**Java程序设计课程总结**

第二章 面向对象程序设计

面向对象(掌握)

(1)面向对象

面向对象是基于面向过程的编程思想

(2)面向对象的思想特点

A:是一种更符合我们思考习惯的思想

B:把复杂的事情简单化

C:让我们从执行者变成了指挥者

注意：如何让我们的操作更符合面向对象思想呢?

A:有哪些类

B:每个类有哪些成员

C:类与类的关系

(4)类与对象

A:现实世界的事物

属性 事物的基本描述

行为 事物的功能

B:Java语言中最基本的单位是类。所以，我们要用类来体现事物

C:类

成员变量 事物属性

成员方法 事物行为

D:类：是一组相关的属性和行为的集合。是一个抽象的概念。

对象：是该类事物的具体存在，是一个具体的实例。(对象)

(5)类的定义及使用

A:类的定义

成员变量

成员方法

B:使用类的内容

a:创建对象? 格式

类名 对象名 = new 类名();

b:如何使用成员变量和成员方法呢

对象名.成员变量

对象名.成员方法()

(7)内存图

(8)Java程序的开发，设计和特征

A:开发：就是不断的创建对象，通过对象调用功能

B:设计：就是管理和维护对象间的关系

C:特征

a:封装

b:继承

c:多态

第三章Java语言基础

1:关键字(掌握)

(1)被Java语言赋予特定含义的单词

(2)特点：

全部小写。

(3)注意事项：

A:goto和const作为保留字存在。

B:类似于Notepad++这样的高级记事本会对关键字有特殊颜色标记

2:标识符(掌握)

(1)就是给类，接口，方法，变量等起名字的字符序列

(2)组成规则：

A:英文大小写字母

B:数字

C:$和\_

(3)注意事项：

A:不能以数字开头

B:不能是java中的关键字

C:区分大小写

(4)常见的命名规则(见名知意)

A:包 全部小写

单级包：小写

举例：liuyi,com

多级包：小写，并用.隔开

举例：cn.itcast,com.baidu

B:类或者接口

一个单词：首字母大写

举例：Student,Demo

多个单词：每个单词首字母大写

举例：HelloWorld,StudentName

C:方法或者变量

一个单词：首字母小写

举例：name,main

多个单词：从第二个单词开始，每个单词首字母大写

举例：studentAge,showAllNames()

D:常量

全部大写

一个单词：大写

举例：PI

多个单词：大写，并用\_隔开

举例：STUDENT\_MAX\_AGE

3:注释(掌握)

(1)就是对程序进行解释说明的文字

(2)分类：

A:单行注释 //

B:多行注释 /\*\*/

C:文档注释(后面讲) /\*\* \*/

(3)把HelloWorld案例写了一个带注释的版本。

后面我们要写一个程序的过程。

需求：

分析：

实现：

代码体现：

(4)注释的作用

A:解释说明程序，提高了代码的阅读性。

B:可以帮助我们调试程序。

后面我们会讲解一个更高端的一个调试工具

4:常量(掌握)

(1)在程序执行的过程中，其值不发生改变的量

(2)分类：

A:字面值常量

B:自定义常量

(3)字面值常量

A:字符串常量 "hello"

B:整数常量 12,23

C:小数常量 12.345

D:字符常量 'a','A','0'

E:布尔常量 true,false

F:空常量 null

(4)在Java中针对整数常量提供了四种表现形式

A:二进制 由0，1组成。以0b开头。

B:八进制 由0，1，...7组成。以0开头。

C:十进制 由0，1，...9组成。整数默认是十进制。

D:十六进制 由0，1，...9,a,b,c,d,e,f(大小写均可)组成。以0x开头。

5:进制转换(了解)

(1)其他进制到十进制

系数：就是每一个位上的数值

基数：x进制的基数就是x

权：对每一个位上的数据，从右，并且从0开始编号，对应的编号就是该数据的权。

结果：系数\*基数^权次幂之和。

(2)十进制到其他进制

除基取余，直到商为0，余数反转。

(3)进制转换的快速转换法

A:十进制和二进制间的转换

8421码。

B:二进制到八进制，十六进制的转换

6:变量(掌握)

(1)在程序的执行过程中，其值在某个范围内可以发生改变的量

(2)变量的定义格式：

A:数据类型 变量名 = 初始化值;

B:数据类型 变量名;

变量名 = 初始化值;

7:数据类型(掌握)

(1)Java是一种强类型语言，针对每种数据都提供了对应的数据类型。

(2)分类：

A:基本数据类型：4类8种

B:引用数据类型：类，接口，数组。

(3)基本数据类型

A:整数 占用字节数

byte 1

short 2

int 4

long 8

B:浮点数

float 4

double 8

C:字符

char 2

D:布尔

boolean 1

注意：

整数默认是int类型，浮点数默认是double。

长整数要加L或者l。

单精度的浮点数要加F或者f。

8:数据类型转换(掌握)

(1)boolean类型不参与转换

(2)默认转换

A:从小到大

B:byte,short,char -- int -- long -- float -- double

C:byte,short,char之间不相互转换，直接转成int类型参与运算。

(3)强制转换

A:从大到小

B:可能会有精度的损失，一般不建议这样使用。

C:格式：

目标数据类型 变量名 = (目标数据类型) (被转换的数据);

**运算符的优先级（从高到低）**

|  |  |  |
| --- | --- | --- |
| 优先级 | 描述 | 运算符 |
| 1 | 括号 | ()、[] |
| 2 | 正负号 | +、- |
| 3 | 自增自减，非 | ++、--、! |
| 4 | 乘除，取余 | \*、/、% |
| 5 | 加减 | +、- |
| 6 | 移位运算 | <<、>>、>>> |
| 7 | 大小关系 | >、>=、<、<= |
| 8 | 相等关系 | ==、!= |
| 9 | 按位与 | & |
| 10 | 按位异或 | ^ |
| 11 | 按位或 | | |
| 12 | 逻辑与 | && |
| 13 | 逻辑或 | || |
| 14 | 条件运算 | ?: |
| 15 | 赋值运算 | =、+=、-=、\*=、/=、%= |
| 16 | 位赋值运算 | &=、|=、<<=、>>=、>>>= |

如果在程序中，要改变运算顺序，可以使用()。

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1:运算符(掌握)

(1)算术运算符

A:+,-,\*,/,%,++,--

B:+的用法

a:加法

b:正号

c:字符串连接符

C:/和%的区别

数据做除法操作的时候，/取得是商，%取得是余数

D:++和--的用法

a:他们的作用是自增或者自减

b:使用

\*\*单独使用

放在操作数据的前面和后面效果一样。

a++或者++a效果一样。

\*\*参与操作使用

放在操作数的前面：先自增或者自减，再参与操作

int a = 10;

int b = ++a;

放在操作数的后面：先参与操作，再自增或者自减

int a = 10;

int b = a++;

(2)赋值运算符

A:=,+=,-=,\*=,/=,%=等

B:=叫做赋值运算符，也是最基本的赋值运算符

int x = 10; 把10赋值给int类型的变量x。

C:扩展的赋值运算符的特点

隐含了自动强制转换。

面试题：

short s = 1;

s = s + 1;

short s = 1;

s += 1;

请问上面的代码哪个有问题?

(3)比较运算符

A:==,!=,>,>=,<,<=

B:无论运算符两端简单还是复杂最终结果是boolean类型。

C:千万不要把==写成了=

(4)逻辑运算符

A:&,|,^,!,&&,||

B:逻辑运算符用于连接boolean类型的式子

C:结论

&:有false则false

|:有true则true

^:相同则false，不同则true。

情侣关系。

!:非true则false，非false则true

&&:结果和&是一样的，只不过有短路效果。左边是false，右边不执行。

||:结果和|是一样的，只不过有短路效果。左边是true，右边不执行。

(5)位运算符(了解)

A:^的特殊用法

一个数据针对另一个数据位异或两次，该数不变

B:面试题

a:请实现两个变量的交换

\*\*采用第三方变量

\*\*用位异或运算符

左边a,b,a

右边a^b

b:请用最有效率的方式计算出2乘以8的结果

2<<3

(6)三元运算符

A:格式

比较表达式?表达式1:表达式2;

B:执行流程：

首先计算比较表达式的值，看是true还是false。

如果是true，表达式1就是结果。

如果是false，表达式2就是结果。

C:案例：

a:比较两个数据是否相等

b:获取两个数据中的最大值

c:获取三个数据中的最大值

2:键盘录入(掌握)

(1)实际开发中，数据是变化的，为了提高程序的灵活性，我们加入键盘录入数据。

(2)如何实现呢?目前就记住

A:导包

import java.util.Scanner;

位置：在class的上边

B:创建对象

Scanner sc = new Scanner(System.in);

C:获取数据

int x = sc.nextInt();

(3)把三元运算符的案例加入键盘录入改进。

3:流程控制语句

(1)顺序结构 从上往下，依次执行

(2)选择结构 按照不同的选择，执行不同的代码

(3)循环结构 做一些重复的代码

4:if语句(掌握)

(1)三种格式

A:格式1

if(比较表达式) {

语句体;

}

执行流程：

判断比较表达式的值，看是true还是false

如果是true，就执行语句体

如果是false，就不执行语句体

B:格式2

if(比较表达式) {

语句体1;

}else {

语句体2;

}

执行流程：

判断比较表达式的值，看是true还是false

如果是true，就执行语句体1

如果是false，就执行语句体2

C:格式3

if(比较表达式1) {

语句体1;

}else if(比较表达式2){

语句体2;

}

...

else {

语句体n+1;

}

执行流程：

判断比较表达式1的值，看是true还是false

如果是true，就执行语句体1

如果是false，就继续判断比较表达式2的值，看是true还是false

如果是true，就执行语句体2

如果是false，就继续判断比较表达式3的值，看是true还是false

...

如果都不满足，就执行语句体n+1

(2)注意事项

A:比较表达式无论简单还是复杂，结果是boolean类型

B:if语句控制的语句体如果是一条语句，是可以省略大括号的；如果是多条，不能省略。

建议：永远不要省略。

C:一般来说，有左大括号，就没有分号，有分号，就没有左大括号。

D:else后面如果没有if，是不会出现比较表达式的。

E:三种if语句其实都是一个语句，只要有一个执行，其他的就不再执行。

(3)案例：

A:比较两个数是否相等

B:获取两个数中的最大值

C:获取三个数中的最大值(if语句的嵌套)

D:根据成绩输出对应的等级

E:根据月份，输出对应的季节

F:根据x计算对应y的值并输出

(4)三元运算符和if语句第二种格式的关系

所有的三元运算符能够实现的，if语句的第二种格式都能实现。

反之不成立。

如果if语句第二种格式控制的语句体是输出语句，就不可以。

因为三元运算符是一个运算符，必须要有一个结果返回，不能是一个输出语句。

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1:switch语句(掌握)

(1)格式：

switch(表达式) {

case 值1:

语句体1;

break;

case 值2:

语句体2;

break;

...

default:

语句体n+1;

break;

}

格式解释说明：

switch:说明这是switch语句。

表达式:可以是byte,short,int,char

JDK5以后可以是枚举

JDK7以后可以是字符串

case:后面的值就是要和表达式进行比较的值

break:表示程序到这里中断，跳出switch语句

default:如果所有的情况都不匹配,就执行这里,相当于if语句中的else

(2)面试题

switch语句的表达式可以是byte吗?可以是long吗?可以是String吗?

可以,不可以,JDK7以后可以

(3)执行流程:

A:首先计算表达式的值

B:和每一个case进行匹配，如果有就执行对应的语句体，看到break就结束。

C:如果没有匹配，就执行default的语句体n+1。

(4)注意事项:

A:case后面只能是常量，不能是变量，而且，多个case后面的值不能出现相同的

B:default可以省略吗?

可以省略，但是不建议，因为它的作用是对不正确的情况给出提示。

特殊情况：

case就可以把值固定。

A,B,C,D

C:break可以省略吗?

可以省略，但是结果可能不是我们想要的。

会出现一个现象：case穿透。

最终我们建议不要省略

D:default一定要在最后吗?

不是，可以在任意位置。但是建议在最后。

E:switch语句的结束条件

a:遇到break就结束了

b:执行到末尾就结束了

(5)案例：

A:键盘录入一个数字(1-7),输出对应的星期几。

B:单项选择题

C:键盘录入一个字符串的问题

String s = sc.nextLine();

D:根据给定的月份,输出对应的季节

(6)if语句和switch语句各自的场景

A:if

针对boolean类型的判断

针对一个范围的判断

针对几个常量的判断

B:switch

针对几个常量的判断

2:循环语句(掌握)

(1)有三种:for,while,do...while

(2)for循环语句

A:格式

for(初始化语句;判断条件语句;控制条件语句){

循环体语句;

}

执行流程：

a:执行初始化语句

b:执行判断条件语句

如果这里是true，就继续

如果这里是false，循环就结束

c:执行循环体语句

d:执行控制条件语句

e:回到b

B:注意事项

a:判断条件语句无论简单还是复杂，结果是boolean类型

b:循环体语句如果是一条，可以省略大括号，但是不建议

c:有分号就没有左大括号，有左大括号就没有分号

C:案例

a:输出10次HelloWorld

b:输出1-10的数据

c:输出10-1的数据

d:求1-10的和

e:求1-100的和,求1-100的偶数和,求1-100的奇数和

f:求5的阶乘

g:在控制台打印水仙花数

h:统计水仙花个数

i:改进版的回文数

一个五位数

个位 = 万位

十位 = 千位

个位 + 十位 + 千位 + 万位 = 百位

j:统计1-1000之间同时满足如下条件的数据有多少个

x%3==2

x%5==3

x%7==2

(3)while循环

A:基本格式

while(判断条件语句) {

循环体语句;

}

扩展格式：

初始化语句;

while(判断条件语句){

循环体语句;

控制条件语句;

}

通过查看这个格式，我们就知道while循环可以和for循环等价转换。

B:while的练习

把for语句的练习用while改进

C:for和while的区别

a:使用上的区别

for语句的那个控制条件变量,在循环结束后不能在使用了。

而while的可以继续使用。

b:理解上的区别

for适合于一个范围的判断

while适合次数不明确的

举例:吃葡萄

D:案例：

a:珠穆朗玛峰问题

b:小芳存钱问题(break以后才能做)

(4)do...while循环

A:基本格式

do {

循环体语句;

}while(判断条件语句);

扩展格式：

初始化语句;

do {

循环体语句;

控制条件语句;

}while(判断条件语句);

通过查看格式，我们就可以看出其实三种循环的格式可以是统一的。

B:三种循环的区别

a:do...while循环至少执行一次循环体

b:for和while必须先判断条件是否是true，然后后才能决定是否执行循环体

(5)循环使用的注意事项(死循环)

A:一定要注意修改控制条件,否则容易出现死循环。

B:最简单的死循环格式

a:while(true){...}

b:for(;;){}

3:控制跳转语句(掌握)

(1)break:中断的意思

A:用在循环和switch语句中，离开此应用场景无意义。

B:作用

a:跳出单层循环

b:跳出多层循环，需要标签语句的配合

(2)continue:继续

A:用在循环中，离开此应用场景无意义。

B:作用

a:跳出单层循环的一次，可以继续下一次

C:填空题

for(int x=1; x<=10; x++) {

if(x%3 == 0) {

//补齐代码

}

System.out.println("Java基础班");

}

如何让控制台输出2次：Java基础班

如何让控制台输出7次：Java基础班

如何让控制台输出13次：Java基础班

(3)return:返回

A:用于结束方法的，后面还会在继续讲解和使用。

B:一旦遇到return,程序就不会在继续往后执行。

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1:方法(掌握)

(1)方法：就是完成特定功能的代码块。

注意：在很多语言里面有函数的定义，而在Java中，函数被称为方法。

(2)格式：

修饰符 返回值类型 方法名(参数类型 参数名1,参数类型 参数名2...) {

方法体语句;

return 返回值;

}

修饰符：目前就用 public static。后面再详细讲解其他修饰符

返回值类型：就是功能结果的数据类型

方法名：就是起了一个名字，方便我们调用该方法。

参数类型：就是参数的数据类型

参数名：就是变量

参数分类：

实参：实际参与运算的数据

形参：方法上定义的，用于接收实际参数的变量

方法体语句：就是完成功能的代码块

return：结束方法

返回值：就是功能的结果，由return带给调用者。

(3)两个明确：

返回值类型：结果的数据类型

参数列表：参数的个数及对应的数据类型

(4)方法调用

A:有明确返回值的方法

a:单独调用，没有意义

b:输出调用，不是很好，因为我可能需要结果进行进一步的操作。 c:赋值调用，推荐方案

B:void类型修饰的方法

a:单独调用

(6)方法的注意事项

A:方法不调用不执行

B:方法之间是平级关系，不能嵌套定义

C:方法定义的时候，参数是用，隔开的

D:方法在调用的时候，不用在传递数据类型

E:如果方法有明确的返回值类型，就必须有return语句返回。

(7)方法重载

在同一个类中，方法名相同，参数列表不同。与返回值无关。

参数列表不同：

参数的个数不同。

参数的对应的数据类型不同。

(8)方法重载案例

不同的类型的多个同名方法的比较。

2:数组(掌握)

(1)数组：存储同一种数据类型的多个元素的容器。

(2)特点：每一个元素都有编号，从0开始，最大编号是长度-1。

编号的专业叫法：索引

(3)定义格式

A:数据类型[] 数组名;

B:数据类型 数组名[];

推荐是用A方式，B方法就忘了吧。

但是要能看懂

(4)数组的初始化

A:动态初始化

只给长度，系统给出默认值

举例：int[] arr = new int[3];

B:静态初始化

给出值，系统决定长度

举例：int[] arr = new int[]{1,2,3};

简化版：int[] arr = {1,2,3};

(5)Java的内存分配

A:栈 存储局部变量

B:堆 存储所有new出来的

C:方法区 存储加载的类、方法、静态数据等

D:本地方法区(系统相关)

E:寄存器(CPU使用)

注意：

a:局部变量 在方法定义中或者方法声明上定义的变量。

b:栈内存和堆内存的区别

栈：数据使用完毕，就消失。

堆：每一个new出来的东西都有地址

每一个变量都有默认值

byte,short,int,long 0

float,double 0.0

char '\u0000'

boolean false

引用类型 null

数据使用完毕后，在垃圾回收器空闲的时候回收。

(6)数组内存图

A:一个数组

B:二个数组

C:三个数组(两个栈变量指向同一个堆内存)

(7)数组的常见操作

A:遍历

方式1：

public static void printArray(int[] arr) {

for(int x=0; x<arr.length; x++) {

System.out.println(arr[x]);

}

}

方式2：

public static void printArray(int[] arr) {

System.out.print("[");

for(int x=0; x<arr.length; x++) {

if(x == arr.length-1) {

System.out.println(arr[x]+"]");

}else {

System.out.println(arr[x]+", ");

}

}

}

B:最值

最大值：

public static int getMax(int[] arr) {

int max = arr[0];

for(int x=1; x<arr.length; x++) {

if(arr[x] > max) {

max = arr[x];

}

}

return max;

}

最小值：

public static int getMin(int[] arr) {

int min = arr[0];

for(int x=1; x<arr.length; x++) {

if(arr[x] < min) {

min = arr[x];

}

}

return min;

}

C:逆序

方式1：

public static void reverse(int[] arr) {

for(int x=0; x<arr.length/2; x++) {

int temp = arr[x];

arr[x] = arr[arr.length-1-x];

arr[arr.length-1-x] = temp;

}

}

方式2：

public static void reverse(int[] arr) {

for(int start=0,end=arr.length-1; start<=end; start++,end--) {

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

}

}

D:查表

public static String getString(String[] strArray,int index) {

return strArray[index];

}

E:基本查找

方式1：

public static int getIndex(int[] arr,int value) {

for(int x=0; x<arr.length; x++) {

if(arr[x] == value) {

return x;

}

}

return -1;

}

方式2：

public static int getIndex(int[] arr,int value) {

int index = -1;

for(int x=0; x<arr.length; x++) {

if(arr[x] == value) {

index = x;

break;

}

}

return index;

}

1:二维数组(理解)

(1)元素是一维数组的数组。

(2)格式：

A:数据类型[][] 数组名 = new 数据类型[m][n];

B:数据类型[][] 数组名 = new 数据类型[m][];

C:数据类型[][] 数组名 = new 数据类型[][]{{...},{...},{...}};

D:数据类型[][] 数组名 = {{...},{...},{...}};

(3)案例(掌握):

A:二维数组的遍历

B:二维数组的求和

1:成员变量和局部变量的区别(理解)

(1)在类中的位置不同

成员变量：类中方法外

局部变量：方法定义中或者方法声明上

(2)在内存中的位置不同

成员变量：在堆中

局部变量：在栈中

(3)生命周期不同

成员变量：随着对象的创建而存在，随着对象的消失而消失

局部变量：随着方法的调用而存在，随着方法的调用完毕而消失

(4)初始化值不同

成员变量：有默认值

局部变量：没有默认值，必须定义，赋值，然后才能使用

2:类作为形式参数的问题?(理解)

(1)如果你看到一个方法需要的参数是一个类，就应该知道这里实际需要的是一个具体的对象。

3:匿名对象(理解)

(1)没有名字的对象

(2)应用场景

A:调用方法，仅仅只调用一次的时候。

b:可以作为实际参数传递。

4:封装(理解)

(1)隐藏实现细节，提供公共的访问方式

(2)好处：

A:隐藏实现细节，提供公共的访问方式

B:提高代码的复用性

C:提高代码的安全性

(3)设计原则

把不想让外界知道的实现细节给隐藏起来，提供公共的访问方式

(4)private是封装的一种体现。

封装：类，方法，private修饰成员变量

5:private关键字(掌握)

(1)私有的意义，可以修饰成员变量和成员方法

(2)特点：

被private修饰的后的成员只能在本类中被访问

(3)private的应用：

以后再写一个类的时候：

把所有的成员变量给private了

提供对应的getXxx()/setXxx()方法

6:this关键字(掌握)

(1)代表当前类的引用对象

记住：哪个对象调用方法，该方法内部的this就代表那个对象

(2)this的应用场景：

A:解决了局部变量隐藏成员变量的问题

B:其实this还有其他的应用，明天讲解。

7:构造方法(掌握)

(1)作用：用于对对象的数据进行初始化

(2)格式：

A:方法名和类名相同

B:没有返回值类型，连void都不能有

C:没有返回值

思考题：构造方法中可不可以有return语句呢?

可以。而是我们写成这个样子就OK了：return;

其实，在任何的void类型的方法的最后你都可以写上：return;

(3)构造方法的注意事项

A:如果我们没写构造方法，系统将提供一个默认的无参构造方法

B:如果我们给出了构造方法，系统将不再提供默认构造方法

如果这个时候，我们要使用无参构造方法，就必须自己给出。

推荐：永远手动自己给出无参构造方法。

(4)给成员变量赋值的方式

A:setXxx()

B:带参构造方法

(5)标准案例

class Student {

private String name;

private int age;

public Student(){}

public Student(String name,int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

8:代码：Student s = new Student();做了哪些事情?(理解)

(1)把Student.class文件加载到内存

(2)在栈内存为s开辟空间

(3)在堆内存为学生对象申请空间

(4)给学生的成员变量进行默认初始化。

(5)给学生的成员变量进行显示初始化。

(6)通过构造方法给成员变量进行初始化。

(7)对象构造完毕，把地址赋值给s变量

10:static关键字(理解)

(1)静态的意思。可以修饰成员变量和成员方法。

(2)静态的特点：

A:随着类的加载而加载

B:优先于对象存在

C:被类的所有对象共享

这其实也是我们判断该不该使用静态的依据。

D:可以通过类名调用

既可以通过对象名调用，也可以通过类名调用，建议通过类名调用。

(3)静态的内存图

静态的内容在方法区的静态区

(4)静态的注意事项；

A:在静态方法中没有this对象

B:静态只能访问静态(代码测试过)

(5)静态变量和成员变量的区别

A:所属不同

静态变量：属于类，类变量

成员变量：属于对象，对象变量，实例变量

B:内存位置不同

静态变量：方法区的静态区

成员变量：堆内存

C:生命周期不同

静态变量：静态变量是随着类的加载而加载，随着类的消失而消失

成员变量：成员变量是随着对象的创建而存在，随着对象的消失而消失

D:调用不同

静态变量：可以通过对象名调用，也可以通过类名调用

成员变量：只能通过对象名调用

(6)main方法是静态的

public:权限最大

static:不用创建对象调用

void:返回值给jvm没有意义

main:就是一个常见的名称。

String[] args:可以接收数据，提供程序的灵活性

格式：java MainDemo hello world java

java MainDemo 10 20 30

3:代码块(理解)

(1)用{}括起来的代码。

(2)分类：

A:局部代码块

用于限定变量的生命周期，及早释放，提高内存利用率。

B:构造代码块

把多个构造方法中相同的代码可以放到这里，每个构造方法执行前，首先执行构造代码块。

C:静态代码块

对类的数据进行初始化，仅仅只执行一次。

(3)静态代码块,构造代码块,构造方法的顺序问题?

静态代码块 > 构造代码块 > 构造方法

4:继承(掌握)

(1)把多个类中相同的成员给提取出来定义到一个独立的类中。然后让这多个类和该独立的类产生一个关系，

这多个类就具备了这些内容。这个关系叫继承。

(2)Java中如何表示继承呢?格式是什么呢?

A:用关键字extends表示

B:格式：

class 子类名 extends 父类名 {}

(3)继承的好处：

A:提高了代码的复用性

B:提高了代码的维护性

C:让类与类产生了一个关系，是多态的前提

(4)继承的弊端：

A:让类的耦合性增强。这样某个类的改变，就会影响其他和该类相关的类。

原则：低耦合，高内聚。

耦合：类与类的关系

内聚：自己完成某件事情的能力

B:打破了封装性

(5)Java中继承的特点

A:Java中类只支持单继承

B:Java中可以多层(重)继承(继承体系)

(6)继承的注意事项：

A:子类不能继承父类的私有成员

B:子类不能继承父类的构造方法，但是可以通过super去访问

C:不要为了部分功能而去继承

(7)什么时候使用继承呢?

A:继承体现的是：is a的关系。

B:采用假设法

(8)Java继承中的成员关系

A:成员变量

a:子类的成员变量名称和父类中的成员变量名称不一样，这个太简单

b:子类的成员变量名称和父类中的成员变量名称一样，这个怎么访问呢?

子类的方法访问变量的查找顺序：

在子类方法的局部范围找，有就使用。

在子类的成员范围找，有就使用。

在父类的成员范围找，有就使用。

找不到，就报错。

B:构造方法

a:子类的构造方法默认会去访问父类的无参构造方法

是为了子类访问父类数据的初始化

b:父类中如果没有无参构造方法，怎么办?

子类通过super去明确调用带参构造

子类通过this调用本身的其他构造，但是一定会有一个去访问了父类的构造

让父类提供无参构造

C:成员方法

a:子类的成员方法和父类中的成员方法名称不一样，这个太简单

b:子类的成员方法和父类中的成员方法名称一样，这个怎么访问呢?

通过子类对象访问一个方法的查找顺序：

在子类中找，有就使用

在父类中找，有就使用

找不到，就报错

1:final关键字(掌握)

(1)是最终的意思，可以修饰类，方法，变量。

(2)特点：

A:它修饰的类，不能被继承。

B:它修饰的方法，不能被重写。

C:它修饰的变量，是一个常量。

(3)面试相关：

A:局部变量

a:基本类型 值不能发生改变

b:引用类型 地址值不能发生改变，但是对象的内容是可以改变的

B:初始化时机

a:只能初始化一次。

b:常见的给值

定义的时候。(推荐)

构造方法中。

2:多态(掌握)

(1)同一个对象在不同时刻体现出来的不同状态。

(2)多态的前提：

A:有继承或者实现关系。

B:有方法重写。

C:有父类或者父接口引用指向子类对象。

多态的分类：

a:具体类多态

class Fu {}

class Zi extends Fu {}

Fu f = new Zi();

b:抽象类多态

abstract class Fu {}

class Zi extends Fu {}

Fu f = new Zi();

c:接口多态

interface Fu {}

class Zi implements Fu {}

Fu f = new Zi();

(3)多态中的成员访问特点

A:成员变量

编译看左边，运行看左边

B:构造方法

子类的构造都会默认访问父类构造

C:成员方法

编译看左边，运行看右边

D:静态方法

编译看左边，运行看左边

为什么?

因为成员方法有重写。

(4)多态的好处：

A:提高代码的维护性(继承体现)

B:提高代码的扩展性(多态体现)

(5)多态的弊端：

父不能使用子的特有功能。

现象：

子可以当作父使用，父不能当作子使用。

(6)多态中的转型

A:向上转型

从子到父

B:向下转型

从父到子（instance of）

3:抽象类(掌握)

(1)把多个共性的东西提取到一个类中，这是继承的做法。

但是呢，这多个共性的东西，在有些时候，方法声明一样，方法体不一样。

也就是说，方法声明一样，但是每个具体的对象在具体实现的时候内容不一样。

所以，我们在定义这些共性的方法的时候，就不能给出具体的方法体。

而一个没有具体的方法体的方法是抽象的方法。

在一个类中如果有抽象方法，该类必须定义为抽象类。

(2)抽象类的特点

A:抽象类和抽象方法必须用关键字abstract修饰

B:抽象类中不一定有抽象方法,但是有抽象方法的类一定是抽象类

C:抽象类不能实例化

D:抽象类的子类

a:是一个抽象类。

b:是一个具体类。这个类必须重写抽象类中的所有抽象方法。

(3)抽象类的成员特点：

A:成员变量

有变量，有常量

B:构造方法

有构造方法

C:成员方法

有抽象，有非抽象

(5)抽象类的几个小问题

A:抽象类有构造方法，不能实例化，那么构造方法有什么用?

用于子类访问父类数据的初始化

B:一个类如果没有抽象方法,却定义为了抽象类，有什么用?

为了不让创建对象

C:abstract不能和哪些关键字共存

a:final 冲突

b:private 冲突

c:static 无意义

4:接口(掌握)

(1)一个类提供一些基本功能。

还需提供一些额外功能，这种额外的功能，java提供了接口表示。

(2)接口的特点：

A:接口用关键字interface修饰

interface 接口名 {}

B:类实现接口用implements修饰

class 类名 implements 接口名 {}

C:接口不能实例化

D:接口的实现类

a:是一个抽象类。

b:是一个具体类，这个类必须重写接口中的所有抽象方法。

(3)接口的成员特点：

A:成员变量

只能是常量

默认修饰符：public static final

B:构造方法

没有构造方法

C:成员方法

只能是抽象的

默认修饰符：public abstract

(4)类与类,类与接口,接口与接口

A:类与类

继承关系，只能单继承，可以多层继承

B:类与接口

实现关系，可以单实现，也可以多实现。

还可以在继承一个类的同时，实现多个接口

C:接口与接口

继承关系，可以单继承，也可以多继承

(5)抽象类和接口的区别

A:成员区别

抽象类：

接口：

B:关系区别:

类与类：

类与接口：

接口与接口：

C:设计理念不同

抽象类：is a，抽象类中定义的是共性功能。

接口：like a，接口中定义的是扩展功能。

1:形式参数和返回值的问题(理解)

(1)形式参数：

类名：需要该类的对象

抽象类名：需要该类的子类对象

接口名：需要该接口的实现类对象

(2)返回值类型：

类名：返回的是该类的对象

抽象类名：返回的是该类的子类对象

接口名：返回的是该接口的实现类的对象

(3)链式编程

对象.方法1().方法2().......方法n();

这种用法：其实在方法1()调用完毕后，应该一个对象；

方法2()调用完毕后，应该返回一个对象。

方法n()调用完毕后，可能是对象，也可以不是对象。

2:包(理解)

(1)其实就是文件夹

(2)作用：

A:区分同名的类

B:对类进行分类管理

a:按照功能分

b:按照模块分

(3)包的定义(掌握)

package 包名;

多级包用.分开。

(4)注意事项：(掌握)

A:package语句必须在文件中的第一条有效语句

B:在一个java文件中，只能有一个package

C:如果没有package，默认就是无包名

(5)带包的编译和运行

A:手动式

B:自动式(掌握)

javac -d . HelloWorld.java

3:导包(掌握)

(1)我们多次使用一个带包的类，非常的麻烦，这个时候，Java就提供了一个关键字import。

(2)格式：

import 包名...类名;

另一种：

import 包名...\*;(不建议)

(3)package,import,class的顺序

package > import > class

4:权限修饰符(掌握)

(1)权限修饰符

本类 同一个包下 不同包下的子类 不同包下的无关类

private Y

默认 Y Y

protected Y Y Y

public Y Y Y Y

(2)这四种权限修饰符在任意时刻只能出现一种。

public class Demo {}

5:常见的修饰符(理解)

(1)分类：

权限修饰符：private,默认,protected,public

状态修饰符：static,final

抽象修饰符：abstract

(2)常见的类及其组成的修饰

类：

默认,public,final,abstract

常用的：public

成员变量：

private,默认,protected,public,static,final

常用的：private

构造方法：

private,默认,protected,public

常用的：public

成员方法：

private,默认,protected,public,static,final,abstract

常用的：public

(3)另外比较常见的：

public static final int X = 10;

public static void show() {}

public final void show() {}

public abstract void show();

6:内部类(理解)

(1)把类定义在另一个类的内部，该类就被称为内部类。

举例：把类B定义在类A中，类B就被称为内部类。

(2)内部类的访问规则

A:可以直接访问外部类的成员，包括私有

B:外部类要想访问内部类成员，必须创建对象

(3)内部类的分类

A:成员内部类

B:局部内部类

(4)成员内部类

A:private 为了数据的安全性

B:static 为了访问的方便性

成员内部类不是静态的：

外部类名.内部类名 对象名 = new 外部类名.new 内部类名();

成员内部类是静态的：

外部类名.内部类名 对象名 = new 外部类名.内部类名();

(5)成员内部类的面试题(填空)

30,20,10

class Outer {

public int num = 10;

class Inner {

public int num = 20;

public viod show() {

int num = 30;

System.out.println(num);

System.out.println(this.num);

System.out.println(Outer.this.num);

}

}

}

(6)局部内部类

A:局部内部类访问局部变量必须加final修饰。

B:为什么呢?

因为局部变量使用完毕就消失，而堆内存的数据并不会立即消失。

所以，堆内存还是用该变量，而改变量已经没有了。

为了让该值还存在，就加final修饰。

通过反编译工具我们看到了，加入final后，堆内存直接存储的是值，而不是变量名。

(7)匿名内部类(掌握)

A:是局部内部类的简化形式

B:前提

存在一个类或者接口

C:格式:

new 类名或者接口名() {

重写方法;

}

D:本质：

其实是继承该类或者实现接口的子类匿名对象

(8)匿名内部类在开发中的使用

我们在开发的时候，会看到抽象类，或者接口作为参数。

而这个时候，我们知道实际需要的是一个子类对象。

如果该方法仅仅调用一次，我们就可以使用匿名内部类的格式简化。

interface Person {

public abstract void study();

}

class PersonDemo {

public void method(Person p) {

p.study();

}

}

class PersonTest {

public static void main(String[] args) {

PersonDemo pd = new PersonDemo();

pd.method(new Person() {

public void study() {

System.out.println("好好学习，天天向上");

}

});

}

}

(9)匿名内部类的面试题(补齐代码)

interface Inter {

void show();

}

class Outer {

//补齐代码

public static Inter method() {

return new Inter() {

public void show() {

System.out.println("HelloWorld");

}

};

}

}

class OuterDemo {

public static void main(String[] args) {

Outer.method().show(); //"HelloWorld"

}

}

2:API的概述(了解)

(1)应用程序编程接口。

(2)就是JDK提供给我们的一些提高编程效率的java类。

3:Object类(掌握)

(1)Object是类层次结构的根类，所有的类都直接或者间接的继承自Object类。

(2)Object类的构造方法有一个，并且是无参构造

这其实就是理解当时我们说过，子类构造方法默认访问父类的构造是无参构造

(3)要掌握的方法：

A:toString()

返回对象的字符串表示，默认是由类的全路径+'@'+哈希值的十六进制表示。

这个表示其实是没有意义的，一般子类都会重写该方法。

如何重写呢?过程我也讲解过了，基本上就是要求信息简单明了。

但是最终还是自动生成。

B:equals()

比较两个对象是否相同。默认情况下，比较的是地址值是否相同。

而比较地址值是没有意义的，所以，一般子类也会重写该方法。

重写过程，我也详细的讲解和分析了。

但是最终还是自动生成。

(4)要了解的方法：

A:hashCode() 返回对象的哈希值。不是实际地址值，可以理解为地址值。

B:getClass() 返回对象的字节码文件对象，反射中我们会详细讲解

C:finalize() 用于垃圾回收，在不确定的时间

D:clone() 可以实现对象的克隆，包括成员变量的数据复制，但是它和两个引用指向同一个对象是有区别的。

(5)两个注意问题；

A:直接输出一个对象名称，其实默认调用了该对象的toString()方法。

B:面试题

==和equals()的区别?

A:==

基本类型：比较的是值是否相同

引用类型：比较的是地址值是否相同

B:equals()

只能比较引用类型。默认情况下，比较的是地址值是否相同。

但是，我们可以根据自己的需要重写该方法。

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2:String类的概述和使用(掌握)

(1)多个字符组成的一串数据。

其实它可以和字符数组进行相互转换。

(2)构造方法：

A:public String()

B:public String(byte[] bytes)

C:public String(byte[] bytes,int offset,int length)

D:public String(char[] value)

E:public String(char[] value,int offset,int count)

F:public String(String original)

下面的这一个虽然不是构造方法，但是结果也是一个字符串对象

G:String s = "hello";

(3)字符串的特点

A:字符串一旦被赋值，就不能改变。

注意：这里指的是字符串的内容不能改变，而不是引用不能改变。

B:字面值作为字符串对象和通过构造方法创建对象的不同

String s = new String("hello");和String s = "hello"的区别?

(4)字符串的面试题(看程序写结果)

A:==和equals()

String s1 = new String("hello");

String s2 = new String("hello");

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

System.out.println(s1.equals(s2));// true

String s3 = new String("hello");

String s4 = "hello";

System.out.println(s3 == s4);// false

System.out.println(s3.equals(s4));// true

String s5 = "hello";

String s6 = "hello";

System.out.println(s5 == s6);// true

System.out.println(s5.equals(s6));// true

B:字符串的拼接

String s1 = "hello";

String s2 = "world";

String s3 = "helloworld";

System.out.println(s3 == s1 + s2);// false

System.out.println(s3.equals((s1 + s2)));// true

System.out.println(s3 == "hello" + "world");// false 这个我们错了，应该是true

System.out.println(s3.equals("hello" + "world"));// true

(5)字符串的功能(自己补齐方法中文意思)

A:判断功能

boolean equals(Object obj)

boolean equalsIgnoreCase(String str)

boolean contains(String str)

boolean startsWith(String str)

boolean endsWith(String str)

boolean isEmpty()

B:获取功能

int length()

char charAt(int index)

int indexOf(int ch)

int indexOf(String str)

int indexOf(int ch,int fromIndex)

int indexOf(String str,int fromIndex)

String substring(int start)

String substring(int start,int end)

C:转换功能

byte[] getBytes()

char[] toCharArray()

static String valueOf(char[] chs)

static String valueOf(int i)

String toLowerCase()

String toUpperCase()

String concat(String str)

D:其他功能

a:替换功能

String replace(char old,char new)

String replace(String old,String new)

b:去空格功能

String trim()

c:按字典比较功能

int compareTo(String str)

int compareToIgnoreCase(String str)

1:StringBuffer(掌握)

(1)用字符串做拼接，比较耗时并且也耗内存，而这种拼接操作又是比较常见的，为了解决这个问题，Java就提供了

一个字符串缓冲区类。StringBuffer供我们使用。

(2)StringBuffer的构造方法

A:StringBuffer()

B:StringBuffer(int size)

C:StringBuffer(String str)

(3)StringBuffer的常见功能(自己补齐方法的声明和方法的解释)

A:添加功能

B:删除功能

C:替换功能

D:反转功能

E:截取功能(注意这个返回值)

(4)StringBuffer的练习(做一遍)

A:String和StringBuffer相互转换

String -- StringBuffer

构造方法

StringBuffer -- String

toString()方法

B:字符串的拼接

C:把字符串反转

D:判断一个字符串是否对称

(5)面试题

小细节：

StringBuffer：同步的，数据安全，效率低。

StringBuilder：不同步的，数据不安全，效率高。

A:String,StringBuffer,StringBuilder的区别

B:StringBuffer和数组的区别?

(6)注意的问题：

String作为形式参数，StringBuffer作为形式参数。

2:数组高级以及Arrays(掌握)

(1)排序

A:冒泡排序

相邻元素两两比较，大的往后放，第一次完毕，最大值出现在了最大索引处。同理，其他的元素就可以排好。

public static void bubbleSort(int[] arr) {

for(int x=0; x<arr.length-1; x++) {

for(int y=0; y<arr.length-1-x; y++) {

if(arr[y] > arr[y+1]) {

int temp = arr[y];

arr[y] = arr[y+1];

arr[y+1] = temp;

}

}

}

}

B:选择排序

把0索引的元素，和索引1以后的元素都进行比较，第一次完毕，最小值出现在了0索引。同理，其他的元素就可以排好。

public static void selectSort(int[] arr) {

for(int x=0; x<arr.length-1; x++) {

for(int y=x+1; y<arr.length; y++) {

if(arr[y] < arr[x]) {

int temp = arr[x];

arr[x] = arr[y];

arr[y] = temp;

}

}

}

}

(2)查找

A:基本查找

针对数组无序的情况

public static int getIndex(int[] arr,int value) {

int index = -1;

for(int x=0; x<arr.length; x++) {

if(arr[x] == value) {

index = x;

break;

}

}

return index;

}

B:二分查找(折半查找)

针对数组有序的情况(千万不要先排序，在查找)

public static int binarySearch(int[] arr,int value) {

int min = 0;

int max = arr.length-1;

int mid = (min+max)/2;

while(arr[mid] != value) {

if(arr[mid] > value) {

max = mid - 1;

}else if(arr[mid] < value) {

min = mid + 1;

}

if(min > max) {

return -1;

}

mid = (min+max)/2;

}

return mid;

}

(3)Arrays工具类

A:是针对数组进行操作的工具类。包括排序和查找等功能。

B:要掌握的方法(自己补齐方法)

把数组转成字符串：

排序：

二分查找：

(4)Arrays工具类的源码解析

(5)把字符串中的字符进行排序

举例：

"edacbgf"

得到结果

"abcdefg"

3:Integer(掌握)

(1)为了让基本类型的数据进行更多的操作，Java就为每种基本类型提供了对应的包装类类型

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

boolean Boolean

(2)Integer的构造方法

A:Integer i = new Integer(100);

B:Integer i = new Integer("100");

注意：这里的字符串必须是由数字字符组成

(3)String和int的相互转换

A:String -- int

Integer.parseInt("100");

B:int -- String

String.valueOf(100);

(4)其他的功能(了解)

进制转换

(5)JDK5的新特性

自动装箱 基本类型--引用类型

自动拆箱 引用类型--基本类型

把下面的这个代码理解即可：

Integer i = 100;

i += 200;

(6)面试题

-128到127之间的数据缓冲池问题

4:Character(了解)

(1)Character构造方法

Character ch = new Character('a');

(2)要掌握的方法：(自己补齐)

A:判断给定的字符是否是大写

B:判断给定的字符是否是小写

C:判断给定的字符是否是数字字符

D:把给定的字符转成大写

E:把给定的字符转成小写

(3)案例：

统计字符串中大写，小写及数字字符出现的次数

1:正则表达式(理解)

(1)就是符合一定规则的字符串

(2)常见规则

A:字符

x 字符 x。举例：'a'表示字符a

\\ 反斜线字符。

\n 新行（换行）符 ('\u000A')

\r 回车符 ('\u000D')

B:字符类

[abc] a、b 或 c（简单类）

[^abc] 任何字符，除了 a、b 或 c（否定）

[a-zA-Z] a到 z 或 A到 Z，两头的字母包括在内（范围）

[0-9] 0到9的字符都包括

C:预定义字符类

. 任何字符。我的就是.字符本身，怎么表示呢? \.

\d 数字：[0-9]

\w 单词字符：[a-zA-Z\_0-9]

在正则表达式里面组成单词的东西必须有这些东西组成

D:边界匹配器

^ 行的开头

$ 行的结尾

\b 单词边界

就是不是单词字符的地方。

举例：hello world?haha;xixi

E:Greedy 数量词

X? X，一次或一次也没有

X\* X，零次或多次

X+ X，一次或多次

X{n} X，恰好 n 次

X{n,} X，至少 n 次

X{n,m} X，至少 n 次，但是不超过 m 次

(3)常见功能：(分别用的是谁呢?)

A:判断功能

String类的public boolean matches(String regex)

B:分割功能

String类的public String[] split(String regex)

C:替换功能

String类的public String replaceAll(String regex,String replacement)

D:获取功能

Pattern和Matcher

Pattern p = Pattern.compile("a\*b");

Matcher m = p.matcher("aaaaab");

find():查找存不存在

group():获取刚才查找过的数据

(4)案例

A:判断电话号码和邮箱

B:按照不同的规则分割数据

C:把论坛中的数字替换为\*

D:获取字符串中由3个字符组成的单词

2:Math(掌握)

(1)针对数学运算进行操作的类

(2)常见方法(自己补齐)

A:绝对值

B:向上取整

C:向下取整

D:两个数据中的大值

E:a的b次幂

F:随机数

G:四舍五入

H:正平方根

(3)案例：

A:猜数字小游戏

B:获取任意范围的随机数

3:Random(理解)

(1)用于产生随机数的类

(2)构造方法:

A:Random() 默认种子，每次产生的随机数不同

B:Random(long seed) 指定种子，每次种子相同，随机数就相同

(3)成员方法:

A:int nextInt() 返回int范围内的随机数

B:int nextInt(int n) 返回[0,n)范围内的随机数

4:System(掌握)

(1)系统类,提供了一些有用的字段和方法

(2)成员方法(自己补齐)

A:运行垃圾回收器

B:退出jvm

C:获取当前时间的毫秒值

D:数组复制

5:BigInteger(理解)

(1)针对大整数的运算

(2)构造方法

A:BigInteger(String s)

(3)成员方法(自己补齐)

A:加

B:减

C:乘

D:除

E:商和余数

6:BigDecimal(理解)

(1)浮点数据做运算，会丢失精度。所以，针对浮点数据的操作建议采用BigDecimal。(金融相关的项目)

(2)构造方法

A:BigDecimal(String s)

(3)成员方法：

A:加

B:减

C:乘

D:除

E:自己保留小数几位

7:Date/DateFormat(掌握)

(1)Date是日期类，可以精确到毫秒。

A:构造方法

Date()

Date(long time)

B:成员方法

getTime()

setTime(long time)

C:日期和毫秒值的相互转换

案例：你来到这个世界多少天了?

(2)DateFormat针对日期进行格式化和针对字符串进行解析的类，但是是抽象类，所以使用其子类SimpleDateFormat

A:SimpleDateFormat(String pattern) 给定模式

yyyy-MM-dd HH:mm:ss

B:日期和字符串的转换

a:Date -- String

format()

b:String -- Date

parse()

C:案例：

制作了一个针对日期操作的工具类。

8:Calendar(掌握)

(1)日历类，封装了所有的日历字段值，通过统一的方法根据传入不同的日历字段可以获取值。

(2)如何得到一个日历对象呢?

Calendar rightNow = Calendar.getInstance();

本质返回的是子类对象

(3)成员方法

A:根据日历字段得到对应的值

B:根据日历字段和一个正负数确定是添加还是减去对应日历字段的值

C:设置日历对象的年月日

(4)案例：

计算任意一年的2月份有多少天?

1:对象数组(掌握)

(1)数组既可以存储基本数据类型，也可以存储引用类型。它存储引用类型的时候的数组就叫对象数组。

(2)案例：

用数组存储5个学生对象，并遍历数组。

2:集合(Collection)(掌握)

(1)集合的由来?

我们学习的是Java -- 面向对象 -- 操作很多对象 -- 存储 -- 容器(数组和StringBuffer) -- 数组

而数组的长度固定，所以不适合做变化的需求，Java就提供了集合供我们使用。

(2)集合和数组的区别?

A:长度区别

数组固定

集合可变

B:内容区别

数组可以是基本类型，也可以是引用类型

集合只能是引用类型

C:元素内容

数组只能存储同一种类型

集合可以存储不同类型(其实集合一般存储的也是同一种类型)

(3)集合的继承体系结构?

由于需求不同，Java就提供了不同的集合类。这多个集合类的数据结构不同，但是它们都是要提供存储和遍历功能的，

我们把它们的共性不断的向上提取，最终就形成了集合的继承体系结构图。

Collection

|--List

|--ArrayList

|--Vector

|--LinkedList

|--Set

|--HashSet

|--TreeSet

(4)Collection的功能概述(自己补齐)

A:添加功能

B:删除功能

C:判断功能

D:获取功能

E:长度功能

F:交集(了解)

G:把集合转数组(了解)

(5)Collection集合的遍历

A:把集合转数组(了解)

B:迭代器(集合专用方式)

(6)迭代器

A:是集合的获取元素的方式。

B:是依赖于集合而存在的。

C:迭代器的原理和源码。

a:为什么定义为了一个接口而不是实现类?

b:看了看迭代器的内部类实现。

(7)Collection集合的案例(遍历方式 迭代器)

集合的操作步骤：

A:创建集合对象

B:创建元素对象

C:把元素添加到集合

D:遍历集合

A:存储字符串并遍历

import java.util.Collection;

import java.util.ArrayList;

import java.util.Iterator;

public class CollectionDemo {

public static void main(String[] args) {

//创建集合对象

Collection c = new ArrayList();

//创建并添加元素

c.add("hello");

c.add("world");

c.add("java");

//遍历集合

Iterator it = c.iterator();

while(it.hasNext()) {

String s =(String) it.next();

System.out.println(s);

}

}

}

B:存储自定义对象并遍历

public class Student {

private String name;

private int age;

public Student(){}

public Student(String name,int age) {

this.name = name;

this.age = age;

}

//getXxx()/setXxx()

}

import java.util.Collection;

import java.util.ArrayList;

import java.util.Iterator;

public class StudentDemo {

public static void main(String[] args) {

//创建集合对象

Collection c = new ArrayList();

//创建学生对象

Student s1 = new Student("林青霞",27);

Student s2 = new Student("风清扬",30);

Student s3 = new Student("刘意",30);

Student s4 = new Student("武鑫",25);

Student s5 = new Student("刘晓曲",16);

//添加元素

c.add(s1);

c.add(s2);

c.add(s3);

c.add(s4);

c.add(s5);

//遍历集合

Iterator it = c.iterator();

while(it.hasNext()) {

Student s = (Student)it.next();

System.out.println(s.getName()+"---"+s.getAge());

}

}

}

3:集合(List)(掌握)

(1)List是Collection的子接口

特点：有序(存储顺序和取出顺序一致)，可重复。

(2)List的特有功能：(自己补齐)

A:添加功能

B:删除功能

C:获取功能

D:迭代器功能

E:修改功能

(3)List集合的特有遍历功能

A:由size()和get()结合。

B:代码演示

//创建集合对象

List list = new ArrayList();

//创建并添加元素

list.add("hello");

list.add("world");

list.add("java");

//遍历集合

Iterator it = list.iterator();

while(it.hasNext()) {

String s =(String) it.next();

System.out.println(s);

}

System.out.println("----------");

for(int x=0; x<list.size(); x++) {

String s =(String) list.get(x);

System.out.println(s);

}

(4)列表迭代器的特有功能；(了解)

可以逆向遍历，但是要先正向遍历，所以无意义，基本不使用。

(5)并发修改异常

A:出现的现象

迭代器遍历集合，集合修改集合元素

B:原因

迭代器是依赖于集合的，而集合的改变迭代器并不知道。

C:解决方案

a:迭代器遍历，迭代器修改(ListIterator)

元素添加在刚才迭代的位置

b:集合遍历，集合修改(size()和get())

元素添加在集合的末尾

(6)常见数据结构

A:栈 先进后出

B:队列 先进先出

C:数组 查询快，增删慢

D:链表 查询慢，增删快

(7)List的子类特点(面试题)

ArrayList

底层数据结构是数组，查询快，增删慢。

线程不安全，效率高。

Vector

底层数据结构是数组，查询快，增删慢。

线程安全，效率低。

LinkedList

底层数据结构是链表，查询慢，增删快。

线程不安全，效率高。

到底使用谁呢?看需求?

分析：

要安全吗?

要：Vector(即使要，也不使用这个，后面再说)

不要：ArrayList或者LinkedList

查询多；ArrayList

增删多：LinkedList

什么都不知道，就用ArrayList。

(8)List集合的案例(遍历方式 迭代器和普通for)

A:存储字符串并遍历

B:存储自定义对象并遍历

1:List的子类(掌握)

(1)List的子类特点

ArrayList:

底层数据结构是数组，查询快，增删慢

线程不安全，效率高

Vector:

底层数据结构是数组，查询快，增删慢

线程安全，效率低

LinkedList:

底层数据结构是链表，查询慢，增删快

线程不安全，效率高

(2)ArrayList

A:没有特有功能需要学习

B:案例

a:ArrayList存储字符串并遍历

b:ArrayList存储自定义对象并遍历

(3)Vector

A:有特有功能

a:添加

public void addElement(E obj) -- add()

b:获取

public E elementAt(int index) -- get()

public Enumeration<E> elements() -- iterator()

B:案例

a:Vector存储字符串并遍历

b:Vector存储自定义对象并遍历

(4)LinkedList

A:有特有功能

a:添加

addFirst()

addLast()

b:删除

removeFirst()

removeLast()

c:获取

getFirst()

getLast()

B:案例

a:LinkedList存储字符串并遍历

b:LinkedList存储自定义对象并遍历

(5)案例：

A:去除集合中的多个字符串的重复元素

如果字符串的内容相同，即为重复元素

B:去除集合中的多个自定义对象的重复元素

如果自定义对象的成员变量值都相同，即为重复元素

C:用LinkedList模拟一个栈数据结构的集合类，并测试。

你要定义一个集合类，只不过内部可以使用LinkedList来实现。

2:泛型(掌握)

(1)泛型概述

是一种把明确类型的工作推迟到创建对象或者调用方法的时候才去明确的特殊的类型。

(2)格式：

<数据类型>

注意：该数据类型只能是引用类型。

(3)好处：

A:把运行时期的问题提前到了编译期间

B:避免了强制类型转换

C:优化了程序设计，解决了黄色警告线问题，让程序更安全

(4)泛型的前世今生

A:泛型的由来

Object类型作为任意类型的时候，在向下转型的时候，会隐含一个转型问题

B:泛型类

C:泛型方法

D:泛型接口

E:泛型高级通配符

?

? extends E

? super E

(5)我们在哪里使用呢?

一般是在集合中使用。

3:增强for循环(掌握)

(1)是for循环的一种

(2)格式：

for(元素的数据类型 变量名 : 数组或者Collection集合的对象) {

使用该变量即可，该变量其实就是数组或者集合中的元素。

}

(3)好处：

简化了数组和集合的遍历

(4)弊端

增强for循环的目标不能为null。建议在使用前，先判断是否为null。

4:静态导入(了解)

(1)可以导入到方法级别的导入

(2)格式：

import static 包名....类名.方法名;

(3)注意事项：

A:方法必须是静态的

B:如果多个类下有同名的方法，就不好区分了，还得加上前缀。

所以一般我们并不使用静态导入，但是一定要能够看懂。

5:可变参数(掌握)

(1)如果我们在写方法的时候，参数个数不明确，就应该定义可变参数。

(2)格式：

修饰符 返回值类型 方法名(数据类型... 变量) {}

注意：

A:该变量其实是一个数组名

B:如果一个方法有多个参数，并且有可变参数，可变参数必须在最后

(3)Arrays工具类的一个方法

asList()把数组转成集合。

注意：这个集合的长度不能改变。

2:Set集合(理解)

(1)Set集合的特点

无序,唯一

(2)HashSet集合(掌握)

A:底层数据结构是哈希表(是一个元素为链表的数组)

B:哈希表底层依赖两个方法：hashCode()和equals()

执行顺序：

首先比较哈希值是否相同

相同：继续执行equals()方法

返回true：元素重复了，不添加

返回false：直接把元素添加到集合

不同：就直接把元素添加到集合

C:如何保证元素唯一性的呢?

由hashCode()和equals()保证的

D:开发的时候，代码非常的简单，自动生成即可。

E:HashSet存储字符串并遍历

F:HashSet存储自定义对象并遍历(对象的成员变量值相同即为同一个元素)

(3)TreeSet集合

A:底层数据结构是红黑树(是一个自平衡的二叉树)

B:保证元素的排序方式

a:自然排序(元素具备比较性)

让元素所属的类实现Comparable接口

b:比较器排序(集合具备比较性)

让集合构造方法接收Comparator的实现类对象

C:把我们讲过的代码看一遍即可

(4)案例：

A:获取无重复的随机数

B:键盘录入学生按照总分从高到底输出

3:Collection集合总结(掌握)

Collection

|--List 有序,可重复

|--ArrayList

底层数据结构是数组，查询快，增删慢。

线程不安全，效率高

|--Vector

底层数据结构是数组，查询快，增删慢。

线程安全，效率低

|--LinkedList

底层数据结构是链表，查询慢，增删快。

线程不安全，效率高

|--Set 无序,唯一

|--HashSet

底层数据结构是哈希表。

如何保证元素唯一性的呢?

依赖两个方法：hashCode()和equals()

开发中自动生成这两个方法即可

|--LinkedHashSet

底层数据结构是链表和哈希表

由链表保证元素有序

由哈希表保证元素唯一

|--TreeSet

底层数据结构是红黑树。

如何保证元素排序的呢?

自然排序

比较器排序

如何保证元素唯一性的呢?

根据比较的返回值是否是0来决定

4:针对Collection集合我们到底使用谁呢?(掌握)

唯一吗?

是：Set

排序吗?

是：TreeSet

否：HashSet

如果你知道是Set，但是不知道是哪个Set，就用HashSet。

否：List

要安全吗?

是：Vector

否：ArrayList或者LinkedList

查询多：ArrayList

增删多：LinkedList

如果你知道是List，但是不知道是哪个List，就用ArrayList。

如果你知道是Collection集合，但是不知道使用谁，就用ArrayList。

如果你知道用集合，就用ArrayList。

5:在集合中常见的数据结构(掌握)

ArrayXxx:底层数据结构是数组，查询快，增删慢

LinkedXxx:底层数据结构是链表，查询慢，增删快

HashXxx:底层数据结构是哈希表。依赖两个方法：hashCode()和equals()

TreeXxx:底层数据结构是二叉树。两种方式排序：自然排序和比较器排序

1:Map(掌握)

(1)将键映射到值的对象。一个映射不能包含重复的键；每个键最多只能映射到一个值。

(2)Map和Collection的区别?

A:Map 存储的是键值对形式的元素，键唯一，值可以重复。

B:Collection 存储的是单独出现的元素，子接口Set元素唯一，子接口List元素可重复。

(3)Map接口功能概述

A:添加功能

B:删除功能

C:判断功能

D:获取功能

E:长度功能

(4)Map集合的遍历

A:键找值

a:获取所有键的集合

b:遍历键的集合,得到每一个键

c:根据键到集合中去找值

B:键值对对象找键和值

a:获取所有的键值对对象的集合

b:遍历键值对对象的集合，获取每一个键值对对象

c:根据键值对对象去获取键和值

代码体现：

Map<String,String> hm = new HashMap<String,String>();

hm.put("it002","hello");

hm.put("it003","world");

hm.put("it001","java");

//方式1 键找值

Set<String> set = hm.keySet();

for(String key : set) {

String value = hm.get(key);

System.out.println(key+"---"+value);

}

//方式2 键值对对象找键和值

Set<Map.Entry<String,String>> set2 = hm.entrySet();

for(Map.Entry<String,String> me : set2) {

String key = me.getKey();

String value = me.getValue();

System.out.println(key+"---"+value);

}

(5)HashMap集合的练习

A:HashMap<String,String>

B:HashMap<Integer,String>

C:HashMap<String,Student>

D:HashMap<Student,String>

(6)TreeMap集合的练习

A:TreeMap<String,String>

B:TreeMap<Student,String>

(7)案例

A:统计一个字符串中每个字符出现的次数

B:集合的嵌套遍历

a:HashMap嵌套HashMap

b:HashMap嵌套ArrayList

c:ArrayList嵌套HashMap

d:多层嵌套

2:Collections(理解)

(1)是针对集合进行操作的工具类

(2)面试题：Collection和Collections的区别

A:Collection 是单列集合的顶层接口，有两个子接口List和Set

B:Collections 是针对集合进行操作的工具类，可以对集合进行排序和查找等

(3)常见的几个小方法：

A:public static <T> void sort(List<T> list)

B:public static <T> int binarySearch(List<?> list,T key)

C:public static <T> T max(Collection<?> coll)

D:public static void reverse(List<?> list)

E:public static void shuffle(List<?> list)

1:异常(理解)

(1)程序出现的不正常的情况。

(2)异常的体系

Throwable

|--Error 严重问题，我们不处理。

|--Exception

|--RuntimeException 运行期异常，我们需要修正代码

|--非RuntimeException 编译期异常，必须处理的，否则程序编译不通过

(3)异常的处理：

A:JVM的默认处理

把异常的名称,原因,位置等信息输出在控制台，但是呢程序不能继续执行了。

B:自己处理

a:try...catch...finally

自己编写处理代码,后面的程序可以继续执行

b:throws

把自己处理不了的，在方法上声明，告诉调用者，这里有问题

B:throw和throws是的区别

throw:

在方法体中,后面跟的是异常对象名,并且只能是一个

throw抛出的是一个异常对象，说明这里肯定有一个异常产生了

throws:

在方法声明上,后面跟的是异常的类名,可以是多个

throws是声明方法有异常，是一种可能性，这个异常并不一定会产生

(5)finally关键字及其面试题

A:finally用于释放资源，它的代码永远会执行。特殊情况：在执行到finally之前jvm退出了

C:异常处理的变形

try...catch...finally

try...catch...

try...catch...catch...

try...catch...catch...fianlly

try...finally

(6)自定义异常

继承自Exception或者RuntimeException,只需要提供无参构造和一个带参构造即可

(7)异常的注意实现

A:父的方法有异常抛出,子的重写方法在抛出异常的时候必须要小于等于父的异常

B:父的方法没有异常抛出,子的重写方法不能有异常抛出

C:父的方法抛出多个异常,子的重写方法必须比父少或者小

2:File(掌握)

(1)IO流操作中大部分都是对文件的操作，所以Java就提供了File类供我们来操作文件

(2)构造方法

A:File file = new File("e:\\demo\\a.txt");

B:File file = new File("e:\\demo","a.txt");

C:File file = new File("e:\\demo");

File file2 = new File(file,"a.txt");

(3)File类的功能(自己补齐)

A:创建功能

B:删除功能

C:重命名功能

D:判断功能

E:获取功能

F:高级获取功能

G:过滤器功能

1:递归(理解)

(1)方法定义中调用方法本身的现象

(2)递归的注意事项；

A:要有出口，否则就是死递归

B:次数不能过多，否则内存溢出

C:构造方法不能递归使用

2:IO流(掌握)

(1)IO用于在设备间进行数据传输的操作

(2)分类：

A:流向

输入流 读取数据

输出流 写出数据

B:数据类型

字节流

字节输入流

字节输出流

字符流

字符输入流

字符输出流

注意：

a:如果我们没有明确说明按照什么分，默认按照数据类型分。

b:除非文件用windows自带的记事本打开我们能够读懂，才采用字符流，否则建议使用字节流。

(3)FileOutputStream写出数据

A:操作步骤

a:创建字节输出流对象

b:调用write()方法

c:释放资源

B:代码体现：

FileOutputStream fos = new FileOutputStream("fos.txt");

fos.write("hello".getBytes());

fos.close();

C:要注意的问题?

a:创建字节输出流对象做了几件事情?

b:为什么要close()?

c:如何实现数据的换行?

d:如何实现数据的追加写入?

(4)FileInputStream读取数据

A:操作步骤

a:创建字节输入流对象

b:调用read()方法

c:释放资源

B:代码体现：

FileInputStream fis = new FileInputStream("fos.txt");

//方式1

int by = 0;

while((by=fis.read())!=-1) {

System.out.print((char)by);

}

//方式2

byte[] bys = new byte[1024];

int len = 0;

while((len=fis.read(bys))!=-1) {

System.out.print(new String(bys,0,len));

}

fis.close();

(5)案例：2种实现

A:复制文本文件

B:复制图片

C:复制视频

(6)字节缓冲区流

A:BufferedOutputStream

B:BufferedInputStream

(7)案例：4种实现

A:复制文本文件

B:复制图片

C:复制视频

3:字符流

IO流分类

字节流：

InputStream

FileInputStream

BufferedInputStream

OutputStream

FileOutputStream

BufferedOutputStream

字符流：

Reader

FileReader

BufferedReader

Writer

FileWriter

BufferedWriter

1:字符流(掌握)

(1)字节流操作中文数据不是特别的方便，所以就出现了转换流。

转换流的作用就是把字节流转换字符流来使用。

(2)转换流其实是一个字符流

字符流 = 字节流 + 编码表

(3)编码表

A:就是由字符和对应的数值组成的一张表

B:常见的编码表

ASCII

ISO-8859-1

GB2312

GBK

GB18030

UTF-8

C:字符串中的编码问题

编码

String -- byte[]

解码

byte[] -- String

(4)IO流中的编码问题

A:OutputStreamWriter

OutputStreamWriter(OutputStream os):默认编码，GBK

OutputStreamWriter(OutputStream os,String charsetName):指定编码。

B:InputStreamReader

InputStreamReader(InputStream is):默认编码，GBK

InputStreamReader(InputStream is,String charsetName):指定编码

C:编码问题其实很简单

编码只要一致即可

(5)字符流

Reader

|--InputStreamReader

|--FileReader

|--BufferedReader

Writer

|--OutputStreamWriter

|--FileWriter

|--BufferedWriter

(6)复制文本文件(5种方式)

2:IO流小结(掌握)

IO流

|--字节流

|--字节输入流

InputStream

int read():一次读取一个字节

int read(byte[] bys):一次读取一个字节数组

|--FileInputStream

|--BufferedInputStream

|--字节输出流

OutputStream

void write(int by):一次写一个字节

void write(byte[] bys,int index,int len):一次写一个字节数组的一部分

|--FileOutputStream

|--BufferedOutputStream

|--字符流

|--字符输入流

Reader

int read():一次读取一个字符

int read(char[] chs):一次读取一个字符数组

|--InputStreamReader

|--FileReader

|--BufferedReader

String readLine():一次读取一个字符串

|--字符输出流

Writer

void write(int ch):一次写一个字符

void write(char[] chs,int index,int len):一次写一个字符数组的一部分

|--OutputStreamWriter

|--FileWriter

|--BufferedWriter

void newLine():写一个换行符

void write(String line):一次写一个字符串

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2:数据操作流(操作基本类型数据的流)(理解)

(1)可以操作基本类型的数据

(2)流对象名称

DataInputStream

DataOutputStream

3:内存操作流(理解)

(1)有些时候我们操作完毕后，未必需要产生一个文件，就可以使用内存操作流。

(2)三种

A:ByteArrayInputStream,ByteArrayOutputStream

B:CharArrayReader,CharArrayWriter

C:StringReader,StringWriter

4:打印流(掌握)

(1)字节打印流，字符打印流

(2)特点：

A:只操作目的地,不操作数据源

B:可以操作任意类型的数据

C:如果启用了自动刷新，在调用println()方法的时候，能够换行并刷新

D:可以直接操作文件

问题：哪些流可以直接操作文件呢?

看API，如果其构造方法能够同时接收File和String类型的参数，一般都是可以直接操作文件的

(3)复制文本文件

BufferedReader br = new BufferedReader(new FileReader("a.txt"));

PrintWriter pw = new PrintWriter(new FileWriter("b.txt"),true);

String line = null;

while((line=br.readLine())!=null) {

pw.println(line);

}

pw.close();

br.close();

5:标准输入输出流(理解)

(1)System类下面有这样的两个字段

in 标准输入流

out 标准输出流

(2)三种键盘录入方式

A:main方法的args接收参数

B:System.in通过BufferedReader进行包装

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

C:Scanner

Scanner sc = new Scanner(System.in);

(3)输出语句的原理和如何使用字符流输出数据

A:原理

System.out.println("helloworld");

PrintStream ps = System.out;

ps.println("helloworld");

B:把System.out用字符缓冲流包装一下使用

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));

6:随机访问流(理解)

(1)可以按照文件指针的位置写数据和读数据。

(2)案例：

A:写数据

B:读数据

C:获取和改变文件指针的位置

7:合并流(理解)

(1)把多个输入流的数据写到一个输出流中。

(2)构造方法：

A:SequenceInputStream(InputStream s1, InputStream s2)

B:SequenceInputStream(Enumeration<? extends InputStream> e)

8:序列化流(理解)

(1)可以把对象写入文本文件或者在网络中传输

(2)如何实现序列化呢?

让被序列化的对象所属类实现序列化接口。

该接口是一个标记接口。没有功能需要实现。

(3)注意问题：

把数据写到文件后，在去修改类会产生一个问题。

如何解决该问题呢?

在类文件中，给出一个固定的序列化id值。

而且，这样也可以解决黄色警告线问题

(4)面试题：

什么时候序列化?

如何实现序列化?

什么是反序列化?

9:Properties(理解)

(1)是一个集合类，Hashtable的子类

(2)特有功能

A:public Object setProperty(String key,String value)

B:public String getProperty(String key)

C:public Set<String> stringPropertyNames()

(3)和IO流结合的方法

把键值对形式的文本文件内容加载到集合中

public void load(Reader reader)

public void load(InputStream inStream)

把集合中的数据存储到文本文件中

public void store(Writer writer,String comments)

public void store(OutputStream out,String comments)

(4)案例：

A:根据给定的文件判断是否有键为"lisi"的，如果有就修改其值为100

B:写一个程序实现控制猜数字小游戏程序不能玩超过5次

10:NIO(了解)

(1)JDK4出现的NIO，对以前的IO操作进行了优化，提供了效率。但是大部分我们看到的还是以前的IO

(2)JDK7的NIO的使用

Path:路径

Paths:通过静态方法返回一个路径

Files:提供了常见的功能

复制文本文件

把集合中的数据写到文本文件

1:多线程(理解)

(1)多线程：一个应用程序有多条执行路径

进程：正在执行的应用程序

线程：进程的执行单元，执行路径

单线程：一个应用程序只有一条执行路径

多线程：一个应用程序有多条执行路径

多进程的意义?

提高CPU的使用率

多线程的意义?

提高应用程序的使用率

(2)Java程序的运行原理及JVM的启动是多线程的吗?

A:Java命令去启动JVM，JVM会启动一个进程，该进程会启动一个主线程。

B:JVM的启动是多线程的，因为它最低有两个线程启动了，主线程和垃圾回收线程。

(3)多线程的实现方案(自己补齐步骤及代码 掌握)

A:继承Thread类

B:实现Runnable接口

(4)线程的调度和优先级问题

A:线程的调度

a:分时调度

b:抢占式调度 (Java采用的是该调度方式)

B:获取和设置线程优先级

a:默认是5

b:范围是1-10

(5)线程的控制(常见方法)

A:休眠线程

B:加入线程

C:礼让线程

D:后台线程

E:终止线程(掌握)

(6)线程的生命周期(参照 线程生命周期图解.bmp)

A:新建

B:就绪

C:运行

D:阻塞

E:死亡

(9)多线程安全问题的原因(也是我们以后判断一个程序是否有线程安全问题的依据)

A:是否有多线程环境

B:是否有共享数据

C:是否有多条语句操作共享数据

(10)同步解决线程安全问题

A:同步代码块

synchronized(对象) {

需要被同步的代码;

}

这里的锁对象可以是任意对象。

B:同步方法

把同步加在方法上。

这里的锁对象是this

C:静态同步方法

把同步加在方法上。

这里的锁对象是当前类的字节码文件对象(反射再讲字节码文件对象)

(11)回顾以前的线程安全的类

A:StringBuffer

B:Vector

C:Hashtable

D:如何把一个线程不安全的集合类变成一个线程安全的集合类

用Collections工具类的方法即可。

1:多线程(理解)

(1)JDK5以后的针对线程的锁定操作和释放操作

Lock锁

(2)死锁问题的描述和代码体现

(3)生产者和消费者多线程体现(线程间通信问题)

以学生作为资源来实现的

资源类：Student

设置数据类：SetThread(生产者)

获取数据类：GetThread(消费者)

测试类：StudentDemo

代码：

A:最基本的版本，只有一个数据。

B:改进版本，给出了不同的数据，并加入了同步机制

C:等待唤醒机制改进该程序，让数据能够实现依次的出现

wait()

notify()

notifyAll() (多生产多消费)

D:等待唤醒机制的代码优化。把数据及操作都写在了资源类中

(4)线程组

(5)线程池

(6)多线程实现的第三种方案

2:GUI(了解)

(1)用户图形界面

GUI:方便直观

CLI:需要记忆一下命令，麻烦

(2)两个包：

java.awt:和系统关联较强

javax.swing:纯Java编写

(3)GUI的继承体系

组件：组件就是对象

容器组件：是可以存储基本组件和容器组件的组件。

基本组件：是可以使用的组件，但是必须依赖容器。

(4)事件监听机制(理解)

A:事件源

B:事件

C:事件处理

D:事件监听

(5)适配器模式(理解)

A:接口

B:抽象适配器类

C:实现类

(6)案例：

A:创建窗体案例

B:窗体关闭案例

C:窗体添加按钮并对按钮添加事件案例。

界面中的组件布局。

D:把文本框里面的数据转移到文本域

E:更改背景色

F:设置文本框里面不能输入非数字字符

G:一级菜单

H:多级菜单