本文由 [简悦 SimpRead](http://ksria.com/simpread/) 转码， 原文地址 [blog.csdn.net](https://blog.csdn.net/qq_39997939/article/details/123449889?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522F414ED57-BB2C-498E-9FA5-5F013062A090%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=F414ED57-BB2C-498E-9FA5-5F013062A090&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-2-123449889-null-null.142^v100^pc_search_result_base5&utm_term=java%E5%9F%BA%E7%A1%80&spm=1018.2226.3001.4187)

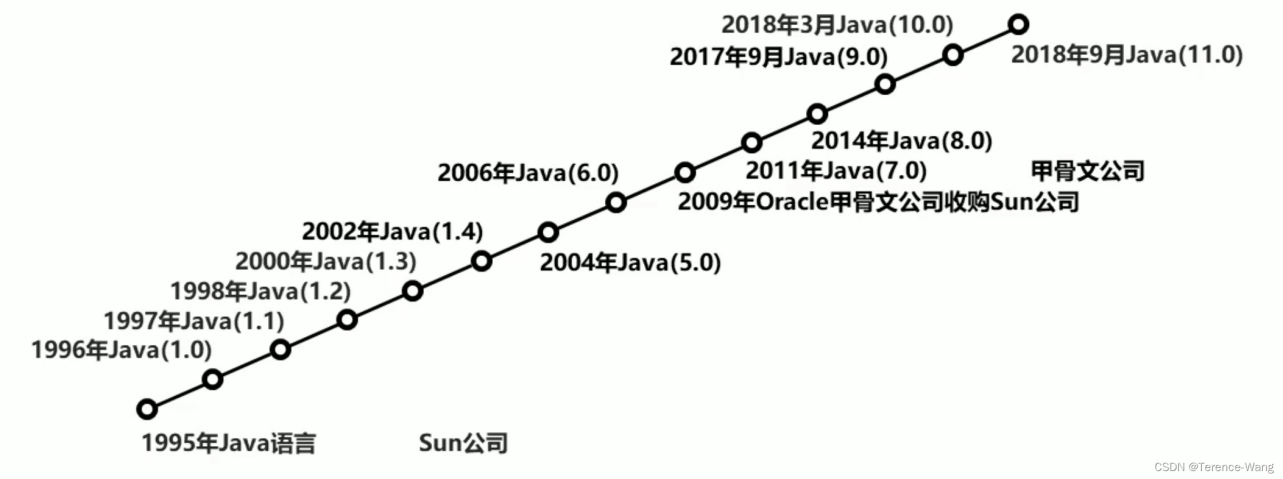
#### 文章目录

* [1. Java 概述](#X4613fa3d7c407d90dd3f9c6e22052d6ba2a63a2)
  + [1.1 Java 语言发展史](#X2239564b2f2f7da1a73c63817174f3860b6038e)
  + [1.2 Java 语言跨平台原理](#X1caece3321c8094f828b01dc52c5e17e57c03e1)
  + [1.3 JRE 和 JDK](#X0b42d6c3e7280ca2ae9394de0e78522e665c701)
  + [1.4 JDK 的下载和安装](#Xa924df301327a562bb45f580d5737913d8b319d)
    - [1.4.1 下载](#Xf99156afe6fe909430d3c5d6613c8ea46a12366)
    - [1.4.2 安装](#X7e5253b896c9a1dad2857a409ee6d9a1ede0031)
    - [1.4.3 JDK 的安装目录介绍](#X1ed1e605c7e2306319ef7942067a9801fd50158)
* [2. 第一个演示程序](#X278c8804037130d5cbd66add607f49193ef4bc6)
  + [2.1 常用 DOS 命令](#X4fa39a019a63dfd8fd0ffb2dcd884e3cfc00d6c)
  + [2.2 Path 环境变量的配置](#Xc4008771d07ce7aa087ecda5b91c5c7968be796)
    - [2.2.1 为什么配置环境变量](#X9cf531003a8fe1cb3941b82279326add6040c49)
    - [2.2.2 配置环境变量步骤](#X4bd83c33510f8dbe3a4932c73b11b6bd3438b93)
  + [2.3 HelloWorld 案例](#X549b7aca776538f8cc8acb1a28986446a44fcbc)
    - [2.3.1 Java 程序开发运行流程](#X8140be2b67373d7c53ebc7b3ef7b8a9aa0db8ff)
    - [2.3.2 HelloWorld 案例的编写](#X13be49d8a7cd781c039a400333f541cf72b6cab)
    - [2.3.3 HelloWorld 案例的编译和运行](#Xd096ddb57c4ead892f6d17378b8bad79ffe677d)
  + [2.4 HelloWorld 案例常见问题 (理解)](#X1b7c8b7e110933ff71ce243d425cf987a50517d)
    - [2.4.1 BUG](#X8e2ce9d4b42b446d2efc068c8f5a541da5a1235)
    - [2.4.2 BUG 的解决](#Xdc1677c90dd232864702cdfdaee7384ada00030)
    - [2.4.3 HelloWorld 案例常见问题](#Xfdc3fb0c801e690cbe46bcf714beaf140d5f87e)
  + [2.5 Notepad++ 软件的安装和使用](#X709766309ddad20b9e75635b7ece5da2414aae3)
    - [2.5.1 什么要使用 Notepad++ 软件](#X38ab7a697a61aa21c93ee3176ce7e28efbd7adf)
    - [2.5.2 Notepad++ 软件安装](#Xcc7280754673083de8f73a36cd027018a121164)
    - [2.5.3Notepad++ 软件配置](#X85658a3f94a9932ba9594d7d9f307e2bf924051)
* [3. java 基础语法](#X0d1daf03455285e610bded9aa2828160cb5e74b)
  + [3.1 注释](#X85e3095db71d2444998e3e27909b3c20fd79dd6)
  + [3.2 关键字](#X4e0487ce1416c4306389e6e950073d4a5d6e71b)
  + [3.3 常量](#Xddafea9f08349fa1bd63b883d7874cba4a0b543)
  + [3.4 数据类型](#Xbae2ac0656fce67d89945faafb9d4092e5d7fdf)
    - [3.4.1 计算机存储单元](#X3fa6fbb584dfe3386951baf1a9af58b94226a7d)
    - [3.4.2 Java 中的数据类型](#Xbd2131b19b839fc495104e27ea41d58f1ac3814)
  + [3.5 变量](#X8a847e0c846f2bbecb09fdca9e5c2ad20c9d165)
    - [3.5.1 变量的定义](#X9f812317440f9fed27230dfe0bb3aa3a9a3843b)
    - [3.5.2 使用变量时的注意事项](#X105a8798809e95a2e41f4397e2cbbee09554127)
    - [3.6 标识符](#X3670f983c57749c62e4b0ca046341f05e80080b)
    - [3.7 类型转换](#Xcc73a87557d8a3437a6e83b36fd58339d37060e)
* [4. 运算符](#Xeb9c74307d073bfbdd60cb62168fb0483e3aeb2)
  + [4.1 算术运算符](#X95f37c91d5ca1ea5c377274638cf99f595baa88)
    - [4.1.1 运算符和表达式](#X700ad76eb742c48140fd6ecfd4857a8c053211a)
    - [4.1.2 算术运算符](#Xc3e1def56f1e1a9bdd52c80f9efa22c06e18a79)
    - [4.1.3 字符的 “+” 操作](#X0ad15c137ab4fcde6e5054a20a02e326e4b5f3e)
    - [4.1.4 字符串的 “+” 操作](#X7dd9f79cd93a86a1a77990d77726e1bc03af7b8)
  + [4.2 赋值运算符 (应用)](#Xc8f2cda81023739d321fb5d0c4a18631d19c1ab)
  + [4.3 自增自减运算符](#Xd7d4fdbc9870cf131d6aa76ea877b2e1c0ea0ff)
  + [4.4 关系运算符](#X69a03682b074ce764f63a7923da30618da8971a)
  + [4.5 逻辑运算符](#X34d23cdc07c820b06014faf49f63b21e360bddc)
  + [4.6 三元运算符](#X83d297ca63c7f658e239fc2f4201d1868418634)
* [5. 数据输入](#Xb07cee4f15f4ff81a2c2591eb8bc2bd1c5dc09d)
* [6. 流程控制语句](#X4e4e17026333a595f12079717bb73de880452d3)
  + [6.1 流程控制语句分类](#X54ae8ad23e0e2830690b1ceb8487b2f33cc180e)
  + [6.2 顺序结构](#X9e07b82d9d09a79b29a851ae5f5bd5f720d9f43)
  + [6.3 分支结构之 if 语句](#Xe7bf8c0cd5fb64b272afebc3f008d90839285c2)
    - [6.3.1 if 语句格式 1](#Xd5845e4be8d515ab1dab2ce7dab12dc15e72c58)
    - [6.3.2 if 语句格式 2](#Xadbcaffafcb2fffc7da9e512002ed3989527fed)
    - [6.3.3 if 语句格式 3](#X0c05e77e3ad23a093e18327252445a3094405f4)
* [7. switch 语句](#X75733142c647876a6b2f79c546c8427072e8df3)
  + [7.1 switch 语句结构](#Xa811001ad54c3bba94453dc553e446af443a8bc)
  + [7.2 switch 语句练习 - 春夏秋冬（应用）](#X9915cda3150ab6133f2ae3b780025505fdc515a)
* [8. for 循环](#Xd399493538e8459727e101c7f48299d0b1ca847)
  + [8.1 for 循环结构（掌握）](#X9f260615cb8bfe5dc8b7dab46e85828c07f8267)
  + [8.2 for 循环练习 - 输出数据（应用）](#Xf722c2793ec208b74655ecfbc94e4bac31e09d6)
  + [8.3 for 循环练习 - 求和](#X9e232eb16e07cbe85195fcee8f0dcaa80779ea2)
  + [8.4 for 循环练习 - 求偶数和](#Xc78f891ddc61c4060ad93efd6eb2c66d30852f0)
  + [8.5 for 循环练习 - 水仙花](#X282d99bcfb018334705fcb101471a4cb9327f35)
  + [8.6 for 循环练习 - 统计水仙花数个数](#Xef7bdba63bb60b8a6baeca02dae5e8f1127b15c)
* [9. while 循环](#Xbe276c303394cecd0d855049b83adce2395b24a)
  + [9.1 while 结构](#X7158584152364e0ce49d7b67a9142218a133802)
  + [9.2 while 循环练习 - 珠穆朗玛峰](#Xf5538a256968a55850ebd26a21f54b1b5c2534c)
* [10. 循环细节](#Xfc0400a3b05d9acf583c3831830e4c999907109)
  + [10.1 do...while 循环结构](#X5752f499afa51dda986651b79728900e05a2f5c)
  + [10.2 三种循环的区别](#X66db8a7445310c31212ceae3c381917cbd36458)
  + [10.3 跳转控制语句](#Xf2da94cd5277d3dc75fa69b9feef1a36cb4aa82)
  + [10.4 循环嵌套](#X1aa3bfd5620befb902001904111177686d48786)
* [11. Random](#X7c58dfd42a43b77b216c361903b61cb00d26fa0)
  + [11.1 Random 产生随机数](#Xd94ae92ea26544dd4add3bccd2666765ca69fe9)
  + [11.2 Random 练习 - 猜数字](#X64b78e5f3a3bb296d3c0992b63ceae1f1fe8cf1)
* [12. IDEA 开发工具](#X983b279f6abf241076354f4729370872ee56799)
* [13. 数组](#Xbc593cf941e2400fdd3a7ec2c712a916d80c4d6)
  + [13.1 什么是数组](#X7af5e2f6068d5a850900e086eab621f7431f5d1)
  + [13.2 数组定义格式](#Xf3e100613fe35059a64e6fc2dcf353e6001f2c6)
    - [13.2.1 第一种](#Xa3de6b494121f81f65c1ad12d39bdd7f3531b1e)
    - [13.2.2 第二种 (不推荐)](#Xaba747c4d0c5713c07b276c6e9eb445b223ecc0)
  + [13.3 数组动态初始化](#X872ff61cd6781859e64f41397bf7e1e05ce19b5)
    - [13.3.1 什么是动态初始化](#X2e570ec984b2be1e1c7facc63ce66ec0ed9ddf0)
    - [13.3.2 动态初始化格式](#Xe213a39f1554aea742d9672b69574414462814f)
    - [13.3.3 动态初始化格式详解](#X96a4d29590e77f38c4a2716af576a19c94eadf6)
  + [13.4 数组元素访问](#Xb7c970b24b5de22762de2555263b1cb1d3f0e8a)
    - [13.4.1 什么是索引](#X38f9dc7cf38a5e0e03b90f1a90cc8988d544454)
    - [13.4.2 访问数组元素格式](#X1d33e4f533e2aa54dc3c6f2b52bd714b1222972)
    - [13.4.3 示例代码](#Xcc00da43a4a6998f13ad2347e8c4da19ea7eb3d)
  + [13.5 内存分配](#X458290a2ac2f8f1ca062ed5340400490862445a)
    - [13.5.1 内存概述](#Xb3ac56aba0e2b4e5a061037c14a5232de615ba4)
    - [13.5.2 java 中的内存分配](#X2d3636edeeb971c3e822174b6bda95d560337d2)
  + [13.6 单个数组的内存图](#X1cce53068c7bc2b10b302224488cc417d1050ea)
  + [13.7 多个数组的内存图](#Xdc276ef0c5f9554bde938a5571d2c029a63fefd)
  + [13.8 多个数组指向相同内存图](#X8c8954b42d09806615896d2e23dc757117dc8df)
  + [13.9 数组静态初始化](#Xda8b87543fc4b20c366bc78aeee4a9bbe4c4d49)
    - [13.9.1 什么是静态初始化](#Xb9f2557328dabc98a5be650d7317ccdd8a64d78)
    - [13.9.2 静态初始化格式](#X975aa67f120f18e08a7be8505a137e435285bf5)
    - [13.9.3 示例代码](#X0f2f43093a1d5f6728bb22a9259e5b80265bce2)
  + [13.10 数组操作的两个常见小问题](#X4c14e1003bba20ed1251d4f282cafb5dc56a185)
    - [13.10.1 索引越界异常](#X1b6c04f32722707f0e4e5210169ab7a022ec939)
    - [13.10.2 空指针异常](#Xbde4aaa9876f626f223aa28166738ee0e9e5ac6)
  + [13.11 数组遍历](#Xa683505e0b7d33792f63d2971642243c5436532)
  + [13.12 数组最值](#X69baa7dd38d685463181f1cf77633dc1621b6e2)
* [14. 方法概述](#X29b2ecbf0b99cd5f9a337f487e36f39a6363746)
  + [14.1 方法的概念](#X903b98c3cca2f9056ad6e9fb696405138cb7eff)
  + [14.2. 方法的定义和调用](#Xdfe680196daa540b77559952c9de6a6059d4dfd)
    - [14.2.1 无参数方法定义和调用](#X91512130209f3b017df79b19d6303015ebd1cf4)
    - [14.2.2 方法调用过程图解](#Xc448959e2e5fdbebf7fd79bd9a4e2b32a0e3b47)
    - [14.2.3 无参数方法的练习](#Xa72a0335cd2565da8702f97ac9845fc5bd2c87c)
  + [14.3 带参数方法定义和调用](#Xfa10048a8a32db9848d5b3c82a712f61de07118)
    - [14.3.1 带参数方法定义和调用](#Xced8866b78fc2d2ddc104bbc28c92a2a1697754)
    - [14.3.2 形参和实参](#Xe9c608b7cebd0799af98b987cadf3c2fe91c1ee)
    - [14.3.3 带参数方法练习](#X97922739aa6399398738c7583f56daaefc90004)
  + [14.4 带返回值方法的定义和调用](#X8744b126112f3ddbbe741fccd7183ac8a4a44df)
    - [14.4.1 带返回值方法定义和调用](#X2f1c21ab38a9077eace3fcbc9aab0dfeb859331)
    - [14.4.2 带返回值方法练习](#Xa78439c399212c81836623711c74742db3cf91b)
  + [14.5 方法的注意事项](#X5aba7ee718e156381e3d573a75408566d9de94b)
    - [14.5.1 方法的注意事项](#Xfdc77e49a9e3b58e9b5f87d7c1dbb4f181027b2)
    - [14.5.2 方法的通用格式](#X3f089506ec5d5f4c66b45144591e3f271893207)
  + [14.6 方法重载](#Xd06e99abc13863241c84569d3a3023a281bd232)
    - [14.6.1 方法重载](#X34f08dc152434b968ead6838e4acaba8637ea95)
    - [14.6.2 方法重载练习](#X645356b872c3cdc0d277a3abad34530f3237af0)
  + [14.7 方法的参数传递](#X27ef53e17103c04d569c1163efee4d93c9659ec)
    - [14.7.1 方法参数传递基本类型](#X15ef5f25bd64af540db89f2e10274fdde33d59e)
    - [14.7.2 方法参数传递引用类型](#X82f1f59998240688bd4e6789ee92b75a6f89774)
    - [14.7.3 数组遍历](#X925c5f68cbb1c3321977add25565cac2f6d6924)
    - [14.7.4 数组最大值](#Xce67449ce9d5245305c19ffb3e7ee95e2cc81ad)
* [15. Debug 模式](#X341610fecaa9c13992d4567f52bb3a36690e9e8)
  + [15.1 什么是 Debug 模式](#X2a780a6f83a4d2702c62fb7129025bcb3cb7223)
  + [15.2 Debug 模式操作流程【应用】](#Xb92d72412f4d421036f1010a5e1abb0a7f884b9)
* [16. 基础练习](#X51370bf6b7a8fee9667869683d2ac58d0fddfda)
  + [16.1 减肥计划 if 版本](#Xc924ae3116f5209f4934b6913a7e265a50c375c)
    - [16.1.1 案例需求](#X70a92893f7b80d7a47f171a8ce020d423fb5251)
    - [16.1.2 代码实现](#X1885788fe858175658614823677f5873b0094ef)
  + [16.2 减肥计划 switch 版本](#Xc1a77648fe02369b61ab4a97137ce4e9037a489)
    - [16.2.1 案例需求](#X09cbaa63a771bfb0192a27af245d883d3179166)
    - [16.2.2 代码实现](#X98620b86a1090f19b09aafa416036eead7fc361)
  + [16.3 逢七跳过](#X953978bac6d67f16bd1c726cefaf831f1033996)
    - [16.3.1 案例需求](#Xb794b97ee26ef9d06e13960c6f09e7a662469c0)
    - [16.3.2 代码实现](#X82025701308c96a7ac3346283e1f0eada5ebf92)
  + [16.4 不死神兔](#X426b622398af170085c2ade94ade02aa0ad358b)
    - [16.4.1 案例需求](#X324936117f9442f2d7f48f2128d659a4c75e0cb)
    - [16.4.2 代码实现](#X4a7fef0053f0bbbcf3500e31c375915c9090ede)
  + [16.5 百钱白鸡](#Xf5d5cfd07b08aacbc243994c88ece3a1c519c97)
    - [16.5.1 案例需求](#X0f1be38d70c518b14e1fa8e15745da5ed2f755a)
    - [16.5.2 代码实现](#Xfcc1b8f02997ea396809a8bf9cfce0bd2401b1f)
  + [16.6 数组元素求和](#X1358aaebb49361762df20684f048aa2d8b3bb1d)
    - [16.6.1 案例需求](#X3c92ccd98d68e7f6045fa7bf476dc7421877c46)
    - [16.6.2 代码实现](#X548fd99e90c64c0bfb4a7c3476c3dfa61014a37)
  + [16.7 判断两个数组是否相同](#Xfbaa01d419e5611c7a814fd874e813444fc77c9)
    - [16.7.1 案例需求](#X0cabebae94fd1d5576beb2777903f94226d46fa)
    - [16.7.2 代码实现](#X2d1e46bf1856f884c3915f20949162a762fff9c)
  + [16.8 查找元素在数组中出现的索引位置](#Xb62f286124df1f15818f7e84e7adfdf9f859d3e)
    - [16.8.1 案例需求](#X3f1d3b0bab94cae6f193ae4f8c8530f58eaeef8)
    - [16.8.2 代码实现](#Xf7d738259bf46d9ff94854a9df032f1fe89db8e)
  + [16.9 数组元素反转](#X24256a05410c4f608191d73d2713986a04210bc)
    - [16.9.1 案例需求](#Xca10ac4f044890adc183fa9697ab3e6259754ff)
    - [16.9.2 代码实现](#X1fff62bd56f6dbc8292732c52e33ec9cbf06ef9)
  + [16.10 评委打分](#X3f4f4df607a295830584f2590527ffd0ede898d)
    - [16.10.1 案例需求](#Xdbe0c4e3ecc82cb0917fa0ad6d88fce5cd1d180)
    - [16.10.2 代码实现](#Xf6fbc0fac70f33b500caa1e97c1a9940deaa4c6)

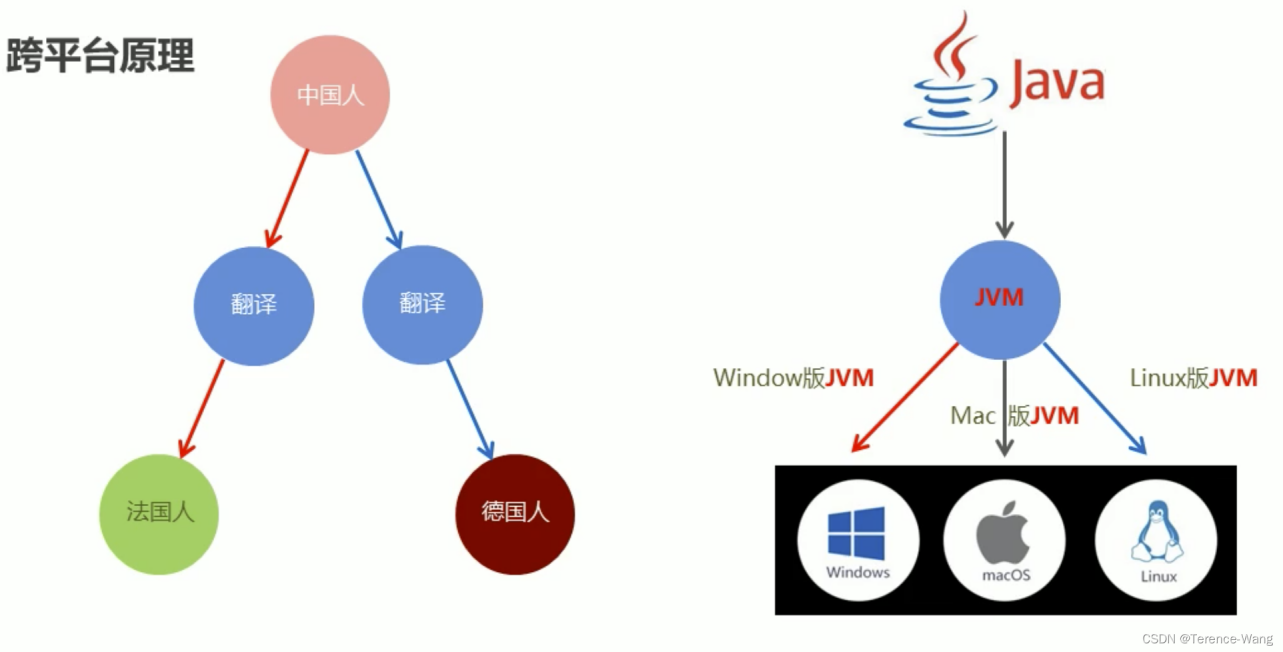
1. Java 概述

### 1.1 Java 语言发展史

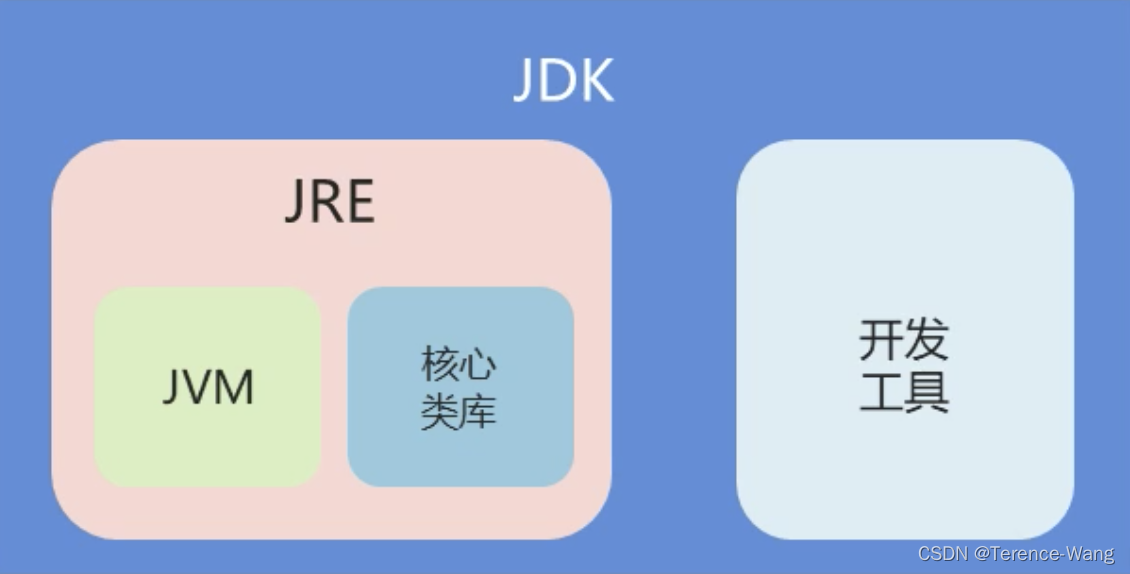
**语言**: 人与人交流沟通的表达方式  
**计算机语言**: 人与计算机之间进行信息交流沟通的一种特殊语言  
Java 语言是美国 **Sun** 公司 (**Stanford University Network**) 在 1995 年推出的计算机语言  
Java 之父: 詹姆斯 · 高斯林 (James Gosling)  
2009 年，Sun 公司被甲骨文公司收购，所以我们现在访问 oracle 官网即可: https://www.oracle.com  
当前，我们课程使用的 JDK 版本: 11.0



### 1.2 Java 语言跨平台原理

Java 程序并非是直接运行的，Java 编译器将 Java 源程序编译成与平台无关的字节码文件 (class 文件)，然后由 Java 虚拟机(JVM) 对字节码文件解释执行。所以在不同的操作系统下，只需安装不同的 Java 虚拟机即可实现 java 程序的跨平台。  


### 1.3 JRE 和 JDK

JVM(Java Virtual Machine)，Java 虚拟机  
JRE(Java Runtime Environment)，Java 运行环境，包含了 JVM 和 Java 的核心类库 (Java API) JDK(Java Development Kit) 称为 Java 开发工具，包含了 JRE 和开发工具  
总结: 我们只需安装 JDK 即可，它包含了 java 的运行环境和虚拟机。  


### 1.4 JDK 的下载和安装

#### 1.4.1 下载

通过官方网站获取 JDK  
<http://www.oracle.com>  
**注意**: 针对不同的操作系统，需要下载对应版本的 JDK。

#### 1.4.2 安装

傻瓜式安装，下一步即可。但默认的安装路径是在 C:\Program Files 下，为方便统一管理建议修改安装路径，将与开发相关的软件都安装到一个目录下，例如: E:\develop。  
**注意**: 安装路径不要包含中文或者空格等特殊字符 (使用纯英文目录)。

#### 1.4.3 JDK 的安装目录介绍

1. 第一个演示程序

### 2.1 常用 DOS 命令

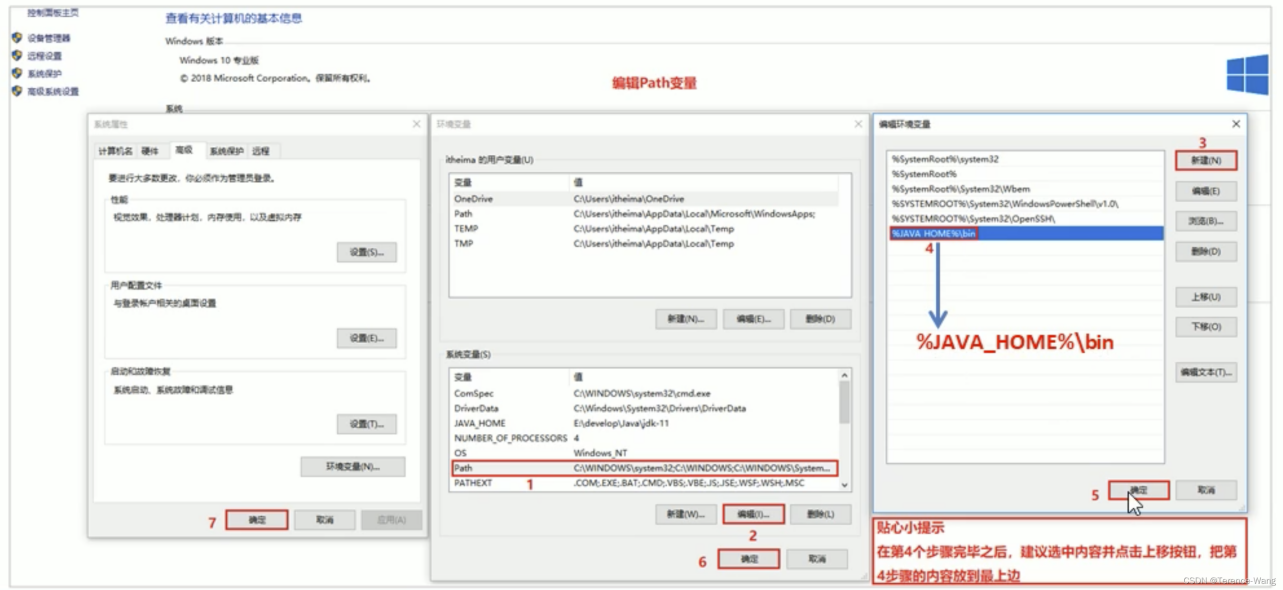
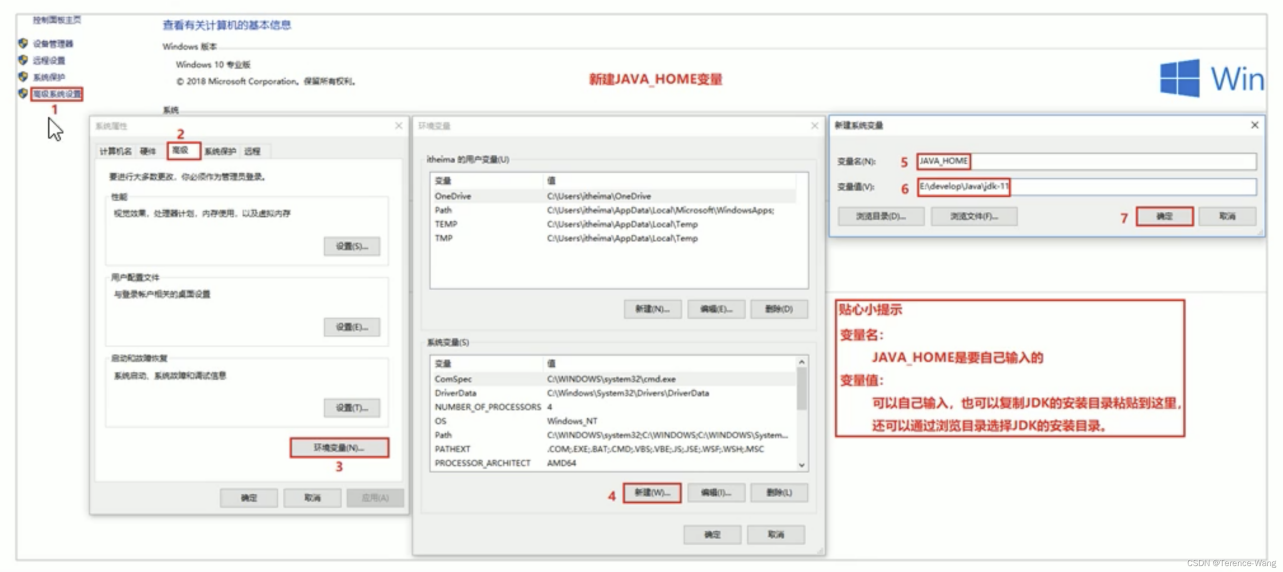
在接触集成开发环境之前，我们需要使用命令行窗口对 java 程序进行编译和运行，所以需要知道一些常用 DOS 命令。  
1、打开命令行窗口的方式: win + r 打开运行窗口，输入 cmd，回车。  
2、常用命令及其作用

### 2.2 Path 环境变量的配置

#### 2.2.1 为什么配置环境变量

开发 Java 程序，需要使用 JDK 提供的开发工具 (比如 javac.exe、java.exe 等命令)，而这些工具在 JDK 的安装目录的 bin 目录下，如果不配置环境变量，那么这些命令只可以在该目录下执行。我们不可能把所有的 java 文件都放到 JDK 的 bin 目录下，所以配置环境变量的作用就是可以使 bin 目录下的 java 相关命令可以在任意目录下使用。

#### 2.2.2 配置环境变量步骤

  
  
**如果命令提示符窗口是配合环境变量前打开的, 需要关闭该窗口, 重新打开一个窗口测试**

### 2.3 HelloWorld 案例

HelloWorld 案例是指在计算机屏幕上输出 “HelloWorld” 这行文字。各种计算机语言都习惯使用该案例作为第一个演示案例。

#### 2.3.1 Java 程序开发运行流程

开发 Java 程序，需要三个步骤: 编写程序，编译程序，运行程序。

#### 2.3.2 HelloWorld 案例的编写

1、新建文本文档文件，修改名称为 HelloWorld.java。  
2、用记事本打开 HelloWorld.java 文件，输写程序内容。

public class HelloWorld {  
 public static void main(String[] args) {  
 System.out.println("HelloWorld");  
 }  
}

#### 2.3.3 HelloWorld 案例的编译和运行

存文件，打开命令行窗口，将目录切换至 java 文件所在目录，编译 java 文件生成 class 文件，运行 class 文件。

编译: javac 文件名. java  
范例: javac HelloWorld.java  
执行: java 类名  
范例: java HelloWorld

### 2.4 HelloWorld 案例常见问题 (理解)

#### 2.4.1 BUG

在电脑系统或程序中，隐藏着的一些未被发现的缺陷或问题统称为 bug(漏洞)。

#### 2.4.2 BUG 的解决

1、具备识别 BUG 的能力: 多看  
2、具备分析 BUG 的能力: 多思考，多查资料  
3、具备解决 BUG 的能力: 多尝试，多总结

#### 2.4.3 HelloWorld 案例常见问题

1、非法字符问题。Java 中的符号都是英文格式的。  
2、大小写问题。Java 语言对大小写敏感 (区分大小写)。  
3、在系统中显示文件的扩展名，避免出现 HelloWorld.java.txt 文件。  
4、编译命令后的 java 文件名需要带文件后缀. java  
5、运行命令后的 class 文件名 (类名) 不带文件后缀. class …

### 2.5 Notepad++ 软件的安装和使用

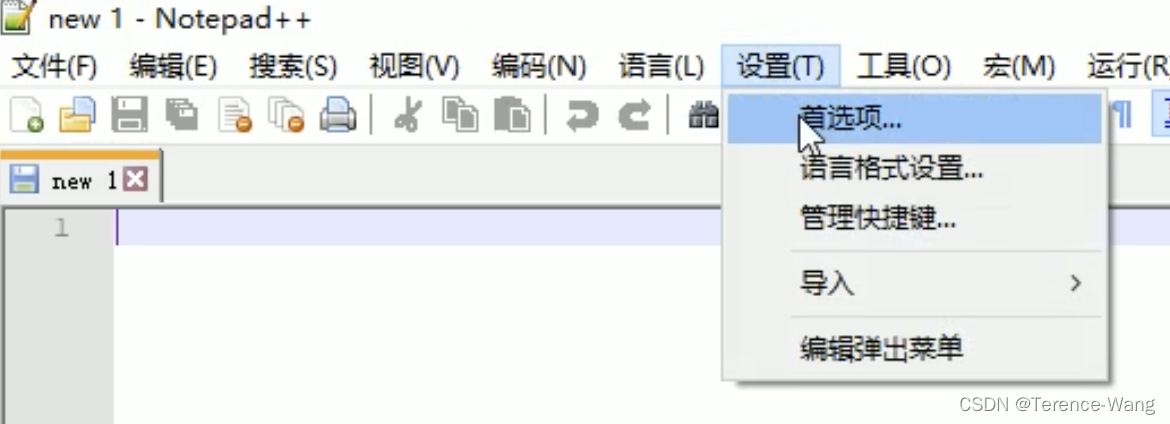
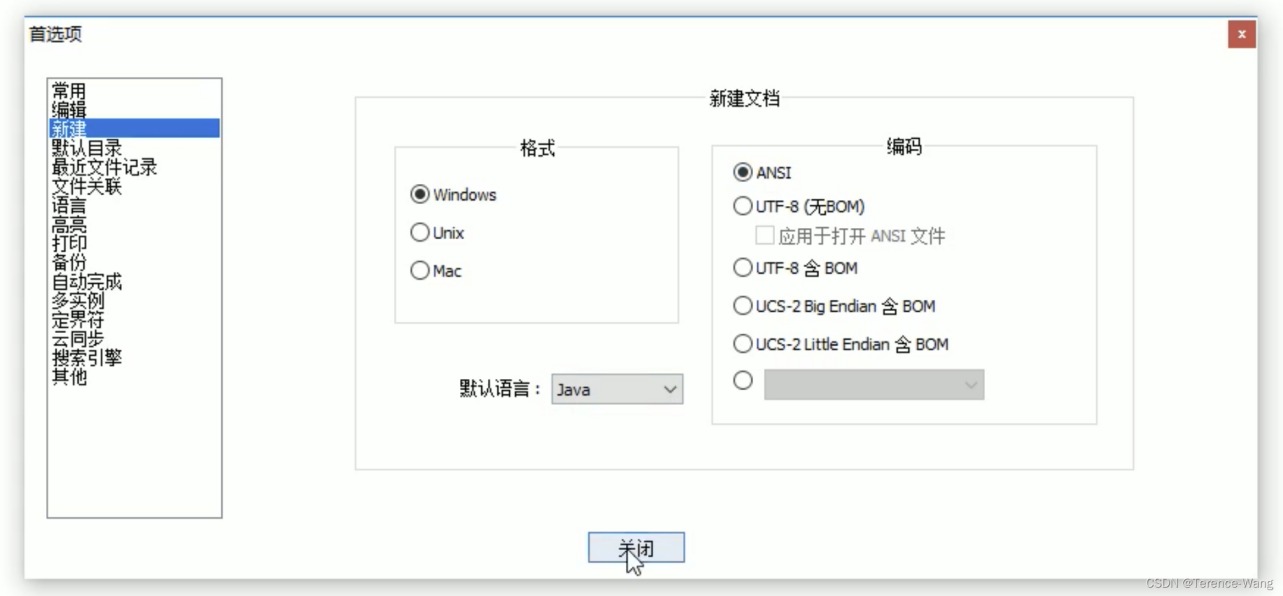
#### 2.5.1 什么要使用 Notepad++ 软件

Notepad++ 功能比 windows 中的自带记事本功能强大，除了可以用来制作一般的纯文字说明文件，也十分适合编写计算机程序代码。Notepad++ 有行号，能够快速定位问题位置，还有语法高亮度显示、代码折叠等功能。而且它是免费的。

#### 2.5.2 Notepad++ 软件安装

安装: 傻瓜式安装，一直下一步即可。建议也安装到统一的开发软件目录下，比如 E:\develop。

#### 2.5.3Notepad++ 软件配置

安装完毕之后，为了使用方便，做一个简单的配置: 修改默认语言和编码。  
  


1. java 基础语法

### 3.1 注释

注释是对代码的解释和说明文字，可以提高程序的可读性，因此在程序中添加必要的注释文字十分重要。Java 中的 注释分为三种:  
单行注释。单行注释的格式是使用 //，从 // 开始至本行结尾的文字将作为注释文字。

// 这是单行注释文字

多行注释。多行注释的格式是使用 /\* 和 \*/ 将一段较长的注释括起来。

/\*  
 这是多行注释文字 这是多行注释文字 这是多行注释文字  
\*/

**注意**: 多行注释不能嵌套使用。

文档注释。文档注释以 /\*\* 开始，以 \*/ 结束。(以后讲)

### 3.2 关键字

关键字是指被 java 语言赋予了特殊含义的单词。  
关键字的特点: 关键字的字母全部小写。  
常用的代码编辑器对关键字都有高亮显示，比如现在我们能看到的 public、class、static 等。

### 3.3 常量

常量: 在程序运行过程中，其值不可以发生改变的量。  
Java 中的常量分类:  
字符串常量 用双引号括起来的多个字符 (可以包含 0 个、一个或多个)，例如 "a"、“abc”、"中国" 等  
整数常量 整数，例如:-10、0、88 等  
小数常量 小数，例如:-5.5、1.0、88.88 等  
字符常量 用单引号括起来的一个字符，例如:‘a’、‘5’、‘B’、'中’等  
布尔常量 布尔值，表示真假，只有两个值 true 和 false  
空常量 一个特殊的值，空值，值为 null  
除空常量外，其他常量均可使用输出语句直接输出。

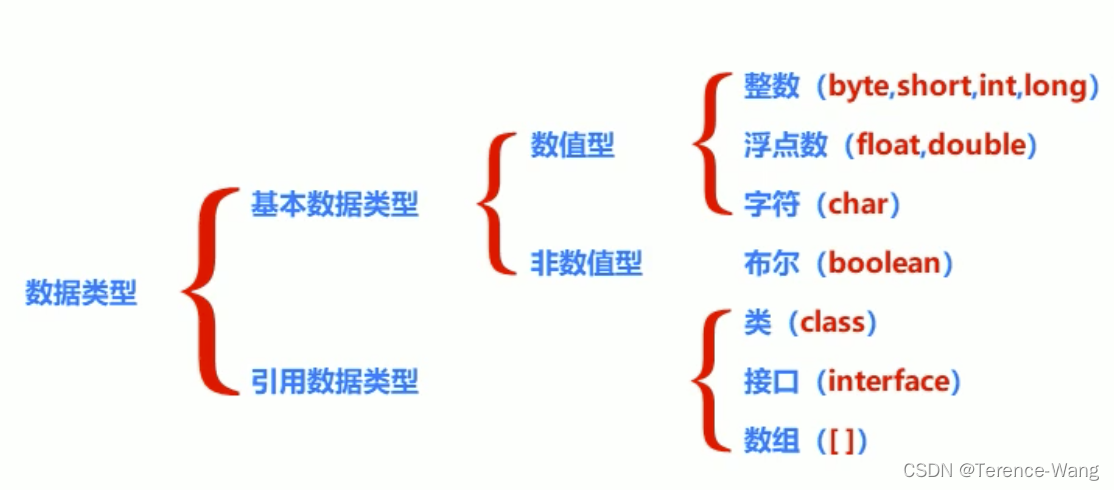
public class Demo {  
 public static void main(String[] args) {  
 System.out.println(10); // 输出一个整数  
 System.out.println(5.5); // 输出一个小数  
 System.out.println('a'); // 输出一个字符  
 System.out.println(true); // 输出boolean值true  
 System.out.println("欢迎来到黑马程序员"); // 输出字符串  
 }  
}

### 3.4 数据类型

#### 3.4.1 计算机存储单元

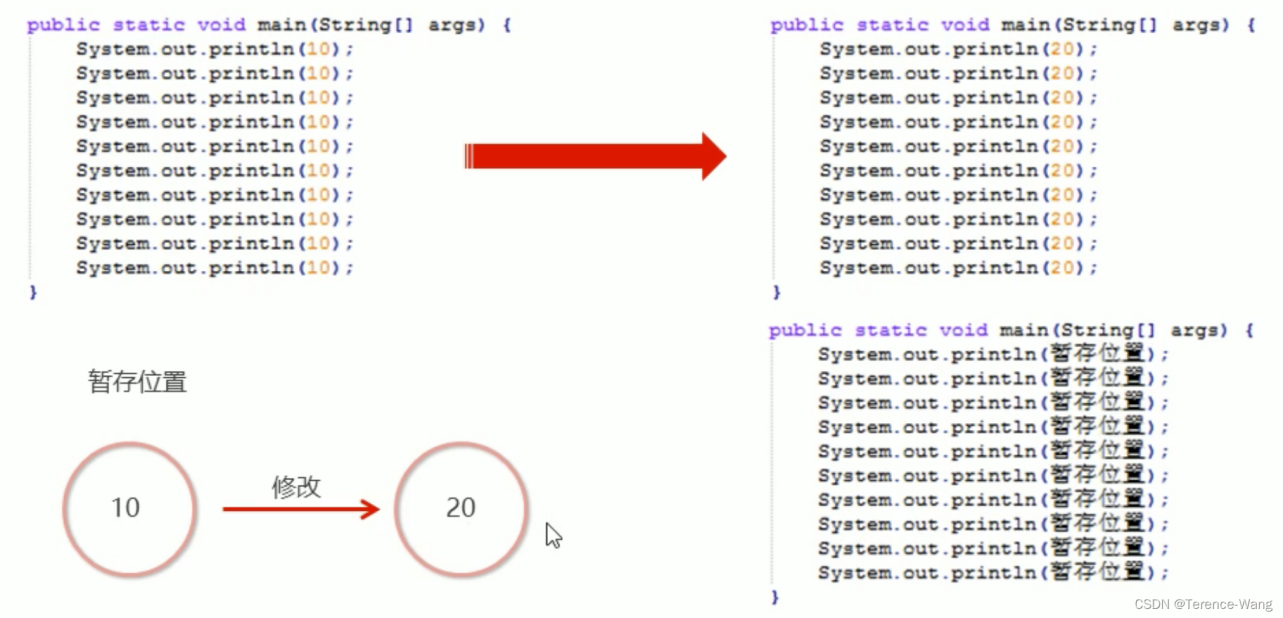
我们知道计算机是可以用来存储数据的，但是无论是内存还是硬盘，计算机存储设备的最小信息单元叫 “位 (bit)”，我们又称之为“比特位”，通常用小写的字母”b” 表示。而计算机中最基本的存储单元叫“字节(byte)”，  
通常用大写字母”B” 表示，字节是由连续的 8 个位组成。  
除了字节外还有一些常用的存储单位，其换算单位如下:  
1B(字节) = 8bit  
1KB = 1024B  
1MB = 1024KB  
1GB = 1024MB  
1TB = 1024GB

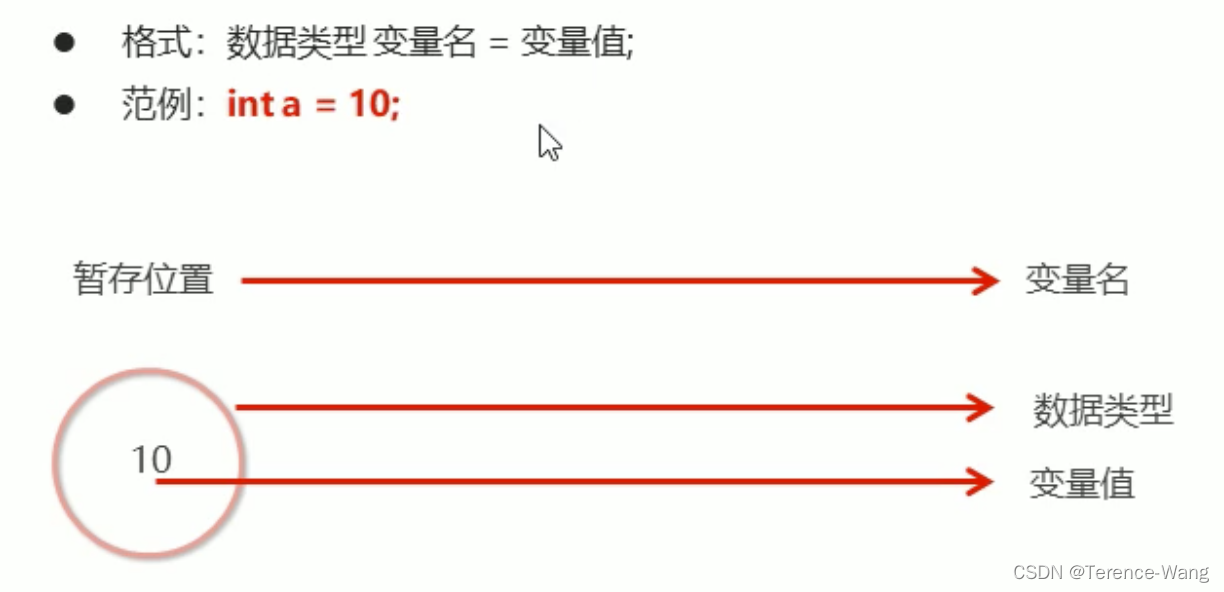
#### 3.4.2 Java 中的数据类型

Java 是一个强类型语言，Java 中的数据必须明确数据类型。在 Java 中的数据类型包括基本数据类型和引用数据类型两种。  
  
Java 中的基本数据类型:

### 3.5 变量

#### 3.5.1 变量的定义

变量: 在程序运行过程中，其值可以发生改变的量。  
从本质上讲，变量是内存中的一小块区域，其值可以在一定范围内变化。  


变量的定义格式:  


数据类型 变量名 = 初始化值; // 声明变量并赋值  
int age = 18;  
System.out.println(age);

或者

// 先声明，后赋值 (使用前赋值即可) 数据类型 变量名;  
变量名 = 初始化值;  
double money;  
money = 55.5;  
System.out.println(money);

还可以在同一行定义多个同一种数据类型的变量，中间使用逗号隔开。但不建议使用这种方式，降低程序的可读性。

int a = 10, b = 20; // 定义int类型的变量a和b，中间使用逗号隔开  
System.out.println(a);  
System.out.println(b);  
int c,d; // 声明int类型的变量c和d，中间使用逗号隔开  
c = 30;  
d = 40;  
System.out.println(c);  
System.out.println(d);

变量的使用: 通过变量名访问即可。

#### 3.5.2 使用变量时的注意事项

1. 在同一对花括号中，变量名不能重复。
2. 变量在使用之前，必须初始化 (赋值)。
3. 定义 long 类型的变量时，需要在整数的后面加 L(大小写均可，建议大写)。因为整数默认是 int 类型，整数太大可能超出 int 范围。
4. 定义 float 类型的变量时，需要在小数的后面加 F(大小写均可，建议大写)。因为浮点数的默认类型是 double， double 的取值范围是大于 float 的，类型不兼容。

#### 3.6 标识符

标识符是用户编程时使用的名字，用于给类、方法、变量、常量等命名。  
Java 中标识符的组成规则:  
由字母、数字、下划线 “\_”、美元符号“$” 组成，第一个字符不能是数字。  
不能使用 java 中的关键字作为标识符。  
标识符对大小写敏感 (区分大小写)。  
Java 中标识符的命名约定:  
小驼峰式命名: 变量名、方法名  
首字母小写，从第二个单词开始每个单词的首字母大写。  
大驼峰式命名: 类名  
每个单词的首字母都大写。  
另外，标识符的命名最好可以做到见名知意  
例如: username、studentNumber 等。

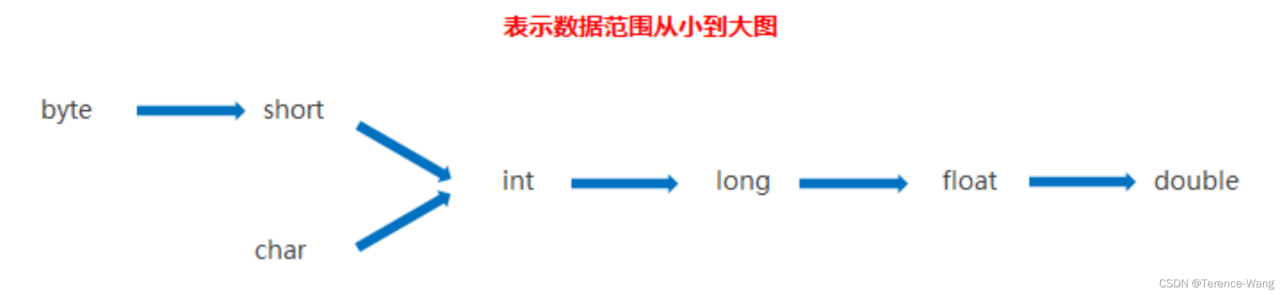
#### 3.7 类型转换

在 Java 中，一些数据类型之间是可以相互转换的。分为两种情况: 自动类型转换和强制类型转换。 自动类型转换:  
把一个表示数据范围小的数值或者变量赋值给另一个表示数据范围大的变量。这种转换方式是自动的，直接书写即可。例如:

double num = 10; // 将int类型的10直接赋值给double类型  
 System.out.println(num); // 输出10.0

强制类型转换:  
把一个表示数据范围大的数值或者变量赋值给另一个表示数据范围小的变量。  
强制类型转换格式: 目标数据类型 变量名 = (目标数据类型) 值或者变量;  
例如:

double num1 = 5.5;  
int num2 = (int) num1; // 将double类型的num1强制转换为int类型  
System.out.println(num2); // 输出5(小数位直接舍弃)

  
说明:

1. char 类型的数据转换为 int 类型是按照码表中对应的 int 值进行计算的。比如在 ASCII 码表中，'a’对应 97。

int a = 'a';  
 System.out.println(a); // 将输出97

1. 整数默认是 int 类型，byte、short 和 char 类型数据参与运算均会自动转换为 int 类型。

byte b1 = 10;  
byte b2 = 20;  
byte b3 = b1 + b2;  
// 第三行代码会报错，b1和b2会自动转换为int类型，计算结果为int，int赋值给byte需要强制类型转换。  
// 修改为:  
int num = b1 + b2;  
// 或者:  
byte b3 = (byte) (b1 + b2);

1. boolean 类型不能与其他基本数据类型相互转换。
2. 运算符

### 4.1 算术运算符

#### 4.1.1 运算符和表达式

运算符: 对常量或者变量进行操作的符号  
表达式: 用运算符把常量或者变量连接起来符合 java 语法的式子就可以称为表达式。  
不同运算符连接的表达式体现的是不同类型的表达式。  
举例说明:

int a = 10;  
int b = 20;  
int c = a + b;

+: 是运算符，并且是算术运算符。  
a + b: 是表达式，由于 + 是算术运算符，所以这个表达式叫算术表达式。

#### 4.1.2 算术运算符

注意: / 和 % 的区别: 两个数据做除法，/ 取结果的商，% 取结果的余数。  
整数操作只能得到整数，要想得到小数，必须有浮点数参与运算。

int a = 10;  
int b = 3;  
System.out.println(a / b); // 输出结果3  
System.out.println(a % b); // 输出结果1

#### 4.1.3 字符的 “+” 操作

char 类型参与算术运算，使用的是计算机底层对应的十进制数值。需要我们记住三个字符对应的数值:  
‘a’ – 97 a-z 是连续的，所以’b’对应的数值是 98，'c’是 99，依次递加  
‘A’ – 65 A-Z 是连续的，所以’B’对应的数值是 66，'C’是 67，依次递加  
‘0’ – 48 0-9 是连续的，所以’1’对应的数值是 49，'2’是 50，依次递加

// 可以通过使用字符与整数做算术运算，得出字符对应的数值是多少  
char ch1 = 'a';  
System.out.println(ch1 + 1); // 输出98，97 + 1 = 98  
  
char ch2 = 'A';  
System.out.println(ch2 + 1); // 输出66，65 + 1 = 66  
  
char ch3 = '0';  
System.out.println(ch3 + 1); // 输出49，48 + 1 = 49

算术表达式中包含不同的基本数据类型的值的时候，整个算术表达式的类型会自动进行提升。  
提升规则:  
byte 类型，short 类型和 char 类型将被提升到 int 类型，不管是否有其他类型参与运算。  
整个表达式的类型自动提升到与表达式中最高等级的操作数相同的类型  
等级顺序: byte,short,char --> int --> long --> float --> double  
例如:

byte b1 = 10;  
byte b2 = 20;  
// byte b3 = b1 + b2; // 该行报错，因为byte类型参与算术运算会自动提示为int，int赋值给byte可能损失精度  
int i3 = b1 + b2; // 应该使用int接收  
byte b3 = (byte) (b1 + b2); // 或者将结果强制转换为byte类型  
// -------------------------------  
int num1 = 10;  
double num2 = 20.0;  
double num3 = num1 + num2; // 使用double接收，因为num1会自动提升为double类型

tips: 正是由于上述原因，所以在程序开发中我们很少使用 byte 或者 short 类型定义整数。也很少会使用 char 类型定义字符，而使用字符串类型，更不会使用 char 类型做算术运算。

#### 4.1.4 字符串的 “+” 操作

当 “+” 操作中出现字符串时，这个”+”是字符串连接符，而不是算术运算。

System.out.println("itheima"+ 666); // 输出:itheima666

在”+”操作中，如果出现了字符串，就是连接运算符，否则就是算术运算。当连续进行 “+” 操作时，从左到右逐个执行。

System.out.println(1 + 99 + "年黑马"); // 输出:100年黑马  
System.out.println(1 + 2 + "itheima" + 3 + 4); // 输出:3itheima34  
// 可以使用小括号改变运算的优先级  
System.out.println(1 + 2 + "itheima" + (3 + 4)); // 输出:3itheima7

### 4.2 赋值运算符 (应用)

赋值运算符的作用是将一个表达式的值赋给左边，左边必须是可修改的，不能是常量。

short s = 10;  
s = s + 10; // 此行代码报出，因为运算中s提升为int类型，运算结果int赋值给short可能损失精度  
s += 10; // 此行代码没有问题，隐含了强制类型转换，相当于 s = (short) (s + 10);

### 4.3 自增自减运算符

注意事项:  
++ 和 -- 既可以放在变量的后边，也可以放在变量的前边。  
单独使用的时候， ++ 和 -- 无论是放在变量的前边还是后边，结果是一样的。  
参与操作的时候，如果放在变量的后边，先拿变量参与操作，后拿变量做 ++ 或者–。  
参与操作的时候，如果放在变量的前边，先拿变量做 ++ 或者–，后拿变量参与操作。  
最常见的用法: 单独使用。

int i = 10;  
i++; // 单独使用 System.out.println("i:" + i); // i:11  
int j = 10;  
++j; // 单独使用 System.out.println("j:" + j); // j:11  
int x = 10;  
int y = x++; // 赋值运算，++在后边，所以是使用x原来的值赋值给y，x本身自增1  
System.out.println("x:" + x + ", y:" + y); // x:11，y:10  
int m = 10;  
int n = ++m; // 赋值运算，++在前边，所以是使用m自增后的值赋值给n，m本身自增1  
System.out.println("m:" + m + ", m:" + m); // m:11，m:11

练习:

int x = 10;  
int y = x++ + x++ + x++;  
System.out.println(y); // y的值是多少?  
/\* 解析，三个表达式都是++在后，所以每次使用的都是自增前的值，  
但程序自左至右执行，所以第一次自增时，使用的是 10进行计算，但第二次自增时，x的值已经自增到11了，  
所以第二次使用的是11，然后再次自增。。。 所以整个式子应该是:int y = 10 + 11 + 12;  
输出结果为33。  
\*/

注意: 通过此练习深刻理解自增和自减的规律，但实际开发中强烈建议不要写这样的代码! 小心挨打!

### 4.4 关系运算符

关系运算符有 6 种关系，分别为小于、小于等于、大于、等于、大于等于、不等于。

int a = 10;  
int b = 20;  
System.out.println(a == b); // false  
System.out.println(a != b); // true  
System.out.println(a > b); // false  
System.out.println(a >= b); // false  
System.out.println(a < b); // true  
System.out.println(a <= b); // true  
// 关系运算的结果肯定是boolean类型，所以也可以将运算结果赋值给boolean类型的变量 boolean flag = a > b;  
System.out.println(flag); // 输出false

### 4.5 逻辑运算符

逻辑运算符把各个运算的关系表达式连接起来组成一个复杂的逻辑表达式，以判断程序中的表达式是否成立，判断的结果是 true 或 false。

//定义变量  
int i = 10;   
int j = 20;   
int k = 30;  
//& “与”，并且的关系，只要表达式中有一个值为false，结果即为false  
System.out.println((i > j) & (i> k)); //false & false,输出false   
System.out.println((i < j) & (i> k)); //true & false,输出false false   
System.out.println((i > j) & (i< k)); //false & true,输出  
System.out.println((i < j) & (i< k)); //true & true,输出true  
System.out.println("--------");  
  
//| “或”，或者的关系，只要表达式中有一个值为true，结果即为true  
System.out.println((i > j) | (i> k)); //false | false,输出false   
System.out.println((i < j) | (i> k)); //true | false,输出true  
System.out.println((i > j) | (i< k)); //false | true,输出true  
System.out.println((i < j) | (i< k)); //true | true,输出true  
System.out.println("--------");  
  
  
//^ “异或”，相同为false，不同为true  
System.out.println((i > j) ^ (i > k)); //false ^ false,输出false  
System.out.println((i < j) ^ (i > k)); //true ^ false,输出true  
System.out.println((i > j) ^ (i < k)); //false ^ true,输出true  
System.out.println((i < j) ^ (i < k)); //true ^ true,输出false  
System.out.println("--------");  
//! “非”，取反  
System.out.println((i > j)); //false  
System.out.println(!(i > j)); //!false,输出true

短路逻辑运算符

* 逻辑与 &，无论左边真假，右边都要执行。
* 短路与 &&，如果左边为真，右边执行; 如果左边为假，右边不执行。
* 逻辑或 |，无论左边真假，右边都要执行。
* 短路或 ||，如果左边为假，右边执行; 如果左边为真，右边不执行。

int x = 3;  
int y = 4;  
System.out.println((x++ > 4) & (y++ > 5)); // 两个表达都会运算  
System.out.println(x); // 4  
System.out.println(y); // 5  
System.out.println((x++ > 4) && (y++ > 5)); // 左边已经可以确定结果为false，右边不参与运算 System.out.println(x); // 4  
System.out.println(y); // 4

### 4.6 三元运算符

三元运算符语法格式:

关系表达式 ? 表达式 1 : 表达式 2;

解释: 问号前面的位置是判断的条件，判断结果为 boolean 型，为 true 时调用表达式 1，为 false 时调用表达式 2。其逻辑为: 如果条件表达式成立或者满足则执行表达式 1，否则执行第二个。  
举例:

int a = 10;  
int b = 20;  
int c = a > b ? a : b; // 判断 a>b 是否为真，如果为真取a的值，如果为假，取b的值

三元运算符案例:  
1、需求: 动物园里有两只老虎，已知两只老虎的体重分别为 180kg、200kg，请用程序实现判断两只老虎的体重是否相同。

public class OperatorTest01 {  
 public static void main(String[] args) {  
 //1：定义两个变量用于保存老虎的体重，单位为kg，这里仅仅体现数值即可。  
 int weight1 = 180;  
 int weight2 = 200;  
  
 //2：用三元运算符实现老虎体重的判断，体重相同，返回true，否则，返回false。  
 boolean b = weight1 == weight2 ? true : false;  
  
 //3：输出结果  
 System.out.println("b:" + b);  
 }  
}

2、需求: 一座寺庙里住着三个和尚，已知他们的身高分别为 150cm、210cm、165cm，请用程序实现获取这三个和尚的最高身高。

public class OperatorTest02 {  
 public static void main(String[] args) {  
 //1：定义三个变量用于保存和尚的身高，单位为cm，这里仅仅体现数值即可。  
 int height1 = 150;  
 int height2 = 210;  
 int height3 = 165;  
  
 //2：用三元运算符获取前两个和尚的较高身高值，并用临时身高变量保存起来。  
 int tempHeight = height1 > height2 ? height1 : height2;  
  
 //3：用三元运算符获取临时身高值和第三个和尚身高较高值，并用最大身高变量保存。  
 int maxHeight = tempHeight > height3 ? tempHeight : height3;  
  
 //4：输出结果  
 System.out.println("maxHeight:" + maxHeight);  
 }  
}

1. 数据输入

我们可以通过 Scanner 类来获取用户的输入。使用步骤如下:  
1、导包。Scanner 类在 java.util 包下，所以需要将该类导入。导包的语句需要定义在类的上面。

import java.util.Scanner;

2、创建 Scanner 对象。

Scanner sc = new Scanner(System.in);// 创建 Scanner 对象，sc 表示变量名，其他均不可变

3、接收数据

int i = sc.nextInt(); // 表示将键盘录入的值作为 int 数返回。

示例:

import java.util.Scanner;  
  
public class ScannerDemo {  
 public static void main(String[] args) { //创建对象  
 Scanner sc = new Scanner(System.in); //接收数据  
 int x = sc.nextInt();  
 //输出数据  
 System.out.println("x:" + x);  
 }  
}

改写三个和尚案例，数据使用键盘录入。

import java.util.Scanner;  
  
public class ScannerTest {  
 public static void main(String[] args) {  
 //身高未知，采用键盘录入实现。首先导包，然后创建对象。  
 Scanner sc = new Scanner(System.in);  
  
 //键盘录入三个身高分别赋值给三个变量。  
 System.out.println("请输入第一个和尚的身高：");  
 int height1 = sc.nextInt();  
  
 System.out.println("请输入第二个和尚的身高：");  
 int height2 = sc.nextInt();  
  
 System.out.println("请输入第三个和尚的身高：");  
 int height3 = sc.nextInt();  
  
 //用三元运算符获取前两个和尚的较高身高值，并用临时身高变量保存起来。  
 int tempHeight = height1 > height2 ? height1 : height2;  
  
 //用三元运算符获取临时身高值和第三个和尚身高较高值，并用最大身高变量保存。  
 int maxHeight = tempHeight > height3 ? tempHeight : height3;  
  
 //输出结果。  
 System.out.println("这三个和尚中身高最高的是：" + maxHeight +"cm");  
 }  
}

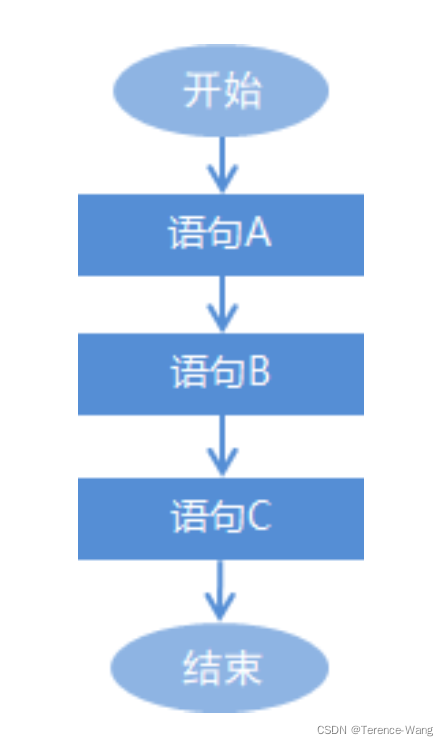
1. 流程控制语句

在一个程序执行的过程中，各条语句的执行顺序对程序的结果是有直接影响的。所以，我们必须清楚每条语句的执行流程。而且，很多时候要通过控制语句的执行顺序来实现我们想要的功能。

### 6.1 流程控制语句分类

顺序结构  
分支结构 (if, switch)  
循环结构 (for, while, do…while)

### 6.2 顺序结构

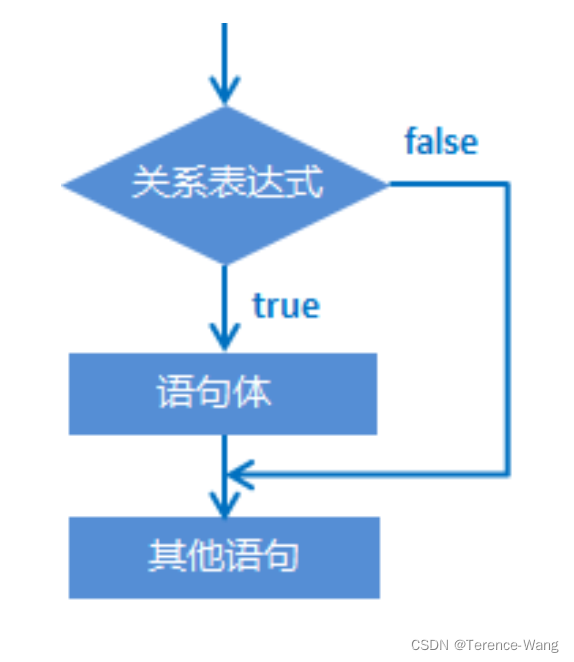
顺序结构是程序中最简单最基本的流程控制，没有特定的语法结构，按照代码的先后顺序，依次执行，程序中大多数的代码都是这样执行的。  
顺序结构执行流程图:  


### 6.3 分支结构之 if 语句

#### 6.3.1 if 语句格式 1

格式:  
if (关系表达式) {  
语句体;  
}

执行流程:  
1 首先计算关系表达式的值  
2 如果关系表达式的值为 true 就执行语句体  
3 如果关系表达式的值为 false 就不执行语句体  
4 继续执行后面的语句内容



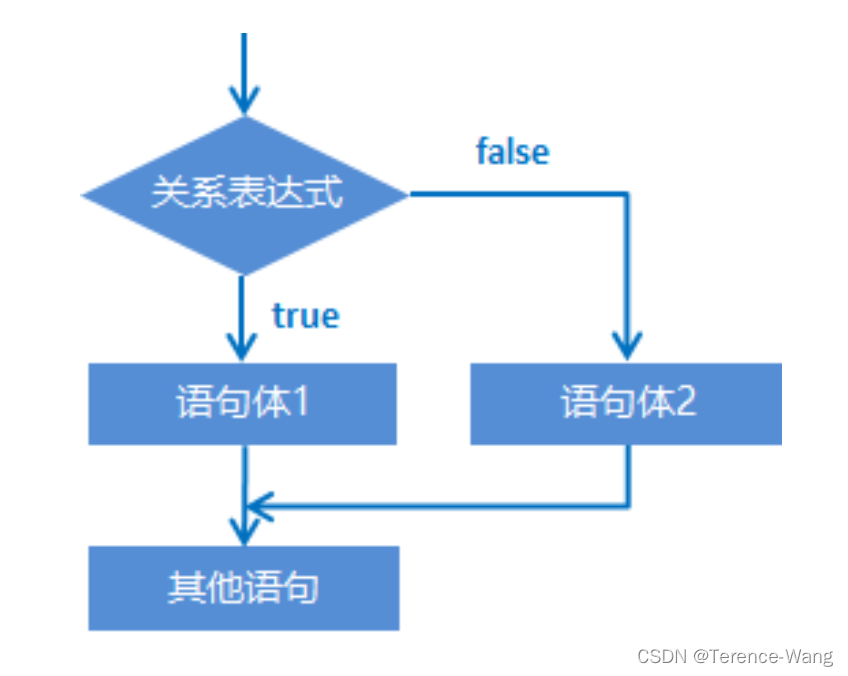
示例:

public class IfDemo {  
 public static void main(String[] args) {  
 System.out.println("开始");  
  
 //定义两个变量  
 int a = 10;  
 int b = 20;  
  
 //需求：判断a和b的值是否相等，如果相等，就在控制台输出：a等于b  
 if(a == b) {  
 System.out.println("a等于b");  
 }  
  
 //需求：判断a和c的值是否相等，如果相等，就在控制台输出：a等于c  
 int c = 10;  
 if(a == c) {  
 System.out.println("a等于c");  
 }  
  
 System.out.println("结束");  
 }  
}

#### 6.3.2 if 语句格式 2

格式:  
if (关系表达式) {  
语句体 1;  
} else {  
语句体 2;  
}

执行流程:

1. 首先计算关系表达式的值
2. 如果关系表达式的值为 true 就执行语句体 1
3. 如果关系表达式的值为 false 就执行语句体 2
4. 继续执行后面的语句内容  
     
   示例:

public class IfDemo02 {  
 public static void main(String[] args) {  
 System.out.println("开始");  
  
 //定义两个变量  
 int a = 10;  
 int b = 20;  
 b = 5;  
  
 //需求：判断a是否大于b，如果是，在控制台输出：a的值大于b，  
 //如果不是，在控制台输出：a的值不大于b  
 if(a > b) {  
 System.out.println("a的值大于b");  
 } else {  
 System.out.println("a的值不大于b");  
 }  
  
 System.out.println("结束");  
 }  
}

if 语句案例: 奇偶数  
需求: 任意给出一个整数，请用程序实现判断该整数是奇数还是偶数，并在控制台输出该整数是奇数还是偶数。  
分析:

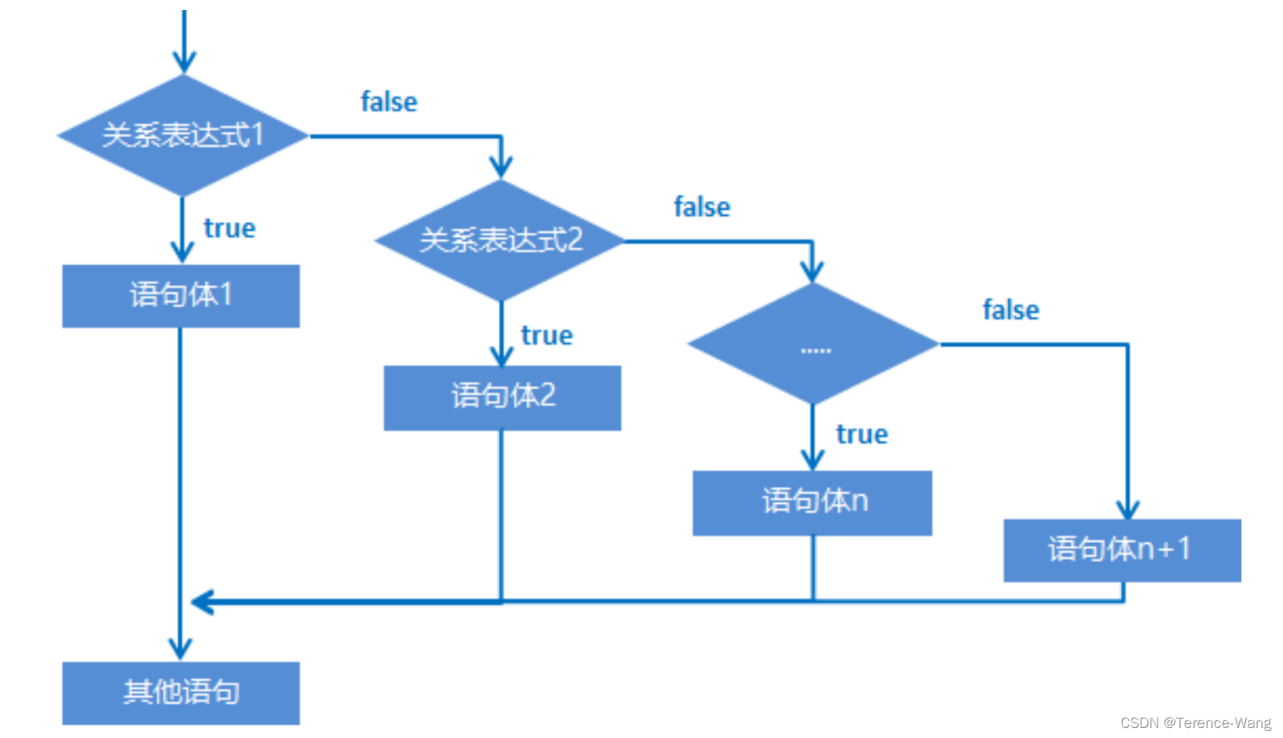
1. 为了体现任意给出一个整数，采用键盘录入一个数据
2. 判断整数是偶数还是奇数要分两种情况进行判断，使用 if…else 结构
3. 判断是否偶数需要使用取余运算符实现该功能 number % 2 == 0
4. 根据判定情况，在控制台输出对应的内容

import java.util.Scanner;  
  
public class IfTest01 {  
 public static void main(String[] args) {  
 //为了体现任意给出一个整数，采用键盘录入一个数据。(导包，创建对象，接收数据)  
 Scanner sc = new Scanner(System.in);  
  
 System.out.println("请输入一个整数：");  
 int number = sc.nextInt();  
  
 //判断整数是偶数还是奇数要分两种情况进行判断，使用if..else结构  
 //判断是否偶数需要使用取余运算符实现该功能 number % 2 == 0  
 //根据判定情况，在控制台输出对应的内容  
 if(number%2 == 0) {  
 System.out.println(number + "是偶数");  
 } else {  
 System.out.println(number + "是奇数");  
 }  
  
 }  
}

#### 6.3.3 if 语句格式 3

格式:  
if (关系表达式 1) {  
语句体 1;  
} else if (关系表达式 2) {  
语句体 2;  
}  
…  
else {  
语句体 n+1;  
}

执行流程:

1. 首先计算关系表达式 1 的值
2. 如果值为 true 就执行语句体 1; 如果值为 false 就计算关系表达式 2 的值
3. 如果值为 true 就执行语句体 2; 如果值为 false 就计算关系表达式 3 的值
4. …
5. 如果没有任何关系表达式为 true，就执行语句体 n+1。  
   

示例:  
键盘录入一个星期数 (1,2,…7)，输出对应的星期一，星期二，… 星期日

import java.util.Scanner;  
  
public class IfDemo03 {  
 public static void main(String[] args) {  
 System.out.println("开始");  
  
 //需求：键盘录入一个星期数(1,2,...7)，输出对应的星期一，星期二，...星期日  
 Scanner sc = new Scanner(System.in);  
  
 System.out.println("请输入一个星期数(1-7)：");  
 int week = sc.nextInt();  
  
 if(week == 1) {  
 System.out.println("星期一");  
 } else if(week == 2) {  
 System.out.println("星期二");  
 } else if(week == 3) {  
 System.out.println("星期三");  
 } else if(week == 4) {  
 System.out.println("星期四");  
 } else if(week == 5) {  
 System.out.println("星期五");  
 } else if(week == 6) {  
 System.out.println("星期六");  
 } else {  
 System.out.println("星期日");  
 }  
  
 System.out.println("结束");  
 }  
}

if 语句格式 3 案例:  
需求: 小明快要期末考试了，小明爸爸对他说，会根据他不同的考试成绩，送他不同的礼物，假如你可以控制小明的得分，请用程序实现小明到底该获得什么样的礼物，并在控制台输出。  
分析:

1. 小明的考试成绩未知，可以使用键盘录入的方式获取值
2. 由于奖励种类较多，属于多种判断，采用 if…else…if 格式实现
3. 为每种判断设置对应的条件
4. 为每种判断设置对应的奖励

import java.util.Scanner;  
  
public class IfTest02 {  
 public static void main(String[] args) {  
 //小明的考试成绩未知，可以使用键盘录入的方式获取值  
 Scanner sc = new Scanner(System.in);  
  
 System.out.println("请输入一个分数：");  
 int score = sc.nextInt();  
  
 //由于奖励种类较多，属于多种判断，采用if...else...if格式实现  
 //为每种判断设置对应的条件  
 //为每种判断设置对应的奖励  
  
 //数据测试：正确数据，边界数据，错误数据  
 if(score>100 || score<0) {  
 System.out.println("你输入的分数有误");  
 } else if(score>=95 && score<=100) {  
 System.out.println("山地自行车一辆");  
 } else if(score>=90 && score<=94) {  
 System.out.println("游乐场玩一次");  
 } else if(score>=80 && score<=89) {  
 System.out.println("变形金刚玩具一个");  
 } else {  
 System.out.println("胖揍一顿");  
 }  
 }  
}

1. switch 语句

### 7.1 switch 语句结构

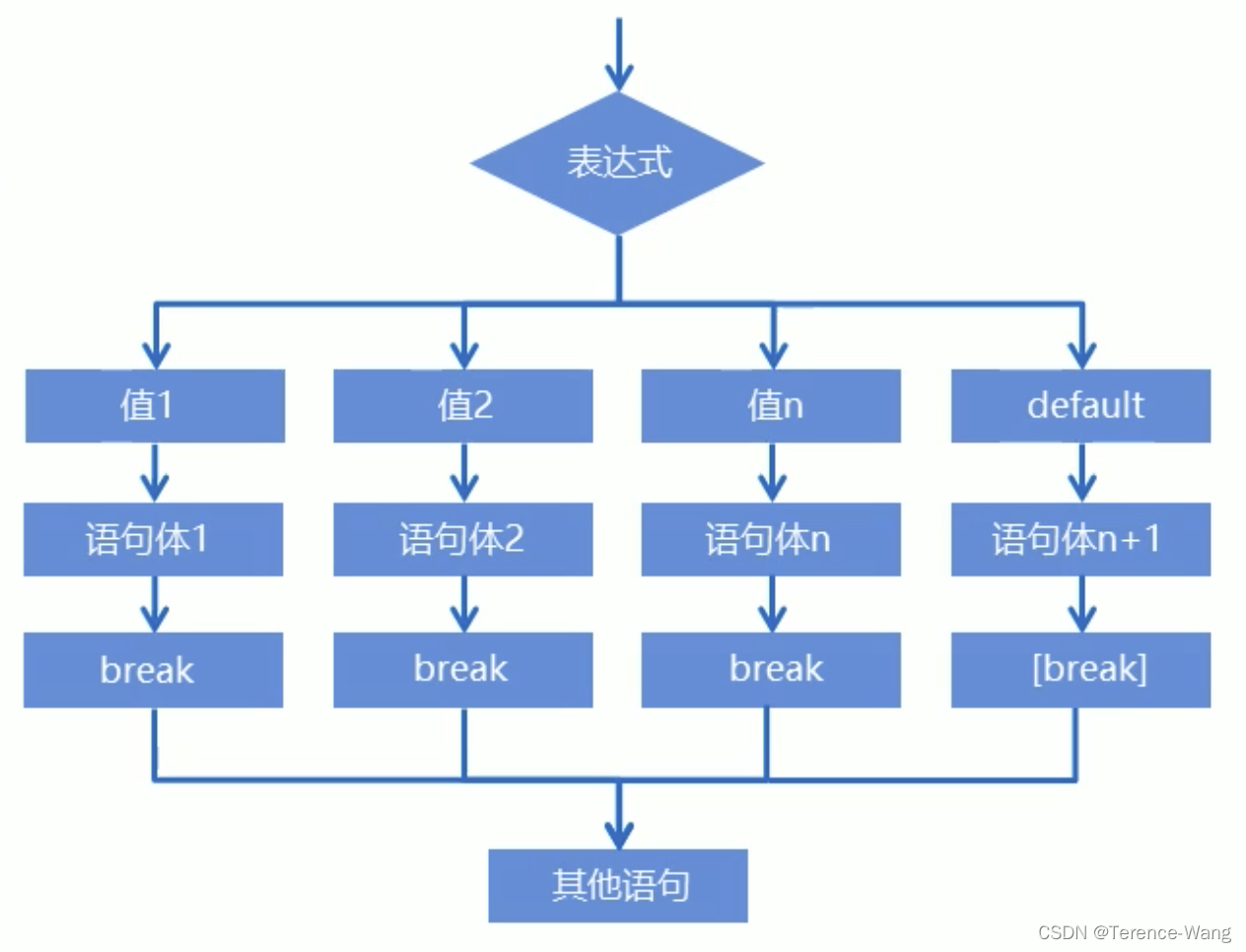
格式

switch (表达式) {  
 case 1:  
 语句体1;  
 break;  
 case 2:  
 语句体2;  
 break;  
 ...  
 default:  
 语句体n+1;  
 break;  
}

格式说明：

* 表达式：取值为 byte、short、int、char，JDK5 以后可以是枚举，JDK7 以后可以是 String。
* case：后面跟的是要和表达式进行比较的值。
* break：表示中断，结束的意思，用来结束 switch 语句。
* default：表示所有情况都不匹配的时候，就执行该处的内容，和 if 语句的 else 相似。

执行流程：

* 首先计算出表达式的值
* 其次，和 case 依次比较，一旦有对应的值，就会执行相应的语句，在执行的过程中，遇到 break 就会结 束。
* 最后，如果所有的 case 都和表达式的值不匹配，就会执行 default 语句体部分，然后程序结束掉。  
  

### 7.2 switch 语句练习 - 春夏秋冬（应用）

* 需求：一年有 12 个月，分属于春夏秋冬 4 个季节，键盘录入一个月份，请用程序实现判断该月份属于哪个季节，并输出。
* 运行结果：

春：3、4、5  
夏：6、7、8  
秋：9、10、11  
冬：1、2、12

* 示例代码：

import java.util.Scanner;  
  
public class SwitchTest {  
 public static void main(String[] args) {  
 //键盘录入月份数据，使用变量接收  
 Scanner sc = new Scanner(System.in);  
  
 System.out.println("请输入一个月份：");  
 int month = sc.nextInt();  
  
 //多情况判断，这里采用switch语句实现  
 //在每种情况中，完成输出对应的季节  
 /\*  
 switch(month) {  
 case 1:  
 System.out.println("冬季");  
 break;  
 case 2:  
 System.out.println("冬季");  
 //break;  
 case 3:  
 System.out.println("春季");  
 break;  
 case 4:  
 System.out.println("春季");  
 break;  
 case 5:  
 System.out.println("春季");  
 break;  
 case 6:  
 System.out.println("夏季");  
 break;  
 case 7:  
 System.out.println("夏季");  
 break;  
 case 8:  
 System.out.println("夏季");  
 break;  
 case 9:  
 System.out.println("秋季");  
 break;  
 case 10:  
 System.out.println("秋季");  
 break;  
 case 11:  
 System.out.println("秋季");  
 break;  
 case 12:  
 System.out.println("冬季");  
 break;  
 default:  
 System.out.println("你输入的月份有误");  
 //break;  
 }  
 \*/  
  
 //case穿透  
 switch(month) {  
 case 1:  
 case 2:  
 case 12:  
 System.out.println("冬季");  
 break;  
 case 3:  
 case 4:  
 case 5:  
 System.out.println("春季");  
 break;  
 case 6:  
 case 7:  
 case 8:  
 System.out.println("夏季");  
 break;  
 case 9:  
 case 10:  
 case 11:  
 System.out.println("秋季");  
 break;  
 default:  
 System.out.println("你输入的月份有误");  
 }  
 }  
}

* 注意：如果 switch 中得 case，没有对应 break 的话，则会出现 case 穿透的现象。

1. for 循环

### 8.1 for 循环结构（掌握）

* 循环：  
  循环语句可以在满足循环条件的情况下，反复执行某一段代码，这段被重复执行的代码被称为循环体语句，当反复 执行这个循环体时，需要在合适的时候把循环判断条件修改为 false，从而结束循环，否则循环将一直执行下去，形 成死循环。
* for 循环格式：

for (初始化语句;条件判断语句;条件控制语句) {  
 循环体语句;  
}

* 格式解释：
  + 初始化语句： 用于表示循环开启时的起始状态，简单说就是循环开始的时候什么样
  + 条件判断语句：用于表示循环反复执行的条件，简单说就是判断循环是否能一直执行下去
  + 循环体语句： 用于表示循环反复执行的内容，简单说就是循环反复执行的事情
  + 条件控制语句：用于表示循环执行中每次变化的内容，简单说就是控制循环是否能执行下去
* 执行流程：
* ①执行初始化语句
* ②执行条件判断语句，看其结果是 true 还是 false
* 如果是 false，循环结束
* 如果是 true，继续执行
* ③执行循环体语句
* ④执行条件控制语句
* ⑤回到②继续

public class ForDemo {  
 public static void main(String[] args) {  
 //需求：在控制台输出5次"HelloWorld"  
 System.out.println("HelloWorld");  
 System.out.println("HelloWorld");  
 System.out.println("HelloWorld");  
 System.out.println("HelloWorld");  
 System.out.println("HelloWorld");  
 System.out.println("--------");  
  
 //用循环改进  
 for(int i=1; i<=5; i++) {  
 System.out.println("HelloWorld");  
 }  
  
 }  
}

### 8.2 for 循环练习 - 输出数据（应用）

* 需求：在控制台输出 1-5 和 5-1 的数据
* 示例代码：

public class ForTest01 {  
 public static void main(String[] args) {  
 //需求：输出数据1-5  
 for(int i=1; i<=5; i++) {  
 System.out.println(i);  
 }  
 System.out.println("--------");  
  
 //需求：输出数据5-1  
 for(int i=5; i>=1; i--) {  
 System.out.println(i);  
 }  
  
 }  
}

### 8.3 for 循环练习 - 求和

* 需求：求 1-5 之间的数据和，并把求和结果在控制台输出
* 示例代码：

public class ForTest02 {  
 public static void main(String[] args) {  
 //求和的最终结果必须保存起来，需要定义一个变量，用于保存求和的结果，初始值为0  
 int sum = 0;  
  
 //从1开始到5结束的数据，使用循环结构完成  
 for(int i=1; i<=5; i++) {  
 //将反复进行的事情写入循环结构内部，此处反复进行的事情是将数据 i 加到用于保存最终求和的变量 sum 中  
 sum += i;  
 /\*  
 sum += i; sum = sum + i;  
 第一次：sum = sum + i = 0 + 1 = 1;  
 第二次：sum = sum + i = 1 + 2 = 3;  
 第三次：sum = sum + i = 3 + 3 = 6;  
 第四次：sum = sum + i = 6 + 4 = 10;  
 第五次：sum = sum + i = 10 + 5 = 15;  
 \*/  
 }  
  
 //当循环执行完毕时，将最终数据打印出来  
 System.out.println("1-5之间的数据和是：" + sum);  
 }  
}

* 本题要点：
  + 今后遇到的需求中，如果带有求和二字，请立即联想到求和变量
  + 求和变量的定义位置，必须在循环外部，如果在循环内部则计算出的数据将是错误的

### 8.4 for 循环练习 - 求偶数和

* 需求：求 1-100 之间的偶数和，并把求和结果在控制台输出 }
* 示例代码：

public class ForTest03 {  
 public static void main(String[] args) {  
 //求和的最终结果必须保存起来，需要定义一个变量，用于保存求和的结果，初始值为0  
 int sum = 0;  
  
 //对1-100的数据求和与1-5的数据求和几乎完全一样，仅仅是结束条件不同  
 for(int i=1; i<=100; i++) {  
 //对1-100的偶数求和，需要对求和操作添加限制条件，判断是否是偶数  
 if(i%2 == 0) {  
 sum += i;  
 }  
 }  
  
 //当循环执行完毕时，将最终数据打印出来  
 System.out.println("1-100之间的偶数和是：" + sum);  
 }  
}

### 8.5 for 循环练习 - 水仙花

* 需求：在控制台输出所有的 “水仙花数”
* 解释：什么是水仙花数？
  + 水仙花数，指的是一个三位数，个位、十位、百位的数字立方和等于原数
    - 例如153 3\*3\*3 + 5\*5\*5 + 1\*1\*1 = 153
* 思路：
  1. 获取所有的三位数，准备进行筛选，最小的三位数为 100，最大的三位数为 999，使用 for 循环获取
  2. 获取每一个三位数的个位，十位，百位，做 if 语句判断是否是水仙花数
* 示例代码

public class ForTest04 {  
 public static void main(String[] args) {  
 //输出所有的水仙花数必然要使用到循环，遍历所有的三位数，三位数从100开始，到999结束  
 /\*  
 for(int i=100; i<=999; i++) {  
  
 }  
 \*/  
 for(int i=100; i<1000; i++) {  
 //在计算之前获取三位数中每个位上的值  
 int ge = i%10;  
 int shi = i/10%10;  
 int bai = i/10/10%10;  
  
 //判定条件是将三位数中的每个数值取出来，计算立方和后与原始数字比较是否相等  
 if(ge\*ge\*ge + shi\*shi\*shi + bai\*bai\*bai == i) {  
 //输出满足条件的数字就是水仙花数  
 System.out.println(i);  
 }  
 }  
 }  
}

### 8.6 for 循环练习 - 统计水仙花数个数

* 需求：统计 “水仙花数” 一共有多少个，并在控制台输出个数
* 示例代码：

public class ForTest05 {  
 public static void main(String[] args) {  
 //定义变量count，用于保存“水仙花数”的数量，初始值为0  
 int count = 0;  
  
 //输出所有的水仙花数必然要使用到循环，遍历所有的三位数，三位数从100开始，到999结束  
 for(int i=100; i<1000; i++) {  
 //在计算之前获取三位数中每个位上的值  
 int ge = i%10;  
 int shi = i/10%10;  
 int bai = i/10/10%10;  
  
 //在判定水仙花数的过程中，满足条件不再输出，更改为修改count的值，使count+1  
 if(ge\*ge\*ge + shi\*shi\*shi + bai\*bai\*bai == i) {  
 count++;  
 }  
 }  
  
 //打印输出最终结果  
 System.out.println("水仙花共有：" + count + "个");  
 }  
}

* 本题要点：
  + 今后如果需求带有统计 xxx，请先想到计数器变量
  + 计数器变量定义的位置，必须在循环外部

1. while 循环

### 9.1 while 结构

* while 循环完整格式：
* 初始化语句;  
  while (条件判断语句) {  
   循环体语句;  
   条件控制语句;  
  }
* while 循环执行流程：
* ①执行初始化语句
* ②执行条件判断语句，看其结果是 true 还是 false
* 如果是 false，循环结束
* 如果是 true，继续执行
* ③执行循环体语句
* ④执行条件控制语句
* ⑤回到②继续
* 示例代码：

public class WhileDemo {  
 public static void main(String[] args) {  
 //需求：在控制台输出5次"HelloWorld"  
 //for循环实现  
 for(int i=1; i<=5; i++) {  
 System.out.println("HelloWorld");  
 }  
 System.out.println("--------");  
  
 //while循环实现  
 int j = 1;  
 while(j<=5) {  
 System.out.println("HelloWorld");  
 j++;  
 }  
  
 }  
}

### 9.2 while 循环练习 - 珠穆朗玛峰

* 需求：世界最高山峰是珠穆朗玛峰 (8844.43 米 = 8844430 毫米)，假如我有一张足够大的纸，它的厚度是 0.1 毫米。请问，我折叠多少次，可以折成珠穆朗玛峰的高度?
* 示例代码：

public class WhileTest {  
 public static void main(String[] args) {  
 //定义一个计数器，初始值为0  
 int count = 0;  
  
 //定义纸张厚度  
 double paper = 0.1;  
  
 //定义珠穆朗玛峰的高度  
 int zf = 8844430;  
  
 //因为要反复折叠，所以要使用循环，但是不知道折叠多少次，这种情况下更适合使用while循环  
 //折叠的过程中当纸张厚度大于珠峰就停止了，因此继续执行的要求是纸张厚度小于珠峰高度  
 while(paper <= zf) {  
 //循环的执行过程中每次纸张折叠，纸张的厚度要加倍  
 paper \*= 2;  
  
 //在循环中执行累加，对应折叠了多少次  
 count++;  
 }  
  
 //打印计数器的值  
 System.out.println("需要折叠：" + count + "次");  
 }  
}

1. 循环细节

### 10.1 do…while 循环结构

* 完整格式：
* 初始化语句;  
  do {  
   循环体语句;  
   条件控制语句;  
  }while(条件判断语句);
* 执行流程：
* ① 执行初始化语句
* ② 执行循环体语句
* ③ 执行条件控制语句
* ④ 执行条件判断语句，看其结果是 true 还是 false
* 如果是 false，循环结束
* 如果是 true，继续执行
* ⑤ 回到②继续
* 示例代码：

public class DoWhileDemo {  
 public static void main(String[] args) {  
 //需求：在控制台输出5次"HelloWorld"  
 //for循环实现  
 for(int i=1; i<=5; i++) {  
 System.out.println("HelloWorld");  
 }  
 System.out.println("--------");  
  
 //do...while循环实现  
 int j = 1;  
 do {  
 System.out.println("HelloWorld");  
 j++;  
 }while(j<=5);  
  
 }  
}

### 10.2 三种循环的区别

* 三种循环的区别
  + for 循环和 while 循环先判断条件是否成立，然后决定是否执行循环体（先判断后执行）
  + do…while 循环先执行一次循环体，然后判断条件是否成立，是否继续执行循环体（先执行后判断）
* for 循环和 while 的区别
  + 条件控制语句所控制的自增变量，因为归属 for 循环的语法结构中，在 for 循环结束后，就不能再次被访问到了
  + 条件控制语句所控制的自增变量，对于 while 循环来说不归属其语法结构中，在 while 循环结束后，该变量还可以继续使用
* 死循环（无限循环）的三种格式
  1. for(;;){}
  2. while(true){}
  3. do {} while(true);

### 10.3 跳转控制语句

* 跳转控制语句（break）
  + 跳出循环，结束循环
* 跳转控制语句（continue）
  + 跳过本次循环，继续下次循环
* 注意： continue 只能在循环中进行使用！

### 10.4 循环嵌套

* 循环嵌套概述：在循环中，继续定义循环
* 示例代码：

public class ForForDemo {  
 public static void main(String[] args) {  
 for(int hour=0; hour<24; hour++) {  
 for(int minute=0; minute<60; minute++) {  
 System.out.println(hour + "时" + minute + "分");  
 }  
 System.out.println("--------");  
 }  
 }  
}

* 理解：
  + 请反复理解这句话（整个内循环，就是外循环的一个循环体，内部循环体没有执行完毕，外循环是不会继续向下执行的）
* 结论：
  + 外循环执行一次，内循环执行一圈

1. Random

### 11.1 Random 产生随机数

* 概述：
  + Random 类似 Scanner，也是 Java 提供好的 API，内部提供了产生随机数的功能
    - API 后续课程详细讲解，现在可以简单理解为 Java 已经写好的代码
* 使用步骤：
  1. 导入包
  + import java.util.Random;
  1. 创建对象
  + Random r = new Random();
  1. 产生随机数
  + int num = r.nextInt(10);
  + 解释： 10 代表的是一个范围，如果括号写 10，产生的随机数就是 0-9，括号写 20，参数的随机数则是 0-19
* 示例代码：

import java.util.Random;  
  
public class RandomDemo {  
 public static void main(String[] args) {  
 //创建对象  
 Random r = new Random();  
  
 //用循环获取10个随机数  
 for(int i=0; i<10; i++) {  
 //获取随机数  
 int number = r.nextInt(10);  
 System.out.println("number:" + number);  
 }  
  
 //需求：获取一个1-100之间的随机数  
 int x = r.nextInt(100) + 1;  
 System.out.println(x);  
 }  
}

### 11.2 Random 练习 - 猜数字

* 需求：
* 程序自动生成一个 1-100 之间的数字，使用程序实现猜出这个数字是多少？
* 当猜错的时候根据不同情况给出相应的提示
* A. 如果猜的数字比真实数字大，提示你猜的数据大了
* B. 如果猜的数字比真实数字小，提示你猜的数据小了
* C. 如果猜的数字与真实数字相等，提示恭喜你猜中了
* 示例代码：

import java.util.Random;  
import java.util.Scanner;  
  
public class RandomTest {  
 public static void main(String[] args) {  
 //要完成猜数字的游戏，首先需要有一个要猜的数字，使用随机数生成该数字，范围1到100  
 Random r = new Random();  
 int number = r.nextInt(100) + 1;  
  
 while(true) {  
 //使用程序实现猜数字，每次均要输入猜测的数字值，需要使用键盘录入实现  
 Scanner sc = new Scanner(System.in);  
  
 System.out.println("请输入你要猜的数字：");  
 int guessNumber = sc.nextInt();  
  
 //比较输入的数字和系统产生的数据，需要使用分支语句。这里使用if..else..if..格式，根据不同情况进行猜测结果显示  
 if(guessNumber > number) {  
 System.out.println("你猜的数字" + guessNumber + "大了");  
 } else if(guessNumber < number) {  
 System.out.println("你猜的数字" + guessNumber + "小了");  
 } else {  
 System.out.println("恭喜你猜中了");  
 break;  
 }  
 }  
 }  
}

1. IDEA 开发工具

参见：IDEA 工具安装详解. pdf  
[IDEA 工具安装详解. pdf](https://download.csdn.net/download/qq_39997939/85072242)

1. 数组

### 13.1 什么是数组

数组 (array) 是一种用于存储多个相同类型数据的存储模型

### 13.2 数组定义格式

#### 13.2.1 第一种

数据类型 [] 数组名

示例：

int[] arr;   
double[] arr;   
char[] arr;

定义了一个 int 类型的数组, 数组名是 arr

#### 13.2.2 第二种 (不推荐)

数据类型 数组名 []

示例：

int arr[];  
double arr[];  
char arr[];

定义了一个 int 类型的变量, 变量名是 arr 数组

### 13.3 数组动态初始化

#### 13.3.1 什么是动态初始化

数组动态初始化就是只给定数组的长度，由系统给出默认初始化值

#### 13.3.2 动态初始化格式

数据类型[] 数组名 = new 数据类型[数组长度];

int[] arr = new int[3];

#### 13.3.3 动态初始化格式详解

* 等号左边：
  + int: 数组的数据类型
  + []: 代表这是一个数组
  + arr: 代表数组的名称
* 等号右边：
  + new: 为数组开辟内存空间
  + int: 数组的数据类型
  + []: 代表这是一个数组
  + 3: 代表数组的长度

### 13.4 数组元素访问

#### 13.4.1 什么是索引

每一个存储到数组的元素，都会自动的拥有一个编号，从 0 开始。

这个自动编号称为数组索引 (index)，可以通过数组的索引访问到数组中的元素。

#### 13.4.2 访问数组元素格式

数组名[索引];

#### 13.4.3 示例代码

public class ArrayDemo {  
 public static void main(String[] args) {  
 int[] arr = new int[3];  
  
 //输出数组名  
 System.out.println(arr); //[I@880ec60  
  
 //输出数组中的元素  
 System.out.println(arr[0]);  
 System.out.println(arr[1]);  
 System.out.println(arr[2]);  
 }  
}

### 13.5 内存分配

#### 13.5.1 内存概述

内存是计算机中的重要原件，临时存储区域，作用是运行程序。

我们编写的程序是存放在硬盘中的，在硬盘中的程序是不会运行的。

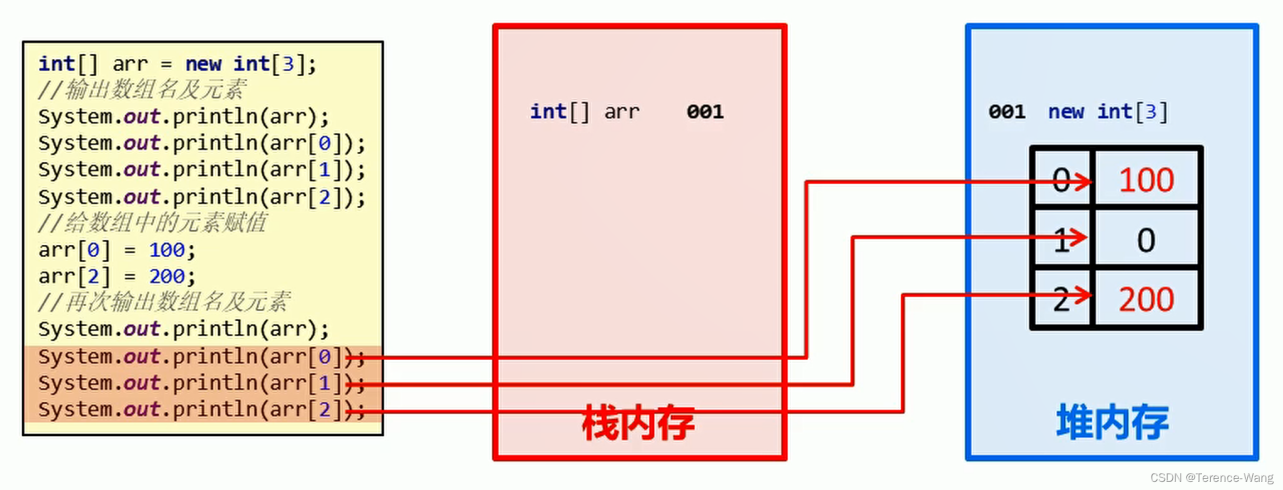
必须放进内存中才能运行，运行完毕后会清空内存。

Java 虚拟机要运行程序，必须要对内存进行空间的分配和管理。

#### 13.5.2 java 中的内存分配

* 目前我们只需要记住两个内存，分别是：栈内存和堆内存

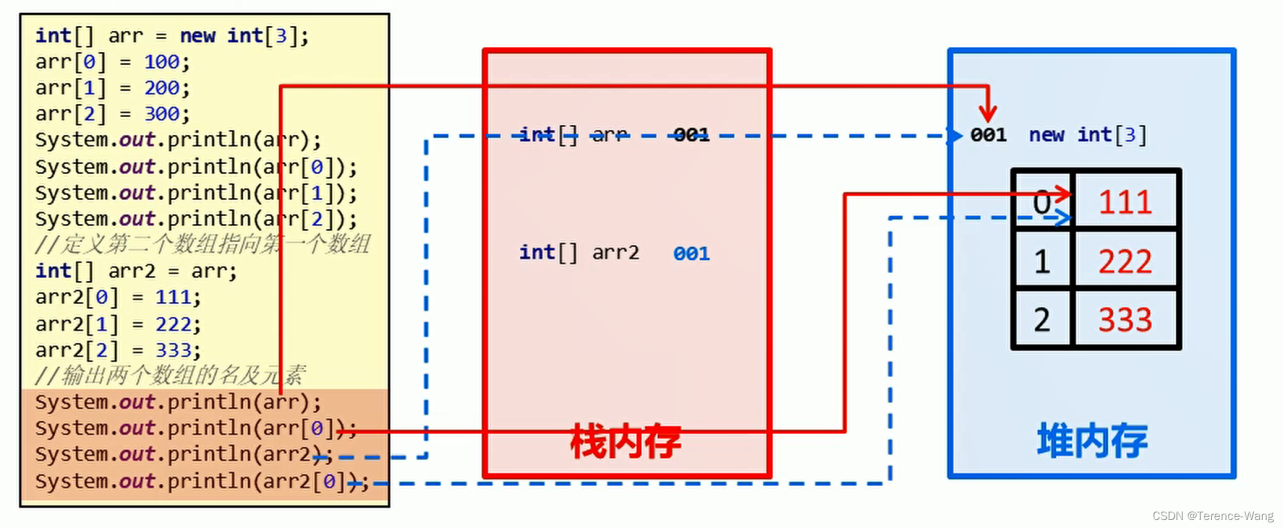
### 13.6 单个数组的内存图



### 13.7 多个数组的内存图



### 13.8 多个数组指向相同内存图



### 13.9 数组静态初始化

#### 13.9.1 什么是静态初始化

在创建数组时，直接将元素确定

#### 13.9.2 静态初始化格式

* 完整版格式
* 数据类型[] 数组名 = new 数据类型[]{元素1,元素2,...};
* 简化版格式
* 数据类型[] 数组名 = {元素1,元素2,...};

#### 13.9.3 示例代码

public class ArrayDemo {  
 public static void main(String[] args) {  
 //定义数组  
 int[] arr = {1, 2, 3};  
  
 //输出数组名  
 System.out.println(arr);  
  
 //输出数组中的元素  
 System.out.println(arr[0]);  
 System.out.println(arr[1]);  
 System.out.println(arr[2]);  
 }  
}

### 13.10 数组操作的两个常见小问题

#### 13.10.1 索引越界异常

* 出现原因

public class ArrayDemo {  
 public static void main(String[] args) {  
 int[] arr = new int[3];  
 System.out.println(arr[3]);  
 }  
}

数组长度为 3，索引范围是 0~2，但是我们却访问了一个 3 的索引。

程序运行后，将会抛出 ArrayIndexOutOfBoundsException 数组越界异常。在开发中，数组的越界异常是不能出现的，一旦出现了，就必须要修改我们编写的代码。

* 解决方案
* 将错误的索引修改为正确的索引范围即可！

#### 13.10.2 空指针异常

* 出现原因

public class ArrayDemo {  
 public static void main(String[] args) {  
 int[] arr = new int[3];  
 //把null赋值给数组  
 arr = null;  
 System.out.println(arr[0]);  
 }  
}

arr = null 这行代码，意味着变量 arr 将不会在保存数组的内存地址，也就不允许再操作数组了，因此运行的时候会抛出 NullPointerException 空指针异常。在开发中，数组的越界异常是不能出现的，一旦出现了，就必须要修改我们编写的代码。

* 解决方案
* 给数组一个真正的堆内存空间引用即可！

### 13.11 数组遍历

* 数组遍历：就是将数组中的每个元素分别获取出来，就是遍历。遍历也是数组操作中的基石。
* public class ArrayTest01 {  
   public static void main(String[] args) {  
   int[] arr = { 1, 2, 3, 4, 5 };  
   System.out.println(arr[0]);  
   System.out.println(arr[1]);  
   System.out.println(arr[2]);  
   System.out.println(arr[3]);  
   System.out.println(arr[4]);  
   }  
  }
* 以上代码是可以将数组中每个元素全部遍历出来，但是如果数组元素非常多，这种写法肯定不行，因此我们需要改造成循环的写法。数组的索引是 0 到 lenght-1 ，可以作为循环的条件出现。

public class ArrayTest01 {  
 public static void main(String[] args) {  
 //定义数组  
 int[] arr = {11, 22, 33, 44, 55};  
  
 //使用通用的遍历格式  
 for(int x=0; x<arr.length; x++) {  
 System.out.println(arr[x]);  
 }  
 }  
}

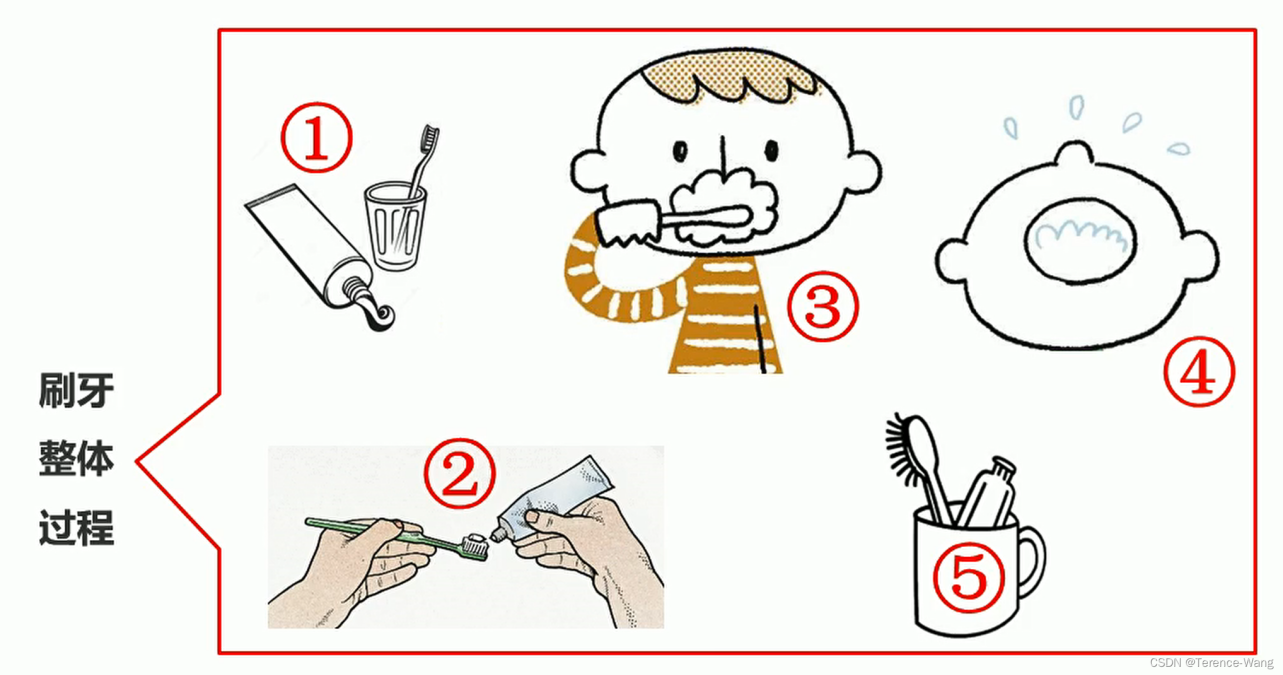
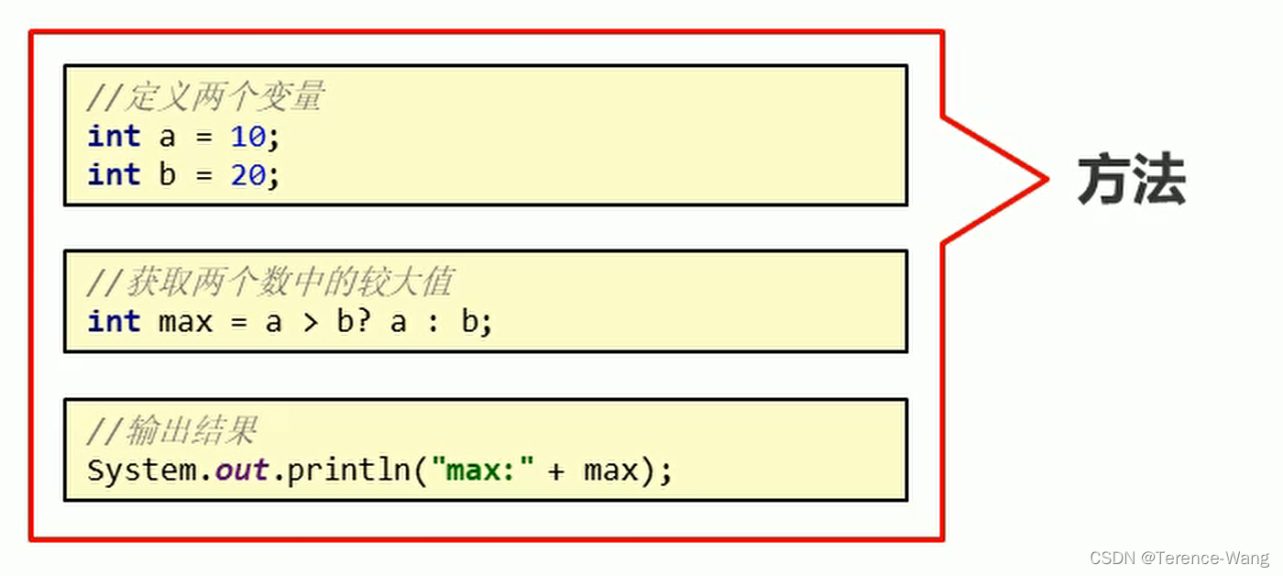
### 13.12 数组最值

* 最大值获取：从数组的所有元素中找出最大值。
* 实现思路：
  + 定义变量，保存数组 0 索引上的元素
  + 遍历数组，获取出数组中的每个元素
  + 将遍历到的元素和保存数组 0 索引上值的变量进行比较
  + 如果数组元素的值大于了变量的值，变量记录住新的值
  + 数组循环遍历结束，变量保存的就是数组中的最大值
* 代码实现：

public class ArrayTest02 {  
 public static void main(String[] args) {  
 //定义数组  
 int[] arr = {12, 45, 98, 73, 60};  
  
 //定义一个变量，用于保存最大值  
 //取数组中第一个数据作为变量的初始值  
 int max = arr[0];  
  
 //与数组中剩余的数据逐个比对，每次比对将最大值保存到变量中  
 for(int x=1; x<arr.length; x++) {  
 if(arr[x] > max) {  
 max = arr[x];  
 }  
 }  
  
 //循环结束后打印变量的值  
 System.out.println("max:" + max);  
  
 }  
}

1. 方法概述

### 14.1 方法的概念

方法（method）是将具有独立功能的代码块组织成为一个整体，使其具有特殊功能的代码集  
  


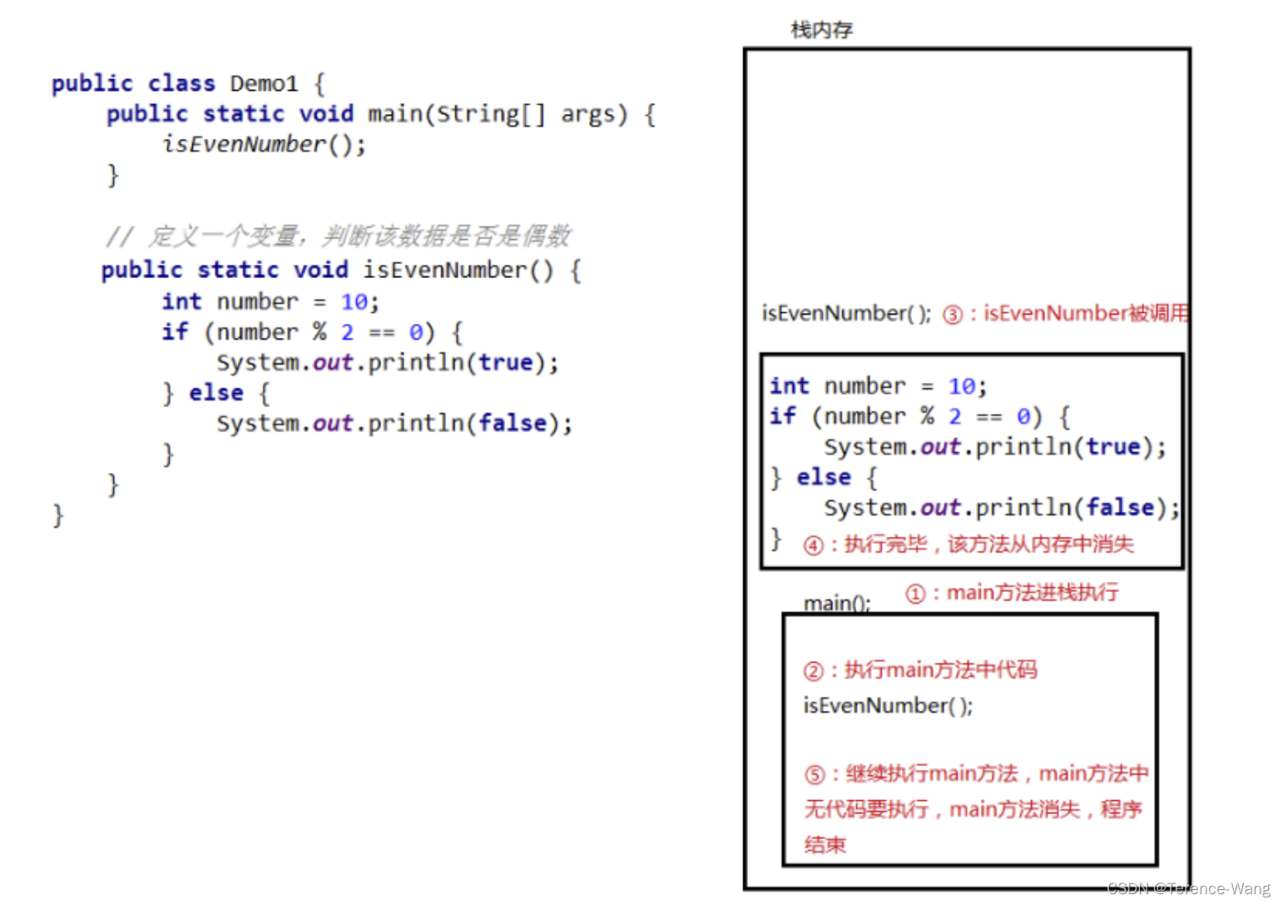
* 注意：
  + 方法必须先创建才可以使用，该过程成为方法定义
  + 方法创建后并不是直接可以运行的，需要手动使用后，才执行，该过程成为方法调用

### 14.2. 方法的定义和调用

#### 14.2.1 无参数方法定义和调用

* 定义格式：
* public static void 方法名 ( ) {  
   // 方法体;  
  }
* 范例：
* public static void method ( ) {  
   // 方法体;  
  }
* 调用格式：
* 方法名();
* 范例：
* method();
* 注意：
* 方法必须先定义，后调用，否则程序将报错

#### 14.2.2 方法调用过程图解



* 总结：每个方法在被调用执行的时候，都会进入栈内存，并且拥有自己独立的内存空间，方法内部代码调用完毕之后，会从栈内存中弹栈消失。

#### 14.2.3 无参数方法的练习

* 需求：设计一个方法用于打印两个数中的较大数
* 思路：
  + ①定义一个方法，用于打印两个数字中的较大数，例如 getMax()
  + ②方法中定义两个变量，用于保存两个数字
  + ③使用分支语句分两种情况对两个数字的大小关系进行处理
  + ④在 main() 方法中调用定义好的方法
* 代码：

public class MethodTest {  
 public static void main(String[] args) {  
 //在main()方法中调用定义好的方法  
 getMax();  
 }  
  
 //定义一个方法，用于打印两个数字中的较大数，例如getMax()  
 public static void getMax() {  
 //方法中定义两个变量，用于保存两个数字  
 int a = 10;  
 int b = 20;  
  
 //使用分支语句分两种情况对两个数字的大小关系进行处理  
 if(a > b) {  
 System.out.println(a);  
 } else {  
 System.out.println(b);  
 }  
 }  
}

### 14.3 带参数方法定义和调用

#### 14.3.1 带参数方法定义和调用

* 定义格式：
* 参数：由数据类型和变量名组成 - 数据类型 变量名
* 参数范例：int a
* public static void 方法名 (参数1) {  
   方法体;  
  }  
    
  public static void 方法名 (参数1, 参数2, 参数3...) {  
   方法体;  
  }
* 范例：
* public static void isEvenNumber(int number){  
   ...  
  }  
  public static void getMax(int num1, int num2){  
   ...  
  }  
  + 注意：
  + 方法定义时，参数中的数据类型与变量名都不能缺少，缺少任意一个程序将报错  
      
    方法定义时，多个参数之间使用逗号( ，)分隔
* 调用格式：
* 方法名(参数)；  
    
  方法名(参数1,参数2);
* 范例：
* isEvenNumber(10);  
    
  getMax(10,20);  
  + 方法调用时，参数的数量与类型必须与方法定义中的设置相匹配，否则程序将报错

#### 14.3.2 形参和实参

1. 形参：方法定义中的参数

等同于变量定义格式，例如：int number

1. 实参：方法调用中的参数

等同于使用变量或常量，例如： 10 number

#### 14.3.3 带参数方法练习

* 需求：设计一个方法用于打印两个数中的较大数，数据来自于方法参数 }
* 思路：
  + ①定义一个方法，用于打印两个数字中的较大数，例如 getMax()
  + ②为方法定义两个参数，用于接收两个数字
  + ③使用分支语句分两种情况对两个数字的大小关系进行处理
  + ④在 main() 方法中调用定义好的方法（使用常量）
  + ⑤在 main() 方法中调用定义好的方法（使用变量）
* 代码：

public class MethodTest {  
 public static void main(String[] args) {  
 //在main()方法中调用定义好的方法（使用常量）  
 getMax(10,20);  
 //调用方法的时候，人家要几个，你就给几个，人家要什么类型的，你就给什么类型的  
 //getMax(30);  
 //getMax(10.0,20.0);  
  
 //在main()方法中调用定义好的方法（使用变量）  
 int a = 10;  
 int b = 20;  
 getMax(a, b);  
 }  
  
 //定义一个方法，用于打印两个数字中的较大数，例如getMax()  
 //为方法定义两个参数，用于接收两个数字  
 public static void getMax(int a, int b) {  
 //使用分支语句分两种情况对两个数字的大小关系进行处理  
 if(a > b) {  
 System.out.println(a);  
 } else {  
 System.out.println(b);  
 }  
 }  
}

### 14.4 带返回值方法的定义和调用

#### 14.4.1 带返回值方法定义和调用

* 定义格式
* public static 数据类型 方法名 ( 参数 ) {   
   return 数据 ;  
  }
* 范例
* public static boolean isEvenNumber( int number ) {   
   return true ;  
  }  
  public static int getMax( int a, int b ) {  
   return 100 ;  
  }  
  + 注意：
    - 方法定义时 return 后面的返回值与方法定义上的数据类型要匹配，否则程序将报错
* 调用格式
* 方法名 ( 参数 ) ;  
  数据类型 变量名 = 方法名 ( 参数 ) ;
* 范例
* isEvenNumber ( 5 ) ;  
  boolean flag = isEvenNumber ( 5 );   
  + 注意：
    - 方法的返回值通常会使用变量接收，否则该返回值将无意义

#### 14.4.2 带返回值方法练习

* 需求：设计一个方法可以获取两个数的较大值，数据来自于参数
* 思路：
  + ①定义一个方法，用于获取两个数字中的较大数
  + ②使用分支语句分两种情况对两个数字的大小关系进行处理
  + ③根据题设分别设置两种情况下对应的返回结果
  + ④在 main() 方法中调用定义好的方法并使用变量保存
  + ⑤在 main() 方法中调用定义好的方法并直接打印结果
* 代码：
* public class MethodTest {  
   public static void main(String[] args) {  
   //在main()方法中调用定义好的方法并使用变量保存  
   int result = getMax(10,20);  
   System.out.println(result);  
    
   //在main()方法中调用定义好的方法并直接打印结果  
   System.out.println(getMax(10,20));  
   }  
    
   //定义一个方法，用于获取两个数字中的较大数  
   public static int getMax(int a, int b) {  
   //使用分支语句分两种情况对两个数字的大小关系进行处理  
   //根据题设分别设置两种情况下对应的返回结果  
   if(a > b) {  
   return a;  
   } else {  
   return b;  
   }  
   }  
  }

### 14.5 方法的注意事项

#### 14.5.1 方法的注意事项

* 方法不能嵌套定义
  + 示例代码：
  + public class MethodDemo {  
     public static void main(String[] args) {  
      
     }  
      
     public static void methodOne() {  
     public static void methodTwo() {  
     // 这里会引发编译错误!!!  
     }  
     }  
    }
* void 表示无返回值，可以省略 return，也可以单独的书写 return，后面不加数据
  + 示例代码：
  + public class MethodDemo {  
     public static void main(String[] args) {  
      
     }  
     public static void methodTwo() {  
     //return 100; 编译错误，因为没有具体返回值类型  
     return;   
     //System.out.println(100); return语句后面不能跟数据或代码  
     }  
    }

#### 14.5.2 方法的通用格式

* 格式：
* public static 返回值类型 方法名(参数) {  
   方法体;   
   return 数据 ;  
  }
* 解释：
  + public static 修饰符，目前先记住这个格式
  + 返回值类型 方法操作完毕之后返回的数据的数据类型
  + 如果方法操作完毕，没有数据返回，这里写 void，而且方法体中一般不写 return
  + 方法名 调用方法时候使用的标识
  + 参数 由数据类型和变量名组成，多个参数之间用逗号隔开
  + 方法体 完成功能的代码块
  + return 如果方法操作完毕，有数据返回，用于把数据返回给调用者
* 定义方法时，要做到两个明确
  + 明确返回值类型：主要是明确方法操作完毕之后是否有数据返回，如果没有，写 void；如果有，写对应的数据类型
  + 明确参数：主要是明确参数的类型和数量
* 调用方法时的注意：
  + void 类型的方法，直接调用即可
  + 非 void 类型的方法，推荐用变量接收调用

### 14.6 方法重载

#### 14.6.1 方法重载

* 方法重载概念
* 方法重载指同一个类中定义的多个方法之间的关系，满足下列条件的多个方法相互构成重载
  + 多个方法在同一个类中
  + 多个方法具有相同的方法名
  + 多个方法的参数不相同，类型不同或者数量不同
* 注意：
  + 重载仅对应方法的定义，与方法的调用无关，调用方式参照标准格式
  + 重载仅针对同一个类中方法的名称与参数进行识别，与返回值无关，换句话说不能通过返回值来判定两个方法是否相互构成重载
* 正确范例：
* public class MethodDemo {  
   public static void fn(int a) {  
   //方法体  
   }  
   public static int fn(double a) {  
   //方法体  
   }  
  }  
    
  public class MethodDemo {  
   public static float fn(int a) {  
   //方法体  
   }  
   public static int fn(int a , int b) {  
   //方法体  
   }  
  }
* 错误范例：
* public class MethodDemo {  
   public static void fn(int a) {  
   //方法体  
   }  
   public static int fn(int a) { /\*错误原因：重载与返回值无关\*/  
   //方法体  
   }  
  }  
    
  public class MethodDemo01 {  
   public static void fn(int a) {  
   //方法体  
   }  
  }   
  public class MethodDemo02 {  
   public static int fn(double a) { /\*错误原因：这是两个类的两个fn方法\*/  
   //方法体  
   }  
  }

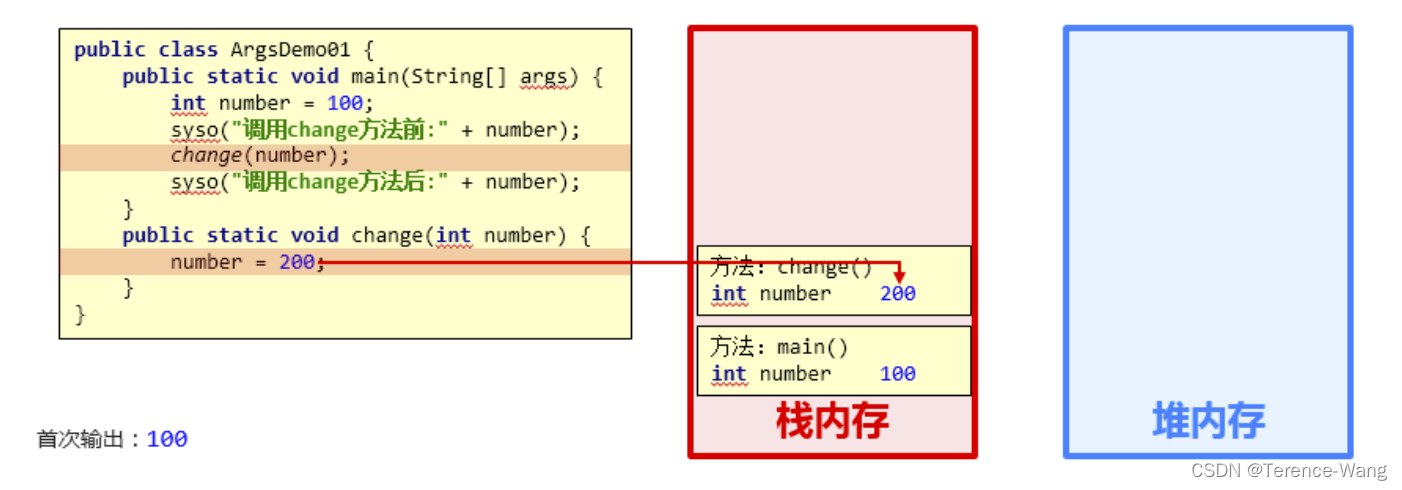
#### 14.6.2 方法重载练习

* 需求：使用方法重载的思想，设计比较两个整数是否相同的方法，兼容全整数类型（byte,short,int,long）
* 思路：
  + ①定义比较两个数字的是否相同的方法 compare() 方法，参数选择两个 int 型参数
  + ②定义对应的重载方法，变更对应的参数类型，参数变更为两个 long 型参数
  + ③定义所有的重载方法，两个 byte 类型与两个 short 类型参数
  + ④完成方法的调用，测试运行结果
* 代码：
* public class MethodTest {  
   public static void main(String[] args) {  
   //调用方法  
   System.out.println(compare(10, 20));  
   System.out.println(compare((byte) 10, (byte) 20));  
   System.out.println(compare((short) 10, (short) 20));  
   System.out.println(compare(10L, 20L));  
   }  
    
   //int  
   public static boolean compare(int a, int b) {  
   System.out.println("int");  
   return a == b;  
   }  
    
   //byte  
   public static boolean compare(byte a, byte b) {  
   System.out.println("byte");  
   return a == b;  
   }  
    
   //short  
   public static boolean compare(short a, short b) {  
   System.out.println("short");  
   return a == b;  
   }  
    
   //long  
   public static boolean compare(long a, long b) {  
   System.out.println("long");  
   return a == b;  
   }  
    
  }

### 14.7 方法的参数传递

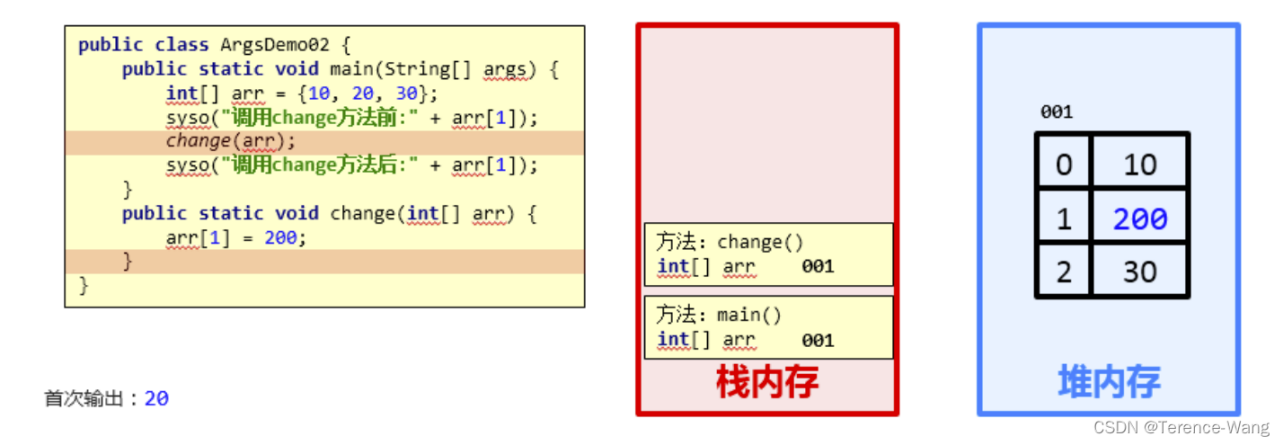
#### 14.7.1 方法参数传递基本类型

* 测试代码：
* public class ArgsDemo01 {  
   public static void main(String[] args) {  
   int number = 100;  
   System.out.println("调用change方法前：" + number);  
   change(number);  
   System.out.println("调用change方法后：" + number);  
   }  
    
   public static void change(int number) {  
   number = 200;  
   }  
  }
* 结论：
  + 基本数据类型的参数，形式参数的改变，不影响实际参数
* 结论依据：
  + 每个方法在栈内存中，都会有独立的栈空间，方法运行结束后就会弹栈消失



#### 14.7.2 方法参数传递引用类型

* 测试代码：
* public class ArgsDemo02 {  
   public static void main(String[] args) {  
   int[] arr = {10, 20, 30};  
   System.out.println("调用change方法前：" + arr[1]);  
   change(arr);  
   System.out.println("调用change方法后：" + arr[1]);  
   }  
    
   public static void change(int[] arr) {  
   arr[1] = 200;  
   }  
  }
* 结论：
  + 对于引用类型的参数，形式参数的改变，影响实际参数的值
* 结论依据：
  + 引用数据类型的传参，传入的是地址值，内存中会造成两个引用指向同一个内存的效果，所以即使方法弹栈，堆内存中的数据也已经是改变后的结果



#### 14.7.3 数组遍历

* 需求：设计一个方法用于数组遍历，要求遍历的结果是在一行上的。例如：[11, 22, 33, 44, 55]
* 思路：
  + ①因为要求结果在一行上输出，所以这里需要在学习一个新的输出语句 System.out.print(“内容”);
  + System.out.println(“内容”); 输出内容并换行
  + System.out.print(“内容”); 输出内容不换行
  + System.out.println(); 起到换行的作用
  + ②定义一个数组，用静态初始化完成数组元素初始化
  + ③定义一个方法，用数组遍历通用格式对数组进行遍历
  + ④用新的输出语句修改遍历操作
  + ⑤调用遍历方法
* 代码：
* public class MethodTest01 {  
   public static void main(String[] args) {  
   //定义一个数组，用静态初始化完成数组元素初始化  
   int[] arr = {11, 22, 33, 44, 55};  
    
   //调用方法  
   printArray(arr);  
   }  
    
   //定义一个方法，用数组遍历通用格式对数组进行遍历  
   /\*  
   两个明确：  
   返回值类型：void  
   参数：int[] arr  
   \*/  
   public static void printArray(int[] arr) {  
   System.out.print("[");  
   for(int x=0; x<arr.length; x++) {  
   if(x == arr.length-1) {  
   System.out.print(arr[x]);  
   } else {  
   System.out.print(arr[x]+", ");  
   }  
   }  
   System.out.println("]");  
   }  
  }

#### 14.7.4 数组最大值

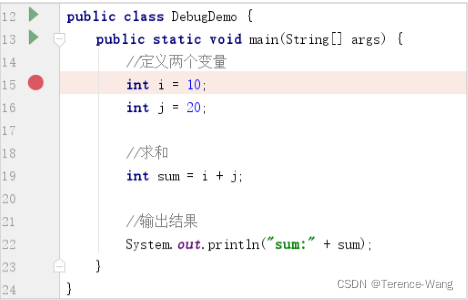
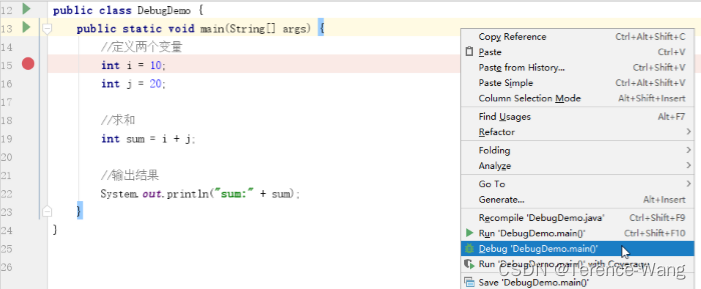
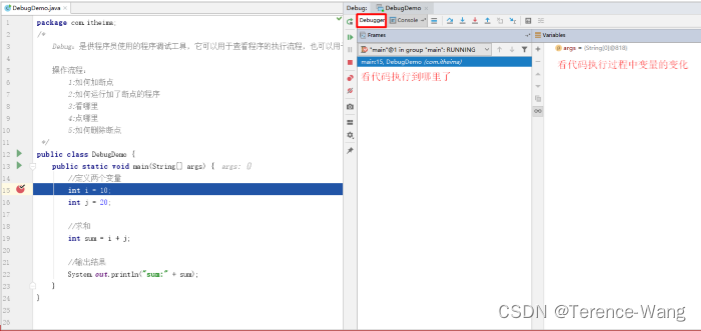
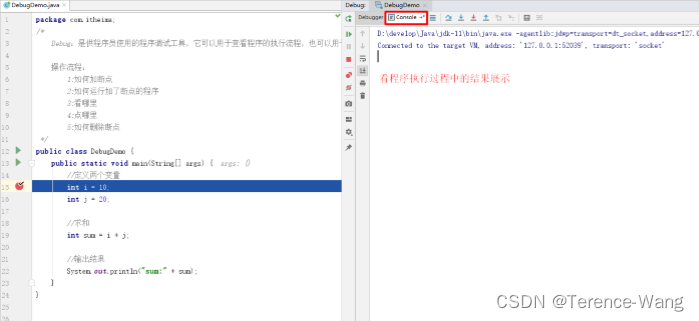
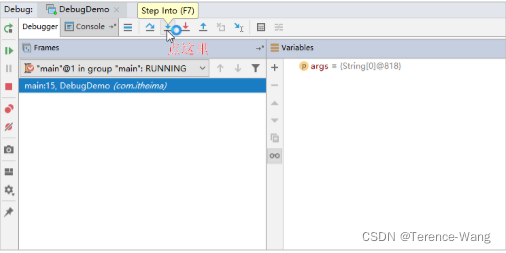
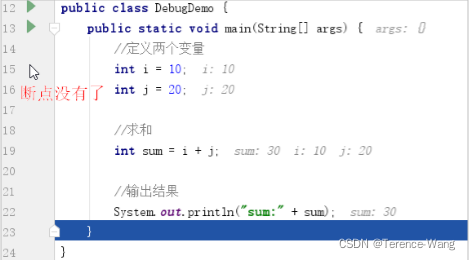
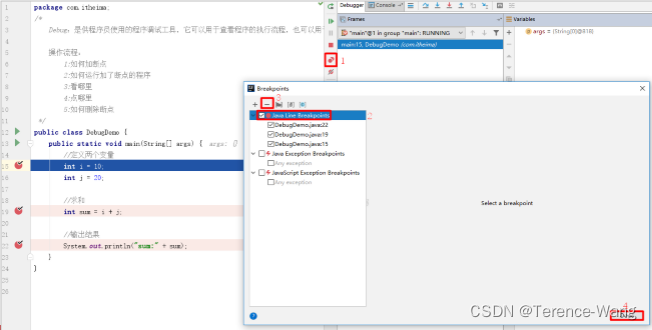
* 需求：设计一个方法用于获取数组中元素的最大值
* 思路：
  + ①定义一个数组，用静态初始化完成数组元素初始化
  + ②定义一个方法，用来获取数组中的最大值，最值的认知和讲解我们在数组中已经讲解过了
  + ③调用获取最大值方法，用变量接收返回结果
  + ④把结果输出在控制台
* 代码：
* public class MethodTest02 {  
   public static void main(String[] args) {  
   //定义一个数组，用静态初始化完成数组元素初始化  
   int[] arr = {12, 45, 98, 73, 60};  
    
   //调用获取最大值方法，用变量接收返回结果  
   int number = getMax(arr);  
    
   //把结果输出在控制台  
   System.out.println("number:" + number);  
   }  
    
   //定义一个方法，用来获取数组中的最大值  
   /\*  
   两个明确：  
   返回值类型：int  
   参数：int[] arr  
   \*/  
   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;  
   }  
  }

1. Debug 模式

### 15.1 什么是 Debug 模式

是供程序员使用的程序调试工具，它可以用于查看程序的执行流程，也可以用于追踪程序执行过程来调试程序。

### 15.2 Debug 模式操作流程【应用】

* 如何加断点
  + 选择要设置断点的代码行，在行号的区域后面单击鼠标左键即可
* 
* 如何运行加了断点的程序
  + 在代码区域右键 Debug 执行
* 
* 看哪里
  + 看 Debugger 窗口
* 
  + 看 Console 窗口
* 
* 点哪里
  + 点 Step Into (F7) 这个箭头，也可以直接按 F7
* 
* 如何删除断点
  + 选择要删除的断点，单击鼠标左键即可
* 
  + 如果是多个断点，可以每一个再点击一次。也可以一次性全部删除
* 

1. 基础练习

### 16.1 减肥计划 if 版本

#### 16.1.1 案例需求

输入星期数，显示今天的减肥活动  
 周一：跑步  
 周二：游泳  
 周三：慢走  
 周四：动感单车  
 周五：拳击  
 周六：爬山  
 周日：好好吃一顿

#### 16.1.2 代码实现

/\*  
 思路：  
 1:键盘录入一个星期数，用一个变量接收  
 2:对星期数进行判断，这里用 if 语句实现  
 3:在对应的语句控制中输出对应的减肥活动  
 \*/  
public class Test01 {  
 public static void main(String[] args) {  
 //键盘录入一个星期数，用一个变量接收  
 Scanner sc = new Scanner(System.in);  
  
 System.out.println("请输入一个星期数：");  
 int week = sc.nextInt();  
  
 //对星期数进行判断，这里用 if 语句实现  
 if (week < 1 || week > 7) {  
 System.out.println("你输入的星期数有误");  
 } else if (week == 1) {  
 System.out.println("跑步");  
 } else if (week == 2) {  
 System.out.println("游泳");  
 } else if (week == 3) {  
 System.out.println("慢走");  
 } else if (week == 4) {  
 System.out.println("动感单车");  
 } else if (week == 5) {  
 System.out.println("拳击");  
 } else if (week == 6) {  
 System.out.println("爬山");  
 } else {  
 System.out.println("好好吃一顿");  
 }  
 }  
}

### 16.2 减肥计划 switch 版本

#### 16.2.1 案例需求

输入星期数，显示今天的减肥活动  
 周一：跑步  
 周二：游泳  
 周三：慢走  
 周四：动感单车  
 周五：拳击  
 周六：爬山  
 周日：好好吃一顿

#### 16.2.2 代码实现

/\*  
 思路：  
 1:键盘录入一个星期数，用一个变量接收  
 2:对星期数进行判断，这里用 switch 语句实现  
 3:在对应的语句控制中输出对应的减肥活动  
  
 导包：  
 1:手动导包 import java.util.Scanner;  
 2:快捷键导包 Alt+Enter  
 3:自动导包  
 \*/  
public class Test02 {  
 public static void main(String[] args) {  
 //键盘录入一个星期数，用一个变量接收  
 Scanner sc = new Scanner(System.in);  
  
 System.out.println("请输入一个星期数：");  
 int week = sc.nextInt();  
  
 //对星期数进行判断，这里用 switch 语句实现  
 switch (week) {  
 case 1:  
 System.out.println("跑步");  
 break;  
 case 2:  
 System.out.println("游泳");  
 break;  
 case 3:  
 System.out.println("慢走");  
 break;  
 case 4:  
 System.out.println("动感单车");  
 break;  
 case 5:  
 System.out.println("拳击");  
 break;  
 case 6:  
 System.out.println("爬山");  
 break;  
 case 7:  
 System.out.println("好好吃一顿");  
 break;  
 default:  
 System.out.println("你输入的星期数有误");  
 }  
 }  
}

### 16.3 逢七跳过

#### 16.3.1 案例需求

朋友聚会的时候可能会玩一个游戏：逢七过。  
 规则是：从任意一个数字开始报数，当你要报的数字包含 7 或者是 7 的倍数时都要说：过。  
 为了帮助大家更好的玩这个游戏，这里我们直接在控制台打印出 1-100 之间的满足逢七必过规则的数据。  
 这样，大家将来在玩游戏的时候，就知道哪些数据要说：过。

#### 16.3.2 代码实现

/\*  
 思路：  
 1:数据在1-100之间，用for循环实现数据的获取  
 2:根据规则，用if语句实现数据的判断：要么个位是7，要么十位是7，要么能够被7整除  
 3:在控制台输出满足规则的数据  
 \*/  
public class Test03 {  
 public static void main(String[] args) {  
 //数据在1-100之间，用for循环实现数据的获取  
 for(int x=1; x<=100; x++) {  
 //根据规则，用if语句实现数据的判断：要么个位是7，要么十位是7，要么能够被7整除  
 if(x%10==7 || x/10%10==7 || x%7==0) {  
 //在控制台输出满足规则的数据  
 System.out.println(x);  
 }  
 }  
 }  
}

### 16.4 不死神兔

#### 16.4.1 案例需求

有一对兔子，从出生后第 3 个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，  
 假如兔子都不死，问第二十个月的兔子对数为多少？

#### 16.4.2 代码实现

/\*  
 思路：  
 1:为了存储多个月的兔子对数，定义一个数组，用动态初始化完成数组元素的初始化，长度为20  
 2:因为第1个月，第2个月兔子的对数是已知的，都是1，所以数组的第1个元素，第2个元素值也都是1  
 3:用循环实现计算每个月的兔子对数  
 4:输出数组中最后一个元素的值，就是第20个月的兔子对数  
 \*/  
public class Test04 {  
 public static void main(String[] args) {  
 //为了存储多个月的兔子对数，定义一个数组，用动态初始化完成数组元素的初始化，长度为20  
 int[] arr = new int[20];  
  
 //因为第1个月，第2个月兔子的对数是已知的，都是1，所以数组的第1个元素，第2个元素值也都是1  
 arr[0] = 1;  
 arr[1] = 1;  
  
 //用循环实现计算每个月的兔子对数  
 for(int x=2; x<arr.length; x++) {  
 arr[x] = arr[x-2] + arr[x-1];  
 }  
  
 //输出数组中最后一个元素的值，就是第20个月的兔子对数  
 System.out.println("第二十个月兔子的对数是：" + arr[19]);  
 }  
}

### 16.5 百钱白鸡

#### 16.5.1 案例需求

我国古代数学家张丘建在《算经》一书中提出的数学问题：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。  
 百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？

#### 16.5.2 代码实现

/\*  
 思路：  
 1:第1层循环，用于表示鸡翁的范围，初始化表达式的变量定义为 x=0，判断条件是x<=20  
 2:第2层循环，用于表示鸡母的范围，初始化表达式的变量定义为 y=0，判断条件是y<=33  
 3:这个时候，用于表示鸡雏的变量 z = 100 – x – y  
 4:判断表达式 z%3==0 和表达式 5\*x + 3\*y + z/3 = 100 是否同时成立，如果成立，输出对应的 x，y，z 的值，就是对应的鸡翁，鸡母，鸡雏的值  
 \*/  
public class Test05 {  
 public static void main(String[] args) {  
 //第1层循环，用于表示鸡翁的范围，初始化表达式的变量定义为 x=0，判断条件是x<=20  
 for(int x=0; x<=20; x++) {  
 //第2层循环，用于表示鸡母的范围，初始化表达式的变量定义为 y=0，判断条件是y<=33  
 for(int y=0; y<=33; y++) {  
 //这个时候，用于表示鸡雏的变量 z = 100 – x – y  
 int z = 100 - x - y;  
  
 //判断表达式 z%3==0 和表达式 5\*x + 3\*y + z/3 = 100 是否同时成立  
 if(z%3==0 && 5\*x+3\*y+z/3==100) {  
 System.out.println(x+","+y+","+z);  
 }  
 }  
 }  
 }  
}

### 16.6 数组元素求和

#### 16.6.1 案例需求

有这样的一个数组，元素是 {68,27,95,88,171,996,51,210}。求出该数组中满足要求的元素和，  
 要求是：求和的元素个位和十位都不能是 7，并且只能是偶数

#### 16.6.2 代码实现

/\*  
 思路：  
 1:定义一个数组，用静态初始化完成数组元素的初始化  
 2:定义一个求和变量，初始值是0  
 3:遍历数组，获取到数组中的每一个元素  
 4:判断该元素是否满足条件，如果满足条件就累加  
 5:输出求和变量的值  
 \*/  
public class Test06 {  
 public static void main(String[] args) {  
 //定义一个数组，用静态初始化完成数组元素的初始化  
 int[] arr = {68, 27, 95, 88, 171, 996, 51, 210};  
  
 //定义一个求和变量，初始值是0  
 int sum = 0;  
  
 //遍历数组，获取到数组中的每一个元素  
 for(int x=0; x<arr.length; x++) {  
 //判断该元素是否满足条件，如果满足条件就累加  
 if(arr[x]%10!=7 && arr[x]/10%10!=7 && arr[x]%2==0) {  
 sum += arr[x];  
 }  
 }  
  
 //输出求和变量的值  
 System.out.println("sum:" + sum);  
 }  
}

### 16.7 判断两个数组是否相同

#### 16.7.1 案例需求

定义一个方法，用于比较两个数组的内容是否相同

#### 16.7.2 代码实现

/\*  
 思路：  
 1:定义两个数组，分别使用静态初始化完成数组元素的初始化  
 2:定义一个方法，用于比较两个数组的内容是否相同  
 3:比较两个数组的内容是否相同，按照下面的步骤实现就可以了  
 首先比较数组长度，如果长度不相同，数组内容肯定不相同，返回false  
 其次遍历，比较两个数组中的每一个元素，只要有元素不相同，返回false  
 最后循环遍历结束后，返回true  
 4:调用方法，用变量接收  
 5:输出结果  
 \*/  
public class Test07 {  
 public static void main(String[] args) {  
 //定义两个数组，分别使用静态初始化完成数组元素的初始化  
 int[] arr = {11, 22, 33, 44, 55};  
 //int[] arr2 = {11, 22, 33, 44, 55};  
 int[] arr2 = {11, 22, 33, 44, 5};  
  
  
 //调用方法，用变量接收  
 boolean flag = compare(arr,arr2);  
 //输出结果  
 System.out.println(flag);  
 }  
  
 //定义一个方法，用于比较两个数组的内容是否相同  
 /\*  
 两个明确：  
 返回值类型：boolean  
 参数：int[] arr, int[] arr2  
 \*/  
 public static boolean compare(int[] arr, int[] arr2) {  
 //首先比较数组长度，如果长度不相同，数组内容肯定不相同，返回false  
 if(arr.length != arr2.length) {  
 return false;  
 }  
  
 //其次遍历，比较两个数组中的每一个元素，只要有元素不相同，返回false  
 for(int x=0; x<arr.length; x++) {  
 if(arr[x] != arr2[x]) {  
 return false;  
 }  
 }  
  
 //最后循环遍历结束后，返回true  
 return true;  
 }  
}

### 16.8 查找元素在数组中出现的索引位置

#### 16.8.1 案例需求

已知一个数组 arr = {19, 28, 37, 46, 50}; 键盘录入一个数据，查找该数据在数组中的索引。

并在控制台输出找到的索引值。如果没有查找到，则输出 - 1

#### 16.8.2 代码实现

/\*  
 思路：  
 1:定义一个数组，用静态初始化完成数组元素的初始化  
 2:键盘录入要查找的数据，用一个变量接收  
 3:定义一个索引变量，初始值为-1  
 4:遍历数组，获取到数组中的每一个元素  
 5:拿键盘录入的数据和数组中的每一个元素进行比较，如果值相同，就把该值对应的索引赋值给索引变量,并结束循环  
 6:输出索引变量  
 \*/  
public class Test08 {  
 public static void main(String[] args) {  
 //定义一个数组，用静态初始化完成数组元素的初始化  
 int[] arr = {19, 28, 37, 46, 50};  
  
 //键盘录入要查找的数据，用一个变量接收  
 Scanner sc = new Scanner(System.in);  
 System.out.println("请输入要查找的数据：");  
 int number = sc.nextInt();  
  
 //调用方法  
 int index = getIndex(arr, number);  
  
 //输出索引变量  
 System.out.println("index: " + index);  
 }  
  
 //查找指定的数据在数组中的索引  
 /\*  
 两个明确：  
 返回值类型：int  
 参数：int[] arr, int number  
 \*/  
 public static int getIndex(int[] arr, int number) {  
 //定义一个索引变量，初始值为-1  
 int index = -1;  
  
 //遍历数组，获取到数组中的每一个元素  
 for(int x=0; x<arr.length; x++) {  
 //拿键盘录入的数据和数组中的每一个元素进行比较，如果值相同，就把该值对应的索引赋值给索引变量,并结束循环  
 if(arr[x] == number) {  
 index = x;  
 break;  
 }  
 }  
  
 //返回索引  
 return index;  
 }  
}

### 16.9 数组元素反转

#### 16.9.1 案例需求

已知一个数组 arr = {19, 28, 37, 46, 50}; 用程序实现把数组中的元素值交换，  
 交换后的数组 arr = {50, 46, 37, 28, 19}; 并在控制台输出交换后的数组元素。

#### 16.9.2 代码实现

/\*  
 思路：  
 1:定义一个数组，用静态初始化完成数组元素的初始化  
 2:循环遍历数组，这一次初始化语句定义两个索引变量，判断条件是开始索引小于等于结束索引  
 3:变量交换  
 4:遍历数组  
 \*/  
public class Test09 {  
 public static void main(String[] args) {  
 //定义一个数组，用静态初始化完成数组元素的初始化  
 int[] arr = {19, 28, 37, 46, 50};  
  
 //调用反转的方法  
 reverse(arr);  
  
 //遍历数组  
 printArray(arr);  
 }  
  
 /\*  
 两个明确：  
 返回值类型：void  
 参数：int[] arr  
 \*/  
 public static void reverse(int[] arr) {  
 //循环遍历数组，这一次初始化语句定义两个索引变量，判断条件是开始索引小于等于结束索引  
 for (int start = 0, end = arr.length - 1; start <= end; start++, end--) {  
 //变量交换  
 int temp = arr[start];  
 arr[start] = arr[end];  
 arr[end] = temp;  
 }  
 }  
  
 /\*  
 两个明确：  
 返回值类型：void  
 参数：int[] arr  
 \*/  
 public static void printArray(int[] arr) {  
 System.out.print("[");  
  
 for (int x = 0; x < arr.length; x++) {  
 if (x == arr.length - 1) {  
 System.out.print(arr[x]);  
 } else {  
 System.out.print(arr[x] + ", ");  
 }  
 }  
  
 System.out.println("]");  
 }  
}

### 16.10 评委打分

#### 16.10.1 案例需求

在编程竞赛中，有 6 个评委为参赛的选手打分，分数为 0-100 的整数分。  
 选手的最后得分为：去掉一个最高分和一个最低分后 的 4 个评委平均值 (不考虑小数部分)。

#### 16.10.2 代码实现

/\*  
 思路：  
 1:定义一个数组，用动态初始化完成数组元素的初始化，长度为6  
 2:键盘录入评委分数  
 3:由于是6个评委打分，所以，接收评委分数的操作，用循环改进  
 4:定义方法实现获取数组中的最高分(数组最大值)，调用方法  
 5:定义方法实现获取数组中的最低分(数组最小值) ，调用方法  
 6:定义方法实现获取数组中的所有元素的和(数组元素求和) ，调用方法  
 7:按照计算规则进行计算得到平均分  
 8:输出平均分  
 \*/  
public class Test10 {  
 public static void main(String[] args) {  
 //定义一个数组，用动态初始化完成数组元素的初始化，长度为6  
 int[] arr = new int[6];  
  
 //键盘录入评委分数  
 Scanner sc = new Scanner(System.in);  
  
 //由于是6个评委打分，所以，接收评委分数的操作，用循环改进  
 for(int x=0; x<arr.length; x++) {  
 System.out.println("请输入第" + (x + 1) + "个评委的打分：");  
 arr[x] = sc.nextInt();  
 }  
  
 //printArray(arr);  
  
 //定义方法实现获取数组中的最高分(数组最大值)，调用方法  
 int max = getMax(arr);  
  
 //定义方法实现获取数组中的最低分(数组最小值) ，调用方法  
 int min = getMin(arr);  
  
 //定义方法实现获取数组中的所有元素的和(数组元素求和) ，调用方法  
 int sum = getSum(arr);  
  
 //按照计算规则进行计算得到平均分  
 int avg = (sum - max - min) / (arr.length - 2);  
  
 //输出平均分  
 System.out.println("选手的最终得分是：" + avg);  
  
 }  
  
 /\*  
 两个明确：  
 返回值类型：int  
 参数：int[] arr  
 \*/  
 public static int getSum(int[] arr) {  
 int sum = 0;  
  
 for(int x=0; x<arr.length; x++) {  
 sum += arr[x];  
 }  
  
 return sum;  
 }  
  
 /\*  
 两个明确：  
 返回值类型：int  
 参数：int[] arr  
 \*/  
 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;  
 }  
  
 /\*  
 两个明确：  
 返回值类型：int  
 参数：int[] arr  
 \*/  
 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;  
 }  
  
 //遍历数组  
 public static void printArray(int[] arr) {  
 System.out.print("[");  
  
 for (int x = 0; x < arr.length; x++) {  
 if (x == arr.length - 1) {  
 System.out.print(arr[x]);  
 } else {  
 System.out.print(arr[x] + ", ");  
 }  
 }  
  
 System.out.println("]");  
 }  
}