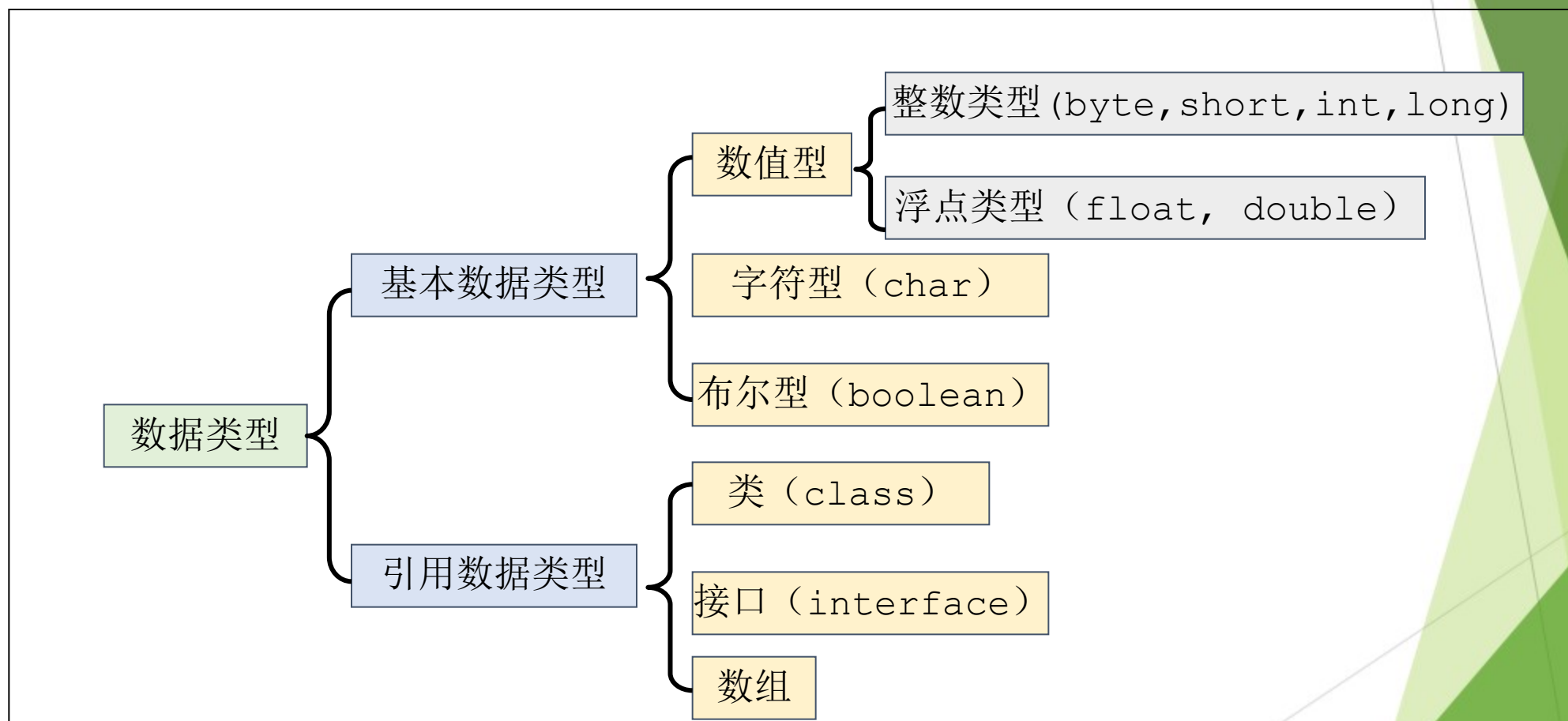
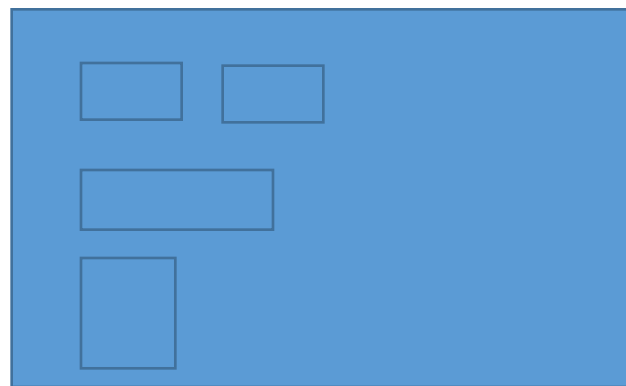


# Java数据类型的划分



# Java基本数据类型

- Java中定义了4类8种基本数据类型。
  - 逻辑型—boolean
  - 字符型—char
  - 整数型—byte, short, int, long
  - 浮点数型—float, double



# 逻辑型Boolean

- `boolean` 类型适于逻辑运算，一般用于程序流程控制。
- `boolean` 类型数据只允许取值 `true` 或 `false`，不可以 0 或非 0 的整数替代 `true` 和 `false`，这点和C语言不同。
- 用法举例：

```
boolean flag;  
flag = true;  
if(flag) {  
    //do something  
}
```

# 字符型char

- char型数据用来表示通常意义上的“字符”。
- 字符常量为用单引号括起来的单个字符，例如：
  - `char eChar = 'a'; char cChar = '中';`
- Java字符采用 **Unicode** 编码，每个字符占两个字节，因而可用十六进制编码形式表示，例如：
  - `char c1 = '\u0061';`
  - 注：Unicode是全球语言统一编码
- Java语言中还允许使用转义字符 ‘\’ 来将其后的字符转变为其它的含义，例如：
  - `char c2 = '\n';` // ‘\n’ 代表换行符
- 补充：2进制、10进制、16进制之间的转换
  - $1101 = 1 \times 1 + 0 \times 2 + 1 \times 4 + 1 \times 8$
  - $13 = 1 + 4 + 8 = 1101$
  - $1101 = D$

# 整数类型

- Java 各整数类型有固定的表数范围和字段长度，其不受具体操作系统的影响，以保证Java程序的可移植性。
- Java 语言整型常量的三种表示形式：
  - 十进制整数，如：12, -314, 0。
  - 八进制整数，要求以 0 开头，如：012。
  - 十六进制数，要求 0x 或 0X 开头，如：0x12 。
- Java语言的整型常量默认为int型，声明long型常量可以后加 ‘l’或 ‘L’，如：
  - `int i1 = 600; //正确`    `long l1 = 888888888888L; //必须加l否则会出错`

类 型	占用存储空间	表数范围
byte	1字节	-128 ~ 127
short	2字节	$-2^{15} \sim 2^{15}-1$
int	4字节	$-2^{31} \sim 2^{31}-1$
long	8字节	$-2^{63} \sim 2^{63}-1$

# 浮点类型

- 与整数类型类似，Java浮点类型有固定的表数范围和字段长度，不受平台影响。
- Java 浮点类型常量有两种表示形式
  - 十进制数形式，例如: 3.14    314.0    .314
  - 科学记数法形式，如 3.14e2    3.14E2    100E-2
- Java 浮点型常量默认为 double 型，如要声明一个常量为 float 型，则需在数字后面加 f 或 F，如：
  - `double d = 12345.6;` //正确    `float f = 12.3f;` //必须加f否则会出错
- 下面列出 Java 的各种浮点类型

类 型	占用存储空间	表数范围
float	4字节	-3.403E38~3.403E38
double	8字节	-1.798E308~1.798E308

TestVar.java

# 基本数据类型转换

- java中可以从**任意基本类型**转型到另外的基本类型
- **例外** → boolean 类型不可以转换为其他的数据类型。
- 转换分为默认转换和强制转换
- 整形，字符型，浮点型的数据在混合运算中相互转换，转换时遵循以下原则：
  - 容量小的类型默认转换为容量大的数据类型；数据类型**按容量大小**排序为：
    - ✓ **byte,short,char->int->long->float->double**
    - ✓ **byte,short,char**之间不会互相转换，他们三者`在计算时首先回转换为int类型`
  - 容量大的数据类型转换为容量小的数据类型时，要加上**强制**转换符，但可能造成精度降低或溢出；使用时要格外注意。 `long a = 123; float a = 12.3;`
  - 有多种类型的数据混合运算时，系统首先自动的将所有数据转换成容量最大的那一种数据类型，然后再进行计算。
  - 实数常量（如：1.2）默认为 double。整数常量（如：123）默认为 int。

TestConvert.java

# 课堂练习

## 课堂练习



说出下面程序片  
断中编译错误或可能  
产生计算溢出的部分

```
void public method() {  
    int i=1,j;  
    float f1=0.1;    float f2=123;  
    long l1 = 12345678,l2=88888888888;  
    double d1 = 2e20,d2=124;  
    byte b1 = 1,b2 = 2,b3 = 129;  
    j = j+10;  
    i = i/10;  
    i = i*0.1;  
    char c1='a',c2=125;  
    byte b = b1-b2;  
    char c = c1+c2-1;  
    float f3 = f1+f2;  
    float f4 = f1+f2*0.1;  
    double d = d1*i+j;  
    float f = (float) (d1*5+d2) ;  
}
```

TestConvert2.java



# 程序格式

- 大括号对齐
- 遇到{缩进，Tab/Shift+Tab
- 程序块之间加空行
- 并排语句之间加空格
- 运算符两侧加空格——有特定条件
- {前面有空格
- 成对编程

# 运算符

- Java 语言支持如下运算符：
  - 算术运算符： +, -, \*, /, %, ++, --
  - 关系运算符： >, <, >=, <=, ==, !=
  - 逻辑运算符： !, &, |, ^, &&, ||
  - **位运算符**： **&, |, ^, ~, >>, <<, >>>**
  - 赋值运算符： =
  - 字符串连接运算符： +

# 自加和自减运算符

```
public class Test {  
    public static void main(String arg[]) {  
        int i1 = 10, i2 = 20;  
        int i = i2++;  
        System.out.print("i=" + i);  
        System.out.println(" i2=" + i2);  
        i = ++i2;  
        System.out.print("i=" + i);  
        System.out.println(" i2=" + i2);  
        i = --i1;  
        System.out.print("i=" + i);  
        System.out.println(" i1=" + i1);  
        i = i1--;  
        System.out.print("i=" + i);  
        System.out.println(" i1=" + i1);  
    }  
}
```

输出:

```
i=20 i2=21  
i=22 i2=22  
i=9 i1=9  
i=9 i1=8
```

※ 注意:

- ++(--)
- 在前时先运算再取值。
- 在后时先取值再运算。

# 逻辑运算符

- 逻辑运算符:   !—逻辑非 &— 逻辑与 |— 逻辑或 ^ — 逻辑异或 &&— 短路与 || — 短路或
- &&, ||, ! 逻辑运算符只能用于boolean身上。

a	b	!a	a&b	a b	a^b	a&& b	a  b
true	true	false	true	true	false	true	true
true	false	false	false	true	true	false	true
false	true	true	false	true	true	false	true
false	false	true	false	false	false	false	false

```
public class Test{
    public static void main(String args[]){
        boolean a,b,c;
        a = true; b = false;
        c = a & b; System.out.println(c);
        c = a | b; System.out.println(c);
        c = a ^ b; System.out.println(c);
        c = !a; System.out.println(c);
        c = a && b; System.out.println(c);
        c = a || b; System.out.println(c);
    }
}
```

```
public class Test{
    public static void main(String args[]){
        int i=1,j=2;
        boolean flag1 = (i>3)&&((i+j)>5);
        //第二个操作数将不再计算
        boolean flag2 = (i<2)||((i+j)<6);
        //第二个操作数将不再计算
    }
}
```

# 赋值运算符

- 赋值运算符 (=)

- 当“=”两侧数据类型不一致时，可以适用默认类型转换或使用强制类型转换原则进行处理

```
long l = 100;          int i = (int)l;
```

- 注意：可以将整型常量直接赋值给byte, short, char等类型变量，而不需要进行强制类型转换，只要不超出其表数范围

```
byte b = 12; char c = 100;
```

```
X byte bb = 256;      X short s = -32769;
```

# 字符串连接符

- “+” 除用于算术加法运算外，还可用于对字符串进行连接操作

```
int id = 800 + 90;
```

```
String s = "hello" + "world";
```

- “+”运算符两侧的操作数中只要有一个是字符串(**String**)类型，系统会自动将另一个操作数转换为字符串然后再进行连接。

```
int c = 12;
```

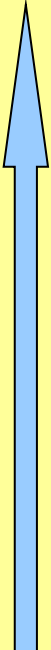
```
System.out.println("c=" + c);
```

- 当进行打印时，无论任何类型，都自动转换为字符串进行打印  

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

# 表达式

- 表达式是符合一定语法规则的运算符和操作数的序列
  - `a`
  - `5.0 + a`
  - `(a-b)*c-4`
  - `i<30 && i%10!=0`
- 表达式的类型和值
  - 对表达式中操作数进行运算得到的结果称为表达式的值。
  - 表达式值的数据类型即为表达式的类型。
- 表达式的运算顺序
  - 应按照运算符的优先级从高到低的顺序进行。
  - 优先级相同的运算符按照事先约定的结合方向进行。

R to L	. ( ) { } ; ,	<div>高</div>  <div>低</div>
L to R	++ -- ~ ! (data type)	
L to R	* / %	
L to R	+ -	
L to R	<< >> >>>	
L to R	< > <= >= instanceof	
L to R	== !=	
L to R	&	
L to R	^	
L to R		
L to R	&&	
L to R		
R to L	? :	
R to L	= *= /= %= += -= <<= >>= >>>= &= ^=  =	

# 三目条件运算符

- “三目条件运算符，语法格式：

`x ? y : z`

- 其中 `x` 为 `boolean` 类型表达式，先计算 `x` 的值，若为`true`，则整个三目运算的结果为表达式 `y` 的值，否则整个运算结果为表达式 `z` 的值。
- 举例：

```
int score = 80; int x = -100;  
String type = score < 60 ? "不及格" : "及格";  
int flag = x > 0 ? 1 : (x == 0 ? 0 : -1);  
System.out.println("type= " + type);  
System.out.println("flag= "+ flag);
```

```
type= 及格  
flag= -1
```



# 程序控制语句

- 条件语句 - 根据不同条件，执行不同语句。
  - if
  - if .. else
  - if .. else if
  - if .. else if .. else if .. else
  - switch
- 循环语句 – 重复执行某些动作
  - for
  - while
  - do .. while;

# if语句

- if
- if .. else ..
- if .. else if ..
- if .. else if .. else if ..
- if .. else if .. else if .. else
- 只有一句需要执行的语句时，可以省略 {}
- 小心不要在if后面加 ;

TestIF.java

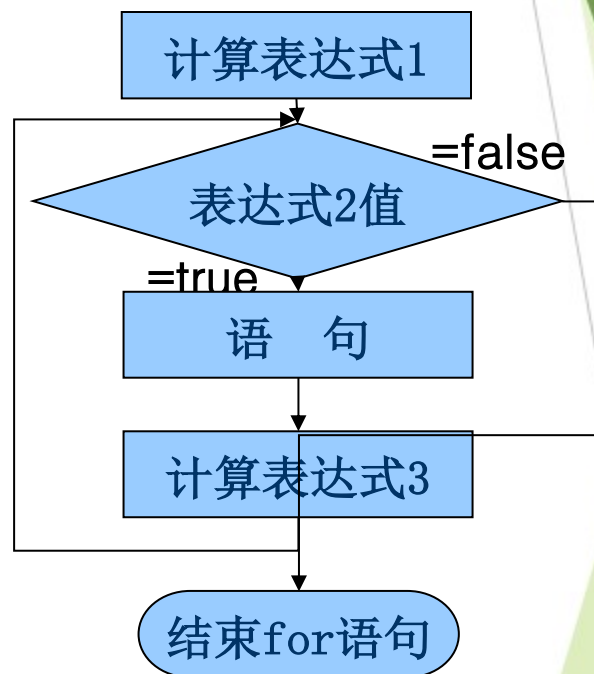
**TIPS:** 不论是不是只有一句话，都要把大括号写上。

# for 循环语句

- for 语句为如下形式:  
`for(表达式1; 表达式2; 表达式3) { 语句; ... ; }`
- 执行过程  
首先计算表达式1, 接着执行表达式2, 若表达式2的值 = true, 则执行语句, 接着计算表达式3, 再判断表达式2的值; 依此重复下去, 直到表达式2的值 = false  
for语句中三个表达式都可以省略

```
public class Test {  
    public static void main(String args[]) {  
        long result = 0;  
        long f = 1;  
        for (int i = 1; i <= 10; i++) {  
            f = f * i;  
            result += f;  
        }  
        System.out.println("result=" + result);  
    }  
}
```

计算  $result = 1! + 2! + \dots + 10!$



## 课堂练习

编写程序, 用一个 for 循环计算  $1+3+5+7 + \dots + 99$  的值, 并输出计算结果。(OddSum.java)

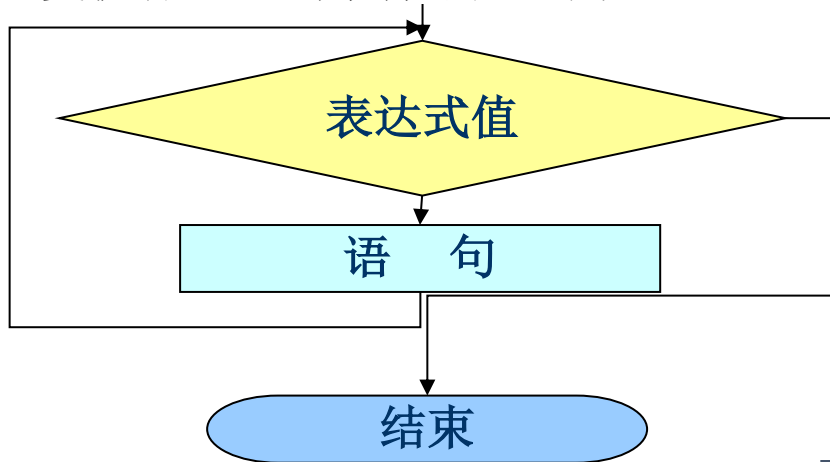
# while & do while 语句

- while 语句为如下形式:

```
while(逻辑表达式){ 语句; ... ;}
```

- 执行过程

先判断逻辑表达式的值。若=true,则执行其后面的语句,然后再次判断条件并反复执行,直到条件不成立为止



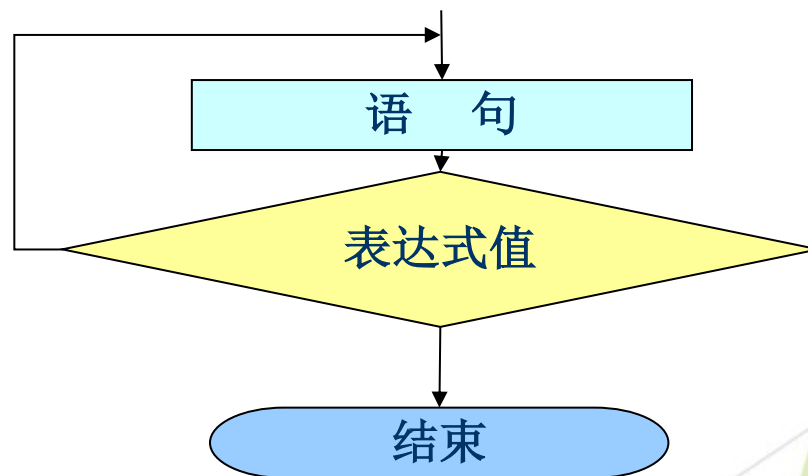
while 语句

- while 语句为如下形式:

```
do { 语句; ... ; } while(逻辑表达式);
```

- 执行过程

先执行语句,再判断逻辑表达式的值,若为true,再执行语句,否则结束循环。



do-while 语句

TestWhile.java

# break & Continue 语句

- **break** 语句用于终止某个语句块的执行。用在循环语句体中，可以强行退出循环；例如：

```
public class Test {  
    public static void main(String  
        args[]) {  
        int stop = 4;  
        for (int i = 1; i <= 10; i++) {  
            //当i等于stop时，退出循环  
            if (i == stop) break;  
            System.out.println(" i= " + i);  
        }  
    }  
}
```

```
i = 1  
i = 2  
i = 3
```

TestBreak.java

- **continue** 语句用在循环语句体中，用于终止某次循环过程，跳过循环体中 **continue** 语句下面未执行的循环，开始下一次循环过程；例如：

```
public class Test {  
    public static void main(String  
        args[]) {  
        int skip = 4;  
        for (int i = 1; i <= 5; i++) {  
            //当i等于skip时，跳过当次循环  
            if (i == skip) continue;  
            System.out.println("i = " + i);  
        }  
    }  
}
```

```
i = 1  
i = 2  
i = 3  
i = 5
```

# 循环语句举例

// 输出1~100内前5个可以被3整除的数。

```
public class Test {
    public static void main(String
        args[]) {
        int num = 0, i = 1;
        while (i <= 100) {
            if (i % 3 == 0) {
                System.out.print(i + " ");
                num++;
            }
            if (num == 5) {
                break;
            }
            i++;
        }
    }
}
```

//输出101~200内的质数,

```
public class Test {
    public static void main(String
        args[]) {
        for (int i=101; i<200; i+=2) {
            boolean f = true;
            for (int j = 2; j < i; j++) {
                if (i % j == 0) {
                    f = false;
                    break;
                }
            }
            if (!f) {continue;}
            System.out.print(" " + i);
        }
    }
}
```

# switch语句（条件语句补充）

- ```
switch() {  
    case xx :  
        ...  
    case xx :  
        ...  
    default:  
        ...  
}
```
- 小心case穿透，推荐使用break语句
- 多个case可以合并到一起
- default可以省略，但不推荐省略
- **Switch的值必须是int类型**

TestSwitch.java

# 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，声明格式：  
[修饰符1 修饰符2 ...] 返回值类型 方法名(形式参数列表) {

Java语句; ... ..

}

- 形式参数： 在方法被调用时用于接收外界输入的数据。
- 实参： 调用方法时实际传给方法的数据。
- 返回值： 方法在执行完毕后返还给调用它的环境的数据。
- 返回值类型： 事先约定的返回值的数据类型，如无返回值，必须给出返回值类型void。

TestMethod.java

- Java语言中使用下述形式调用方法：对象名.方法名(实参列表)
- 实参的数目、数据类型和次序必须和所调用方法声明的形参列表匹配，
- return 语句终止方法的运行并指定要返回的数据。

- Java中进行函数调用中传递参数时，遵循值传递的原则：  
基本类型传递的是该数据值本身。引用类型传递的是对对象的引用，而不是对象本身。



# 递归调用

//分析下面程序的运行结果。

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

... ..  
method(3)  
... ..

return 1\*2\*3=6

n=3

if(n == 1)  
return 1;  
else  
return  
n\*method(n-1)

n=3

n=2

if(n == 1)  
return 1;  
else  
return  
n\*method(n-1)

n=2

return 1\*2=2

n=1

if(n == 1)  
return 1;  
else  
return  
n\*method(n-1)

n=1

return 1

# 递归调用

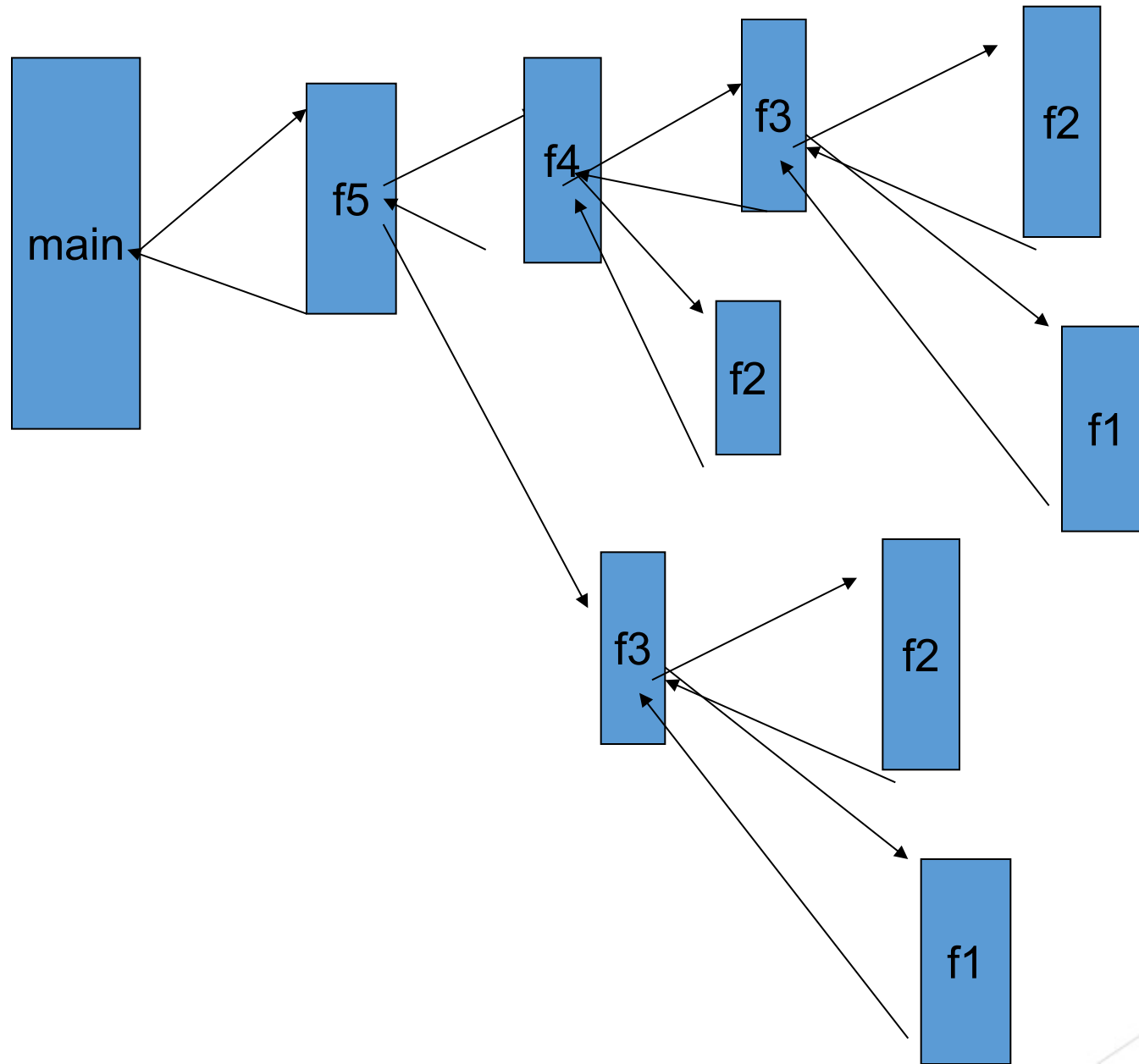
- 递归调用指在方法执行过程中出现该方法本身的调用例如：  
求Fibonacci数列: 1, 1, 2, 3, 5, 8, ...第40个数的值。 数列满足递推公式:

$$F_1 = 1, F_2 = 1 \quad F_n = F_{n-1} + F_{n-2} \quad (n > 2)$$

```
public class Test {  
    public static void main(String arg[]) {  
        System.out.println(f(5));  
    }  
  
    public static int f(int n) {  
        if (n == 1 || n == 2) {  
            return 1;  
        } else {  
            return f(n - 1) + f(n - 2);  
        }  
    }  
}
```

Fab.java

课堂练习  
试用非递归调用的方法解决上面  
Fibonacci 数列问题



```
public static int f(int n) {  
    if (n == 1 || n == 2) {  
        return 1;  
    } else {  
        return f(n - 1) + f(n - 2);  
    }  
}
```

# 第二章总结

- 标识符  
不用记规则，动手
- 关键字  
if else switch for while do while break continue void
- 局部变量&成员变量  
变量作用域  
内存布局
- 基础类型变量  
4类8种 互相转换
- 分支、循环 if switch for while do while
- 方法  
形参、实参、返回值、返回值类型  
递归调用（画图理解）