DOS命令行常用操作

小写盘符:——打开对应的盘

dir——列出当前目录下的文件以及文件夹

cd——打开文件夹

cd..——返回上一级

cd\——返回根目录

md——创建文件夹

rd——删除文件夹

del——删除文件

exit——退出命令行

JAVA跨平台性

只需要安装一个JVM（JAVA Virtual Machine）就可以在不同的平台运行

JAVA两种核心机制

JVM虚拟机

java程序编译成字节码文件在虚拟机中运行

垃圾回收机制GC,garbage collection

即内存回收机制，开通一个系统级线程自动检测那些内存不用了然后回收

优点：不会出现忘记回收；缺点：回收不及时

一般的观点是宁可回收不及时但是一定要回收，使用自动的垃圾回收更加合适

JAVA语法

public class name{

public static void main(String[] args){

...; # 严格区分大小写，记得加分号

}

}

单行注释 //...

多行注释 /\*

...

\*/

文档注释（java特有）

/\*\*

\*...

\*...

\*@author

\*@version 1.0.0

\*/

标识符

26个英文字母大小写，0~9，\_或$

凡是可以自己起名字的地方都叫标识符

数字不可以用在标识符的开头

不可以单独使用关键字和保留字，但是可以包含

严格区分大小写，长度无限制

不能包含空格

命名规范

包名：多单词组成时所有字母都小写

类名、接口名：多单词组成时，所有单词的首字母大写

变量名、方法名：多单词组成时，第一个单词首字母小写，第二个单词开始每个单词首字母大写

常量名：所有字母都大写，多单词时每个单词用下划线连接

变量定义

数据类型 变量名 = 变量名

int i = 1

变量在一对大括号之间有效，必须初始化

数据类型

基本数据类型

数值型

整数类型(1bit即一个二进制位)

占用内寸 （表数范围）

byte： 8bit(-128~127)

short： 2byte=16bit(-2^15~2^15-1)

int： 4byte=32bit(-2^31~2^31-1)

long： 8byte=64bit(-2^63~2^63-1)

long i = 3l(long类型变量赋值的时候要在后面加上一个l)

浮点类型

占用内存 表数范围 精度

float（单精度） 4byte -2^128~2^128 7位有效数字

float i = 3.11f/F(float类型变量赋值的时候加上一个f/F)

double（双精度）8byte -2^1024~2^1024 8位有效数字

字符型

char:"...";'...'

转义字符 说明

\b 退格符

\n 换行符

\r 回车符

\t 制表符

\" 双引号

\' 单引号

\\ 反斜线

布尔型

boolean：适于逻辑运算，一般用于程序流程控制；分为true和false，没有null，不可以用0和非0来替代

boolean b1 = true/false

引用数据类型:用来接受字符串

String字符串：由0到多个字母数字符号共同组成的一个串，这个串要用英文的双引号括起来

不同的字符出赋值相同时内存中不会出现几个相同的字符串，而是将字符串指向同一个内存地址，内存地址中有这个字符串

引用类型可以用null作为值，可以在初始化时赋值null

类：class

接口：interface

数组：[]

java中除了8种基本数据类型都是引用数据类型

集成开发环境:文本编辑 编译 运行

eclipse IDEA NetBeans

基本数据类型转换

自动类型转换：容量小的类型自动转换为容量大的类型

char/byte/short-->int-->long-->float-->double

前三个之间不会相互转换，需要先转换为int

char与int运算用ASCII码运算

ASCII码

128个字符的编码，大写字母S是65，char类型参与数学运算时，转换为相应的ASCII码值再计算

把任何类型的数据类型和字符串类型相加都会转换为字符串类型

Java在运算时按照顺序计算3+4+"abc"-->7abc;"abc"+3+4-->abc34

强制类型转换：容量大的类型强制转换为容量小的类型，要小心使用，防止溢出

boolean类型不能转换为其他类型的数据

运算符

算术运算符

% 取模 取余数

a++ 先赋值后运算

++a 先运算后赋值

整数/整数=整数

小数/整数=小数

+ 字符串连接

赋值运算符

符号：= 可以自动类型转换或强制类型转换支；持连续赋值

扩展赋值运算符：+=；-=；\*=；/=；%= 变量在参与运算时，变量会强制转换

比较运算符

== 输出为true or false

>= 输出为true or false

< 输出为true or false

> 输出为true or false

逻辑运算符

&逻辑与

|逻辑或

!逻辑非

^逻辑异或

!= 输出为true or false

<= 输出为true or false

&&短路与：前面不成立就不成立，前面成立则要看后面的是否成立

||短路或：前面成立就成立,前面不成立再看后面

位运算符

<< 左移 m<<n ——> m\*2^n

>> 右移 m>>n ——> m\*2^-n 有符号的右移是看右移之后的首位是0还是1，是0前面空位补0，是1就补1

>>> 无符号右移 正数m>>>n ——> m\*2^n 无符号的右移不管移动之后首位是0还是1，空位都是直接补0

& 与运算 转换为二进制按位与

| 或运算 转换为二进制按位或

^ 异或运算 转换为二进制按位异或

~ 反码

位运算符是直接对二进制进行运算

三元（目）运算符

（条件表达式）？表达式1：表达式2 ——> 条件运算符true运算结果为表达式1，false运算结果为表达式2

()可以提高运算符优先级

程序流程控制

顺序结构：定义成员变量时采用合法的前向引用

分支（选择）结构

if(判断){

执行语句;

}else{

执行语句;

}

switch(变量){

case 常量1:

语句1；

break;

case 常量2:

语句2；

break;

case 常量1:

语句3；

break;

}

default:

语句4；

break;

case子句中的值必须是常量

break执行完就会跳出执行，否则就会从头执行到尾

当不知道使用什么的时候，使用if，if的使用范围更广

switch执行效率高，判断的数值类型有byte、short、int、char、string、枚举

循环结构

初始化部分

循环条件部分

循环体部分

迭代部分

循环语句分类

for循环

for(初始化表达式；布尔值测试表达式；更改表达式){

代码块；

}

while循环

初始化语句

while(布尔值测试表达式){

代码块;

更改语句；

}

do/while循环

初始化语句

do{

代码块；

更改语句；

}

while(布尔值测试表达式)

嵌套循环

尽量保证外层循环次数小于内层循环次数

特殊流程控制语句

break语句：用于中止当前所在的循环

continue语句：跳出当前所在循环，继续下一次循环

return语句：结束一个方法，当一个方法执行到一个return语句时，这个方法将被结束，无论处在多少层循环之中

break只能用于switch语句和循环语句

continue只能用于循环语句

break、continue之后不能有其他的语句，因为程序永远不会执行其后的语句

一维数组

一维数组声明

type var[]或type[] var

int[] a1

int a1[]

double b[]

Mydate[] c //对象数组

一维数组初始化

动态初始化：数组声明且为数组元素分配空间与赋值的操作分开进行

int[] arr = new int[2];

arr[0] = 3;

arr[1] = 9;

arr[2] = 8;

静态初始化：在定义数组的同时就为数组元素分配空间并赋值

int a[] = new int[]{3, 9, 8};

int[] a = {3, 9, 8};

数组元素引用

String[] strs = new String[]{...}

strs[0~n] strs.length

数组元素的默认初始化

使用动态初始化的时候，数组的元素会有默认值，数字的默认值是0，对象的默认值是null

数组元素的赋值

数组名[下标]=赋值；

多维数组

一维数组中的每个元素都是一个数组，这种数组就是二维数组

动态初始化

int[][] ii = new int[3][2]

3:第一维部分的长度是3，第二维也就是第一维的每个元素的长度是2

{

{1，2}

{2，3}

{3，4}

}

int[][] ii = new int[3][]

也可以只定义第一维的长度，第二维不定义

静态初始化

和一维数组类似

取值

ii[0][0]

特殊写法

int[] x,y[];x是一维数组，y是二维数组

arr.length获取的是数组的一维长度

数组中涉及的常见算法

1、最大值、最小值、总和、平均数

2、数组的复制、反转

3、数组元素的排序

插入排序：直接插入排序、折半插入排序、shell排序

交换排序：冒泡排序、快速排序（分区交换排序）

选择排序：简单选择排序、堆排序

归并排序

基数排序

数组操作常见问题

数组下标越界异常（ArrayIndexOutOfBoundException）

访问到了数组中不存在的下标时发生，例如原来长度为3，却访问了a[3]

空指针异常（NullPointerException）

arr引用没有指向实体，却在操作实体中的元素

面向对象编程OOP

三大特征

封装

继承

多态

Java类及类的成员

属性

方法或函数

类的语法格式

修饰符 class 类名{

属性声明；

方法声明；

}

类的使用就是指类的实例化

对象的创建和使用

使用new+构造器创建一个新的对象

类的属性

修饰符 类型 属性名=初值

private：该属性只能由该类的方法访问

public：该属性可以被该类以外的方法访问

类型：任何基本类型，如int、boolean

变量的分类：成员变量和局部变量

在方法体外，类体内声明的变量称为成员变量

实例变量：不以static修饰

类变量：以static修饰

在方法体内部声明的变量称为局部变量

形参：方法签名中定义的变量

方法局部变量：在方法内定义

代码块局部变量：在代码块内定义

方法的属性

修饰符 返回值类型 方法名（参数列表）{

具体语句；

}

方法的调用：方法只有在被调用的时候才会被执行

方法内可以调用方法，但是不能定义方法

同一个类中，所有的方法可以互相调用，不用new去实例化对象

匿名对象

不定义对象的句柄，直接调用这个对象的方法，例如new Person().shout()

对一个对象只需要进行一次方法调用，就可以使用匿名对象

经常将匿名对象作为实参传递给一个方法调用

类的访问机制

在一个类中的访问机制：类中的方法可以直接访问类中的成员变量，static方法访问非static方法，编译不通过

在不同类中的访问机制：先创建要访问类的对象，再用对象访问类中定义的成员

面向对象思想落地法则

1、关注类的设计，即设计类的属性、方法

2、类的实例化，即创建类的对象

3、通过对象.属性，对象.方法执行

方法的重载（overload）

在同一个类中，允许存在一个以上的同名方法，只要它们的参数个数或者参数类型不同即可

可变参数的形参

参数类型...参数名

方法的参数传递

基本数据类型

方法必须有其所在类或对象调用才有意义，若方法含有参数：

形参：方法声明时的参数

实参：方法调用时实际传递给形参的参数值

java中参数传递的方法只有一种：值传递

基本数据类型在参数传递过程中，就是把实参的值复制到形参上

JVM内存模型

栈：基础数据类型的值/对象的引用（对象的地址），它的值存放在堆中

堆：所有的对象（包括自己定义的对象和字符串对象）

方法区：所有的class和static变量

引用对象

包

Java中报的概念和文件夹的概念类似，包就是为了解决同名的问题

包下还可以有包，package语句中，用“.”来指明包（目录）的层次

引用包下面的类

import 包名.子包名.类名

可以引用包下面所有的类

import 包名.\*

也可以在使用类时，把名称写全 包名.子包名.类名 = new 包名.子包名.类名()

JDK中主要的包介绍

java.lang——包含一些Java语言的核心类：String Math Integer System Thread 提供常用功能

java.net——包含执行与网络相关的操作的类和接口

java.io——包含能提供多种输入输出功能的类

java.util——包含一些实用工具类，如定义系统特性、接口的集合框架类、使用与日期日历相关的函数

java.text——包含一些java格式化相关的类

java.sql——包含java进行JDBC数据库编程的相关类和接口

java.awt——包含了构成抽象窗口工具集（abstract window toolkits）的多个类，这些类被用来构建和管理应用程序的图形用户界面（GUI）

java.applet——包含applet运行所需的一些类

信息的封装和隐藏

通过将数据声明为私有的private，再提供公共public的方法getXxx()和setXxx()实现对该属性的操作

隐藏细节、只能用实现制定好的方法来访问数据，可以方便逻辑控制，限制不合理的操作、便于修改，增加可维护性

具体过程：

把属性设置成私有的，用private关键字

编写public的get和set方法设置属性和获取属性

四种访问权限修饰符

类内部 同一个包 子类 除了前面之外的任何地方

private： yes

（缺省）： yes yes

protected： yes yes yes

public： yes yes yes yes

如果子类和父类在同一个包下，对于父类的成员修饰符只要不是私有的private就都可以使用

子类创建方式 子类 extends 父类

对于class的权限修饰符只有public和default:

public的类到处可以被访问

default只有在同一个包内部的类访问

一个Java文件中写多个类只有一个能写public

构造器

根据参数不同，构造器可以被分为：

隐式无参数构造器（系统默认提供）

显示定义一个或多个构造器（无参、有参：实现不同new对象赋予的初值不同）

注意：

Java语言中每个类至少都有一个构造器

默认构造器的修饰符和所属类的修饰符一致

一旦显示定义了构造器，则系统不再提供默认构造器

一个类可以创建多个重载的构造器

父类的构造器不可以被子类继承

new对象实际上就是调用类的构造方法

构造器的重载

构造器可以同名，但是可以设置不同个数的参数

为了方便调用方可以灵活创建处不同需要的对象

重载的多个构造方法实际上提供了多个初始化new对象的模板

使用this，调用属性、方法

当形参与成员变量重名时，如果在方法内部需要使用成员变量，必须添加this来表明该变量是类成员

在任意方法内，如果使用当前类的成员变量或成员方法可以在其前面添加this，增强程序的阅读性

this可以作为一个类中，构造器相互调用的特殊格式

注意：

this()必须放在构造器的首行

使用this调用本类的其他构造器时，保证至少有一个构造器时不用this的，实际上就是不能出现构造器自己调用自己

JavaBean

Java语言写的可重用组件 私有属性 公有get&&set方法

高级类属性

面向对象特征之二：继承

利用extends关键字继承：class Subclass extend Suoerclass()

父类(基类或超类)：共性代码->子类：特有代码，通过继承父类省掉子类的共性代码

子类不是父类的子集而是父类的扩展

注意

不要仅为了使用某些共有的属性去继承，会有逻辑问题 例如狗和人

Java只支持单继承，不支持多继承

方法的重写（override）

在子类中可以根据需要对从父类中继承来的方法进行改造，也称方法的重置、覆盖。在程序执行时，子类的方法将覆盖父类的方法

子类的方法不能比父类同名的方法有更高的权限

子类不能访问父类私有的东西

方法的重载，一个类可以有多个同名方法

方法的重写，子类可以重新写父类的方法，覆盖父类的方法

四种访问权限修饰符

关键字super

Java类中使用super来调用父类中的指定操作

super可用于访问父类中定义的属性

super可用于调用父类中定义的成员方法

super可用于在子类构造方法中调用父类的构造器

注意：

当子父类出现同名成员时，可以用super进行区分

super的追溯不仅限于直接父类，还可以调用父类的父类 supersuper<-super<-sub

super代表父类的内存空间的标识

调用父类的构造器

子类中所有的构造器默认都会访问父类中空参数的构造器

当父类没有空参数的构造器时，子类的构造器必须通过this（参数列表）或super（参数列表）语句指定调用本类或父类中响应的构造器，且必须放在构造器的第一行

如果子类构造器中既未显式调用父类或本类的构造器，且父类中有没有无参的构造器，则编译出错

子类对象实例化过程：栈内存 堆内存 方法区

面向对象特征之三：多态

Java中有两种体现：

方法的重载（overload本类中的同名方法：体现了相同名称的方法实现了不同的逻辑）

重写（overwrite子类对父类方法的覆盖：体现了子类可以使用和父类相同的方法名，覆盖掉父类的逻辑）

Java引用对象有两个类型：编译时类型（由声明对象使用的类型决定）和运行时类型（由实际幅给该变量的对象决定）。

若编译时类型和运行时类型不一样时，就出现多态（polymorphism）

对象的多态：子类的对象可以替代父类的对象使用

一个变量只能有一种确定的数据类型；一个引用类型变量可能指向多种不同类型的对象

子类可以看作是特殊的父类，所有父类类型的引用可以指向子类的对象：向上转型（upcasting）

一个引用类型变量如果声明为父类的类型，实际引用的是子类对象，那么该变量就不能再访问子类中添加的属性和方法

虚拟方法调用（virtual method invocation）

动态绑定 编译时类型和运行时类型 方法的调用是在运行时确定的

成员方法的多态性，也就是动态绑定，必须得存在于方法的重写之上

instanceof操作符

x instanceof A：检验x是否是类A的对象/子类，返回值为Boolean类型

Object类、包装类

Object类时所有Java类的根父类，也叫基类

如果在类的生命中未使用extends关键字指明器父类，则默认父类为Object类

Object类中的主要方法

方法名称 类型 描述

public Object() 构造 构造方法

public boolean equals(Object obj) 普通 对象比较

public int hashCode() 普通 取得hash码

public String toString() 普通 对象打印时调用

对象类型转换（Casting）

自动类型转换：小的数据类型自动转化为大的数据类型

强制类型转换：大的数据类型强制转化为小的数据类型

对Java对象的强制类型转换称为造型

无继承关系的引用类型间的转换是非法的

String对象的创建

字面量创建String对象

String s = "abc"

new创建String对象

String s = new String("abc")

用字面量创建String比较节省内存

包装类（Wrapper）

针对八种基本数据类型定义了相应的引用类型——包装类（封装类）

首字母大写，除了int->integer、char->character

装箱：

通过包装类的构造器实现

通过字符串参数构造包装类对象，数字型的字符串要是数字，不然会报错NumberFormatException

拆箱：?.xxValue

基本数据类的包装类主要用于基本数据类型与字符串之间的转换

关键字static

有static关键字的变量属于类变量，类名.变量名可以直接访问，是类的一部分，被类的所有实例化对象共享，也可以叫做静态变量

没有static关键字的变量属于实例变量，只有实例化之后才可以使用

设计模式：在大量的时间中总结和理论化之后优选的代码结构、编程风格、以及解决问题的思考方式。

（也就是在实际编程过程中，逐渐总结出的一些解决问题的套路）

单例（singleton）设计模式：在开发过程中只有一个实例化对象，在整个软件系统运行过程中，这个类只被实例化一次，以后不管在哪都只调用这一个实例

使用场景：实例化对象的创建需要消耗大量的时间和资源

饿汉式：类加载之后直接new一个私有对象，每次调用都使用这个对象

懒汉式：最开始，对象是null，直到有第一个人调用才new一个对象，以后所有调用者都用这个对象（暂时懒汉式存在线程安全问题，多线程可以修复）

再识main方法，main方法的形参相当于无限个字符串，可以通过命令行输出调用Testmain类，调用格式java str1 str2，然后会自动输出

类的成员之四：初始化块

对Java对象进行初始化

执行顺序：

声明成员变量的默认值

显式初始化、多个初始化块一次被执行（同级别下按先后顺序执行）

构造器再对成员进行赋值操作

静态代码块：一个类中初始化块若有修饰符，只能用static修饰，当类被载入时，类属性的声明和静态代码块先后顺序被执行，且只被执行一次

关键字：final

final修饰类不能被继承

final修饰方法不能被重写

final修饰变量不能改变值，即常量，名称大写，只能赋值一次

抽象类（abstract class）

abstract修饰类的时候，类叫做抽象类

abstract修饰方法的时候，方法叫做抽象方法

抽象方法只有方法的声明，没有方法的实现，以分号结束，abstractMethod(int a);

抽象类不能被实例化，抽象类是用来作为父类被继承的，抽象类的子类必须重写父类的抽象方法，并提供方法体。若没有重写全部的抽象方法，仍未抽象类

不能用abstract修饰属性、私有方法、构造器、静态方法、final的方法

Q1：为什么抽象类不可以用final声明

Ans：抽象类不能被实例化，且是用来被继承的，抽象类的子类必须重写父类的抽象方法，并提供方法体

Q2:一个抽象类可以定义构造器吗

Ans:抽象类可以构造方法，只是不能直接创建抽象类的实力对象而已

模板方法设计模式（TemplateMethod）

抽象类体现的是一种模板模式的设计，抽象类作为多个子类的通用模板，子类在抽象类的基础上进行扩展、改造，但子类总体上会保留抽像类的行为方式

抽象类——大纲，抽象方法——每个章节的标题

子类——根据标题把每个章节细化出来

接口：interface 是抽象方法和常量值的定义的集合

本质上讲，接口是一个特殊的抽象类，这种抽象类中只包含常量和方法的定义，而没有变量和方法的实现

实现接口类：class Subclass implements InterfaceA{}

一个类可以实现多个接口，接口也可以继承其它接口

所以成员变量都默认是由public static final修饰 不管写没写都会默认是这个

所有方法默认是由public abstract修饰

接口没有构造器

接口采用多层继承机制

实现接口的类中必须提供接口中所有方法的具体实现内容，方可实例化。否则，仍为抽象类

接口的主要用途就是被实现类实现（面向接口编程）

java类的语法格式

<modifier> class <name> extends [<superclass>] [implements <interface> [,<interface>]\*]{

<declarations>\*

}

为什么需要接口？

如果抽象类增加新的抽象方法，那么子类需要完全实现或者变成抽象类，很麻烦

只能新建一个接口，在接口上扩展方法，其他需要的子类自行去实现接口

如果实现接口的类中没有实现接口中的全部方法，必须将此类定义为抽象类

接口可以继承另一个接口，使用extends关键字，并且可以多重继承

抽象类：是对一类事物的高度抽象，既有属性也有方法

接口：是对方法的抽象，也就是对一系列动作的抽象

工厂模式(FactoryMethod)

通过工厂把new对象给隔离，通过产品的接口可以接收不同实际产品的实现类、实例的类名的改变不影响其他合作开发人员的编程

内部类 inner class

主要解决Java不能多重继承的问题

可以使用内部类来变相的实现类的多重继承，可以同时继承多个类