2020红岩网校移动开发部Android方向第一节课💪

> **课前BB两句**😆
>
> - **本课内容概述**
>
> 1. 什么是移动开发?
> 2. Android方向第一学期的计划介绍
> 3. 提问的Q&A☆
> 4. 作业提交的Q&A💢
> 5. 什么是Android，什么是Java?
> 6. 基本数据类型
> 7. 变量（修饰符）
> 8. Java运算符，运算优先级
> 9. 输入输出函数JavaIO
> 10. 条件语句
> 11. 循环语句
> 12. 数组
> 13. 冒泡排序
> 14. 总结

- **什么是移动开发**🧐

！！我们不是移动办卡的！！

！！没有啥电信开发！！

！！没有啥联通开发！！

🍤**移动是指移动互联网，移动开发是指移动应用的开发**

以下摘抄百度百科

移动开发也称为手机开发，或叫做移动互联网开发，移动应用开发等。移动应用的形成对移动设备的功能有了长足的拓展。设备可以不单单只靠自带的简陋功能，而是可以像计算机一样通过安装应用程序、游戏程序等进行扩展，使移动设备成为更能帮助人们解决事物的个人智能终端。

说白了就是，你可以通过一个小小的手机，做你任何想做的事！

- **Android方向的本学期学习计划（详情请见群文档的入群pdf）**

移动Android开发，前段时间大概1个多月是进行Java的学习（大佬请无视）。万丈高楼起于垒土，然后，后面就可以跳出黑框框，进行司马行空的Android开发，设计属于你自己的app。最后，使用Kotlin（Java的一种进阶进行高速开发Android）

- **提问的方法和常见的问题**

俗话说得好，要想高效的学习，没有问题是不可能的，所以提问很重要，但是会提问更重要

曾经我也在等一个小红心，到头来是这样。好了言归正传，如何提问

1. 程序编译或者软件安装问题(Java方面也还好，爆红不会太多）（请一定要截图，不然拍照的时候请寄上一盒眼药水）
2. 提问的方式
    - 组织你的问题，提问学会停顿。别这样好吗（下面配图）
    - 尊重双方（别这样好吗，下面配图）

- **作业提交的方法**

俗话说的好，光学习，不练习，怎么学的好呢，这个作业呢，你们可以做也可以不做，不是强制性的，但是，你要想摸鱼谁也拦不住你哈。网校就是这么一个开发自由的平台（至少移动是哈）

1. 写作业的时候先看看作业的要求
    - Lv0和Lv1建议必须完成
    - 所有作业必须用Java完成，当然你能直接Kotlin也行，提交到mredrock@163.com

2. 邮件的标题

```
第0次-level1-张涛-2019210xxx
```

Ps:记得将你写的Java文件压缩成zip包，可以下一个BandZip便于压缩。压缩后的**文件名也需要和标题名一样**

3. 上传文件（zip）的方法

在发送邮件中有添加附件。添加即可

如果这样都不会！😊

正式开讲

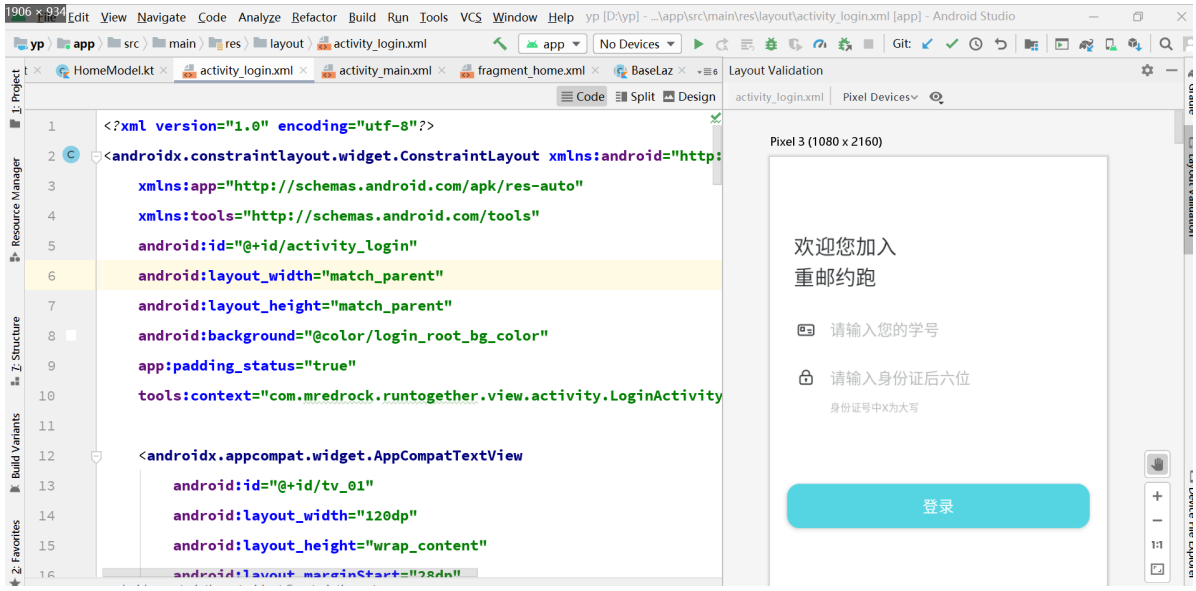
- **什么是Android，什么是Java**

Android，俗称“安卓”，本义“机器人”，Android是一款基于Linux系统的自由及开放源代码的**手机操作系统**，主要应用于移动设备，如智能手机和平板电脑，他的父亲，就是谷歌公司（Google）

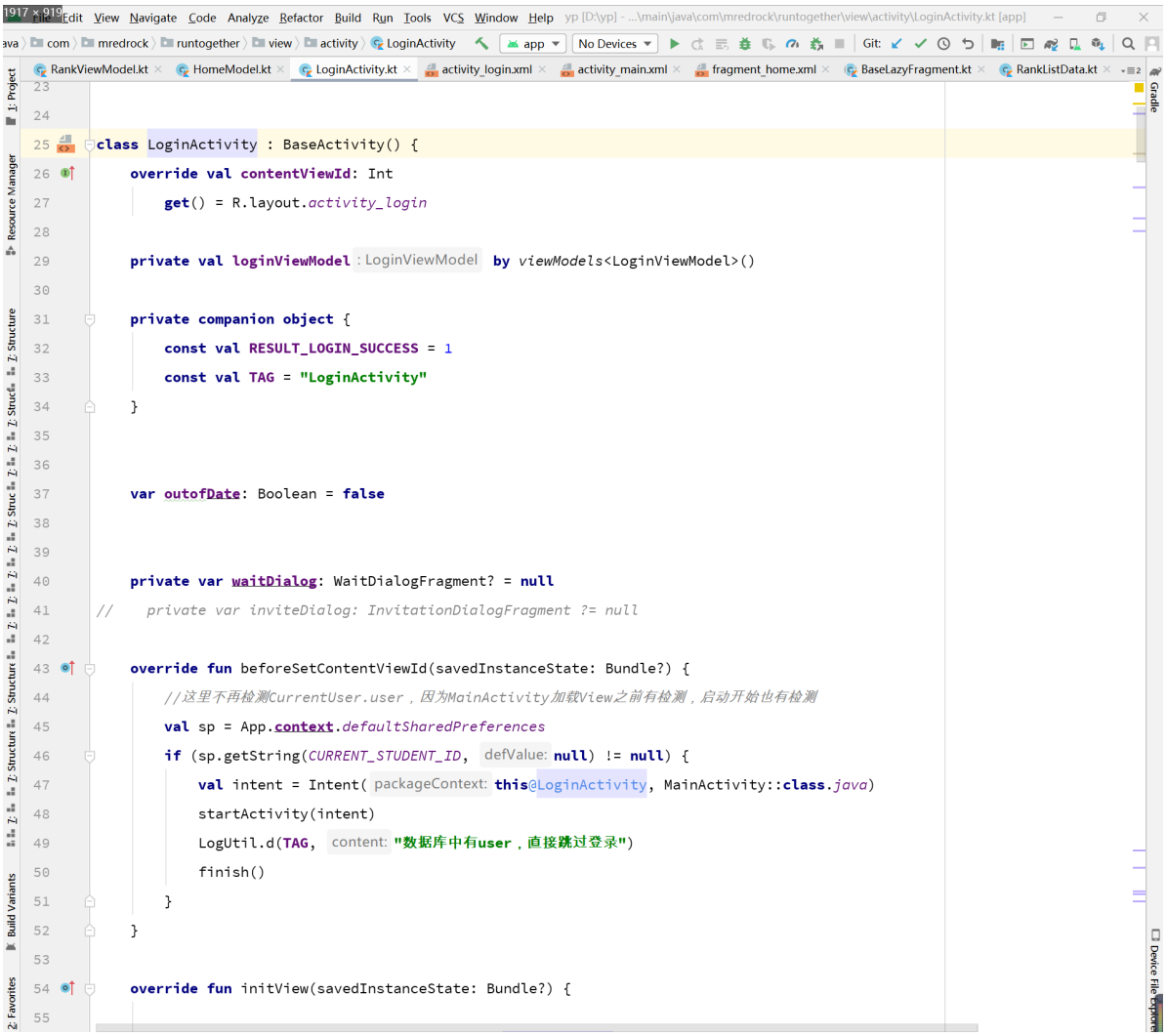
Java是一般认为是一种编程语言，暂时没有真正中译意思，但网传命名源于爪哇岛，爪哇小岛盛产咖啡，这种咖啡也叫做JAVA

- **移动开发的工作**

1. 静态布局



2.动态逻辑代码



好了，多的不说，万丈高楼垒土起，大家先从最基础的Java了解吧！

- **Java的几大特性（先了解就好，后面慢慢的会潜移默化的去理解它）**
  - **Java是简单的**

1. Java 语言不使用指针，而是引用。并提供了自动的废料收集，使得程序员不必为内存管理而担忧。

  - **Java是面向对象的**

1. Java 语言提供类、接口和继承等面向对象的特性，为了简单起见，只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口之间的实现机制（关键字为 implements）。

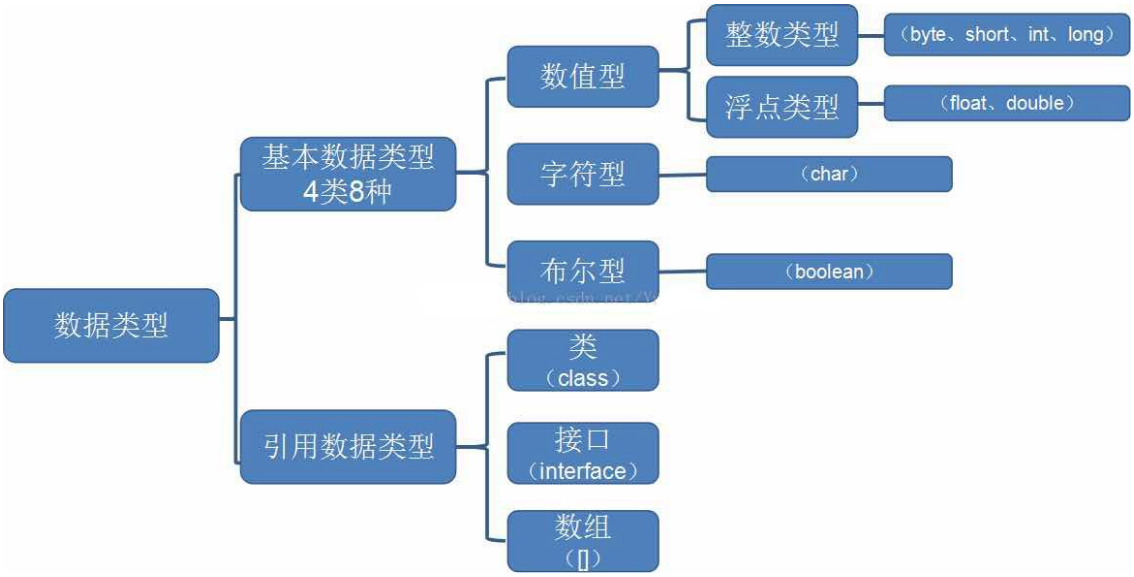
  - **Java是多线程的**

1. 线程的活动由一组方法来控制。Java 语言支持多个线程的同时执行，并提供多线程之间的同步机制（关键字为 synchronized）。

.....

- **Java的一些基本语法👊**
  - 对象：对象是一个类的实例，有状态（属性）和行为（函数方法）例如，一条狗是一个对象，它的状态有：颜色、名字、品种；行为有：摇尾巴、叫、吃等。
  - 类：类是一个模板，它描述一类对象的行为和状态
  - 方法：方法就是行为，一个类可以有很多方法，逻辑运算、数据修改以及所有动作都是在方法中完成的。
  - 实例变量：每个对象都有独特的实例，对象的状态由这些实例变量的值确定。

- **Java基本数据类型😮**



数据类型之前，需要大家理解几个概念，计算机存储的是一堆01010的机械码。

0代表1bit。1Byte（B）（一个字节）=8bit（8位）。1KB=1024B.....

😮

计算机的原码：用二进制定点表示法，最高位是符号位，0代表正数，1代表负数。其余的余位表示数值的大小。

例如：10的原码就是**0**0001010.-10的原码就是**1**0001010

计算机的反码：正数的反码和其原码相同，负数的反码是对其原码逐位取反，**但符号位除外**（即把除去符号位的剩余位1改成0，0改成1）

例如 10的反码是**0**0001010，-10的反码就是**1**1110101

计算机的补码：正数的补码等于其原码，负数的补码等于负数的反码+1.

例如：10的补码是00001010，-10的补码就是11110110

为什么要引用补码，反码呢？：先搞清楚一点反码、补码、原码是针对二进制数而言，计算机若用原码相加减，正数加正数不会出错，然而正数和负数原码相加就会出错，说白了,补码、反码就是为了简化减法而来的,不管是正整数还是正小数,原码,反码,补码都全部相同。计算机中所有数都是以补码形式存储的。

例如：

1.10+(1)时：1010 + 0001 = 1011，即11。

2.10+(-1)时：00001010 + 10000001 = 10001011 ，即 -11。（原码参与加法运算）

3.10+(-1)时：00001010 + 11111111 = 00001001，即 9。（补码参与加法运算）

各位大佬，不知道我讲清楚了没有？

简单介绍下两种数据类型的存储：

- 对于基本类型来说，**栈**区域包含的是基本类型的内容，也就是值；

  例如 int a =2;

- 对于引用类型来说，**栈**区域包含的是指向**真正内容的指针**（真正内容在堆中的地址），真正的内容被分配在了**堆**中。

  例如 Person dingshen=new Person();

1. **byte**
   - byte数据类型是8位，也就是一个字节。
   - 最小值是-2^7=-128，最小应该是11111111.这不是等于-127吗？这离谱。关键是0.是正数还是负数呢？+0表示为：00000000，-0表示为：1000000。**因为最高位是符号位不算在里面，所以就会有两个0，所以从一开始发明二进制的时候，就把-0规定为-128，这里就是把最高位-符号位，既是符号位，又是最高位，因此这里的1既代表是负数，又代表2^7次方）**
   - 最大值是127（2^7-1）也就是01111111=127

     小结：**2^8=256，即能有256个数字，包括最高位是0和1的情况，相当于-127到-0，0到127，这样正负数各128个，由于-0算成最小负数，负数没有少，所以负数是128个**

     由于正数的0没有了，正数少一个，所以正数是127个
2. **short**
   - short 数据类型是 16 位、有符号的以二进制补码表示的整数（2个字节）
   - 最小值是 **-32768（-2^15）**；

- 最大值是 **32767（2^15 - 1）**；

3. **int**
   - int 数据类型是32位、有符号的以二进制补码表示的整数；（4个字节）
   - 最小值是 **-2,147,483,648（-2^31）**；
   - 最大值是 **2,147,483,647（2^31 - 1）**；
4. **long:**
   - long 数据类型是 64 位、有符号的以二进制补码表示的整数；
   - 最小值是 **-9,223,372,036,854,775,808（-2^63）**；
   - 最大值是 **9,223,372,036,854,775,807（2^63 -1）**；
5. **float**
   - float 数据类型是单精度、32位、符合IEEE 754标准的浮点数；
   - float 在储存大型浮点数组的时候可节省内存空间；
   - 默认值是 **0.0f**；
6. **double:**
   - double 数据类型是双精度、64 位、符合IEEE 754标准的浮点数；
   - 浮点数的默认类型为double类型；
7. **boolean**
   - boolean数据类型表示一位的信息；
   - 只有两个取值：true 和 false；
   - 这种类型只作为一种标志来记录 true/false 情况；
   - 默认值是 **false**；
8. **char**
   - char类型是一个单一的 16 位 Unicode 字符；

如何使用：

```java
public class Main {
    static boolean bool;
    static byte by;
    static char ch;
    static double d;
    static float f;
    static int i;
    static long l;
    static short sh;
    static String str;

    public static void main(String[] args) {
        System.out.println("Bool :" + bool);
        System.out.println("Byte :" + by);
        System.out.println("Character:" + ch);
        System.out.println("Double :" + d);
        System.out.println("Float :" + f);
        System.out.println("Integer :" + i);
        System.out.println("Long :" + l);
        System.out.println("Short :" + sh);
        System.out.println("String :" + str);
    }
```

- **Java变量😃**
    - 类变量：独立于方法之外的变量，用 static 修饰。
    - 实例变量：独立于方法之外的变量，不过没有 static 修饰。
    - 局部变量：类的方法中的变量。

例如：

```java
package com.company.classidentifier;

public class demo {
    static  int a=0;//类变量
    String str="hello world";  // 实例变量
    public  void Methond(){
        int i =0;// 局部变量
    }
    public static void main(String[] args) {
    }

}
```

简单介绍一下他们的区别：

局部变量：

局部变量在方法、构造方法、或者语句块被执行的时候创建，当它们执行完成后，变量将会被销毁。**局部变量没有默认值**，所以局部变量被声明后，**必须经过初始化**，才可以使用

```java
1  package com.company.classidentifier;
2
3  public class demo {
4      static  int a=0;//类变量
5      String str="hello world";  // 实例变量
6      public  void Methond(){
7          int i ;// 局部变量
8          System.out.println(i);
9      }
10 }
11
```

Variable 'i' might not have been initialized

Initialize variable 'i' Alt+Shift+Enter More actions... Alt+Enter

实例变量

- 实例变量声明在一个类中，但在方法、构造方法和语句块之外；
- 当一个对象被实例化之后，每个实例变量的值就跟着确定；
- 实例变量在对象创建的时候创建，在对象被销毁的时候销毁；

类变量

- 类变量也称为静态变量，在类中以 static 关键字声明，但必须在方法之外。
- 静态变量储存在静态存储区。经常被声明为常量，很少单独使用static声明变量。
- 静态变量在第一次被访问时创建，在程序结束时销毁

这些大家先能大致分别他们是啥就好了，具体的区别涉及到另外的一些东西，建议后面学完后再看看。

- **Java修饰符😉**
  - 访问修饰符
  - 非访问修饰符

访问修饰符😳

Java中，可以使用Java中，可以使用访问控制符来保护对类、变量、方法和构造方法的访问。来保护对类、变量、方法和构造方法的访问。

- **default** (即默认，什么也不写）: 在同一包内可见，不使用任何修饰符。使用对象：类、接口、变量、方法。
- **private** : 在同一类内可见。使用对象：变量、方法。 **注意：不能修饰类（外部类）**
- **public** : 对所有类可见。使用对象：类、接口、变量、方法
- **protected** : 对同一包内的类和所有子类可见。使用对象：变量、方法。 **注意：不能修饰类（外部类）。**
- 下表为Java访问控制符的含义和使用情况

	类内部	本包	子类	外部包
public	√	√	√	√
protected	√	√	√	×
default	√	√	×	×
private	√	×	×	×

非访问修饰符

static 修饰符

- **静态变量:**

  static 关键字用来声明独立于对象的静态变量，无论一个类实例化多少对象，它的静态变量只有一份拷贝。 静态变量也被称为类变量。局部变量不能被声明为 static 变量

- **静态方法:**

  static 关键字用来声明独立于对象的静态方法。静态方法不能使用类的非静态变量。静态方法从参数列表得到数据，然后计算这些数据。

  对类变量和方法的访问可以直接使用 **classname.variablename** 和 **classname.methodname** 的方式访问

这些概念大家可以后面慢慢去理解，我只是在这里给大家整理下学习的思路。

具体的使用：😳

```
public class InstanceCounter {
    private static int numInstances = 0;//静态变量

    protected static int getCount() {
        return numInstances;
    }

    private static void addInstance() {
        numInstances++;
    }

    //构造方法
    InstanceCounter() {
        InstanceCounter.addInstance();
    }

    public static void main(String[] arguments) {
        System.out.println("Starting with " +
                InstanceCounter.getCount() + " instances");
        for (int i = 0; i < 500; ++i){
            new InstanceCounter();
        }
        System.out.println("Created " +
                InstanceCounter.getCount() + " instances");
    }
}
```

final修饰符😳

- **final变量**:

  final表示"最后的，最终的"含义，变量一旦赋值后，不能被重新赋值。被 final 修饰的实例变量必须显式指定初始值，final 修饰符通常和 static 修饰符一起使用来创建类常量

- **final方法**

父类中的final方法可以被子类继承，但不能被重写。所以，只需要记住，声明final方法的主要目的是防止该方法的内容被修改。

- **final类**😳

final类不能被继承，没有类能够继承final类的任何特性

abstract修饰符😳

这个是有关抽象类的修饰符，这里不赘述。

.....

- **Java运算符**😃
  - **算术运算符**（+,-,*,/,%,++,--）

```java
package com.company.classidentifier;

public class JavaCaculate {

    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int c = 25;
        int d = 25;
        System.out.println("a + b = " + (a + b));
        System.out.println("a - b = " + (a - b));
        System.out.println("a * b = " + (a * b));
        System.out.println("b / a = " + (b / a));
        System.out.println("b % a = " + (b % a));
        System.out.println("c % a = " + (c % a));
        System.out.println("a++   = " + (a++));
        System.out.println("a--   = " + (a--));
        // 查看  d++ 与 ++d 的不同
        System.out.println("d++   = " + (d++));
        System.out.println("++d   = " + (++d));
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_192\bin\java.exe" ...
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++   = 10
a--   = 11
d++   = 25
++d   = 27
```

d++和++d的区别是，d++是先使用d，再自加，++d是先加后再使用。

- **关系运算符**（只会返回ture或者false），包括（==，！=，>, <, >=,<=）
- **位运算符**（可能数电，或者后面Android涉及到Color的时候会用到）

操作符	描述	例子
&	如果相对应位都是1，则结果为1，否则为0	（A&B），得到12，即0000 1100
|	如果相对应位都是 0，则结果为 0，否则为 1	（A | B）得到61，即 0011 1101
^	如果相对应位值相同，则结果为0，否则为1	（A ^ B）得到49，即 0011 0001
~	按位取反运算符翻转操作数的每一位，即0变成1，1变成0。	（~A）得到-61，即1100 0011
<<	按位左移运算符。左操作数按位左移右操作数指定的位数。	A << 2得到240，即 1111 0000
>>	按位右移运算符。左操作数按位右移右操作数指定的位数。	A >> 2得到15即 1111
>>>	按位右移补零操作符。左操作数的值按右操作数指定的位数右移，移动得到的空位以零填充。	A>>>2得到15即0000 1111

- **逻辑运算符**（且&&，或||，非！）
- **赋值运算符**（=，+=，-=，*=，/=，（%）=....）
- **三元运算符，条件运算符**（？：），是if...else的缩写。

```
public class conditionCaculate {
    public static void main(String[] args) {
        int a , b;
        a = 10;
        // 如果 a 等于 1 成立，则设置 b 为 20，否则为 30
        b = (a == 1) ? 20 : 30;
        System.out.println( "Value of b is : " +  b );
        /*
            等价于
         */
        if (a==1){
            b=20;
        }else {
            b=30;
        }
        System.out.println("Value of b is : "+b);

        // 如果 a 等于 10 成立，则设置 b 为 20，否则为 30
        b = (a == 10) ? 20 : 30;
        System.out.println( "Value of b is : " + b );
    }
```

- **instanceof 运算符**

  该运算符用于操作对象实例，检查该对象是否是一个特定类型（类类型或接口类型）

```java
package com.company.classidentifier;

class instanceofdemo {
}

class Var extends instanceofdemo {
    public static void main(String[] args) {
        instanceofdemo a = new Var();
        boolean result = a instanceof Var;
        System.out.println(result);

    }
}
```

  - **Java运算符优先级**，这个一般的开发中不怎么会为难你，除非是考试，或者炫技的程序员（不过这种一般会被喷）

- **Java输入输出函数**😀

  - **sout**(缩写)

    System.out.println（）输出

  - 输入函数（Scanner类）

    使用方法：

```java
public class scanner {
    public static void main(String [] args) {
        Scanner sc = new Scanner(System.in);
        //定义一个输入的对象
        System.out.println("请输入你的姓名：");
        //定义一个变量去接受输入的东西
        String name = sc.nextLine();
        System.out.println("请输入你的年龄：");

        int age = sc.nextInt();
        System.out.println("请输入你的工资：");
        float salary = sc.nextFloat();
        System.out.println("你的信息如下：");
        System.out.println("姓名："+name+"\n"+"年龄："+age+"\n"+"工资："+salary);
    }
}
```

    只需要一个Scanner对象，即可完成多次的输入接受

- **条件语句**😠

  - **if**
  - **if....else**
  - **if...else if ...else**

```
if(布尔表达式 1){
   //如果布尔表达式 1的值为true执行代码
}else if(布尔表达式 2){
   //如果布尔表达式 2的值为true执行代码
}else if(布尔表达式 3){
   //如果布尔表达式 3的值为true执行代码
}else {
   //如果以上布尔表达式都不为true执行代码
}
```

```
int x = 30;

if( x == 10 ){
    System.out.print("Value of X is 10");
}else if( x == 20 ){
    System.out.print("Value of X is 20");
}else if( x == 30 ){
    System.out.print("Value of X is 30");
}else{
    System.out.print("这是 else 语句");
}
```

- **嵌套的if...else语句**

```
int x = 30;
int y = 10;

if( x == 30 ){
    if( y == 10 ){
        System.out.print("X = 30 and Y = 10");
    }
}
```

- **Swith case语句**

```
switch(expression){
    case value :
       //语句
       break; //可选
    case value :
       //语句
       break; //可选
    //你可以有任意数量的case语句
    default : //可选
       //语句
}
```

当遇到 break 语句时，switch 语句终止。程序跳转到 switch 语句后面的语句执行。case 语句不必须要包含 break 语句。如果没有 break 语句出现，程序会继续执行下一条 case 语句，直到出现 break 语句。default 分支不需要 break 语句。

- **switch case 执行时，一定会先进行匹配，匹配成功返回当前 case 的值，再根据是否有 break，判断是否继续输出，或是跳出判断**

```
char grade = 'C';

switch(grade)
{
    case 'A' :
        System.out.println("优秀");
        break;
    case 'B' :
    case 'C' :
        System.out.println("良好");
        break;
    case 'D' :
        System.out.println("及格");
        break;
    case 'F' :
        System.out.println("你需要再努力努力");
        break;
    default :
        System.out.println("未知等级");
}
System.out.println("你的等级是 " + grade);
```

如果 case 语句块中没有 break 语句时，JVM 并不会顺序输出每一个 case 对应的返回值，而是继续匹配，匹配不成功则返回默认 case

```java
int i = 5;
switch(i){
    case 0:
        System.out.println("0");
    case 1:
        System.out.println("1");
    case 2:
        System.out.println("2");
    default:
        System.out.println("default");
```

如果 case 语句块中没有 break 语句时，匹配成功后，从当前 case 开始，后续所有 case 的值都会输出。

```java
int i = 1;
switch(i){
    case 0:
        System.out.println("0");
    case 1:
        System.out.println("1");
    case 2:
        System.out.println("2");
    default:
        System.out.println("default");
}
```

总的来说，Switch case一定会**先进行匹配**，匹配成功返回当前 case 的值，再根据是否有 break，判断是否继续输出，或是跳出判断

- **循环语句**😠

  - **while循环**

```
while( 布尔表达式 ) {
  //循环内容
}
```

只有布尔表达式为true时，才会执行

```
int x = 10;
while( x < 20 ) {
    System.out.print("value of x : " + x );
    x++;
    System.out.print("\n");
}
```

- **do..while循环**

  do...while 循环和 while 循环相似，不同的是，do...while 循环至少会执行一次

```
do {
       //代码语句
}while(布尔表达式);
```

```
int x=10;
do {
    System.out.print("value of x : " + x );
    x++;
    System.out.print("\n");
}while(x>20);
}
```

- **for循环**

  快捷键fori

```
for(初始化; 布尔表达式; 更新) {
    //代码语句
}
```

```
for(int x = 10; x < 20; x = x+1) {
    System.out.print("value of x : " + x );
    System.out.print("\n");
}
```

  - **嵌套的for循环**

```
        Scanner input=new Scanner(System.in);
        System.out.println("请输入一个数字:");
        int in=input.nextInt();
        for (int i=0;i<in;i++){
            for (int j=0;j<=in;j++){
                System.out.print(" *");
            }
            System.out.print("\n");
        }
    }
```

- **Java中的for循环**

  Java5 引入了一种主要用于数组的增强型 for 循环。

  快捷键foreach

```
for(声明语句 : 表达式)
{
   //代码句子
}
```

```
        int []numbers={10,23,34,56};
        for (int x:numbers) {
            System.out.println(x);
        }
```

- **break语句**

  break 主要用在循环语句或者 switch 语句中，用来跳出整个语句块。

```
    public static void main(String[] args) {
        int []numbers={10,20,30,40};
        for (int x:numbers
             ) {
            if (x==30){
                break;
            }
            System.out.println(x);
        }
    }
```

- **continue关键字**

  continue 适用于任何循环控制结构中。作用是让程序立刻跳转到下一次循环的迭代。

```java
public class breakandcontinue {
    public static void main(String[] args) {
        int []numbers={10,20,30,40};
        for (int x:numbers
             ) {
            if (x==30){
//                break;
                continue;
            }
            System.out.println(x);
        }
    }
}
```

- **数组(存储固定大小的同类型元素)**

  - **声明数组变量**

```
double[] myList;          // 首选的方法

或
 double myList[];          //  效果相同，但不是首选方法，这种属于C/C++的方法
```

- **创建数组**

```
double []myList=new double[arraysize];
或
double []myList={value0,value1...}//直接赋值类型
```

-

```java
public static void main(String[] args) {
    // 数组大小
    int size = 10;
    // 定义数组
    double[] myList = new double[size];

    double[] myList2={5.6,4.5,3.3,13.2,4.0,34.33,34.0,45.45,99.993,11123};
    myList[0] = 5.6;
    myList[1] = 4.5;
    myList[2] = 3.3;
    myList[3] = 13.2;
    myList[4] = 4.0;
    myList[5] = 34.33;
    myList[6] = 34.0;
    myList[7] = 45.45;
    myList[8] = 99.993;
    myList[9] = 11123;
    // 计算所有元素的总和
    double total = 0;
    for (int i = 0; i <size; i++) {
        total += myList2[i];
    }
    System.out.println("总和为:  " + total);
```

- **简单的数组操作**
  - **遍历所有数组元素**

    for-each循环实现

```java
double[] myList = {1.9, 2.9, 3.4, 3.5};

// 打印所有数组元素
  for (int i = 0; i < myList.length; i++) {
      System.out.println(myList[i] + " ");
  }
for (double x:myList
     ) {
    System.out.println(x);
}
```

- **多维数组**

  例如二维数组：其二维数组的每一个元素都是一个一维数组

  {{3,3},{2,2},{1},{2}}

  使用方法：

```
type[][] typeName = new type[typeLength1][typeLength2];
```

  ype 可以为基本数据类型和复合数据类型，arraylength1 和 arraylength2 必须为正整数，arraylength1 为行数，arraylength2 为列数。

```
/*
    二维数组
 */
String [][]s=new String[2][2];
s[0][0] = new String( original: "Good");
s[0][1] = new String( original: "Luck");
s[1][0] = new String( original: "to");
s[1][1] = new String( original: "you");
for (int i = 0; i <2 ; i++) {
    for (int j = 0; j <2 ; j++) {
        System.out.print(" "+s[i][j]);
    }
}
```

- **冒泡排序😀grinning:**

一个排序的算法，对数组中的数字进行有序的排列。可能才开始理解起来有点困难。开启飞行模式

冒泡排序的不过是通过**无数次比较。交换位置**，所以我们来分解一下步骤

1. 比较（这个简单一个if语句判断就可以实现）
2. 交换位置（交换数组元素的位置）

```
int temp = num[j];
num[j]=num[j+1];
num[j+1]=temp;
```

给定一个无序数组:

A	B	C	D	E	F	G	H
冒泡排序原理							
初始状态:	15	25	10	30	40	20	50

在冒泡排序中,我们首先比较前两个数的大小,然后将大的数放在较小数的右侧。

A	B	C	D	E	F	G	H
冒泡排序原理							
初始状态:	15	25	10	30	40	20	50
第一次排序	15	10	25	30	20	40	50

A	B	C	D	E	F	G	H
冒泡排序原理							
初始状态:	15	25	10	30	40	20	50
第一次排序	15	10	25	30	20	40	50
第二次排序	10	15	25	20	30	40	50

我们可以发现,在第二次排序时,我们再比较F和G列的数字大小后,就可以不需要在去比较G和H列了

A	B	C	D	E	F	G	H
冒泡排序原理							
初始状态:	15	25	10	30	40	20	50
第一次排序	15	10	25	30	20	40	50
第二次排序	10	15	25	20	30	40	50
第三次排序	10	15	20	25	30	40	50

第四次排序

A	B	C	D	E	F	G	H
冒泡排序原理							
初始状态:	15	25	10	30	40	20	50
第一次排序	15	10	25	30	20	40	50
第二次排序	10	15	25	20	30	40	50
第三次排序	10	15	20	25	30	40	50
第四次排序	10	15	20	25	30	40	50

第五次排序

A	B	C	D	E	F	G	H
冒泡排序原理							
初始状态:	15	25	10	30	40	20	50
第一次排序	15	10	25	30	20	40	50
第二次排序	10	15	25	20	30	40	50
第三次排序	10	15	20	25	30	40	50
第四次排序	10	15	20	25	30	40	50
第五次排序	10	15	20	25	30	40	50

第六次排序

A	B	C	D	E	F	G	H
冒泡排序原理							
初始状态:	15	25	10	30	40	20	50
第一次排序	15	10	25	30	20	40	50
第二次排序	10	15	25	20	30	40	50
第三次排序	10	15	20	25	30	40	50
第四次排序	10	15	20	25	30	40	50
第五次排序	10	15	20	25	30	40	50
第六次排序	10	15	20	25	30	40	50

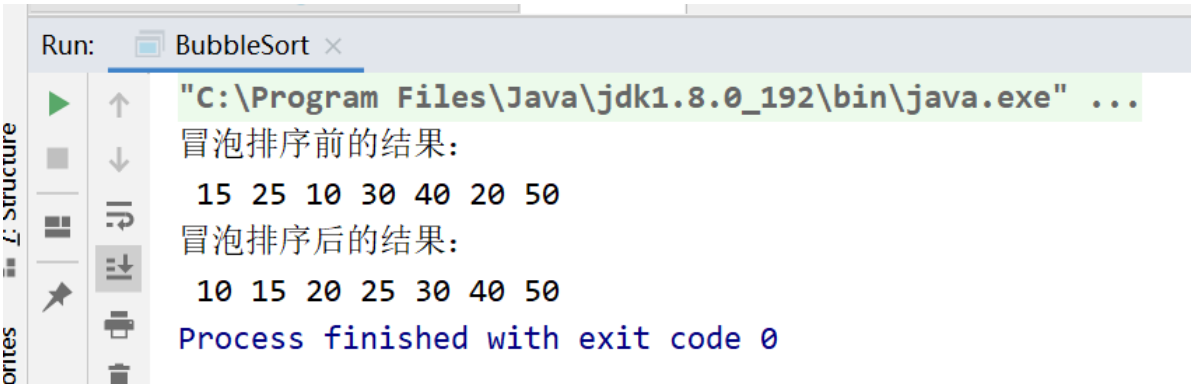
- 一共**7**个数
- 第一轮排序，进行了**6**次比较，选出了最大的一个数放在最下面
- 第二轮排序，进行了**5**次比较，选出了第二大的数
- 第三轮排序，进行了**4**次比较，选出了第三大的数
- 第四轮排序，进行了**3**次比较，选出了第四大的数
- 第五轮排序，进行了**2**次比较，选出了第五大的数
- 第六轮排序，进行了**1**次比较，选出了第六大的数
- 排序结束

总结：

1. 含有7个元素的数组最多通过6次排序便可以完成冒泡排序,这不是巧合,每次排序我们都将最大的数放在右侧,7个数我们只需要找到6个相对的最大的数便可完成排序
2. (针对一个含有n个元素的数组进行冒泡)
    - 含有n个元素的数组,至多进行(n-1)次排序便可以完成冒泡排序.
    - 针对每次排序:第(n-1)次,即最后1次排序,只需要比较1次便可完成本次排序

代码实现

```
            冒泡排序具体实现
         */
        /*
            打印排序前的样子
         */
        System.out.println("冒泡排序前的结果：");
        for (int x:
                arr1) {
            System.out.print(" "+x);
        }
        System.out.println();
        //外部循环冒泡次数,7个数，进行6轮排序
        for (int a = 0; a < arr1.length - 1; a++) {
            //内部循环每次冒泡的比较次数
            for (int b = 0; b < arr1.length - 1 - a; b++) {
                //交换相邻位置的元素,小的排在前面
                if (arr1[b] > arr1[b + 1]) {
                    int temp = arr1[b];
                    arr1[b] = arr1[b + 1];
                    arr1[b + 1] = temp;
                }
            }
        }
        /*
            打印排序后的输出结果
         */
        System.out.println("冒泡排序后的结果：");
        for (int x:
             arr1) {
            System.out.print(" "+x);
        }
    }
}
```



- **写在最后😁**

> 很高兴大家加入红岩网校，网校的学习速度确实很快，需要大家下来更多的花时间去理解和消化，不过没关系，你要相信你的代码不会白敲。编程中我们会遇到很多挫折。表放弃，沙漠尽头必是绿洲。希望大家在今后的学习生活中能做到**心里有火，眼里有光**！加油！

💪💪💪

: