

1、纲要.....	1
2、内容.....	2
2.1、标识符.....	2
2.2、关键字.....	2
2.3、数据类型.....	3
2.3、进制换算.....	3
2.4、字符编码.....	4
2.5、变量.....	5
2.6、数据类型详解.....	6
2.6.1、整数型.....	6
2.6.2、浮点类型.....	6
2.6.3、布尔类型.....	7
2.6.4、基本类型的转换.....	8
2.7、运算符.....	10
2.6.1、算术运算符.....	11
2.7.2、关系运算符和布尔运算符.....	12
2.7.3、赋值类运算符.....	13
2.7.4、条件运算符.....	14
2.8、控制语句.....	14
2.8.1、if 语句和 if else 语句.....	14
2.8.2、switch 语句.....	16
2.8.3、for 语句.....	18
2.8.4、while 语句.....	18
2.8.5、do while 语句.....	19
2.8.6、break 语句.....	20
2.8.7、continue 语句.....	21
2.9、方法初步.....	21
2.9.1、普通方法示例.....	21
2.9.2、方法的重载（Overload）.....	23
3.0、递归问题.....	25
3、练习题.....	27

1、纲要

标识符
关键字
数据类型
运算符
控制语句
方法说明
递归问题

2、内容

2.1、标识符

标识符可以标识类名，接口名，变量名，方法名

1. Java 标识符的命名规则
 - a) 标识符是由，**数字，字母，下划线和美元符号**构成，其他符号不可以
 - b) 必须以**字母、下划线或美元符号**开头，不能以**数字**开头
2. 关键字不能作为标识符
3. 标识符区分大小写
4. 标识符理论上没有长度限制

参见表格

合法标识符	不合法标识符
_123Test	123Test
HelloWorld	Hello-World
HelloWorld	HelloWorld#
public1	public
HelloWord	Hello World

命名 Java 标识符，最好**见名知意**

2.2、关键字

在 Java 中关键字都是**小写**的

class	extends	implements	interface	import
package	Break	case	continue	default
do	if	else	for	return
switch	while	false	true	null

boolean	byte	char	short	int
long	float	double	try	catch
throw	throws	finally	abstract	final
native	private	protected	public	static
synchronized	transient	volatile	instanceof	new
super	void	assert	enum	<u>goto</u>
<u>const</u>				

2.3、数据类型

Java 总共有两种数据类型，主要有基本类型和引用类型，基本类型有 8 种，引用数据类型有 3 种

数据类型

- 基本数据类型
 - 数值类型
 - ◆ 整数型(byte,short,int,long)
 - ◆ 浮点型(float,double)
 - 字符类型(char)
 - 布尔类型(boolean,只能取值 true 和 false)
- 引用数据类型
 - 数组
 - 类
 - 接口

八种数据类型的取值范围

类型描述	关键字	字节数	取值范围	默认值
字节型	byte	1个字节	$-2^7 \sim 2^7-1$ (-128~127)	0
短整型	short	2个字节	$-2^{15} \sim 2^{15}-1$ (-32768~32767)	0
整型	int	4个字节	$-2^{31} \sim 2^{31}-1$ (-2147483648~ 2147483647)	0
长整型	long	8个字节	$-2^{63} \sim 2^{63}-1$ (-9223372036854775808~ 9223372036854775807)	0
单精度浮点型	float	4个字节	大约 $\pm 3.40282347E+38F$ (有效位数6~7位)	0.0f
双精度浮点型	double	8个字节	大约 $\pm 1.79769313486231570E+308$ (有效位数15位)	0.0d
字符型	char	2个字节	$0 \sim 2^{16}-1$ 从0~65535	'\u0000'
布尔型	boolean	1个字节	true/false	false

在计算机内部，所有信息都采用二进制表示，每个二进制由 0 和 1 两种状态，一个字节有 8 位，也就是由 8 个 0 或 1 构成，如果 short 类型的 6 在计算机中是如何存储的，short 是两个字节，那么 short 6 的二进制为：00000000 00000110,int 类型的 6 在计算机中存储为 32 位：

00000000 00000000 00000000 00000110

2.3、进制换算

- ## ● 简单了解十进制到二进制的换算

规则：除 2 取余，逆序输出

如 10 进制 6 二进制换算方式为:

$$6/2=3 \text{ 余 } 0$$
$$3/2=1 \text{ 余 } 1$$
 $1/2=0$ 余 1

将余数逆序输出就是 6 的二进制表示: 110, 位数不够补零

- ## ● 简单了解二进制到十进制的换算

规则：取出最后一位，从2的0次方开始乘，将得到的结果相加即可。

如：二进制的 110 的十进制换算：

0×2 的 0 次方 $= 0$

1*2 的 1 次方=2

1*2 的 2 次方=4

110 的十进制为: $0+2+4=6$

2.4、字符编码

ASCII 字符编码	采用一个字节编码，主要针对英文编码
ISO-8859-1	有称 latin-1,是国际化标准或组织 ISO 制定的，主要为了西欧语言中的字符编码，和 ASCII 兼容
GB2312/GBK/GB18030	主要是汉字编码，三种编码从容量上看是包含关系
unicode	Unicode 统一了全世界上的所有文字编码，unicode 有几种实现：UTF-8,UTF-16,UTF-32

UTF-8 存储格式 (UTF8 主要就是为了节省空间):

0																
1	1	0	1	0								
1	1	1	0	1	0	1	0

Char 的测试

```
public class CharTest {
```

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

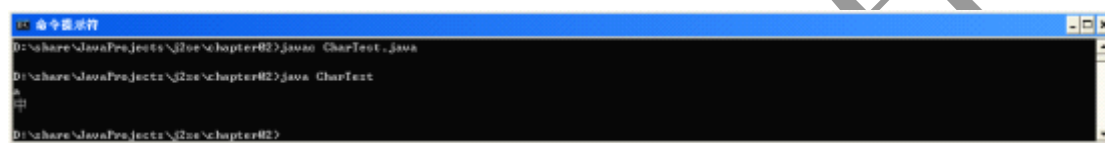
//不能采用双引号声明字符类型，必须采用单引号声明字符类型

```
//char c1 = "a";

//声明一个字符
char c1 = 'a';

//char 类型可以存放一个汉字，java 中的 char 使用 utf-16 编码
//所以 char 占用两个字节
char c2 = '中';

System.out.println(c1);
System.out.println(c2);
}
}
```



2.5、变量

变量其实是 java 中的一个最基本的单元，也就是内存中的一块区域，Java 中的变量有四个基本属性：变量名，数据类型，存储单元和变量值

- 变量名：合法的标识符
- 变量的数据类型：可以是基本类型和引用类型（必须包含类型）
- 存储单元：存储单元大小是由数据类型决定的，如：int 为 4 个字节 32 位
- 变量值：在存储单元中放的就是变量值（如果是基本类型放的就是具体值，如果是引用类型放的是内存地址，如果 null，表示不指向任何对象）

变量的声明格式：

类型 变量名；

【示例代码】

```
public class VarTest01 {

    public static void main(String[] args) {

        //定义变量
        int age;

        //输出变量
        System.out.println(age);

    }
}
```

采用 javac 编译 VarTest01.java，出现如下错误：

出现错误的原因是：变量没有初始化，变量初始化的过程就是赋值，变量声明后必须初始化，以下示例是正确的：

```
public class VarTest02 {  
  
    public static void main(String[] args) {  
  
        //定义变量,赋值为 100  
        int age = 100;  
  
        //输出变量  
        System.out.println(age);  
    }  
}
```

2.6、数据类型详解

2.6.1、整数型

Java 整型包括：byte/short/int/long

Java 语言整数型常量有三种表示方法

- 十进制
- 八进制，八进制 0 开头，如：013
- 十六进制，十六进制 0x 开头，如：0x23

Java 语言整数型默认为 int 类型，如果要声明成 long 类型在变量值后加入 L，如：

```
long l = 9999999999999L
```

2.6.2、浮点类型

Java 语言中浮点类型包括：float/double

Java 语言中浮点类型默认为 double

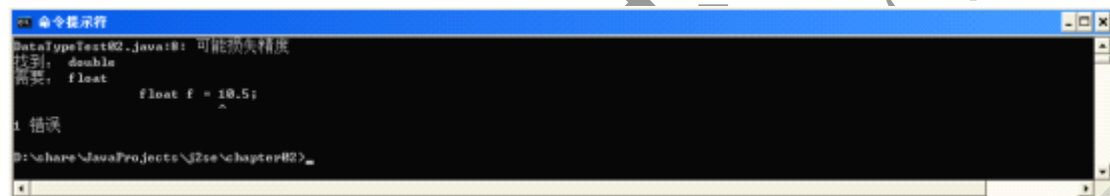
【代码示例】

```
public class DataTypeTest01 {
```

```
public static void main(String[] args) {  
  
    //正确，因为默认为 double 类型  
    double d = 10.5;  
  
}
```

【示例代码】

```
public class DataTypeTest02 {  
  
    public static void main(String[] args) {  
  
        //会出现错误  
        //因为 10.5 默认为 double 类型，double 为 8 个字节，  
        //而 float 为 4 个字节，所以 double 向 float 赋值会出现精度丢失的问题  
        float f = 10.5;  
  
    }  
}
```



【示例代码】，改善以上示例

```
public class DataTypeTest03 {  
  
    public static void main(String[] args) {  
  
        //声明为 float 类型的变量，数值后面必须加入 f  
        float f = 10.5f;  
  
    }  
}
```

2.6.3、布尔类型

布尔类型的取值只能是 **true** 和 **false**，不能取其他的

【代码示例】

```
public class DataTypeTest04 {  
  
    public static void main(String[] args) {  
  
        //boolean 类型只能取值为 true 和 false  
  
        boolean a = true;
```

```
        boolean b = false;
    }
}
```

【代码示例】

```
public class DataTypeTest05 {

    public static void main(String[] args) {

        //boolean 类型只能取值为 true 和 false

        boolean a = 1;

        boolean b = 0;

    }
}
```

2.6.4、基本类型的转换

- 在 java 中基本类型可以相互转换，boolean 类型比较特殊不可以转换成其他类型
- 转换分为默认转换和强制转换：
 - 默认转换：容量小的类型会默认转换为容量大的类型
 - ◆ byte-->short-->int-->long-->float-->double
 - ◆ char-->
 - ◆ byte、short、char 之间计算不会互相转换，首先先转换成 int
 - 强制转换
 - ◆ 将容量大的类型转换成容量小的类型，需要进行强制转换
 - ◆ 注意：只要不超出范围可以将整型值直接赋值给 byte，short，char

在多种类型混合运算过程中，首先先将所有数据转换成容量最大的那种，再运算

【示例代码】

```
public class DataTypeTest06 {

    public static void main(String[] args) {

        //出现错误，1000 超出了 byte 的范围
        //byte a = 1000;

        //正确，因为 20 没有超出 byte 范围
        //所以赋值
        byte a = 20;

        //变量不能重名
        //short a = 1000;
```


//正确，因为数值 1000 没有超出 short 类型的范围

//所以赋值正确

```
short b = 1000;
```

//正确，因为默认就是 int，并且没有超出 int 范围

```
int c = 1000;
```

//正确，可以自动转换

```
long d = c;
```

//错误，出现精度丢失问题，大类型-->小类型会出现问题

```
//int e = d;
```

//将 long 强制转换成 int 类型

//因为值 1000，没有超出 int 范围，所以转换是正确的

```
int e = (int)d;
```

//因为 java 中的运算会会转成最大类型

//而 10 和 3 默认为 int,所以运算后的最大类型也是 int

//所以是正确的

```
int f = 10/3;
```

//声明 10 为 long 类型

```
long g = 10;
```

//出现错误，多个数值在运算过程中，会转换成容量最大的类型

//以下示例最大的类型为 double，而 h 为 int，所以就会出现大类型（long）到小类型（int）

//的转换，将会出现精度丢失问题

```
//int h = g/3;
```

//可以强制转换,因为运算结果没有超出 int 范围

```
//int h = (int)g/3;
```

//可以采用 long 类型来接收运算结果

```
//long h = g/3;
```

//出现精度损失问题，以下问题主要是优先级的的问题

//将 g 转换成 int，然后又将 int 类型的 g 转换成 byte,最后 byte 类型的 g 和 3 运算，那么

//它的运算结果类型就是 int，所以 int 赋值给 byte 就出现了精度损失问题

```
//byte h = (byte)(int)g/3;
```

```
//正确
//byte h = (byte)(int)(g/3);

//不能转换,还有因为优先级的问题
//byte h = (byte)g/3;

//可以转换, 因为运算结果没有超出 byte 范围
//byte h = (byte)(g/3);

//可以转换, 因为运算结果没有超出 short 范围
short h = (short)(g/3);

short i = 10;

byte j = 5;

//错误, short 和 byte 运算, 首先会转换成 int 再运算
//所以运算结果为 int, int 赋值给 short 就会出现精度丢失问题
//short k = i + j;

//可以将运算结果强制转换成 short
//short k = (short)(i + j);

//因为运算结果为 int, 所以可以采用 int 类型接收
int k = i + j;

char l = 'a';
System.out.println(l);

//输出结果为 97, 也就是 a 的 ascii 值
System.out.println((byte)l);

int m = l + 100;
//输出结构为 197,取得 a 的 ascii 码值, 让后与 100 进行相加运算
System.out.println(m);

}
}
```

2.7、运算符

按功能划分主要运算符如下:

算术运算符	+, -, *, /, ++, --, %
-------	-----------------------

关系运算符	<, <=, >, >=, ==, !=
布尔运算符	&&, , &, , !, ^
位运算符（目前用的比较少，以后有时间再讲）	&, , ~, ^, >>, >>>, << & 按位与（AND）[真真为真,真假为假] 按位或（OR）[假假为假,其余全真] ^ 按位异[相同为假,不同为真] ~按位非（NOT）[真则假,假则真] >> 右移 >>> 右移，左边空出的位以 0 填充 << 左移
赋值类运算符	=, +=, -=, *=, /=, %=
字符串连接运算符	+
条件运算符	?:
其他运算符	instanceof, new

2.6.1、算术运算符

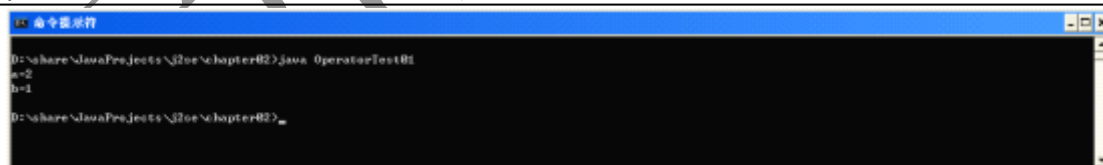
【示例代码】

```
public class OperatorTest01 {
```

```

    public static void main(String[] args) {
        int a = 1;

        //a++相当于 a=a+1;
        int b = a++;
        System.out.println("a=" + a);
        System.out.println("b=" + b);
    }
}
```



以上会看到 a=2,b=1,为什么会出现这种结果？

++在变量的后面，先把值赋值给 b，然后 a 再加（也就是先赋值再自加）所以就输出了 a=2,b=1

【示例代码】，将++放到变量的前面

```
public class OperatorTest02 {
```

```

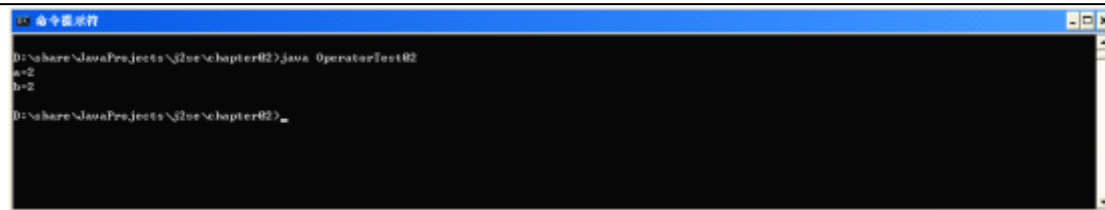
    public static void main(String[] args) {
        int a = 1;

        int b = ++a;
    }
}
```

```

        System.out.println("a=" + a);
        System.out.println("b=" + b);
    }
}

```



输出结果为 a=2,b=2,如果++在变量的前面,是先自加在赋值

【示例代码】取余/取模

```

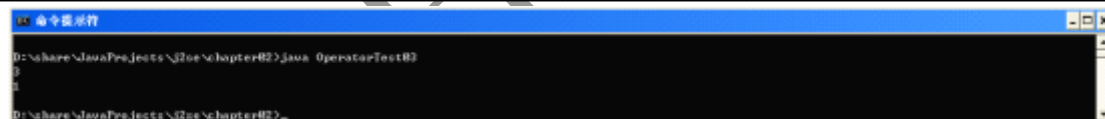
public class OperatorTest03 {

    public static void main(String[] args) {

        int a = 10/3;
        System.out.println(a);

        //取余
        int b = 10 % 3;
        System.out.println(b);
    }
}

```



2.7.2、关系运算符和布尔运算符

- 与：两个操作数相与，如果都为 true，则为 true
- 或：两个操作数相或，有一个为 true，则为 true
- 异或：相异为 true，两个操作数不一样就为 true
- 短路与和逻辑与的区别？
 - 短路与，从第一个操作数推断结果，只要有一个为 false，不再计算第二个操作数
 - 逻辑与，两个操作数都计算
- 短路或和逻辑或的区别？
 - 短路或，从第一个操作数推断结果，只要为 true，不再计算第二个操作数
 - 逻辑或，两个操作数都计算

短路与和逻辑与、短路或和逻辑或虽然计算方式不一样，但结果都是一样的

注意：操作数必须是 boolean 型

op1	op2	op1&&op2	op1 op2	op1^op2	!op1
-----	-----	----------	----------	---------	------

		op1&op2	op1 op2		
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

【代码示例】

```

public class OperatorTest04 {

    public static void main(String[] args) {

        boolean op1 = (10 > 5);

        //输出 true
        System.out.println("op1=" + op1);

        boolean op2 = (10 < 5);
        System.out.println("op2=" + op2);

        System.out.println("op1 && op1 =" + (op1 && op2));
        System.out.println("op1 & op1 =" + (op1 & op2));

        System.out.println("op1 || op1 =" + (op1 || op2));
        System.out.println("op1 | op1 =" + (op1 | op2));

        System.out.println("op1 ^ op1 =" + (op1 ^ op2));

        System.out.println("!op1 =" + !op1);

    }
}

```

```

D:\share\java\projects\j2se\chapter02>java OperatorTest04
op1=true
op2=false
op1 && op1 =false
op1 & op1 =false
op1 || op1 =true
op1 | op1 =true
op1 ^ op1 =false
!op1 =false
D:\share\java\projects\j2se\chapter02>

```

2.7.3、赋值类运算符

+=	a+=b	a=a+b
-=	a-=b	a=a-b
=	a=b	a=a*b

/=	a/=b	a=a/b
%=	a%=b	a=a%b

【代码示例】

```
public class OperatorTest05 {  
  
    public static void main(String[] args) {  
        int a = 1;  
        int b = 2;  
  
        //a=a + b  
        a+=b;  
        System.out.println(a);  
    }  
}
```

2.7.4. 条件运算符

条件运算符是 java 语言中的三元运算，格式如下：

op1 ? op2 : op3

如果操作数 op1 为 true 则输出 op2，否则输出 op3

```
public class OperatorTest06 {  
  
    public static void main(String[] args) {  
        int a = 11;  
  
        int b = a>0?1:-1;  
  
        System.out.println(b);  
  
        boolean c = a%2==0?true:false;  
        System.out.println(c);  
    }  
}
```

2.8、控制语句

java 控制语句可以分为 7 种：

- 控制选择结构语句
 - if、if else
 - switch
- 控制循环结构语句
 - for
 - while

- do while
- 改变控制语句顺序
 - break
 - continue

2.8.1、if 语句和 if else 语句

条件语句只要包括 if 语句和 if else 语句

- if 语句格式：
if (布尔表达式) {
 一条或多条语句
}

如果语句为一条，可以省略大括号，如：

```
if(布尔表示式)  
    一条语句
```

只有表示为 true 时，才会进入 if 语句中

- if else 语句格式
if (布尔表示式) {
 一条或多条语句
} else {
 一条或多条语句
}

如果 else 时，还需要条件，可以写成如下可是：


```
if (布尔表示式) {  
    一条或多条语句  
} else if(布尔表示式) {  
    一条或多条语句  
} else {  
    一条或多条语句  
}
```

【示例代码】

```
public class IfelseTest01 {  
  
    public static void main(String[] args) {  
  
        int age = 3;  
  
        if (age % 2 == 0) {  
            System.out.println("是偶数");  
        }  
  
        //建议只有一条语句也加入大括号  
        if (age % 2 == 0)  
            System.out.println("是偶数");  
    }  
}
```

```
if ( age >0 && age <=5){
    System.out.println("幼儿");
}
if (age >5 && age <=10) {
    System.out.println("儿童");
}
if (age >10 && age<=18) {
    System.out.println("少年");
}

//这种方法效率会好一些，如果年龄为 5 岁
//以下代码只判断一次就可以，而上面的代码会判断多次
if (age >0 && age <=5) {
    System.out.println("幼儿");
} else if (age >5 && age <=10){
    System.out.println("儿童");
} else if (age >10 && age<=18) {
    System.out.println("少年");
} else {
    System.out.println("青年");
}
}
```



```
D:\share\javaProjects\j2se\chapter02>java IfElseTest01 .java
幼儿
儿童
D:\share\javaProjects\j2se\chapter02>
```

2.8.2、switch 语句

switch 也称为多重分支，具体格式如下

```
switch (表达式) {
    case 值 1:
        语句
        break;
    case 值 2:
        语句
        break;
    default:
        语句
        Break;
}
```

说明：

- 表达式的值只能为：char、byte、short、int 类型，boolean、long、float、double 都是非法的
- break 语句可以省略,但会出现 switch 穿透
- default 语句也可以省略，一般不建议省略，并且放置在最后

注意类的命名：首字母要大写，单词之间首字母大写，这种命名方式称为“驼峰标识”

【代码示例】

```
public class SwitchTest01 {  
  
    public static void main(String[] args) {  
        char c = 'd';  
  
        switch(c) {  
            case 'a':  
                System.out.println("优秀");  
                break; //注意 break  
            case 'b':  
                System.out.println("良好");  
                break;  
            case 'c':  
                System.out.println("一般");  
                break;  
            default:  
                System.out.println("很差");  
        }  
        System.out.println("switch 执行结束！");  
    }  
}
```

【代码示例】

```
public class SwitchTest02 {  
  
    public static void main(String[] args) {  
        //byte c = 1;  
        //short c = 1;  
        //int c = 1;  
        //不能为 long，switch 只能为 byte、short、int、char  
        long c = 1;  
  
        switch(c) {  
            case 1:  
                System.out.println("优秀");  
                break;  
            case 2:  
                System.out.println("良好");  
                break;  
        }  
    }  
}
```

```
        case 3:
            System.out.println("一般");
            break;
        default:
            System.out.println("很差");
    }
    System.out.println("switch 执行结束! ");
}
```

2.8.3、for 语句

for 语句格式如下：

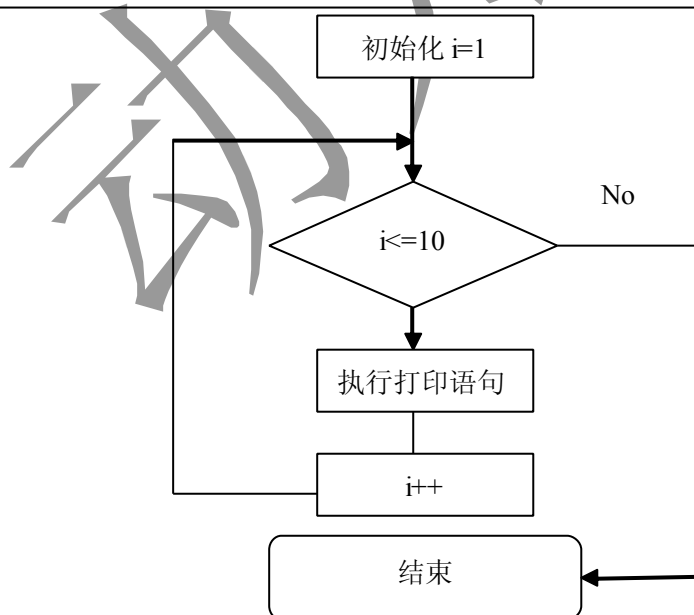
```
for(初始化部分表达式； 条件表达式； 更新表达式) {
    一条或多条语句
}
```

【代码示例】

```
public class ForTest01 {

    public static void main(String[] args) {

        for (int i=1; i<=10; i++) {
            System.out.println(i);
        }
    }
}
```



2.8.4、while 语句

while 语句格式

```
while(布尔表达式) {  
    一条或多条语句  
}
```

【代码示例】

```
public class WhileTest01 {  
  
    public static void main(String[] args) {  
  
        int i = 1;  
  
        //注意死循环问题  
        while(i<=10) {  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

以上程序同样完成了 1~10 的输出，可以看出 for 更加简单，可以看做 for 是 while 语句的便利方式，采用 for 语句完全可以模仿 while 语句，如将 for 语句中的“初始化部分表达式”和“更新表达式”省略

```
for( ; 条件表达式 ; ) {  
    语句  
}
```

```
public class WhileTest02 {  
  
    public static void main(String[] args) {  
  
        int i = 1;  
        for(;i<=10;) {  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

2.8.5、do while 语句

do while 格式

```
do {  
    语句
```

}while(布尔表达式);

注意 while 括号后必须写分号

do while 与 while 非常相似，不同点在于 do while 先执行循环体，也就是说不管条件符不符合，循环体至少执行一次

【代码示例】

```
public class DoWhileTest01 {  
  
    public static void main(String[] args) {  
  
        int i = 1;  
        do {  
            System.out.println(i);  
            i++;  
        } while(i<=10); //注意分号  
    }  
}
```

2.8.6、break 语句

break 可以用在 switch、循环语句和带标号的语句块中

- 在循环语句中主要是为了终止循环

```
public class BreakTest01 {  
  
    public static void main(String[] args) {  
        for (int i=1; i<=100; i++) {  
            System.out.println(i);  
            if (i == 50) {  
                break; //会跳出当前循环  
            }  
        }  
        //break 跳到这里  
    }  
}
```

- 在循环语句中主要是为了终止循环（多重循环）

```
public class BreakTest02 {  
  
    public static void main(String[] args) {  
        for (int i=1; i<=5; i++) {  
            System.out.println("i===== " + i);  
            for (int j=1; j<=10; j++) {  
                System.out.println(j);  
  
                if (j == 5) {
```

```
        break;
    }
}
//以上 break 会到此为止
}
//以上 break 不会跳到这里
}
}
```

2.8.7、continue 语句

continue 只能用在循环语句中，表示在循环中执行到 continue 时，自动结束本次循环，然后判断条件，决定是否进行下一次循环（多个循环的跳出使用---标签：的方式）

```
public class ContinueTest01 {

    public static void main(String[] args) {
        for (int i=1; i<=100; i++) {
            //if (i % 2 != 0) {
            //    System.out.println(i);
            //}
            if (i % 2 == 0) {
                continue; //继续下一次循环
            }
            System.out.println(i);
        }
    }
}
```

2.9、方法初步

方法是可以重复调用的代码块，通常为了实现各种功能
方法的定义格式：

```
[方法修饰列表] 返回值类型 方法名（方法参数列表）{
    方法体
}
```

- 方法修饰列表

是可选项，方法的修饰符可以包括：**public**,**protected**,**private**,**abstract**,**static**,**final**,**synchronized**，其中 **public**,**protected**,**private** 不能同时存在

- 返回值类型

如果没有返回值使用 **void** 关键字，如果存在返回值可以是基本类型和引用类型，**如果存在返回值，使用 return 语句。Return 语句后面不能再执行语句，因为不可能执行到，编译器会发生错误。**

- 方法名
任意合法的标识符
- 方法参数列表
参数列表可以多个，如：method1(int a, int b),多个采用逗号分割

2.9.1、普通方法示例

【代码示例】，存在返回值

```
public class MethodTest01 {  
  
    public static void main(String[] args) {  
        String s = method1(1);  
        System.out.println(s);  
    }  
  
    public static String method1(int c) {  
        String retValue = "";  
        switch(c) {  
            case 1:  
                //System.out.println("优秀");  
                retValue = "优";  
                break;  
            case 2:  
                //System.out.println("良好");  
                retValue = "良好";  
                break;  
            case 3:  
                //System.out.println("一般");  
                retValue = "一般";  
                break;  
            default:  
                //System.out.println("很差");  
                retValue = "很差";  
        }  
        return retValue;  
    }  
}
```

【代码示例】，没有返回值

```
public class MethodTest02 {  
  
    public static void main(String[] args) {  
        method1(1);  
    }  
}
```

```
}

public static void method1(int c) {

    switch(c) {
        case 1:
            System.out.println("优秀");
            break;
        case 2:
            System.out.println("良好");
            break;
        case 3:
            System.out.println("一般");
            break;
        default:
            System.out.println("很差");
    }
}
}
```

2.9.2、方法的重载（Overload）

重载的条件

- 方法名相同
- 方法的参数类型，个数，顺序至少有一个不同
- 方法的返回类型可以不同（不依靠返回类型来区分重载）
- 方法的修饰符可以不同，因为方法重载和修饰符没有任何关系
- 方法重载只出现在同一个类中

【代码示例】

```
public class OverloadTest01 {

    public static void main(String[] args) {
        int retInt = sumInt(10, 20);
        System.out.println(retInt);
        float retFloat = sumFloat(1.5f, 2.5f);
        System.out.println(retFloat);
        double retDouble = sumDouble(2.2, 3.2);
        System.out.println(retDouble);
    }

    //对 int 求和
    public static int sumInt(int v1, int v2) {
        return v1+v2;
    }
}
```

```
}

//对 float 求和
public static float sumFloat(float v1, float v2) {
    return v1+v2;
}

//对 double 求和
public static double sumDouble(double v1, double v2) {
    return v1+v2;
}
}
```

【代码示例】，采用重载改善以上代码，重载会使我们的编程风格会更好

```
public class OverloadTest02 {

    public static void main(String[] args) {
        int retInt = sum(10, 20);
        System.out.println(retInt);
        float retFloat = sum(1.5f, 2.5f);
        System.out.println(retFloat);
        double retDouble = sum(2.2, 3.2);
        System.out.println(retDouble);
    }

    //对 int 求和
    public static int sum(int v1, int v2) {
        return v1+v2;
    }

    //对 float 求和
    public static float sum(float v1, float v2) {
        return v1+v2;
    }

    //对 double 求和
    public static double sum(double v1, double v2) {
        return v1+v2;
    }

    //正确
    public static double sum() {
        return 0L;
    }
}
```



```
//错误，重载不依赖返回值
//public static void sum() {
//    return 0L;
//}

//正确
public static double sum(double v1, double v2, int v3) {
    return 0L;
}

//正确
public static double sum(int v3, double v1, double v2) {
    return 0L;
}

//正确
public static double sum(double v1, int v3, double v2) {
    return 0L;
}

//不正确
//public static double sum(double v2, double v1) {
//    return 0L;
//}
}
```

3.0、递归问题

递归：指方法调用自身

先不使用递归计算 1+2+3+4+5 的和，要求程序设计灵活，如果传入一个 5 过求出 1+2+...+5 的和

- 不使用递归求和

```
public class RecursionTest01 {

    public static void main(String[] args) {
        int retValue = method1(5);
        System.out.println(retValue);
    }

    //给指定的值求和
    public static int method1(int n) {
```

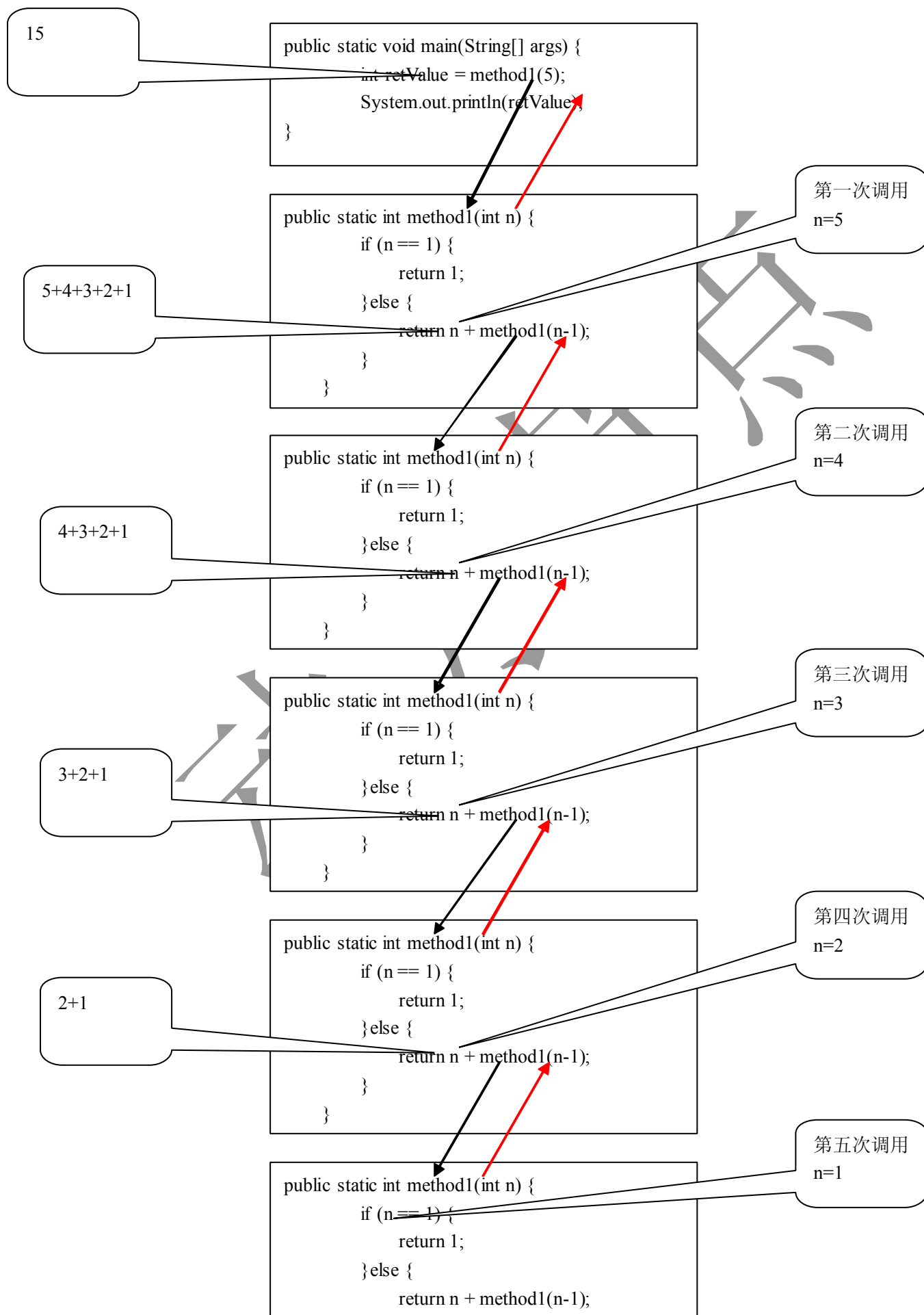
```
int s = 0;
for (int i=1; i<=n; i++) {
    //s=s+i;
    s+=i;
}
return s;
}
```

- 使用递归求和

```
public class RecursionTest02 {

    public static void main(String[] args) {
        int retValue = method1(5);
        System.out.println(retValue);
    }

    //采用递归求和
    public static int method1(int n) {
        if (n == 1) {
            return 1;
        } else {
            //递归调用，调用自身
            return n + method1(n-1);
        }
    }
}
```



阅读：《深入 java 虚拟机第二版》第五章

3、练习题

1. 将 1~100 之间的奇数求和，需要采用+=来取得和；

```
public class TrainingTest01 {  
  
    public static void main(String[] args) {  
  
        int s = 0;  
  
        for (int i=1; i<=100; i++) {  
            if (i % 2 != 0) {  
                //s=s+i;  
                s+=i;  
            }  
        }  
        System.out.println(s);  
    }  
}
```

2. 分别采用递归和非递归的方式求 n 的阶乘： $n! = n*(n-1)*\dots$ ，其中 n 为大于 0 的整数，要求在控制台中输出 1 到 5 的阶乘

- 非递归

```
public class TrainingTest02 {  
  
    public static void main(String[] args) {  
        int retValue = method1(5);  
        System.out.println(retValue);  
    }  
  
    public static int method1(int n) {  
        int s = 1;  
        for (int i=1; i<=n; i++) {  
            //s=s*i;  
            s*=i;  
        }  
    }  
}
```

```
    }  
    return s;  
}  
}
```

● 递归

```
public class TrainingTest03 {  
  
    public static void main(String[] args) {  
        int retValue = method1(5);  
        System.out.println(retValue);  
    }  
  
    public static int method1(int n) {  
        if (n == 1) {  
            return 1;  
        } else {  
            return n * method1(n-1);  
        }  
    }  
}
```

