1. 方法概述

1.1 方法的概念

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

2. 方法的定义和调用

2.1 无参数方法定义和调用

- 定义格式：

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

- 范例：

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

- 调用格式：

```
方法名();
```

- 范例：

```
method();
```

- 注意：

  方法必须先定义，后调用，否则程序将报错

2.2 方法的调用过程

- 总结：每个方法在被调用执行的时候，都会进入栈内存，并且拥有自己独立的内存空间，方法内部代码调用完毕之后，会从栈内存中弹栈消失。

2.3 方法练习-奇偶数判断

- 需求：判断一个数是奇数还是偶数
- 代码：

```
public class Demo1Method {
    /*

        带参数方法的定义格式：
                public static void 方法名  （ 参数 ）  { … … }
                public static void 方法名  （ 数据类型 变量名 ）  { … … }

        带参数方法的调用格式：
                方法名 （ 参数 ）；
                方法名 （ 变量名/常量值 ）；

        tips：参数可以是一个，也可以是多个.

        需求：判断一个数是奇数还是偶数
     */
    public static void main(String[] args) {
        isEvenNumber(10);
    }

    public static void isEvenNumber(int num){
        if(num % 2 == 0){
            System.out.println("偶数");
        }else{
            System.out.println("奇数");
        }
    }
}
```

3. 带参数方法的定义和调用

3.1 带参数方法定义和调用

- 定义格式：

  参数：由数据类型和变量名组成 - 数据类型 变量名

  参数范例：int a

  ```
  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;
  }
  ```

- 范例：

```
public static void isEvenNumber(int number){
    ...
}
public static void getMax(int num1, int num2){
    ...
}
```

- 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
  ```

- 调用格式：

```
方法名(参数);

方法名(参数1,参数2);
```

- 范例：

```
isEvenNumber(10);

getMax(10,20);
```

  - 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

3.2 形参和实参

1. 形参：方法定义中的参数

等同于变量定义格式，例如：int number

2. 实参：方法调用中的参数

等同于使用变量或常量，例如： 10 number

3.3 带参数方法的练习-打印n-m之间所有的奇数

- 需求：设计一个方法（print） 用于打印 n 到 m 之间所有的奇数
- 思路：

  1：定义方法，名称为print 2：为方法添加两个int类型的形参，准备接受调用者传递过来的实参 3：方法中设计for循环，循环从n开始，到m结束 4：循环中加入if判断，是奇数，则打印 5：main方法中调用print方法，传入两个实际参数

- 代码：

```
package com.itheima.method2;

```

```java
public class Demo2Method {
    public static void main(String[] args) {
        // 5：main方法中调用print方法，传入两个实际参数
        print(20,10);
    }

    //1：定义方法，名称为print
    // 2：为方法添加两个int类型的形参，准备接受调用者传递过来的实参
    public static void print(int n, int m){
        System.out.println(n + "到" + m + "之间的奇数为:");
        // 3：方法中设计for循环，循环从n开始，到m结束
        for(int i = 20; i <= 10; i++){
            // 4：循环中加入if判断，是奇数，则打印
            if(i % 2 == 1){
                System.out.println(i);
            }
        }
    }

}
```

4. 带返回值方法的定义和调用

4.1 带返回值方法定义和调用（掌握）

- 定义格式

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

- 范例

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

- 调用格式

```
方法名 ( 参数 ) ;
数据类型 变量名 = 方法名 ( 参数 ) ;
```

- 范例

```
isEvenNumber ( 5 ) ;
boolean  flag =  isEvenNumber ( 5 );
```

- 注意：
  - 方法的返回值通常会使用变量接收，否则该返回值将无意义

4.2 带返回值方法的练习-求两个数的最大值(应用)

- 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，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 】
- 代码：

```
/*
        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;
        }
    }

}

```

5. 方法的注意事项

5.1 方法的通用格式（掌握）

- 格式：

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

- 解释：
  - 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
- 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
- 调用方法时的注意：
  - 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

5.2 方法的注意事项

- 方法不能嵌套定义
  - 示例代码：

```
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!!!
        }
    }
}
```

- 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

- 示例代码：

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

    }
    public static void methodTwo() {
        //return 100; 编译错误，因为没有具体返回值类型
        return;
        //System.out.println(100); return语句后面不能跟数据或代码
    }
}
```

6. 方法重载

6.1 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

- 正确范例：

```
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
    }
}
```

- 错误范例：

```java
public class MethodDemo {
    public static void fn(int a) {
        //方法体
    }
    public static int fn(int a) {   /*错误原因：重载与返回值无关*/
        //方法体
    }
}

public class MethodDemo01 {
    public static void fn(int a) {
        //方法体
    }
}
public class MethodDemo02 {
    public static int fn(double a) { /*错误原因：这是两个类的两个fn方法*/
        //方法体
    }
}
```

6.2 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

  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

  ③定义所有的重载方法，两个byte类型与两个short类型参数

  ④完成方法的调用，测试运行结果
- 代码：

```java
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;
    }

}
```

7. 方法的参数传递

7.1 方法参数传递基本类型（理解）

- 测试代码：

```
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

7.2 方法参数传递引用类型

- 测试代码：

```
package com.itheima.param;

public class Test2 {
    /*
         方法参数传递为引用数据类型 ：

                传入方法中的，是内存地址.
     */
    public static void main(String[] args) {
        int[] arr = {10, 20, 30};
        System.out.println("调用change方法前:" + arr[1]);
        change(arr);
        System.out.println("调用change方法后:" + arr[1]);
    }

    public static void change(int[] arr) {
        arr[1] = 200;
    }
}
```

- 结论：
  - 对于引用类型的参数，形式参数的改变，影响实际参数的值
- 结论依据：
  - 引用数据类型的传参，传入的是地址值，内存中会造成两个引用指向同一个内存的效果，所以即使方法弹栈，堆内存中的数据也已经是改变后的结果

7.3 数组遍历

- 需求：设计一个方法用于数组遍历，要求遍历的结果是在一行上的。例如：[11, 22, 33, 44, 55]
- 思路：
  - 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
  - 定义一个数组，用静态初始化完成数组元素初始化
  - 定义一个方法，用数组遍历通用格式对数组进行遍历
  - 用新的输出语句修改遍历操作
  - 调用遍历方法
- 代码：

```
package com.itheima.test;
```

```java

public class Test1 {
    /*
        需求：设计一个方法用于数组遍历，要求遍历的结果是在一行上的。例如：[11, 22, 33, 44, 55]
        思路：
            1.定义一个数组，用静态初始化完成数组元素初始化
            2.定义一个方法，对数组进行遍历
            3.遍历打印的时候，数据不换行
            4.调用遍历方法
     */
    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初始化
        int[] arr = {11, 22, 33, 44, 55};
        // 4.调用遍历方法
        printArray(arr);

        System.out.println("另外一段代码逻辑 ");
    }

    /*
        2.定义一个方法，对数组进行遍历

        1，参数           int[] arr
        2，返回值类型      void
     */
    public static void printArray(int[] arr){

        System.out.print("[");

        for (int i = 0; i < arr.length; i++) {

            if(i == arr.length -1){
                // 如果满足条件，说明是最后一个元素，最后一个元素，特殊处理
                System.out.println(arr[i] + "]");
            }else{
                // 3.遍历打印的时候，数据不换行
                System.out.print(arr[i] + ", ");
            }


        }
    }
}

```

7.4 数组最大值

- 需求：设计一个方法用于获取数组中元素的最大值
- 思路：
  - ①定义一个数组，用静态初始化完成数组元素初始化
  - ②定义一个方法，用来获取数组中的最大值，最值的认知和讲解我们在数组中已经讲解过了
  - ③调用获取最大值方法，用变量接收返回结果

- ④把结果输出在控制台

- 代码：

```
package com.itheima.test;

public class Test2 {
    /*
        需求：设计一个方法用于获取数组中元素的最大值

        思路：
            1.定义一个数组，用静态初始化完成数组元素初始化
            2.定义一个方法，用来获取数组中的最大值
            3.调用获取最大值方法，用变量接收返回结果
            4.把结果输出在控制台
     */
    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初始化
        int[] arr = {11, 55, 22, 44, 33};
        // 3.调用获取最大值方法，用变量接收返回结果
        int max = getMax(arr);
        //  4.把结果输出在控制台
        System.out.println(max);
    }

    /*
        2.定义一个方法，用来获取数组中的最大值

        1, 参数       int[] arr
        2, 返回值类型  int
     */
    public static int getMax(int[] arr){
        int max = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if(max < arr[i]){
                max = arr[i];
            }
        }
        return max;
    }
}

```

7.5 方法同时获取数组最大值和最小值

- 需求：设计一个方法，该方法能够同时获取数组的最大值，和最小值
- 注意: return语句, 只能带回一个结果.
- 代码：

```java
public class Test3 {
    /*
        需求：设计一个方法，该方法能够同时获取数组的最大值，和最小值

        注意：return语句，只能带回一个结果.
     */
    public static void main(String[] args) {

        int[] arr = {11,55,33,22,44};

        int[] maxAndMin = getMaxAndMin(arr);

        System.out.println(maxAndMin[0]);
        System.out.println(maxAndMin[1]);

    }

    public static int[] getMaxAndMin(int[] arr){
        int max = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if(max < arr[i]){
                max = arr[i];
            }
        }

        int min = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if(min > arr[i]){
                min = arr[i];
            }
        }

        int[] maxAndMin = {min, max};

        return maxAndMin;
    }
}

```