

一、实验名称

第2次上机--第3、4章内容

二、实验目的及要求

1. **第3章： page 18, 基础训练1** 目的: 统计3位数中不能被3或5或7整除的数的个数。 要求:

- (1) 利用条件或进行表达。
- (2) 利用逻辑 (&&) 运算符表达

2. **第4章： page 29, 基础训练1:**

目的: 一维数组的定义与使用。掌握数组的定义与赋值访问。 要求:

- (1) 定义一个含20个元素的整型数组并将数组内容输出。
- (2) 增加代码, 利用随机函数产生三位数给数组赋值, 观察输出结果。
- (3) 增加代码, 求所有元素的平均值, 并输出结果。
- (4) 增加代码, 求所有元素的最大值和最小值, 并输出结果。
- (5) 增加代码, 用如下增强for循环输出数组的所有元素。

3. **第4章： page 29, 基础训练2** 目的: 掌握二维数组的定义以及对数组的操作访问。 要求:

- (1) 定义4行5列的整型数组, 给数组赋值并输出。
- (2) 增加如下代码, 用如下增强for循环输出数组的所有元素。
- (3) 增加如下代码, 理解Arrays 类的deepToStringO方法的使用。

4. **第4章： page 30, 基础训练3** 目的: 掌握基本类型和引用类型的方法参数传递特点, 了解可变长参数的使用 要求:

- (1) 调试以下程序, 理解基本类型参数传递。
- (2) 调试以理解引用类型参数传递
- (3) 定义一个方法求两个整数的最大公约数。

5. **第四章, page34, 编程练习 (1)** 目的: 利用随机函数产生 100 个值为 50 以内的整数, 并用这些整数给一维数组赋值. 输出该数组, 每行5个数据输出。求最大元素值, 并指出它在数组中所有出现位置。

6. **第四章, page36, 编程题 (6)** 目的: 利用求素数的方法, 找出3 ~ 99的所有姐妹素数。所谓姐妹素数, 是指两个素数为相邻奇数。

三、实验环境

调试工具: Version: 1.98.0 Commit: 6609ac3d66f4eade5cf376d1cb76f13985724bcb OS: Windows_NT x64 10.0.26100

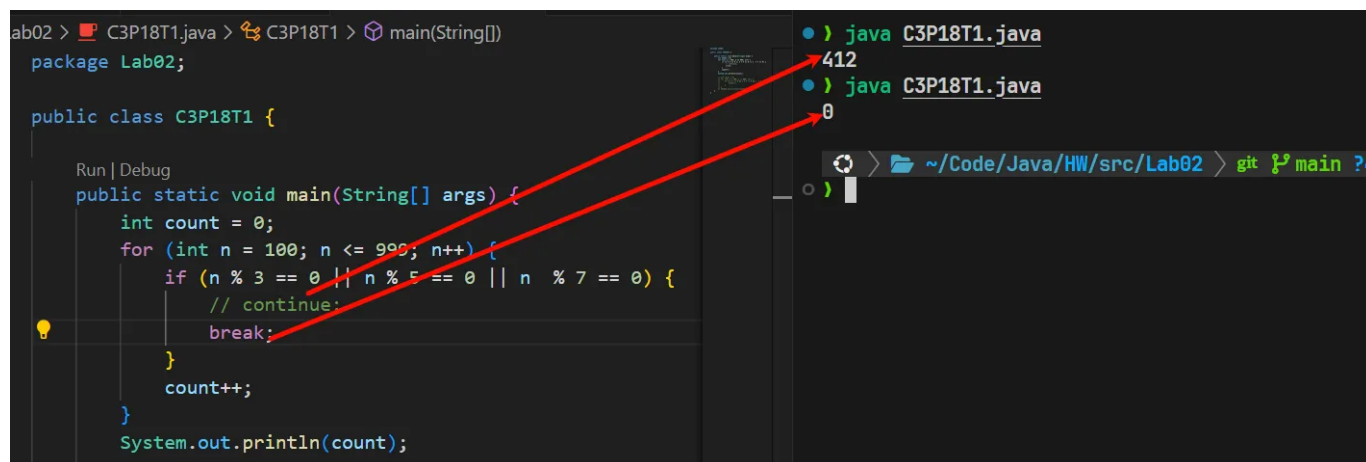
服务配置: openjdk 21.0.6 2025-01-21 OpenJDK Runtime Environment (build 21.0.6+7-Ubuntu-124.04.1) OpenJDK 64-Bit Server VM (build 21.0.6+7-Ubuntu-124.04.1, mixed mode, sharing)

四、实验设计

第3章: page 18, 基础训练1

(1) 利用条件或进行表达

思考: 使用continue时结果为412, 使用break时结果为0。



The screenshot shows an IDE with a Java file named C3P18T1.java. The code is as follows:

```
package Lab02;

public class C3P18T1 {

    Run | Debug
    public static void main(String[] args) {
        int count = 0;
        for (int n = 100; n <= 999; n++) {
            if (n % 3 == 0 || n % 5 == 0 || n % 7 == 0) {
                // continue;
                break;
            }
            count++;
        }
        System.out.println(count);
    }
}
```

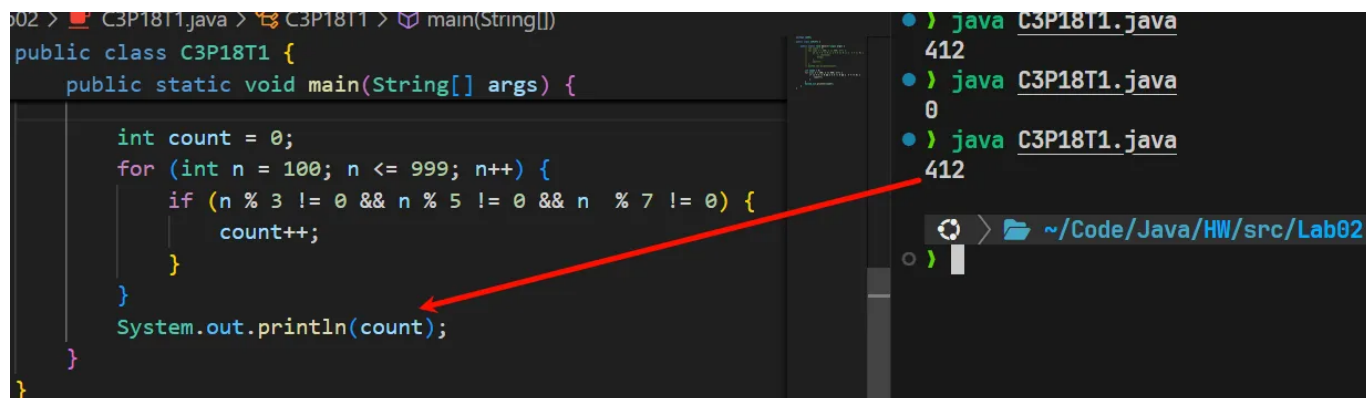
On the right, the console output shows the results of running the program:

```
java C3P18T1.java
412
java C3P18T1.java
0
```

Red arrows point from the `break;` statement in the code to the `0` output in the console.

(2) 利用逻辑&&运算符表达

执行结果为412



The screenshot shows an IDE with a Java file named C3P18T1.java. The code is as follows:

```
package Lab02;

public class C3P18T1 {

    Run | Debug
    public static void main(String[] args) {
        int count = 0;
        for (int n = 100; n <= 999; n++) {
            if (n % 3 != 0 && n % 5 != 0 && n % 7 != 0) {
                count++;
            }
        }
        System.out.println(count);
    }
}
```

On the right, the console output shows the results of running the program:

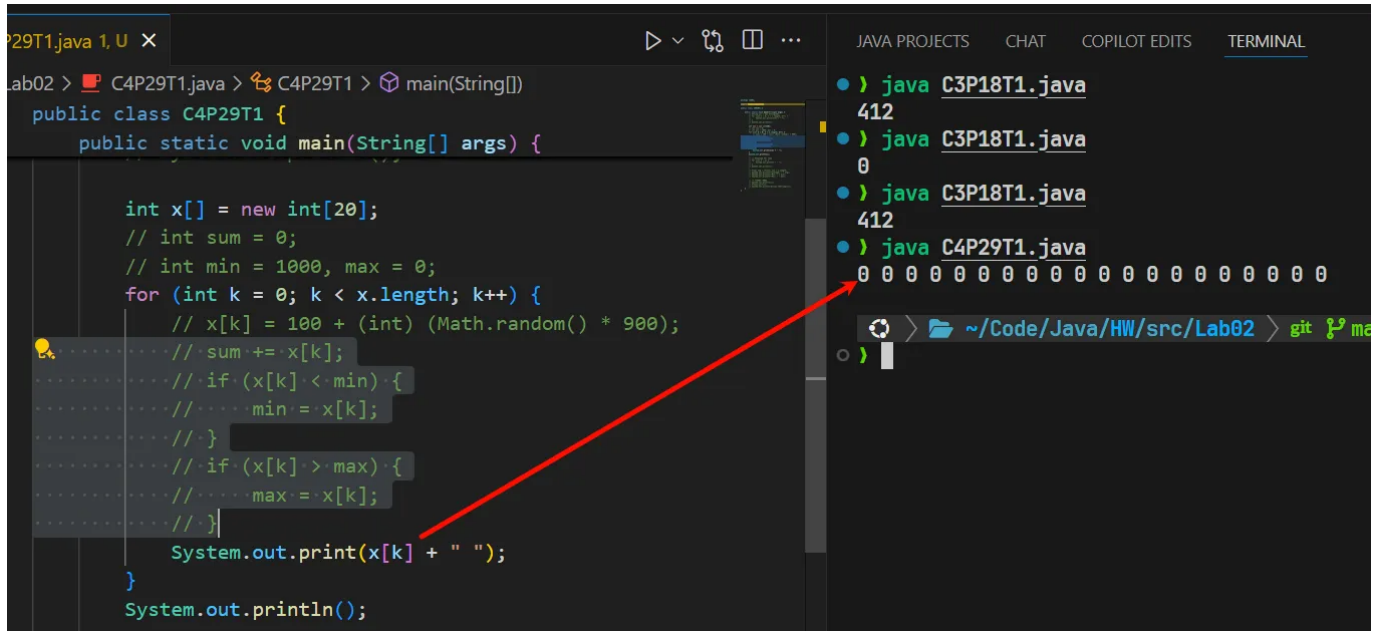
```
java C3P18T1.java
412
java C3P18T1.java
0
java C3P18T1.java
412
```

A red arrow points from the `count++;` statement in the code to the first `412` output in the console.

第4章：page 29, 基础训练1

(1) 定义一个含20个元素的整形数组并将数组内容输出

编译运行输出20个零，因为数组还没有被赋值，输出int类型数组元素的默认值0

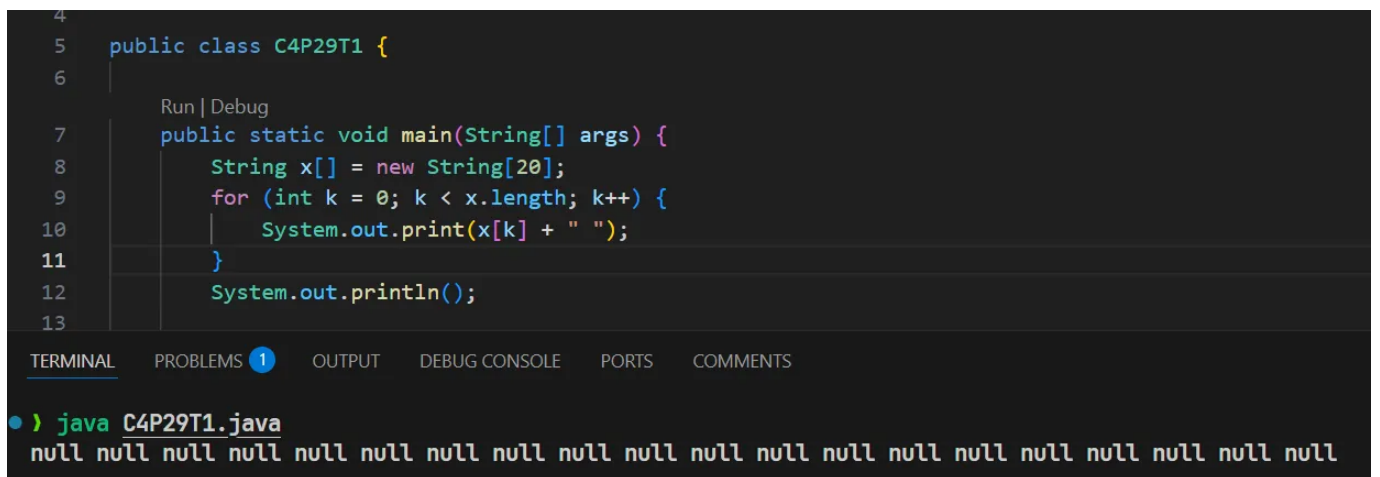


```
29T1.java 1, U X
lab02 > C4P29T1.java > C4P29T1 > main(String[])
public class C4P29T1 {
    public static void main(String[] args) {

        int x[] = new int[20];
        // int sum = 0;
        // int min = 1000, max = 0;
        for (int k = 0; k < x.length; k++) {
            // x[k] = 100 + (int) (Math.random() * 900);
            // sum += x[k];
            // if (x[k] < min) {
            //     min = x[k];
            // }
            // if (x[k] > max) {
            //     max = x[k];
            // }
            System.out.print(x[k] + " ");
        }
        System.out.println();
    }
}
```

```
• > java C3P18T1.java
412
• > java C3P18T1.java
0
• > java C3P18T1.java
412
• > java C4P29T1.java
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
~/Code/Java/HW/src/Lab02 > git
```

思考：将数组类型改为String，则输出的内容全为null



```
4
5 public class C4P29T1 {
6
7     Run | Debug
8     public static void main(String[] args) {
9         String x[] = new String[20];
10        for (int k = 0; k < x.length; k++) {
11            System.out.print(x[k] + " ");
12        }
13        System.out.println();
14    }
15 }
```

```
• > java C4P29T1.java
null null null null null null null null null null null null null null null null null null null null null
```

(2) 利用随机函数产生三位数给数组赋值

输出产生值的范围大致在100~1000之间。

(3) 求出所有元素的平均值，并输出结果

通过在循环外定义一个float类型的sum变量，在循环内 `sum += sum[k];` 得到加和，最后除以20，得到平均值。

(4) 求出最小值和最大值

在循环外定义一个min和max，在循环内逐元素与min，max进行比较，如果满足条件就对其进行更新。最后输出最大值和最小值

(5) 使用增强的for循环

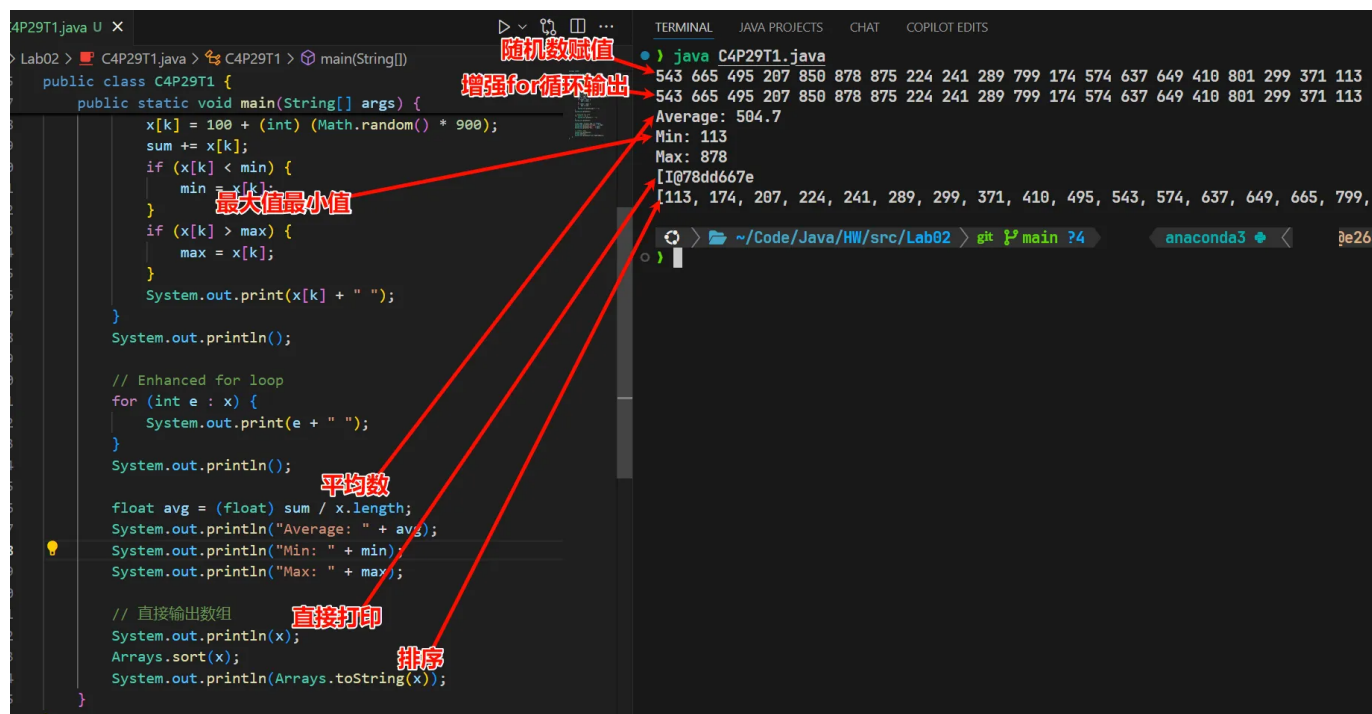
使用增强的for循环：`for (int e : x)` 起到了遍历数组中每个元素的作用，方便了编码。输出效果和之前的for循环相同。

(6) 使用sort方法，观察Arrays类方法的使用

通过添加下面的代码，实现了数组的排序，并且讲数组元素作为一个字符串输出

```
System.out.println(x);
Arrays.sort(x);
System.out.println(Arrays.toString(x));
```

总结Arrays类的作用：作为一个数组的工具类，可以使用Arrays里的各种方法来简化对数组的各种操作，例如排序和输出整个数组。



第4章：page 29, 基础训练2

(1) 定义四行五列的整形数组，给数组赋值并输出

二维数组的输出需要使用两层循环，第一层遍历第一维，第二层遍历第二维，也就是第一维按行遍历，第二位就是遍历行里面的列。

(2) 使用for循环输出二维数组中的所有元素

```

for (int[] row : x) {
    for (int col : row) {
        System.out.print(col + " ");
    }
    System.out.println();
}

```

通过使用增强的for循环，使得遍历输出二维数组的代码变得非常简洁。

(3) 理解Arrays类的deepToString()方法的使用。

```

System.out.println(Arrays.toString(x));
System.out.println(Arrays.deepToString(x));

```

通过增加以上两行代码，发现第一行代码输出的是一串地址值，也就是说普通的 `toString` 方法只能将第一维的元素转成 `String` 类型，而在二维数组中的第一维元素是这一行的首元素地址，所以只能输出四个地址。

而第二行正确地输出了这个二维数组中的所有元素，说明 `deepToString` 方法能够正确地讲所有元素变为字符串，然后完整输出整个数组

```

package Lab02;
import java.util.Arrays;

public class C4P29T2 {
    public static void main(String[] args) {
        int x[][] = new int[4][5];
        for (int row = 0; row < x.length; row++) {
            for (int col = 0; col < x[row].length; col++) {
                x[row][col] = (int) (Math.random() * 50);
                System.out.print(x[row][col] + " ");
            }
            System.out.println();
        }

        // 增强for循环
        System.out.println("----增强for循环----");
        for (int[] row : x) {
            for (int col : row) {
                System.out.print(col + " ");
            }
            System.out.println();
        }

        System.out.println("----Arrays----");
        System.out.println(Arrays.toString(x));
        System.out.println(Arrays.deepToString(x));
    }
}

```

```

java C4P29T2.java
19 16 2 17 1
46 31 15 17 7
9 0 21 40 25
7 41 27 19 6
----增强for循环----
19 16 2 17 1
46 31 15 17 7
9 0 21 40 25
7 41 27 19 6
----Arrays----
[[I@3b2da18f, [I@5906ebcb, [I@258e2e41, [I@3d299e3]
[[19, 16, 2, 17, 1], [46, 31, 15, 17, 7], [9, 0, 21, 40, 25], [7, 41, 27, 19, 6]]

```

第4章：page 30, 基础训练3

(1) 理解基本类型参数转递

通过使用了以下的swap函数，运行之后发现实际上m、n的值在调用swap方法之后没有发生变化，因为m、n传值给了swap方法内定义的x、y变量，swap交换的实际上是方法内的x、y变量。

```

static void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
    System.out.println(x + "," + y);
}

public static void main(String[] args) {
    int m = 24, n = 28;
    swap(m, n);
    System.out.println(m + "," + n);
}

```

```

Lab02 > C4P30T3.java > C4P30T3 > main(String[])
package Lab02;

public class C4P30T3 {
    // Test3
    static void swap(int x, int y) {
        int temp = x;
        x = y;
        y = temp;
        System.out.println(x + "," + y);
    }

    public static void main(String[] args) {
        int m = 24, n = 28;
        swap(m, n);
        System.out.println(m + "," + n);
    }
}

```

Run | Debug

```

> java C4P30T3.java
28,24
24,28

```

(2) 理解引用类型参数传递

通过使用引用传递，我们可以发现参数实际上正常交换了

```

static void swap2(int x[]) {
    int temp = x[0];
    x[0] = x[1];
    x[1] = temp;
}

public static void main(String[] args) {
    int m[] = {23, 40};
    System.out.println(m[0] + "," + m[1]);
    swap2(m);
    System.out.println(m[0] + "," + m[1]);
}

```

```
> Lab02 > C4P30T3.java > C4P30T3 > swap2(int[])
public class C4P30T3 {
    // Test4
    static void swap2(int x[]) {
        int temp = x[0];
        x[0] = x[1];
        x[1] = temp;
    }

    Run | Debug
    public static void main(String[] args) {
        int m[] = {23, 40};
        System.out.println(m[0] + "," + m[1]);
        swap2(m);
        System.out.println(m[0] + "," + m[1]);
    }
}
```

```
● java C4P30T3.java
28,24
24,28
● java C4P30T3.java
23,40
40,23
```

```
> ~/Code/Java/HW/src/Lab02
```

总结

- 基本类型（如int、float等）在方法调用时是通过值传递的，方法内部对参数的修改不会影响到原始变量。
- 引用类型（如数组、对象等）在方法调用时是通过引用传递的，方法内部对参数的修改会直接影响到原始变量。

(3) 求两个整数的最大公约数

```
private static int comm(int x, int y) {
    for (int k = Math.min(x,y); k > 1; k--) {
        if (x % k == 0 && y % k == 0) {
            return k;
        }
    }
    return 1;
}

public static void main(String[] args) {
    System.out.println(comm(24,78));
    System.out.println(comm(6,9));
}
```

以上程序输出结果为

```
6
3
```

```
// Test5
private static int comm(int x, int y) {
    for (int k = Math.min(x, y); k > 1; k--) {
        if (x % k == 0 && y % k == 0) {
            return k;
        }
    }
    return 1;
}

Run | Debug
public static void main(String[] args) {
    System.out.println(comm(x:24,y:78));
    System.out.println(comm(x:6,y:9));
}
```

```
24,20
• java C4P30T3.java
23,40
40,23
• java C4P30T3.java
6
3
~/Code/Java/
```

思考:

1. 为什么循环设计为先尝试数据范围中较大的数?

因为最大公约数不会大于两个数中的最小值, 从较大的数开始尝试可以更快地找到最大公约数并提前结束循环, 提高运行效率

2. 分析 `return` 语句的作用, 执行特点

`return` 语句用于终止方法的执行并返回一个值, 在 `comm` 方法中, 当找到一个公约数时, 立即返回该值并结束方法的执行, 如果循环结束后没有找到公约数, 则返回1, 表示两个数互质.

3. 总结方法的结果的类型定义与返回处理的特点

- 方法的返回类型决定了方法返回值的类型, 从 `comm` 中可以看出: `comm` 被定义为返回结果类型为 `int` 最后返回值也是 `int` 类型的 `k` 或者是 `1`.
- 返回处理: 当完成了这个方法的目的, 可以立即返回, 结束方法的执行; 但是如果执行完了方法内所有逻辑语句, 也需要有一个返回处理, 来结束这个方法.

4. 定义一个方法求任意多个实数的平均值

任意多个元素 就涉及到 **可变长参数** 通过参考书上的样例可以写出以下的程序:

```

public static double average (double ... numbers) {
    double sum = 0;
    for (double num : numbers) {
        sum += num;
    }
    return numbers.length > 0 ? sum / numbers.length : 0;
}

public static void main(String[] args) {
    System.out.println(average(1, 2, 3, 4, 5));
    System.out.println(average(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
    System.out.println(average());
}

```

因为数组长度可能为零, 所以在 `return` 时进行了特判, 经过main函数的测试, 基本能够实现正确的求任意多个实数的平均值

```

// 思考4
public static double average (double ... numbers) {
    double sum = 0;
    for (double num : numbers) {
        sum += num;
    }
    return numbers.length > 0 ? sum / numbers.length : 0;
}

Run | Debug
public static void main(String[] args) {
    System.out.println(average(...numbers:1, 2, 3, 4, 5));
    System.out.println(average(...numbers:1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
    System.out.println(average());
}

```

参数长度为0

28,24
24,28
23,40
40,23
6
3
3.0
5.5
0.0

~ /Code/Java/HW/src/ b7c < 03:32:16
~ /Code/Java/HW/src/ 7c < 03:32:16

第4章: page34, 编程练习 (1)

首先使用了 `int[] x = new int[100];` 开辟了一个长度为100的 `int` 类型数组;

然后在一个一重循环中对这个数组进行遍历, 使用 `x[i] = random.nextInt(50);` 为数组中每个元素赋值, 其取值范围为 $x \in [0, 50)$

```

int[] x = new int[100];
Random random = new Random();

// 赋值, 100个值, 五十以内[0, 50)
for (int i = 0; i < x.length; i++) {
    x[i] = random.nextInt(50);
}

```

在下一个循环中, 逐个输出每个数据元素. 要求每五个一行, 因为 0 ~ 4 五个为一组, 所以采取 `if (i % 5 == 4)` 的控制语句来控制输出一个换行符.

```
// 输出, 五个一行
for (int i = 0; i < x.length; i++) {
    System.out.print(x[i] + " ");
    if (i % 5 == 4) {
        System.out.println();
    }
}
```

最后一个循环用于找出最大值及其数组下标, 首先假定数组第一个元素的值最大, `int max = x[0];` 并且记录其下标 `int idx = 0;`, 然后再这个遍历整个数组的循环中逐个将记录下来的值于当前的值做对比, 如果 `x[i] > max` 那么就对 `max` 和 `idx` 进行一个更新, 直至循环结束.

```
// 求最大值, 并指出位置
int max = x[0];
int idx = 0;
for (int i = 0; i < x.length; i++) {
    if (x[i] > max) {
        max = x[i];
        idx = i;
    }
}

System.out.println("最大值: " + max + ", 位置: " + idx);
```

```
> Lab02 > C4P34E1.java > C4P34E1 > main(String[])
1 package Lab02;
2
3 import java.util.Random;
4
5 public class C4P34E1 {
6     Run | Debug
7     public static void main(String[] args) {
8         int[] x = new int[100];
9         Random random = new Random();
10
11         // 赋值, 100个值, 五十以内[0, 50)
12         for (int i = 0; i < x.length; i++) {
13             x[i] = random.nextInt(50);
14         }
15
16         // 输出, 五个一行
17         for (int i = 0; i < x.length; i++) {
18             System.out.print(x[i] + " ");
19             if (i % 5 == 4) {
20                 System.out.println();
21             }
22         }
23
24         // 求最大值, 并指出位置
25         int max = x[0];
26         int idx = 0;
27         for (int i = 0; i < x.length; i++) {
28             if (x[i] > max) {
29                 max = x[i];
30                 idx = i;
31             }
32         }
33
34         System.out.println("最大值: " + max + ", 位置: " + idx);
35     }
36 }
```

```
• java C4P34E1.java
15 33 31 2 35
29 31 25 9 36
30 33 3 36 47
35 35 5 41 21
18 34 33 26 22
15 15 2 25 31
27 46 37 27 10
48 15 49 17 3
27 5 29 26 27
25 2 13 17 25
47 32 40 15 0
7 30 2 33 39
42 38 26 35 10
2 15 43 31 21
26 18 0 16 40
4 42 29 6 16
28 12 38 23 44
28 37 37 9 45
14 39 31 23 48
6 48 49 48 17
最大值: 49, 位置: 37
```

第4章: page36, 编程题 (6)

首先定义出找素数的方法

```
for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
        return false;
    }
}
```

其中增加了一点优化: $i * i \leq n$ 而不是朴素的 $i < n$, 原因如下:

如果一个数 n 不是素数, 那么它一定可以分解为两个因数的乘积, 即 $n = a * b$ 。其中, a 和 b 至少有一个小于或等于 $\sqrt{n}$ 。如果两个因数都大于 $\sqrt{n}$, 那么它们的乘积将大于 n , 这与 $n = a * b$ 矛盾。因此, 只需要检查到 $\sqrt{n}$ 就可以确定 n 是否为素数

然后遍历 3 ~ 99 的所有数, 调用刚才定义的 `isPrime` 方法来检查这个数和相邻的奇数是否是素数, 如果是就一起输出, 说明她们是题目要求的姐妹素数.

完整代码如下

```
static boolean isPrime (int n) {
    if (n < 2) {
        return false;
    }

    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return false;
        }
    }

    return true;
}

public static void main(String[] args) {
    for (int i = 3; i < 100 - 2; i++) {
        if (isPrime(i) && isPrime(i + 2)) {
            System.out.println(i + " " + (i + 2));
        }
    }
}
```

```
> Lab02 > C4P36E6.java > C4P36E6
1 package Lab02;
2
3 public class C4P36E6 {
4
5     static boolean isPrime (int n) {
6         if (n < 2) {
7             return false;
8         }
9
10        for (int i = 2; i * i <= n; i++) {
11            if (n % i == 0) {
12                return false;
13            }
14        }
15
16        return true;
17    }
18
19    Run | Debug
20    public static void main(String[] args) {
21        for (int i = 3; i < 100 - 2; i++) {
22            if (isPrime(i) && isPrime(i + 2)) {
23                System.out.println(i + " " + (i + 2));
24            }
25        }
26    }
27 }
```

```
● java C4P36E6.java
3 5
5 7
11 13
17 19
29 31
41 43
59 61
71 73
~/C/Java/HW/src/La
```

五、实验体会

我从本次实验解决的问题中学到了：

- **数组的初始值机制** 通过实验观察，我深入理解了Java不同数据类型数组的初始化特点：整型数组默认初始化为0，而引用类型如String数组默认为null。这有助于在实际编程中正确预期和处理未显式初始化的数组元素。
- **参数传递** 实验中对基本类型和引用类型的参数传递机制进行了对比，清晰地认识到Java中所有参数本质上都是值传递。基本类型传递的是值的副本，而引用类型传递的是引用的副本，这解释了为何对引用类型的操作能改变原始数据。
- **算法优化思维** 在实现最大公约数算法时，学习到从较小数开始递减查找并及时返回的优化思路，理解了算法设计中效率考量的重要性。同样在素数判断中，仅检查到平方根而非全部可能因子的优化方法也让我认识到数学原理对编程效率的影响。
- **工具类的应用** 通过使用Arrays类的各种方法如toString()、deepToString()、sort()等，理解了Java提供的工具类可以大大简化常见操作，提高代码的可读性和维护性，特别是在处理复杂数据结构如多维数组时。

总而言之，本次实验让我深入理解了Java中的数组操作、方法参数传递机制以及常用算法的实现技巧。特别是对基本类型与引用类型的区别、方法返回处理、可变长参数等概念有了实践性的把握。。