3-JavaSE之程序逻辑控制和方法的定义与使用

> *本节目标*
>
> 1. Java中程序的结构与逻辑控制
> 2. Java中方法的定义与使用

上课之前，我们先看以下这段程序的输出

```
int line = 10;
for (int x = 0; x < line; x++) { //控制行数
    for (int y = 0; y < line -x  ; y++) {//控制空格
            System.out.print(" ");
        }
    for (int z = 0; z < x ; z++ ) {
            System.out.print("* ");
        }
    System.out.println();
    }
```

1.Java程序结构与逻辑控制

在Java中，程序一共有三种结构：顺序结构、分支结构、循环结构

1.1 Java分支结构

分支结构：进行逻辑判断，当满足某些条件时才会执行某些语句

1.1.1 if语句

if语句一共有两种形式：

```
if(布尔表达式){
    //条件满足时执行代码
    }else{
    //条件不满足时执行代码
    }
```

```
if(布尔表达式){
    //条件满足时执行代码
    }else if(布尔表达式){
    //条件满足时执行代码
    } ...
    else{
    //条件都不满足时执行代码
    }
```

使用if、else可以实现对条件的判断，但是如果进行多值判断，可以使用switch语句

1.1.2 switch语句

switch语句语法如下：

```
switch(数字|枚举|字符|字符串){
    case 内容1 : {
        内容满足时执行语句;
        [break;]
        }
    case 内容2 : {
        内容满足时执行语句;
        [break;]
        }
    ...
    default:{
        内容都不满足时执行语句;
        [break;]
        }
}
```

switch范例：

```
System.out.println("请输入字符:");
        char x = (char) System.in.read();
        switch (x){
            case 'a':{
                System.out.println("我是猪");
            }
            case 'b':
            {
                System.out.println("我不是猪");
            }
            default:{
                System.out.println("我真的不是猪");
            }
        }
```

switch开关语句，若case之后没有break，则会满足case之后的所有语句一直执行直到break或全部结束

1.2 循环结构

循环结构：某几行代码被一直重复执行

while循环语法:

```
while(循环结束条件判断){
    循环语句;
    修改修改循环结束判断;
    }
```

do-while循环：

```
do{
    循环语句;
    修改循环结束判断;
}while(循环结束条件判断);
```

使用while的最大特点：如果条件不成立，一次也不执行；而do-while，至少执行一次

do-while范例：

```
        int x =10;
        do {
            System.out.println("我是猪");
            x++;
        }while (x<10);
```

以后的开发之中，对于do、while基本不使用，了解即可。

for循环语法：

```
for(循环初始化;循环结束判断;修改循环结束条件){
    循环体;
}
```

课后编程：使用for循环打印乘法口诀表。输出如下：

```
1*1=1
2*1=2   2*2=4
3*1=3   3*2=6   3*3=9
4*1=4   4*2=8   4*3=12  4*4=16
5*1=5   5*2=10  5*3=15  5*4=20  5*5=25
6*1=6   6*2=12  6*3=18  6*4=24  6*5=30  6*6=36
7*1=7   7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
8*1=8   8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64
9*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

提示(System.out.println("\t")表示空格

循环使用原则：

1. 对于不知道循环执行次数，但知道循环结束条件的，使用while
2. 明确知道循环次数的，使用for

1.3 循环控制(continue、break)

1. continue:执行到此语句时会跳过当前循环的剩余部分，返回循环判断。
2. break:退出整个循环

范例：观察continue和break在循环中的作用。

2.Java中方法的定义与使用

2.1 方法的定义

方法是一段可以被重复调用的代码块

注：本节课的所有方法必须在主类中定义，并且在主方法中调用。

方法的声明:

```
public static 方法返回值 方法名称（[参数类型 变量 ...]){
    方法体代码;
    [return 返回值];
}
```

当方法以void关键字声明，那么此方法没有返回值；若有返回值，返回值可以为基本类型和引用类型。

```
public class Test{
    public static void main(String[] args) {
        System.out.println(add(5,5));
    }
    public static int add(int x,int y){
        return x+y;
    }
}
```

(重要) 如果方法以void声明，那么可以使用return来结束调用（常常与if语句配合使用）

```
public class Test{
    public static void main(String[] args) {
        myPrint(1);
        myPrint(2);
        myPrint(3);
        myPrint(4);
    }
    public static void myPrint(int x){
        if (x==2) {
            return ;//若执行此语句，则此语句后面的代码不被执行，方法结束调用。
        }else {
            System.out.println(x);
        }
    }
}
```

2.2 方法重载（重要）

定义：方法名称相同，参数的类型或个数不同

方法的签名：指的是方法名与参数，返回类型不是签名的一部分

(重要) 不能有两个名字相同、参数类型也相同却返回不同类型值的方法

范例：重载方法实例

```
public class Test{
        public static void main(String[] args) {
        System.out.println(add(5,5));
        System.out.println(add(5,5,55));
    }
    public static int add(int x,int y){
        return x+y;
    }
    public static int add(int x,int y,int z){
        return x+y+z;
    }
}
```

开发原则：在进行方法的重载时，要求：方法的返回值一定相同！

2.3 方法递归

定义：指的是一个方法自己调用自己的方式。递归方法的特点：

1. 方法必须有递归的结束条件
2. 方法在每次递归处理的时候一定要作出一些变更

范例：递归实现从1到100的叠加

```
public class Test{
        public static void main(String[] args) {
            System.out.println(sum(100));
    }
    public static int sum(int num){
        if (num == 1) {
            return 1;
        }else {
            return num+sum(num-1);
        }
    }
}
```

通过代码发现，使用while循环的操作，大部分都可以使用递归替换。之所以使用递归，因为方法可执行的操作更多，结构也更好

递归是数据结构的第一步，同学们需要时间去理解、掌握

课后编程作业

1. 使用for循环打印乘法口诀表
2. 递归实现60!
3. 附加题：使用递归实现快速排序