Java Notes

Java语言基础

- Java语言的基本组成单位是类,类体中又包含属性和方法
- 文件名必须和类名相同
- Java是严格区分大小写的
- 定义`long`型数据时,要在数据后加L或l. 例:

```
long number = 63826482L;
```

- 八进制数必须以0开头,十六进制数必须以0x或0X开头
- 定义`float`型浮点数,要在数据后加f或F 例:

```
float f1=123.23F;
```

- 's'表示字符,"s"表示字符串
- Unicode编码采用无符号编码0-65536
- 将转义字符赋值给字符变量时,与字符常量值一样需要使用单引号
- 布尔值不能与整数类型进行转换
- 其值不能改变的量成为常量,其值可以改变的量称为变量
- Java语言规定标识符由任意顺序的字母,下划线,美元符号和数字组成,并且第一个字符不能是数字
- 在Java中声明一个常量,除了要指定数据类型,还需要通过`final`关键字进行限定 标准语法: `final` 数据类型 常量名称 [=值]; 常量名通常大写 例:

```
final double PI = 3.14D;//(只可赋值一次,不可更改)
```

- 在类体中声明的变量称为成员变量,在整个类中都有效,其可分为静态变量(又称类变量,前有`static`修饰,可用类名.静态变量直接调用)和实例变量
- 在类方法体中声明的变量称为局部变量,只在当前代码块中有效
- 当局部变量与成员变量同名时,成员变量将隐藏
- 比较运算符的运算结果是`boolean`型
- 通常将这中在逻辑表达式中从左端的表达式可推断出整个表达式的值的情况称为"短路",而那些始终需要执行逻辑运算符两边的表达式才能推断出整个表达式的值的情况称为"非短路"."&&"属于"短路"运算符,而"&"属于"非短路"运算符
- "按位与"(&)当两个整型数据对应位都为1,结果才为1,否则为0;"按位或"(|)当两个整型数据对应位都为0,结果才为0,否则为1;"按位取反"(~)将操作数二进制中的1修改为0,0修改为1;"按位异或"(^)当两个操作数的二进制相同时,结果为0,否则为1.*若两个操作数的精度不同,结果的精度与精度高的操作数相同.
- `<<`左移,`>>`右移,`>>>`无符号右移;向左移n位,就是将这个数乘以2的n次方,向右移n位,就是将这个数处以2的n次方.
- 三元运算符 格式 : 条件式 ? 值1 : 值2; 例:

```
boolean b = 20 < 45  ? true : false;
```

- 优先级由高到低:
  1.增量和减量运算符;

2.算术运算符;
3.比较运算符;
4.逻辑运算符;
5.赋值运算符.
*运算符相同时,从左向右.

- 类型精度由低到高的顺序为: `byte` < `short` < `int` < `long` < `float` < `double` ; 由低级类型向高级类型的转换,系统将自动完成,无需用户操着;由高级类型向低级类型的转换,必须使用显式类型(强制类型转换)转换运算 语法如下: (类型名)要转换的值,例:

```
int a = (int)45.23D;
```

- 执行显式类型转换时,可能导致精度的损失;除`boolean`类型外,其他基本类型都能以显式类型转换的方法实现转换.
- 多行注释中可嵌套单行注释,但多行注释中不可嵌套多行注释
- `if`条件语句 语法: `if`(布尔表达式){语句序列} *当语句序列仅有一条语句时,可省略大括号(建议不省略) `if...else`与`if...else if...`类似
- `switch`语句句首先计算表达式的值,如果表达式的计算结果和某个`case`后面的常量值相同,则执行该`case`语句后的若干个语句直到遇到`break`语句为止.此时,如果该`case`语句中没有`break`语句,将继续执行后面`case`中的若干个语句，直到遇到`break`为止.如果没有一个常量的值与表达式的值相同,则执行的`default`后面的语句.`default`语句为可选的,如果它不存在,且`switch`语句中表达式的值不与任何`case`的常量值相同,`switch`语句则不做任何处理.
  - 同一个`switch`语句中`case`的常量值必须互不相同
  - 在`switch`语句中,`case`语句后常量表达式的值可以为整数,但绝不可以是非整数的实数

```
switch(表达式){
case 常量值1 :
    语句块1;
    [break;]
......
case 常量值n :
    语句块n;
    [break;]
default:
    语句块n+1;
    [break;]
}
```

- 循环语句
  - `while`循环
    - 利用某一条件来控制是否继续反复执行这个语句
    - 语法如下

```
while(条件表达式){
    语句序列
}
```

  - `do...while`循环语句

        - while循环语句先判断条件是否成立,在进入循环体;而do...while循环先执行循环体,再判断条件
        - 语法如下

```
do{
    语句序列
}while(条件表达式);
```

    - `for`循环语句(传统)
        - 表达式1：初始化表达式
        - 表达式2：循环条件表达式
        - 表达式3：每次循环结束后执行的表达式
        - 语法如下

```
for(表达式1;表达式2;表达式3){
    语句序列
}
```

    - `for`循环语句(`foreach`语句)
        - 不必对元素x进行初始化
        - 语法如下

```
for(元素类型x:历便对象obj){
    引用x的Java语句
}

int arr[] = {5,13,96};
for(int x:arr){
    System.out.println(x);
}
```

- 循环控制
    - `break`语句
        - 跳出当前循环体(内层循环)
        - 标签功能跳出指定循环
        - 语法如下
            - 标签名:任意标识符
            - 循环体:任意循环语句
            - `break` 标签名:跳出指定的循环体

```
标签名:循环体{
    break 标签名;
}
```

    - `continue`语句
        - 跳出本次循环，执行本循环体的下一次循环
        - `continue`与`break`一样也支持标签功能,用法与`break`一样

数组

- 概念：数组是相同类型的用一个标识符封装到一起的基本类型数据序列或对象序列
- 在Java中将数组看作一个对象
- 一维数组
  - 数组作为对象允许使用 `new` 关键字进行内存分配
  - 使用数组之前，必须首先定义数组变量所属的类型
  - 语法如下:

```
数组元素类型 数组名称[];
数组元素类型[] 数组名称;

int arr[];
int[] arr;
```

  - 声明数组仅给出数组名称和元素的数据类型，还应用 `new` 初始化,已分配内存空间
  - 数组通过下标来区分数组中不同的元素,从0开始
  - 语法如下:

```
数组名称 = new 数组元素类型[数组元素个数];

arr = new int[5];
```

  - 声明的同时为数组分配内存(普遍做法)
  - 语法如下:

```
数组元素类型 数组名 = new 数组元素类型[数组元素个数];

int month[] = new int[12];
```

  - 数组的初始化(两种方法相同)
  - 示例如下:

```
int arr[] = new int[]{1,2,3,5,23};
int arr[] = {1,2,3,5,23};
```

- 二维数组
  - 数组作为对象允许使用 `new` 关键字进行内存分配
  - 使用数组之前，必须首先定义数组变量所属的类型
  - 语法如下:

```
数组元素类型 数组名称[][];
数组元素类型[][] 数组名称;

int arr[][];
int[][] arr;
```

  - 声明数组仅给出数组名称和元素的数据类型，还应用 `new` 初始化,已分配内存空间
  - 数组通过下标来区分数组中不同的元素,从0开始

- 语法如下:

```
数组名称 = new 数组元素类型[一级数组元素个数][二级数组元素个数];
```

- 两种为数组分配内存的方法
- 示例如下:

```
arr = new int[2][4];

arr = new int[2][];
arr[0] = new int[2];
arr[1] = new int[3];
```

- 声明的同时为数组分配内存(普遍做法)
- 示例如下:

```
int a = new int [2][4];
```

- 二维数组的初始化与一维数组类似
- 语法如下:

```
数组数据类型 数组名称[][] = {n个一维数组};

int arr[][] = {{12,0},{45,10}};
```

- 对于整型都数组,创建成功之后系统会给数组中都每个元素赋予初始值0

- 数组的基本操作(详见java.util.Arrays)
  - 遍历数组
    - 利用`for`循环实现(`foreach`语句更简单)
    - 通过数组的`length`属性可获得数组长度
  - 填充替换数组元素(`Arrays`类的静态方法`fill()`)
    - `a`:要进行元素替换的数组
    - `value`:要储存数组中所有元素的值
    - `fromIndex`:要使用指定值填充的第一个元素的索引(包括)
    - `toIndex`:要使用指定值填充的最后一个元素的索引(不包括)
    - 语法如下:

```
fill(int[] a, int value)
fill(int[] a,int fromIndex,int toIndex,int value)
```

  - 对数组进行排序(`Arrays.sort()`)
    - `object`指进行排序的数组的名称
    - 语法如下:

```
Arrays.fill(object)
```

    - Java中`String`类型数组的排序算法是根据字典编排顺序排序的
  - 复制数组

- `Arrays.copyOf()`方法
  - `arr`:要进行复制的数组对象
  - `newlength`: `int`型常量指复制后的新数组的长度,超出`arr`的部分用0(`int`型数组)或`null`(`char`型数组)
  - 语法如下

```
copyOf(arr,int newlength);
```

- `Arrays.copyOfRange()`方法
  - `arr`:要进行复制的数组对象
  - `fromIndex`(包括):指定开始复制数组的索引位置,必须在0至整个数组之间,`toIndex`(不包括)可在其之外
  - 语法如下:

```
copyOfRange(arr,int fromIndex,toIndex);
```

- 查询函数
  - `Arrays.binarySearch()`方法
    - `arr`要索引的数值
    - `fromIndex`:要搜索的第一个元素的索引(包括)
    - `toIndex`:要搜索的最后一个元素的索引(不包括)
    - `key`:要搜索的值
    - 若数组包含多个带有指定值的元素,则无法保证找到的是哪一个
    - 示例如下:

```
binarySearch(Object[] arr,int fromIndex,int toIndex,Object key);  //必须排序

binarySearch(Object[] arr,Object kry);//必须排序
```

- 数组排序方法 详见更多
  - 冒泡排序法
    - 由双层循环实现
    - 外层循环控制排序轮数,一般为要排序的数组长度减1次
    - 核心方法示例如下:

```
for(int i=1;i<arrays.length;i++){
    for(int j=0;j<arrays.length-i;j++){
        if(arrays[j]>arrays[j+1]){
            int tmp = arrays[j];
            arrays[j] = arrays[j+1];
            arrays[j+1] = tmp;
        }
    }
}
```

  - 直接选择排序法
    - 将指定排序位置元素与其他数组分别对比,如果满足条件就交换元素值

- 核心方法示例如下:

```
for(int i=1;i<arrays.length;i++){
    index=0;
    for(int j=1;j<=arrays.length-i;j++){
        if(arrays[j]>arrays[index]){
            index=j;
        }
    }
    int tmp = arrays[arrays.length-i];
    arrays[arrays.length-i] = arrays[index];
    arrays[index] = tmp;
}
```

- 反转排序法
  - 将数组最后一个元素与第一个元素替换,倒数第二个元素与第二个元素替换.....以此类推
  - 核心代码示例如下:

```
int tmp;
int length = arrays.length;
for(int i=0;i<(length/2);i++){
    tmp=arrays[i];
    arrays[i]=arrays[length-i-1];
    arrays[length-i-1]=tmp;
}
```

> 请多关照