

1.1 什么是方法

- **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, 该过程称为方法定义
- 方法创建后并不是直接运行的, 需要手动使用后才执行, 该过程称为方法调用

2.1 方法定义

- 格式:

```
public static void 方法名() {  
    //方法体  
}
```
- 范例:

```
public static void isEvenNumber() {  
    //方法体  
}
```

2.2 方法调用

- 格式: `方法名();`
- 范例: `isEvenNumber();`

注意:

- 方法必须先定义后调用, 否则程序将报错

2.3 方法调用过程

```
public class MethodDemo {  
    public static void main(String[] args) {  
        isEvenNumber();  
    }  
  
    //定义一个变量, 判断该数据是否是偶数  
    public static void isEvenNumber() {  
        int number = 10;  
        if (number % 2 == 0) {  
            System.out.println(true);  
        } else {  
            System.out.println(false);  
        }  
    }  
}
```

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

```
public static void isEvenNumber() {  
    int number = 10;  
    if (number % 2 == 0) {  
        System.out.println(true);  
    } else {  
        System.out.println(false);  
    }  
}
```

方法的调用都是在栈内存中, 方法被调用时进栈, 调用完毕后出栈

3.1 带参数方法定义

- 格式：

```
public static void 方法名 ( 参数 ) { ... }
```
- 格式 (单个参数) :

```
public static void 方法名 ( 数据类型 变量名 ) { ... }
```
- 范例 (单个参数) :

```
public static void isEvenNumber( int number ) { ... }
```
- 格式 (多个参数) :

```
public static void 方法名 ( 数据类型 变量名1, 数据类型 变量名2, ..... ) { ... }
```
- 范例 (多个参数) :

```
public static void getMax( int number1 , int number2 ) { ... }
```

注意:

- 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
- 方法定义时, 多个参数之间使用逗号(,)分隔

3.2 带参数方法调用

- 格式:

```
方法名 ( 参数 );
```
- 格式 (单个参数) :

```
方法名 ( 变量名/常量值 );
```
- 范例 (单个参数) :

```
isEvenNumber ( 5 );
```
- 格式 (多个参数) :

```
方法名 ( 变量名1/常量值1, 变量名2/常量值2 );
```
- 范例 (多个参数) :

```
getMax ( 5, 6 );
```

注意:

- 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

3.3 形参和实参

形参: 方法定义中的参数

等同于变量定义格式, 例如: `int number`

实参: 方法调用中的参数

等同于使用变量或常量, 例如: `10 number`

```
public class MethodDemo02 {  
    public static void main(String[] args) {  
        //直接传递常量  
        isEvenNumber(10);  
        //定义变量, 传递  
        int number = 10;  
        isEvenNumber(number);  
    }  
    //接收一个变量, 判断该数据是否是偶数  
    public static void isEvenNumber(int number) {  
        if (number % 2 == 0) {  
            System.out.println(true);  
        } else {  
            System.out.println(false);  
        }  
    }  
}
```

3.4 带参数方法练习

需求：设计一个方法用于打印两个数中的较大数，数据来自于方法参数
思路：

① 定义一个方法，用于打印两个数字中的较大数，例如getMax()

```
public static void getMax() {  
}
```

② 为方法定义两个参数，用于接收两个数字

```
public static void getMax(int a, int b) {  
}
```

③ 使用分支语句分两种情况对两个数字的大小关系进行处理

```
if(a > b) {  
    System.out.println(a);  
}else {  
    System.out.println(b);  
}
```

④ 在main()方法中调用定义好的方法（使用常量）

```
public static void main(String[] args) {  
    //直接传递常量  
    getMax(10, 20);  
}
```

⑤ 在main()方法中调用定义好的方法（使用变量）

```
public static void main(String[] args) {  
    //定义变量，传递  
    int a = 10;  
    int b = 20;  
    getMax(a, b);  
}
```

4.1 带返回值方法定义

● 格式：

```
public static 数据类型    方法名( 参数){  
    return 数据;  
}
```

● 范例1：

```
public static boolean isEvenNumber( int number ) {  
    return true;  
}
```

● 范例2：

```
public static int    getMax( int a, int b ) {  
    return 100;  
}
```

注意：

● 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

5.1 方法注意事项

- 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 method() {  
        //代码片段  
    }  
}
```



```
public class MethodDemo {  
    public static void method() {  
        //代码片段  
        return;  
    }  
}
```



```
public class MethodDemo {  
    public static void method() {  
        //代码片段  
        return 100;  
    }  
}
```



方

法

也

4.3 带返回值方法练习

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

思路：

- ① 定义一个方法，用于获取两个数字中的较大数

```
public static int getMax(int a, int b) {  
}
```

- ② 使用分支语句分两种情况对两个数字的大小关系进行处理

```
if(a > b) {  
} else {  
}
```

- ③ 根据题设分别设置两种情况下对应的返回结果

```
if (a > b) {  
    return a;  
} else {  
    return b;  
}
```

- ④ 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) {  
    int result = getMax(10, 20);  
    System.out.println(result);  
}
```

- ⑤ 在main()方法中调用定义好的方法并直接打印结果

```
public static void main(String[] args) {  
    // 输出调用  
    System.out.println(getMax(10, 20));  
}
```

5.2 方法的通用格式

- 格式：

```
public static 返回值类型 方法名(参数) {  
    方法体;  
    return 数据;  
}
```

- public static 修饰符，目前先记住这个格式
- 返回值类型 方法操作完毕之后返回的数据的数据类型
如果方法操作完毕，没有数据返回，这里写void，而且方法体中一般不写return
- 方法名 调用方法时候使用的标识
- 参数 由数据类型和变量名组成，多个参数之间用逗号隔开
- 方法体 完成功能的代码块
- return 如果方法操作完毕，有数据返回，用于把数据返回给调用者

5.2 方法的通用格式

- 格式：

```
public static 返回值类型 方法名(参数) {  
    方法体;  
    return 数据;  
}
```

- 定义方法时，要做到两个明确
明确返回值类型：主要是明确方法操作完毕之后是否有数据返回，如果没有，写void；如果有，写对应的数据类型
明确参数：主要是明确参数的类型和数量
- 调用方法时
void类型的方法，直接调用即可
非void类型的方法，推荐用变量接收调用

6.3 方法重载练习

需求：使用方法重载的思想，设计比较两个整数是否相同的方法，兼容全整数类型（byte,short,int,long）

思路：

- ① 定义比较两个数字的是否相同的方法compare()方法，参数选择两个int型参数

```
public static boolean compare(int a, int b) {  
    return a == b;  
}
```

- ② 定义对应的重载方法，变更对应的参数类型，参数变更为两个long型参数

```
public static boolean compare(long a, long b) {  
    return a == b;  
}
```

- ③ 定义所有的重载方法，两个byte类型与两个short类型参数

```
public static boolean compare(byte a, byte b) {  
    //代码片段  
}  
public static boolean compare(short a, short b) {  
    //代码片段  
}
```

- ④ 完成方法的调用，测试运行结果

```
public static void main(String[] args) {  
    System.out.println(compare(10, 20));  
}
```

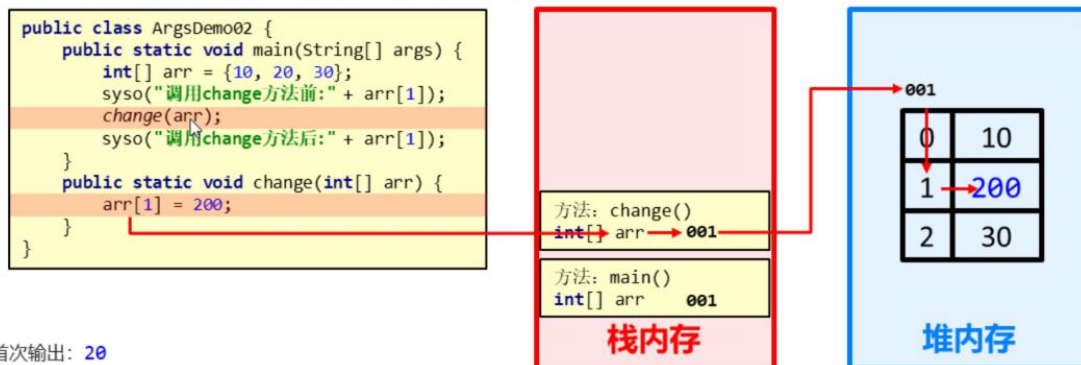
默认输出 int 类型的参数，byte.short 类型需要强制转化，long 类型的需要在参数后加“L”

```
package com.itheima_06;  
  
/*  
    对于基本数据类型的参数，形式参数的改变，不影响实际参数的值  
*/  
public class ArgsDemo01 {  
    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;  
    }  
}
```

对于基本数据类型的参数，形式参数的改变，不影响实际参数的值

7.2 方法参数传递(引用类型)

对于引用类型的参数，形式参数的改变，影响实际参数的值



首次输出：20

对于引用数据类型的参数，形式参数的改变，影响实际参数的值

思路：

- ① 因为要求结果在一行上输出，所以这里需要在学习一个新的输出语句`System.out.print(“内容”)`;

`System.out.println(“内容”)`;输出内容并换行

`System.out.print(“内容”)`;输出内容不换行

`System.out.println()`;起到换行的作用