方法

回顾

```
循环语句:
    循环：重复执行一段代码，直到条件为false为止。
    循环的组成:
    (1) 循环变量初始化部分，int i=0
    (2) 循环条件，如果为true，执行循环体，如果false不执行
    (3) 循环体，重复执行的特定代码
    (4) 更新循环变量部分
java中三个循环语句：
    1.while(boolean条件){
      ...
    }
    先判断，再执行
    2.do{
      ...
    }while(boolean条件);
    先执行，再判断
    3.for(初始化变量;条件;更新循环变量){
      ...
    }
    先判断，再执行

    三种循环比较：
        1 语法不同
        2 特点：while for 先判断 再执行  do while 先执行 再判断
        3 循环次数固定优先 使用for ，不固定使用while do{} while();

二重循环:
    循环中嵌套循环，
    特点：外层执行一次，内层执行一遍。
    (1) 打印矩形
    (2)打印直角三角形
    (3)打印等腰三角形
    (4)九九乘法表
跳转语句 :
    break:
        使用场合: switch 和循环中
        作用:跳出(终止) switch和循环语句

    continue:
        使用场合:用在循环中
        作用:跳过本次循环，继续下一次循环
```

今天任务

```
1.方法
2.方法重载
3.递归算法
```

教学目标

1.理解什么是方法
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.了解递归算法

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

1 把复用的逻辑抽取出来，封装成方法，提高代码的重用性
2 实现相对独立的逻辑，提高代码的维护性
3 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，可维护性，可重用性。

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  方法名称(参数列表) {
        //方法体【函数体】
        return 返回值; //如果返回值类型void ,可以不用写return
    }
```

1.4.1方法的分类

根据方法有没有参数,可分为:

1.无参方法

2.有参方法

根据有没有返回值，可分为：

1.无返回值方法

2.有返回值方法

上机练习:

```
1. 最简单的无参方法
void sum1(){
  System.out.println("加法操作");
}
2. 拥有修饰符的无参方法
public static void sum2(){
  System.out.println("加法操作");
}
3. 拥有参数的方法
public static void sum3(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 sum4(int a,int b){
```

```
  return a+b;
}
5.声明一个无参数带返回值
public static int sum5(){
        int x=20;
        int y=28;
        int z=x+y;
        return z;
}
```

1.5 方法的调用格式

语法：方法名称(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。

b.方法之间只能进行相互的调用，而不能在方法中声明方法，就目前而言声明的方法都和main方法时并列的

c.如果定义方法有返回值，运算的结果会返回给调用者，调用者需要定义变量接收数据

1.5.1 方法调用练习

```
class  TextDemo01
{
    public static void main(String[] args)
    {
        //需求：打印2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();

        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】

上机练习：

```
//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**关键字的作用**：结束方法,返回结果，

return关键字的使用

表示一个方法执行完成之后所得到的结果.

(1) 如果方法的返回类型是void：表示没有返回值，可以不用写return，如果要是写 return; 建议不写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) 在一个有返回值的方法中使用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，并且返回结果给调用者

(3) 如果一个自定义的方法有返回值，并且在方法中遇到了分支结构，在每一个分支后面都需要出现一个return。

1.7.1 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;//结束方法
        }
        // 不能执行
        System.out.println("Hello World!========");
    }
}
```

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

    /*
    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;
    }
}
```

```
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) {
        if(a>b){
            return a;
        }else if(a<b){
            return b;
        }else{
            return 0;
        }
    }
}
```

第二节：方法重载

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，则是方法重载。

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

```
//演示方法的重载
//测试类
class TextDemo04
{
    public static void main(String[] args)
    {
        show("10");
        show("10",10);
    }
    public static void show() {
        System.out.println("无参无返回值的show");
    }
    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");
    }
}
```

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方法本身，称为方法的递归。简单理解自己调用自己。
>
> 演示案例：
>
> 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

出现问题： StackOverFlowError 栈空间溢出异常，所以递归不能一直运行，一定要有结束条件。

3.2 案例一：求10的阶乘

```
//使用递归实现10 的阶乘
//求 10阶乘 , 求9的阶乘 * 10
//求 9阶乘， 求8的阶乘 *9
//...
//...
//求2的阶乘 , 1的阶乘*2
//1  1

//使用递归求10的阶乘
public class Demo4{
    public static void main(String[] args){
        int result=jiecheng(10);
        System.out.println(result);
    }
    public static int jiecheng(int num){
        if(num==1){
            return 1;
        }else{
            return jiecheng(num-1)*num;
        }
    }
}
```

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;
        }
    }
}
```

3.4 案例三： 求斐波那契数列中的第30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 {
            return fun(n - 1)+fun(n-2);
        }
```

```
    }
}
```

递归算法的使用：

1 正常思维无法解决问题时才采用递归算法 (汉诺塔问题)

2 使用递归能够大大提高执行效率。高效的排序算法：快速排序

3.5 上机练习

练习1 输出100~200之间能被3整除的数

练习2 判断一个数是否为素数

练习3 求1--某个数之间可以被7整除的数的个数

```
class PracticeDemo01
{
    public static void main(String[] args)
    {
        method1();
        method2(10);
    }
  public static void method1() {
    //练习1 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 boolean method2(int num) {
    //练习2 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;
  }
  //练习3.求1--某个数之间可以被7整除的数的个数
    public static int method3(int n) {
        int count = 0;
        for(int i = 1;i <= n;i++) {
            if(i % 7 == 0) {
                count++;
            }
```

```
        }
        return count;
    }
}
```

总结

1 方法：完成特定功能的一段代码。

作用：提高代码的可重用性、可维护性、 可读性。

语法：

访问修饰符　其他修饰符　返回类型　方法名(参数列表){

方法体

}

```
public static  void method1(){

}

public static void  method2(int x){

}

public static  int method(){

    return 10;

}
```

方法调用：

方法名();

方法名(参数);

如果方法有返回值，需要定义变量接收。

2 方法重载

同一个类中，方法名相同，参数列表不同

（1）参数列表不同： 个数不同，类型不同，顺序不同

（2）和修饰符、返回值没有关系

3 递归算法 ： 在一个方法内部调用自己本身。

3.1 阶乘

3.2 求1-100的和

3.3 求第30个斐波那契数

默写

1.Java中的循环语句有那些
2.各种循环语句的特点?

作业

编程题(使用方法实现，调用方法)

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）。

6.求2∽某个数之内的素数。【素数： 只能被1或本身整除的数】

7.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。

8.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）

9.求2+22+222+2222。

```
int a=0,sum=0;
for(int n=1;n<=4;n++){
        a=(a*10)+2;
        sum+=a;
}
System.out.print("sum="+sum);
```

10.使用递归实现10 的阶乘。

11.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

面试题

1.方法的传参过程是如何工作的

在调用方法的使用，在方面参数中写入实参，jvm运行时，会把实参赋值给形参。

2.return关键字的用法有哪些，举例说明

```
return;  //结束方法 ,可以省略

return  sum;  //返回结果 sum ，结束方法
```

3.什么是方法的重载？举例说明

同一个类中，方法名相同，方法参数列表不同