

# 基础语法

@M了个J  
李明杰

<https://github.com/CoderMJLee>

<http://cnblogs.com/mjios>



实力IT教育 [www.520it.com](http://www.520it.com)

码拉松



# 语法须知

FirstClass.java

```
1
2 public class FirstClass {
3
4     public static void main(String[] args) {
5         System.out.println("1只羊");
6         System.out.println("2只羊");
7     }
8
9 }
10
```

- 每一条语句都必须以分号 ; 结尾
- Java 中的方法，就是其他编程语言中的函数
- 程序的入口是 main 方法
- 没有 main 方法，Java 程序是无法启动的
- 方法必须包含在 class 内部，先有 class，再有方法

# 关于左大括号 { 的位置

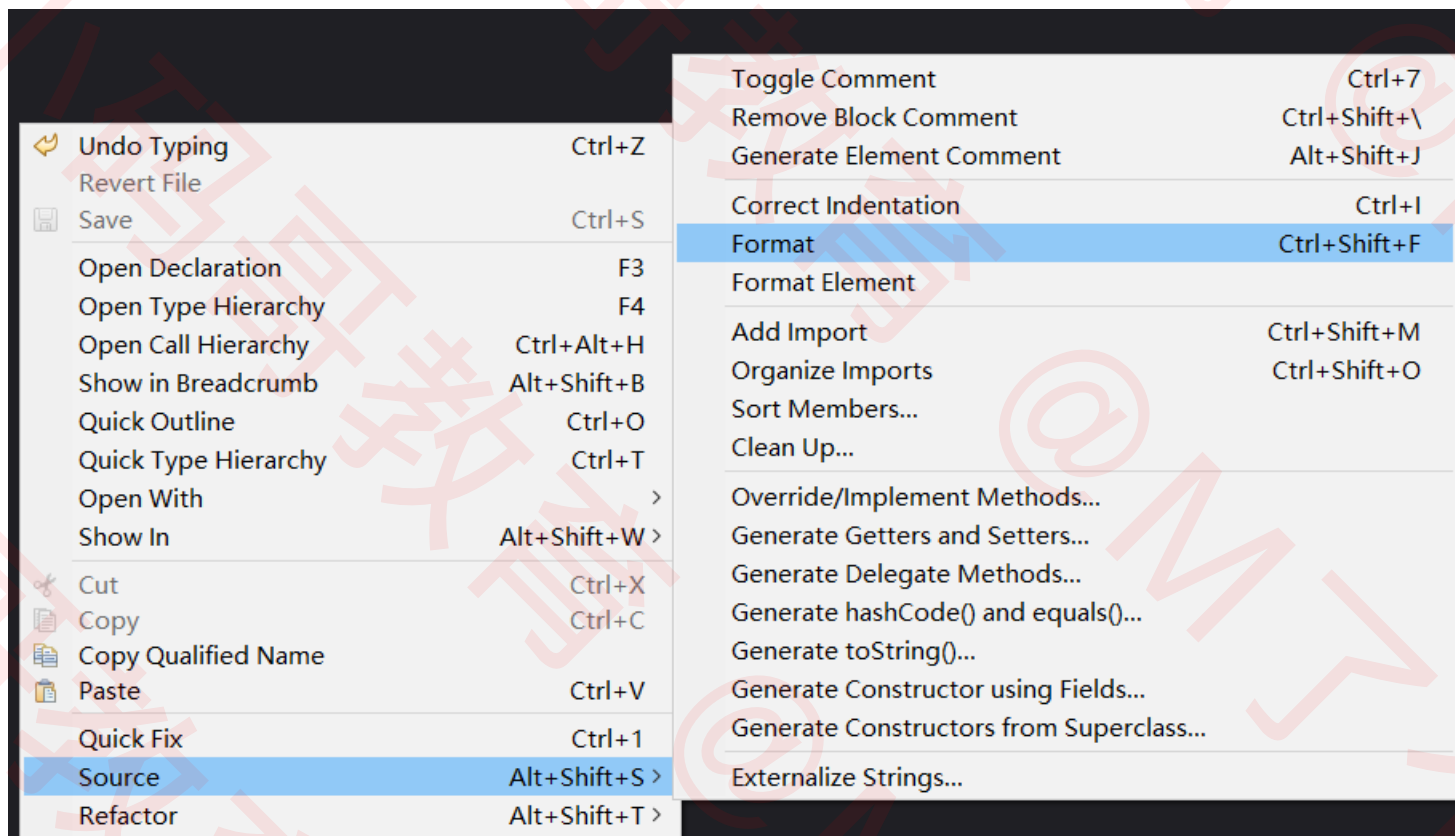
// 推荐

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

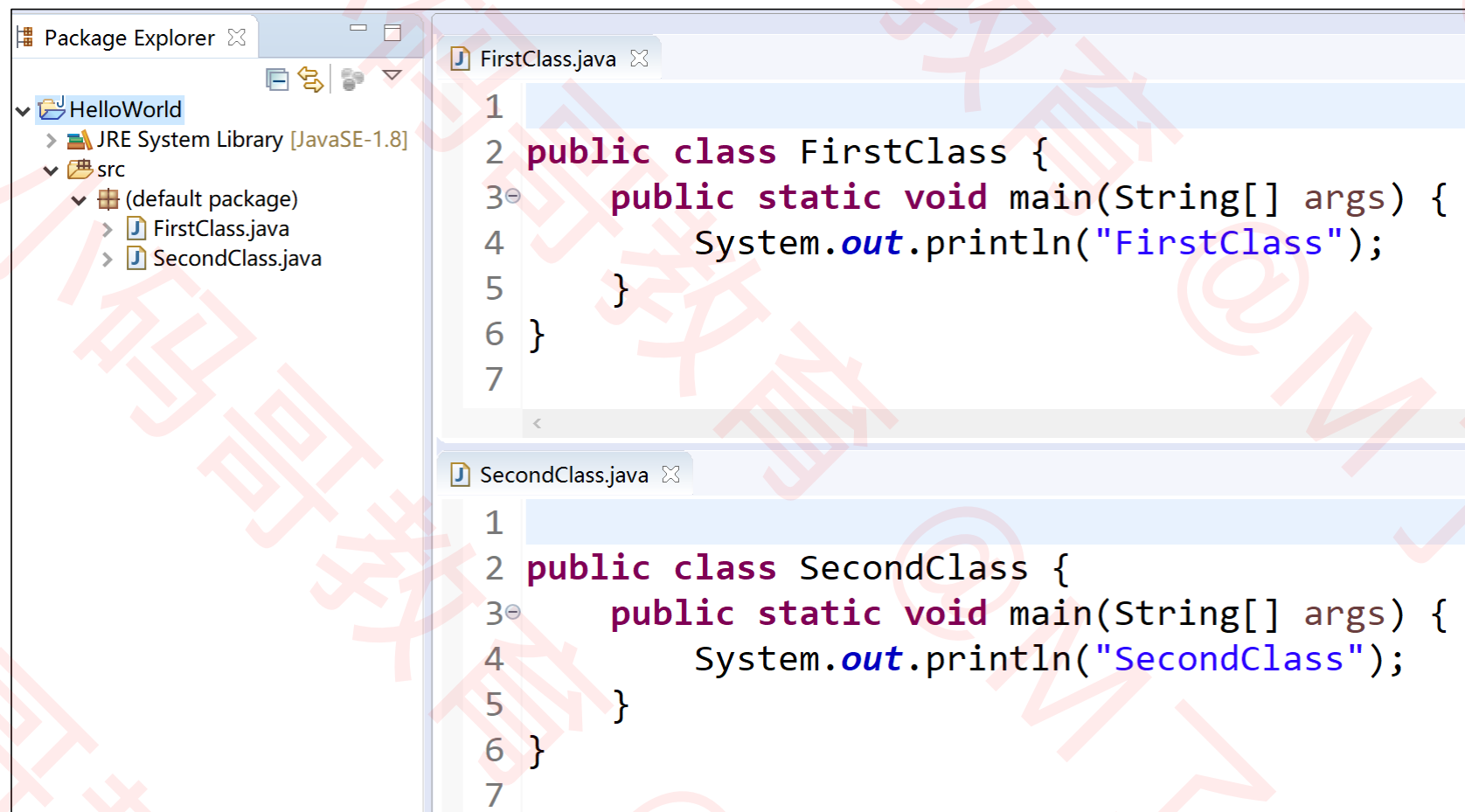
// 不推荐

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

# 代码格式化



- 如果一个 Java 项目中有 2 个不同的 **class**，它们都有自己的 main 方法，那岂不是有 2 个 Java 程序入口？
- 只能选择其中一个入口开始执行程序



The screenshot shows an IDE window with a Package Explorer on the left and two Java source files open on the right. The Package Explorer shows a project named 'HelloWorld' with a 'src' folder containing two files: 'FirstClass.java' and 'SecondClass.java'. The 'FirstClass.java' file is selected, and its code is displayed in the editor. The code defines a public class 'FirstClass' with a public static void main method that prints 'FirstClass' to the console. The 'SecondClass.java' file is also visible in the editor, showing a similar structure with a public class 'SecondClass' and a public static void main method that prints 'SecondClass' to the console.

```
1  
2 public class FirstClass {  
3     public static void main(String[] args) {  
4         System.out.println("FirstClass");  
5     }  
6 }  
7
```

```
1  
2 public class SecondClass {  
3     public static void main(String[] args) {  
4         System.out.println("SecondClass");  
5     }  
6 }  
7
```

# 注释 (Comment)

## ■ Java 的注释有 3 种书写格式

□ 单行注释

□ 多行注释

□ 文档注释 (一种特殊的多行注释)

## ■ 多行注释不能嵌套多行注释

```
/*
这是一段多行注释
/*
这也是一段多行注释
*/
*/
```

**错误示范**

```
// 这句代码的作用是向屏幕输出一段话
System.out.println("这是我的第一个Java程序");
```

**单行注释**

```
/*
这句代码的作用是
向屏幕输出
一段话
*/
System.out.println("这是我的第一个Java程序");
```

**多行注释**

```
/**
 * 计算2个整数的和
 * @param a 第1个整数
 * @param b 第2个整数
 * @return 2个整数的和
 */
public static int add(int a, int b) {
    return a + b;
}
```

**文档注释**

# 数据类型

■ Java 的数据类型主要分为 2 大类

□ 基本类型 (Primitive Type)

✓ **byte**: 8-bit 的整数, 取值范围是  $[-128, 127]$

✓ **short**: 16-bit 的整数, 取值范围是  $[-32768, 32767]$

✓ **int**: 32-bit 的整数, 取值范围是  $[-2^{31}, 2^{31} - 1]$

✓ **long**: 64-bit 的整数, 取值范围是  $[-2^{63}, 2^{63} - 1]$

✓ **float**: 单精度 32-bit IEEE 754 浮点数, 取值范围是  $[1.40E-45F, 3.4028235E38F]$

✓ **double**: 双精度 64-bit IEEE 754 浮点数, 取值范围是  $[4.9E-324, 1.7976931348623157E308]$

✓ **boolean**: 布尔类型, 有 **true**、**false** 两个取值

✓ **char**: 单个 16-bit 的 Unicode 字符

□ 引用类型 (Reference Type)

✓ 引用类型的值是对对象的引用

# 字面量 (Literal)

## ■ 整数

```
// 十进制
byte v1 = 123;
// 二进制 (或者0B11001)
short v2 = 0b11001;
// 十六进制 (或者0XF78A、0Xf78a)
int v3 = 0xF78A;
// 以用L或者l结尾表示long类型 (或者199L)
long v4 = 199L;
```

## ■ 字符和字符串

```
// 用单引号表示字符
char v1 = 'A';
// 用双引号表示字符串
String v2 = "ABCD";
```

## ■ 布尔

```
boolean v1 = true;
boolean v2 = false;
```

## ■ 浮点数

```
// 以F或者f结尾表示float类型 (或者123.4f)
float v1 = 123.4F;
// 以D或者d结尾表示double类型 (或者123.4d)
double v2 = 123.4D;
// 默认就是double类型
double v3 = 123.4;
// 可以用科学计数法 (E或者e)
float v4 = 1.234E2F;
double v5 = 1.234e2;
```

## ■ 空值

```
String string = null;
```



# 转义序列 (Escape Sequences)

- \b (退格, \u0008)
- \t (制表符, \u0009)
- \n (换行, \u000a)
- \f (换页, \u000c)
- \r (回车, \u000d)
- \" (双引号, \u0022)
- \' (单引号, \u0027)
- \\ (反斜杠, \u005c)

# 在数字中使用下划线

- 从 Java 7 开始，可以给数字添加下划线增强可读性

```
int v1 = 1_0000_0000;  
int v2 = 0xFF_EC_DE_5E;  
int v3 = 0b11010010_01101001_10010100_10010010;  
double v4 = 1.23_45_67;  
long v5 = 1_0000_0000;
```

- 下面的用法是错误的

```
// 不能在浮点数的小数点前后使用下划线  
double r1 = 1._23;  
double r2 = 1_.23;  
// 不能在数字的前后使用下划线  
int r3 = _123;  
int r4 = 123_;  
// 不能在X、B、F、D、L、E等特殊字母的前后使用下划线  
byte r5 = 0x_12;  
byte r6 = 0_b10010;  
float r7 = 1.23F_;  
long r8 = 189_L;
```

# 变量的初始化

- 任何变量在使用之前都必须要先初始化（赋值）
- 局部变量：需要程序员手动初始化
- 非局部变量（实例变量、类变量）：编译器会自动给未初始化的变量设置一个初始值

类型	默认初始值
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
char	'\u0000'
boolean	false
对象（引用）	null

# 运算符 (Operator)

运算符	优先级
后缀	<code>expr++ expr--</code>
一元（单目）	<code>++expr --expr +expr -expr ~ !</code>
乘除模	<code>* / %</code>
加减	<code>+ -</code>
位移	<code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code>
关系	<code>&lt; &gt; &lt;= &gt;= instanceof</code>
等价	<code>== !=</code>
按位与	<code>&amp;</code>
按位异或	<code>^</code>
按位或	<code> </code>
逻辑与	<code>&amp;&amp;</code>
逻辑或	<code>  </code>
三元（三目）	<code>? :</code>
赋值	<code>= += -= *= /= %= &amp;= ^=  = &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</code>

- 上面一行的优先级比下面一行高
- 同一行的优先级一样
- 当多个优先级一样的运算符一起使用时
  - 按照结合性进行运算
  - ✓ 只有赋值运算符的结合性是从右至左
  - ✓ 其他运算符的结合性都是从左至右
- 为了保证运算符按照预期执行，尽量多使用小括号
  - 比如 `5 * ((a + b) / c)`
- 算数表达式的结果必须被使用

# 字符串拼接

- 可以使用加号 (+) 进行字符串的拼接

```
int age = 18;  
String name = "Jack";  
double height = 1.78;  
System.out.println(  
    "My name is " + name  
    + ", age is " + age  
    + ", height is " + height);
```

# 位运算

■ >> 与 >>>

□ >> (有符号右移)：最左用符号位补齐

□ >>> (无符号右移)：最左用 0 补齐

```
-128          = 11111111111111111111111111110000000
-128 >> 2     = 11111111111111111111111111111000000
-128 >>> 2    = 00111111111111111111111111111100000
```

■ &、|、^ 也能用在 `boolean` 类型上

□ 对比 &&、||，&、| 少了短路功能

```
System.out.println(true & true); // true
System.out.println(false & true); // false
System.out.println(false & false); // false

System.out.println(true | true); // true
System.out.println(true | false); // true
System.out.println(false | false); // false

System.out.println(true ^ true); // false
System.out.println(false ^ false); // false
System.out.println(true ^ false); // true
```

# 类型转换 (Type Conversion)

## ■ 拓宽基本类型转换 (Widening Primitive Conversion)

□ 数据范围小的转为数据范围大的 (19种), 可以自动转换 (隐式转换)

✓ byte 转 short、int、long、float、double

✓ short 转 int、long、float、double

✓ char 转 int、long、float、double

✓ int 转 long、float、double

✓ long 转 float、double

✓ float 转 double

```
byte b = 12;  
short s = b;  
int i1 = s;  
  
char c = 'A';  
int i2 = c;  
  
long l = i1;  
float f = l;  
double d = f;
```

# 类型转换 (Type Conversion)

## ■ 窄化基本类型转换 (Narrowing Primitive Conversion)

□ 数据范围大的转为数据范围小的 (22种), 可能会丢失精度和范围, 需要强制转换

✓ short 转 byte、char

✓ char 转 byte、short

✓ int 转 byte、short、char

✓ long 转 byte、short、char、int

✓ float 转 byte、short、char、int、long

✓ double 转 byte、short、char、int、long、float

```
short s = 512;  
char c = (char) s;  
byte b = (byte) c;  
  
double d = 1.23;  
float f = (float) d;  
int i = (int) d;
```



# 一元数字提升 (Unary Numeric Promotion)

- 一元数字提升：将 `byte`、`short`、`char` 类型的一元数字自动提升为 `int` 类型（拓宽基本类型转换）
- 下面的情况会执行一元数字提升
  - 数组的索引、创建数组时的数组长度
  - 一元运算符 `+`
  - 一元运算符 `-`
  - 按位取反 (`~`)
  - 位移 (`<<`、`>>`、`>>>`)

```
char c1 = 'A';  
System.out.println(c1); // A  
System.out.println(+c1); // 65  
char c2 = +c1; // error  
char c3 = 65; // ok
```

# 二元数字提升 (Binary Numeric Promotion)

■ 二元数字提升：提升一个或者两个数字（拓宽基本类型转换）

□ 如果任意一个数字是 `double` 类型，那么另一个就会被转换为 `double` 类型

□ 否则，如果任意一个数字是 `float` 类型，那么另一个就会被转换为 `float` 类型

□ 否则，如果任意一个数字是 `long` 类型，那么另一个就会被转换为 `long` 类型

□ 否则，两个数字都被转换为 `int` 类型

■ 下面的情况会执行二元数字提升

□ 乘 (`*`)、除 (`/`)、取余 (`%`)

□ 加法 (`+`)、减法 (`-`)

□ 比较 (`<`、`<=`、`>`、`>=`)

□ 判等 (`==`、`!=`)

□ 位运算 (`&`、`^`、`|`)

□ 三目 (`? :`)

```
byte v1 = 1;
byte v2 = 2;
byte v3 = v1 + v2; // error
byte v4 = v1 + 2; // error
byte v5 = 1 + v2; // error
byte v6 = 1 + 2; // ok
```

```
byte v1 = 1;
v1 = v1 + 2; // error
v1 += 2; // ok (复合赋值运算自带转换)
```

# 关键字 (Keyword)

■ 关键字，也叫做保留字 (reserved word)

■ Java 的关键字如下所示

□ abstract、continue、for、new、switch、assert、default、goto、package、synchronized

□ boolean、do、if、private、this、break、double、implements、protected、throw

□ byte、else、import、public、throws、case、enum、instanceof、return、transient

□ catch、extends、int、short、try、char、final、interface、static、void

□ class、finally、long、strictfp、volatile、const、float、native、super、while

■ 虽然 goto、const 未被使用，但也属于关键字

■ true、false、null 不是关键字，是字面量

# 标识符 (Identifier)

■ 标识符：变量名、方法名、类名等，命名规则如下

- ① 不限长度的 **Java 字母**、**Java 数字** 序列，但必须以 **Java 字母** 开头（区分大小写）
- ② 不能使用关键字
- ③ 不能使用字面量 `true`、`false`、`null`

## ■ Java 字母

□ `Character.isJavaIdentifierStart` 方法返回 `true` 的字符

□ 包括 ASCII 中的 A~Z、a~z，美元符 (\$)，下划线 (\_)，中文，韩文，日文等字符

## ■ Java 字母 或者 Java 数字

□ `Character.isJavaIdentifierPart` 方法返回 `true` 的字符

□ **Java 数字** 包括 ASCII 中的 0~9

# 命名建议

- 变量名、方法名：小驼峰，比如 myNameAndAge
- 类名：大驼峰，比如 MyNameAndAge
- 常量：比如 MY\_NAME\_AND\_AGE

## ■ Java 语言规范

- <https://docs.oracle.com/javase/specs/index.html>
- <https://docs.oracle.com/javase/specs/jls/se13/html/index.html>
- <https://docs.oracle.com/javase/specs/jls/se8/html/index.html>

## ■ Java 虚拟机规范

- <https://docs.oracle.com/javase/specs/jvms/se13/html/index.html>
- <https://docs.oracle.com/javase/specs/jvms/se8/html/index.html>

## ■ Java 教程

- <https://docs.oracle.com/javase/tutorial/java/index.html>

## ■ Java API 文档

- <https://docs.oracle.com/en/java/javase/13/docs/api/index.html>
- <https://docs.oracle.com/javase/8/docs/api/>