop\Programming%20Software\乾包\jdkapi18chm\jdk%20api%201.8.CHM::/java/lang/String.html#String-char:A-int-int-)(char[] value, int offset, int count)

判定指定的文件名是否是按照指定的格式结尾的（在io流中经常使用）:

### public boolean endsWith(String suffix)

返回一个字符串中第一次出现该字符的下标(如果没有返回-1):

### public int indexOf(int ch)

### public int indexOf(int ch, int fromIndex)

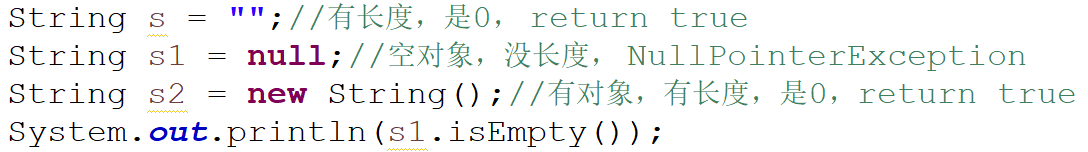
返回一个字符串中第一次出现该子字符串的第一个字符(原字符串)的下标(如果没有返回-1):

### public int indexOf(String str)

### public int indexOf(String str, int fromIndex)

若字符串的长度为0，则返回true。

### public boolean isEmpty()



截取一段字符串

### public String substring(int beginIndex), from begin to end

### public String substring(int beginIndex, int endIndex), [beginIndex, endIndex)

# 正则表达式

1. 含义：就是给获取到的字符串，规定一个格式
2. 常用正则表达式符号

[]-------表示一个位置，如果该位置上只有一种可能，可以忽略不写

()-------表示一个组合（至少是两位，并且是连续的）,例如：手机号开头：(\+86)

{}-------用来定义字符串长度的范围

{n}------正好出现n次

{n,}-----出现的次数是[n,正无穷）

{n,m}----出现的次数在[n,m]范围内,例如:qq号:String qqFormat = “[1-9]\\d{4,9}”;

|--------按位或；表示或者

^--------定义在括号外：行起始符 定义在括号内：非，即取反

$--------行结束符

+--------表示字符至少出现一次（即>=1）

注意：如果用\表示特殊字符，Java必须书写\\

1. public boolean matches(String regex)判断指定字符串是否符合给定的正则表达式（regex）

public String[] split(String regex)指定字符串根据给定正则表达式去分隔

从控制台录入一个手机号，满足以下要求输出手机号

1. 首个字母只能是1
2. 第二位可能是3578
3. 长度只能是11位
4. 不能出现特殊符号

1[3|5|7|8]\\d{9}

# StringBuffer、 StringBuilder

StringBuffer、 StringBuilder是String的增强类

String一旦声明，长度固定，换任何内容，都会改变其内存地址。

StringBuffer和StringBulider：在步改变对象的内存地址的前提下，对String的值的内容进行改变。

以StringBuffer为例：

主要构造方法：

StringBuffer(),StringBuffer(String str)

append（Object obj）：在结尾追加

insert（int index, Object obj）：在指定位置插入元素

delete（int beginIndex, int endIndex）：删除指定范围[b,e)的所有元素

deleteCharAt（int index）：删除指定位置上的元素。

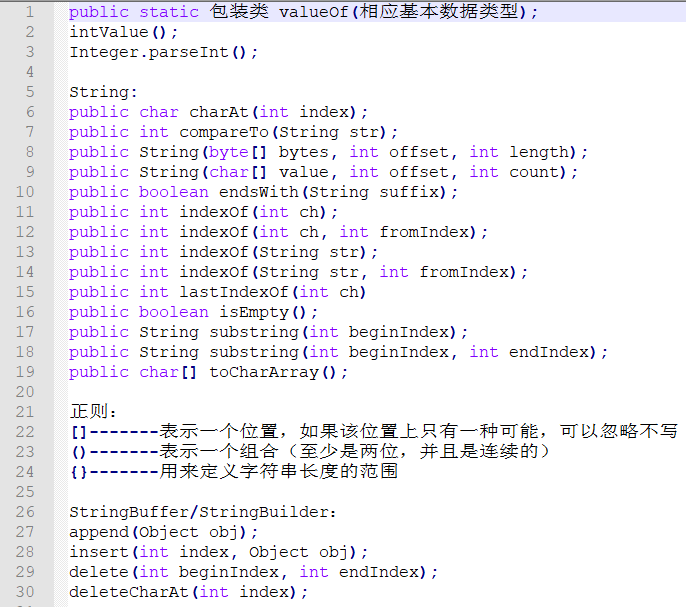
Buffer与Builder的异同点：

相同点：都是String的增强类

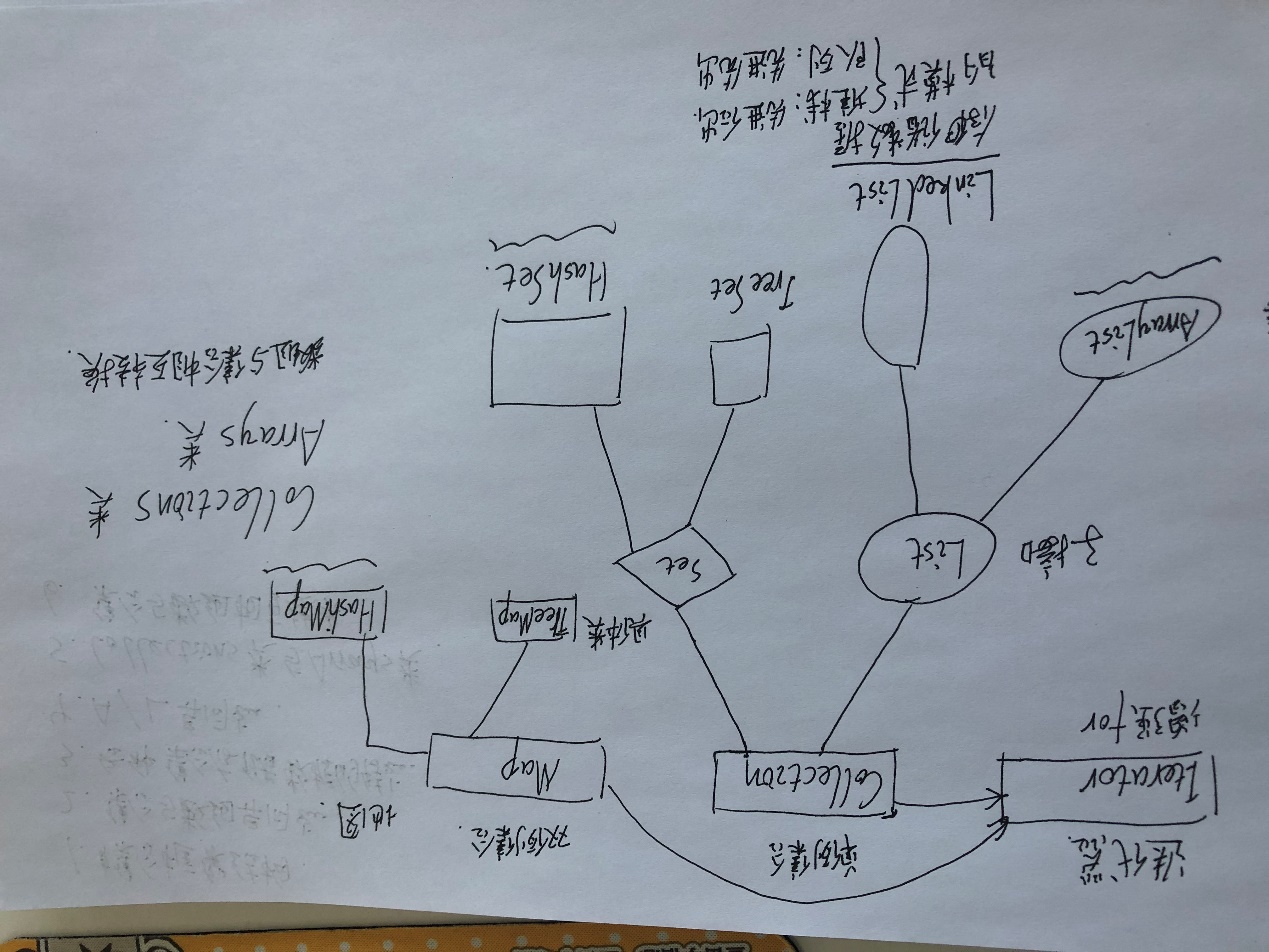
不同点：在多线程Buffer更安全，完成时用Buffer

在单线程用Builder更快捷，用来测试

StringBuffer<=>String<=>StringBuilder



# 集合



* + 1. 集合框架结构
    2. 集合与数组异同点

相同点：都可以存储多个数据

不同点：数组=>任意的相同数据类型，长度不可变

集合=>只能存放引用数据类型，长度可变

* + 1. 各种集合存储数据的特点

1. 单列集合：

Collection：既能有序，也可无序

既能唯一，也能不唯一

List：有序不唯一

ArrayList:基于数组存储，“纵向”存储，便于查找和修改，不便于增删

LinkedList：基于链表存储，“横向”存储，便于增删，相对ArrayList来说，不便于查找和修改，内存占用较多。

Set：无序唯一

HashSet：基于哈希码值存储数据

1. 双列集合：

Map：K-V，键值对形式存储，具体类：HashMap

HashMap：键值对关系：一一对应，K无序唯一（返回Set），V可重复（返回Collection，因为是不唯一的，K又是无序的，所以只有Collection满足无须不唯一的条件），K与V都可以存储null（无实际意义）。

* + 1. ArrayList/LinkedList异同点

1. 常用方法：

boolean add(Object obj),末尾追加元素，返回true/false

void add(int index, Object obj),指定位置插入元素（注意：指定位置，index>=0&&index<=size();

int size()，返回集合元素的个数。

void clear(),清空集合的所有元素，但集合的实例对象仍然存在

boolean isEmpty(), 当且仅当size()为0时，返回true。

boolean contains(),如果此列表包含指定的元素，则返回 true 。

E get(int index),返回指定位置的元素（只能用于List中）。

E set(int index, E element), 用指定的元素替换此列表中指定位置的元素，并返回替换之前的元素（只能用于List中）

。

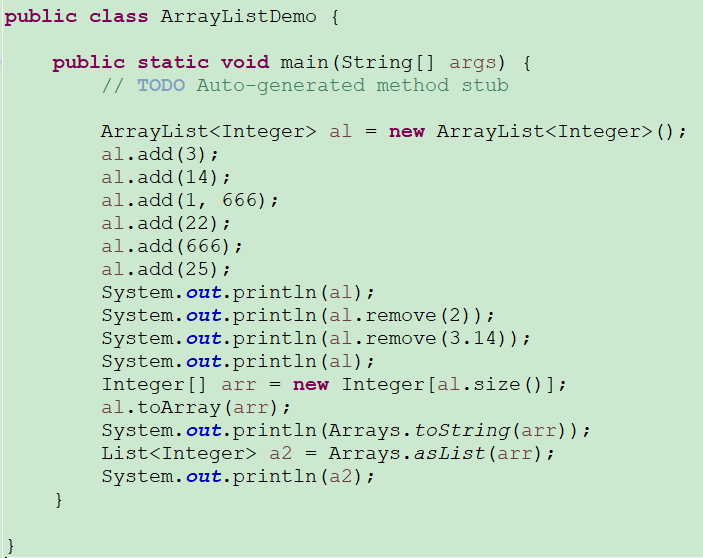
E remove(int index): 删除该列表中指定位置的元素,返回被删除的E且默认使用。

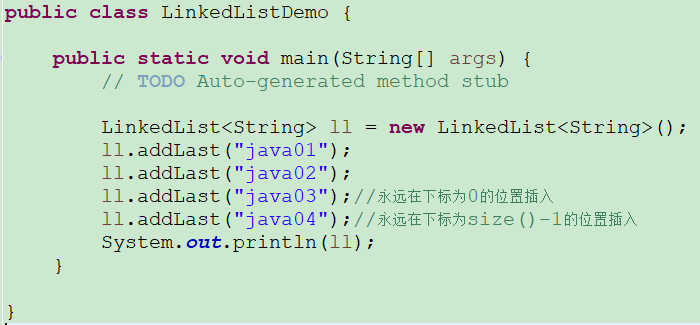
boolean remove(Object obj):删除该列表中指定的（出现的第一个）元素，若删除则返回true，无则返回false。

（如果指定元素恰巧是Integer，则需要通过Integer.valueOf(int num); 如果指定元素Character，则需要通过Character.valueOf(char ch);）

void addFirst(E e),在该列表开头插入指定的元素,只用在LinkedList中，模仿堆栈。

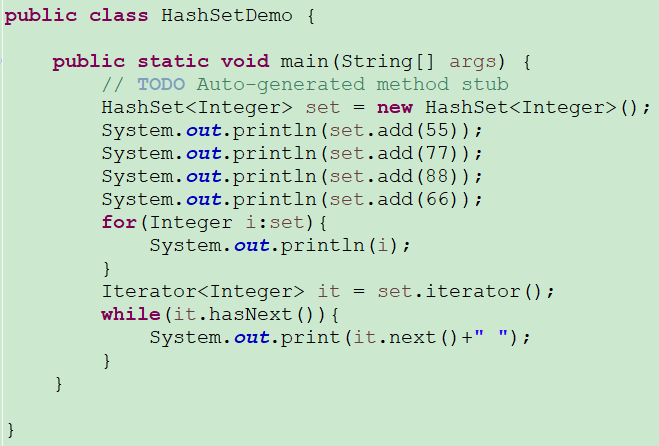
void addLast(E e),将指定的元素追加到此列表的末尾，只用在LinkedList中，模仿队列。





* + 1. Set(参数带index或返回值带index的方法都不存在，因为无序)：

boolean add（E e），将指定的元素添加到此集合（如果尚未存在,则添加并返回true，否则返回false）。



* + 1. Hashmap:

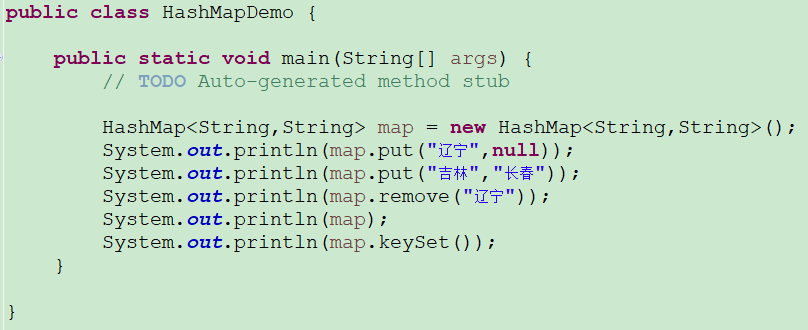
V put(K k, V v),将指定的值与此映射中的指定键相关联,若K不存在则返回null，若k存在，则用新v替换旧v，并返回旧v。

Set<K> keySet(), 返回此地图中包含的键的[Set](mk:@MSITStore:C:\Users\Zicheng%20Qu\Desktop\Programming%20Software\乾包\jdkapi18chm\jdk%20api%201.8.CHM::/java/util/Set.html)视图。

V remove(K k), 从该地图中删除指定键的映射（如果存在，则删除此键值对，并返回被删除的V;如果不存在则返回null）。

boolean remove(K k, V v), 仅当指定的密钥当前映射到指定的值时删除该条目,并返回true，若不匹配则返回false。

Collection<V> values(), 返回此地图中包含的值的[Collection](mk:@MSITStore:C:\Users\Zicheng%20Qu\Desktop\Programming%20Software\乾包\jdkapi18chm\jdk%20api%201.8.CHM::/java/util/Collection.html)视图。



* + 1. Iterator(迭代器)：

boolean hasNext(),判断集合中是否还有元素。

E next(),获取该元素。

增强for和Iterator都是取数据的，for更适合list，Iterator更适合set（因为无序）。往set里添加指定个数的元素时要用while(size()<???)

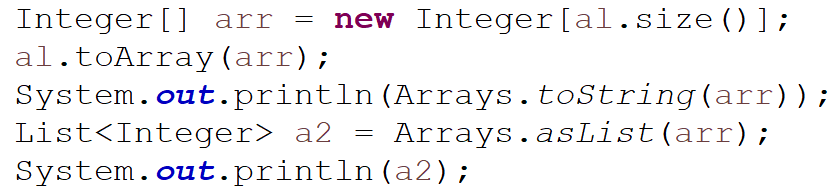
* + 1. 集合与数组相互转换，Collections类与Arrays类：

静态方法，Collections.sort(List<T> list),升序，无返回值。

静态方法，Array.asList(T... a)，返回由指定数组支持的固定大小的列表。

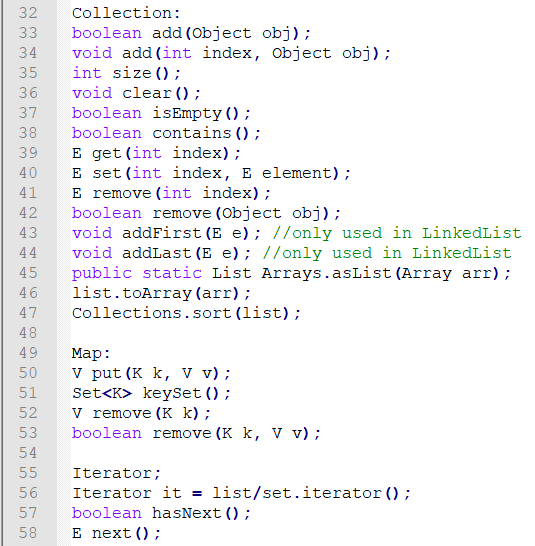
Array -> List, asList(),存储灵活，测试代码；

List -> Array, toArray (T[] a),安全性更高，运维使用。



* + 1. 泛型

1. 定义：规范集合的数据类型(注意：泛型也时数据类型 ，并且只能引用数据类型)
2. 格式：集合<泛型> 集合名 = new 集合<泛型>();
3. 优势：a.增强集合存储数据的安全性。 b.等效于数组（只有集合使用泛型的时候）。



# 异常

1. 含义：代码运行时，并没有得到想要的结果。
2. 分类：运行时异常，非运行时异常，检测性异常，非检测性异常。
3. 异常类结构（在java中只要想成为异常，就必须继承Exception:

Exception->RuntimeException->ArrayIndexOutOfBoundsException

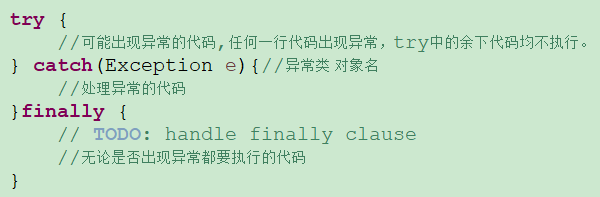
1. 构造方法：public Exception(String msg),通过指定字符串解释该异常。
2. 异常处理机制（5个关键字）：

方法定义时使用：

throw：抛出单个异常，定义在方法体内。

Throws：抛出多个异常，定义在方法参数列表后，多个异常用“，”分割。

方法调用时，在主函数中使用：



注意：

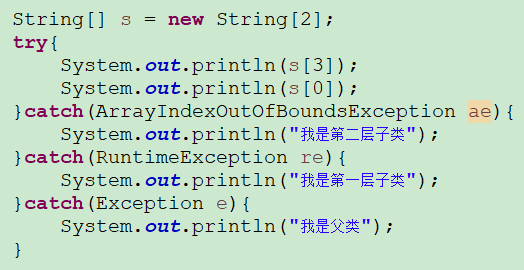
(1) 一旦出现异常代码，其后代码都不执行。

(2) try与catch，catch与finally之间不能书写任何代码。

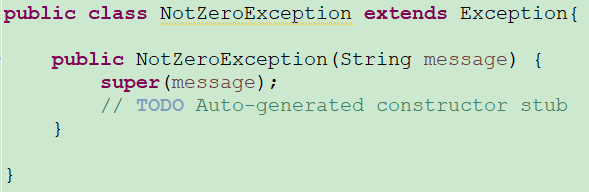
(3)try可以单独和catch组合，也可以单独和finally组合，但catch不能和finally单独组合。

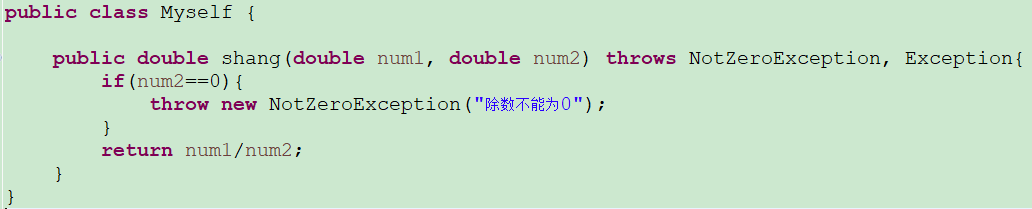
(4)throw和throws只抛出异常，并不对异常进行处理，因此只有方法定义时使用。

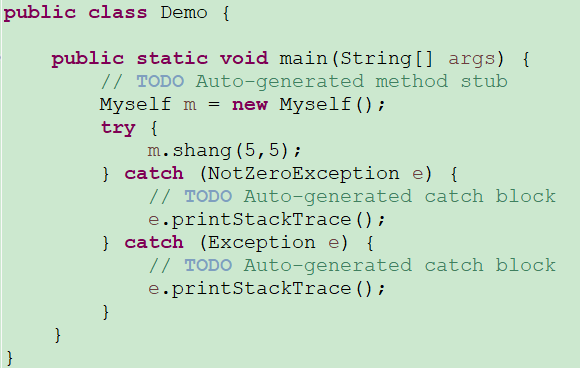
(5)catch可以叠加，但定义顺序是越具体的子类先定义，并且只执行与该异常最贴近的处理。



例子：







# 文件与流（IO流）：

1. 文件类（File类）：程序员为了通过代码来操作电脑的文件，该类封装多个对文件操作的方法。
2. IO流：Input/Output => 操作文件的对象
3. 文件类：File => 路径
4. 路径分类：相对路径（无盘符）绝对路径（有盘符））。
5. public File(String url): 只是创建了一个File类的对象，并不能直接在指定的url下创建实体文件。
6. public boolean createNewFile() throws IOException:当且仅当创建文件的路径真实有效的情况下，且指定的路径下该文件不存在，则创建该文件并return true。否则，不会创建并return false。异常为指定盘符不存在。

如果url为相对路径，则在项目所在文件夹创建；若为绝对路径，则在指定路径创建。

1. public boolean equals(File f):当且仅当参数对象不为null，且路径有效并相同，对象类型相同时，返回true，否则返回false。
2. public boolean delete(): 当且仅当了路径有效，且文件已被创建，返回true，否则返回false。
3. public boolean exists():当且仅当路径有效且文件存在，返回true。
4. public String getAbsolutePath():返回一个String，如果后续操作的是路径。
5. public File getAbsoluteFile():返回一个File，如果后续操作的是文件。
6. public String getName():返回一个指定文件的字符串型式的文件名。
7. public boolean isDirectory():判定指定路径是否是文件夹，如果是则返回true。
8. public File[] listFiles():返回指定路径下的所有文件。
9. public String[] list():返回指定路径下的所有文件名。
10. IO流（Stream）:

字节：

InputStream（FileInputStream，BufferedInputStream）/

OutputStream（File OutputStream，Buffered OutputStream）。

字符：

Reader（FileReader/BufferedReader）/

Writer（FileWriter/BufferedWriter）

字符流：

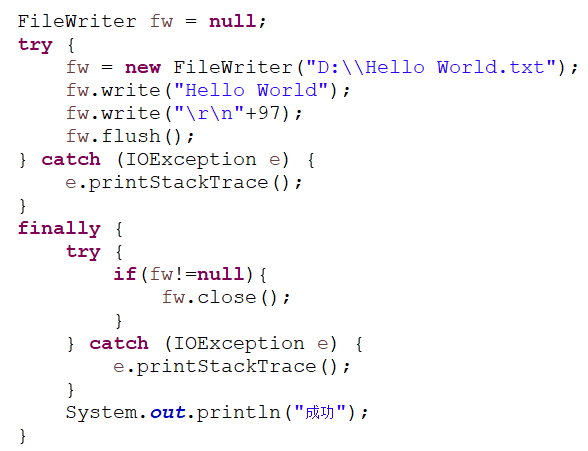
Writer -> FileWriter:

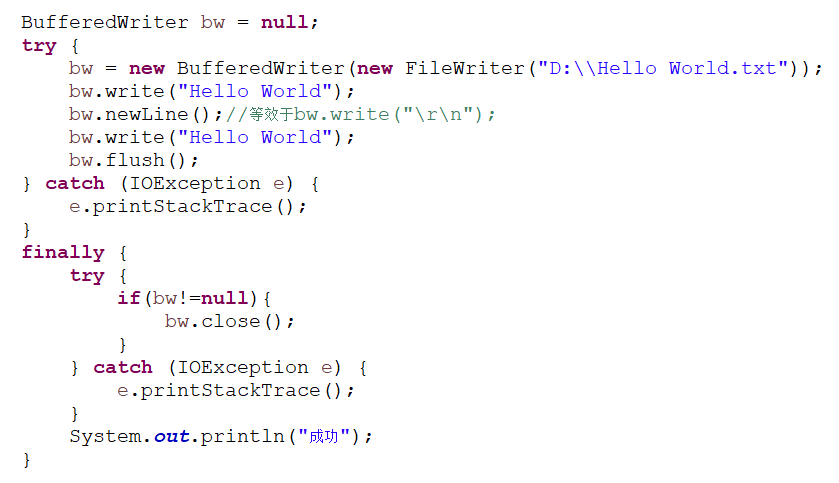
public FileWriter(String url) throws IOException,如果指定url下不存在文件则创建，如果存在则覆盖。

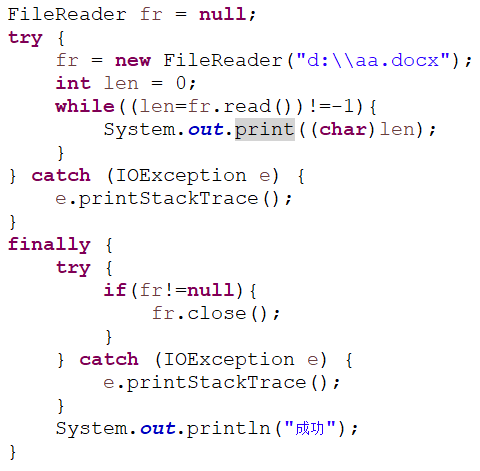
public void write(int c/ char[] buf/ String str) => 将参数写道流中，并没有直接到文件中。（Windows中换行是\r\n）

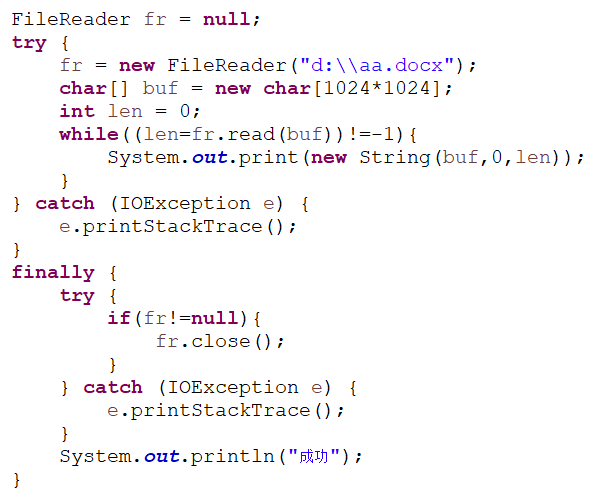
public void flush() => 将流中的数据刷新到指定文件中。

public void close() => 关闭流资源，但先刷新。一定要关闭流









# 其它补充的：

# Java高级---文件与流

## 文件类

1、出现的原因：为了方便程序员对文件进行一系列的操作

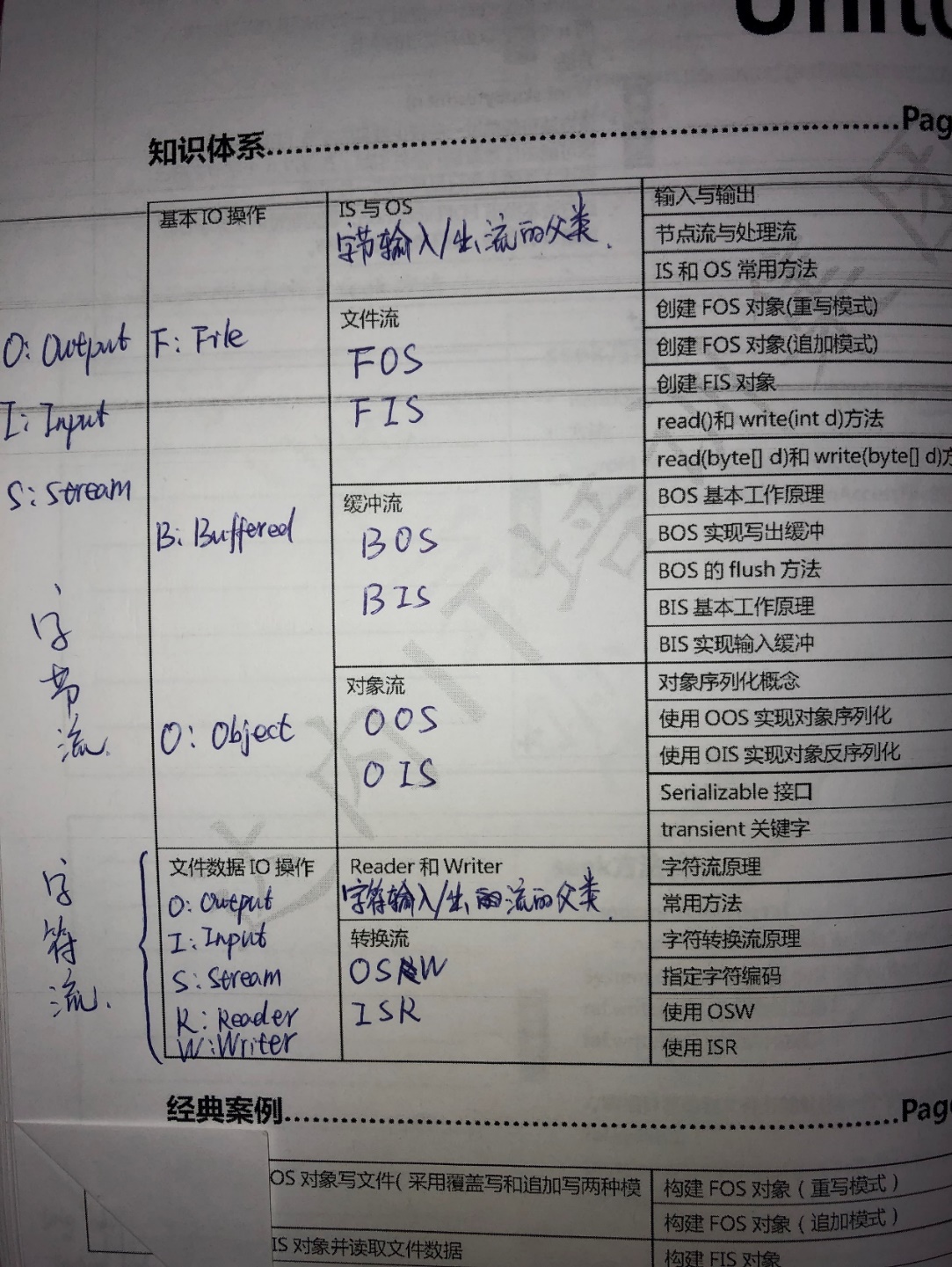
2、分类

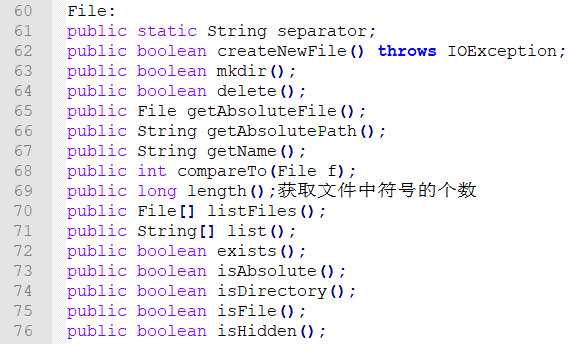
计算机中所有的文件都是字节文件

人为的将文件进行如下分类，便于程序操作

文本文档文件：字符文件，如word文档，txt文档

媒体文件：字节文件，exe文件





## File类：

常用类成员

public File(String url)

在指定的路径下实例化一个文件类的对象，并没有创建实体文件

计算机中路径的分类-----区分：看所写路径带不带盘符

绝对路径

相对路径

public static String separator

根据系统显示的路径分隔符（提现了Java语言的跨平台性）

public boolean createNewFile() throws IOException

在指定路径下创建实体文件

如果指定路径下没有该文件，则创建，返回true

如果指定路径下已存在该文件，则不创建，返回false

如果指定路径不存在，则抛出异常

如果是绝对路径，在指定盘符下创建

如果是相对路径，在项目所在的文件夹下创建

常用方法：

增：

public boolean mkdir()

在制定路径下创建文件夹

如果存在则不创建，返回false

如果不存在则创建，返回true

删：

public boolean delete()

删除指定文件（如果存在）----实际开发过程中常用

public void deleteOnExit()

虚拟机结束时，删除文件

区别：

deleteOnExit()会造成“垃圾”文件在执行内存的滞留，

造成运行速度降低；但是由于是虚拟机结束后删除，删除彻底

有时会同时使用delete()和deleteOnExit()，一般会将deleteOnExit()

写在finally块中

改：

public File getAbsoluteFile()

获取指定文件的绝对路径，返回一个文件对象的形式

public String getAbsolutePath()

获取指定文件的绝对路径，返回一个字符串的形式

区别：

如果接下来的操作依然是对文件进行，使用getAbsoluteFile()

如果接下来的操作是对路径进行的，使用getAbsolutePath()

public String getName()

获取指定文件的文件名

查：

public int compareTo(File f)

返回指定文件名与参数文件名的阿斯克码差值

如果都是绝对路径，并且内容完全一致，返回0

如果都是绝对路径，但是内容不一致，

返回第一次出现不同字符的阿斯克码差值

如果都是相对路径，并且内容完全一致，返回0

如果都是相对路径，但是内容不一致

返回第一次出现不同字符的阿斯克码差值

如果是相对路径与绝对路径比较，返回第一次出现不同字符的阿斯克码差值

public long length()

获取文件中符号的个数

public File[] listFiles()

返回指定路径下的所有文件的绝对路径，以一个File类数组的形式

public String[] list()

返回指定路径下的所有文件的文件名，以一个字符串数组的形式

判：

public boolean exists()

判定指定路径下文件是否存在

public boolean isAbsolute()

判定指定文件是否是绝对路径

public boolean isDirectory()

判定指定的文件是否是文件夹

public boolean isFile()

判定指定的文件是否是标准文件

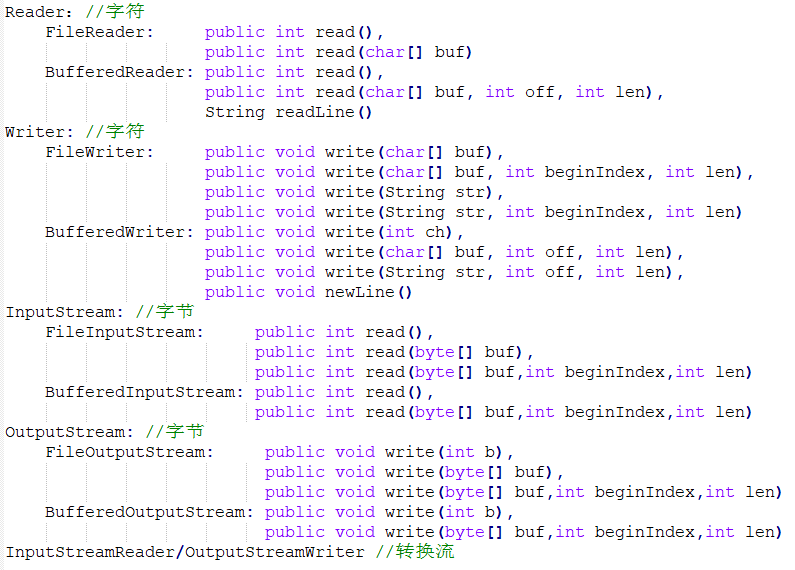
（标准文件：由Java虚拟机创建的，符合相应操作系统标准的文件）

当且仅当指定的文件存在，并且是标准文件时，返回true

public boolean isHidden()

判定指定文件是否是隐藏文件

## IO流



1、什么是流？

生活中

如果是名词：河流。水流

如果是动词：流向

起到“搬运”的作用

Java中

操作计算机中所有文件的工具

2、IO流----InputStream/OutputStream

Java中的流----四大基类（抽象类），八大子类\*\*\*\*\*\*\*\*\*

流的分类

流向：输入和输出

按操作数据形式：

字符流

Reader

FileReader/BufferedReader

Writer

FileWriter/BufferedWriter

字节流

InputStream

FileInputStream/BufferedInputStream

OutputStream

FileOutputStream/BufferedOutputStream

3、关于Java中流的命名规则

前缀是解释该流的作用

后缀表示该流是操作哪种数据资源的流

### 字符流

1、字符流中的写入流----Writer

#### FileWriter类

常用方法：

public FileWriter(String url) throws IOException

在指定路径下创建文件

如果是绝对路径：在指定盘符下创建

如果是相对路径：在项目所在文件夹下创建

可能抛出指定盘符不存在异常

如果指定路径下没有该文件，就创建

如果有在创建的同时，将原有的覆盖掉

public Writer append(char ch)

向指定文件中的末尾出追加字符（必须是字符型数据）

public void write(char[] ch)

public void write(char[] ch,int beginIndex,int len)

将字符数组的的一部分写入到流资源中

beginIndex----从数组中那个元素下标开始写

len-----将数组中的几个元素写入流资源中

public void write(String str)

public void write(String str,int beginIndex,int len)

注意：

以上方法并不是将数据直接写入目的地文件，而是缓冲在流资源中

public void flush()

将流资源中的缓冲刷新到目的地文件中

public void close() throws IOException

关闭流资源，但在关闭之前进行刷新

close()与flush()异同点

相同点：都能将流资源中缓冲的数据刷新到目的地文件中

不同点：

flush()：时时刷新，或同步刷新，

刷新后，流资源依然可以正常运行

close()：异步刷新，刷新后，流资源关闭

流操作文件书写格式：

1、在try块外声明流资源类对象，并赋初值为null

2、对象实例化在try快内，并将所有操作文件的方法写在try块中

3、必须关闭流资源，将close()方法书写在finally块中，

在调用close()之前，必须先判定已生成流资源对象是否为null

字符流

Reader---字符流中的读取流

#### FileReader类

常用方法：

public FileReader(String url) throws FileNotFoundException

在指定路径下寻找文件

如果文件不存在抛出文件找不到异常

读数据的方法

public int read()

public int read(char[] buf)

注意：

以上两个读取的方法，如果读到文件结尾，返回-1

如何定义数组？

格式：char[] buf = new char[1024的整数倍];

如何将读到的一组数据打印在控制台？

格式：System.out.print(new String(buf,0,len));

public void close() throws IOException

关闭读取流资源

通过分析：无论使用哪种流资源操作文件，无论调用的是read()

还是read(char[] buf)；无论调用的是write(char ch)

还是write(char[] buf)；实际上调用的都是计算机底层

的read()或write()

通过文件的赋值粘贴代码----分析

1）无论是创建文件，还是读取指定文件，必须明确定义出文件的文件格式

2）对于多个操作文件的流资源，需要分别关闭

#### BufferedWriter类

Writer的子类：字符写入缓存流

常用方法：

public BufferedWriter(Writer out)

实例化BufferedWriter类对象

由于Writer类是抽象，因此只能通过实例化具体子类（FileWriter）传递参数

该构造函数不抛异常，实际上抛出异常的是FileWriter类

创建文件参考FileWriter类构造函数

write()/flush()/close()参照FileWriter类

public void newLine()

等效于write("\r\n")，换行

#### BufferedReader类

Reader类的子类：字符流的读取缓存流

常用方法：

public BufferedReader(Reader in)

实例化BufferedReader类对象

由于Reader类是抽象，因此只能通过实例化具体子类（FileReader）传递参数

该构造函数不抛异常，实际上抛出异常的是FileReader类

read()/close()参照FileReader类

public String readLine()

一行行读取文件

装饰设计模式：

1、什么是装饰设计模式？

对已有类进行功能性上的增强

2、Java继承

1）只支持单继承

2）子类可以直接调用父类的所有类成员

父类不能直接调用子类的特有类成员

3）方法特性---方法的重写（override）

子类对父类已有方法进行功能性上的增强

3、装饰设计模式与继承的异同点

相同点：都可以对类中的方法进行功能增强

不同点：方法重写，类与类之间必须存在继承关系，换而言之，类和类之间

联系紧密

装饰设计模式，类与类之间不一定存在继承关系，降低了类与类

之间的联系，降低耦合度

LineNumberReader类------BufferedReader类的子类：提供行号的字符读取流

常用方法：

public LineNumberReader(Reader in)

实例化一个LineNumberReader类的对象

由于Reader类是抽象，因此只能通过实例化具体子类（FileReader）传递参数

该构造函数不抛异常，实际上抛出异常的是FileReader类

public int getLineNumber()

获取行号

public void setLineNumber(int num)

用参数给行号符初始值

注意：如果用setLineNumber(int num)给行号赋初始值

那么getLineNumber()是从num++开始

### 字节流

万能流，可以操作计算机中的任何类型文件

#### OutputStream

字节流中的写流，是基类

FileOutputStream

常用方法：

public FileOutputStream(File file) throws FileNotFoundException

实例化字节写入流对象

如果指定路径下不存在，则创建

如果指定路径下存在，则覆盖

抛FileNotFoundException的原因：

1）指定的路径是文件夹，并不是文件

2）该文件不存在，并且在计算机中无法创建该格式文件

3）该文件存在，但是其格式并不是计算机支持的常规格式

public void write(int b)

public void write(byte[] buf)

public void write(byte[] buf,int beginIndex,int len)

以上三个方法是字节写入流中的写方法，

注意：参数一定是字节型

将输入的字符串转换成字节数组的方法，书写格式

"".getBytes();（参照String类中的getBytes()）

public void close() throws IOException

关闭流资源的方法

#### BufferedOutputStream

字节写入流的高效流

常用方法：

public BufferedOutputStream(OutputStream out)

实例化一个BufferedOutputStream类的对象

由于OutputStream类是抽象的，因此只能通过实现

具体子类（FileOutputStream）进行参数传递

注意：在BufferedOutputStream类中的所有方法使用，

参考FileOutputStream类

BufferedOutputStream类与FileOutputStream类异同点

相同点：都是字节写入流OutputStream的子流

不同点：

在底层代码调用（其实就是调用write()）上，

BufferedOutputStream类要高效与FileOutputStream类

#### InputStream

字节流中的读取流，是基类

FileInputStream

常用方法：

public FileInputStream(String url) throws FileNotFoundException

实例化一个FileInputStream类的对象

public int read()

public int read(byte[] buf)

public int read(byte[] buf,int beginIndex,int len)

以上三个方法的使用参照字符流中的相应方法

注意：只是将字符流中的字符数组换成字节数组

public void close() throws IOException

关闭流资源的方法

#### BufferedInputStream

常用方法：

public BufferedInputStream(InputStream in)

实例化BufferedInputStream类一个对象

由于InputStream类是抽象的，只能通过

实例化具体子类（FileInputStream）传递参数

其余方法参照FileInputStream类

BufferedInputStream类与FileInputStream类异同点

相同点：都是字节读取流的子流

不同点：BufferedInputStream类在实例化对象的同时，会在计算机底层

直接创建一个字节缓冲数组，相比于FileInputStream类更高效

## 关于文件与流的总结

1、流的分类

字符流中的输入流：Reader

字符流中的输出流：Writer

字节流中的输入流：InputStream

字节流中的输出流：OutputStream

2、关于Buffered所有流资源类

如果是字符流中的Buffered流资源类是在继承Reader/Writer的基础上

提供了更高效的方法。换而言之，即使用了继承，同时，也使用了

装饰设计模式

如果是字节流中的Buffered流资源类仅仅是实现类继承，并没有提供高效

方法；只不过是在底层调用时，更加简便

3、同过File类与流资源类创建文件的区别

1）如果是File类，通过构造函数只能实例化该类对象，并不能直接创建

实体文件，需调用createNewFile()，如果文件存在，则不创建；否则

创建

2）如果是流资源类，在通过构造函数实例化本类对象的同时，创建了

实体文件；如果文件存在，进行覆盖，否则，创建

4、write()写一个的方法

如果是字符流：char ch

如果是字节流：int b

5、read()读一个的方法在字符流与字节流中的异同点

相同点：返回值都是int，并且读到文件末尾返回-1

不同点：在字节流中虽然表面上返回int，但是在内存结构上看属于byte

因此，不能直接进行char转换（1char=2byte）

6、关于流资源类操作文件的格式：

1）将所有流资源类声明的对象写在try块外，并赋值为null

2）将实例化对象的过程写在try块内，并同时编写所有操作文件的方法

3）在finally块中关闭所有流资源，同时，判定该流资源是否是空对象

并且必须分别关闭

# Java高级---多线程

（Java语言的四大特点之一，是一门技术应用手段）

## 进程和线程的区别：

进程：指计算机中一个正在运行的应用程序（当前可以直接操控的）

多进程：指计算机中所有正在运行的应用程序

线程：是每个进程的最基本控制单元

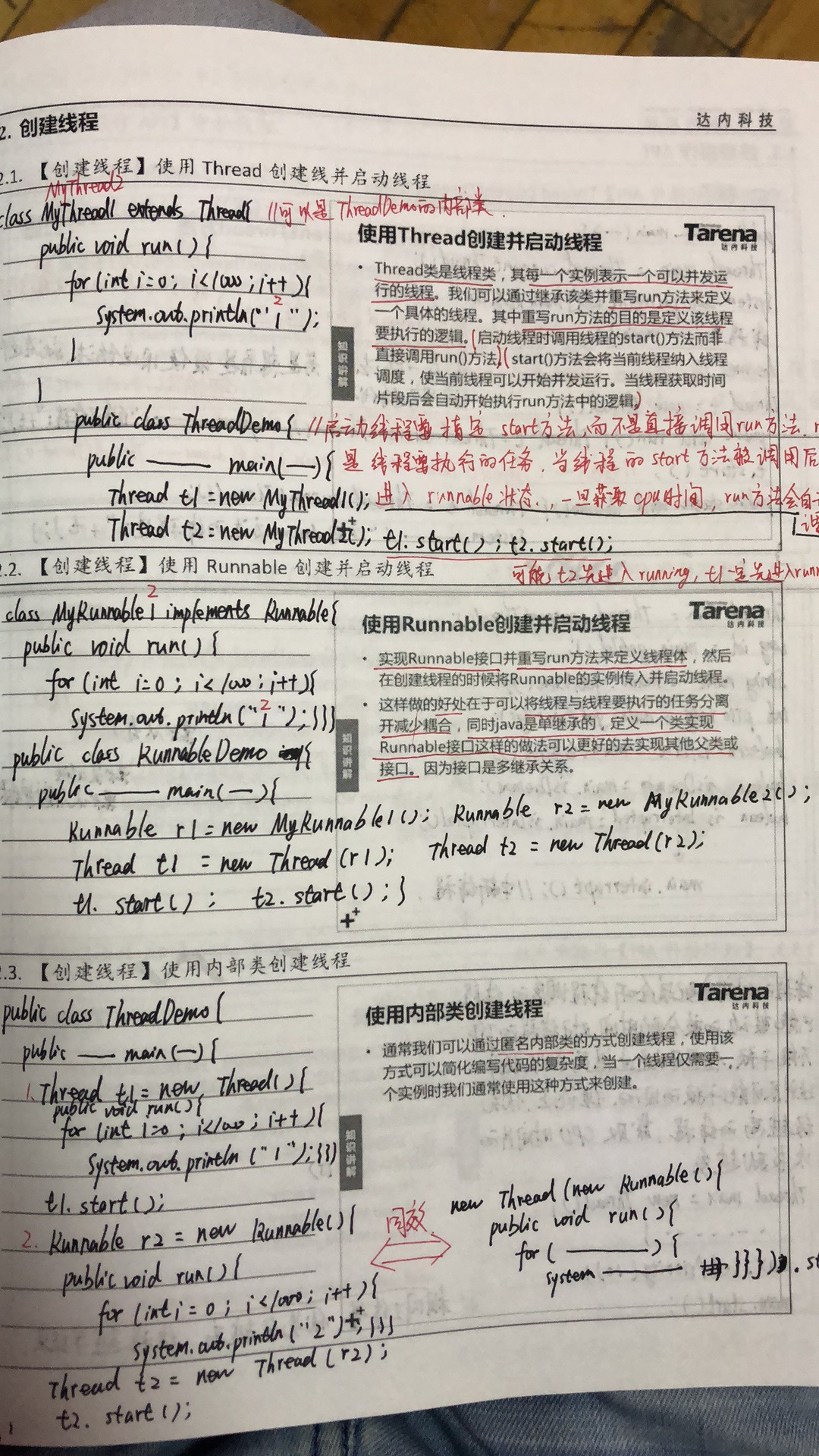
多线程：在同一个进程，在同一时刻执行多个事务

优势：提高软件的工作效率

## 特性：

Java是一门多线程软件编程语言，在执行过程中，至少会执行两个线程：一是主线程（这是必须执行的）；二是垃圾回收处理机制（不定期，但是可以保证内存执行空间足够大）

## Java中如何创建线程？



### 继承Thread类

1.创建一个类继承Thread类

2.重写Thread类中的run()

3.开启多线程：调用start()

4.注意：

A.必须重写run()---存放真正执行多线程的代码

B.如果仅调用run()实现的依然是单线程

C.必须调用start()开启多线程----在调用start()的同时，

会调用Thread类子类中重写的run()，这也就满足了多线程

的操作

D.通过多次运行多线程代码，发现每次运行结果并不一致

为什么会不一致？通过分析，程序运行是由CPU决定的；

CPU在同一时刻只能执行一个线程，是在执行内存上进行

多个线程快速的切换，达到同时执行的效果；并且保证

多个线程都是存活的

E.我们形象称多个线程，去“抢占”CPU的执行权，谁抢到谁执行

通过CPU去执行操作的特点是----随机性

F.正在执行的线程处在运行状态下；而没有执行的线程处于

临时（阻塞）状态下。在临时（阻塞）状态的线程存在于

Java虚拟机提供的线程池中（特点：先进先出）。当正在

执行的线程结束，另一个线程启动时，是不确定的；因为

需要由CPU（特点：随机性）去决定

5.常用方法：

public void run()

子类中真正存放多线程代码的方法，必须重写

public void start()

真正开启多线成的方法

public String getName()

获取当前正在执行的线程名

public static Thread currentThread()

获取当前正在执行的线程

注意：

1）主线程名为----main

2）如果没有明确定义线程名，Java虚拟机会提供一个

默认的名字，格式：Thread-num；num是从0开始的

3）如果类是Thread类的子类，可以直接调用getName()

省略super/this/currentThread()

4）如果不是Thread类的子类，只能通过

Thread.currentThread().getName()

获取当前正在执行的线程名

public static void sleep(long m)

使正在执行的线程“睡眠”m毫秒

### 实现Runnable接口---建议使用

1.创建一个类实现Runnable接口

2.重写Runnable接口当中的run()

3.通过public Thread(Runnable r)将实现Runnable接口的类与

Thread类进行关联，来调用Thread类中的start()

4.使用Runnable接口的好处：避免了Java中单继承的局限性

5.通过买票系统分析，一定会出现0，-1等错票，

这就存在着安全隐患问题！如何解决安全隐患问题？

分析问题：

1.是否操作的是同一数据资源-----根本问题

2.是否在同一时刻，同时调用一个以上方法

Java中提供了专业的解决方法----同步

1.同步代码块

格式：

synchronized(任意类的对象---Object类对象){

同步代码;（真正操作资源的代码）

}

2.同步函数

格式：

public synchronized void 方法名(){

同步代码;（真正操作资源的代码）

}

3.根据分析，同步代码块与同步函数持有的并不是同一把锁

同步代码块持有的锁是任意对象，而同步函数持有的锁是

当前对象，即this；因此，在同时使用同步函数和同步代

码块时，需使用同一把锁

# 网络编程

1. Java中的网络编程主要是Socket编程，事实上就是通过java代码和网络实现多台电脑的数据交互。
2. 需要使用的包：java.io.\* 和 java.net.\*
3. 网络编程三要素（重点）：
   1. 获取IP地址

方式1：查看高级属性

方式2：命令窗口（Windows+R，cmd，ipconfig）

IP地址的形式（v4/v6）：v6防止v4被用尽。

v4: 四段，0-255的数字。

v6: 六段，数字和字母组成。

作用：作为本计算机的唯一标识。

了解：127.0.0.1本地回环地址。

Localhost 本地主机名称。

* 1. 指定应用程序：

将数据从本地计算机的指定应用程序上发送到对方计算机的指定应用程序上。

为了识别不同的应用程序就给这些网络应用程序都用数字进行表示，这些数字成为端口号。

常用端口号：MySql 3306， Oracle 1521， tomcat 8080。

注意：有效端口号 0—65535之间。其中0-1024为系统使用或预留端口号

* 1. 遵从传输协议：

含义：定义了通讯规则

网络参考模型： OSI、TCP/IP

国际通用：TCP/IP

方式：

TCP协议：面向连接（类似于语音通话、打电话）

* + - * 1. 建立连接，形成数据传输通道
        2. 可以进行大量的数据传输
        3. 通过连接进行完成，协议可靠
        4. 必须建立连接，所以效率稍低

UDP协议：面向无连接（类似于发送语音消息）

1. 将数据和目的封装在数据包中，不需要建立连接，直接发送。
2. 每个数据包的大小是有限制在64k之内。
3. 因为无连接，所以协议不可靠，效率稍快。
4. 常用方法：
5. 发送端流程（UDP）
6. 创建DatagramSocket对象，建立udpsocket服务。
7. 创建字符数组，来接受控制台录入的数据。使用getBytes()可以将读到的字符串转为byte数组。
8. 创建DatagramPacket对象，将数组中的数据放在数据包中。
9. 使用DatagramSocket发送包数据。
10. tcp案例练习（转换大小写）

客户端：Socket类

1. 实例化Socket类对象，指定主机和端口号
2. 发送数据：控制台数据解读（输入），将解读的数据写入Socket对象中（输出）
3. 关闭资源。

服务端：ServerSocket类

1. 实例化ServerSocket类对象，指定监听的端口号。
2. 获取Socket对象
3. 获取数据，处理数据
4. 关闭资源