

Java 编程技术

徐行健

计算机与信息工程学院
内蒙古师范大学

2018 年 9 月 15 日

目录

1

第一部分：导论

- 第一章：概述
- 第二章：开发环境的搭建

2

第二部分：Java 语言基础

- 第三章：基本数据类型与数组
- 第四章：运算符、表达式和语句

3

第三部分：面向对象

- 第五章：面向对象编程
- 第六章：类与对象

- 第七章：子类与继承

- 第八章：接口与实现

- 第九章：泛型

第四部分：实用功能

- 第十章：常用工具类

- 第十一章：容器类

- 第十二章：输入输出流

第五部分：高级功能

- 第十三章：多线程技术

- 第十四章：基于 Socket 的网络编程

- 第十五章：反射机制

第一部分：导论

第一章：概述

- Java 是一门**非常重要的**课程，以后很多同学的毕业设计乃至工作都会使用到 Java
- C/C++ 没学好，不代表你学不好 Java！与前两者相比，这是一门相对“简单”的语言
- 数据结构没学好，不代表你学不好 Java！本课程**偏向工程应用**，很少涉及纯粹的算法
- 本课程包含绝大部分 Java 的基础知识，不包含 Web 相关内容

课程要求

- 实验课代码自己敲，不要复制粘贴！只会照着书或者 PPT 敲代码的，最后肯定学不好本课程！
- 作业算做实验课成绩。作业会有自动查重，不管谁抄谁，所有重复者作业（重复率大于 70%）一律 0 分！
- 本课程不设所谓的及格率下限！考试作弊者重修从严处理！大四清考按照正常出卷和判卷标准执行！

File 1

[LuLu_970822667015581696_1016658554810880000.cpp \(97%\)](#)
20171103091 913006001235767296_1016658554810880000.cpp (97%)
20171103091 913006001235767296_1016658554810880000.cpp (97%)
20161106056 970901552537300992_1016658554810880000.cpp (97%)
20161106056 970901552537300992_1016658554810880000.cpp (97%)
20161106056 970901552537300992_1016658554810880000.cpp (97%)
20161106056 970901552537300992_1016658554810880000.cpp (97%)
20161106056 950156016778170368_1016658554810880000.cpp (97%)
20161106052 950156016778170368_1016658554810880000.cpp (97%)
20161106052 950156016778170368_1016658554810880000.cpp (97%)
20161106052 950156016778170368_1016658554810880000.cpp (97%)
20161106052 950156016778170368_1016658554810880000.cpp (97%)
20161102904 970823612122271744_1016658554810880000.cpp (97%)
20171105121 913595960112001024_1016658554810880000.cpp (97%)

File 2

[刘建星_949620162255081472_1016658554810880000.cpp \(97%\)](#)
刘建星_949620162255081472_1016658554810880000.cpp (97%)
[LuLu_970822667015581696_1016658554810880000.cpp \(97%\)](#)
刘建星_949620162255081472_1016658554810880000.cpp (97%)
[LuLu_970822667015581696_1016658554810880000.cpp \(97%\)](#)
20171103091 913006001235767296_1016658554810880000.cpp (97%)
刘建星_949620162255081472_1016658554810880000.cpp (97%)
[LuLu_970822667015581696_1016658554810880000.cpp \(97%\)](#)
20171103091 913006001235767296_1016658554810880000.cpp (97%)
20161106056 970901552537300992_1016658554810880000.cpp (97%)
刘建星_949620162255081472_1016658554810880000.cpp (97%)
[LuLu_970822667015581696_1016658554810880000.cpp \(97%\)](#)
20171103091 913006001235767296_1016658554810880000.cpp (97%)
20161106056 970901552537300992_1016658554810880000.cpp (97%)
刘建星_949620162255081472_1016658554810880000.cpp (97%)
[LuLu_970822667015581696_1016658554810880000.cpp \(97%\)](#)
20171103091 913006001235767296_1016658554810880000.cpp (97%)
20161106056 970901552537300992_1016658554810880000.cpp (97%)
刘建星_949620162255081472_1016658554810880000.cpp (97%)
[LuLu_970822667015581696_1016658554810880000.cpp \(97%\)](#)
20171103091 913006001235767296_1016658554810880000.cpp (97%)
20161106056 970901552537300992_1016658554810880000.cpp (97%)
刘建星_949620162255081472_1016658554810880000.cpp (97%)
[LuLu_970822667015581696_1016658554810880000.cpp \(97%\)](#)
20171103091 913006001235767296_1016658554810880000.cpp (97%)
20161106052 950156016778170368_1016658554810880000.cpp (97%)
20161106052 950156016778170368_1016658554810880000.cpp (97%)
20171105137 913593991976792064_1016658554810880000.cpp (97%)

- ① 第一代语言：打孔机，纯粹的机器语言，易于机器理解和执行。程序员需要掌握全部的硬件细节
- ② 第二代语言：汇编语言，比第一代语言更加容易编写，用抽象的符号表示指令、数据和寄存器，程序员需要掌握大量的硬件细节
- ③ 第三代语言：高级语言，易于人类设计程序，对大部分硬件做了抽象
 - 分类标准一：
 - 面向过程的语言：C、Fortran、Pascal
 - 不纯粹的面向对象的语言：C++
 - 纯粹的面向对象的语言：Java
 - 函数式语言：Lisp
 - 分类标准二：
 - 编译型语言：C、C++、Java
 - 解释型语言：Python、Javascript、Perl、PHP

Java 和 Javascript 是两种完全不同、毫无关系的语言！

- “纯粹”的面向对象：不像 C++ 那样包含过程化编程语言的要素
- 跨平台：通过 Java 虚拟机（JVM）实现了平台无关性，一次编译，到处运行（write once, run everywhere）
- 开发效率高：易于编写、易于调试，易于团队合作，更加适合大型软件项目
- 健壮：吸收了 C/C++ 的优点，同时去掉了其不太健壮的部分（如指针，内存的申请和释放），很大程度上缓解了“空指针”、“内存泄漏”等问题
- 易于学习：使得 Java 的使用人数多，且对 Java 程序员的要求较低
- 生态圈非常发达：可用软件工具包、中间件非常多，对于大多数类型的软件项目都不需要从头写起

- 运行时效率较低（与 C/C++ 相比）：由“跨平台”的特点导致，编译后的 Java 程序并非直接运行在操作系统中，而是运行在 JVM 上
- 语法冗繁，不够简明：由“易于学习”的特点导致，语言的表达能力不够，同样的一个语义，Java 往往需要多行代码才能实现
- 对系统底层操控能力差：由“健壮”和“跨平台”的特点导致，不能使用 JVM 提供的功能和接口

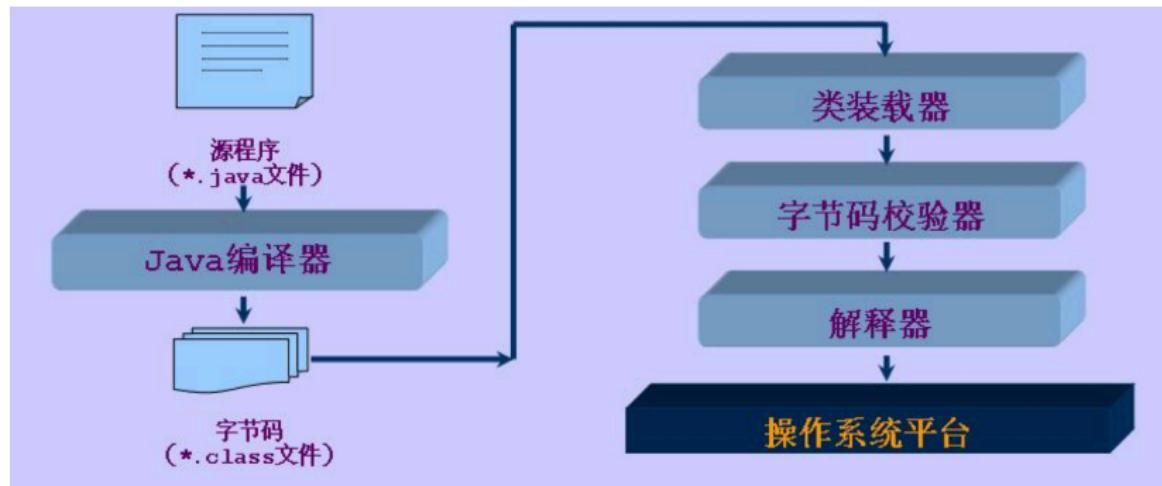
设计语言时的考量

- “易于人类理解”还是“易于机器理解”：前者开发效率高，后者运行效率高
- “易于一般人使用”还是“易于聪明人使用”：前者表达力高，但是学习难度大；后者表达力低，但是学习难度低
- “更加底层”还是“更加抽象”：前者对系统操控能力大，后者不容易犯错（如空指针），更加安全

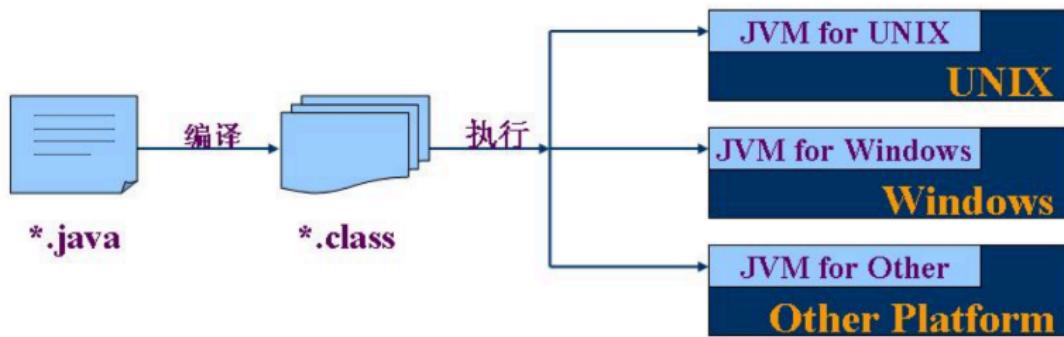
- 适合的领域：
 - Web 服务器系统：如办公自动化、企业信息化、电子政务等
 - 分布式应用：如 Hadoop、HDFS 等
 - 数据密集型作业
 - 有跨平台需求
 - Android 开发（采用 Java 语法和部分 API）
- 不适合的领域：
 - 实时性要求高的系统：如军工、航控等
 - 计算密集型作业：如科学计算、仿真模拟等
 - 对性能要求苛刻的软件：如 HTTP 服务器、数据库等
 - 底层设备驱动

Java 程序运行机制

- Java 虚拟机 (JVM, Java Virtual Machine)
- 垃圾收集机制



- Java 源代码编译后形成的字节码无法被操作系统直接执行，只能被 JVM 执行
- JVM 可以理解为一个以 Java 字节码为“机器指令”的虚拟 CPU
- 不同的操作系统和平台有不同的 JVM 实现
- JVM 位于操作系统之上，对其功能进行了封装，以此屏蔽了不同平台底层系统的差异性，实现“一次编译，到处运行”



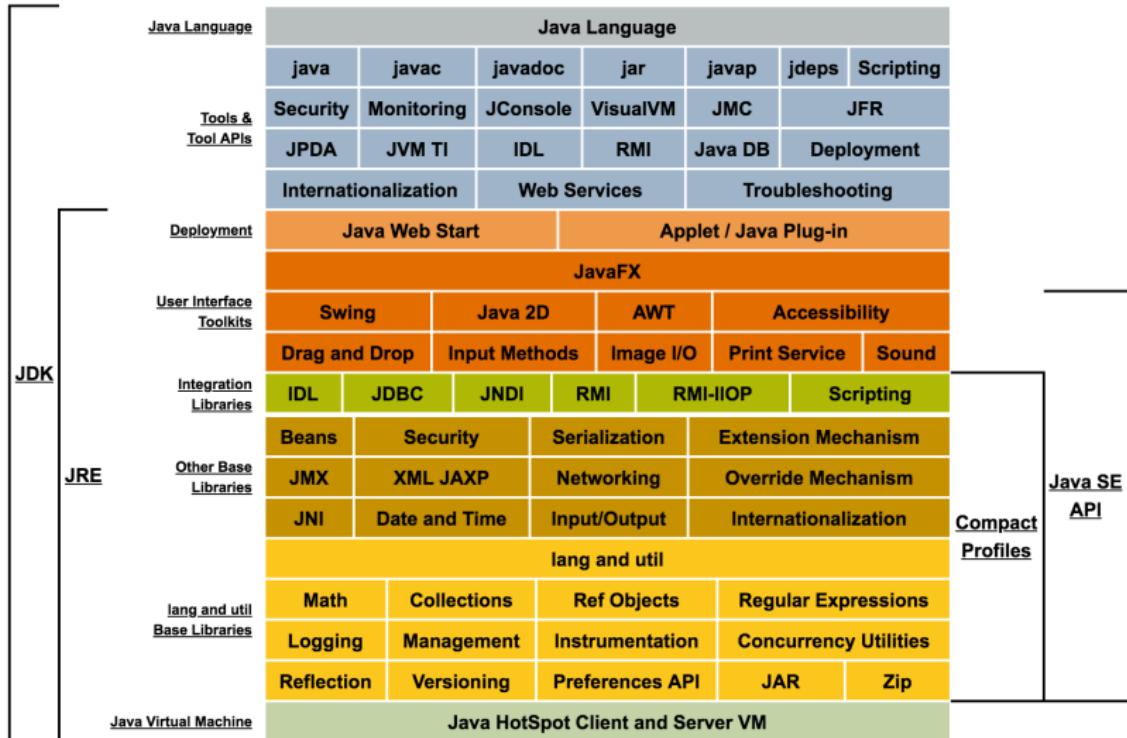
- 程序员无需也不能像 C/C++ 中那样手动分配和回收内存 (`malloc`、`free` 等函数)
- 不再使用的内存空间会被 JVM 按照特定算法定期予以释放回收，此过程是 JVM 自动执行的，程序员无法精准的干预和控制
- Java 的垃圾回收机制使程序员无需再承担内存管理的责任，有效降低了 Java 程序设计的复杂度和学习成本

JVM 和垃圾收集带来的“负面影响”

JVM 在进行垃圾回收时是需要占用 CPU 资源的，此时整个 JVM 中运行的用户线程会停止工作，直到垃圾收集结束，这极大的影响了 Java 程序的实时性。对于 C/C++ 这种手动管理内存的语言就不存在此问题，因为是程序员负责管理内存。

- 1995 · JDK Beta
- 1996 · JDK 1.0
- 1998 · J2SE 1.2
- 2002 · J2SE 1.4
- 2004 · J2SE 5.0, 升级大版本号, 直接从 1.4 升级为 5.0
- 2006 · Java SE 6, 将 J2SE 改名为 Java SE (还包括 Java EE, Java ME)
- 2009 · 年 Oracle 收购 Sun, 取得 Java 所有权
- 2014 · Java SE 8 (LTS, Long Term Support, 长支持版本)
- 2018 · Java SE 10

Java 平台体系



- JDK 和 JRE：
 - JRE: Java Runtime Environment, 包含用户运行编译后的 **Java** 程序的运行时环境
 - JDK: Java Development Kit, 包含 JRE 以及 **Java** 程序的开发工具包（编译、调试等）。常用的 JDK 有官方的 Oracle JDK、开源的 Open JDK 和 IBM 的 J9（现开源为 OpenJ9）
 - 终端用户只需安装 JRE 即可，只有开发人员才需要安装 JDK。JDK 安装包较大
- Java 的版本
 - Java ME: Java Micro Edition (逐渐被 Android 等所取代，用途越来越少，almost dead)
 - Java SE: Java Standard Edition (主流版本)
 - Java EE: Java Enterprise Edition (系统过于庞大复杂，应用场景也不多)

第二章：开发环境的搭建

JDK 的下载和安装

- ① 利用搜索引擎，找到 Oracle JDK 的官方下载页面：Java SE 下载
- ② 根据自己操作系统的类型，选择下载对应版本的 Java SE 8

Java SE 8u171/ 8u172

Java SE 8u171 包括重要的错误修复。Oracle 强烈建议所有 Java SE 8 用户升级到该版本。Java SE 8u172 是一个补丁集更新，包括所有 8u171 特性及其他错误修复（版本说明中进行了介绍）。

[更多信息](#) ➔

- 安装说明
- 版本说明
- Oracle 许可
- Java SE 许可信息用户手册
 - 包括第三方许可
- 认证的系统配置
- 自述文件
 - JDK 自述文件
 - JRE 自述文件

JDK

JDK

[DOWNLOAD](#) *

服务器 JRE

[DOWNLOAD](#) *

JRE

[DOWNLOAD](#) *

- ③ 双击安装

环境变量一般是指在操作系统中用来指定操作系统运行环境的一些参数，这些参数一般以键值对的形式存在。

Windows 中可以通过“系统属性”->“高级”选项卡->“环境变量”对话框的方式增加、删除或者编辑修改环境变量。

- 增加 `JAVA_HOME` 环境变量，将其赋值为 JDK 的安装目录：有些程序凭 `JAVA_HOME` 寻找 JDK 的安装目录
- 修改 `PATH` 环境变量，将 `JAVA_HOME/bin` 目录加入：使得可以在任何目录直接使用 JDK 提供的软件工具，如 `javac` 等

环境变量 `PATH` 的作用

当你在计算机安装 JDK 之后，输入 `javac` 或者 `java` 之类的命令是不能马上被计算机正确执行的，因为计算机不知道到哪里去找这两个命令。那么计算机该如何查找你输入的命令呢？Windows 操作系统是根据环境变量 `PATH` 来查找命令的。环境变量 `PATH` 的值是一系列路径，Windows 操作系统将在这一系列的路径中依次查找命令对应的程序文件，如果能找到这个命令，则该命令是可执行的；否则将出现“`XXX`不是内部命令或外部命令，也不是可运行的程序或批处理文件”的提示。

编写并运行第一个 Java 程序

- ① 使用任意纯文本编辑器（如记事本、np++ 等，不要使用 Word）编写程序
- ② 使用 `javac` 编译程序
- ③ 打开命令行窗口，使用 `java` 运行程序

```
Last login: Sun Jul 15 14:35:18 on ttys001
→ ~ vi HelloWorld.java
→ ~ javac HelloWorld.java
→ ~ java HelloWorld
Hello, World
```

- ① 右键新建“文本文件”，文件名为 `HelloWorld.java`
- ② 使用记事本打开该文件，并逐行输入如下代码：
- ③ 保存文件

_____ `HelloWorld.java` _____

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         // Prints "Hello, World"  
4         System.out.println("Hello, World");  
5     }  
6 }
```

- 源代码文件必须使用 `.java` 做为文件扩展名
- 源代码文件名（不包含扩展名）必须和代码中的类名相同！
- 源代码严格区分大小写
- 如果希望该源代码可以执行，必须定义程序入口函数，函数签名不能改变！

public static void main(String[] args)

函数签名

函数签名包含了：函数名、函数的参数列表、函数的返回值类型、函数的其他修饰 在 Java 中，函数也称为“方法”

- ① 打开命令行，使用 `cd` 命令切换到 `HelloWorld.java` 所在目录
- ② 编译 Java 源文件：`javac HelloWorld.java`，得到字节码程序 `HelloWorld`
- ③ 执行编译后的字节码程序：`java HelloWorld`，观察屏幕输出

- 代码风格又称编码风格、编码规范，是指编写源代码的书写风格，大体包括了：
 - 如何给变量、函数、类命名
 - 如何使用代码缩进
 - 代码布局，如花括号的摆放位置等
- 好的代码风格是出于工程上的要求，没有绝对的对错，只有好坏，代码风格的好坏和不同并不会直接导致程序出错
- 好的代码风格可以使得：
 - 代码更美观、更加容易阅读理解
 - 更容易发现编码过程中的错误
- 很多人不重视代码风格，认为数学、算法才是根本，或者认为只要代码能运行，怎么书写并不重要。这种思想是错误的！因为正规的现代软件工程已经脱离了一个人小作坊式的编写模式，好的、统一的代码风格有助于提高团队工作效率！

本课程代码风格的要求（一）

本课程要求同学们编写代码时，必须遵守如下代码风格（这也是最常见的 Java 代码风格，参考 Google 编码风格制定）：

- ① 变量的命名采用小驼峰命名法：除第一个单词小写之外，其他单词首字母大写，如：

int studentCount

- ② 类的命名采用大驼峰命名法：单词首字母均大写，如：

public class HelloWorld

- ③ 静态常量命名全大写，单词之间使用下划线（_）隔开，如：

public static final String CONSTANT_PI

- ④ 左花括号写在行末尾，不能新起一行；右花括号独占新的一行书写（也称 K & R 风格），如：

```
1 public static void main(String[] args) {  
2     ...  
3 }
```

本课程代码风格的要求（二）

- ⑤ 即使是可选的情况下，也要使用大括号，如：

```
1 if (count > 6) { //此花括号在语法上可以省略，但是按照我们的代码风
  ↪ 格，予以保留
2   flag = true;
3 }
```

- ⑥ 除非在 **for** 循环的定义中，操作符、运算符两边需各有一个空格（做为一个整体的操作符中间不需要加空格，如!=），如：

```
int studentCount = 10 - 1;
```

- ⑦ 如果字符的前面是逗号或分号，需要在前加一个空格：

```
for (int a=0; a++; a<10)
```

- ⑧ 小括号和中括号前留一个空格，后面不留空格

本课程代码风格的要求（三）

- ⑨ 采用四个空格或者一个制表符 tab 做为一个缩进层级
- ⑩ 一行最多只有一个以分号结尾的语句，一行最长 100 个字符
- ⑪ 不要像 C 语言那样在函数头部一次性声明很多变量，用到的时候再声明

错误代码风格举例

```
1 int a=3; //等号前后需要由空格
2 double d = 1.0; double c = 2.0; //一行只应写一个带分号的语句
3 public class studentScore {} //类名应该采取大驼峰写法，第一个单词首字
   ↳ 母应大写
4 static final String numCount; //静态常量应全部使用大写，单词间使用下划
   ↳ 线隔开
5 String student_name; //变量名应采取小驼峰写法
6
7 public void getScore()
8 { //此为 C 的代码风格，本课程中左花括号应写在函数声明行的最后
9 return this.score; //此处应该有四个空格或者一个 tab 的缩进
10 }
```

代码注释

注释的类型：

- 单行注释：

```
1 // 我是单行注释!
```

- 多行注释：

```
1 /* 我是  
2 * 多行注释!  
3 */
```

- 文档 (javadoc) 注释：

```
1 /**  
2 * 演示了文档注释  
3 * @author Ayan Ahmed  
4 * @version 1.2  
5 */
```

为什么必须写注释：

- 你的代码可能是要被别人阅读的，别人可能不理解你的逻辑
- 你的代码自己以后也要重新阅读，你自己可能也不能理解写作代码当时的逻辑
- 写作代码时，帮助自己理清思路

注意

注释是程序代码的有机组成部分，一边写代码，一边写注释。本课程作业要求同学们必须写注释解释自己的思路！

- IntelliJ IDEA：功能强大，高度集成，易于使用。除了 Java 也可作为常见绝大多数语言的开发环境
 - 社区版 (Community)：免费使用
 - 旗舰版 (Ultimate)：商业使用收费，教育和开源项目免费
- Eclipse：功能强大，开源，免费。有多个版本，分别打包了不同的插件，用于 Java SE、Java Web 等开发
- NetBeans：功能强大，开源，免费，使用人数较前两者更少

IDE 的选择

- 个人开发程序：三种 IDE 都很成熟，都可以很好的开发 Java 程序，写不好程序和 IDE 的选择无关！
- 团队开发程序：尽量和团队其他人使用的 IDE 保持一致
- 本课程选择 IntelliJ IDEA 进行授课，因为该 IDE 更加遵守 Java 项目的最佳实践，且大家以后如果学习 Android，官方推荐的也是该 IDE 的衍生产品

第二部分：Java 语言基础

第三章：基本数据类型与数组

- 定义：事先定义的，有特别意义的单词，被 Java 语言保留下来有着特殊含义和用途，所以有时又叫保留字
- 所有的 Java 关键字都是小写
- `goto` 和 `const` 虽目前未被 Java 使用，但是依旧是保留关键字
- 一般情况下，IDE 编辑器会将关键字以特殊高亮的形式展现

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	null
continue	goto	package	synchronized	

- 定义：用于给 Java 程序中变量、类、方法等要素命名的字符串序列
- Java 标识符规则：
 - 只能由字母、数字、下划线“_”、美元符“\$”号组成
 - 不能以数字开头，只能以字符、下划线、美元符号开头
 - 不能是 Java 中的关键字
 - 大小写敏感
- 通俗地说，凡是可以说自己起名字的地方，都要遵守标识符规则
- 标志符中的字母指 Unicode 中的字母，不光包括英文 26 个字母，甚至包括中文的汉字、希腊字母、日文片假名等
- 合法标志符举例：myName, My_name, Points, \$points,_sys_ta, OK, _23b, _3_
- 非法标志符举例：#name, 25name, class, &time, if

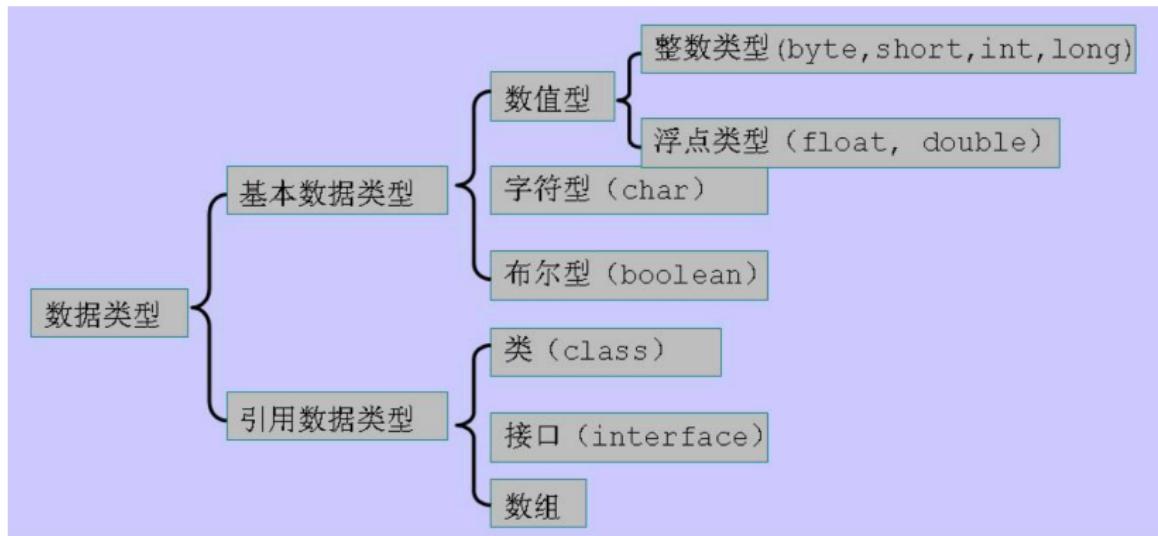
- 变量是 Java 中最基本的数据存储单元，其要素包括：变量名、变量的数据类型和作用域
- 变量必须先声明，后使用，声明格式为：
`type varName [= valValue]`

```
1 String a; // 声明了一个变量，但未赋值，相当于 C 中的空指针
2 int i = 100; // 声明了一个变量，并为其赋值
3 double a, b, c = 1.1; // 一次赋值三个变量
```

- 本质上讲，Java 中的变量其实就是内存中的一小块区域，这块区域可以使用变量名来访问

Java 中的数据类型

- Java 中共两大类数据类型：基本数据类型和引用数据类型
- 引用数据类型实际就是 C 语言中的指针引用！Java 中除了基本数据类型，其它本质上都是指针！
- Java 函数的参数如果是基本数据类型，那么传值调用；如果是引用类型，则本质上是传址调用！



数字型



为什么“1/2=0”？



- 布尔型数据用于逻辑运算，一般用于程序流程控制，使用 `boolean` 声明一个布尔型变量
- Java 中的布尔型数据只能取两个值 `true` 和 `false`，不能使用 `0` 或者 `null` 来表示（注意此处和 C 语言不同）

```
1 boolean a = true;
2 boolean b = false;
3
4 if (a) { // 使用布尔型数据用于流程控制
5 }
6
7 boolean c = 0; // 错误，Java 中布尔型只能取 true 或者 false
8
9 int d = -1;
10 if (d) { // 错误，Java 中只能使用布尔型做为程序流程控制
11 }
```

- 字符型数据表示一个字符，用单引号'包围（注意不能用双引号，双引号包围的是字符串！）
- Java 字符采用 Unicode 编码，每个字符占用两个字节（称码点），所以字符类型数据可以转换为整数型数据
- Unicode 编码的字符不光包含了 ASCII 字符，还包含了很多外语的字母，如汉字等
- 允许使用反斜杠来表示具有特殊意义的字符

```
1 char a = 'x';
2 char b = "x"; // 错误，双引号括起来的表示字符串
3 char c = 'abc'; // 错误，字符型数据只能是 1 个字符
4 char d = '\n'; // 表示一个换行符
5 char e = '1'; // e 是一个字符型数据，不是一个整数!
6 char f = '中'; // Java 中的字是 Unicode 编码的字符，包括中文
7 char f = '\u0061'; // 采用 Unicode 的形式给字符赋值
```

数字型



字符和字符串的区别

- 字符只能包含一个字母，而字符串是一个或多个字母的有序集合
- 字符使用单引号，而字符串使用双引号
- **char**是一个关键字，而字符串**String**是一个类
- 字符可以通过强制类型转换变为与之对应的数字，而字符串不可以
- 字符串不等同于字符数组！

- 数组是多个相同类型数据的集合
- 和 C 语言中的数组类似，Java 中的数组本质上也是一个指针，属于 Java 的引用类型，做为函数参数是进行传址调用
- 声明方式（建议使用第一种方法，更为明了）：

`type[] valName`

或

`type valName[]`

- 声明数组时不能指定其元素个数！只有创建时可以指定元素个数！

数组的创建和初始化

```
1 //数组的创建
2 int[] arr1 = new int[5];
3
4 //数组的动态初始化
5 for (int i=0;i<5;i++) {
6     arr1[i] = i;
7 }
8
9 //数组的静态初始化
10 int[] arr2 = {1, 2, 3};
```

- 获得数组的元素个数:

`arr.length`

- 获得数组指定位置的元素:

`arr[i]`

- 数组从 0 开始索引，获得数组的最后一个元素:

`arr[arr.length - 1]`

- 如果数组的长度为 L ，则当访问不在 $[0, L]$ 范围内的元素时，会出现`IndexOutOfBoundsException`错误（数组索引越界错误）

- 访问未被赋值的元素时，返回`null`

- 关键字**null**表示“空”的概念
- 访问**null**时，会报空针错误**NullPointerException**
- **null**不等于**false**，请使用**a == null**判断一个变量是否为空！

```
1 int[] a = {1, 2, 3};  
2 //请问下行会报错么?  
3 int x = a[3];  
4  
5 int[] b = new int[3];  
6 b[0] = 1;  
7 //请问下行会报错么?  
8 int y = b[1] + 1;
```

读取终端用户的输入参数

- Java 将用户输入的参数保存在 main 方法的 args 参数中
- args 参数是一个字符串数组，每个元素是一个用户输入的参数

```
1 public static void main(String[] args) {  
2  
3 }
```

二维数组

- 可以看出组成元素为数组的数组
- 声明和初始化应当遵循从高维到低维的顺序

```
1 int[][] a = new int[3][];
2 int[][] b = {{1,2}, {3, 4}};
3 int[][] c = new int[] [3]; //非法!
```

实验一：累加器

读取终端输入的数字并求和

第四章：运算符、表达式和语句

- 算术运算符: +、 -、 *、 /、 %、 ++、 --
- 关系运算符: >、 >=、 <、 <=、 ==、 !=
- 逻辑运算符: !、 &&、 ||
- 位运算符: |, &, ...
- 赋值运算符: =
- 扩展赋值运算符: +=、 -=、 *=、 /=
- 字符串链接符: +

算术运算符

```
1 int m = 6 % 5; //为 1, 因为余数为 1
2 int m = 6 % 2; //为 0, 因为可以整除
3
4 int x = 2, y = 2;
5 int b = x++; //先赋值, 再自增, b=2。计算顺序: b=x, x=x+1
6 int c = ++y; //先自增, 再赋值, c=3。计算顺序: y=y+1, c=y
7
8 3/2; //结果为 1!
9
10 //Java 没有“幂运算符”, 通过下述函数完成幂运算
11 Math.pow(3, 2); //3 的 2 次方
```

- 短路现象

- 三目运算语法格式：

$x ? y : z$

- 解释：其中 x 为布尔型类型的表达式，如果 x 为真，则返回 y ；否则，返回 z

```
1 int score = 80;
2 String type = score > 60 ? "及格" : "不及格";
3
4 int x = -1;
5 int flag = x > 0 ? 1 : (x == 0 ? 0 : -1)
```

- 条件语句：根据不同条件，执行不同的代码块

if

if ... else ...

if ... else if ... else ...

switch ... case ... case ... default ...

- 循环语句：当满足某条件时，重复执行代码块

for

while

do ... while ...

条件语句

IF 语句

```
1 if (x) {  
2  
3 }  
4  
5 if (x) {  
6 } else {  
7 }  
8  
9 }  
10  
11 if (x) {  
12 } else if (y) {  
13 } else {  
14  
15 }  
16  
17 }
```

Switch 语句

```
switch(参数) {  
    //如果“参数”的值等于“常量表达式 1”，则执  
    → 行  
    case 常量表达式 1:  
        ...  
        break;  
    //不要忘记加 break，否则会有副作用  
    case 常量表达式 2:  
        ...  
        break;  
    default:  
        break;  
}
```

- 小心 case 穿透，注意加上必要的 break
- default 可以省略，单不推荐

FOR 循环语句

传统形式

```
1 int[] arr = new int[]{1, 2, 3, 4};  
2  
3 for (int i=0; i< arr.length; i++) {  
4     System.out.println(x);  
5 }
```

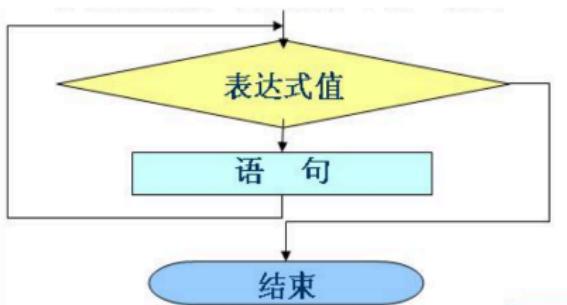
Java 5.0 增强形式

```
1 for (int x in arr) {  
2     System.out.println(x);  
3 }
```

WHILE 循环

WHILE 语句

```
1 while (逻辑表达式 1) {  
2     代码块 1  
3 }  
4 //最后没有分号
```

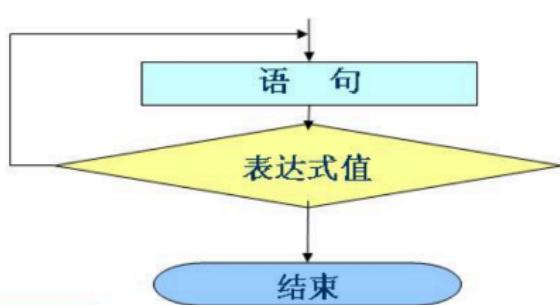


当不满足“逻辑表达式 1”时，“代码块 1”一次都不会执行！

如果没有特殊需求，请用第一种 WHILE 语句！因为第一种 WHILE 语句意思更明了，更易理解，更加不容易出现副作用！

DO WHILE 语句

```
1 do {  
2     代码块 2  
3 } while (逻辑表达式 2);  
4 //不要忘记最后有分号
```



当不满足“逻辑表达式 2”时，“代码块 2”会执行 1 次！

循环控制：break 语句和 continue 语句

break 语句用于提前终止循环体的执行，可以强制退出循环，并且不再进行下一轮循环的判定和执行

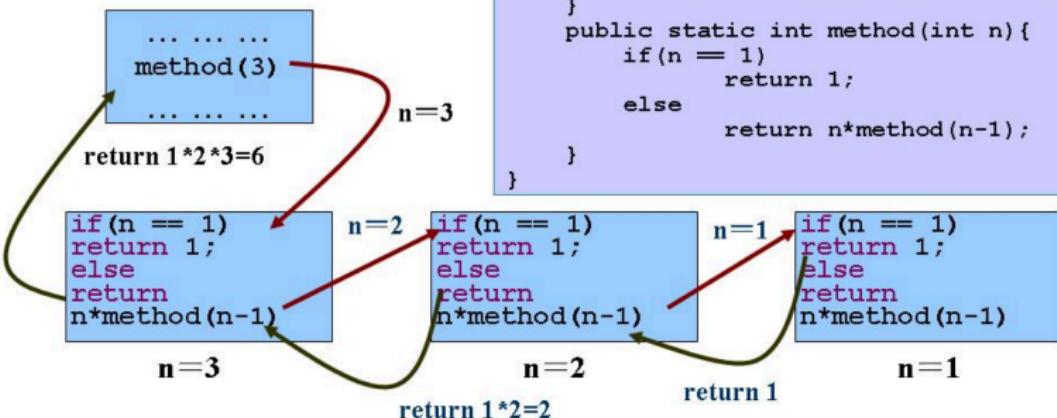
continue 语句可以提前终止本次循环体的执行，不再执行该语句后面的循环体，开始下一轮循环的判定和执行

```
1 for (int i=0; i< arr.length;  
2   → i++) { //arr 是 [1,2,3,4]  
3     if (i == 3) {  
4       break;//跳出本循环  
5     }  
6     System.out.println(x);  
7 }  
//只会打印“1, 2”!
```

```
1 for (int i=0; i< arr.length;  
2   → i++) { //arr 是 [1,2,3,4]  
3     if (i == 3) {  
4       continue;//跳出本轮循环  
5     }  
6     System.out.println(x);  
7 }  
//会打印“1, 2, 4”，不会打印“3”!
```

- 相同点：立即退出本轮循环的循环体，后续循环体的语句都不执行
- 不同点：**break** 语句不再进入下一轮的循环判定，而 **continue** 语句会进入下一轮的循环判定

迭代 (recursion)



//分析下面程序的运行结果。

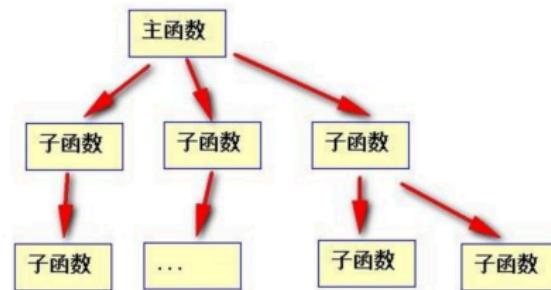
```
public class Test {  
    public static void main(String arg[]){  
        System.out.println(method(5));  
    }  
    public static int method(int n){  
        if(n == 1)  
            return 1;  
        else  
            return n*method(n-1);  
    }  
}
```

第三部分：面向对象

第五章：面向对象编程

- 粗略的说，程序 = 数据结构 + 算法：
 - 数据结构：数据是如何在计算机中存储的
 - 算法：如何操作数据结构
- “编程思想”要解决的问题：在软件编程中，如何将真实世界中的复杂事物刻画映射为计算机程序中可编程的数据结构和算法
- 目前最为常用的两类编程思想：面向过程 和 面向对象

- 面向过程的程序设计把计算机程序视为一系列的命令集合，即一组函数的顺序执行。“过程”就是函数！
- “面向过程编程”的基本思路：把事物设计为函数，然后把大块函数通过切割成小块函数来降低系统的复杂度。
- 程序的设计、编写和运行，本质上都是采用了一种线性的思维
- 大家以前学习的 C 语言就是一门最常见的“面向过程”的编程语言。C 语言的入口为主函数 `main`，在该函数中调用其它用户设计实现的程序子函数



“面向对象编程”(OOP, Object Oriented Programming)
的基本思路：

将真实世界中的复杂事物抽象、提取、归纳为“对象”

举例——历史学家要编写史书，主要有两种形式：

- 纪传史：以人物为中心编写，可以类比为“面向对象”
- 编年史：按照事件发生的事件顺序编写，可以类比为“面向过程”

- OOP 把对象作为程序的基本单元，一个对象包含了：
 - 属性：即数据，描述了“该对象有什么数据”
 - 方法：即操作数据的函数，描述了“该对象能干什么”
- 对象 = 属性 (attribute)+ 方法 (method)
- 例子一：将“人”这个概念抽象为“对象”
 - 属性：姓名、年龄、性别...
 - 方法：吃饭、睡觉、走路...
- 例子二：将“汽车”这个概念抽象为“对象”
 - 属性：发动机、底盘、车身、油箱...
 - 方法：启动、行车、刹车、加油...

问题：人把大象装冰箱

面向过程的思路

程序入口主函数：

- ① 函数一：打开冰箱
- ② 函数二：把大象塞进去
- ③ 函数三：关冰箱门

面向对象的思路

设计对象（只定义了“方法”）：

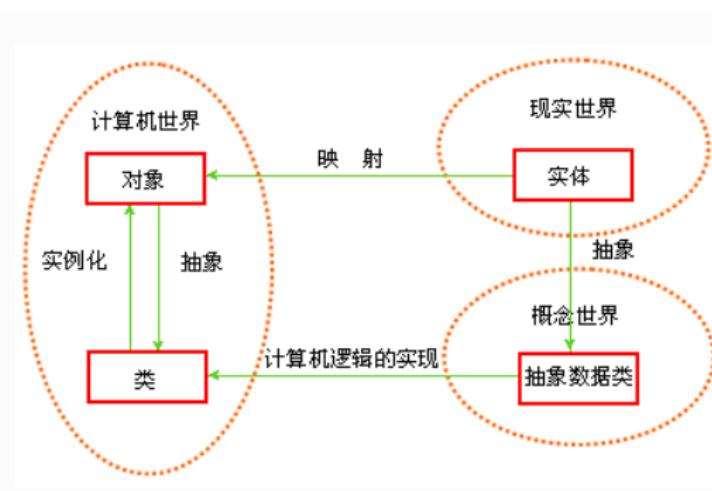
- 人：操作(冰箱), 操作(大象)
- 大象：进入()
- 冰箱：开门(), 关门()

程序入口主函数：

- ① 人. 操作(冰箱): 冰箱. 开门()
- ② 人. 操作(大象): 大象. 进入()
- ③ 人. 操作(冰箱): 冰箱. 关门()

- “类”是对一类事物的总结，可以把“类”理解为：对大量对象共性的抽象，是对象的模板。“类”中定义了这一类对象的共有属性和共有方法
- “类”具有属性（也称成员变量），它是对象的属性状态的抽象，用数据结构来描述类的属性，人类有姓名、年龄、性别等属性
- “类”具有操作（也称方法），它是对象的操作行为的抽象，用操作名和实现该操作的方法来描述，人有吃()、说话()、学习() 等方法
- “类”的具体化就是对象，也可以说类的实例是对象。如有“人”这个类，统称人类；对象则是具体的某个人，如张三、李四

类的举例



- Java 是一门 OOP 语言¹，在 Java 中对象是第一公民，程序的设计和编写都是围绕着各种对象展开的
- 和 Java 相对的，C 语言这种面向过程的编程语言，函数往往是第一公民
- 面向对象的好处：易维护、易复用、易扩展
- 在 Java 中，定义在类中的函数一般称为“方法”

¹严格来说，Java 只是一门较为纯粹的 OOP 语言，仍然有一些不属于 OOP 的元素

第六章：类与对象

类的定义

```
1 //定义了类：采用驼峰标记
2 public class Person {
3     //定义成员变量：驼峰标记法
4     String id;
5     String name;
6     int age;
7
8     //定义构造方法
9     public Person(String fullName, int age) {
10         this.fullName = fullName;
11         this.age = age;
12     }
13
14     //定义方法：驼峰标记法
15     void speak(String something) {
16         System.out.println(something);
17     }
18
19 }
```

- 使用**class**关键字定义一个类：
-

```
1 class 类名 {  
2     类体  
3 } //注意，此处没有“;”
```

- **class** 后面的花括号括住的内容称为“类体”

- 代码风格要求：

- 类名采用大驼峰标记
 - 包围类体的左花括号写在行尾，前面和类名留一个空格，不要新起一行！
 - 包围类体的右花括号单独新起一行
-

```
1 class Test  
2 {//错误！左花括号应该在上一行的末尾  
3 }  
4 class test //错误！类名使用大驼峰标记
```

- 对象的属性也称对象的成员变量 (member variable, field)
- 成员变量是属于对象的，同一个类实例化出的不同对象虽然有着相同类型的成员变量，但是成员变量的赋值却可以不一样
- 也即，成员变量的值是绑定在对象上的，而不是类上的
- 成员变量的作用域为整个类体
- 定义成员变量的方法和定义变量的方法类似：

```
1 public class Test {  
2     int a; // 定义一个成员变量  
3     int b = 2; // 定义一个成员变量，并赋予默认值（也称为初始化）  
4     private String c; // 成员变量的前部可以加修饰符  
5 }
```

成员变量声明中的代码风格

- 一行只声明一个成员变量
- 如果要为成员变量赋予默认值，等号两边要各有一个空格！
- 成员变量的命名采用小驼峰标记法

```
1 public class Test {  
2     int a; int c;// 错误！一行只推荐定义一个成员变量  
3     int b=2; // 错误！等号两边需要有空格  
4     String MyName; // 错误！请使用小驼峰标记法  
5     String my_name; // 错误！请使用小驼峰标记法  
6 }
```

- 构造函数是定义在类体中，用于初始化对象的一种特殊函数，在创建类的对象时，会调用该类的构造函数
- 和类体中的一般方法不同，其函数名必须和类名相同，且无需声明返回值类型，也没有返回值（即不能使用**return**语句）
- 当类体中不定义任何构造函数时，Java 会为该类自动添加一个默认构造函数。默认构造函数本质是一个无参的、空函数体的构造函数
- 当类体中定义了任何一个构造函数时，Java 则不会再为该类添加一个默认构造函数
- 在类体中可以使用**this**关键字指代“当前对象本身”，通过**this**可以调用本对象的所有方法和属性（不管其控制访问修饰符为何）：**this.** [方法名或属性名]
- 使用**this**可以区分同名变量

构造函数举例

```
1 public class Person {  
2     public String name;  
3  
4     //相当于无参构造函数  
5     public Person(){  
6     }  
7  
8     //通过构造函数给成员变量进行初始化赋值  
9     public Person(String name) {  
10         //this.name 指成员变量 name, 等号右边的 name 为构造函数的形参  
11         this.name = name;  
12     }  
13  
14     public String getName() {  
15         return this.name;  
16     }  
17 }
```

- 类体中除了构造函数之外的函数称为“方法”
- 如果类的方法签名中定义了返回值类型，方法体中最后需要使用**return**返回
- 如果该方法不返回任何值，应该在方法签名中将函数返回值类型设置为**void**
- 代码风格：
 - 方法名采用小驼峰命名法
 - 同类定义中花括号的使用原则一样，方法体的左花括号写在方法签名行的末尾，右花括号新起一行顶头写
 - 多个形参之间使用逗号隔开时，逗号后面需要有一个空格：
int add(int x, int y)

定义方法举例

```
1 public class Person {  
2     public String name;  
3  
4     public String getName() { //有返回值  
5         return this.name;  
6     }  
7  
8     public void printName() { //无返回值  
9         System.out.println(this.name);  
10    }  
11  
12    public void setName(String name) {  
13        this.name = name; //直接使用形参  
14    }  
15}
```

对象的创建和使用

- 对象的创建也称为类的实例化
- 使用“**new** 构造函数([args])”的方式创建对象
- 使用对象.[方法名或属性名] 的方式访问该对象的方法或属性

```
1 public class Test {  
2     public static void main(String[] args) {  
3         Person p = new Person("Jim");  
4         System.out.println(p.name); //访问对象的属性  
5         System.out.println(p.getName()); //调用对象的方法  
6     }  
7 }
```

静态成员变量和静态方法

使用访问控制符来保护对类、变量、方法和构造方法的访问：

- **default**: 默认的访问控制符(即缺省，什么也不写)，表示在同一包内可见。使用对象：类、接口、变量、方法
- **private**: 在同一类内可见。使用对象：变量、方法。不能修饰类（外部类）
- **protected**: 对同一包内的类和所有子类可见。使用对象：变量、方法。不能修饰类（外部类）
- **public**: 对所有类可见。使用对象：类、接口、变量、方法

访问控制修饰符的应用举例

```
1 public class Person {  
2     private String name = "x";  
3     public int age = 21;  
4 }  
5 public class Test() {  
6     public static void main(String[] args) {  
7         Person p = new Person();  
8         p.name; //错误！无法从别地类中访问 private 修饰的属性和方法  
9         p.age; //正确  
10    }  
11 }
```

外部类与内部类

- 必须使用 new 关键字

- 使用 **new** 关键字创建类的一个实例对象，本质是在内存中分配一片内存，并将该内存的指针赋值给新创建的对象变量
- 除了基本类型外，所有 Java 中的对象，本质上都是指针
- 所以 Java 中方法的形参，如果是基本类型，则为传值调用，如果为对象，则为传址调用！

第七章：子类与继承

第八章：接口与实现

第九章：泛型

第四部分：实用功能

第十章：常用工具类

第十一章：容器类

- 列表： **List**， 数据的一维有序集合
- 集合： **Set**， 类似列表，但是集合容器内没有重复的元素
- 哈希表： **Map**， 表示键值对

实现了 List 的主要容器类：

- **ArrayList**: 内部使用数组实现，但是长度可变
- **LinkedList**: 内部使用链表结构实现，可以快速的在某一位置删除和插入元素

- **boolean add(E e)**: 添加元素
- **boolean add(int index, E e)**: 在指定位置添加元素
- **E set(int index, E e)**: 将指位置的元素替换为新元素
- **boolean get(int index)**: 取得指位置的元素
- **int indexOf(E e)**: 获得元素在容器里的位置
- **boolean remove(int index)**: 删除指位置的元素
- **boolean remove(E e)**: 删除制定元素
- **boolean removeAll(Collection c)**: 从本容器中删除 c 中指定的所有元素
- **void clear()**: 清空本容器
- **void toArray()**: 转换为数组

实现了 Map 的主要容器类：

- **HashMap**: 不保证键值对之间的有序性
- **LinkedHashMap**: 保证键值对可以保持插入顺序

- `V get(K key)`: 根据 `key` 取出一个与之对应的值
- `V put(K key, V value)`: 插入一个键值对
- `V remove()`: 根据 `key` 取出一个元素
- `boolean containsKey(K key)`: 移除容器内键为 `key` 对应的键值对
- `int size()`: 返回容器内键值对的数目
- `boolean isEmpty()`: 返回容器是否为空
- `void putAll(Map map)`: 将参数 `map` 中的所有键值对插入本容器
- `Set<Map.Entry> entrySet()`: 以 `Set` 的方式取出容器内所有的键值对，方便遍历

HashMap

LinkedHashMap

定义为包含了不重复的元素集合，主要实现类为 `HashSet`，主要方法包括：

- `boolean add(E e):`
- `boolean contains(E e):`
- `boolean isEmpty():`
- `boolean remove(E e):`
- `boolean removeAll(Collection<? super E> collection):`
- `void clear():`
- `void toArray():`

第十二章：输入输出流

第五部分：高级功能

第十三章：多线程技术

第十四章：基于 Socket 的网络编程

第十五章：反射机制