



武汉大学
WUHAN UNIVERSITY

面向对象程序设计

第 2 讲 Java 语法规语

法 -1

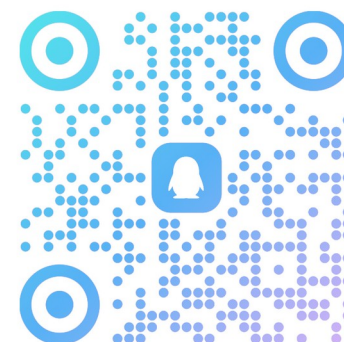
刘进

2230652597@qq.com

OOP 教辅 2025 秋季上午 QQ 群 :



OOP教辅2024秋季...
群号: 837966056



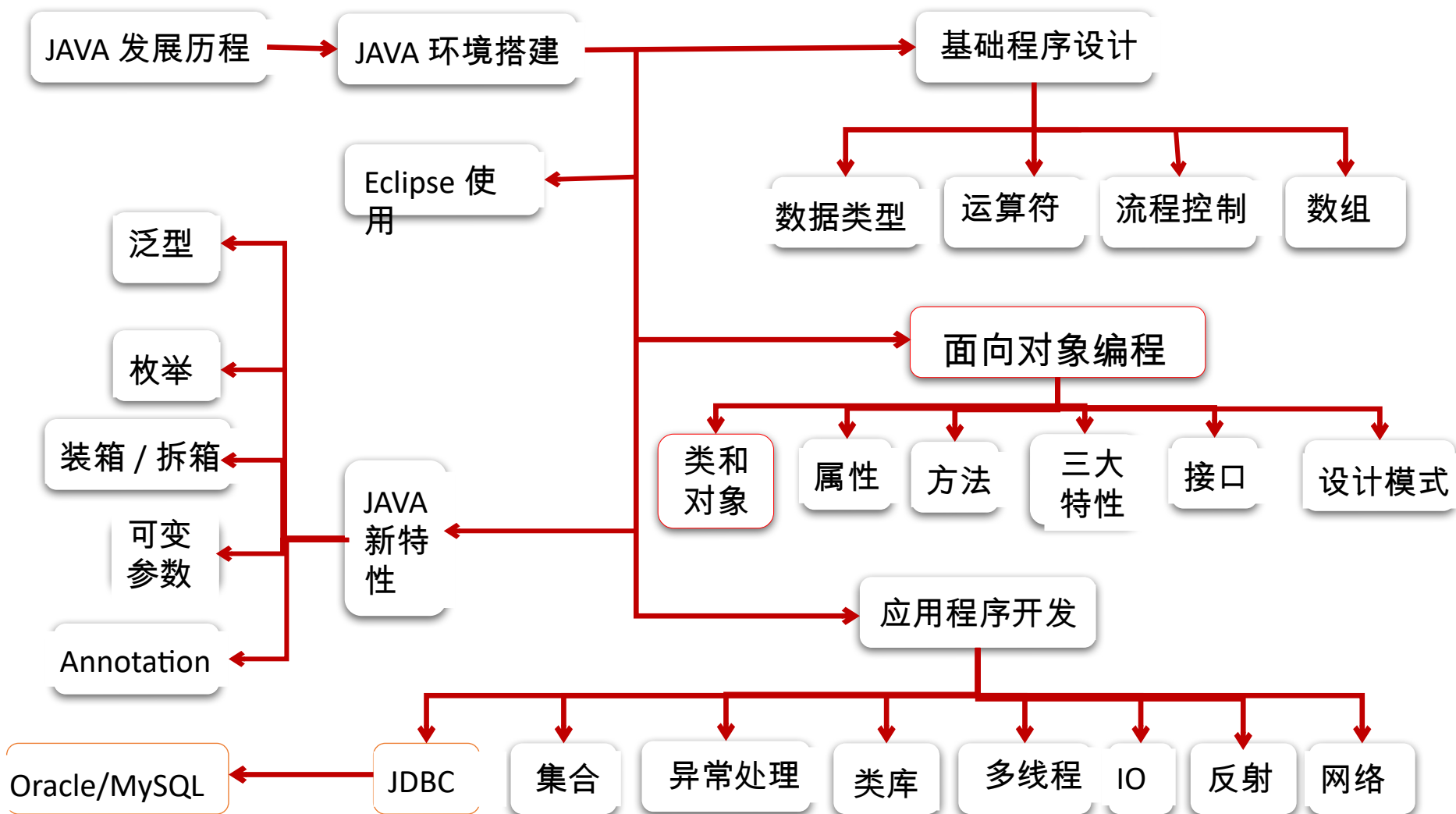
扫一扫二维码，加入群聊



此间有山水

Java 基础知识图解

主要知识点



软件危机与面向对象

保留字与
升级预留

软件危机是指在计算机软件开发与维护过程中所遇到的一系列严重问题，主要出现在 1960s-1970s 。

本质原因：

软件规模越来越大，复杂度急剧升高，而当时的结构化程序设计方法难以管理这种复杂性。

核心表现是：

- ✓ 开发成本失控、工期严重超期
- ✓ 开发出的软件质量低劣、漏洞百出
- ✓ 软件难以维护和升级

01/08/2026

面向对象程序设计 是一种全新的软件开发范型，被认为是解决软件危机的重要途径之一。

它通过以下思想来**降低软件复杂度，提高可维护性和复用性**：

封装：将数据和对数据的操作捆绑在一起，隐藏内部细节，只暴露接口。这降低了系统的耦合度。

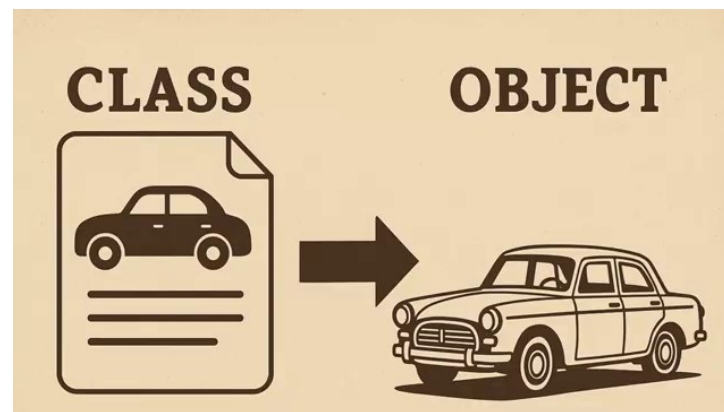
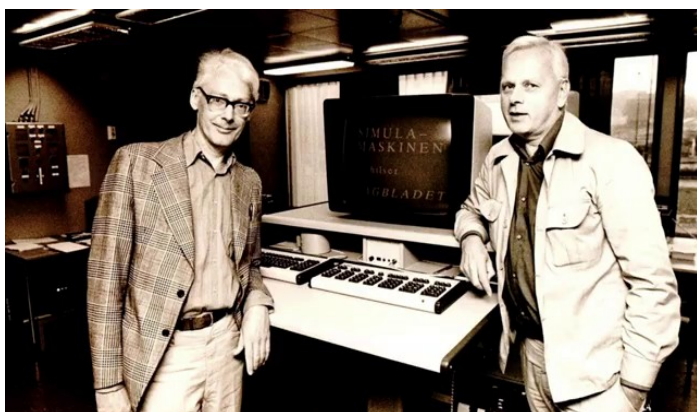
继承：可以扩展已有的代码，实现代码复用，并建立清晰的层次关系。

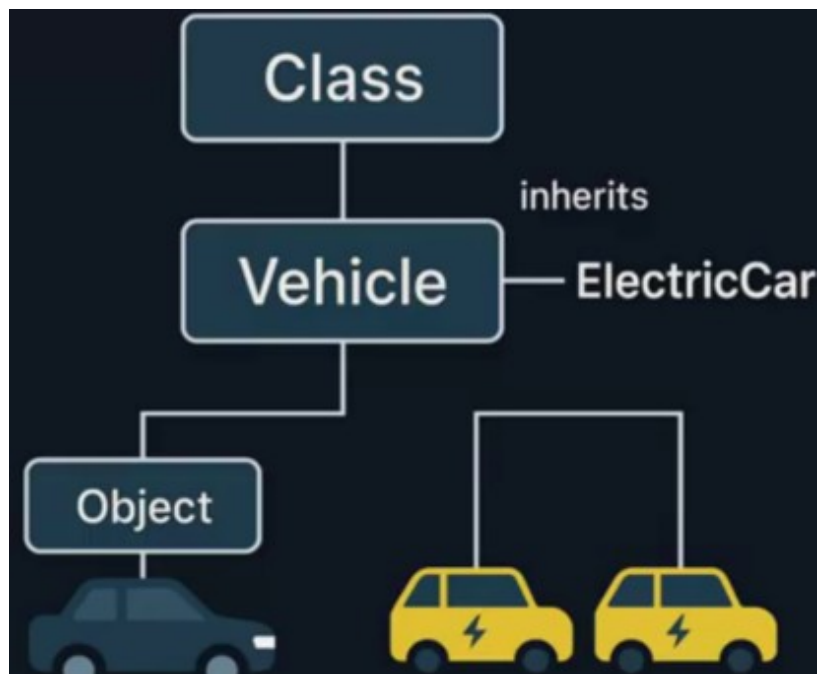
多态：允许不同类的对象对同一消息做出不同响应，提高了代码的灵活性和可扩展性。

OOP 让程序员能够以更符合人类思维的方式（对象和类）来设计和构建大型软件，从而更有效地管理复杂性。



软件危机（**问题** **Systemv360 Model 65 的操作台**）——→ 催生了新的编程范型如面向对象程序设计（**解决方案** **软件复杂性 / 1960 年代初挪威奥斯陆，克里斯汀尼加德奥勒、约翰达尔在思考一个问题：如何让计算机不只是计算数字，而是模拟现实世界的运？** **class 类**）——→ → Java 是践行并推广这一解决方案的最成功的编程语言之一

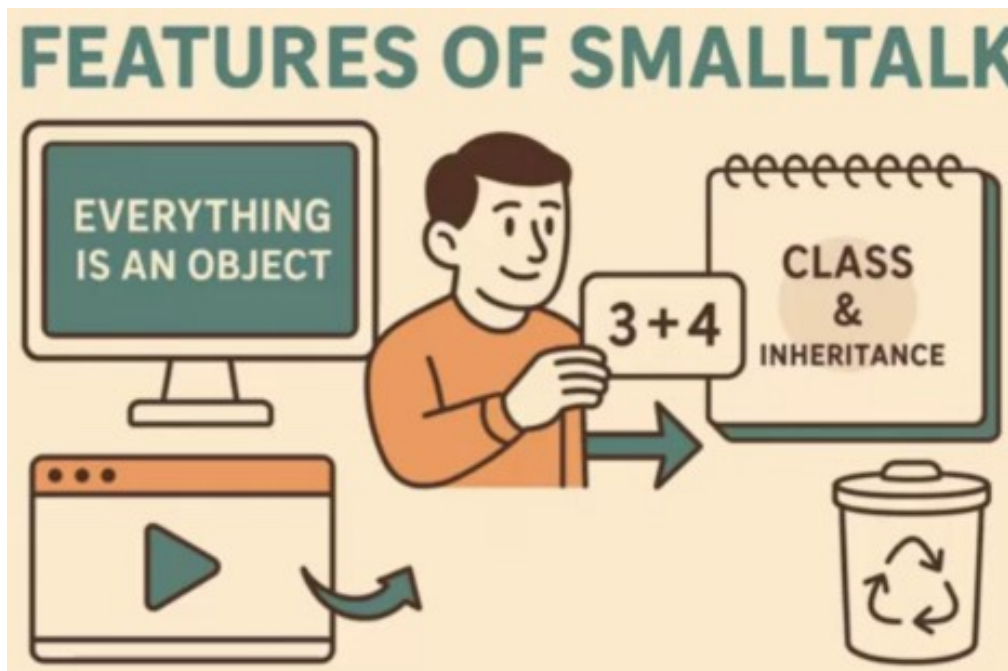




1970 年代加州施乐研究中心艾伦凯，一切皆对象，small talk 诞生 + 一个完整的交互环境。
计算机应该是每个人的创造工具，而不是冰冷的机器。

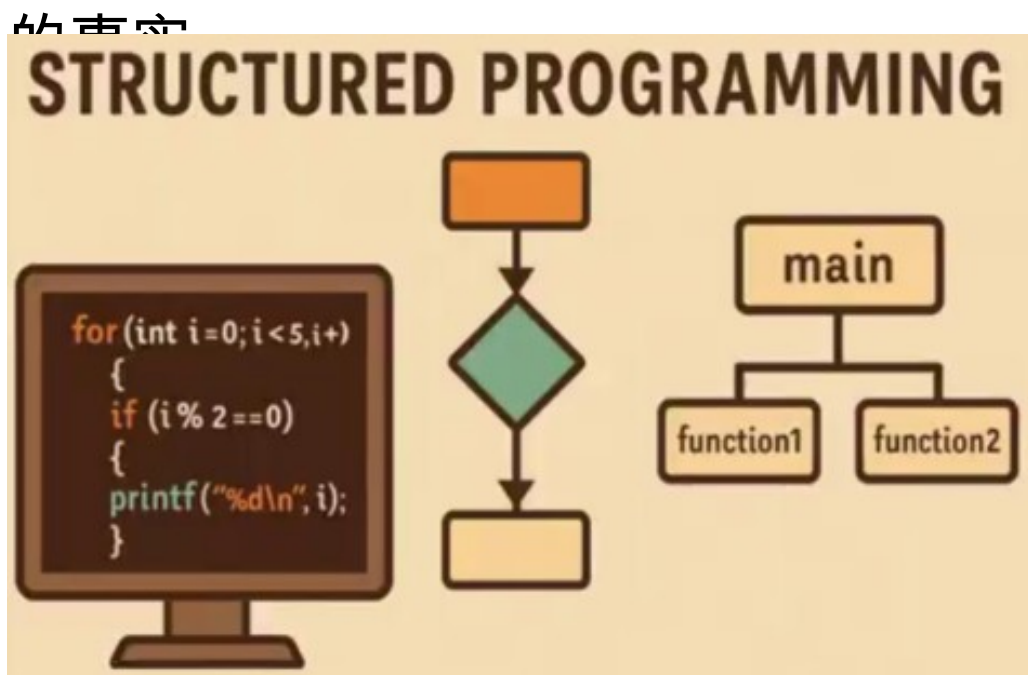
```
begin
  class Car(initialColor);
  text initialColor, color;
  integer speed;
  begin
    speed := 0;
    color := initialColor;
  end;

  Car car1, car2;
  car1 := new Car("red");
  car2 := new Car("blue");
end;
```

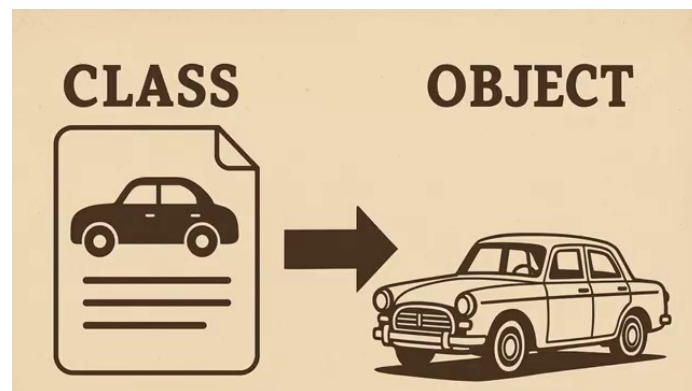


然而，理想主义也有局限，smart k 运行缓慢，硬件昂贵。没能进入主流，却留下了深远的影响

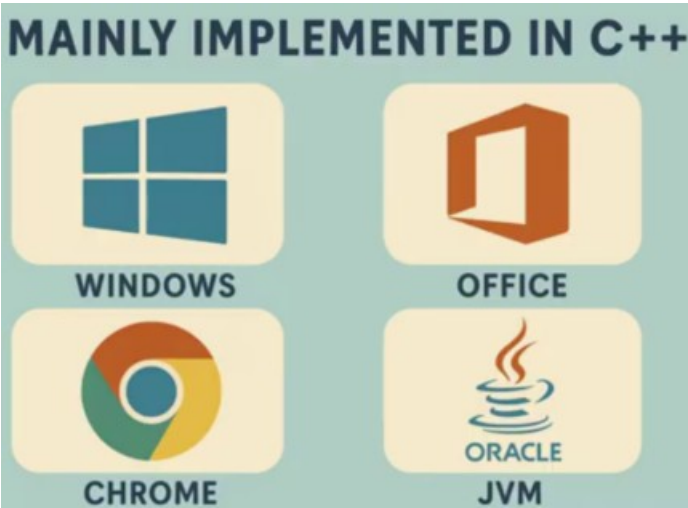
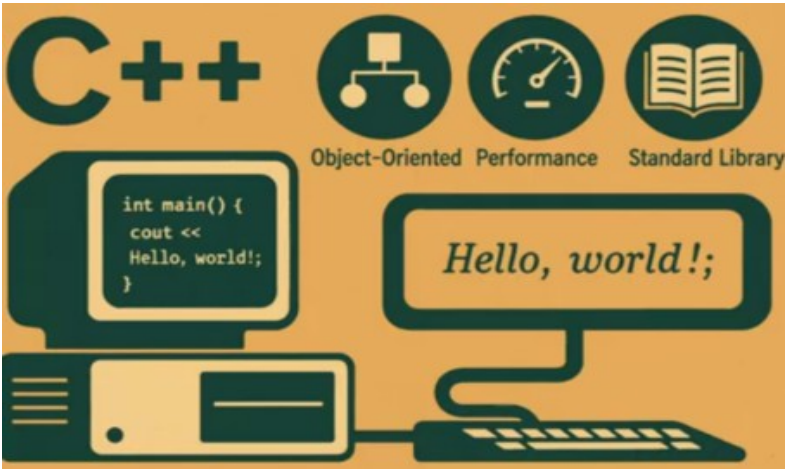
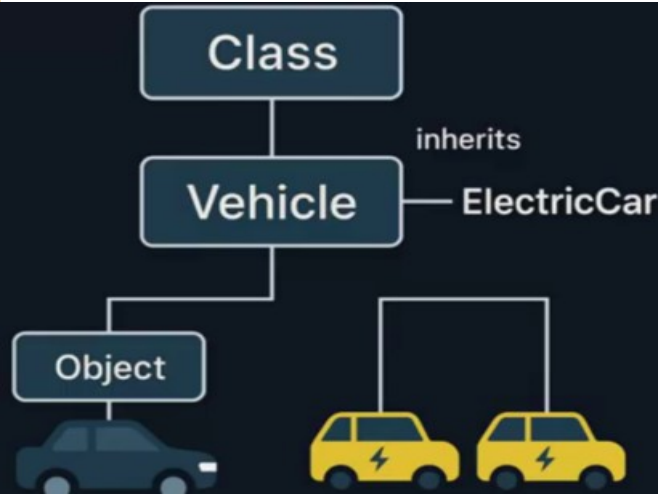
与 small talk 崭露头角的同时，另一股力量也在主导的软件世界。结构化编程、pascal、c 等语言强调清晰的控制流函数与模块化，让程序更可读，也更可靠。它们确实比早期混乱代码更好，但依然存在局限，函数彼此割裂，难以表达世界是由对象组成的事实。



进入 1980 年代，软件复杂度持续攀升，等系统越来越庞大，模块越来越多，内存管理、并发控制等问题接踵而至。开发者不仅要编写逻辑，还要与底层资源反复较量，结构化编程虽是主流，却已显得力不从心。



C++ 核心主张是**零成本抽象**，抽象不应该带来性能损耗，为此它引入了 RIRI 对象的生命周期及资源管理周期，**STL 强类型零开销的模板库**。这些特性让 C 加加成为构建高性能、大规模规模系统的利器。



90 年代初，C 加加已在企业流行，却在复杂现在内存隐患面前显得笨重

这是迄今为止我所
知道的最容易的
21 天精通 C++

C++ 不是速成的语言，而是一门需要
终身修行的语言，
即便如此，C++ 依
然是工业界的事实
标准。

第1-10天

学习变量，常量，数组，字符串，表达式，语句，函数……



第11-21天

学习程序流程，指针，引用，类，对象，接口，多态……



第22-697天（两年）

进行大量的“休闲娱乐”方式的编程，并在Hack代码中找到乐趣，从错误中学习总结。



第698-3648天（八年）

与其它程序员互相影响，并一同开发项目，并向他们学习。



第3649-7781天（十年）

学习高等理论物理，使用公式证明量子物理理论。



第7782-14611天（二十年）

学习生物化学，分子生物学，以及遗传学。



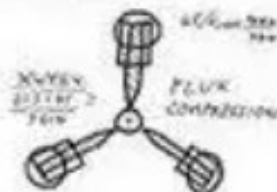
第14611天
（自学习编程来的四十年）

使用在生物学方面的知识造一个“返老还童”药剂。



第14611天

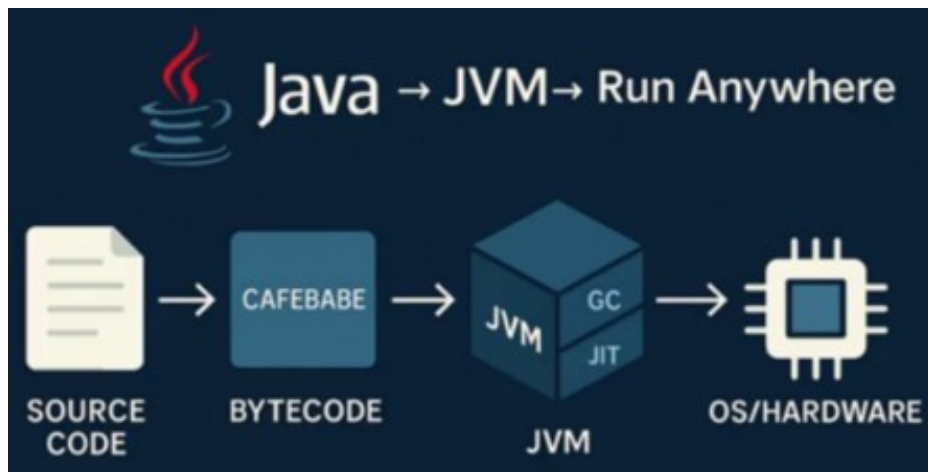
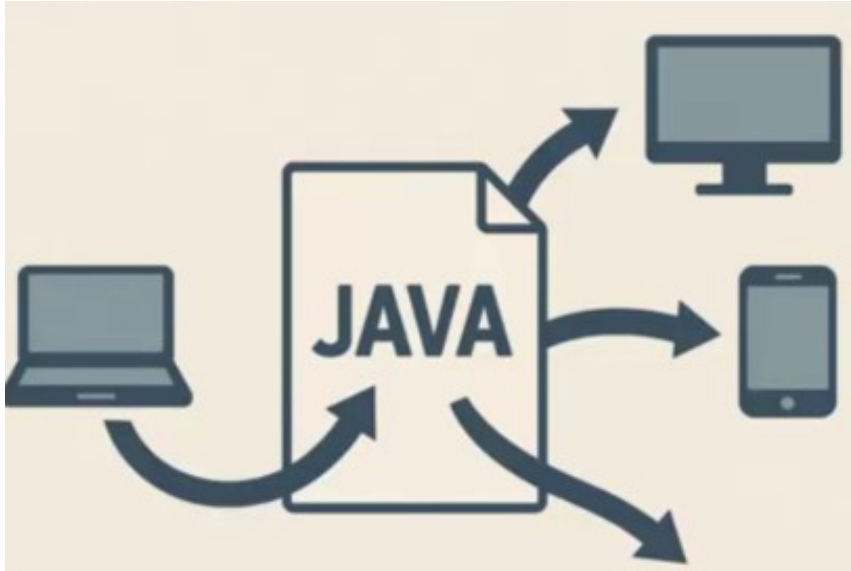
使用在物理学上的知识造一个通量电容器，把自己传送回学编程的第21天。



第21天

把那时的自己给替换了。





Java 的最大突破在于 **JVM Java 虚拟机** 与直接编译成机器码的 C 加加不同，java 的程序被编译成字节码 com 在不同平台上解释或即时编译运行。**J2EE**，一整套企业应用标准、数据库、访问事务管理、安全认证、分布式组件全都被纳入统一

规范

Java 极大地推广和证实了面向对象 OOP 的思想：

- **设计理念**：Java 的几乎所有代码都必须写在类中，强制程序员使用面向对象的方式思考。
- **内置支持**：对封装、继承、多态提供了完整且语法清晰的支持。
- **生态系统**：Java 强大的标准库和第三方库都是基于 OOP 构建的，提供了大量可复用的组件，这直接应对了软件危机中“重复造轮子”和难以复用的问题。

社区驱动的框架时代



Software 1.0 : 【传统编程】

特点：程序员用代码一步一步写清楚“规则”。

方式：人类先思考逻辑，再把规则写成 if-else、for 循环、函数调用。

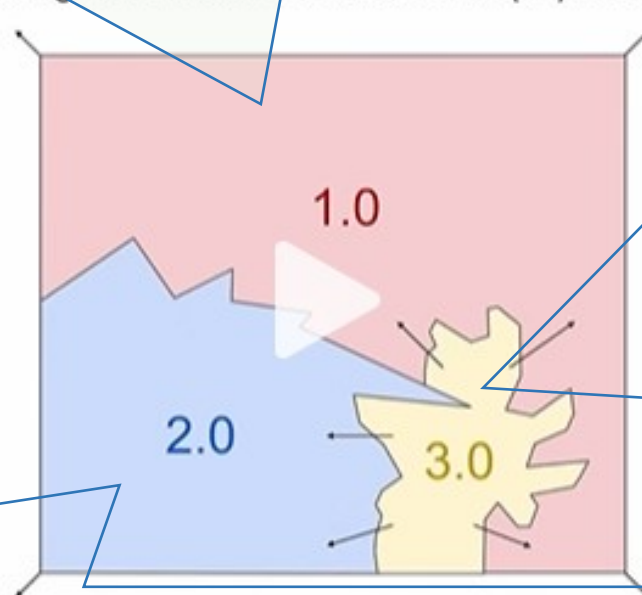
例子：写一个排序程序时，程序员明确写下比较和交换的步骤。

本质：规则是人写的，计算机只执行。



- 各有优缺点
- 在 3 种范式中流畅转换

A huge amount of Software will be (re-)written.



Software 3.0 : 【大模型编程】

(知识在大模型里，大模型编程，靠 Prompt 和微调)

特点：程序员不需要再自己准备专门的数据和训练，而是直接利用预训练的大模型（如 GPT、LLaMA）。

方式：通过提示词（prompt）、指令调优（fine-tune）、上下文数据，让大模型完成任务。

例子：想写一个文章摘要工具，不需要收集几百万篇文章再训练模型，而是直接调用 GPT，并写提示词：“请总结以下文章内容”。

本质：规则和知识都在大模型中，人主要做“提示工程”和“轻量适配”。

Software 2.0 : 【机器学习编程】（规则由机器学，机器学习，靠数据训练）

特点：程序员不再手写规则，而是定义一个模型架构和训练目标。

方式：把数据喂给神经网络，靠优化算法“学出”规则。

例子：图像识别，程序员不会写“眼睛 = 两个黑点，鼻子 = 中间三角形”，而是给模型成千上万张照片，它自动学会“这是人脸”。

本质：规则是机器学的，人写的是“框架”和“目标函数”。

本章介绍 Java 的基本编程知识，例如：变量和数据类型，运算符，表达式，控制流程，以及其他的特点。

语言设计

Java 语言抛弃了 C、C++ 中不合理的内容，主要有如下几点：

- (1) 全局变量：Java 程序中，不能在类之外定义全局变量，只能通过在一个类中定义公用、静态的变量来实现一个类中的全局变量。
- (2) goto 语句：Java 不支持 goto 语句，而是通过异常处理语句 try、catch、final 等来处理遇到错误时跳转的情况。
- (3) 指针：Java 语言不支持指针的操作，但通过引用等特性实现了指针的功能及灵活性。
- (4) 内存管理：Java 语言系统能创建并动态维护数据结构所需的内存，并自动完成内存垃圾的收集工作。

语法、语义、语用

理念（与上相关）

第 2 讲 Java 语法规 法 -1

目录

2.1 关键字、保留字、标识符

2.2 变量和数据类型

2.3 运算符

2.4 数据类型转换

关键字

●关键字的定义和特点

- 定义：被 Java 语言赋予了特殊含义，用做专门用途的字符串（单词）
- 特点：关键字中所有字母都为小写

关键字

用于定义数据类型的关键字

class	interface	enum	byte	short
int	long	float	double	char
boolean	void			

用于定义数据类型值的关键字

true	false	null		
------	-------	------	--	--

用于定义流程控制的关键字

if	else	switch	case	default
while	do	for	break	continue
return				

关键字

用于定义访问权限修饰符的关键字				
private	protected	public		
用于定义类，函数，变量修饰符的关键字				
abstract	final	static	synchronized	
用于定义类与类之间关系的关键字				
extends	implements			
用于定义建立实例及引用实例，判断实例的关键字				
new	this	super	instanceof	
用于异常处理的关键字				
try	catch	finally	throw	throws
用于包的关键字				
package	import			
其他修饰符关键字				
native	strictfp	transient	volatile	assert

保留字

保留字与
升级预留

- Java 保留字：现有 Java 版本尚未使用，但以后版本可能会作为关键字使用的字符串。自己命名标识符时要避免使用这些保留字
byValue 、 cast 、 future 、 generic 、 inner 、
operator 、 outer 、 rest 、 var 、 goto 、 const

标识符

- 标识符：

- 对使用的各种**变量**、**方法**和**类**等要素进行标识时使用的字符序列称为标识符

- 合法标识符定义规则：

- 由 26 个英文字母大小写，数字 (0-9)，下划线“_”和“\$” 符号组成

- 数字不可以开头。

- Java 中**严格区分字母大小写**，长度无限制。

- 不可以使用关键字和保留字作为标识符

例如：username、\$My_Java、Age、java、_privateValue 都是合法标识符；

2022year、user#name 都不是合法标识符。

- **核心规则：在起名字时，为了提高阅读性，要尽量有意义，“见名知意”。**

标识符
望文知义

Java 中的名称命名规范

- Java 中的名称命名规范：

- **包名**：多单词组成时所有字母都小写：xxxyyyzzz
- **类名、接口名**：多单词组成时，所有单词的首字母大写：XxxYyyZzz
- **变量名、方法名**：多单词组成时，第一个单词首字母小写，第二个单词开始每个单词首字母大写：xxxYyyZzz
- **常量名**：所有字母都大写。多单词时每个单词用下划线连接：
XXX_YYY_ZZZ

- 用处：从名称可以防止引用错误，如对常量再次赋值等

标识符
有风格 Style 和规
律

程序书写规范

标识符 规范

- 包、类、变量、方法等命名：要体现各自的含义。
 - 包名全部小写，io，awt
 - 类名第一个字母要大写，HelloWorldApp
 - 变量名第一个字母要小写，userName
 - 方法名第一个字母要小写，setName
- 程序书写格式：保证良好的可读性，使程序一目了然。
 - 大括号 { } 的使用与对齐
 - 同层语句段的对齐
 - 在语句段之间适当空行
- 程序注释：帮助了解程序的功能。

类注释

变量注释

语句段注释

方法注释

语句注释

//PersonalInfo.Java

package ch001homework;

public class PersonalInfo {

public static void main(String args[]) {

 System.out.println(*"My name is Cooper."*);

 System.out.println(*"Sex: Man"*);

 System.out.println(*"I was born in Wuhan."*);

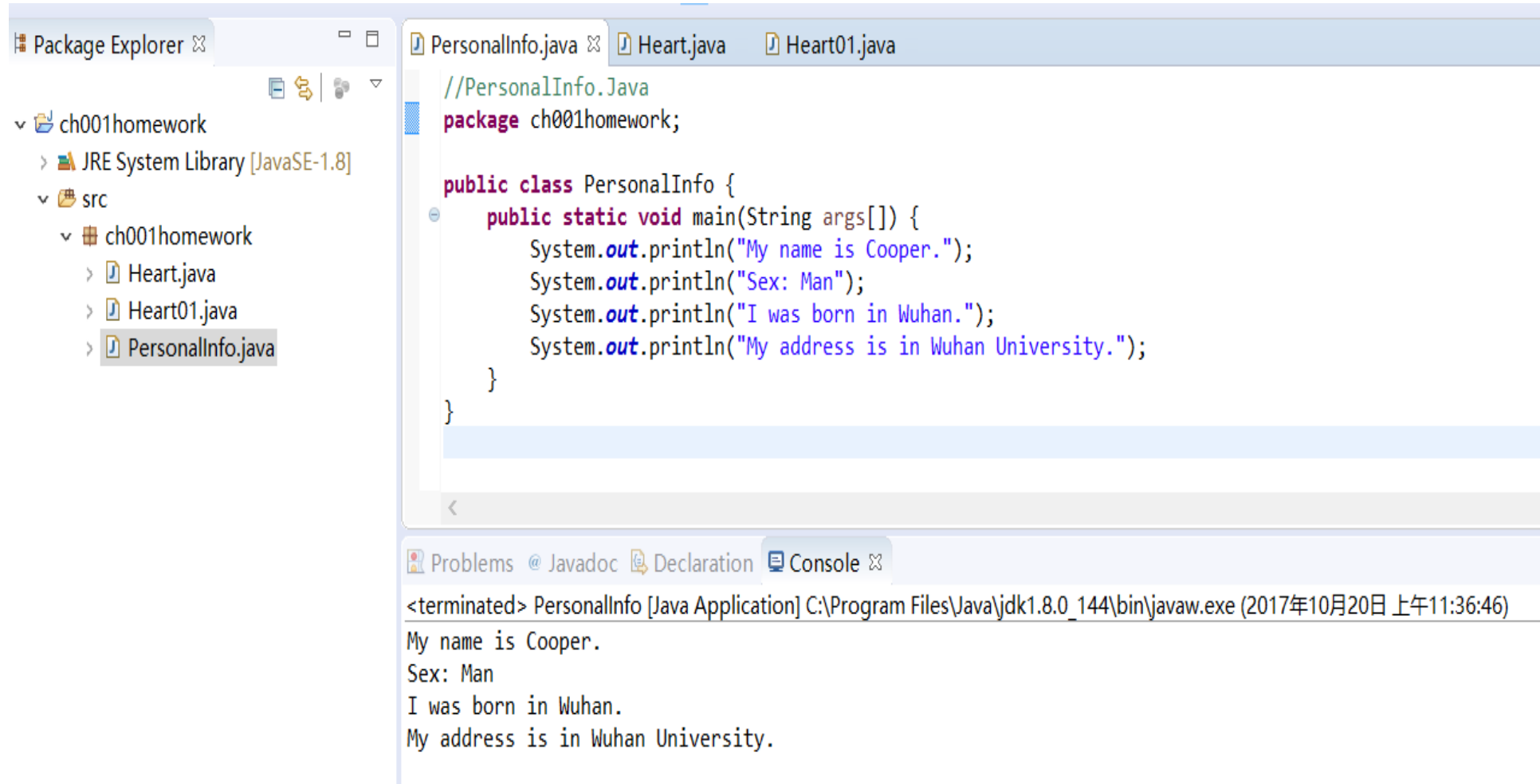
 System.out.println(*"My address is in Wuhan University."*);

 }

}

标识符
规范

标识符 规范



第 2 讲 Java 语法基础 -1

2.1 关键字、保留字、标识符

2.2 变量和数据类型

2.3 运算符

2.4 数据类型转换

变 量

●变量的概念：

- 内存中的一个存储区域
- 该区域有自己的名称（变量名）和类型（数据类型）
- Java 中每个变量必须先声明，后使用
- 该区域的数据可以在同一类型范围内不断变化

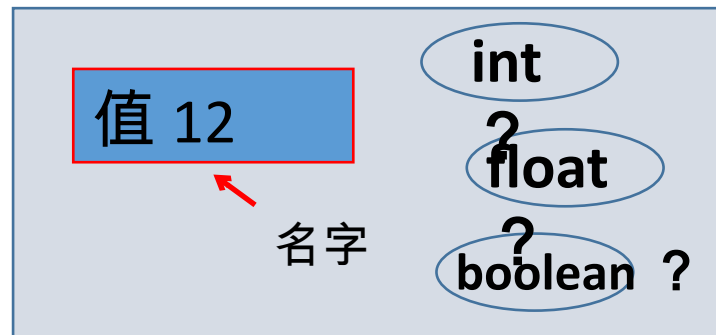
●使用变量注意：

- 变量的作用域：一对 {} 之间有效
- 初始化值

●定义变量的格式：数据类型 变量名 = 初始化值

●变量是通过使用变量名来访问这块存储区域

变量
大致的存储方式



变量	地址	存储的内容
b	ffc1	0000 1010
	ffc2	0000 0000
	ffc3	0000 0000
	ffc4	0000 0000
a	ffc5	0100 0001

- 变量是用标识符命名的数据项，是程序运行过程中其值可以改变的量。
- 在程序中使用的每一个变量必须提供一个名字。
- Java 是**强类型语言**，这就意味着每一个变量都必须有一个数据类型。
为了描述一个变量的类型和名字，必须用如下方式编写变量声明：

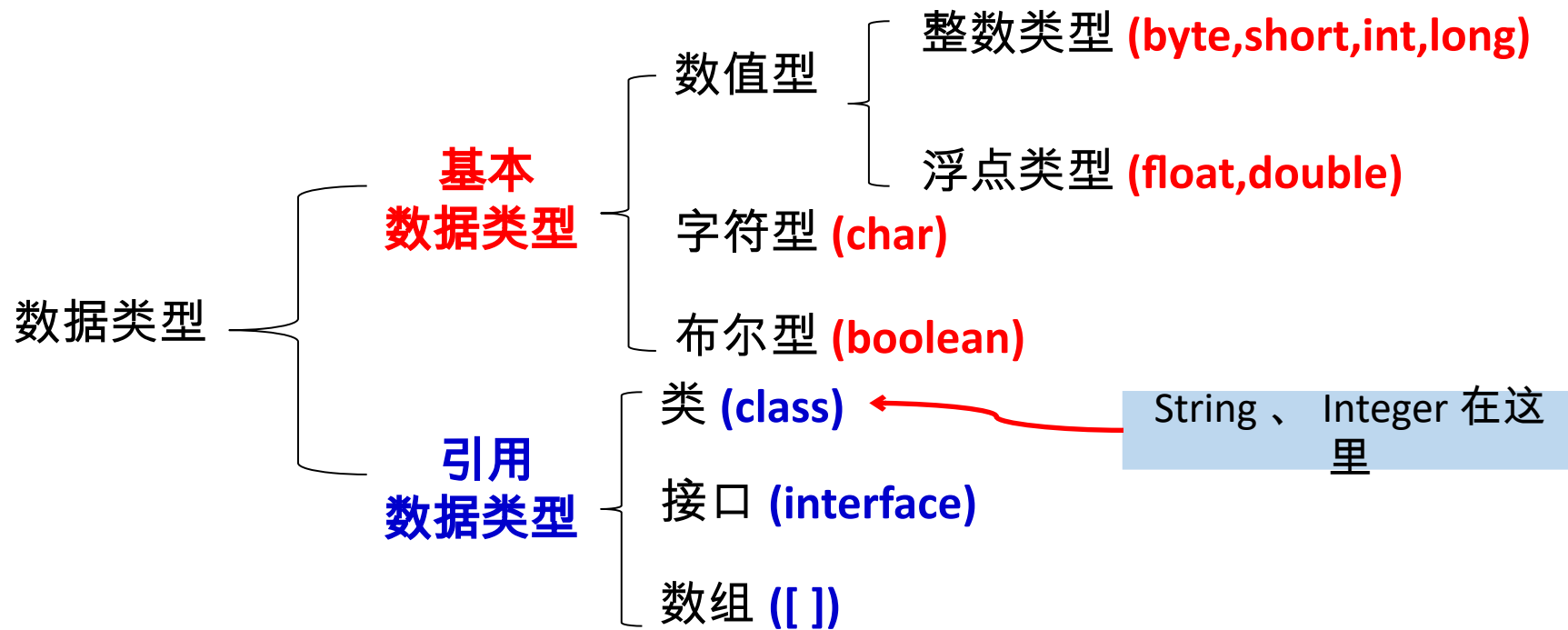
类型 **变量名**；

变量
同前

- 使用变量之前必须先声明变量。
- 声明变量包括两项内容：变量名和变量的类型。通过变量名可使用变量包含的数据。变量的类型决定了它可以容纳什么类型的数值以及可以对它进行什么样的操作。
- 变量声明的位置，决定了该变量的**作用域**。

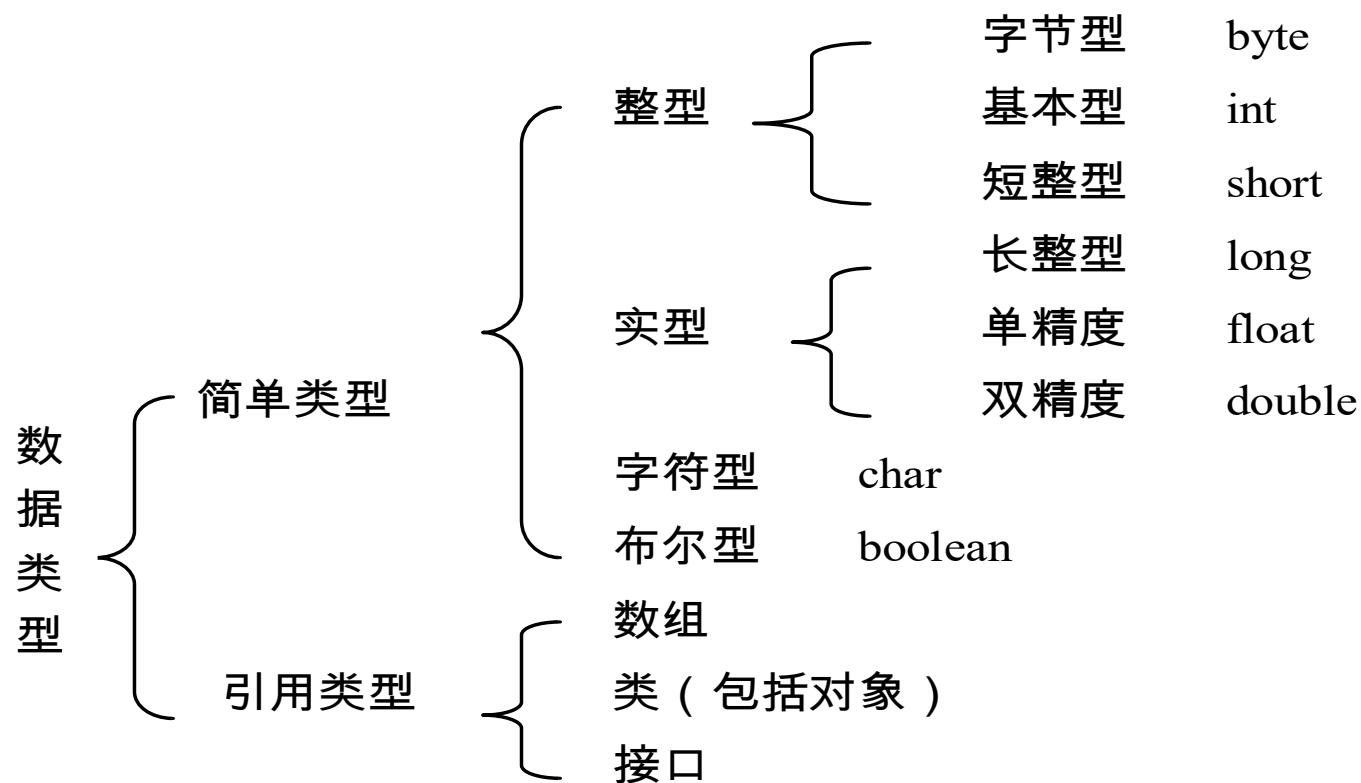
变量的分类 - 按数据类型

- 对于每一种数据都定义了明确的具体数据类型，
在内存中分配了不同大小的内存空间。



红的是 8 个基础类型
蓝的是复合类型

变量的类型

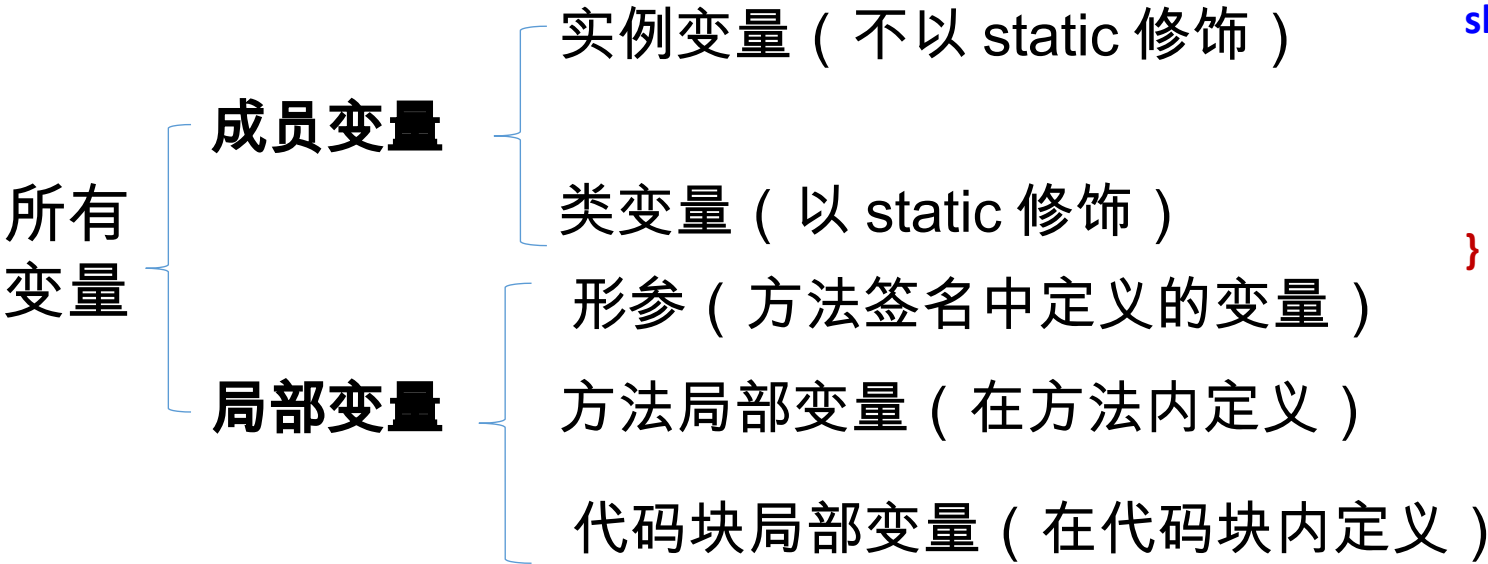


- Java 语言规范提供了两种数据类型：简单类型和引用类型。引用类型可使用一个引用变量得到它的值或者得到由它所表示的值的集合，通过一个简单变量名获取该变量的真实值。

补充：变量的分类 - 按声明的位置的不同

- 在方法体外，类体内声明的变量称为**成员变量**。
- 在方法体内部声明的变量称为**局部变量**。

```
public class Person{  
    private int age ;           // 声明私有变量 age  
    public void showAge(int i) { // 声明方法  
        showAge()  
        age = i;  
    }  
}
```



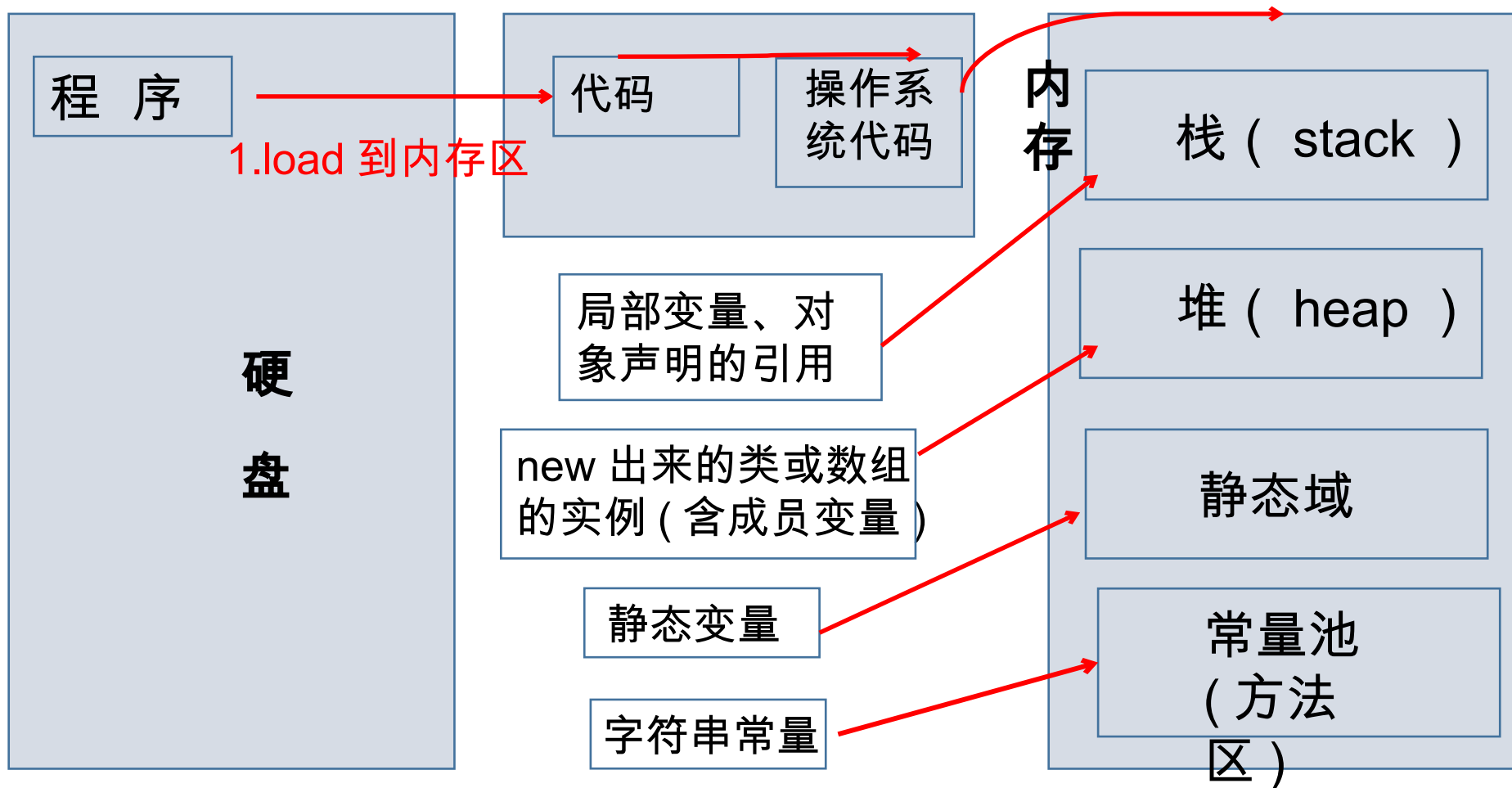
变量是有作用范围的

- 注意：二者在初始化值方面的异同：
同：都有生命周期 异：局部变量除形参外，需显式初始化。

(补充) 程序的执行过程

2. 找到 main
方法开始执行

3. 执行过程中的
内存管理



变量
内存中大致的存储方式

整数类型： byte 、 short 、 int 、 long

- Java 各整数类型有固定的表数范围和字段长度，不受具体 OS 的影响，以保证 java 程序的可移植性。
- java 的整型常量默认为 int 型，声明 long 型常量须后加‘l’或‘L’

类型	占用存储空间	表数范围	使用举例
byte	1 字节 =8bit	-128 ~ 127	对字节操作时使用，如文件读 / 写
short	2 字节	$-2^{15} \sim 2^{15}-1$	保存较小的整数时使用
int	4 字节	$-2^{31} \sim 2^{31}-1$	保存一般的整数时使用，如陕西省总人口
long	8 字节	$-2^{63} \sim 2^{63}-1$	保存较长的整数时使用

浮点类型：float、double

- 与整数类型类似，Java 浮点类型也有固定的表数范围和字段长度，不受具体 OS 的影响。
- Java 的浮点型常量默认为 double 型，声明 float 型常量，须后加‘f’或‘F’。
- 浮点型常量有两种表示形式：
 - 十进制数形式：如：5.12 512.Of .512（必须有小数点）
 - 科学计数法形式：如：5.12e2 512E2 100E-2

类 型	占用存储空间	表数范围	使用举例
单精度 float	4 字节	-3.403E38 ~ 3.403E38	保存小数时使用，如身高、体重
双精度 double	8 字节	-1.798E308 ~ 1.798E308	保存精度较高的小数时使用，如圆周率等

```
//PrimitiveTypeTest.java
package ch002;
```

```
public class PrimitiveTypeTest {
    public static void main(String[] args) {
        // byte
        System.out.println(" 基本类型 : byte  二进制位数 : " + Byte.SIZE);
        System.out.println(" 包装类 : java.lang.Byte");
        System.out.println(" 最小值 : Byte.MIN_VALUE=" + Byte.MIN_VALUE);
        System.out.println(" 最大值 : Byte.MAX_VALUE=" + Byte.MAX_VALUE);
        System.out.println();
        // short
        System.out.println(" 基本类型 : short 二进制位数 : " + Short.SIZE);
        System.out.println(" 包装类 : java.lang.Short");
        System.out.println(" 最小值 : Short.MIN_VALUE=" + Short.MIN_VALUE);
        System.out.println(" 最大值 : Short.MAX_VALUE=" + Short.MAX_VALUE);
        System.out.println();
        // int
        System.out.println(" 基本类型 : int  二进制位数 : " + Integer.SIZE);
        System.out.println(" 包装类 : java.lang.Integer");
        System.out.println(" 最小值 : Integer.MIN_VALUE=" + Integer.MIN_VALUE);
        System.out.println(" 最大值 : Integer.MAX_VALUE=" + Integer.MAX_VALUE);
        System.out.println();
        // long
        System.out.println(" 基本类型 : long 二进制位数 : " + Long.SIZE);
        System.out.println(" 包装类 : java.lang.Long");
        System.out.println(" 最小值 : Long.MIN_VALUE=" + Long.MIN_VALUE);
        System.out.println(" 最大值 : Long.MAX_VALUE=" + Long.MAX_VALUE);
        System.out.println();

        // float
        System.out.println(" 基本类型 : float 二进制位数 : " + Float.SIZE);
        System.out.println(" 包装类 : java.lang.Float");
        System.out.println(" 最小值 : Float.MIN_VALUE=" + Float.MIN_VALUE);
        System.out.println(" 最大值 : Float.MAX_VALUE=" + Float.MAX_VALUE);
        System.out.println();

        // double
        System.out.println(" 基本类型 : double 二进制位数 : " + Double.SIZE);
        System.out.println(" 包装类 : java.lang.Double");
        System.out.println(" 最小值 : Double.MIN_VALUE=" + Double.MIN_VALUE);
        System.out.println(" 最大值 : Double.MAX_VALUE=" + Double.MAX_VALUE);
        System.out.println();

        // char
        System.out.println(" 基本类型 : char 二进制位数 : " + Character.SIZE);
        System.out.println(" 包装类 : java.lang.Character");
        // 以数值形式而不是字符形式将 Character.MIN_VALUE 输出到控制台
        System.out.println(" 最小值 : Character.MIN_VALUE="
            + (int) Character.MIN_VALUE);
        // 以数值形式而不是字符形式将 Character.MAX_VALUE 输出到控制台
        System.out.println(" 最大值 : Character.MAX_VALUE="
            + (int) Character.MAX_VALUE);
    }
}
```

01/08/2026

基本类型 : byte 二进制位数 : 8

包装类 : java.lang.Byte

最小值 : Byte.MIN_VALUE=-128

最大值 : Byte.MAX_VALUE=127

基本类型 : short 二进制位数 : 16

包装类 : java.lang.Short

最小值 : Short.MIN_VALUE=-32768

最大值 : Short.MAX_VALUE=32767

基本类型 : int 二进制位数 : 32

包装类 : java.lang.Integer

最小值 : Integer.MIN_VALUE=-2147483648

最大值 : Integer.MAX_VALUE=2147483647

基本类型 : long 二进制位数 : 64

包装类 : java.lang.Long

最小值 : Long.MIN_VALUE=-9223372036854775808

最大值 : Long.MAX_VALUE=9223372036854775807

基本类型 : float 二进制位数 : 32

包装类 : java.lang.Float

最小值 : Float.MIN_VALUE=1.4E-45

最大值 : Float.MAX_VALUE=3.4028235E38

基本类型 : double 二进制位数 : 64

包装类 : java.lang.Double

最小值 : Double.MIN_VALUE=4.9E-324

最大值 : Double.MAX_VALUE=1.7976931348623157E308

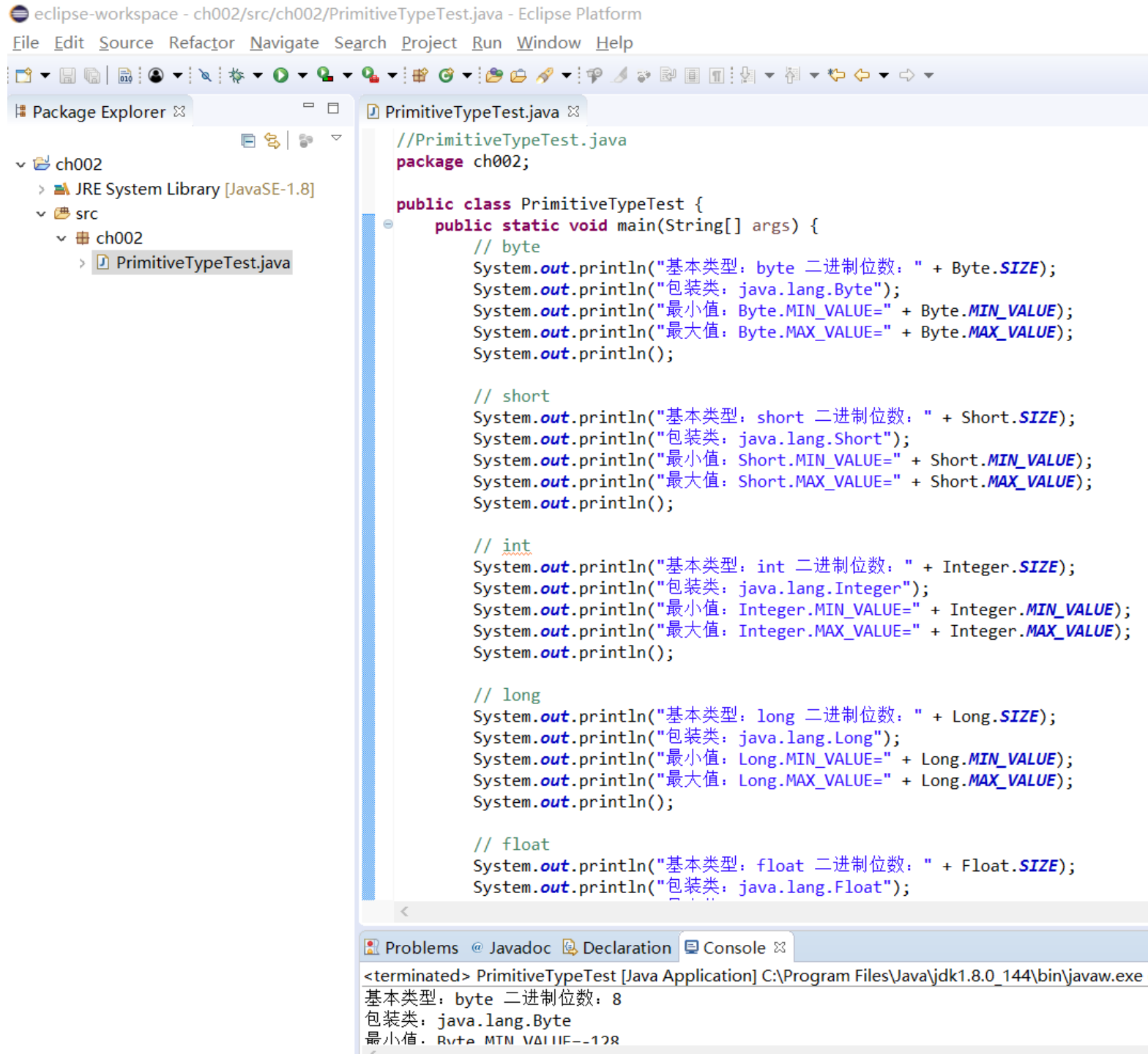
基本类型 : char 二进制位数 : 16

包装类 : java.lang.Character

最小值 : Character.MIN_VALUE=0

最大值 : Character.MAX_VALUE=65535

简单类型的变量



简单类型的变量

字符类型：char

转义字符
Unicode
e

●char 型数据用来表示通常意义上“字符” (2 字节)

类 型	占用存储空间	表数范围	使用举例
字符型 char	2 字节	0~ 65535	保存单个字母或汉字时使用

●字符型常量的三种表现形式：

- 字符常量是用单引号 (‘ ’) 括起来的单个字符，涵盖世界上所有书面语的字符。例如： char c1 = 'a'; char c2 = ' 中 '; char c3 = '9';
- Java 中还允许使用转义字符 ‘ \ ’ 来将其后的字符转变为特殊字符型常量。例如： char c3 = ‘ \n ’ ; // ‘ \n ’ 表示换行符
- 直接使用 Unicode 值来表示字符型常量： ‘ \uXXXX ’。其中， XXXX 代表一个十六进制整数。如： \u000a 表示 \n 。

转义字符	说明
\b	退格符
\n	换行符
\r	回车符
\t	制表符
\"	双引号
'	单引号
\\	反斜线

●char 类型是可以进行运算的。因为它都对应有 Unicode 码。

在 Java 中字符仅以一种形式存在，那就是 Unicode（不选择任何特定的编码，直接使用他们在字符集中的编号，这是统一的唯一方法）。

由于 java 采用 unicode 编码，char 在 java 中占 2 个字节。2 个字节（16 位）来表示一个字符。

简单类型的变量

ASCII 码

- 在计算机内部，所有数据都使用**二进制**表示。每一个二进制位 (bit) 有 0 和 1 两种状态，因此 8 个二进制位就可以组合出 **256 种**状态，这被称为一个字节 (byte)。一个字节一共可以用来表示 256 种不同的状态，每一个状态对应一个符号，就是 256 个符号，从 0000000 到 11111111。
- ASCII 码：上个世纪 60 年代，美国制定了一套字符编码，对**英语字符与二进制位之间的关系，做了统一规定**。这被称为 ASCII 码。ASCII 码一共规定了 **128 个**字符的编码，比如空格“SPACE”是 32 (二进制 00100000)，大写的字母“A”是 65 (二进制 01000001)。这 128 个符号 (包括 32 个不能打印出来的控制符号)，只占用了一个字节的后面 7 位，最前面的 1 位统一规定为 0。
- **缺点：**
 - 不能表示所有字符。
 - 相同的编码表示的字符不一样：比如，130 在法语编码中代表了 é，在希伯来语编码中却代表了字母 Gimel (ג)

Unicode 编码

- 乱码：世界上存在着多种编码方式，同一个二进制数字可以被解释成不同的符号。因此，要想打开一个文本文件，就必须知道它的编码方式，否则用错误的编码方式解读，就会出现乱码。
- Unicode：一种编码，将世界上所有的符号都纳入其中。每一个符号都给予一个独一无二的编码，使用 Unicode 没有乱码的问题。
- Unicode 的缺点：Unicode 只是一个符号集，它只规定了符号的二进制代码，却没有规定这个二进制代码应该如何存储：无法区别 Unicode 和 ASCII：计算机无法区分三个字节表示一个符号还是分别表示三个符号

UTF-8

- UTF-8 是在互联网上使用最广的**一种 Unicode 的实现方式**。
- UTF-8 是一种变长的编码方式。它可以使用 1-6 个字节表示一个符号，根据不同的符号而变化字节长度。常用的英文字母被编码成 1 个字节，汉字通常是 3 个字节，只有很生僻的字符才会被编码成 4-6 个字节。
- 现在计算机系统通用的字符编码工作方式：在计算机内存中，统一使用 Unicode 编码，当需要保存到硬盘或者需要传输的时候，就转换为 UTF-8 编码。用记事本编辑的时候，从文件读取的 UTF-8 字符被转换为 Unicode 字符到内存里，编辑完成后，保存的时候再把 Unicode 转换为 UTF-8 保存到文件：

●UTF-8 的编码规则：

- 对于单字节的 UTF-8 编码，该字节的最高位为 0，后面七位为这个字符的 Unicode 码。
(因此对于英文字符，UTF-8 编码和 ASCII 码是相同的)。
- 对于 n 字节的字符 (n>1)，第一个字节的前 n 位都设为 1，第 n+1 位设为 0，后面字节的前两位一律设为 10。剩下的没有提及的二进制位，全部为这个字符的 Unicode 编码。

●UTF-8 每次传送 8 位数据，是一种可变长的编码格式，其可分为四个区间：

0x0000 0000 至 0x0000 007F:0xxxxxxx

0x0000 0080 至 0x0000 07FF:110xxxxx 10xxxxxx

0x0000 0800 至 0x0000 FFFF:1110xxxx 10xxxxxx 10xxxxxx

0x0001 0000 至 0x0010 FFFF:11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

- UTF-8 解码过程：

对于采用 UTF-8 编码的任意字符 B

- 如果 B 的第一位为 0，则 B 为 ASCII 码，并且 B 独立的表示一个字符；
- 如果 B 的前两位为 1，第三位为 0，则 B 为一个非 ASCII 字符，该字符由多个字节表示，并且该字符由两个字节表示；
- 如果 B 的前三位为 1，第四位为 0，则 B 为一个非 ASCII 字符，该字符由多个字节表示，并且该字符由三个字节表示；比如汉字“王”，它的二进制形式为：0x0000 738B, 属于第三区间, 0x0000 738B - 00000000 00000000 01110011 10001011, 第三区间的编码是 1110xxxx 10xxxxxx 10xxxxxx 把 x 都给替换，则最终“王”字对应的 Unicode 的编码是 11100111 10001110 10001011 转换成 16 进制 0xe7 0x8e 0x8b
- 如果 B 的前四位为 1，第五位为 0，则 B 为一个非 ASCII 字符，该字符由多个字节表示，并且该字符由四个字节表示；

布尔类型：boolean

- boolean 类型适于逻辑运算，一般用于程序流程控制：
 - if 条件控制语句；
 - while 循环控制语句；
 - do-while 循环控制语句；
 - for 循环控制语句；
- boolean 类型数据只允许取值 true 和 false，无 null。
 - 不可以 0 或非 0 的整数替代 false 和 true，这点和 C 语言不同。

类 型	占用存储空间	表数范围	使用举例
布尔型 boolean	1 字节	true 和 false	保存性别、婚否

Java 变量类型

在 Java 语言中，所有的变量在使用前必须声明。声明变量的基本格式如下：

`type identifier [= value][, identifier [= value] ...];`

说明：type 为 Java 数据类型。identifier 是变量名。可以使用逗号隔开来声明多个同类型变量。

以下列出了一些变量的声明实例。注意有些包含了初始化过程。

- `int a, b, c; // 声明三个 int 型整数：a、b、c`
- `int d = 3, e = 4, f = 5; // 声明三个整数并赋予初值`
- `byte z = 22; // 声明并初始化 z`
- `String s = "runoob"; // 声明并初始化字符串 s`
- `double pi = 3.14159; // 声明了双精度浮点型变量`
- `char x = 'x'; // 声明变量 x 的值是字符 'x'。`

先定义，再使用

Java 语言支持的变量类型有：

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

```
public class Person{  
    private int age ;           // 声明私有变量 age  
    public void showAge(int i) { // 声明方法  
        showAge( )  
        age = i;  
    }  
}
```

Java 局部变量

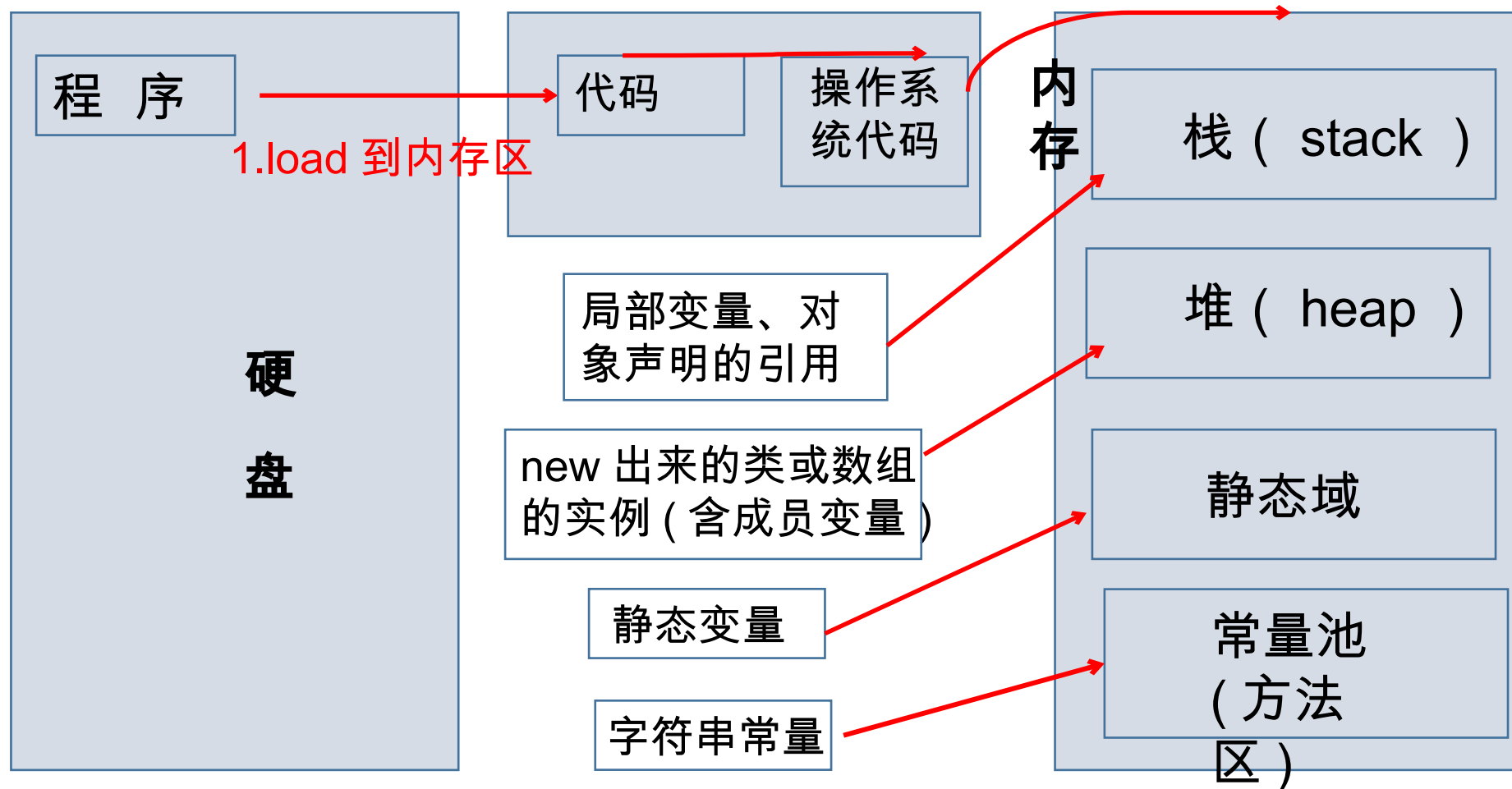
- 局部变量声明在方法、构造方法或者语句块中；
- 局部变量在方法、构造方法、或者语句块被执行的时候创建，当它们执行完成后，变量将会被销毁；
- 访问修饰符（ `public/protected/private` ）不能用于局部变量；
- 局部变量只在声明它的方法、构造方法或者语句块中可见；
- 局部变量是在栈上分配的。
- **局部变量没有默认值**，所以局部变量被声明后，必须经过初始化，才可以使用。

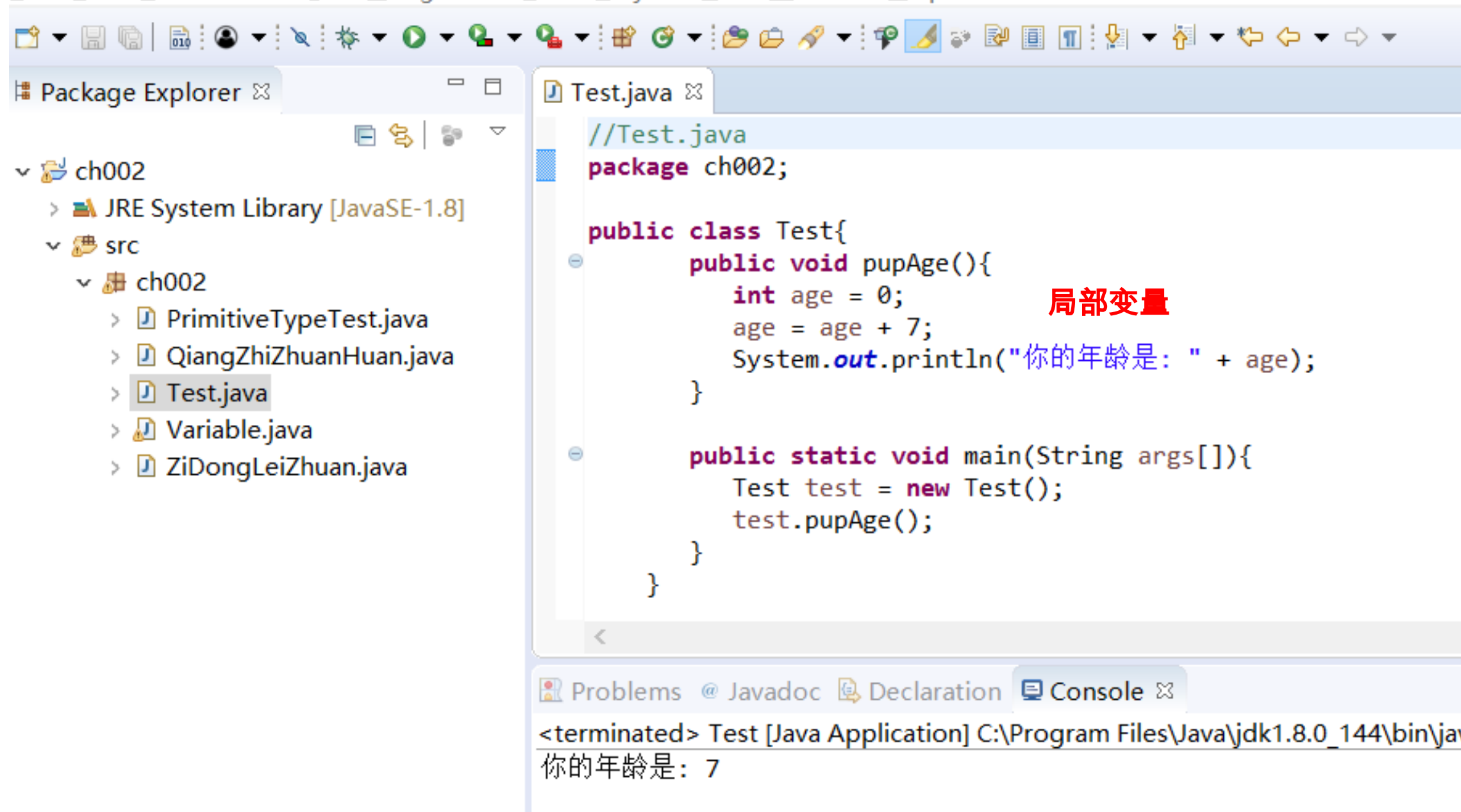
程序的执行过程

(补充) 程序的执行过程

2. 找到 main
方法开始执行

3. 执行过程中的
内存管理





The screenshot displays the Eclipse IDE interface. On the left, the Package Explorer shows the project structure: 'ch002' containing 'JRE System Library [JavaSE-1.8]', 'src', and 'ch002' (which contains 'PrimitiveTypeTest.java', 'QiangZhiZhuanHuan.java', 'Test.java', 'Variable.java', and 'ZiDongLeiZhuan.java'). The main editor shows the code for 'Test.java':

```
//Test.java
package ch002;

public class Test{
    public void pupAge(){
        int age = 0;
        age = age + 7;
        System.out.println("你的年龄是: " + age);
    }

    public static void main(String args[]){
        Test test = new Test();
        test.pupAge();
    }
}
```

A red annotation '局部变量' (Local Variable) is placed next to the line 'int age = 0;'. The bottom of the IDE shows the Console tab with the output: '<terminated> Test [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\jav' followed by '你的年龄是: 7'.

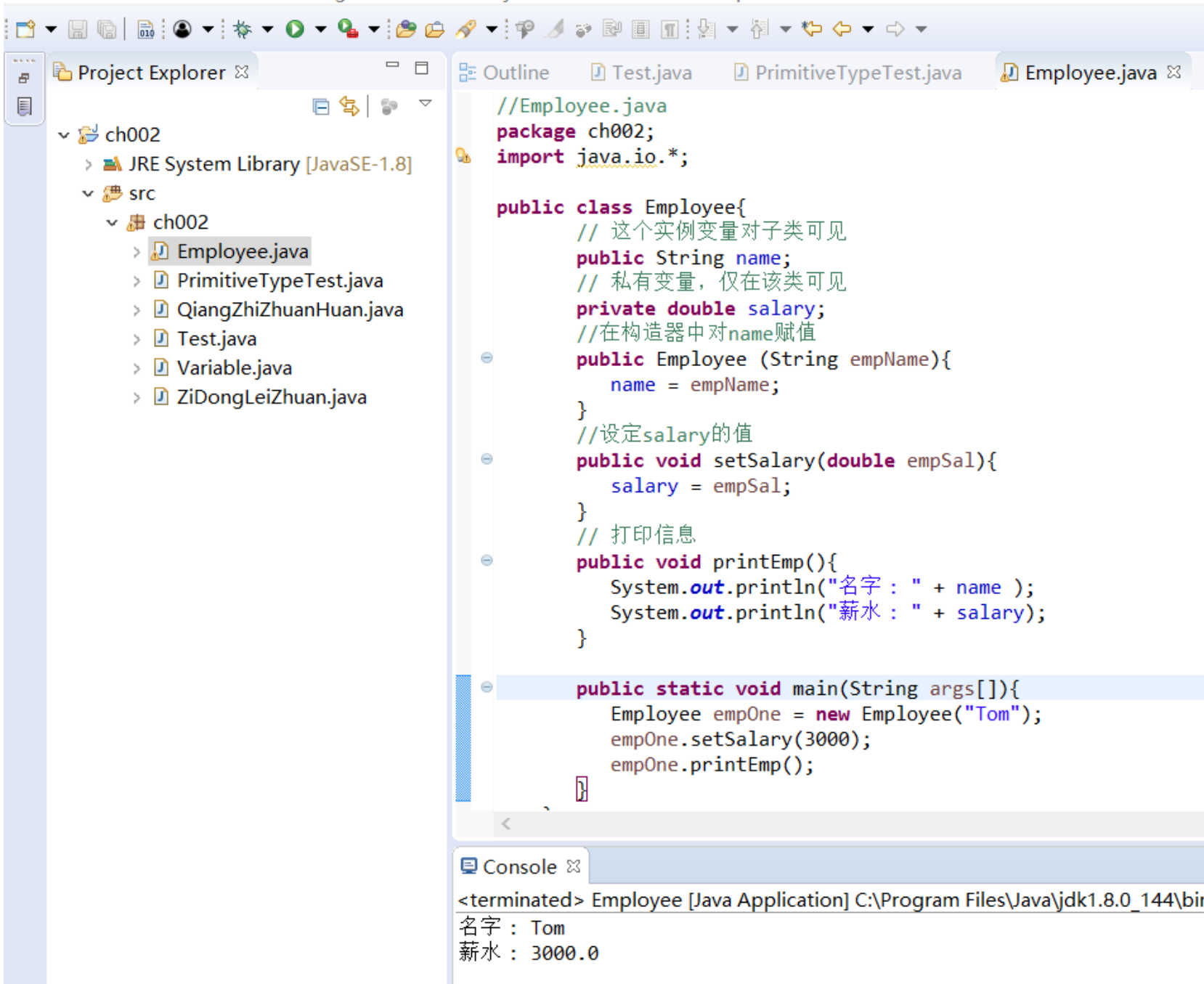
```

//Employee.java
package ch002;
import java.io.*;

public class Employee{
    // 这个实例变量对子类可见
    public String name;           实例变量
    // 私有变量，仅在该类可见
    private double salary;
    // 在构造器中对 name 赋值
    public Employee (String empName){
        name = empName;          局部变量
    }
    // 设定 salary 的值
    public void setSalary(double empSal){
        salary = empSal;
    }
    // 打印信息
    public void printEmp(){
        System.out.println(" 名字 : " + name );
        System.out.println(" 薪水 : " + salary);
    }

    public static void main(String args[]){
        Employee empOne = new Employee("Tom");
        empOne.setSalary(3000);
        empOne.printEmp();
    }
}

```



Project Explorer

- ch002
 - JRE System Library [JavaSE-1.8]
 - src
 - ch002
 - Employee.java
 - PrimitiveTypeTest.java
 - QiangZhiZhuanHuan.java
 - Test.java
 - Variable.java
 - ZiDongLeiZhuan.java

Outline

Test.java

PrimitiveTypeTest.java

Employee.java

```
//Employee.java
package ch002;
import java.io.*;

public class Employee{
    // 这个实例变量对子类可见
    public String name;
    // 私有变量, 仅在该类可见
    private double salary;
    //在构造器中对name赋值
    public Employee (String empName){
        name = empName;
    }
    //设定salary的值
    public void setSalary(double empSal){
        salary = empSal;
    }
    // 打印信息
    public void printEmp(){
        System.out.println("名字 : " + name );
        System.out.println("薪水 : " + salary);
    }

    public static void main(String args[]){
        Employee empOne = new Employee("Tom");
        empOne.setSalary(3000);
        empOne.printEmp();
    }
}
```

Console

<terminated> Employee [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin

名字 : Tom

薪水 : 3000.0

类变量（静态变量）

某种意义上的公共 / 全局变量

- 类变量也称为静态变量，在类中以 `static` 关键字声明，但必须在方法构造方法和语句块之外。
- 无论一个类创建了多少个对象，类只拥有类变量的一份拷贝。
- 静态变量除了被声明为常量外很少使用。常量是指声明为 `public/private`，`final` 和 `static` 类型的变量。常量初始化后不可改变。
- 静态变量储存在静态存储区。经常被声明为常量，很少单独使用 `static` 声明变量。
- 静态变量在程序开始时创建，在程序结束时销毁。
- 与实例变量具有相似的可见性。但为了对类的使用者可见，大多数静态变量声明为 `public` 类型。
- 默认值和实例变量相似。数值型变量默认值是 `0`，布尔型默认值是 `false`，引用类型默认值是 `null`。变量的值可以在声明的时候指定，也可以在构造方法中指定。此外，静态变量还可以在静态语句块中初始化。
- 静态变量可以通过：`ClassName.VariableName` 的方式访问。
- 类变量被声明为 `public static final` 类型时，类变量名称一般建议使用大写字母。如果静态变量不是 `public` 和 `final` 类型，其命名方式与实例变量以及局部变量的命名方式

//Employee01.java

package ch002;

import java.io.*;

public class Employee01 {

//salary 是静态的私有变量

private static double *salary*; 类变量 / 全局变量

// DEPARTMENT 是一个常量

public static final String *DEPARTMENT* = " 开发人员 "; **final** 让其充当“常量”

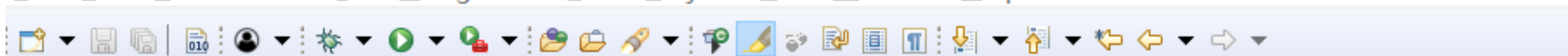
public static void main(String args[]){

salary = 10000;

 System.*out.println*(*DEPARTMENT*+" 平均工

资:"*salary*);

}



Project Explorer

- ch002
 - JRE System Library [JavaSE-1.8]
 - src
 - ch002
 - Employee.java
 - Employee01.java
 - PrimitiveTypeTest.java
 - QiangZhiZhuanHuan.java
 - Test.java
 - Variable.java
 - ZiDongLeiZhuan.java

Employee01.java

```
//Employee01.java
package ch002;
import java.io.*;

public class Employee01 {
    //salary是静态的私有变量
    private static double salary;

    // DEPARTMENT是一个常量
    public static final String DEPARTMENT = "开发人员";

    public static void main(String args[]){
        salary = 10000;
        System.out.println(DEPARTMENT+"平均工资:"+salary);
    }
}
```

类变量 / 全局变量

Console

<terminated> Employee01 [Java Application] C:\Program Files\Java\jdk1.8.0_1
开发人员平均工资:10000.0

Java 中静态变量和实例变量区别

- 静态变量属于类，通过类名就可以调用静态变量。

静态变量与实例变量

- 实例变量属于该类的对象，必须产生该类对象，才能调用实例变量。

在程序运行时的区别：

- 实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个实例变量。

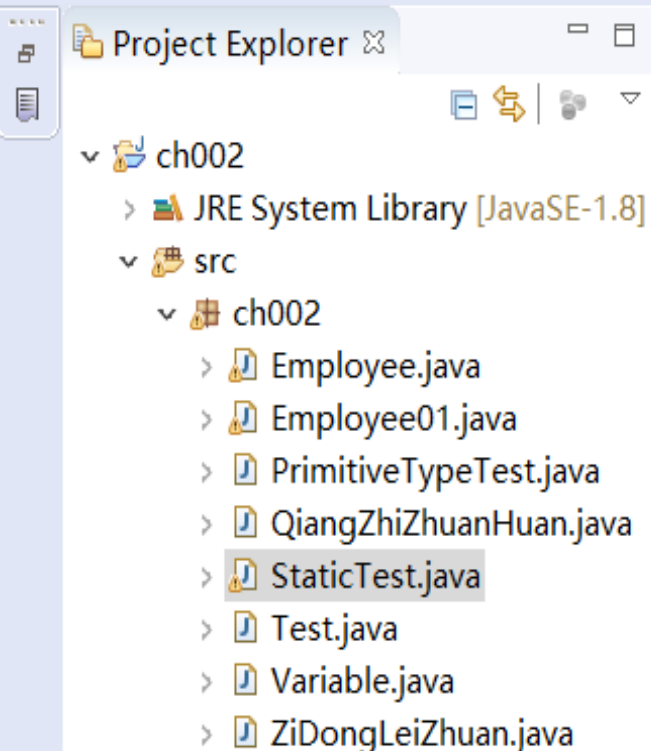
- 静态变量不属于某个实例对象，而是属于类，所以也称为类变量，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。

总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用。

例如，对于下面的程序，无论创建多少个实例对象，

永远都只分配了一个 `static Int` 变量，并且每创建一个实例对象，这个 `staticInt` 就会加 1；但是，每创建一个实例对象，就会分配一个 `random`，

即可能分配多个 `random`，并且每个 `random` 的值都只自加了 1 次。



```
//StaticTest.java
package ch002;

public class StaticTest {
    private static int staticInt = 2; 静态变量
    private int random = 2;

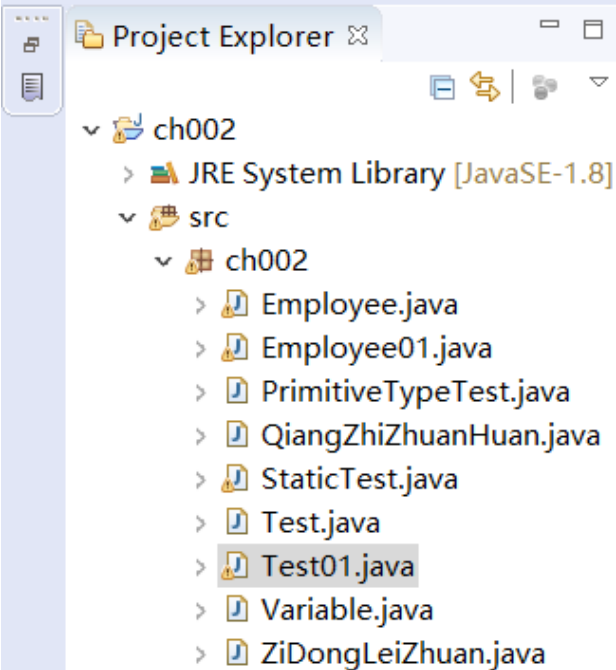
    public StaticTest() {
        staticInt++;
        random++;
        System.out.println("staticInt = "+staticInt+" random = "+random);
    }

    public static void main(String[] args) {
        StaticTest test = new StaticTest();
        StaticTest test2 = new StaticTest();
    }
}
```

静态变量

Console

```
<terminated> StaticTest [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年10月21日 下午4:46:57)
staticInt = 3 random = 3
staticInt = 4 random = 3
```



```
Employee01.java StaticTest.java Test01.java
//Test01.java
/*类的静态数据成员值被所有对象共享，任何对象都可以访问类的静态数据成员。
 * 但是他们使用的是同一个数据，操作的是同一块内存，无论哪个对象修改了它，
 * 对其他对象来说，他已经变了。*/
package ch002;

class A{
    static int i;
    void change(int i1){i=i1;}
}

public class Test01{
    public static void main(String args[]){
        A.i=10;
        A a=new A();
        A b=new A();
        System.out.println(A.i+","+a.i+","+b.i);//10,10,10
        a.change(40);
        System.out.println(A.i+","+a.i+","+b.i);//40,40,40
        b.i+=10;
        System.out.println(A.i+","+a.i+","+b.i);//50,50,50
    }
}
```

Console

```
<terminated> Test01 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (20
10,10,10
40,40,40
50,50,50
```

第 2 讲 Java 语法基础 -1

2.1 关键字、保留字、标识符

2.2 变量和数据类型

2.3 运算符

2.4 数据类型转换

运算符

运算符是一种特殊的符号，用以表示数据的运算、赋值和比较等。

- 算术运算符
- 赋值运算符
- 比较运算符（关系运算符）
- 逻辑运算符
- 位运算符
- 三元运算符

1. 算术运算符

运算符	运算说明	示例	结果
+	正号	+3	3
-	负号	b=4; -b	-4
+	加法运算符	3+2	5
-	减法运算符	6-1	5
*	乘法运算符	2*3	6
/	求两个整数的商	8/5	1
%	求两个整数的余数	9%7	2
++ ++	自增（前）：先运算后取值 自增（后）：先取值后运算	a=2;b=++a; a=2;b=a++;	a=3;b=3 a=3;b=2
-- --	自减（前）：先运算后取值 自减（后）：先取值后运算	a=2;b=- -a a=2;b=a- -	a=1;b=1 a=1;b=2
+	字符串相加	"He"+"llo"	"Hello"

算术运算

算术运算符的注意事项

- 如果对负数取模，可以把模数负号忽略不记，如： $5\%-2=1$ 。但被模数是负数则不可忽略。此外，取模运算的结果不一定总是整数。
- 对于除号“/”，它的整数除和小数除是有区别的：**整数之间做除法时，只保留整数部分而舍弃小数部分。**

例如：int x=3510;x=x/1000*1000; x 的结果是？

//3*1000

除号“/”

- “+”除字符串相加功能外，还能把非字符串转换成字符串。例如：System.out.println("5+5="+5+5); // 打印结果是？

//5+5=55

字符串

- 以下二者的区别：

➤ System.out.println('*' + '\t' + '*');
//unicode 字符操作

➤ System.out.println("*" + '\t' + '*');
//字符串操作

// 字符

小练习

String str1 = 4; // 判断对错：错

System.out .println(3+4+"Hello! "); // 输出：7Hello!

System.out.println("Hello! "+3+4); // 输出：Hello!34

System.out.println(' a ' +1+"Hello! "); // 输出：98Hello!

System.out.println("Hello ! "+"a"+1); // 输出：Hello!a1

The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer showing a project named 'ch002' with a source folder 'src' containing several Java files. 'TestPlusOperation.java' is selected. The main editor displays the code for 'TestPlusOperation.java', which includes package declarations, a class definition, and a main method with various arithmetic and increment/decrement operations. The console at the bottom shows the output of the program, displaying the results of these operations.

Project Explorer:

- ch002
 - JRE System Library [JavaSE-1.8]
 - src
 - ch002
 - Employee.java
 - Employee01.java
 - PrimitiveTypeTest.java
 - QiangZhiZhuanHuan.java
 - StaticTest.java
 - Test.java
 - Test01.java
 - TestPlusOperation.java**
 - Variable.java
 - ZiDongLeiZhuan.java

TestPlusOperation.java:

```
//TestPlusOperation.java
package ch002;

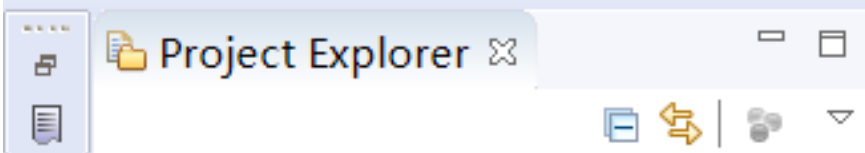
public class TestPlusOperation {

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

Console:

```
<terminated> TestPlusOperation [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\ja
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++ = 10
a-- = 11
d++ = 25
++d = 27
```

运算符



Project Explorer

- ch002
 - JRE System Library [JavaSE-1.8]
 - src
 - ch002
 - Employee.java
 - Employee01.java
 - PrimitiveTypeTest.java
 - QiangZhiZhuanHuan.java
 - selfAddMinus.java
 - StaticTest.java
 - Test.java
 - Test01.java
 - TestPlusOperation.java
 - Variable.java
 - ZiDongLeiZhuan.java

Employee01.java StaticTest.java Test01.java TestPlu

//selfAddMinus.java

package ch002;

public class selfAddMinus{

public static void main(String[] args){

int a = 3; //定义一个变量;

int b = ++a; //自增运算

int c = 3;

int d = --c; //自减运算

System.out.println("进行自增运算后的值等于"+b);

System.out.println("进行自减运算后的值等于"+d);

}

}

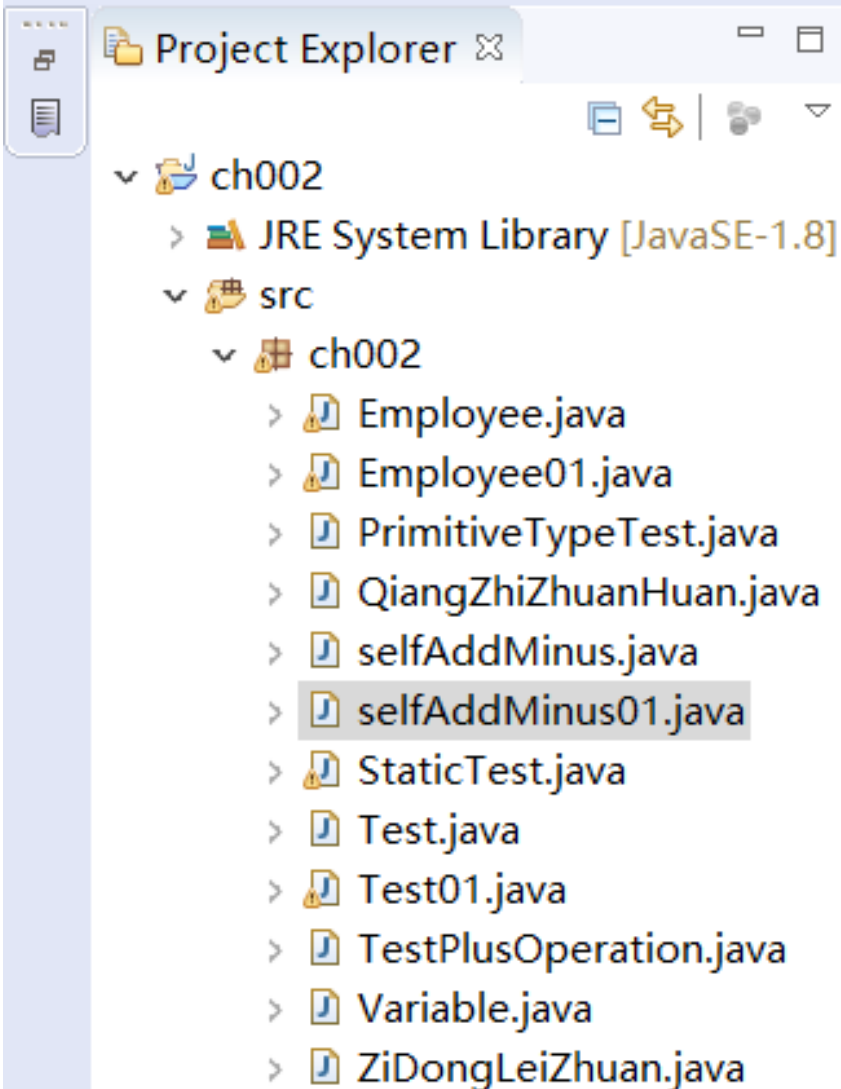
运算符

Console

<terminated> selfAddMinus [Java Application] C:\Program Files\Java\jdk

进行自增运算后的值等于4

进行自减运算后的值等于2



Employee01.java StaticTest.java Test01.java TestPlusOperation.java

//selfAddMinus01.java

package ch002;

运算符

public class selfAddMinus01{

public static void main(String[] args){

int a = 5; //定义一个变量;

int b = 5;

int x = 2*++a;

int y = 2*b++;

System.out.println("自增运算符前缀运算后a="+a+",x="+x);

System.out.println("自增运算符后缀运算后b="+b+",y="+y);

}

}

Console

<terminated> selfAddMinus01 [Java Application] C:\Program Files\Java\jdk1.8.0_144\b

自增运算符前缀运算后a=6,x=12

自增运算符后缀运算后b=6,y=10

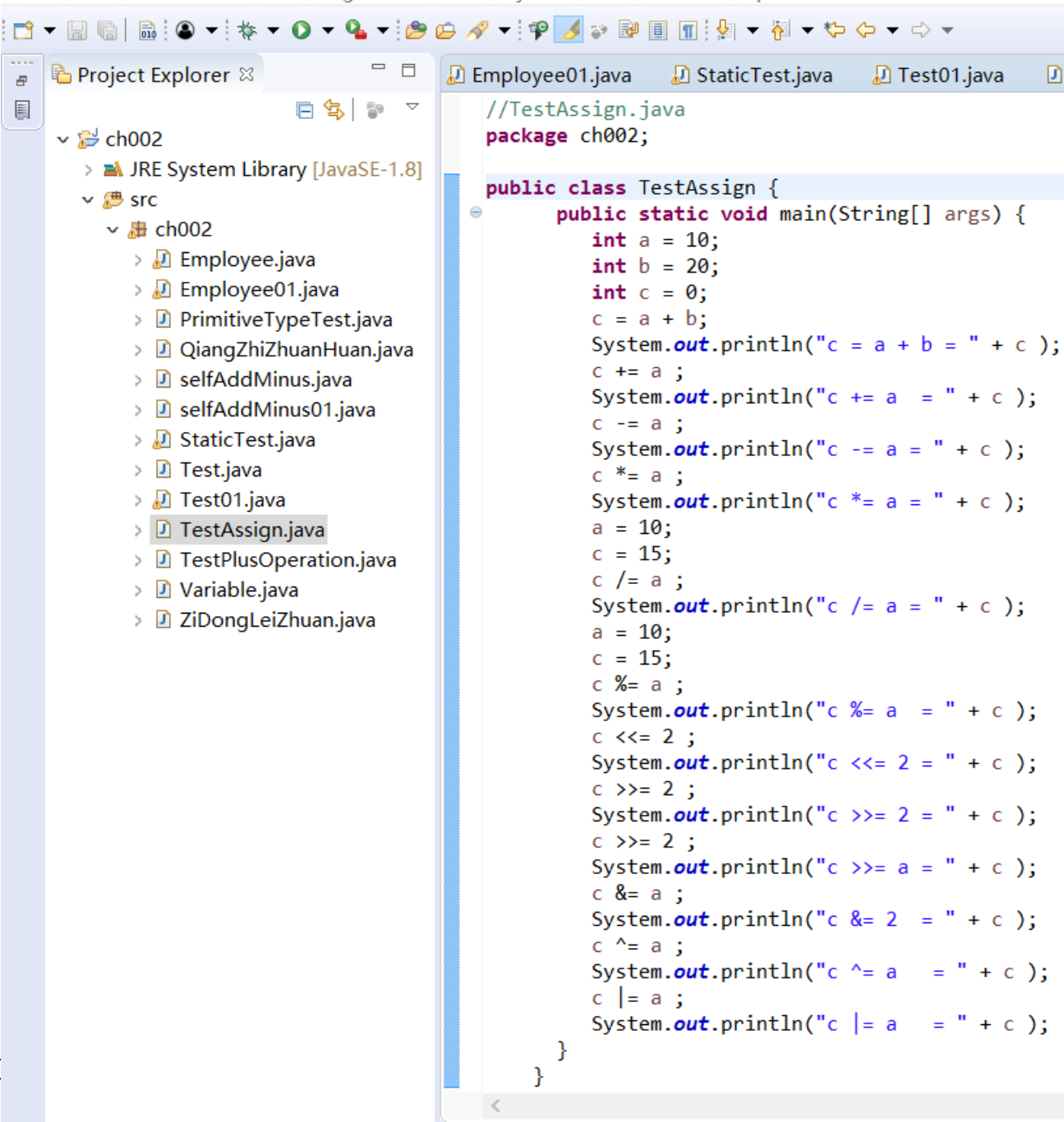
2. 赋值运算符

- 符号： =

赋值运算符

- 当“=”两侧数据类型不一致时，可以使用自动类型转换或使用强制类型转换原则进行处理。
- 支持连续赋值。

- 扩展赋值运算符： +=, -=, *=, /=, %=



赋值运算符

Console

<terminated> TestAssign

c = a + b = 30

c += a = 40

c -= a = 30

c *= a = 300

c /= a = 1

c %= a = 5

c <<= 2 = 20

c >>= 2 = 5

c >>= a = 1

c &= 2 = 0

c ^= a = 10

c |= a = 10

3. 比较运算符

运算符	运算	范例	结果
==	相等于	4==3	false
!=	不等于	4!=3	true
<	小于	4<3	false
>	大于	4>3	true
<=	小于等于	4<=3	false
>=	大于等于	4>=3	true
instanceof	检查是否是类的对象	“ Hello ” instanceof String	true

- 比较运算符的结果都是 boolean 型，也就是要么是 true ，要么是 false 。
- 比较运算符“ == ”不能误写成“ = ”。

关系运算符

关系运算符

下表为Java支持的关系运算符

表格中的实例整数变量A的值为10，变量B的值为20：

运算符	描述	例子
==	检查如果两个操作数的值是否相等，如果相等则条件为真。	(A == B) 为假(非真)。
!=	检查如果两个操作数的值是否相等，如果值不相等则条件为真。	(A != B) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是那么条件为真。	(A> B) 非真。
<	检查左操作数的值是否小于右操作数的值，如果是那么条件为真。	(A <B) 为真。
> =	检查左操作数的值是否大于或等于右操作数的值，如果是那么条件为真。	(A> = B) 为假。
<=	检查左操作数的值是否小于或等于右操作数的值，如果是那么条件为真。	(A <= B) 为真。

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays the project structure: ch002 (JRE System Library [JavaSE-1.8], src, ch002). The ch002 folder contains several Java files, with Compare.java selected. The main editor window shows the code for Compare.java, which defines a public class Compare with a main method. The main method contains several System.out.println statements that output the results of relational operations between variables a and b. The Console window at the bottom shows the output of the program, which matches the expected results of the operations.

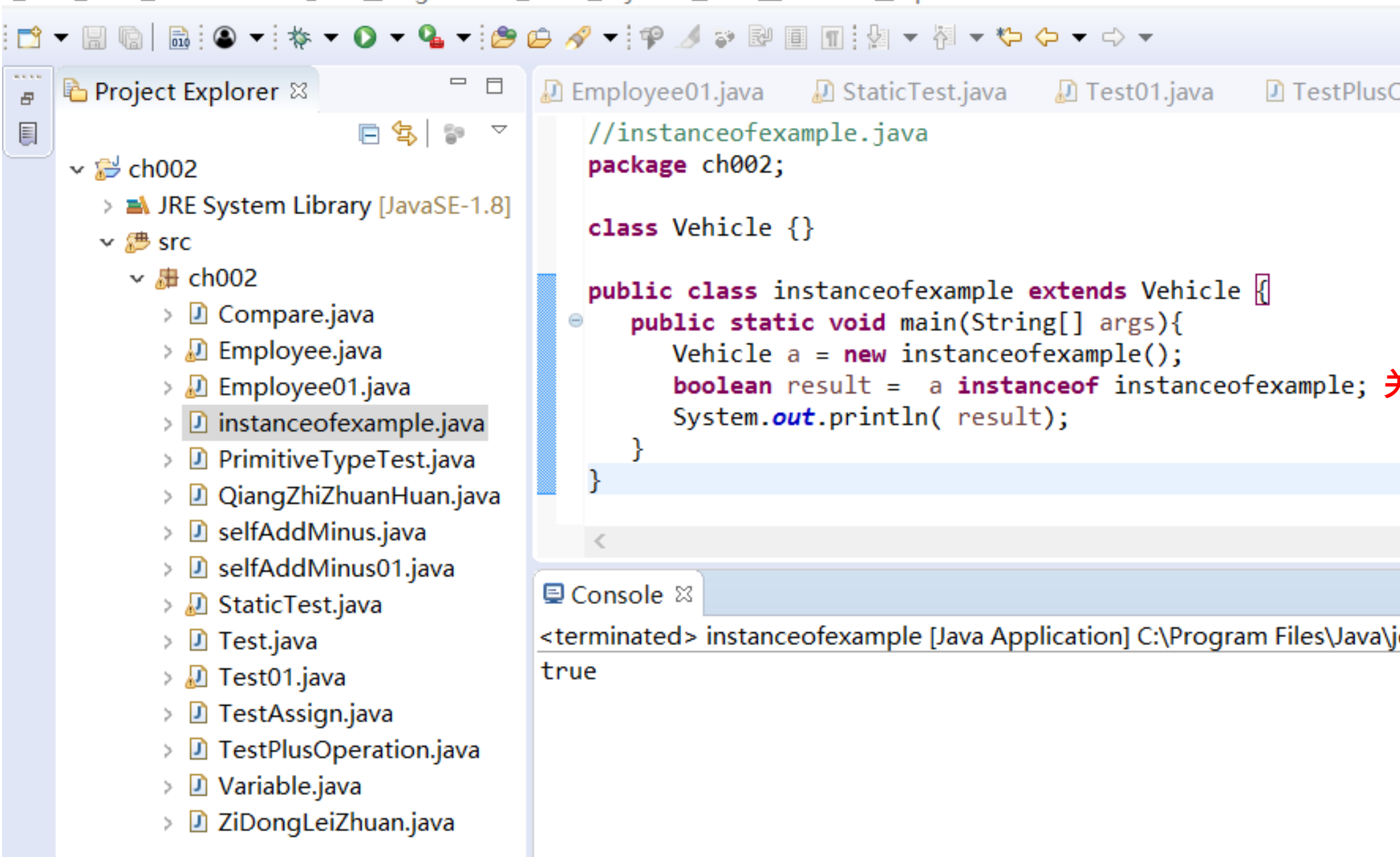
```
//Compare.java
package ch002;

public class Compare {

    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        System.out.println("a == b = " + (a == b) );
        System.out.println("a != b = " + (a != b) );
        System.out.println("a > b = " + (a > b) );
        System.out.println("a < b = " + (a < b) );
        System.out.println("b >= a = " + (b >= a) );
        System.out.println("b <= a = " + (b <= a) );
    }
}
```

关系运算符

<terminated> Compare [Java Application] C:\Program Files\Java\jdk1.8.0.
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false



The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer, which displays a project named 'ch002' containing a 'src' folder. Inside 'src', there is a sub-folder 'ch002' which contains several Java files. The file 'instanceofexample.java' is currently selected and highlighted. The main editor window on the right shows the code for 'instanceofexample.java'. The code defines a 'Vehicle' class and a 'instanceofexample' class that extends 'Vehicle'. The 'main' method in 'instanceofexample' creates a new 'instanceofexample' object, assigns it to a 'Vehicle' reference 'a', and then uses the 'instanceof' operator to check if 'a' is an instance of 'instanceofexample'. The result of this check is printed to the console. A red annotation '关系运算符' (Relationship Operator) points to the 'instanceof' keyword in the code. At the bottom, the Console window shows the output of the program, which is 'true'.

Project Explorer

- ch002
 - JRE System Library [JavaSE-1.8]
 - src
 - ch002
 - Compare.java
 - Employee.java
 - Employee01.java
 - instanceofexample.java
 - PrimitiveTypeTest.java
 - QiangZhiZhuanHuan.java
 - selfAddMinus.java
 - selfAddMinus01.java
 - StaticTest.java
 - Test.java
 - Test01.java
 - TestAssign.java
 - TestPlusOperation.java
 - Variable.java
 - ZiDongLeiZhuan.java

Employee01.java StaticTest.java Test01.java TestPlusC

```
//instanceofexample.java
package ch002;

class Vehicle {}

public class instanceofexample extends Vehicle {
    public static void main(String[] args){
        Vehicle a = new instanceofexample();
        boolean result = a instanceof instanceofexample; 关系运算符
        System.out.println( result);
    }
}
```

Console

<terminated> instanceofexample [Java Application] C:\Program Files\Java\j...
true

4. 逻辑运算符

逻辑运算符

&& — 短路与
逻辑非

|| — 短路或

! — 逻辑非

运算符	操作数
逻辑与 &&	2
逻辑或	2
逻辑非 !	1

a	b	!a	a&& b	a b
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

练习：请写出每题的输出结果

(1)

```
int x = 1;
int y=1;

if(x++==2 & ++y==2){
    x =7;
}
System.out.println("x="+x+",y="+y);
```

(2)

```
int x = 1,y = 1;

if(x++==2 && ++y==2){
    x =7;
}
System.out.println("x="+x+",y="+y);
```

(3)

```
int x = 1,y = 1;

if(x++==1 | ++y==1){
    x =7;
}
System.out.println("x="+x+",y="+y);
```

(4)

```
int x = 1,y = 1;

if(x++==1 || ++y==1){
    x =7;
}
System.out.println("x="+x+",y="+y)
;
```

```
//LuoJi.java
package ch002;
public class LuoJi01{
    ... public static void main(String[] args){
        ... int x=-1;
        ... int y=1;
        ... //Case-1
        ... if(x++==2 & ++y==2){
        ...     x=7;
        ... }
        ... System.out.println("x="+x+",y="+y);
        ... //Case-2
        ... x=-1;
        ... y=-1;
        ... if(x++==2 && ++y==2){
        ...     x=7;
        ... }
        ... System.out.println("x="+x+",y="+y);
        ... //Case-3
        ... x=-1;
        ... y=-1;
        ... if(x++==1 || ++y==1){
        ...     x=7;
        ... }
        ... System.out.println("x="+x+",y="+y);
        ... //Case-4
        ... x=-1;
        ... y=-1;
        ... if(x++==1 || ++y==1){
        ...     x=7;
        ... }
        ... System.out.println("x="+x+",y="+y);
    }
}
```

Problems Console

<terminated> LuoJi01 [Java Application] C:\Program Files\Java\j...

x=2,y=2
x=2,y=1
x=7,y=2
x=7,y=1

&& 和 & 都是表示与，区别是 && 只要第一个条件不满足，后面条件就不再判断。而 & 要对所有的条件都进行判断

逻辑运算符

a	b	a&b	a b	!a	a^b	a&&b	a b
true	true	true	true	false	false	true	true
true	false	false	true	false	true	false	true
false	true	false	true	true	true	false	true
false	false	false	false	true	false	false	false

“||” 如果左边计算后的操作数为 true, 右边则不再执行，返回 true；“|”：前后两个操作数都会进行计算

5. 位运算符

注意：无 <<<

位运算符		
运算符	运算	范例
<<	左移	$3 \ll 2 = 12 \rightarrow 3 * 2 * 2 = 12$
>>	右移	$3 \gg 1 = 1 \rightarrow 3 / 2 = 1$
>>>	无符号右移	$3 \ggg 1 = 1 \rightarrow 3 / 2 = 1$
&	按位与运算	$6 \& 3 = 2$
	按位或运算	$6 3 = 7$
^	按位异或运算	$6 \wedge 3 = 5$
~	按位取反码	$\sim 6 = -7$

相当于乘以 2 次
2
相当于除以 1 次
2

位操作

●位运算是直接对二进制进行运算

位运算符

Java定义了位运算符，应用于整数类型(int)，长整型(long)，短整型(short)，字符型(char)，和字节型(byte)等类型。

位运算符作用在所有的位上，并且按位运算。假设a = 60，b = 13;它们的二进制格式表示将如下：

```
A = 0011 1100
```

```
B = 0000 1101
```

```
-----
```

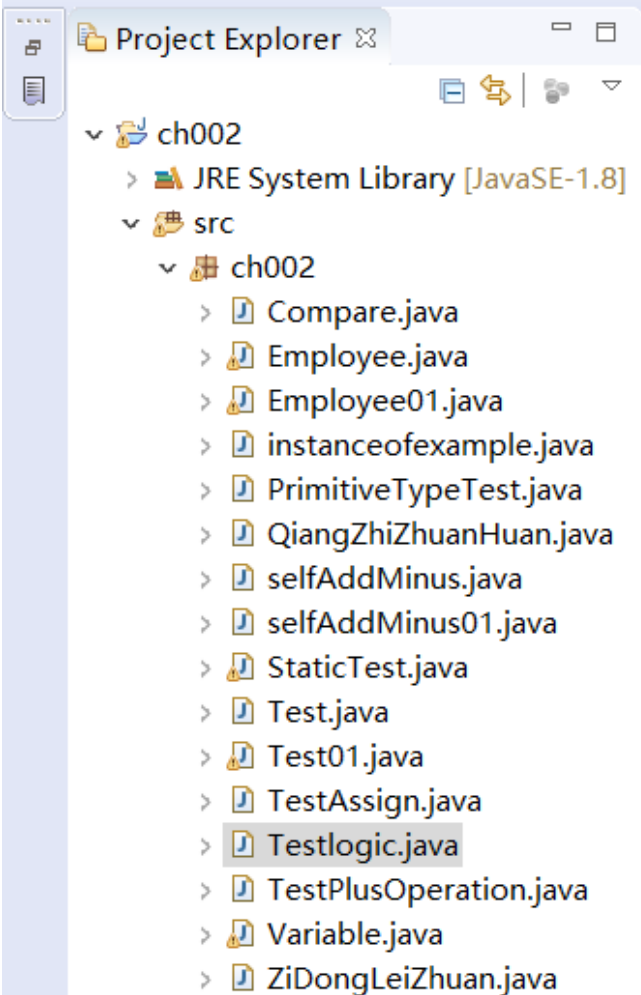
```
A&b = 0000 1100
```

```
A | B = 0011 1101
```

```
A ^ B = 0011 0001
```

```
~A= 1100 0011
```

位操作



Testlogic.java

```
//Testlogic.java
package ch002;

public class Testlogic {
    public static void main(String[] args) {
        int a = 60; /* 60 = 0011 1100 */
        int b = 13; /* 13 = 0000 1101 */
        int c = 0;
        c = a & b;          /* 12 = 0000 1100 */
        System.out.println("a & b = " + c );

        c = a | b;          /* 61 = 0011 1101 */
        System.out.println("a | b = " + c );

        c = a ^ b;          /* 49 = 0011 0001 */
        System.out.println("a ^ b = " + c );

        c = ~a;             /* -61 = 1100 0011 */
        System.out.println("~a = " + c );

        c = a << 2;          /* 240 = 1111 0000 */
        System.out.println("a << 2 = " + c );

        c = a >> 2;          /* 15 = 1111 */
        System.out.println("a >> 2 = " + c );

        c = a >>> 2;         /* 15 = 0000 1111 */
        System.out.println("a >>> 2 = " + c );
    }
}
```

Console

<terminated> Testl

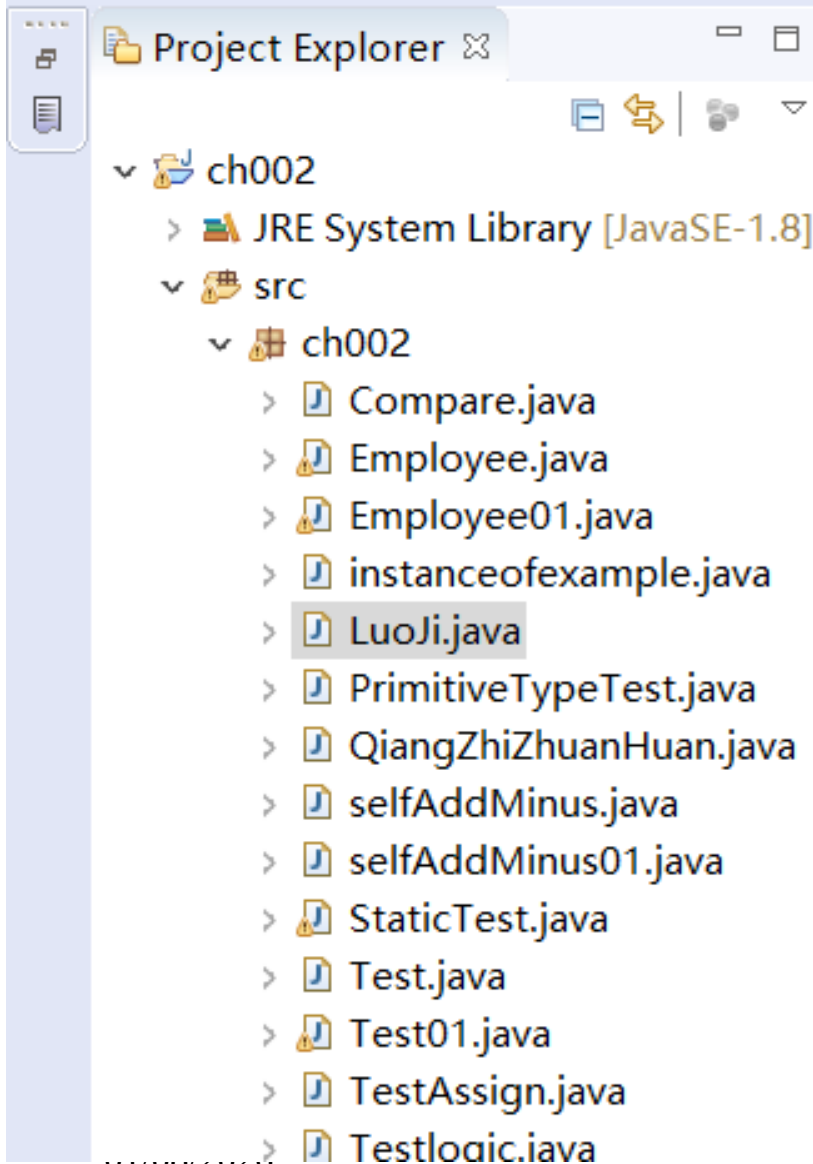
```
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 2 = 15
a >>> 2 = 15
```

位操作

位运算符

位运算符的细节	
<<	空位补 0 ，被移除的高位丢弃，空缺位补 0 。
>>	被移位的二进制最高位是 0 ，右移后，空缺位补 0 ； 最高位是 1 ，空缺位补 1 。
>>>	被移位二进制最高位无论是 0 或者是 1 ，空缺位都用 0 补。
&	二进制位进行 & 运算，只有 1&1 时结果是 1 ，否则是 0;
	二进制位进行 运算，只有 0 0 时结果是 0 ，否则是 1;
^	相同二进制位进行 ^ 运算，结果是 0 ； $1^1=0$, $0^0=0$ 不相同二进制位 ^ 运算结果是 1 。 $1^0=1$, $0^1=1$
~	正数取反，各二进制码按补码各位取反 负数取反，各二进制码按补码各位取反

位操作



```
//LuoJi.java
/* 当使用与逻辑运算符时，在两个操作数都为true时，结果才为true，
 * 但是当得到第一个操作为false时，其结果就必定是false，
 * 这时候就不会再判断第二个操作了。*/
package ch002;

public class LuoJi{
    public static void main(String[] args){
        int a = 5; //定义一个变量；
        boolean b = (a<4)&&(a++<10);
        System.out.println("使用短路逻辑运算符的结果为"+b);
        System.out.println("a的结果为"+a);
    }
}
```

复合操作

Console

```
<terminated> LuoJi [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\java
使用短路逻辑运算符的结果为false
a的结果为5
```

6. 三元运算符

●格式：

➤ (条件表达式) ? 表达式 1 : 表达式 2 ;

↑ ↑
为 true , 运算后的结果是表达式 1 ;
为 false , 运算后的结果是表达式 2 ;

➤ 表达式 1 和表达式 2 为 **同种类型**

三元运算

➤ 三元运算符与 if-else 的联系与区别：

- 1) 三元运算符可简化 if-else 语句
- 2) 三元运算符要求必须返回一个结果。
- 3) if 后的代码块可有多多个语句

条件运算符也被称为三元运算符。该运算符有 3 个操作数，并且需要判断布尔表达式的值。该运算符的主要是决定哪个值应该赋值给变量。

variable x = (expression) ? value if true : value if false

The screenshot displays the Eclipse IDE interface. On the left, the Project Explorer shows a project named 'ch002' with a source folder 'src' containing a sub-folder 'ch002'. Inside 'ch002', the file 'Choose.java' is selected. The main editor window shows the code for 'Choose.java', which defines a public class 'Choose' with a static method 'main'. The code uses ternary operators to assign values to 'b' based on conditions for 'a'. The console at the bottom shows the output of the program, indicating that 'b' is 30 when 'a' is 1 and 20 when 'a' is 10.

```
//Choose.java
package ch002;

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

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

Console Output:

```
<terminated> Choose [Java Application] C:\Program Files\Java\jdk1.8.0_144\bi
Value of b is : 30
Value of b is : 20
```

三元运算

运算符的优先级

- 运算符有不同的优先级，所谓优先级就是表达式运算中的运算顺序。如右表，上一行运算符总优先于下一行。
- 只有单目运算符、三元运算符、赋值运算符是从右向左运算的。

提示：最简单的方法就是在表达式中使用小括号表示运算符的运算次序！！！！

	. () { } ; ,	<div>高</div> <div>↑</div> <div>优先级</div> <div>↓</div> <div>低</div>
R→L	++ -- ~ !(data type)	
L→R	* / %	
L→R	+ -	
L→R	<< >> >>>	
L→R	< > <= >= instanceof	
L→R	== !=	
L→R	&	
L→R	^	
L→R		
L→R	&&	
L→R		
R→L	? :	
R→L	= *= /= %=	
	+= -= <<= >>=	
	>>>= &= ^= =	

表达式

表达式是由运算符、操作数和方法调用，按照语言的语法构造而成的符号序列。表达式可用于计算一个公式、为变量赋值以及帮助控制程序的执行流程。表达式任务有：

- 利用表达式的元素来进行计算。
- 返回计算结果的值。

表达式

表达式返回数值的数据类型取决于在表达式中使用的元素。例如，如果 aChar 是字符型，则表达式 aChar = 'S' 返回一个字符型的值。

Java 语言允许将多个子表达式构造成复合表达式。下面是一个复合表达式的例子：

$$(x + y) * ((z - t) / w)$$

第 2 讲 Java 语法基础 -1

2.1 关键字、保留字、标识符

2.2 变量和数据类型

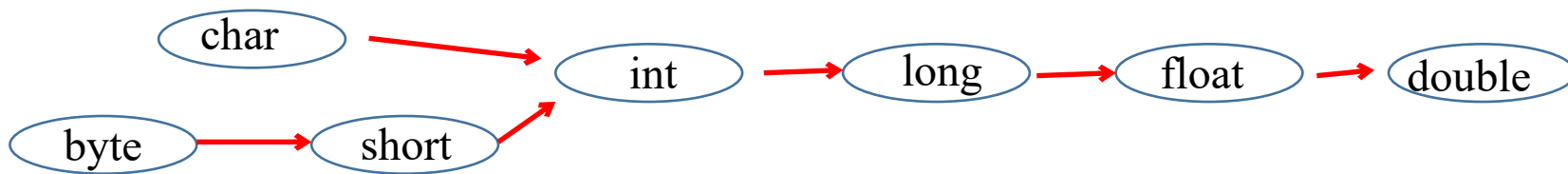
2.3 运算符

2.4 数据类型转换

基本数据类型转换

- **自动类型转换**：容量小的类型自动转换为容量大的数据类型。数据类型按容量大小排序为：

类型转换规则



- 有多种类型的数据混合运算时，系统首先自动将所有数据转换成容量最大的那种数据类型，然后再进行计算。
- byte, short, char 之间不会相互转换，他们三者在计算时首先转换为 int 类型。
- 当把任何基本类型的值和字符串值进行连接运算时 (+)，基本类型的值将自动转化为字符串类型。

小练习

String str1 = 4; // 判断对错：错

System.out .println(3+4+"Hello! "); // 输出：7Hello! 见前述

System.out.println("Hello! "+3+4); // 输出：Hello!34

System.out.println(' a ' +1+"Hello! "); // 输出：98Hello!

System.out.println("Hello ! "+"a"+1); // 输出：Hello!a1

强制类型转换

- 自动类型转换的逆过程，将容量大的数据类型转换为容量小的数据类型。
使用时要加上强制转换符 (())，但可能造成精度降低或溢出，格外要注意。
- 通常，字符串不能直接转换为基本类型，但通过基本类型对应的包装类则可以实现把字符串转换成基本类型。

强制转换

➤如： `String a = "43"; int i = Integer.parseInt(a);`

➤boolean 类型不可以转换为其它的数据类型。

```
//ZiDongLeiZhuan.java
```

```
package ch002;
```

```
public class ZiDongLeiZhuan{
```

```
    public static void main(String[] args){
```

```
        char c1='a';// 定义一个 char 类型
```

```
        int i1 = c1;//char 自动类型转换为 int
```

```
        System.out.println("char 自动类型转换为 int 后的值等于 "+i1);
```

自动转换

```
        char c2 = 'A';// 定义一个 char 类型
```

```
        int i2 = c2+1;//char 类型和 int 类型计算
```

```
        System.out.println("char 类型和 int 计算后的值等于 "+i2);
```

```
    }
```

```
}
```



Package Explorer

ch002

> JRE System Library [JavaSE-1.8]

src

ch002

> PrimitiveTypeTest.java

> ZiDongLeiZhuan.java

PrimitiveTypeTest.java

ZiDongLeiZhuan.java

//ZiDongLeiZhuan.java

package ch002;

public class ZiDongLeiZhuan{

public static void main(String[] args){

char c1='a';//定义一个char类型

int i1 = c1;//char自动类型转换为int

System.out.println("char自动类型转换为int后的值等于 "+i1);

char c2 = 'A';//定义一个char类型

int i2 = c2+1;//char 类型和 int 类型计算

System.out.println("char类型和int计算后的值等于 "+i2);

}

}

自动转换

Problems @ Javadoc Declaration Console

<terminated> ZiDongLeiZhuan [Java Application] C:\Program Files\Java\jdk1.8.0_144\

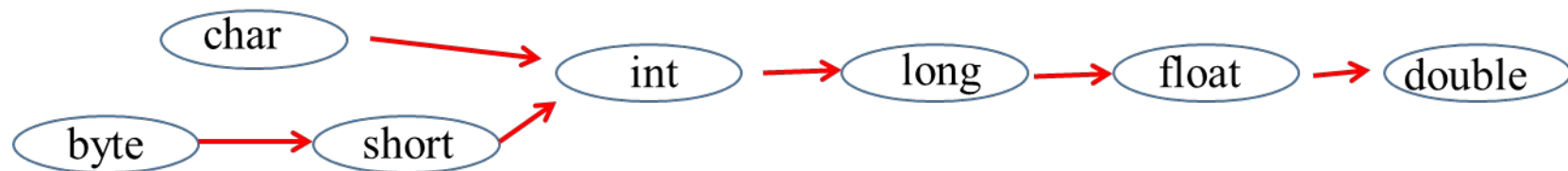
char自动类型转换为int后的值等于 97

char类型和int计算后的值等于 66

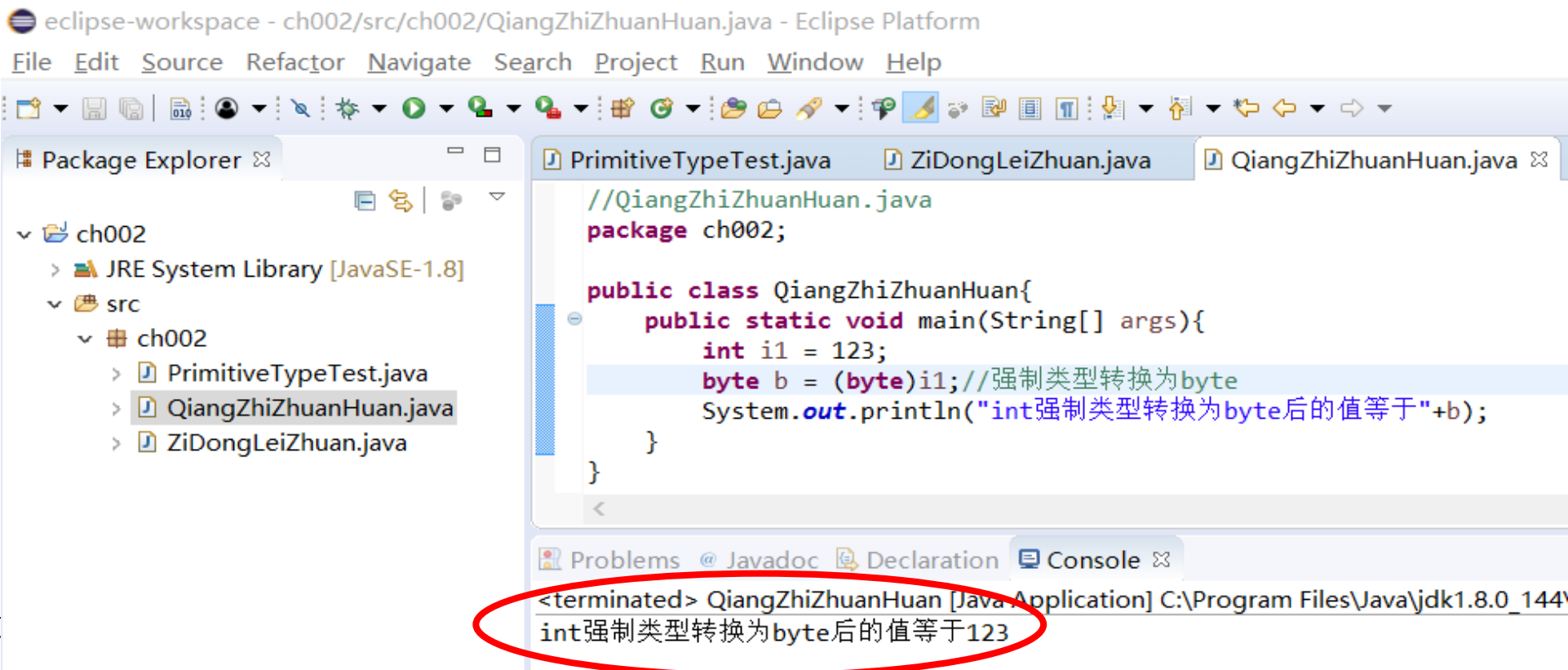
```
//QiangZhiZhuanHuan.java
```

```
package ch002;
```

```
public class QiangZhiZhuanHuan{  
    public static void main(String[] args){  
        int i1 = 123;  
        byte b = (byte)i1; // 强制类型转换为 byte  
        System.out.println("int 强制类型转换为 byte 后的值等于 "+b);  
    }  
}
```



强制转换



作业发邮件到：

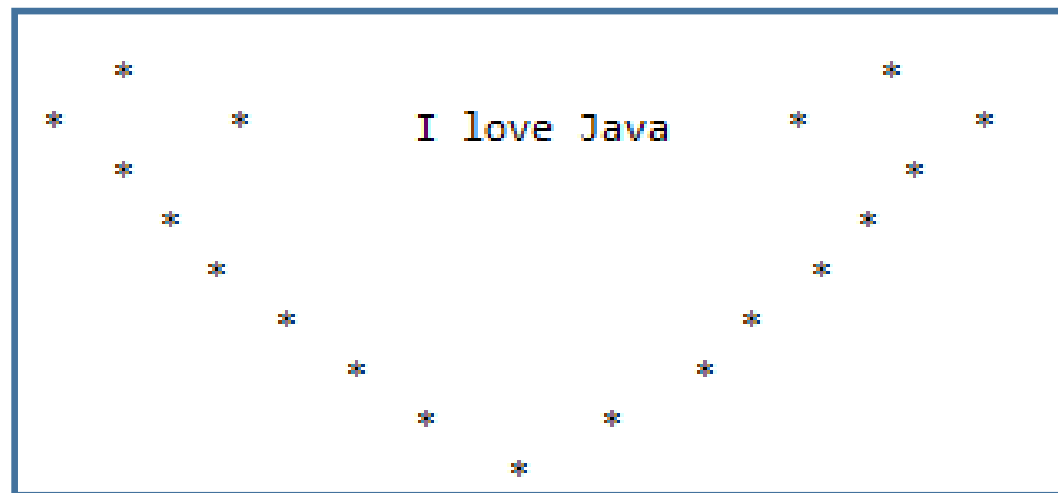
2230652597@qq.com

本章作业

1. 面向过程和面向对象有什么区别？各有什么优缺点？
2. 输入一行字符（大于 30 个），分别统计出其中中英文字母、空格、数字和其他字符的个数。
3. 输入一个正整数，对其分解质因数。输出结果间用空格隔开
4. 第 2 讲 ppt 上的所有程序代码编写（与调试）
5. 《孙子算经》中记载了这样一道题目：今有雉兔同笼，上有三十五头，下有九十四足，问鸡兔各几只？
6. 百钱百鸡，鸡翁一，值钱五，鸡母一，值钱三，鸡雏三，值钱一，百钱买百鸡，问翁、母、雏各几何？

上次（第一讲）作业讲解

1. 调试所有本章 java 示例程序，观察结果
2. 自己独立编写 HelloJava 程序，并配上必要的注释。
3. 将个人的基本信息（姓名、性别、籍贯、住址）打印到控制台上输出。各条信息分别占一行。
4. 结合 `\n` (换行)，`\t` (制表符)，空格等在控制台打印出如下图所示的效果。



```
//PersonalInfo.Java
```

```
package ch001homework;
```

```
public class PersonalInfo {
```

```
public static void main(String args[]) {
```

```
System.out.println("My name is Cooper.");
```

```
System.out.println("Sex: Man");
```

```
System.out.println("I was born in Wuhan.");
```

```
System.out.println("My address is in Wuhan University.");
```

```
}
```

```
}
```

```
//Heart.java
```

```
package ch001homework;
```

```
public class Heart {
```

```
public static void main(String args[]) {
```

```
System.out.print("\t"+"*\t\t\t\t"+"*\n");
```

```
System.out.print("  "+"*\t\t"+"* I love Java *\t\t"+"*\n");//0
```

```
System.out.print("\t"+"1\t\t\t\t\t"+"1\n");//1
```

```
System.out.print("\t"+"*\t\t\t\t"+"*\n");
```

```
System.out.print("\t\t"+"2\t\t\t\t"+"2\n");//2
```

```
System.out.print("\t\t"+"*\t\t"+"*\n");
```

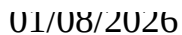
```
System.out.print("\t\t\t"+"3\t\t"+"3\n");//3
```

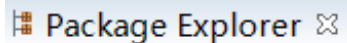
```
System.out.print("\t\t\t"+"*\t"+"*\n");
```

```
System.out.print("\t\t\t\t"+"4\n");//4
```





```
}
```

```
01/08/2026 }
```





- ch001homework
 - JRE System Library [JavaSE-1.8]
- src
 - ch001homework
 - Heart.java
 - Heart01.java
 - PersonallInfo.java

 PersonalInfo.java
  Heart.java
  Heart01.java
 

```
//Heart01.java
package ch001homework;

public class Heart01 {
    public static void main(String args[]) {
        System.out.print("\t"+"*\t\t\t\t\t"+"*\n");
        System.out.print("      "+"*\t\t\t\t"+"*   I love Java  *\t\t"+"          *\n");//0
        System.out.print("\t"+"*\t\t\t\t\t\t\t"+"*\n");//1
        System.out.print("\t"+"      *\t\t\t\t\t\t\t"+"      *\n");
        System.out.print("\t\t"+"*\t\t\t\t\t\t"+"*\n");//2
        System.out.print("\t\t\t"+"*\t\t\t\t\t"+"*\n");
        System.out.print("\t\t\t\t"+"*\t\t\t\t"+"*\n");//3
        System.out.print("\t\t\t\t\t"+"*\t\t\t\t\t"+"*\n");
        System.out.print("\t\t\t\t\t\t\t"+"*\n");//4
    }
}
```

Problems @ Javadoc Declaration Console

<terminated> Heart01 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年10月20日)

A heart shape is formed by a collection of asterisks (*) arranged in a symmetrical pattern. The text "I love Java" is centered within the heart. The asterisks are arranged in a way that they form the outline and internal structure of the heart, with the text "I love Java" placed in the middle. The asterisks are arranged in a way that they form the outline and internal structure of the heart, with the text "I love Java" placed in the middle.