day01-java

Java程序开发运行流程

开发Java程序，需要三个步骤：

1. 编写程序
2. 编译程序
3. 运行程序

HelloWorld案例的编写

新建文本文档文件，修改名称为HelloWorld.java。

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

HelloWorld案例的编译和运行

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

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

public表示公共的，Class代表定义类的关键字， HelloWorld是类名， static 表示为静态的

System.out.println();这行代码的作用是向控制台输出一句话。

public static void **main**(String[] args){} *//这是一个入口方法。*

整个这一块的代码被称为：main方法，也就是说：JVM在执行程序的时候，会主动去找这样一个方法。没有这个规格的方法，程序是无法执行的。

任何一个程序都要有一个入口，没有入口进不来，无法执行。

任何一条java语句必须以";"结尾，并且这个分号还得是英文的，不能用中文分号。

HelloWorld案例常见问题

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

注释

注释是对代码的解释和说明文字，可以提高程序的可读性，因此在程序中添加必要的注释文字十分重要。Java中的 注释分为三种：

```
// 这是单行注释文字

/*
这是多行注释文字
这是多行注释文字
这是多行注释文字
*/
文档注释：文档注释以 /** 开始，以 */ 结束
```

关键字

关键字是指被java语言赋予了特殊含义的单词。关键字的特点：

关键字的字母全部小写。

常用的代码编辑器对关键字都有高亮显示，比如现在我们能看到的public、class、static等。

常量

常量：在程序运行过程中，其值不可以发生改变的量。 Java中的常量分类：

字符串常量 用双引号括起来的多个字符（可以包含0个、一个或多个），例如"a"、"abc"、"中国"等 整数常量

1. **整数**，例如：-10、0、88等
2. **小数常量** 小数，例如：-5.5、1.0、88.88等
3. **字符常量** 用单引号括起来的一个字符，例如：'a'、'5'、'B'、'中'等
4. **布尔常量** 布尔值，表示真假，只有两个值true和false
5. **空常量** 一个特殊的值，空值，值为null 除空常量外，其他常量均可使用输出语句直接输出

```
public class Demo {
public static void main(String[] args) {
System.out.println(10); // 输出一个整数
System.out.println(5.5); // 输出一个小数
System.out.println('A'); // 输出一个字符
System.out.println(true); // 输出boolean值true
System.out.println("欢迎来到我的世界"); // 输出字符串
}
}
```

变量的介绍

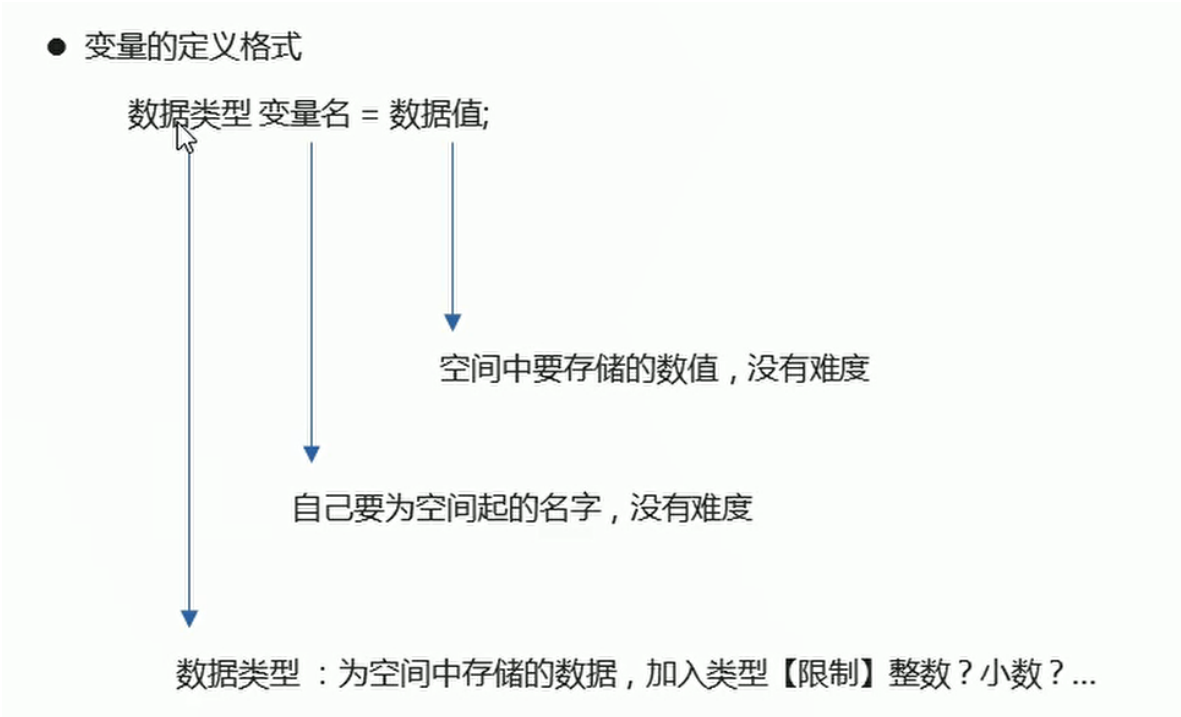
变量的定义格式： 数据类型 变量名 = 数据值;

数据类型：为空间中存储的数据加入类型限制。

变量名：自己要为空间起的名字，没有难度

数据值： 空间中要存储的数值，没有难度

```
数据类型 变量名 = 初始化值; // 声明变量并赋值
int age = 18;
System.out.println(age);
// 先声明，后赋值（使用前赋值即可）
数据类型 变量名;
变量名 = 初始化值;
double money;
money = 55.5;
System.out.println(money);
//变量的修改
int a = 10;
a = 30; //修改变量的值
System.out.println(a);
```



Java中的数据类型

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

- 不同的**数据类型**也分配了不同的**内存空间**，所以它们表示的**数据大小**也是不一样的。

数据类型	关键字	内存占用(字节)	取值范围
整数	byte	1	-128~127
	short	2	-32768~32767
	int	4	-2的31次方到2的31次方-1
	long	8	-2的63次方到2的63次方-1
浮点数	float(单精度)	4	1.401298e-45到3.402823e+38
	double(双精度)	8	4.9000000e-324到1.797693e+308
字符	char	2	0-65535
布尔	boolean	1	true，false

说明：e+38表示是乘以10的38次方，同样，e-45表示乘以10的负45次方

键盘录入

我们可以通过 Scanner 类来获取用户的输入。使用步骤如下：

1. 导包。Scanner 类在java.util包下，所以需要将该类导入。导包的语句需要定义在类的上面
2. 创建Scanner对象。
3. 接收数据

```
import java.util.Scanner;//导入包
public class ScannerDemo {
public static void main(String[] args) {
//创建对象
Scanner sc = new Scanner(System.in);// 创建Scanner对象，sc表示变量名，其他均不可变
//接收数据
int a = sc.nextInt();// 表示将键盘录入的值作为int数返回。
//输出数据
System.out.println(a);
}
}
```

标识符

标识符是用户编程时使用的名字，用于给类、方法、变量、常量等命名。

Java中标识符的组成规则：

1. 由字母、数字、下划线“_”、美元符号“$”组成，第一个字符不能是数字。
2. 不能使用java中的关键字作为标识符。
3. 标识符对大小写敏感（区分大小写）。

常见命名约定

小驼峰命名法：方法、变量

- 约定1：标识符是一个单词的时候，首字母小写
- 范例1：**name**
- 约定2：标识符由多个单词组成的时候，第一个单词首字母小写，其他单词首字母大写
- 范例2：**firstName**

大驼峰命名法：类

- 约定1：标识符是一个单词的时候，首字母大写
- 范例1：**Student**
- 约定2：标识符由多个单词组成的时候，每个单词的首字母大写
- 范例2：**GoodStudent**

类型转换

隐式转换

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

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

强制转换

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

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

运算符和表达式

运算符：对常量或者变量进行操作的符号

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

+：是运算符，并且是算术运算符。

a + b：是表达式，由于+是算术运算符，所以这个表达式叫算术表达式。

Java 中 + 操作符的优先级大于 ==

整数操作只能得到整数，要想得到小数，必须有浮点数参与运算。

字符的“+”操作

char类型参与算术运算，使用的是计算机底层对应的十进制数值。

需要我们记住三个字符对应的数值：

'a' -- 97 a-z是连续的，所以'b'对应的数值是98，'c'是99，

依次递加 'A' -- 65 A-Z是连续的，

所以'B'对应的数值是66，'C'是67，

依次递加 '0' -- 48 0-9是连续的，所以'1'对应的数值是49，'2'是50，依次递加

提升规则：

byte类型，short类型和char类型将被提升到int类型，不管是否有其他类型参与运算。

整个表达式的类型自动提升到与表达式中最高等级的操作数相同的类型

等级顺序：byte,short,char --> int --> long --> float --> double

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

逻辑运算符

& 逻辑与 a&b，a和b都是true，结果为true，否则为false

| 逻辑或 a|b，a和b都是false，结果为false，否则为true

^ 逻辑异或 a^b，a和b结果不同为true，相同为false

! 逻辑非 !a，结果和a的结果正好相反

&& 短路与 作用和&相同，但是有短路效果

|| 短路或 作用和|相同，但是有短路效果

> 1. **逻辑与&，无论左边真假，右边都要执行。**
> 2. **短路与&&，如果左边为真，右边执行；如果左边为假，右边不执行。**
> 3. **逻辑或|，无论左边真假，右边都要执行。**
> 4. **短路或||，如果左边为假，右边执行；如果左边为真，右边不执行。**

三元运算符语法格式：

关系表达式？表达式1：表达式2;

如果表达式为真执行表达式1，否则为表达式2

分支结构之if语句

```
格式：
if (关系表达式) {
语句体;
}

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

}else{

}
```

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

分析：

①小明的考试成绩未知，可以使用键盘录入的方式获取值

②由于奖励种类较多，属于多种判断，采用if...else...if格式实现

③为每种判断设置对应的条件

④为每种判断设置对应的奖励

```
import java.util.Scanner;
public class IfTest02 {
public static void main(String[] args){
// 1. 使用Scanner录入考试成绩
Scanner sc = new Scanner(System.in);
System.out.println("请输入您的成绩:");// 录入的格式为int类型
int score = sc.nextInt();
// 2. 判断成绩是否在合法范围内 0~100
if(score >=0 && score <= 100){
// 合法成绩
// 3. 在合法的语句块中判断成绩范围符合哪一个奖励
if(score >= 95 && score <= 100){
System.out.println("自行车一辆");
}else if(score >= 90 && score <= 94){
System.out.println("游乐场一次");
}else if(score >= 80 && score <= 89){
System.out.println("变形金刚一个");
}else {
System.out.println("挨顿揍，这座城市又多了一个伤心的人~");
}
}else{
// 非法的话，给出错误提示
System.out.println("您的成绩输入有误!");
```

```
        }
    }
}
```

分支语句switch语句

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

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

```
public static void main(String[] args){
// 1. 键盘录入星期数据，使用变量接收
//Scanner是一个类，nextDouble()是Scanner的成员函数，
//System.in作为参数传递给Scanner的构造函数，使Scanner用键盘作为输入，然后用new在内存中实例化一个Scanner出来，使得其它变量能调用这块内存区。
Scanner sc = new Scanner(System.in);
System.out.println("请输入");
int week = sc.nextInt();
// 2. 多情况判断，采用switch语句实现
switch(week){
// 3. 在不同的case中，输出对应的减肥计划
case 1:
System.out.println("跑步");
break;
case 2:
System.out.println("游泳");
break;
case 3:
System.out.println("慢走");
break;
case 4:
System.out.println("动感单车");
break;
case 5:
System.out.println("拳击");
break;
case 6:
System.out.println("爬山");
```

```
break;
case 7:
System.out.println("好好吃一顿");
break;
default:
System.out.println("您的输入有误");
break;
        }
    }
}
```

如果switch语句中,case省略了break语句, 就会开始case穿透.

现象： 当开始case穿透，后续的case就不会具有匹配效果，内部的语句都会执行 直到看见break，或者将整体switch语句执行完毕，才会结束。

循环语句-for循环

```
for (初始化语句;条件判断语句;条件控制语句) {
循环体语句;
}
//初始化语句： 用于表示循环开启时的起始状态，简单说就是循环开始的时候什么样

//条件判断语句：用于表示循环反复执行的条件，简单说就是判断循环是否能一直执行下去

//循环体语句： 用于表示循环反复执行的内容，简单说就是循环反复执行的事情

//条件控制语句：用于表示循环执行中每次变化的内容，简单说就是控制循环是否能执行下去
```

执行流程：

1. ①执行初始化语句
2. ②执行条件判断语句，看其结果是true还是false 如果是false，循环结束 如果是true，继续执行
3. ③执行循环体语句
4. ④执行条件控制语句 ⑤回到②继续

for循环案例-求1-100偶数和

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

```
    }
}
```

for循环案例-水仙花数

水仙花数，指的是一个三位数，个位、十位、百位的数字立方和等于原数

例如 153 3*3*3 + 5*5*5 + 1*1*1 = 153

```

```

循环语句-while循环

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

while循环执行流程：

①执行初始化语句

②执行条件判断语句，看其结果是true还是false 如果是false，循环结束 如果是true，继续执行

③执行循环体语句

④执行条件控制语句 ⑤回到②继续

循环语句-dowhile循环

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

执行流程：

① 执行初始化语句

② 执行循环体语句

③ 执行条件控制语句

④ 执行条件判断语句，看其结果是true还是false 如果是false，循环结束 如果是true，继续执行

⑤ 回到②继续

三种循环的区别

三种循环的区别

for循环和while循环先判断条件是否成立，然后决定是否执行循环体（先判断后执行）

do...while循环先执行一次循环体，然后判断条件是否成立，是否继续执行循环体（先执行后判断） for循环和while的区别

条件控制语句所控制的自增变量，因为归属for循环的语法结构中，在for循环结束后，就不能再次被访问到了

条件控制语句所控制的自增变量，对于while循环来说不归属其语法结构中，在while循环结束后，该变量还可以继续使用

死循环（无限循环）的三种格式

1. for(;;){}
2. while(true){}
3. 3. do {} while(true);

```
for死循环格式 :
for(;;){
}
while死循环格式 :
while(true){
}
do..while死循环格式 :
do{
}while(true);
```

跳转控制语句（break）

跳出循环，结束循环

跳转控制语句（continue）

跳过本次循环，继续下次循环

注意： continue只能在循环中进行使用!

Random产生随机数

Random类似Scanner，也是Java提供好的API，内部提供了产生随机数的功能

API后续课程详细讲解，现在可以简单理解为Java已经写好的代码

使用步骤:

1. 导入包 import java.util.Random; //
2. 创建对象 Random r = new Random(); // 创建随机数对象
3. 产生随机数 int num = r.nextInt(10); // 产生随机数（10）--0—9之间

   解释： 10代表的是一个范围，如果括号写10，产生的随机数就是0-9，括号写20，参数的随机数则是0-19

Random练习-猜数字（应用）

需求:

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

Random : 产生随机数

1. 导包 : import java.util.Random; 导包的动作必须出现在类定义的上面
2. 创建对象 : **Random r = new Random();** 上面这个格式里面，r 是变量名，可以变，其他的都不允许变
3. 获取随机数 : int number = r.nextInt(10);

   //获取数据的范围：[0,10) 包括0,不包括10 上面这个格式里面，number是变量名，可以变，数字10可以变。其他的都不允许变

数组

数组介绍

数组就是存储数据长度固定的容器，存储多个数据的数据类型要一致。

第一种格式： 数据类型[] 数组名

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

第二种格式 数据类型 数组名[]

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

数组的动态初始化

动态初始化格式

```
//数据类型[] 数组名 = new 数据类型[数组长度];
int[] arr = new int[3];
```

等号左边:

int:数组的数据类型

[]:代表这是一个数组

arr:代表数组的名称

等号右边:

new:为数组开辟内存空间

int:数组的数据类型

[]:代表这是一个数组

5:代表数组的长度

访问数组元素格式

数组元素访问

什么是索引

每一个存储到数组的元素，都会自动的拥有一个编号，从0开始。 这个自动编号称为数组索引(index)，可以通过数组的索引访问到数组中的元素。

访问数组元素格式

数组名[索引];

内存分配

1.5 内存分配

1.5.1 内存概述

内存是计算机中的重要原件，临时存储区域，作用是运行程序。

我们编写的程序是存放在硬盘中的，在硬盘中的程序是不会运行的。

必须放进内存中才能运行，运行完毕后会清空内存。

Java虚拟机要运行程序，必须要对内存进行空间的分配和管理。

1.5.2 java中的内存分配

- 目前我们只需要记住两个内存，分别是：栈内存和堆内存

区域名称	作用
寄存器	给CPU使用，和我们开发无关。
本地方法栈	JVM在使用操作系统功能的时候使用，和我们开发无关。
方法区	存储可以运行的class文件。
堆内存	存储对象或者数组，new来创建的，都存储在堆内存。
方法栈	方法运行时使用的内存，比如main方法运行，进入方法栈中执行。

数组的静态初始化

什么是静态初始化 在创建数组时，直接将元素确定

静态初始化格式

完整版格式

数据类型[] 数组名 = new 数据类型[]{元素1,元素2,...};

简化版格式

数据类型[] 数组名 = {元素1,元素2,...};

数组操作的两个常见问题

索引越界异常

出现原因

数组长度为3，索引范围是0~2，但是我们却访问了一个3的索引。 程序运行后，将会抛出 ArrayIndexOutOfBoundsException 数组越界异常。在开发中，数组的越界异常是不 能出现的，一旦出现了，就必须要修改我们编写的代码。

解决方案

将错误的索引修改为正确的索引范围即可！

空指针异常 出现原因

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

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

解决方案 给数组一个真正的堆内存空间引用即可！

数组遍历

数组遍历：就是将数组中的每个元素分别获取出来，就是遍历。遍历也是数组操作中的基石。

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

数组基本查找【应用】

```
public static void main(String[] args) {
// 1.定义一个数组，用静态初始化完成数组元素的初始化
int[] arr = {19, 28, 37, 46, 50};
// 2.键盘录入要查找的数据，用一个变量接收
Scanner sc = new Scanner(System.in);
System.out.println("请输入您要查找的元素:");
int num = sc.nextInt();
// 3.定义一个索引变量，初始值为-1
// 假设要查找的数据，在数组中就是不存在的
int index = -1;
// 4.遍历数组，获取到数组中的每一个元素
for (int i = 0; i < arr.length; i++) {
// 5.拿键盘录入的数据和数组中的每一个元素进行比较，如果值相同，就把该值对应的索引赋
```

```
值给索引变量，并结束循环
if(num == arr[i]){
// 如果值相同，就把该值对应的索引赋值给索引变量，并结束循环
index = i;
break;
}
}
// 6.输出索引变量
System.out.println(index);
}
}
```

方法概述

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

注意：

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

无参数方法定义和调用

```
public static void 方法名 ( ) {
// 方法体;
}
```

调用格式：

```
方法名();
```

带参数方法定义和调用

```
public static void 方法名 (参数1) {
方法体;
}
public static void 方法名 (参数1, 参数2, 参数3...) {
方法体;
}
```

方法定义时，参数中的数据类型与变量名都不能缺少，缺少任意一个程序将报错

方法调用时，参数的数量与类型必须与方法定义中的设置相匹配，否则程序将报错

带返回值方法的定义和调用

```
public static 数据类型 方法名 ( 参数 ) {
return 数据 ;
}
// eg
public static boolean isEvenNumber( int number ) {
return true ;
}
public static int getMax( int a, int b ) {
return 100 ;
}
```

方法定义时return后面的返回值与方法定义上的数据类型要匹配，否则程序将报错

带返回值方法的练习-求两个数的最大值

```
/*
需求：设计一个方法可以获取两个数的较大值，数据来自于参数
1. 定义一个方法，声明两个形参接收计算的数值，求出结果并返回
2. 使用 if 语句 得出 a 和 b 之间的最大值，根据情况return具体结果
3. 在main()方法中调用定义好的方法并使用 【 变量保存 】
*/
public static void main(String[] args) {
// 3. 在main()方法中调用定义好的方法并使用 【 变量保存 】
System.out.println(getMax(10,20)); // 输出调用
int result = getMax(10,20);
System.out.println(result);
for(int i = 1; i <= result; i++){
System.out.println("HelloWorld");
}
}
// 方法可以获取两个数的较大值
public static int getMax(int a, int b){
if(a > b){
return a;
}else{
return b;
}
}
}
```

方法的通用格式

```
public static 返回值类型 方法名(参数) {
方法体;
return 数据 ;
}
//方法不能嵌套定义
//void表示无返回值，可以省略return，也可以单独的书写return，后面不加数据
public class MethodDemo {
public static void main(String[] args) {
}
public static void methodOne() {
public static void methodTwo() {
// 这里会引发编译错误!!!
```

```
}
}
}
```

方法重载

方法重载概念

方法重载指同一个类中定义的多个方法之间的关系，满足下列条件的多个方法相互构成重载

多个方法在同一个类中

多个方法具有相同的方法名

多个方法的参数不相同，类型不同或者数量不同

注意：

重载仅对应方法的定义，与方法的调用无关，调用方式参照标准格式

重载仅针对同一个类中方法的名称与参数进行识别，与返回值无关，换句话说不能通过返回值来判定两个方法是否相互构成重载

```
public class MethodDemo {
public static void fn(int a) {
//方法体
}
public static int fn(double a) {
//方法体
}
}
public class MethodDemo {
public static float fn(int a) {
//方法体
}
public static int fn(int a , int b) {
//方法体
}
}
```

方法重载练习

```
public class MethodTest {
public static void main(String[] args) {
//调用方法
System.out.println(compare(10, 20));
System.out.println(compare((byte) 10, (byte) 20));
System.out.println(compare((short) 10, (short) 20));
System.out.println(compare(10L, 20L));
}
//int
public static boolean compare(int a, int b) {
System.out.println("int");
return a == b;
}
//byte
public static boolean compare(byte a, byte b) {
System.out.println("byte");
```

```
return a == b;
}
//short
public static boolean compare(short a, short b) {
System.out.println("short");
return a == b;
}
//long
public static boolean compare(long a, long b) {
System.out.println("long");
return a == b;
}
}
```

方法的参数传递

```
package com.itheima.param;
public class Test1 {
/*
方法参数传递为基本数据类型 :
传入方法中的, 是具体的数值.
*/
public static void main(String[] args) {
int number = 100;
System.out.println("调用change方法前:" + number);
change(number);
System.out.println("调用change方法后:" + number);
}
public static void change(int number) {
number = 200;
}
}
```

结论:

基本数据类型的参数，形式参数的改变，不影响实际参数

结论依据:

每个方法在栈内存中，都会有独立的栈空间，方法运行结束后就会弹栈消失