方法

回顾

今天任务

```
1.方法
2.方法的重载
3.递归
```

教学目标

```
1.什么是方法
2.掌握方法的声明格式
3.掌握方法的使用
4.掌握方法的重载
5.掌握递归
```

第一节：方法

1.1 什么是方法

```
Java的方法类似于其它语言的函数，是一段用来完成特定功能的代码片段
```

1.2 为什么要声明方法

```
DRY原则,把能被复用的逻辑抽取出来
实现相对独立的逻辑
实现比较复杂的逻辑
可以对具体实现进行隐藏/封装
```

1.3 方法的作用

```
简化代码，提高代码的可读性，提高代码的可维护性【版本迭代】
```

1.4 方法的声明格式

```
语法：
    访问权限修饰符   其他修饰符   返回值类型   函数名称(参数列表) {
        //函数体【方法体】
        return 返回值;
    }
```

1.4.2 方法声明中需要注意

```
a.一个完整的函数包含声明部分和实现部分
b.访问权限修饰符:public,default【如果没有添加任何的访问权限修饰符，则默认为default，而default不需要显式的写出来】，目前使用的访问权限修饰符都和main函数保持一致，使用public
c.其他修饰符：static 【静态的】，要么不写【非静态函数】，要么写上【静态函数】
d.返回值类型：函数运行之后所得到的结果的数据类型，如果没有运行结果，则直接为void【空】
e.函数名称：标识符【遵循小驼峰】，尽量做到顾名思义
f.参数列表：如果函数所实现的功能中有未知项参与运算，就可以将未知项设置为参数
        实际参数：实参，在函数外面定义，表示实际参与运算的值或者变量，作用为了给形参进行赋值
        形式参数：形参，在函数中定义，用于接收实参的值，相当于是一个未被赋值的变量
        形参数据类型  形参变量名称
        形参 = 实参;
g.函数体：抽取出来的具有特殊功能的代码段
h.return 返回值：将当前函数运行之后的结果进行返回，返回给当前函数的调用者
        return：结束整个方法
```

1.4.3 练习

```
1. 最简单的方法
void sum(){
  System.out.println("加法操作");
}
2. 拥有修饰符的方法
public static void sum(){
  System.out.println("加法操作");
}
3. 拥有参数的方法
public static void sum(int a,int b){
   System.out.pritln("两数相加结果"+a+b);
}
4.拥有返回值的方法
public static int sum(int a,int b){
  return a+b;
}
```

1.5 方法的调用格式

语法：函数名称(实参列表)

注意：

a.实参的数量和类型必须和形参保持完全的一致，实现书写的顺序也必须和形参中的顺序保持完全一致

b.函数之间只能进行相互的调用，而不能在函数中声明函数，就目前而言声明的函数都和main函数时并列的

c.定义函数的时候，运算的结果会返回给调用者【在哪个函数中调用，运算的结果返回给哪个函数】

1.5.2 方法调用练习

```java
class  TextDemo01
{
    public static void main(String[] args)
    {
        //需求：打印多遍九九乘法表
        /*
        for(int i = 1;i <= 9;i++) {
            for(int j= 1;j <= i;j++) {
                System.out.print(j + "x" + i + "=" + i * j + " ");
            }
            System.out.println();
        }
        for(int i = 1;i <= 9;i++) {
            for(int j= 1;j <= i;j++) {
                System.out.print(j + "x" + i + "=" + i * j + " ");
            }
            System.out.println();
        }
        */

        System.out.println("start");

        print();
        print();
        print();
        print();
        System.out.println("end");
    }

    //对于打印九九乘法表的功能提取出来一个函数
    /*
    访问权限修饰符  其他修饰符  返回值类型  函数名称(参数列表) {

        //函数体【方法体】

        return 返回值;
    }
    */
    public static void print() {
        for(int i = 1;i <= 9;i++) {
            for(int j= 1;j <= i;j++) {
                System.out.print(j + "x" + i + "=" + i * j + " ");
            }

            System.out.println();
        }
    }
}
```

1.6 方法中的参数

> 工作原理：调用方法的时候，用实参给形参进行赋值，这个过程被称为传参
>
> 形参就是一个变量，实参就是一个常量或者携带着值的变量，传参就是把实参赋值给形参 传参时需要注意的事项：实参的数量和类型必须和形参的数量和类型保持一致【相兼容的数据类型】

1.6.2 方法参数练习

```java
//演示参数的使用
class FunctionUsageDemo03
{
    public static void main(String[] args)
    {
        //需求：交换两个变量的值

        //实参
        int a = 10;
        int b = 20;

        //调用函数
        swap(a,b);

        System.out.println("main函数中的a=" + a);//10
        System.out.println("main函数中的b=" + b);//20
    }

    //分析：需要参数（两个参数）
    //       不需要返回值
    //形参：没有携带值的变量，多个变量之间使用逗号分隔
    public static void swap(int a,int b) {
        //定义一个中间的临时变量
        int temp = 0;
        temp = a;
        a = b;
        b = temp;

        System.out.println("swap函数中的a=" + a);//20
        System.out.println("swap函数中的b=" + b);//10
    }
}
```

1.7 方法的返回值

return关键字的使用

a.表示一个函数执行完成之后所得到的结果 void：表示没有返回值

b.return的使用 1>在没有返回值的函数中使用return return单独成立一条语句，类似于break或者continue，后面不能跟任何的数值 作用：结束整个方法

2>在一个有返回值的函数中使用return 这种情况下函数中必须出现return return后面必须跟一个具体的数值，而且数值的类型和返回值类型必须保持一致 作用：结束整个方法，并且将返回值携带给调用者

> 3>如果一个自定义的函数有返回值，并且在方法中遇到了分支结构，使用return 在每一个分支后面都需要出现一个return

1.7.2 方法的返回值练习

```
class ReturnUsageDemo01
{
    public static void main(String[] args)
    {
        show();
    }
    /*
    1>在没有返回值的函数中使用return
            return单独成立一条语句，类似于break或者continue，后面不能跟任何的数值
            作用：结束整个方法
    */
    public static void show() {
        System.out.println("Hello World!");

        int x = 10;
        if(x > 5) {
            return;//在某些情况下，可以使用return替换break
        }

        // 错误: 无法访问的语句
        System.out.println("Hello World!========");
    }
}
```

```java
class  ReturnUsageDemo02
{
    public static void main(String[] args)
    {
        int result =  add(10,20);
        System.out.println(result);

        System.out.println(add(11,22));

        //如果一个函数返回值类型为void，则不能再调用函数的同时直接打印
        // 错误: 此处不允许使用 '空' 类型
        //System.out.println(show(11,22));
        show(11,22);
    }

    /*
    2>在一个有返回值的函数中使用return
            这种情况下函数中必须出现return
            return后面必须跟一个具体的数值，而且数值的类型和返回值类型必须保持一致
            作用：结束整个方法，并且将返回值携带给调用者
    */

    //需求：求两个变量的和
    public static int add(int a,int b) {
        int sum = a + b;
        //谁调用，返回给谁
        //return每次只能携带一个数据返回
        return sum;
    }

    public static void show(int a,int b) {

        int sum = a + b;
        System.out.println(sum);
    }
}
```

```
class  ReturnUsageDemo03
{
    public static void main(String[] args)
    {
        int result = compare(34,67);
        System.out.println(result);
    }
    /*
    3>如果一个自定义的函数有返回值，并且在方法中遇到了分支结构，使用return
            在每一个分支后面都需要出现一个return
    */
    //需求：比较两个变量的大小，返回较大的一个
    public static int compare(int num1,int num2) {
        //多分支
        /*
        if(num1 > num2) {
            return num1;
        } else if(num1 < num2) {
            return num2;
        } else {
            return num1;
        }
        */

        //出现的问题：在使用单分支的时候，分支内部有return，这时编译还是不通过
        //解决办法：在单分支的外面再添加一个返回值，返回值随意，只要类型和返回值类型匹配就ok
        //单分支
        if(num1 > num2) {
            return num1;
        }

        //没有实际意义，作用只是为了匹配语法
        return 0;
    }
}
```


第二节：方法的重载

2.1 方法重载的概念

同一个类中，方法名字相同，参数列表不同。则是重载
注意：

1. 参数列表的不同包括，参数个数不同，参数数据类型不同，参数顺序不同
2. 方法的重载与方法的修饰符和返回值没有任何关系

2.2 方法重载练习

```java
//演示方法的重载
//测试类
class TextDemo04
{
    public static void main(String[] args)
    {
        //对于重载函数而言，具体调用的是哪个函数，取决于所传的参数
        Check.show("10");
        Check.show("10",10);
    }
}
//实体类
/*
在同一个类中，如果满足以下的条件，则称为这几个方法之间彼此重载
        a.方法名相同
        b.参数不同【数量不同或者类型不同】
        c.访问权限修饰符和返回值类型没有影响
*/
class Check
{
    public static void show() {
        System.out.println("无参无返回值的show");
    }

    //1.改变参数
    public static void show(int a) {
        System.out.println("int的show");
    }

    public static void show(String a) {
        System.out.println("String的show");
    }

    public static void show(String a,int b) {
        System.out.println("String  int的show");
    }

    //2.改变返回值:返回值对方法的重载没有任何影响
    //只改变返回值类型，其他都不改变，则对于编译器而言，则认为是同一个方法
    /*
    public static String show() {
        System.out.println("String返回值的show");

        return "abc";
    }
    */

    //3.访问权限修饰符
    //只改变访问权限修饰符，其他都不改变，则对于编译器而言，则认为是同一个方法
    /*

    static void show() {
```

```
            System.out.println("show");
        }
        */
    }
```

第三节： 递归算法

3.1 递归算法的概念

> 在一个方法的方法体内调用该函数本身，称为函数的递归
>
> 方法递归包含了一种隐式的循环，会重复执行某段代码，但是这种重复不需要使用循环语句来进行控制

3.2 案例： 求斐波那契数列中的某个数

```
class  DiGuiUsageDemo01
{
    public static void main(String[] args)
    {
        /*
        斐波那契数列
        1,2,3,4,5,6, 7, 8, 9,10,11,.....
        1,1,2,3,5,8,13,21,34,55,89....

        分析：
        1.第一个位置和第二个位置上的数是固定的，都是1
        2.第n个位置上的数 = 第n - 1个位置上的数 + 第n - 2个位置上的数

        fun(1)  = 1
        fun(2) = 1
        fun(3) = fun(2) + fun(1) = 1  + 1
        fun(4) = fun(3) + fun(2) = fun(2) + fun(1) +fun(2)
        fun(5) = fun(4) + fun(3) = fun(3) + fun(2) + fun(2) + fun(1) = fun(2) + fun(1) + fun(2)
+ fun(2) + fun(1)
        ....
        fun(n) = fun(n - 1) + fun(n -2)
        */

        int result1  = fun(10);
        System.out.println(result1);
    }
    //需求：报个数，获取在斐波那契数列中对应的数
    public static int fun(int n) {
        if(n == 1 || n == 2) {
            return 1;
        } else {
            int num1 = fun(n - 1);
            int num2 = fun(n -2);
            int sum = num1  + num2;

            System.out.println("num1=" + num1 + ",num2=" + num2);
            return sum;
        }
    }
}
```

3.3 案例二：求1~某个数之间所有整数的和

```
class DiGuiUsageDemo02
{
    public static void main(String[] args)
    {
        int result = total(100);
        System.out.println(result);
    }

    //需求：求1~某个数之间所有整数的和
    //普通方式
    public static int add(int n) {
        int sum = 0;
        for(int i = 1;i <= n;i++) {
            sum += i;
        }
        return sum;
    }

    //使用递归实现
    /*
    total(1) = 1
    total(2) = total(1) + 2
    total(3) = total(2) + 3 = total(1) + 2 + 3
    ....
    total(n) = total(n - 1) + n
    */
    public static int total(int n) {
        if(n == 1) {
            return 1;
        } else {
            return total(n - 1) + n;
        }
    }
}
```

第四节：课堂练习

4.1 练习

```java
class PracticeDemo01
{
    public static void main(String[] args)
    {
        method1();
        method2(10);
    }
  public static void method1() {
    //输出100~200之间能被3整除的数
    for(int i = 100;i <= 200;i++) {
        if(i % 3 != 0) {
            continue;
        }

        System.out.println(i);
    }
  }
  public static void method2(int num) {
    //判断一个数是否为质数
    //质数：除了1和本身能整除，如果出现一个数可以将这个数整除的话，那么这个数就不是质数
    //1.假设是质数
    boolean isPrime = true;
    //2.寻找能够整除num的数，只要出现一个，则原来的假设被推翻
    for(int i = 2;i < num;i++) {
        //3.大对小求余
        if(num % i == 0) {
            //4.修改原来假设的状态
            isPrime = false;
            break;
        }
    }
    return isPrime;
  }
}
```

```java
class PracticeDemo02
{
    public static void main(String[] args)
    {
        getNum(100);
        getNum1(10);
        exchange();
    }

    //1.求1--某个数之间可以被7整除的数的个数
    public static int getNum(int n) {
        int count = 0;
        for(int i = 1;i <= n;i++) {
            if(i % 7 == 0) {
                count++;
            }
        }
        return count;
    }

    //2.计算1到某个数以内能被7或者3整除但不能同时被这两者整除的数的个数。
    public static int getNum1(int n) {
        //&&
        int count = 0;
        for(int i = 1;i <= n;i++) {
            if((i % 7 == 0 || i % 3 == 0) && i % 21 != 0) {
                count++;
            }
        }

        //嵌套if语句
        int count1 = 0;
        for(int i = 1;i <= n;i++) {
            if(i % 7 == 0 || i % 3 == 0) {
                if(i % 21 != 0) {
                    count1++;
                }
            }
        }
        return count;
    }

    //3.从键盘输入两个数，赋值给两个变量，交换这两个变量的值【三种方法】
    public static void exchange() {
        int a = 10;
        int b = 20;

        //方式一：异或【面试题：不采用第三方变量，交换两个变量的值】
        a = a ^ b;   //10 ^ 20
        b = a ^ b;   //10 ^ 20 ^ 20 = 10
        a = a ^ b;   //10 ^ 20 ^  1 0  =  2 0
```

```
        //方式二：加法
        int c = a  + b;//30
        a = c - a;//30 - 10 =20
        b = c - a; //30 - 20 = 10

        a = a  + b;
        b = a - b;//10
        a = a - b;//20

        //方式三：
        int temp = a;
        a = b;
        b = temp;
    }
}
```

第五节：总结

第六节：课前默写

> 根据要求完成下面的题目（注:下面的三步为同一道题）
>
> 1.分别使用静态初始化和动态初始化的方式定义一个数组并初始化
>
> 2.对静态初始化的数组进行排序，其中，冒泡降序，选择升序
>
> 3.使用简单for循环和增强for循环对排好序的数组进行遍历

第七节：作业

> 编程题
>
> 1.计算从1到某个数以内所有奇数的和。
>
> 2.计算从1到某个数以内所有能被3或者17整除的数的和。
>
> 3.计算1到某个数以内能被7或者3整除但不能同时被这两者整除的数的个数。

4.计算1到某个数以内能被7整除但不是偶数的数的个数。

5.从键盘输入一个数n，判断是不是一个质数（质数是只能被1和它自身整除的数）。

中级： 编程题

1.求2∽某个数之内的素数。【素数 ： 只能被1或本身整除的数】

2.判断某个年份是否是闰年。 A：能被4整除，并且不能被100整除 (2020) B：或者能被400整除。

3.已知有一个数列：f(0) = 1,f(1) = 4,f(n+2) = 2 * f(n+1) + f(n),其中n是大于0的整数，求f(n)的值（提示：使用递归）

4.求2+22+222+2222。。。【递归】

高级 编程题

1.求某个三位数以内的水仙花数： 水仙花数：一个数各个位上的立方之和，等于本身 例如： 153 = 1（3） + 5（3）+3（3） = 1+125+27 = 153

第八节：面试题

1.方法的传参过程是如何工作的

2.return关键字的用法有哪些，举例说明

3.什么是函数的重载？举例说明