

Java 程序设计入门指引

Java Learning

—a study guide for freshmen

学业支持中心

Academic Support Center

编者的话

同学们，你们好！为了帮助大家 JAVA 语言程序设计课程的学习，学业支持中心的志愿者们参考了各大网站相关资源，编写了本册《Java 程序设计入门指引》。

本册指南主要讲解了 Java 语言的编程基础内容和简单的面向对象编程，涵盖了 Java 语言学习初期的主要知识点，推荐辅助教学课程进行预复习。希望本册对你的学习有所帮助。

本《指引》内容选自网上公开信息部分版权归原网站所有，此纸质册子仅供校内学生参考使用，禁止任何形式外传。如有知识不当或错误之处，欢迎读者多多批评指正，以便于我们今后进一步改进。

联系方式

邮件: bjtuwhascwk@163.com

电话: 0631-3806520

地址: 思源东楼 508 室

QQ: 967348142

目录

1.	Java 开发环境安装	1
1.1.	JDK 下载和安装	1
1.2.	IDEA 下载和安装	1
1.3.	第一个 Java 程序	2
2.	变量	5
2.1.	变量类型	5
2.2.	变量的赋值	5
2.3.	算术运算符	6
2.4.	类型转换	6
2.5.	字符与字符串	7
3.	选择结构	8
3.1.	逻辑表达式	8
3.2.	条件语句	8
3.3.	switch 语句	9
4.	循环结构	11
4.1.	while 循环和 do-while 循环	11
4.2.	for 循环	11
5.	数组	13
5.1.	数组的创建	13
5.2.	利用数组的 for 循环	14
6.	方法	15
6.1.	方法的创建	15
6.2.	方法重载	16
7.	类	17
7.1.	类和对象	17
7.2.	类的继承	19
7.3.	方法重写	20

1. JAVA 开发环境安装

1.1. JDK 下载和安装

为了进行 Java 程序开发，JDK 是必不可少的。在官网选择下载安装包时需要确认系统和位数。以 64 位 Windows 系统为例，应当选择 Windows 后下载 x64 MSI Installer。

Java 18

Java 17

Java SE Development Kit 18 downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

Linux

macOS

Windows


Product/file description	File size	Download
x64 Compressed Archive	172.54 MB	https://download.oracle.com/java/18/latest/jdk-18_windows-x64_bin.zip (sha256 🔗)
x64 Installer	153.2 MB	https://download.oracle.com/java/18/latest/jdk-18_windows-x64_bin.exe (sha256 🔗)
x64 MSI Installer	152.08 MB	https://download.oracle.com/java/18/latest/jdk-18_windows-x64_bin.msi (sha256 🔗)

图 1：JDK 下载界面

下载地址：oracle.com/java/technologies/downloads/#jdk18-windows

1.2. IDEA 下载和安装

一款不错的开发工具对开发者是必不可少的。常用的 Java 开发工具如 IDEA、Eclipse 和 Netbeans，本文考虑到高校学生的综合需求推荐使用 IDEA。注意除 IDEA 外的开发工具需要手动设置环境变量。



Version: 2021.3.3
Build: 213.7172.25
17 March 2022
[Release notes](#)

Download IntelliJ IDEA

[Windows](#) [macOS](#) [Linux](#)

Ultimate

For web and enterprise development

[Download](#) [.exe](#)

Free 30-day trial

Community

For JVM and Android development

[Download](#) [.exe](#)

Free, built on open source

图 2：IDEA 下载界面

下载地址：jetbrains.com/idea/download/#section=windows

此处我们作为示范，选择右侧免费的 Community 版本。安装步骤正常进行即可，安装选项界面（Installation Options）的设置推荐全部勾选。

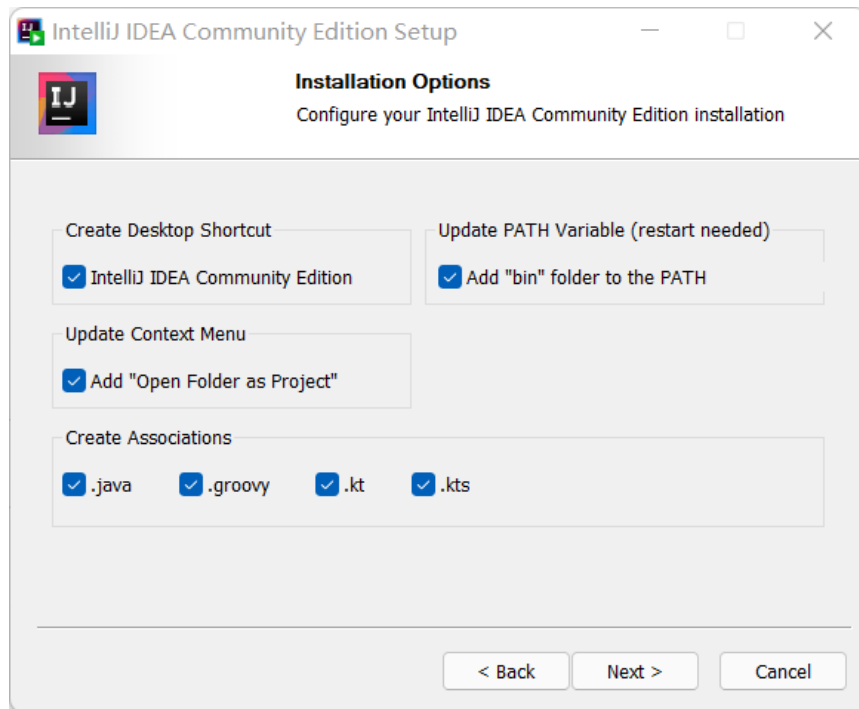


图 3：IDEA 安装界面

1.3. 第一个 JAVA 程序

IDEA 安装完成后，打开即可看到如下的页面：

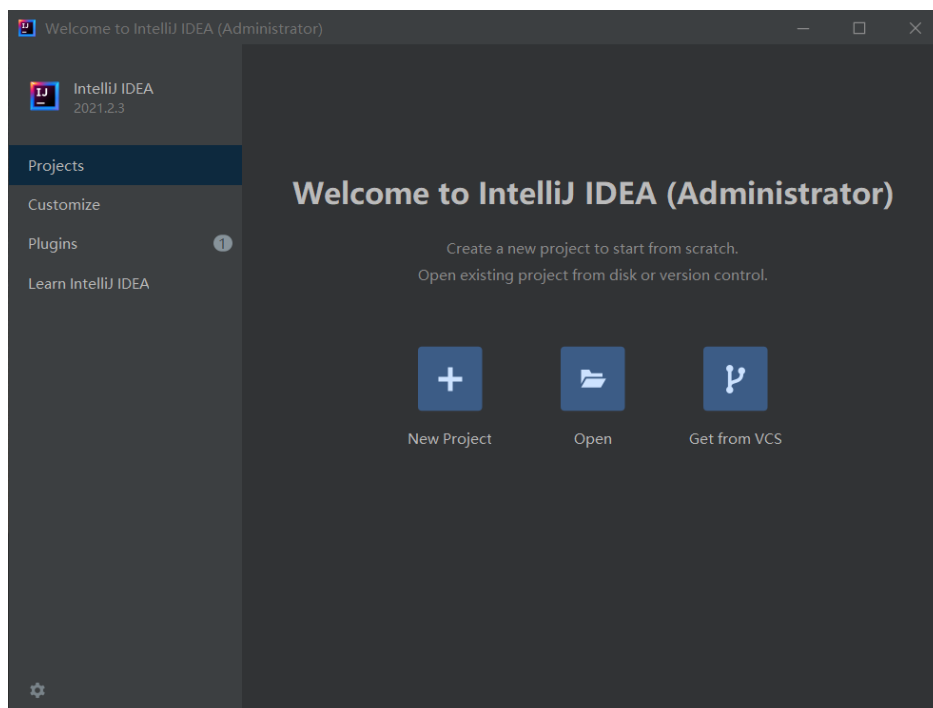


图 4：IDEA 起始页面

要编写我们的第一个 Java 程序，选择 **New Project** 新建项目，其他选项不作改动，选择项目的存放位置和名称即可。

进入新项目，我们需要建立一个 **Java** 类。右击左侧项目文档列表中的 **src** 文件夹，选择新建 **Java Class** 并为之命名即可。

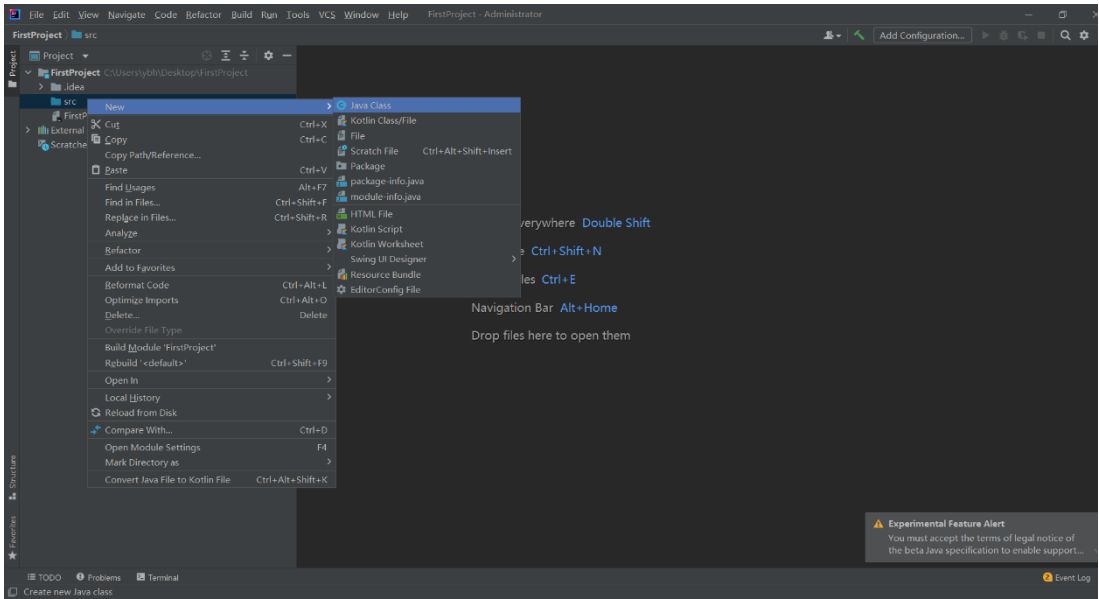


图 5：新建 Java 类的操作

如我们将该类命名为 **HelloWorld**，IDEA 将会在项目 **src** 文件夹下建立一个新的 **.java** 文件并以类名称为其命名。一段代码会被自动生成，这表示这个 **.java** 文件对应我们新建的 **HelloWorld** 类。类名称与 **.java** 文件名称必须相同。

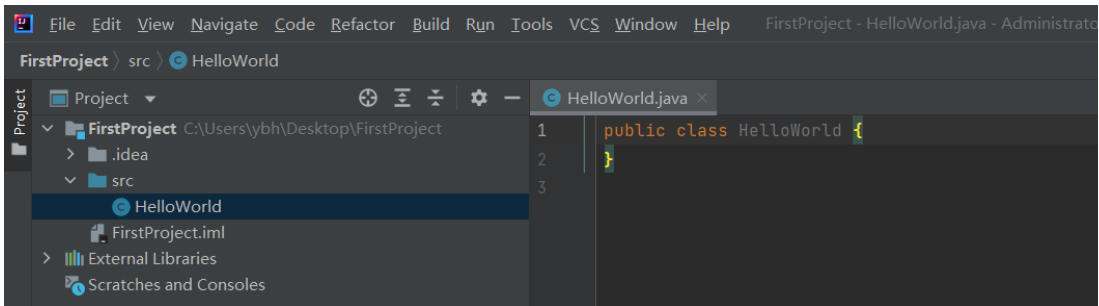


图 6：新建 Java 类完成后的界面

我们输入如下代码：

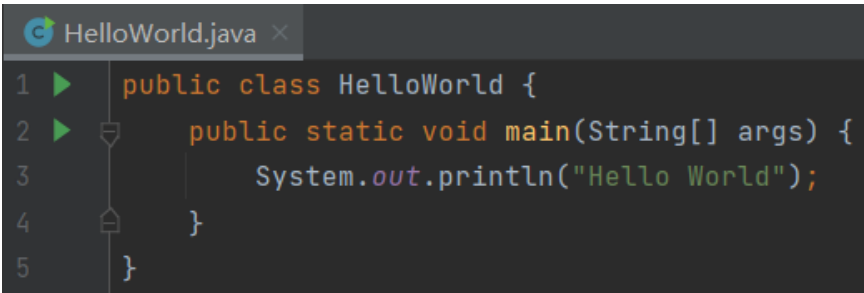


图 7：输出 Hello world 对应的代码

第 2 行为当前类建立了一个 `main` 方法。当我们运行一个 `.java` 文件时，实际运行的是类中的 `main` 方法代码内容；运行代码时遵从从上至下的顺序。

第 3 行的语句表示输出字符串 `"Hello world"`，这将会让我们的电脑呈现对应的内容。后面我们提到输出变量和字符串的地方会经常用到 `System.out.println()`；这一语句，只需将要输出的内容填入这个括号内。注意每个完整的语句都以分号作结尾。

能执行输出任务的语句还有 `System.out.print()`；它与前者的区别是，前者的输出末尾会出现换行；因此前者使用后再用语句输出的内容会在新的一行中，而后者不会。

代码输入完成后，在页面下方选择绿色的运行按钮；或者通过快捷键 `Ctrl+F5` 实现同样的操作，我们就可以直观地看到我们所输入代码的结果了。

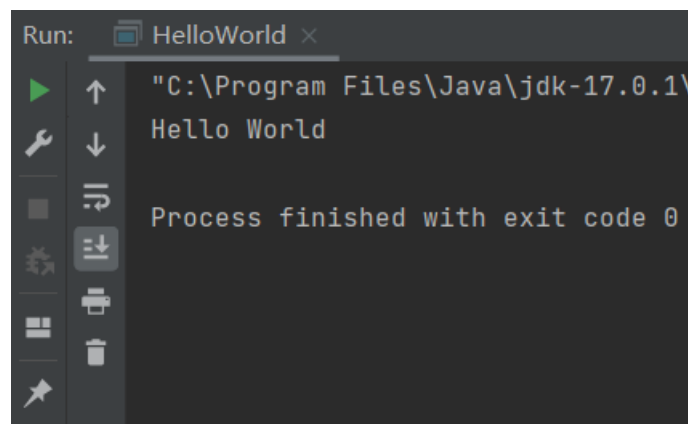


图 8：代码运行后的结果

输出结果为一行 `Hello World`，后面的内容表示程序正常结束。

2. 变量

在编程语言中，为了存储数值，我们需要在电脑内存中申请一块空间。这一行为通常被称为定义一个变量。在定义变量时，我们需要给该变量起一个变量名。

定义变量的语句遵循以下格式：

```
变量类型 变量名;
```

变量名需要遵循由字母、数字、符号\$和下划线组成，且不能以数字开头。字母的大小写被认为是不同的，如 **A** 和 **a** 是两个不同的变量名，可以指代两个不同的变量。

2.1. 变量类型

不同种类的值需要被存储在不同类型的变量中。Java 语言中有八种基本变量类型，其中四种是整数类型，两种是浮点数（即小数）类型，一种字符类型，还有一种布尔型。这些变量类型的区别将在后文逐个解释。

表 1: Java 中的变量类型

类型	对应的值	取值范围	默认值
byte	整数	$-2^7 \sim 2^7-1$	0
short		$-2^{15} \sim 2^{15}-1$	
int		$-2^{31} \sim 2^{31}-1$	
long		$-2^{63} \sim 2^{63}-1$	
float	浮点数		0.0
double			
char	字符	\u0000~\uffff	\u0000
boolean	布尔值	true/false	false

例如，当我们要定义一个浮点数变量 **num**，并将其初始化为 **3.5**，对应的语句如下：

```
double num = 3.5;
```

在变量类型前加关键词 **final**，可使变量成为不可被修改的常量。

```
final double pi = 3.14;
```

2.2. 变量的赋值

在编程语言中，将某一数值赋给某个变量的过程，称为赋值。赋值语句遵循以下格式：

```
变量 = 数值;
```

其中“=”被称为赋值号，而非数学意义上的等号。赋值语句会将赋值号右边的值或表达式的结果，赋给左边的变量。一些赋值语句的示例如下：


```
int a, b = 2, c = 5;
a = b + c; //赋值后变量 a 为表达式 b+c 的结果, 即 7
b = c - a; //赋值后变量 b 的值为-2
```

2.3. 算术运算符

编程语言通常用数字变量和算术运算符组合, 形成一定的算术表达式用来进行运算。这些算术运算符的作用和在数学中基本相同。

表 2: 常见的算术运算符

运算符	名称	例子(其中 A 为 20, B 为 15)
+	加号	A + B 等于 35
-	减号	A - B 等于 5
*	乘号	A * B 等于 300
/	除号	A / B 等于 1
%	模/取余	A % B 等于 5

注意, 对于整数变量之间的运算, 除法结果只取整数商; 例如 7 除以 2 的整数商为 3, 余数为 1。但小数和整数、小数之间的运算结果为小数, 因而它们不能进行取余运算。

一些表达式和赋值语句结合的示例如下:

```
int a = 2, b = 3, c;
c = a * a; //变量 c 的值为 4
b = b + c; //变量 b 的值为 7
c = b % a; //变量 c 的值为 1
```

为了简化变量运算的操作, Java 语言除赋值号外, 还有其他一些赋值运算符。比如加和赋值运算符(+=), 语句 a+=b; 相当于 a=a+b; 类似地, 我们还有 -=、*=、/=、%=。位运算也有相应的赋值运算符, 关于位运算的内容此处不作讲解。

比较特殊的自增、自减运算符(++和--) 用于加在变量前或后, 改变变量的值。例如 a++; 和 ++a; 都相当于 a=a+1。但参与运算时, 如 b=a++; 此时赋给 b 的是 a 改变前的值; 而 b=++a; 赋给 b 的是 a 改变后的值。

2.4. 类型转换

2.3 中所提到的运算都是两个同类变量间的运算。当我们要进行不同变量类型间的运算时, Java 会默认将较低级的变量数值转换为较高级的数值。



图 9: Java 默认变量转换方向

这个变量转换方向是遵循数据损失最小的原则，例如小数转化为整数时会损失小数点后的数据。同样的原因，导致了当我们使用如下代码时：

```
double a = 3.5;
int b = a;
```

编译器会报错，并提示“不兼容的类型：从 double 转换到 int 可能会有损失”。此时，如果我们仍需运行高等级到低等级的赋值操作，需要进行强制类型转换。强制类型转换的方法是在被转换的值或表达式前，加上(变量类型)。如上述代码需要修改为：

```
double a = 3.5;
int b = (int)a; //变量 b 的值为 3
```

注意变量 a 仍为 double 类型，只是赋值时将表达式 a 的结果改为了 int 型。

2.5. 字符与字符串

char 类型变量用于存储单个字符。在 Java 语言中，单个字符用单引号表示。

```
char c = 'x';
```

每个字符都与固定的一个正整数对应，这个对应关系就是 ASCII 表。在数字类型与字符类型强制转换时，便遵照 ASCII 表进行；如 ASCII 表中字符 a 对应的数字是 97，因此

```
System.out.println((int)'a');
```

输出结果也为 97。

注意，字符 '0' 到 '9' 的 ASCII 码依次是 48 到 57。(int)'1' 对应的值不是 1，而是 49。

Java 语言中还提供字符串 String 类，这是由多个字符组成的高级类型。字符串用双引号表示，我们在 1.3 中初次接触 println 方法时就是用它输出了 "Hello world" 这一字符串。

创建一个字符串的方式如下：

```
String str = "Hello world"; //创建字符串 str
System.out.println(str); //输出结果为 Hello world
System.out.println("str"); //输出结果为 str
```

注意辨别此处输出字符串 str 和输出字符串 "str" 的区别。

在 Java 中字符串可以使用加法运算进行连接，如 "a"+"b" 的运算结果为两个字符串连接成的 "ab"。因此我们也能得出 "12"+"3" 的结果为 "123"。

显然字符串加法是不满足交换律的。

3. 选择结构

3.1. 逻辑表达式

逻辑表达式是可被判断真假的表达式，如 $a > 3$ 、 $b < 5$ 等。它们大多由关系运算符组成，表达式的结果是 `true/false` 之一，即布尔值。

表 3：常见的关系运算符

运算符	>	>=	<	<=	==	!=
名称	大于	大于或等于	小于	小于或等于	等于	不等于

复杂的逻辑表达式还包含与、或、非这类逻辑运算符。

表 4：常见的逻辑运算符

运算符	&&		!
名称	逻辑与	逻辑或	逻辑非

用如下的代码举例：

```
int a = 3, b = 5;  
System.out.println(a > 3 && b < 6);
```

由于 $a > 3$ 为假而 $b < 6$ 为真，逻辑表达式的结果为假，输出结果为 `false`。

3.2. 条件语句

如果我们希望一些语句仅在满足某条件时执行，我们可以使用如下格式的 `if` 语句：

```
if (逻辑表达式) {  
    要执行的语句;  
}
```

如果逻辑表达式的结果为 `true`，即对应的条件满足时执行大括号内的语句。当大括号内的语句仅有一句时，大括号可以省略。

如果我们要在满足某条件与不满足时分别执行不同语句，我们可以使用 `if-else` 语句：

```
if (逻辑表达式) {  
    要执行的语句;  
} else {  
    不满足条件执行的语句;  
}
```

以上条件语句可以进行嵌套，即可以在 `if` 或 `else` 中再使用 `if/else` 语句。

```

int n = 7;
if (n >= 10)
    if (n <= 15)
        System.out.println("10~15");
    else
        System.out.println(">15");
else
    if (n >= 5)
        System.out.println("5~9");
    else
        System.out.println("<5");

```

输出结果为 5~9。我们也可以看出，仅有一个 if-else 语句时大括号可以省略。

3.3. SWITCH 语句

当我们要针对一个值的各种情况进行枚举、并分别执行不同的语句时，利用 if-else 嵌套语句我们可以写出如下的代码：

```

char grade = 'C';
if (grade == 'A')
    System.out.println("优秀");
else if (grade == 'B' || grade == 'C')
    System.out.println("良好");
else if (grade == 'D' || grade == 'E')
    System.out.println("及格");
else if (grade == 'F')
    System.out.println("不及格");
else
    System.out.println("未知等级");

```

针对这种特殊情况，Java 语言提供了较为简洁、易于进行修改和维护的 switch 语句。switch 语句的格式如下：

```

switch (表达式) {
    case 值 1:
        //表达式结果为该值对应的语句
        break; //可选
    case 值 2:
        //表达式结果为该值对应的语句
        break; //可选
    ...
    default: //可选
}

```

表达式的结果会依次与 **case** 中的值进行对应，如果相等便进入对应 **case** 中的语句；遇到 **break** 时跳出整个 **switch** 语句。如果没有 **break**，执行完当前 **case** 后会继续执行下一个 **case** 中的语句，无论表达式与其是否相等。

default 用于处理表达式的值与所有 **case** 都不相等的情况，只要遇到就会执行其中的语句；一般放在最后。

本节开始的例子可以用 **switch** 语句改成如下的形式：

```
char grade = 'C';
switch (grade) {
    case 'A':
        System.out.println("优秀");
        break;
    case 'B':
    case 'C':
        System.out.println("良好");
        break;
    case 'D':
    case 'E':
        System.out.println("及格");
        break;
    case 'F':
        System.out.println("不及格");
        break;
    default:
        System.out.println("未知等级");
}
```

4. 循环结构

4.1. WHILE 循环和 DO-WHILE 循环

此前我们所讲的语句都只被执行一次。当我们需要反复执行某个语句时，我们需要使用循环语句。最简单的 **while** 循环结构如下：

```
while (逻辑表达式) {  
    要执行的语句;  
}
```

只要逻辑表达式为 **true**，循环结构内的语句就会执行。如要输出 1~50 的循环：

```
int i = 1;  
while (i <= 50) {  
    System.out.println(i);  
    i++;  
}
```

如果逻辑表达式一开始就不满足，整个循环就不会进行。但有时候我们需要即使不满足条件，也至少执行一次，这时可以使用如下结构的 **do-while** 循环：

```
do {  
    要执行的语句;  
} while (逻辑表达式);
```

do-while 循环会先执行一遍语句，再判断表达式是否为 **true**，如果满足就再执行一次。

4.2. FOR 循环

for 循环的使用可以使 4.1 中的循环更加简洁。它的格式如下：

```
for (初始化;逻辑表达式;更新) {  
    要执行的语句;  
}
```

初始化语句会在最开始执行，仅执行一次；如果满足逻辑表达式，就进行循环，并在一次循环结束时执行更新语句。如要输出 1~50 的循环：

```
for (int i = 1; i <= 50; i++) {  
    System.out.println(i);  
}
```

类似选择语句，循环语句也可以进行嵌套；`if`、`while`、`for` 也可以互相组合嵌套。我们这里展示通过嵌套循环，输出所有和小于 10 的正整数对的程序。

```
for (int i = 1; i < 10; i++)  
    for (int j = 1; i + j < 10; j++)  
        System.out.println(i + " + " + j);
```

注意，`for` 循环括号内定义的变量，其作用范围仅在该循环内；`for` 循环结束后再使用作用范围外的变量会报错。此处嵌套 `for` 循环时，若在外层循环中使用内层循环 `j` 的变量会因此报错。

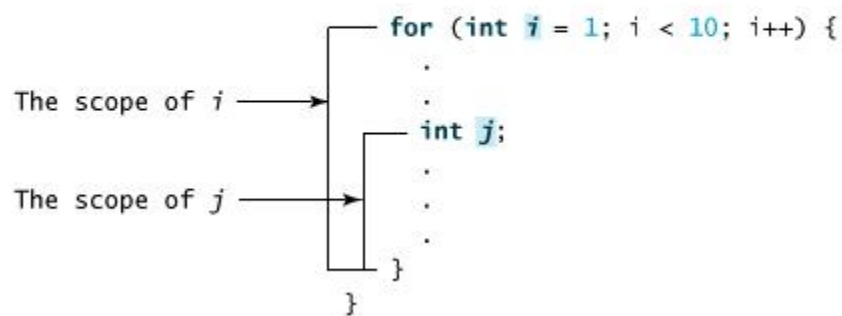


图 10: 变量作用范围的示例

5. 数组

5.1. 数组的创建

数组是编程语言中重要的数据结构。创建一个数组相当于创建多个同类型变量。Java 数组的创建语句遵循以下格式：

```
变量类型[] 数组名 = new 变量类型[数组大小];
```

`new` 操作符的使用是 Java 面向对象的重要特色，在类与对象的讲解中我们会详细介绍。

若定义数组大小为 5 的整数数组，则相当于定义了 5 个整数变量：

```
int[] numbers = new int[5];  
numbers[0] = 47;  
numbers[1] = 63;  
numbers[2] = 18;  
numbers[3] = 26;  
numbers[4] = 40;
```

由此可见，数组的每个元素都是一个变量，通过数组名[索引]的方式访问；索引是从 0 开始的正整数。注意：若定义数组大小为 5，则最后的索引是 4。超出索引范围使用数组会导致程序报错。

我们也可以使用以下的方法创建数组：

```
变量类型[] 数组名 = {值 1, 值 2, 值 3...值 n};
```

大括号内的值将从索引 0 开始依次填入新的数组中。上面的初始化过程可被简化：

```
int[] numbers = {47, 63, 18, 26, 40};
```


5.2. 利用数组的 FOR 循环

Java 语言中引入了一种主要用于数组的加强型 `for` 循环，又称 `for-each` 循环。

```
for (定义循环变量 : 数组名) {  
    要执行的语句;  
}
```

`for-each` 循环需要我们定义一个和数组类型相同的变量，在循环时该变量会逐一取出数组中的值。一个字符串类型数组的例子如下：

```
String[] names = {"Leonard", "Sheldon", "Howard", "Rajesh"};  
for (String name : names) {  
    System.out.print(name);  
    System.out.print(", ");  
} //输出结果为 Leonard, Sheldon, Howard, Rajesh,
```

6. 方法

先前我们经常使用的 `System.out.print()` 和 `System.out.println()` 是两个系统类中的方法，即 Java 语言自带的方法。方法是一段语句，在程序中执行固定的某些功能。如 `print` 和 `println` 就执行输出内容的功能。程序设计中我们也可以使用自定义的方法，从而使得程序变得清晰、有利于程序的维护。

方法的命名要求与变量一致。

6.1. 方法的创建

定义方法遵循以下格式：

```
修饰符 返回值类型 方法名(参数类型 参数 1, 参数类型 参数 2...) {  
    方法执行的语句;  
    return 返回值;  
}
```

程序运行的一般是 `main` 方法。在 `main` 方法中调用其他方法时，需要根据方法的参数类型和数量给出相应的参数；方法的返回值将作为方法的运算结果。

```
public static int max(int num1, int num2) {  
    int res;  
    if (num1 > num2)  
        res = num1;  
    else  
        res = num2;  
    return res;  
}  
public static void main(String[] args) {  
    int num = max(3, 5) + 4;  
}
```

语句中出现自定义方法 `max` 时，先将 3 和 5 分别赋值给对应的参数；在方法中经过相应的运算后，`return` 语句将 `res` 的值返回 `main` 方法，作为 `max(3, 5)` 的计算结果，即 5；最终 `num` 的值为 9。

方法可能没有返回值，也可能没有参数。当方法无返回值时，返回值类型写为 `void`，此时方法通过简单语句使用如 `method();`。

6.2. 方法重载

在同一个类中，允许出现参数个数或参数类型不同的同名函数（仅参数名不同是非法的）。这使得建立函数时对不同参数情况分别讨论成为可能。这些同名函数的返回值类型可以相同也可以不同。

```
public void eat() {  
    System.out.println("eat something");  
}  
public void eat(String food) {  
    System.out.println("eat " + food);  
}
```

7. 类

7.1. 类和对象

类是 **Java** 语言中最重要的概念之一。类是一个模板，它包含了一类对象的状态和行为。例如，将所有的狗作为一个类的话，它的状态有品种、颜色，行为有吃、跑、吠叫。这些状态和行为是所有的狗所共有的。

具体到每条狗，它们的属性各不相同。这里的每条狗都可被称作狗类的一个对象。

在编程语言中，通常把每个对象的状态称为属性，行为称作方法。



图 11: 类和对象的关系示例

通过上图，我们可以建立一个简单的 **Dog** 类，以便我们理解类的概念。

```
public class Dog {  
    String breed, colour; //狗类共有的状态  
    //狗类共有的行为  
    void eat() {  
        System.out.println("eat...");  
    }  
    void run() {  
        System.out.println("run...");  
    }  
    void bark() {  
        System.out.println("bark...");  
    }  
}
```

对象是类的一个实例。为了创建对象，每个类都有构造方法；如果没有定义，Java 会为该类提供一个默认的构造方法。构造方法定义时需要与类同名。可以利用方法重载给一个类设置多个构造方法。

```
public class Dog {  
    private String name;  
    Dog() { //无参数时的构造方法  
        name = "unknown";  
    }  
    Dog(String name) { //有单一字符串参数时的构造方法  
        this.name = name;  
    }  
}
```

此处方法参数和类变量名称都为 `name`，则 `name` 在方法中优先指代参数。为了访问类变量 `name`，需要用 `this` 关键词表示当前类；`this.name` 即当前类中的变量 `name`。

创建类的一个对象时，使用 `new` 关键词并调用相应的构造方法。

```
public class Dog {  
    private String name;  
    Dog(String name) {  
        this.name = name;  
    }  
    public void eat(String food) {  
        System.out.println(name + " eats " + food);  
    }  
}
```

Dog.java

```
public class Main {  
    public static void main(String[] args) {  
        Dog dog = new Dog("Mark");  
        dog.eat("steak");  
    }  
}
```

Main.java

此处的 `dog` 是 `Dog` 类的一个对象，创建时调用了参数为字符串 `"Mark"` 的构造方法。`dog` 对象拥有 `Dog` 类所有的属性（`name`）和方法（`eat`），使用 `dog` 对象的 `eat` 方法后输出结果为一行 `Mark eats steak`。

在实际操作中，类的属性（成员变量）定义时通常加上修饰符 **private**，即仅可被当前类访问；这样杜绝了其他类对这些属性的直接操作和修改，减少了非法操作的可能。此时为得到这些属性通常手动建立访问类型为 **public** 的 **getter/setter** 方法。

```
public class Album {  
    private String title;  
    public String getTitle() {  
        return title;  
    }  
    public void setTitle(String title) {  
        this.title = title;  
    }  
}
```

7.2. 类的继承

继承就是子类继承父类的属性和方法，使得子类具有与父类相同的行为；并且子类可以拥有自己独特的属性和方法，即对父类进行扩展。

继承使用关键词 **extends**。在建立子类时用 **A extends B** 表示 A 继承自 B，即 A 是 B 的一个子类。通过继承，A 类会拥有 B 类中除构造器外非 **private** 的成员变量和成员方法。注意一个类不能继承多个类，即每个类只能有一个父类。

```
public class Animal {  
    public void eat() {  
        System.out.println("eat");  
    }  
}
```

Animal.java

```
public class Bird extends Animal {  
    public void fly() {  
        System.out.println("fly");  
    }  
}
```

Bird.java

Bird 类继承了 Animal 类，因此有 **eat** 和 **fly** 两种方法。

当一个类没有 **extends** 关键词时，默认它继承自 **Object** 类。这是 Java 库中自带的一个类，所有类都继承了 **Object** 中的内容。

就像 **this** 关键词指代当前类一样，**super** 关键词用于指代父类。

```

public class Album {
    private String title;
    public Album(String title) {
        this.title = title;
    }
    public String getTitle() {
        return title;
    }
}

```

Album.java

```

public class DVD extends Album {
    private String director;
    public DVD(String title, String director) {
        super(title);
        this.director = director;
    }
    public void print() {
        System.out.println("Title:" + super.getTitle());
        System.out.println("Director:" + director);
    }
}

```

DVD.java

由于 `title` 的访问权限为 `private`，在子类中不可对其直接进行访问，在 `DVD` 的构造方法中，通过 `super(title)` 调用了其父类的构造方法来完成构造；而在 `print` 方法中则通过其父类访问权限为 `public` 的 `getTitle` 方法得到 `title`。

在类前加关键词 `final`，可将该类声明为不能被继承的最终类。

```

public final class Class {}

```

7.3. 方法重写

在子类中可以对父类方法进行重新编写，此时方法的返回值类型和参数不可变。

```

public class Animal {
    public void eat() {
        System.out.println("eat");
    }
}

```

Animal.java

```
public class Dog extends Animal {  
    public void eat() {  
        System.out.println("eat meat");  
    }  
    public void bark() {  
        System.out.println("bark");  
    }  
}
```

Dog.java

```
public class Main {  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        Animal dog = new Dog();  
        animal.eat();  
        dog.eat();  
    }  
}
```

Main.java

输出结果共两行，第一行调用了 `Animal` 类的方法，输出 `eat`；第二行调用了 `Dog` 类重写的方法，输出 `eat meat`。注意由于 `dog` 对象为 `Animal` 类型，使用 `dog.bark()` 将由于该类型没有 `bark` 方法导致编译错误。

在方法前加关键词 `final`，使得该方法不能被子类重写。

制作人员

• 参与制作名单 •

李雨戈 李明翀

李嘉琦 杨柏翰

杨景雯 蒋瀚祺

• 编审人员名单 •

王一攀 杨柏翰



关注微信公众号

获取更多资讯

