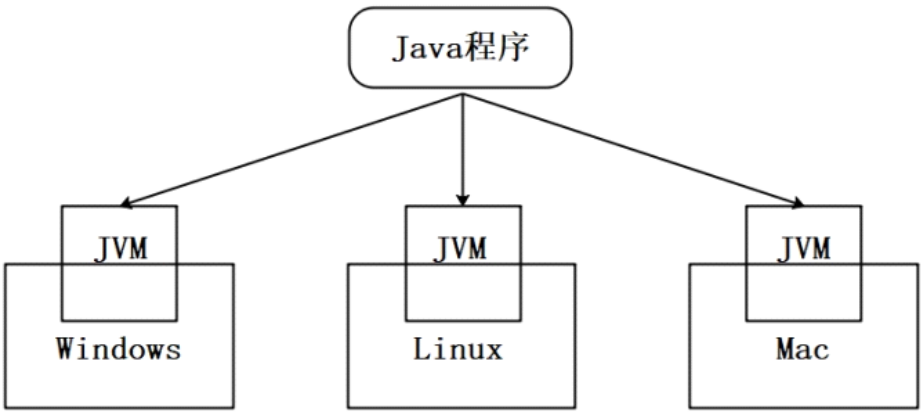
Java简介

一、Java简介

1. Java是1995年由SUN(Stanford University Network，斯坦福大学校园网)公司正式推出的一门编程语言
2. 到目前为止，市面上几乎大部分的WEB应用的首选语言都是Java
3. 大部分的大数据框架的底层实现语言也是Java
4. 因为Java是由詹姆斯•高斯林（James•Gosling）带领小组研发的，所以将James•Gosling成为Java之父
5. Java一共有10个特点：简单性、**面向对象**、分布性、编译和解释性、健壮性、安全性、**可移植性**（与平台无关）、高性能、多线索性、动态性

二、Java的跨平台

1. Java与平台无关的特性也称之为是Java的**跨平台**。所谓的跨平台是指写好一段代码之后不用做任何改动就能在不同的操作系统中运行
2. Java之所以能够跨平台是因为有Java虚拟机（Java Virtual Machine，简称为JVM）
3. Java小组针对不同的操作系统开发了不同的Java虚拟机，而所有的Java程序并不是直接运行在操作系统中而是先交由Java虚拟机。Java虚拟机屏蔽了不同操作系统之间的差异性，将同一段Java程序翻译为当前操作系统所能理解的指令，从而执行Java程序。所以Java语言的跨平台是基于Java虚拟机的
4. 需要注意的是**Java语言是跨平台的**，但是**Java虚拟机不是跨平台的**。



三、Java的技术结构

1. J2SE - JAVASE - Java Standard Edition。Java的标准版/基础版。JAVASE是JAVA语言入门的基础，也是另外两种技术结构的基础
2. J2EE - JAVAEE - Java Enterprise Edition。Java的企业版/商务版。JAVAEE是用于Java的企业开发
3. J2ME - JAVAME - Java Micro Edition。Java的微型版/移动版。JAVAME是一些小型电子设备的嵌入提供了解决方案

入门程序

一、JDK、JRE、JVM的介绍

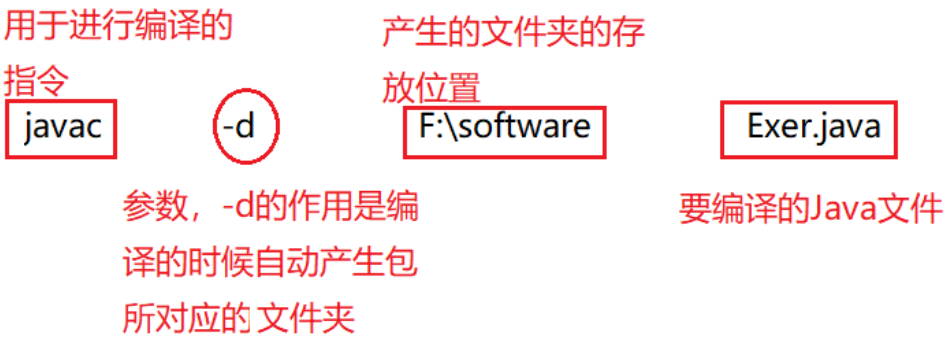
1. JVM（Java Virtual Machine）：Java虚拟机，是Java程序能够跨平台的前提
2. JRE（Java Runtime Environment）：Java运行时环境。主要作用是为Java程序的执行提供基本的环境支持。包含了JVM和Java的核心类库（核心类库中包含了Java程序最基本的支持）
3. JDK（Java Development Kit）：Java开发工具包。为Java程序的开发和使用提供了基本的工具，包含了JRE和基本的开发工具

二、JDK的下载和安装

1. 因为Java语言后来被甲骨文（Oracle）公司收购，所以可以去Oracle的官网下载。下载地址是：www.oracle.com
2. JDK版本：JDK1.0 - JDK1.1 - **JDK1.2** - JDK1.3 - JDK1.4 - **JDK1.5** - JDK1.6 - JDK1.7 - **JDK1.8** - JDK9 - JDK10 - JDK11(LTS，Long Time Support) - JDK12 - JDK13
3. 大部分的软件安装路径默认是C:\Program Files，但是此路径默认是在C盘下，而且路径中出现了空格，会导致后续其他软件的使用出现问题。所以建议将此路径改为其他路径，并且路径中不要出现空格

三、入门程序

1. 注意事项：
   a. Java程序的文件名的后缀必须是.java
   b. 在Java中，所有的程序都必须写在class结构中
   c. 需要将程序翻译为当前操作系统能够理解的指令，这个过程称之为编译(compile)
   d. 在Java中，一个程序如果想要执行，需要有一个入口，这个入口就是main方法（主方法，也称之为主函数）
   e. 类在编译完成之后会产生一个class文件，class文件的名字和类名是对应的
   f. 用public修饰的类称之为公共类。公共类的类名和Java文件的文件名必须一致
   g. 一个Java文件中只能包含1个公共类，但是一个Java文件中可以包含多个类
2. 环境变量：实际上是指给当前的操作系统来指定一些参数，使指定的内容能够在任意一个路径下使用
   JAVA_HOME=F:\software\Java\jdk1.8.0_151
   Path=%JAVA_HOME%\bin;
3. 课后作业：打印I Love Java
4. 包的作用：用于区分同名类。每一层包都对应一个目录（文件夹）



5. 如果类中定义了包，那么在运行的时候也需要带上包名，例如：java cn.tedu.demo.Exer

基本概念

一、关键字

1. 关键字指的是Java程序中**具有特殊含义**的单词，例如package，class，public等
2. 在Java中，到目前为止，一共有53个关键字，其中有2个关键字没有被使用，因此这2个关键字也称之为保留字：goto、const

二、标识符

1. 标识符指的是在程序中自定义的名称，例如类名，包名等
2. 命名规则：
   a. 可以由字母（不只是英文字母，包括了常见语言的基本字符。例如中文、俄语、韩语、日语等）、数字、_、$组成。其中需要注意的是，虽然可以使用$命名，但是实际过程中很少使用$
   b. 数字不能开头。例如Demo1、Demo2是可以的，但是1Demo、2Demo这是错误的
   c. 不能使用关键字，例如class public是错误的
   d. 由于Java是一门大小写敏感（Java严格区分大小写）的语言，所以两个标识符只要有一点不一样就认为是两个不同的标识符。例如demo和Demo是两个不同的标识符
   e. 起名的时候尽量做到**见名知意**。例如Teacher、Cat、Car等，但是不建议使用中文
3. 驼峰命名法：
   a. 对于类名、接口名，如果由多个单词组成，那么每一个单词的首字母要大写，其余字母小写。例如HelloWorld，Demo
   b. 对于变量名、方法名，如果由多个单词组成，那么第一个单词的首字母小写，其余单词的首字母大写。例如playGame，sleep
   c. 对于包名，如果由多个单词组成，那么所有的字母都是小写，每一个单词之间用.隔开。例如：cn.tedu.demo
   d. 对于常量名，如果由多个单词组成，那么所有的字母都是大写，每一个单词之间用_隔开。例如INITIAL_CAPACITY

三、注释

1. 注释指的是在程序中用于进行解释说明的文字
2. 注释格式：

类型	格式	备注
单行注释	// 注释文字	
多行注释	/* 注释文字 */	
文档注释	/** 注释文字 */	不同其他两种注释，可以命令工具将其中的注释文字提取出来形成文档 文档可以理解为这个类的一份说明书 例如：javadoc -d F:\software Demo.java 文档注释一般使用在类、接口或者方法上

四、字面量

1. 将计算机中的常见数据进行分类，而且这些数据的特点是它**本身不可变**。因此字面量也称之为是计算机常量
2. 字面量分类

分类	解释	示例
整数常量	所有的整数	-3，1，10，200，0
小数常量	所有的小数	-5.2，5.5，100.08，6.0，0.0
字符常量	将一个字母、数字或者符号用单引号标识起来	'a'，'5'，'+'，' '
字符串常量	将一个或者多个字符用双引号标识起来	"ab"，"123"，"2e"，"f5"，"+"，" "，""
布尔常量	用于表示逻辑值，只有两个值	true，false

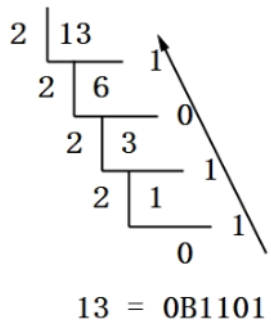
空常量	一个特殊的值，只有一个值	null

五、进制

1. 进制实际上是生活中的一种计数方式。例如现实生活中习惯上是满十进一，所以常用的是十进制。但是也有其他进制的存在，例如在时间单位中，每满60秒钟则变为1分钟，那么秒和分之间就是六十进制
2. 计算机中常用进制

进制	基本元素	特点	运算	标记	备注
二进制	0~1	满二进一	1+1=10 11+1=100	从JDK1.7开始，允许在程序中以0b或者0B开头标识一个二进制数字 所以在程序中二进制数字应写为：0b1001，0B01011等	计算机中**最基本**的机制
八进制	0~7	满八进一	7+1=10 17+1=20 77+1=100	在程序中需要以0作为开头标识一个八进制数字 所以八进制数字应该写为：05，017等	
十进制	0~9	满十进一	9+1=10 19+1=20 99+1=100	无标记	程序中的数字默认为是十进制
十六进制	0~9，A~F/a~f	满十六进一	9+1=A A+1=B B+1=C F+1=10 19+1=1A 2F+1=30 99+1=9A 9F+1=A0 FF+1=100	在程序中需要以0x或者0X开头标识一个十六进制的数字 所以十六进制数字应该写为：0x2，0X3A，0xF5A等	

3. 进制的转换
   a. 十进制转化为二进制：将这个十进制数字不断地除以2，然后取余数将余数倒序排列：

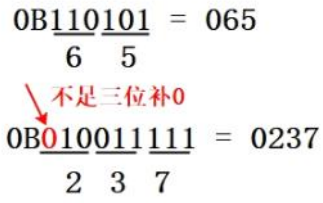


$13 = 0B1101$

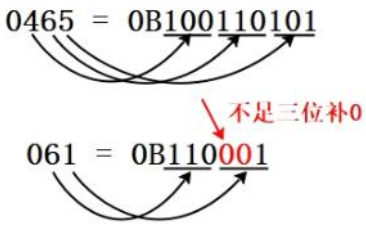
   b. 二进制转化为十进制：从这个二进制数字的低位次开始，按位次乘以2的位次次幂，然后将这些幂求和：

$$0B1011 = 1 * 2^0 + 1 * 2^1 + 0 * 2^2 + 1 * 2^3 = 1 + 2 + 0 + 8 = 11$$

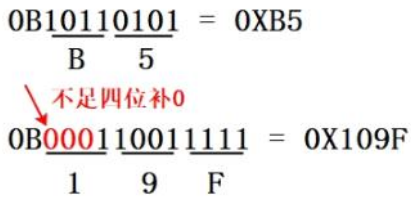
   c. 二进制转化为八进制：从这个二进制数字的低位次开始，每三位划分为一组，每一组产生一个八进制数字。如果最高位不足三位，则补0。所以二进制向八进制转化是一个三位变一位的过程：



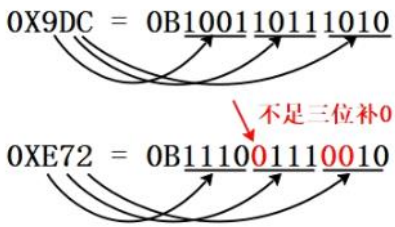
d. 八进制转化为二进制：每一位八进制数字产生三位二进制数字，如果产生的二进制数字不足三位补0：



e. 二进制转化为十六进制：二进制数字转化为十六进制的数字的过程和二进制数字转化为八进制数字的过程有些类似，是从低位次开始，每四位二进制数字产生一位十六进制数字，同样，最高位不足四位补0：



f. 十六进制转化为二进制：十六进制数字转化为二进制数字的过程和八进制数字转化为二进制数字的过程有些类似，也是从低位次开始，每一位十六进制数字产生四位二进制数字，同样，如果产生的二进制数字不足四位需要补0：



六、原码、反码、补码

1. 在计算机中，数据是以二进制形式来存储的。任意数据在计算机中都存在三种形式：原码、反码、补码
2. 原码是直接就能计算出来的二进制数据，其中该二进制数据的最高位是符号位：如果最高位为0，表示该数字是一个正数；如果最高位为1，表示该数字是一个负数
3. 原码、反码、补码的计算
   a. 对于正数而言，它的原码、补码、反码是一致的。例如：
      int i = 10;
      原码：00000000 00000000 00000000 00001010
      反码：00000000 00000000 00000000 00001010
      补码：00000000 00000000 00000000 00001010
   b. 对于负数而言，直接计算出来的是它的原码；反码是在原码的基础上保持最高位不变，其余位置上的0变1,1变0；补码是在反码的基础上加1。例如：
      int i = -19;
      原码：10000000 00000000 00000000 00010011
      反码：11111111 11111111 11111111 11101100
      补码：11111111 11111111 11111111 11101101
4. 计算机在存储的时候存储的是数据的**补码**，对数据计算，算的也是数据的补码

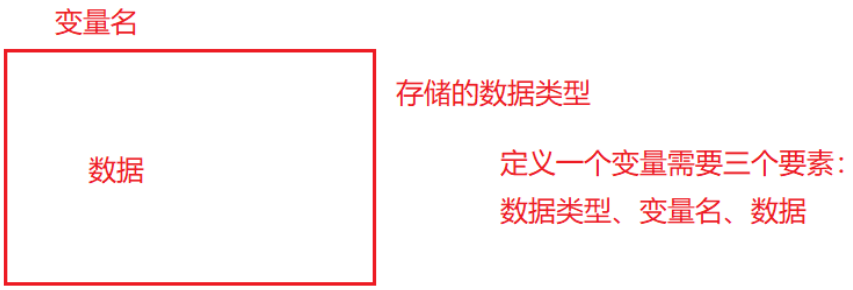
七、内存计量单位

1. 在计算机中，数据是以二进制形式来存储数据，而每一个二进制数字称之为是一个bit(位)，简写为b。1bit只能表示2种可能
2. 单位划分：

计量单位（中文）	计量单位（英文）	缩写	倍率关系
位	bit	b	0bit或1bit
字节	byte	B	1B = 8b
千字节	kilobyte	KB	1KB = 1024B = 2^{10}B
兆字节	Megabyte	MB	1MB = 1024KB
吉字节	Gigabyte	GB	1GB = 1024MB
太字节	Terabyte	TB	1TB = 1024GB
拍字节	Petabyte	PB	1PB = 1024TB
艾字节	Exabyte	EB	1EB = 1024PB
泽字节	Zettabyte	ZB	1Zb = 1024EB
尧字节	Yottabyte	YB	1YB = 1024ZB
--	Brontobyte	BB	1BB = 1024YB
--	NonaByte	NB	1NB = 1024BB
--	DoggaByte	DB	1DB = 1024B

八、变量

1. 变量是在程序中用于临时记录或者存储数据的容器。该容器中的值可以改变
2. 变量的三个要素：



3. 定义格式：

格式	示例	解释
数据类型 变量名 = 数据;	int i = 10;	表示定义了一个数据类型为int（整型）变量名为i数据值为10的变量
数据类型 变量名; 变量名 = 数据值;	int j; j = 7;	表示定义了一个数据类型为int变量名为j的变量，然后将变量j的值设置为7
数据类型 变量名1 = 数据1, 变量名2 = 数据2, ...;	int i = 3, j = 5;	表示定义了一个数据类型为int变量名为i数据值为3的变量，定义了一个数据类型为int变量名为j数据值为5的变量

4. 注意问题：
   a. 变量在哪儿定义在哪儿使用，不能超出所定义的范围使用
   b. 变量必须先定义后使用，否则在代码编译期间会出现找不到符号的问题
   c. 变量必须先赋值后使用，否则会在代码编译期间出现变量尚未初始化的问题（初始化指的是变量第一次赋值）

九、数据类型

1. 数据类型分类
   a. 基本数据类型

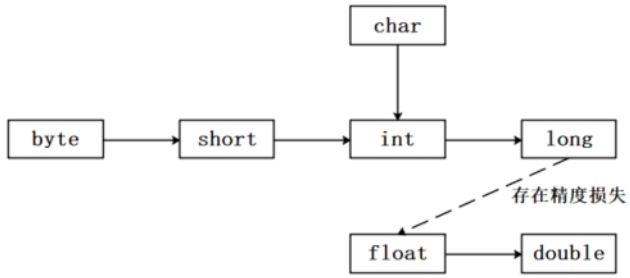
类型				内存（字节）	范围	例子	备注
数值类	整数型	字节型	byte	1个	-2^7~2^7-1 即-128~127	byte b = 125;	
		短整型	short	2个	-2^{15}~2^{15}-1 即-32768~32767	short s = 305;	
		整型	int	4个	-2^{31}~2^{31}-1 大概范围是-2.1*10^{10}~2.1*10^{10}	int i = 52689;	在Java中，整数如果不指定默认就是int类型
		长整型	long	8个	-2^{63}~2^{63}-1 大概范围是-9.9*10^{18}~9.9*10^{18}	long l = 25L;	在Java中，long类型的数据的末尾需要添加l/L作为标记
	浮点型	单精度	float	4个	大概范围是-10^{38}~10^{38}	float f = 2.53F;	在Java中，float类型的数据的末尾需要添加f/F作为标记
		双精度	double	8个	大概范围是-10^{308}~10^{308}	double d = 2.58;	在Java中，小数默认就是double类型
字符型			char	2个	0~2^{32}-1 即0~65535	char c = ‘a’; char c2 = '\u458e'; char c3 = '\t';	char类型默认采用的是UTF-16，所以一个char类型字符也可以用'\u0000'
布尔型			boolean	目前，boolean一般是占用4个字节，但是要区分JDK和操作系统		boolean b = true; boolean b2 = false;	

   b. 引用数据类型：数组[]，类class，接口interface
2. 数据类型转换
   a. 自动类型转换/隐式转换

规律	示例

当一个占用内存较小的类型的变量赋值给一个占用内存较大的类型的变量的时候会进行自动的转换	例一：
	byte b = 15;
	int i = b;
	// 例二：
	float f = 3.2f;
	double d = f;

当一个整数类型的变量赋值给一个浮点类型的变量的时候也会发生自动的类型转换 需要注意的是，当把整数赋值给浮点型的时候，可能会产生精度的损失	int i = 10; float f = i;
当把字符型的变量赋值给整数类型的变量的时候也会产生自动的类型转换	char c = ‘a’; int i = c;



b. 显式类型转换/强制类型转换

规律	示例

当把一个占用内存较大的类型的变量赋值给一个占用内存较小的类型的变量的时候，需要进行强制转换
但是需要注意的是由于较大类型和较小类型所占用的字节个数不一样，在赋值的时候需要舍弃一部分字节，所以可能导致强制转换后的数据不准确
	int i = 10;
	byte b = (byte)i;
	int i1 = 200;
	byte b1 = (byte)i1;
	00000000 00000000 00000000 11001000
	~~00000000 00000000 00000000~~ 11001000
	补码：11001000
	反码：11000111
	原码：10111000
	-(8 + 16 + 32) = -56

当把一个浮点类型的变量赋值给一个整数类型的变量的时候，需要进行强制转换。在进行强制转换的时候是舍弃所有的小数部分	double d = 3.85; int i = (int)d; // 此时i的值是3
当把任意的数字转化为字符型变量的时候需要进行强制类型转换	int i = 97; char c = (char)i; // 此时c的值是 ‘a’

十、编码

1. 按照指定规则，将字符映射成数字，这个映射过程就是编码。记录这个映射的表格称之为编码表，简称为码表
2. 在世界上，比较通用的码表：
   a. 西欧码表ISO-8859-1，在这个码表中规定一个字符对应一个字节
   b. 中国标准码表，简称为国标码gbk(gb2312)，在这个码表中规定一个字符对应2个字节。这套码表中收录了常用的简体汉字以及一部分的繁体汉字，这套码表是在中国大陆范围内使用，港澳台使用不是gbk编码
   c. Unicode编码体系，包含了很多码表，其中比较通用的是UTF-8和UTF-16码表。UTF-8中，规定一个字符对应3个字节；UTF-16中，规定一个字符对应2个字节
3. 规定其他的所有码表必须兼容西欧码表，即所有码表的前256个字符是一样的
4. 需要记住：A~Z：65~90，a~z：97~122，0~9：48~57
5. 转义字符：

字符	含义
\t	制表符
\n	换行
\r	回车
\'	单引号
\"	双引号
\\	反斜杠

运算符/操作符

一、算术运算符

1. 符号

符号	名称	示例	结果
+	正号	int i = +3;	3
-	负号	int i = -5;	-5
+	加	int i = 3, j = 5; int z = i + j;	8
-	减	int i = 10, j = 12; int z = i - j;	-2
*	乘	int i = 7, j = 9; int z = i * j;	63
/	除	int i = 20, j = 5; int z = i / j;	4
%	取模	int i = 14, j = 3; int z = i % j;	2
++	自增	int i = 8; i++;	9
--	自减	int i = 5; i--;	4

2. 注意事项

a. 在Java中，byte/short/char类型在参与运算的时候会自动提升为int类型。例如：

```
byte b1 = 3, b2 = 5;
int i = b1 + b2;
```

b. double类型在运算的时候不能报保证精确度

c. 整型在运算的时候，结果类型一定是整型。例如：

```
int i = 4300 / 1000 * 1000; // i的值为4000
```

d. 小的类型和大的类型在参与运算的时候，结果一定是大的类型。例如：

```
double d = 3.2;
int i = 5;
System.out.println(d * i); // 输出结果是16.0
```

e. 对于 / 运算，不同的数字除以零的结果不同。

示例	结果
任意整数/0	ArithmeticException（算术异常）
非零小数/0	±Infinty（正/负无穷）
非零数字/0.0	±Infinty（正/负无穷）

0/0.0	NaN（Not a Number，非数字）
0.0/0	NaN（Not a Number，非数字）
0.0/0.0	NaN（Not a Number，非数字）

f. %又叫取余运算。在运算的时候，结果的符号看的是%左边的数字的符号：
   i. 如果%左边的数字是正数，则结果是正数，例如：5 % 3 = 2; 7 % (-3) = 1
   ii. 如果%右边的数字是负数，则结果是负数，例如：(-5) % 3 = -2; (-7) % (-3) = -1
   iii. 在java中，%可以针对小数进行运算。例如：3.2 % 1.2 = 0.8
g. ++/--是自增自减运算，在运算的时候是在原来的数字的基础上自增/自减一个单位
   i. ++/--在变量之前的时候，需要先自增，然后再参与运算。例如：

      ```
      int i = 5;
      int j = ++i; // 此时i的值为6，j的值为6
      ```

   ii. 如果++/--在变量之后，需要先参与运算，然后再自增。例如：

      ```
      int i = 5;
      int j = i++; // 此时i的值为6，j的值为5
      ```

   iii. byte/short/char可以参与自增/自减运算，并且结果类型是原类型不会发生改变

二、赋值运算符

1. 符号

符号	名称	示例	结果
=	等于	int i = 3;	3
+=	加等于	int i = 5; i += 2;	7
-=	减等于	int i = 8; i -= 3;	5
*=	乘等于	int i = 10; i *= 4;	40
/=	除等于	int i = 15; i /= 5;	3
%=	取模等于	int i = 16; i %= 5;	1
&=	与等于	int i = 5; i &= 3;	1
|=	或等于	int i = 5; i |= 3;	7
^=	亦或等于	int i = 5, i ^= 3;	6
<<=	左移等于	int i = 5;	20

		i <<= 2;	
>>=	右移等于	int i = 10; i >>= 1;	5
>>>=	无符号右移等于	int i = 15; i >>>= 2;	3

2. 注意事项
   a. 在Java中，不支持变量的连等定义，但是支持连等运算。例如：

```
int i = j = 5; // 不支持
int i, j;
i = j = 7; // 支持
int i = 5;
i += i -= 4; // 支持
```

   b. 除了=以外，其余的赋值运算要求变量要先有值才能参与运算
   c. byte/short/char在参与赋值运算的时候，结果类型与原类型一致

三、关系/比较运算符

1. 符号

符号	名称	示例	结果
==	相等	3 == 5	false
!=	不等	3 != 5	true
>	大于	3 > 5	false
<	小于	3 < 5	true
>=	大于等于	3 >= 3	true
<=	小于等于	3 <= 3	true

2. 注意，关系运算符的结果类型一定是布尔类型

四、逻辑运算符

1. 符号

符号	名称	示例	结果
&	与	true & true	true
|	或	true | false	true
!	非	!true	false
^	亦或	true ^ true	false
&&	短路与	true && false	false
||	短路或	true || false	true

2. 运算规则

符号	运算一	运算二	运算三	运算四
&	true & true = true	true & false = false	false & true = false	false & false = false
|	true | true = true	true | false = true	false | true = true	false | false = false
!	!true = false	!false = true		
^	true ^ true = false	true ^ false = true	false ^ true = true	false ^ false = false
&&	true && true = true	true && false = false	false && true = false	false && false = false
||	true || true = true	true || false = true	false || true = true	false || false = false

3. 注意事项
   a. 逻辑运算针对布尔值进行运算
   b. &运算的特点是两者都成立，结果才能成立；只要有false，那么结果一定是false
   c. |运算的特点是两者都不成立，结果才能不成立；只要有true，那么结果一定是true
   d. ^运算的特点是相同为假，不同为真
   e. &&：称之为短路与，有短路特性，即&&前边的结果为false的时候，后边不再运算
   f. ||：称之为短路或，有短路特性，即||前边的结果为true的时候，后边不再运算

五、位运算符

1. 符号

符号	名称
&	与
|	或
^	亦或
<<	左移
>>	右移
>>>	无符号右移
~	取反

2. 运算规则
   a. &：将数据转化为其补码形式之后，按位次相与，将0看作false，将1看作true进行计算，最后将计算的结果转化为十进制形式显示

```
   3      00000011
& 5      00000101
-------------------
   1      00000001
```

   b. |：将数据转化为其补码形式之后，按位次相或，将0看作false，将1看作true进行计算，最后将计算的结果转化为十进制形式显示

```
  3      00000011
| 5      00000101
-------------------
  7      00000111
```

c. ^：将数据转化为其补码形式之后，按位次亦或，将0看作false，将1看作true进行计算，最后将计算的结果转化为十进制形式显示。一个数和自己本身亦或，运算结果为0：a^a=0。一个数亦或同一个数两次，那么结果仍然是原数：a^b^b=a

```
    3   00000011
^   5   00000101
-----------------
    6   00000110
```

d. <<：将数据转化为其补码形式之后，按照指定的数字向左移动，高位次移出的数字舍弃，低位次空出的位置补0，最后将结果转化为十进制显示。在一定范围内，左移几位就是乘以2的几次方

```
7 << 2      00000111
-------------------------
28          00000111 00
```

e. >>：将数据转化为其补码形式之后，按照指定的数字向右移动，低位次移出的数字舍弃，如果是正数，高位次空出补0，如果是负数高位次空出补1，最后将结果转化为十进制显示。在一定范围内，右移几位，就是除以2的几次方。正数右移，越移越小，最小到0；负数右移，越移越大，最大到-1

```
10 >> 2     00001010
-------------------------
2           00000010 10
```

```
-9 >> 2     10001001  原码
            11110110  反码
            11110111  补码
-------------------------
-3          11111101 11 补码
            11111100  反码
            10000011  原码
```

f. >>>：将数据转化为其补码形式之后，按照指定的数字向右移动，低位次移出的数字舍弃，高位次空出补0，最后将结果转化为十进制显示

g. ~：将数据转化为其补码形式之后，无论高低位，一律将0变为1，将1变为0，最后将结果转化为十进制显示。对一个数取反，实际上是获取这个数的相反数然后减1：~i = - i - 1

```
~7      00000111
-----------------
-8      11111000  补码
        11110111  反码
        10001000  原码
```

3. 注意事项：

a. 位运算是针对**整数的补码**进行计算的

b. 位运算的计算结果是补码，需要先将其转化为原码形式，才能转化为十进制形式显示

4. 交换值的方式：

a. 追尾法。这种方式相对容易掌握理解，而且适用范围最广，但是效率最低

```
int i = 5, j = 7;
int temp = i;
i = j;
j = temp;
```

b. 加减法。这种方式的效率高于追尾法，但效率是低于亦或法，适合于交换数值型的变量

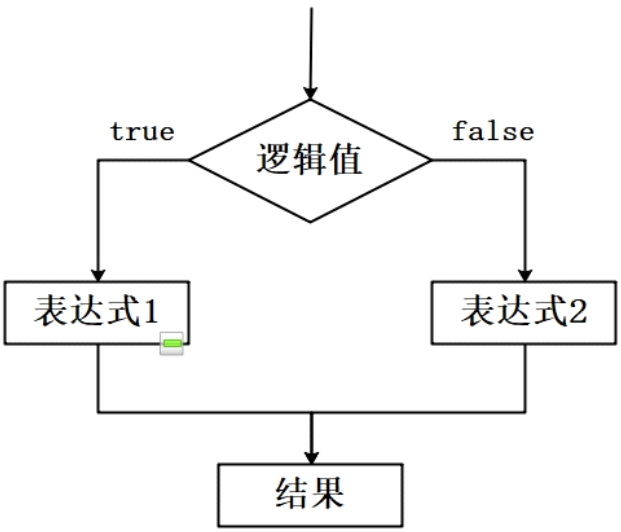
```
int i = 5, j = 7;
i = i + j;
j = i - j;
i = i - j;
```

c. 亦或法。这种方式的效率最高，但是使用范围较小，只能用于交换整数的值

```
int i = 5, j = 7;
i = i ^ j;
j = i ^ j;
i = i ^ j;
```

六、三元运算符

1. 格式：逻辑值 ? 表达式1 : 表达式2;
2. 执行顺序：先执行逻辑值，如果逻辑值为true，则执行表达式1；如果逻辑值为false，则执行表达式2：



3. ：注意事项

a. 三元运算在计算完成之后一定有计算结果，而所谓有计算结果是指能定义一个变量来记录结果。例如：int i = a > b ? 2 : 5;是允许的，但是 a > b ? System.out.println(a) : System.out.println(b);是不允许的

b. 需要注意的是表达式1和表达式2的计算结果的类型要相同或者能够自动转化

4. 练习：输入三个整数，获取较大值

扩展：

1. 从控制台获取数字：

   ```
   import java.util.Scanner;
   Scanner s = new Scanner(System.in);
   int num1 = s.nextInt();
   ```

2. 运算符的优先级：一元 > 二元 > 三元

   ()、!、~、++ --、* / % + -、<< >> >>>、关系、逻辑、& | ^、三元、赋值

流程控制

一、顺序结构

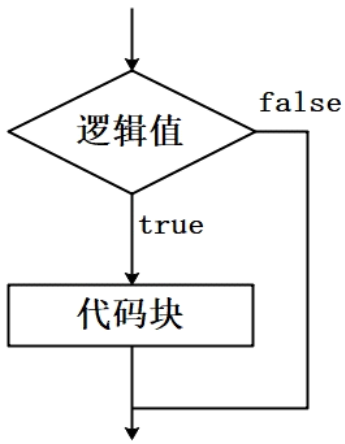
1. 所谓顺序结构是指代码从上到下从左到右依次进行编译运行
2. 在使用过程中，代码一般都是顺序结构的

二、分支结构

1. 判断结构
    a. if语句
        i. 语句规范

```
if(逻辑值){
    代码块;
}
```

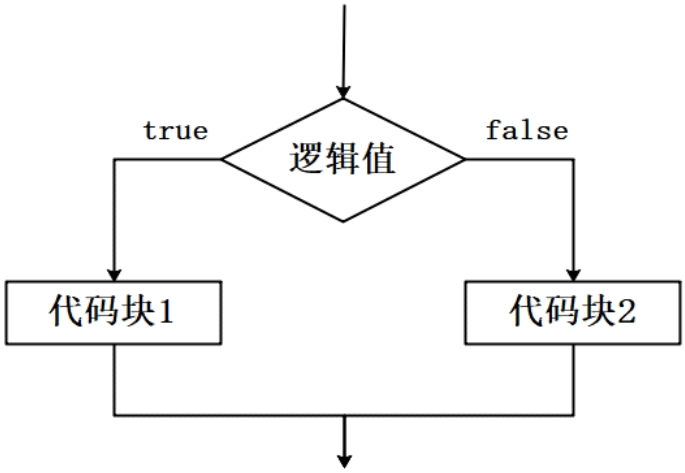
        ii. 执行顺序：先执行逻辑值，如果逻辑值为true，则执行代码块；如果逻辑值为false，则不执行代码块：



    b. if-else语句
        i. 语句规范

```
if(逻辑值){
    代码块1;
} else {
    代码块2;
}
```

        ii. 执行顺序：先执行逻辑值，如果逻辑值为true，则执行代码块1；如果逻辑值为false，则执行代码块2：

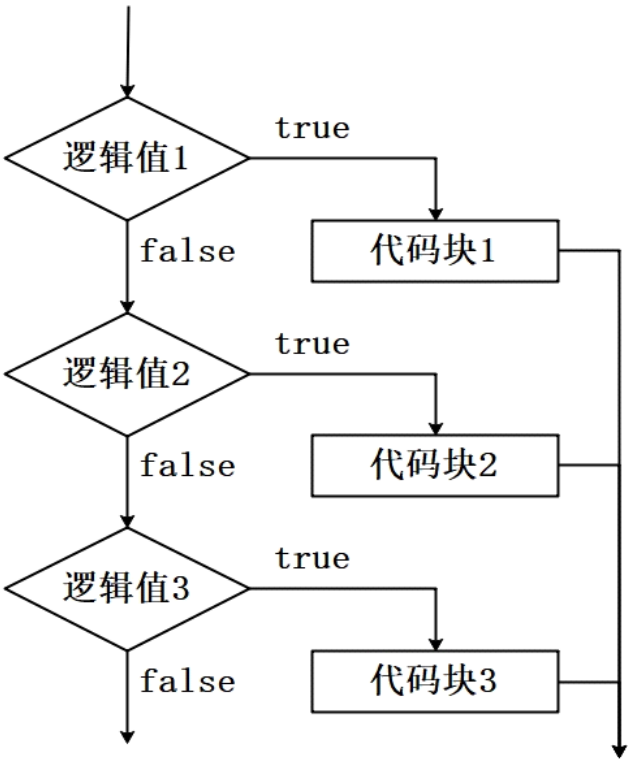


c. if-else if语句

i. 语句规范

```
if(逻辑值1){
    代码块1;
} else if(逻辑值2){
    代码块2;
} else if(逻辑值3){
    代码块3;
}...
else {
    代码块;
}
```

ii. 执行顺序：先执行逻辑值1，如果逻辑值1为true，则执行代码块1；如果逻辑值1为false，则执行逻辑值2；如果逻辑值2为true，则执行代码块2；如果逻辑值2为false，则执行逻辑值3；如果逻辑值3为true，则执行代码块3；如果逻辑值3为false，继续往下执行，依次类推：



i. 练习：

1) 输入一个数字表示月份，输出这个月份对应的季节：3-5-Spring，6-8-Summer，9-11-Autumn，12-2-Winter

2) 输入一个数字，输出这个数字对应的星期的英文：1-Monday，2-Tuesday，3-Wednesday，4-Thursday，5-Friday，6-Saturday，7-Sunday

2. 选择结构

a. 语句规范

```
switch(选项){
    case 选项1: 代码块1; break;
    case 选项2: 代码块2; break;
    …
    default: 代码块; break;
}
```

b. 执行顺序:先执行switch中的选项，如果选项和选项1符合，则执行代码块1；如果选项和选项2符合，则执行代码块2；如果选项和任何一个case都不符合，则执行default对应的代码块

c. 练习：输入两个数字以及一个符号，然后根据输入的负号完成四则运算 - 可以利用s.next()来获取符号

d. 注意事项:

i. 如果每一个case以及default之后都有break，那么case以及default的顺序不影响结果

ii. 如果default、一个或者多个case之后没有break，则case以及default的顺序影响结果

iii. switch以及case的选项的类型必须是byte/short/char/int之一，从JDK1.5开始，允许使用枚举常量；从JDK1.7开始，允许使用String

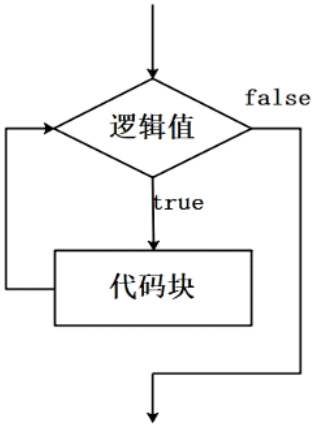
三、循环结构

1. while语句

a. 语句规范

```
while(逻辑值){
    代码块;
}
```

b. 执行顺序：先执行逻辑值，如果逻辑值为true，则执行代码块。执行完代码块再次执行逻辑值，如果逻辑值依然为true，则再次执行代码块；以此类推，直到逻辑值为false，循环结束：



c. 练习：

i. 求0-100之间所有的偶数的和

ii. 打印0-300之间的所有能被5整除但是不能被7整除的数字

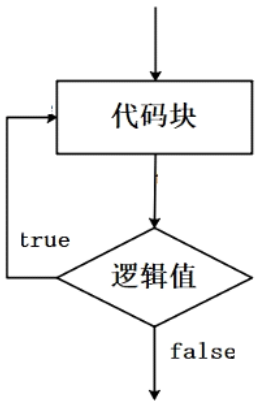
iii. 输入一个数字，输出这个数字是一个几位数。例如输入3891，输出这是一个4位数

2. do-while语句

a. 语句规范

```
do{
    代码块;
} while(逻辑值);
```

b. 执行顺序：先执行代码块，然后执行逻辑值。如果逻辑值为true，则继续执行代码块，再执行逻辑值；依次类推，直到逻辑值为false，循环结束：



c. do-while循环中，无论条件是否成立，代码块都至少执行一次

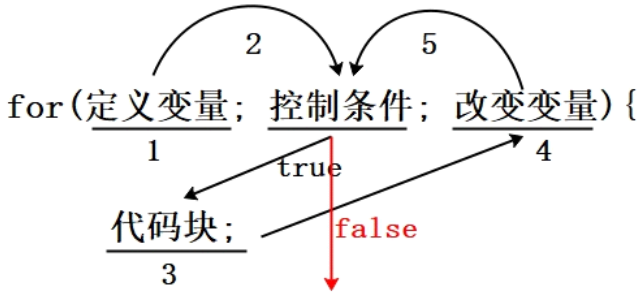
3. for语句

a. 语句规范

```
for(定义变量; 控制条件; 改变变量){
    代码块;
```

}

b. 执行顺序：先执行变量的定义，然后执行控制条件，如果控制条件的值为true则执行代码块，然后改变变量，改变完成之后再执行控制条件，如果为true再次执行代码块；依次类推，直到控制条件为false为止：



}

c. 练习：输入一个数字，判断这个数字是否是一个质数。质数：除了1和本身以外，没有别的因数。例如8有因数1,2,4,8，所以8不是质数；11的因数有1和11，所以11是质数

4. 注意事项：
   a. 循环适用于相同的或者相似的事情重复执行的场景
   b. 在循环次数不确定或者步长变化不规律的情况下，建议使用while循环
   c. 在循环次数确定或者变化比较规律的情况下，建议使用for循环
   d. do-while的循环体无论如何要执行一次
5. 循环的嵌套：
   a. 多个循环相互嵌套用于完成某个需求
   b. 练习：打印九九乘法表

四、break和continue

1. break的作用：可以作用在选择或者循环结构中，用于终止一层结构
2. continue的作用：只能作用在选择结构中，表示跳出当前一层循环继续下次循环
3. break和continue都支持标号形式

数组

一、概述

1. 在Java中，数组是一个用于存储多个同一类型的数据的容器，而且这个容器的大小是固定的
2. 数组会对存入其中的元素进行自动的编号，这些编号是从0开始计算的，称之为是下标
3. 数组中存储的数据称之为是元素，数组的大小也叫数组的长度

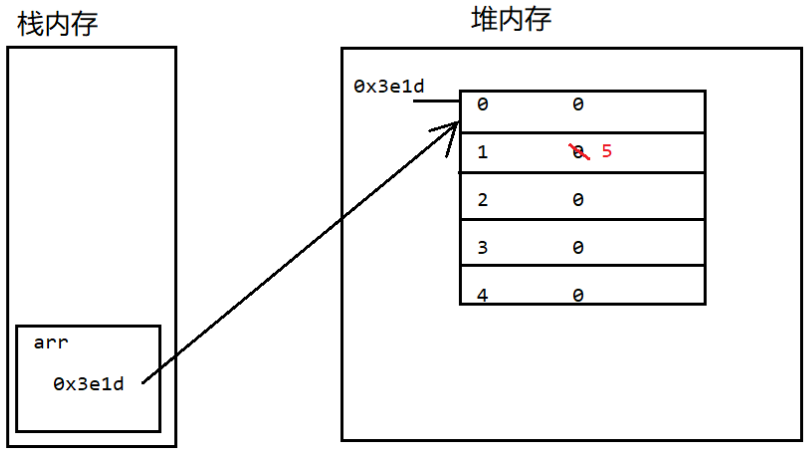
二、定义格式

格式	示例	说明
数据类型[] 数组名 = new 数据类型[数组的长度或者是元素的个数];	int[] arr = new int[5];	表示定义了一个名为arr的数组，该数组中能存储5个数据类型为int的元素、
数据类型[] 数组名 = new 数据类型[]{元素1, 元素2, 元素3, ...};	int[] arr = new int[]{2, 8, 9, 1, 3, 5};	表示定义了一个名为arr的数组，数组中存储的元素为2, 8, 9, 1, 3, 5, 数组的长度为6
数据类型[] 数组名 = {元素1, 元素2, 元素3, ...};	int[] arr = {2, 8, 9, 1, 3, 5};	表示定义了一个名为arr的数组，数组中存储的元素为2, 8, 9, 1, 3, 5, 数组的长度为6

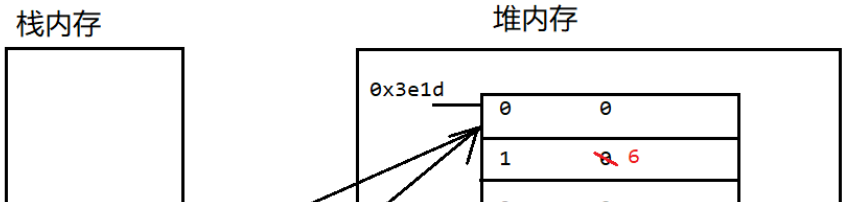
三、内存存储

1. 在Java中，将内存进行了划分，划分为不同的区间（栈内存、堆内存、方法区、本地方法栈、PC计数器），其中比较重要的是栈内存以及堆内存两块空间
   a. 栈内存用于存储变量。变量在使用的时候是存入栈内存中的，当使用完之后会立即从栈内存中移出。变量在栈内存中不会自动赋值
   b. 堆内存用于存储对象，而数组是对象的一种。当数组存入堆内存中的时候，堆内存会自动对数组中的每一个位置上赋予一个默认值，其中byte/short/int类型的默认值是0，long类型的默认值是0L，float类型的默认值是0.0f，double类型的默认值是0.0，char类型的默认值是‘\u0000’，boolean类型的默认值是false，其他类型的默认值都是null。对象在使用完成之后并不会立即从堆内存中移出而是在不确定的某个时刻被回收
2. 数组的内存存储

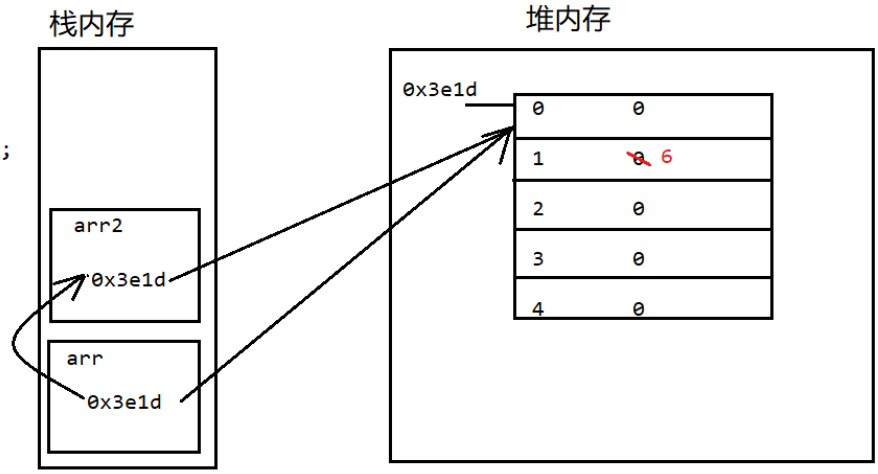
```
int[] arr = new int[5];
arr[1] = 5;
```




```
int[] arr = new int[5];
int[] arr2 = arr;
arr2[1] = 6;
System.out.println(arr[1]);
                   6
```



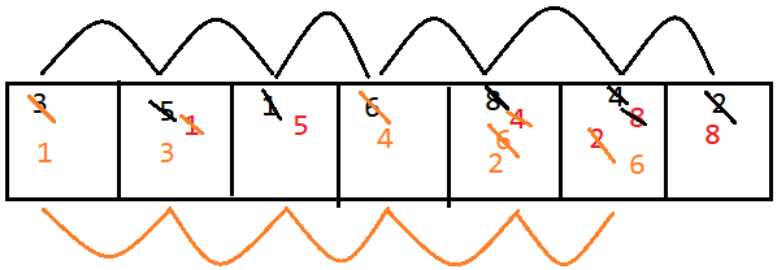
```
int[] arr = new int[5];
int[] arr2 = arr;
arr2[1] = 6;
System.out.println(arr[1]);
                   6
```



四、数组应用

1. 获取数组元素：通过 数组名[下标] 的方式来获取对应下标位置上的元素
2. 获取数组长度：通过 数组名.length 的方式来获取数组的长度。需要注意的是数组的长度是一个不可变的值
3. 数组元素遍历
   a. 普通for循环：因为数组中的下标是依次递增的，所以可以先遍历下标，通过下标获取对应位置上的元素
   b. 增强for循环：可以直接获取数组中的元素
   c. Arrays.toString()：将数组中的元素拼接成字符串来进行展示
4. 获取元素最值（最大值/最小值），例如获取最大值：
   a. 直接记录数组中的最大值
   b. 直接记录数组中的最大值的下标
5. 数组元素排序
   a. 冒泡排序：相邻两个元素按照指定规则两两比较，如果不符合规则则交换位置。冒泡排序的时间复杂度是$O(n^2)$，空间复杂度是O(1)

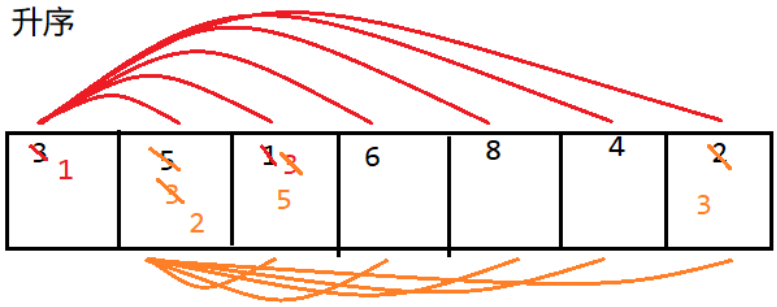
   升序排序：从小到大



   冒泡排序：相邻两个元素两两比较，按照指定顺序来进行交换

   比较轮数i：n-1
   比较次数j：n - 轮数
   每一次要比较的下标：j - 1和j

   b. 选择排序：选定一位元素依次与数组中其他位置上的元素按照指定的规则进行比较，如果不符合指定的规则则交换位置。选择排序的时间复杂度是$O(n^2)$，空间复杂度是O(1)



选择排序：选定一位和其他位来依次比较，如果不符合规则则交换位置

轮数i：n - 1

每一轮的比较次数j：n - i

每一轮要确定的位置：i - 1

每一次要比较的位置：j

c. Arrays.sort：Arrays.sort是Java中原生提供的用于数组排序的方法，但是只能进行升序排序。Arrays.sort是基于快速排序和归并排序进行计算的，从JDK1.7开始，对Arrays.sort进行了调整，利用了快速排序和合并排序，所以时间复杂度是O(nlogn)

6. 数组元素查找
   a. 如果数组无序，那么使用遍历数组的方式来查找指定的元素
   b. 如果数组有序，可以使用二分查找的方式找指定的元素。二分查找的时间复杂度是O(logn)
7. 反转数组：将数组的首尾对应位置上的元素进行交换，就能将数组反转过来：
8. 数组的复制以及扩容
   a. 数组的复制：数组的复制是利用System.arraycopy方法来完成的。其格式如下：
   System.arraycopy(要复制的数组, 复制的起始位置, 要存放的数组, 存放的起始位置, 复制的元素个数)
   b. 数组的扩容：数组的扩容实际上是基于数组的复制来完成的，所以数组在扩容完成之后的地址会发生改变

五、注意事项

1. 如果访问的数组下标超过了数组的最大下标，则会出现数组下标越界异常ArrayIndexOutOfBoundsException
2. 数组本身是引用类型，所以可以赋值为null。当数组被赋值为null的时候，表示数组不指向任何地址

六、二维数组

1. 可以将数组看作一个元素存入另一个数组中，那么这个时候就形成了一个存储数组的数组，即二维数组。可以将之前学习的数组看作是一个小盒子中存放了很多东西，那么二维数组就是一个可以存储多个小盒子的大盒子。
2. 定义格式

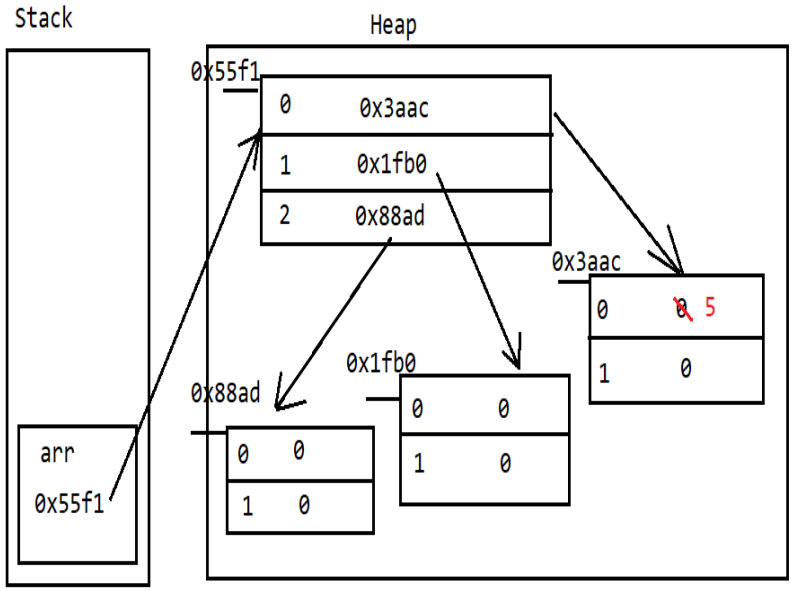
格式	示例	解释
数据类型[][] 数组名 = new 数据类型[存储的一维数组的个数][一维数组中元素的个数];	int[][] arr = new int[3][5];	表示定义了一个二维数组arr，arr中能存储3个一维数组，每一个一维数组能存储5个int类型的元素
数据类型[][] 数组名 = new 数据类型[存储的一维数组的个数][];	int[][] arr = new int[3][];	表示定义了一个二维数组arr，arr中能存储3个一维数组

数据类型[][] 数组名 = {数组1, 数组2, ...};	int[][] arrr = {{2, 5}, {1, 3, 2, 6}, {3, 8, 9}};	表示定义了一个二维数组arr，arr中存储了3个一维数组，第一个数组中存储的元素是2, 5；第二个数组中存储的元素是1, 3, 2, 6；第三个数组中存储的元素是3, 8, 9。

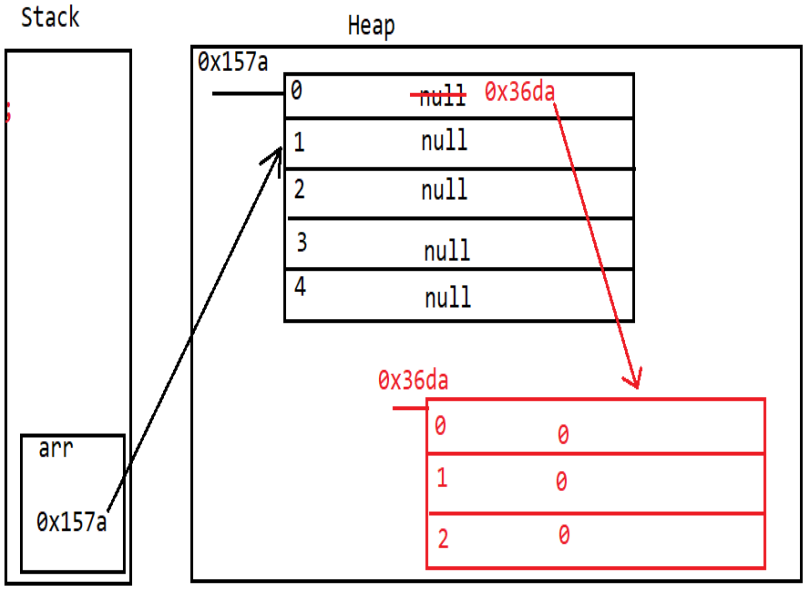
3. 内存存储：二维数组的每一个位置上存储的是对应的一维数组的地址

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

arr[0][0] = 5;
```




```
int[][] arr = new int[5][];
arr[0] = new int[3];
System.out.println(arr[0][0]);
```



4. 练习：杨辉三角

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
1    6    15   20   15   6    1
```

输入一个数字n表示行数，输出前n行的杨辉三角

方法

一、概述

1. 在书写代码的过程中，如果发现某段代码需要重复使用，那么可以将这段代码提取出来封装成一种新的形式，那么这种新的形式称之为方法。方法也叫函数
2. 定义格式

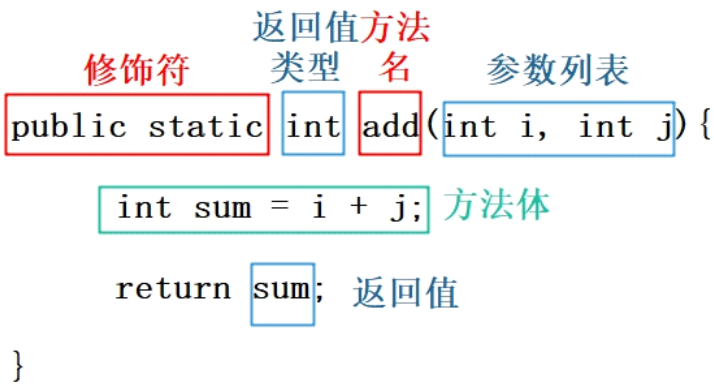
   修饰符 返回值类型 方法名(参数列表){

   方法体;

   return 返回值;

   }

   例如：



3. 注意事项
   a. 明确方法的结果类型。当方法计算完成之后结果是什么类型那么返回值类型就是对应的类型。如果方法在计算完成之后没有结果，那么返回值类型定义为void
   b. 明确方法的未知量。在定义方法的时候，需要确定方法在执行过程中是否有未知量参与运算，如果有未知量参与运算，那么需要以参数的形式在方法中体现定义
   c. 方法在定义的过程中声明的参数称之为形式参数，简称为形参
   d. 方法在调用的时候传入的数据称之为实际参数，简称为实参
   e. 方法名和参数列表合称为方法签名
4. 练习：定义一个方法，传入一个整数，判断这个整数是否是一个质数

二、方法的重载

1. 当在同一个类中存在方法名相同而参数列表不同的方法的时候，就构成了方法的重载
2. 注意事项:
   a. 参数列表不同指的是参数个数不同或者是对应位置上的参数类型不同，实际过程中考虑调用方法的时候是否会产生歧义
   b. 当方法构成了重载，那么在调用方法的时候会遵循最优匹配原则，即在方法名相同的前提下，哪一个方法的参数列表最相近则匹配哪个，但是会出现多个匹配优先级一样的情况，所以在实际过程中，一旦出现重载要考虑将所有的情况都尽量重载
3. 练习：定义一个方法，求数字类型的数组中所有元素的和

三、方法的递归

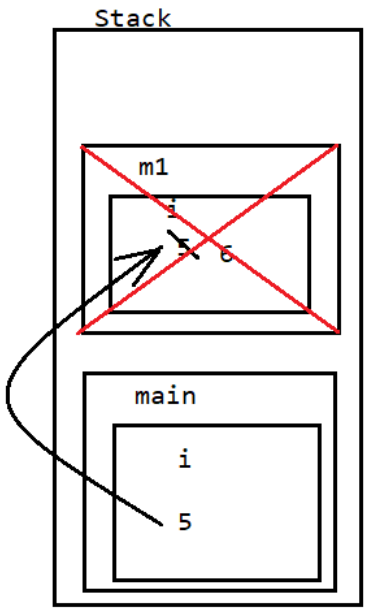
1. 当在一个方法中调用自己本身的时候就形成了方法的递归
2. 注意事项
   a. 方法的递归一般使用的逆推的思想，即找这一项与前一项或者前几项的关系
   b. 方法的递归次数如果过多，会导致程序出现StackOverflowError（栈溢出错误）--- 方法是栈内存中执行的
3. 练习：利用递归来求n!。例如5!=5*4*3*2*1=120

四、方法的传值

1. 对于基本类型的数据而言，传值传递的是实际值，所以在新方法中改变数据不会导致原方法中的数据产生改变：
2. 对于引用类型的数据（例如数组）而言，传值传递的是地址，所以在新的方法中产生改变的时候会影响原来的方法。但是如果在新的方法中改变了地址，则不再影响原方法

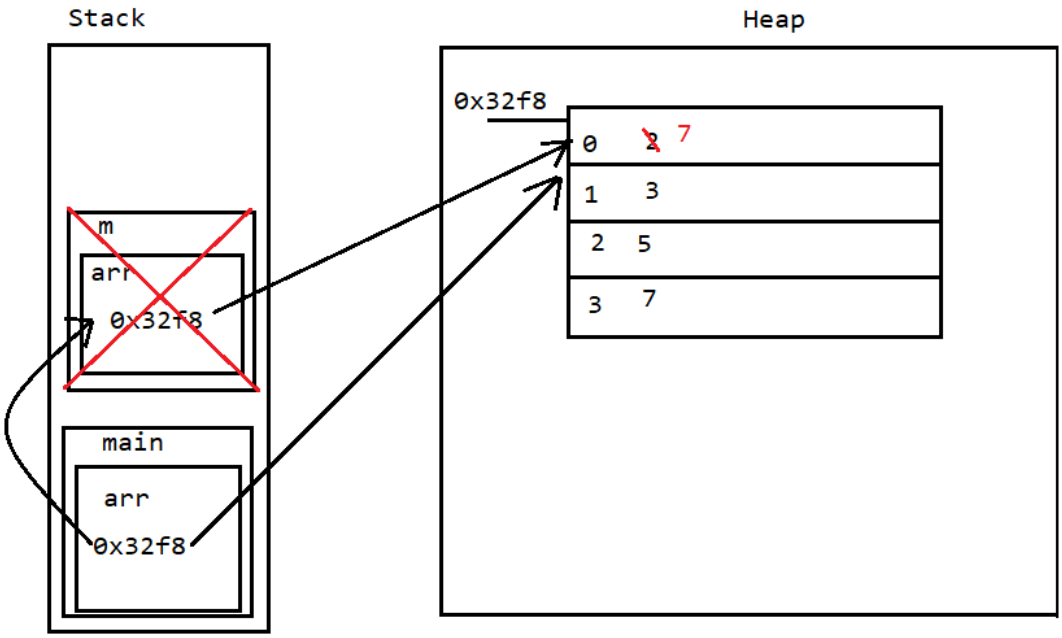
```
public static void main(String[] args){
    int i = 5;
    m1(i);
    System.out.println(i);
}

public static void m1(int i){
    i++;
}
```



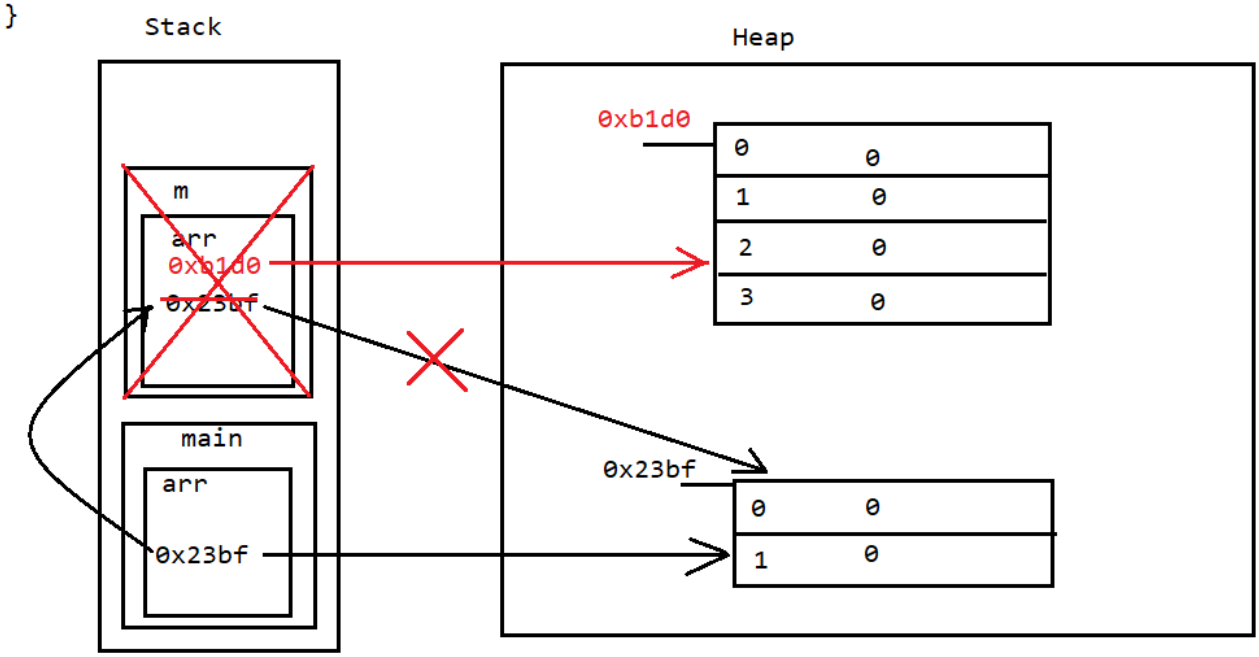
```
public static void main(String[] args){
    int[] arr = {2,3,5,7};
    m(arr);
    System.out.println(arr[0]);
}

public static void m(int[] arr){
    arr[0] += 5;
}
```




```
public static void main(String[] args){
    int[] arr = new int[2];
    m(arr);
    System.out.println(arr.length);
}

public static void m(int[] arr){
    arr = new int[4];
}
```



五、IDE

1. 目前在世面上常用的智能开发工具：Eclipse、Intelli J

2. IDEA中只需要Ctrl+S保存一下就可以进行编译