

1.3 Java 控制流程

- 分支语句
- 循环语句
- 方法

目录

1

分支语句

2

循环语句

3

方法

分支语句

- 目前为止，我们写的程序都是自上而下，按顺序执行的，没有代码会被跳过或重复。
- 当我们想让某段代码只在满足某种条件时执行，就需要使用**分支语句**[分岐文]。
- Java 提供以下几种实现分支的方法：
 - if-else 语句；
 - switch-case 语句；
 - 三元运算符以达到和分支语句等价的效果。

if 语句

- **if** 语句提供最基本的条件分支。

- 先看句法：

```
if (condition) {  
    codes;  
}
```

- **condition** 需要为布尔值。如果 **condition** 结果为 **true**，则运行花括号 “{}” 里面的程序。否则，此花括号会被跳过。
- 练习： 以下代码会输出什么？

```
1 int x = 20;  
2 int y = 18;  
3 if (x > y) {  
4     System.out.println("x is bigger: " + x);  
5 }  
6 System.out.println("y is bigger: " + y);
```

else 语句

- **else** 语句用于处理 if 不满足时的情况。

- 先看句法：

```
if (condition) {  
    codes 1;  
} else {  
    codes 2;  
}
```

- 如果 condition 的值为 true，则只运行第一个花括号里的程序。如果为 false，则只运行第二个花括号里的程序。
- 练习： 以下代码会输出什么？

```
1 int time = 10;  
2 if (time < 18) {  
3     System.out.println("Good day.");  
4 } else {  
5     System.out.println("Good evening.");  
6 }
```


if-else 语句的嵌套

- 如果 if 或 else 后的花括号里只有一行语句，则花括号可以被省略：

```
1 int time = 10;  
2 if (time < 18)  
3     System.out.println("Good day.");  
4 else  
5     System.out.println("Good evening.");
```

- 利用这点，我们可以简洁地连接多个分支语句判断有很多条件分支的情况。

```
1 int time = 10;  
2 if (time < 10) System.out.println("Good morning.");  
3 else if (time < 18) System.out.println("Good afternoon.");  
4 else System.out.println("Good evening.");
```

switch-case 语句

- **switch-case** 语句能够用来简化分支语法。

- 先看句法：

```
switch (expression) {  
    case value1:  
        codes1;  
        break;  
    case value2:  
        codes2;  
        break;  
    default:  
        break;  
}
```

Note !

expression 只能是整数（除了 long）或字符串类型！

- Java 将根据 expression 的值选择一个 case 块执行。
- **break** 关键字将跳出 switch 块。如果不添加，那么在被选中的块下面的 case 块也都会被执行。
- 如果没有 case 匹配，则执行 default 关键字指定的代码。

Try 

Switch.java

三元运算符

- **三元运算符**[条件运算符]被用来简洁的表达分支运算。

- 先看句法：

```
condition ? expression1 : expression2
```

- 和 if 语句不同，三元运算符直接计算出一个值并返回。我们可以直接把这个值用在其他命令或运算上：

```
1 int time = 10;  
2 System.out.println(time < 18 ? "Good day." : "Good evening.");
```

- 当我们使用分支语句的目的是为了根据不同的条件计算出某个值，就可以考虑使用三元运算符。

Q & A

Question and answer

目录

1

分支语句

2

循环语句

3

方法

循环语句

- 有时我们需要将同一段代码重复多次，这时我们就需要用到**循环语句**[繰り返し文]。
- Java 提供以下几种循环语句：
 - for;
 - for-each;
 - while;
 - do-while。

for 语句

- 先看句法：

```
for (code init; condition; code after) {  
    code body;  
}
```

- 其中：

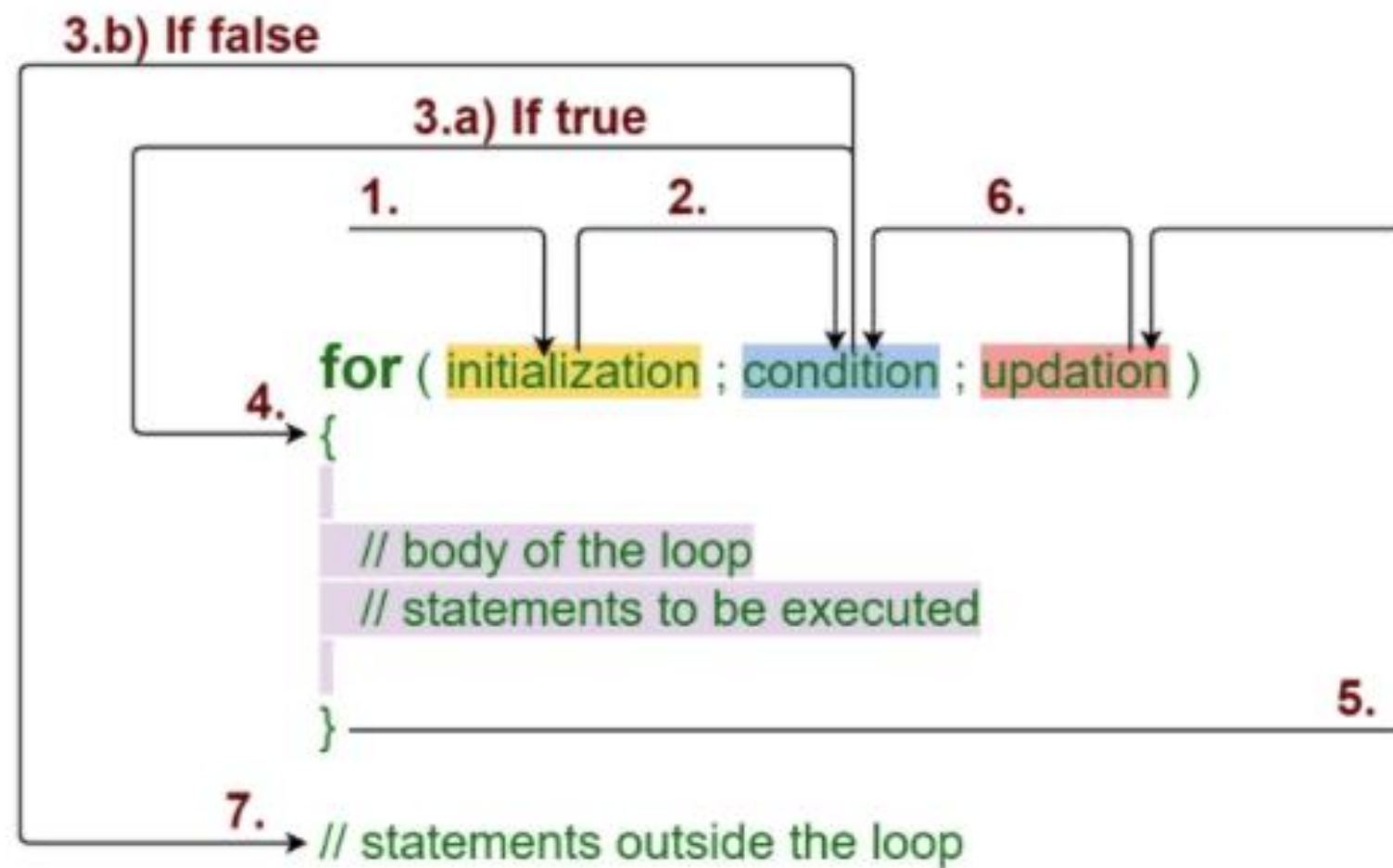
- code init 在执行代码块之前执行一次。
- condition 在前都会判断。如果为 false，则循环结束。
- 每次循环，在执行 code body 后执行 code after。

- for 语句一般用于执行特定次数的循环：

```
for (int i = 0; i < 10; i++) {  
    System.out.println(i);  
}
```

for 语句

- 流程图:



Try
For.java

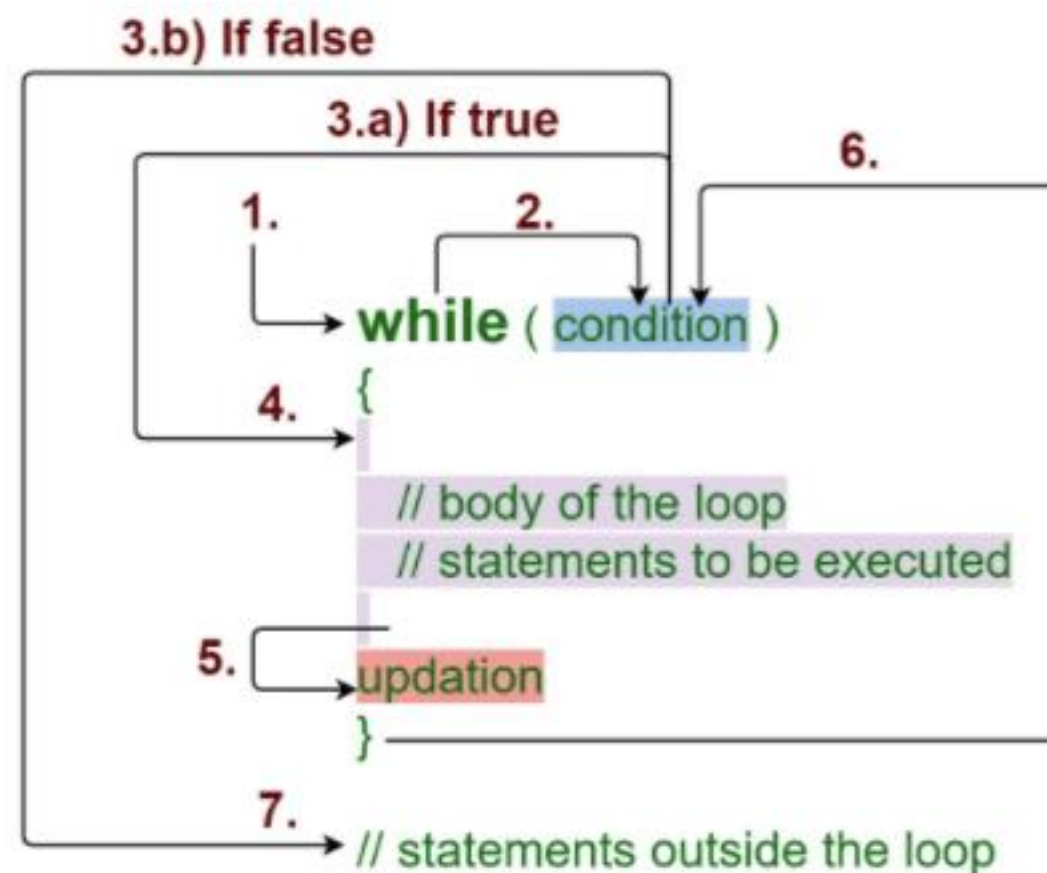
while 语句

- **while** 语句会在条件满足时反复执行代码块。

- 先看句法：

```
while (condition) {  
    // code  
}
```

- 流程图：



while 语句



- 思考： 以下代码会怎么样？

```
1 int i = 10;  
2 while (i > 0) {  
3     System.out.println(i);  
4     i++;  
5 }
```

- 如果条件始终满足， **while** 语句将永不停止！

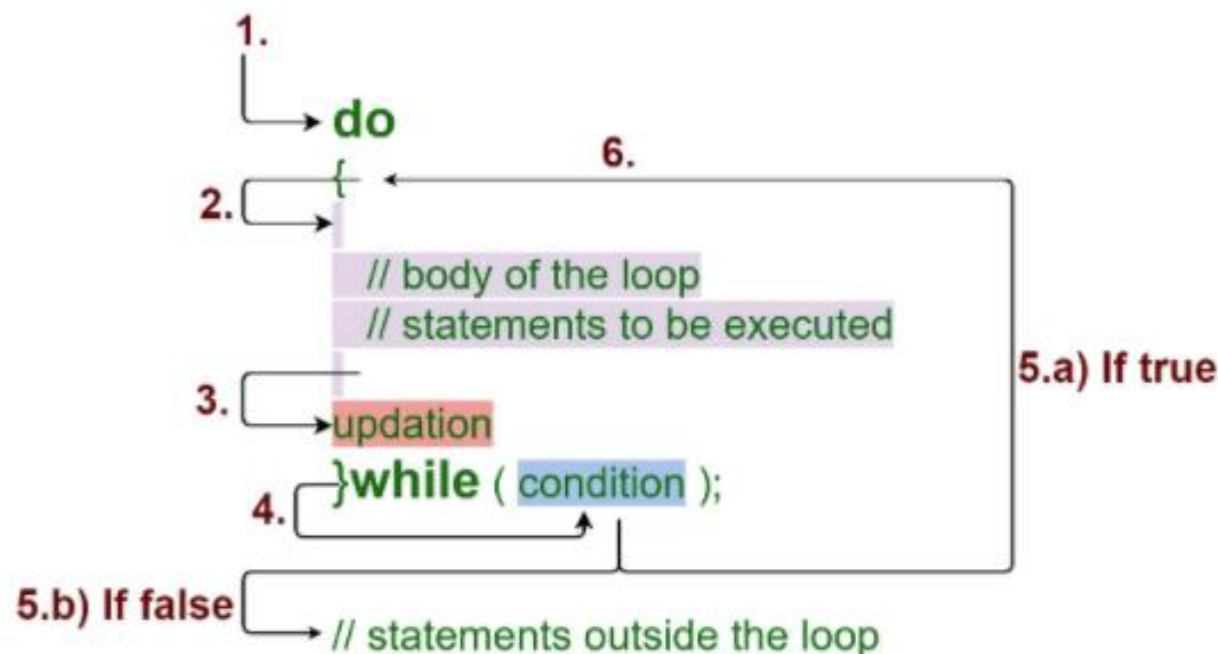
do-while 语句

- **do-while** 语句是 **while** 语句的一个变种，它一定会先执行一次循环体，无论条件是否满足。

- 先看句法：

```
do {  
    // code  
} while (condition);
```

- 流程图



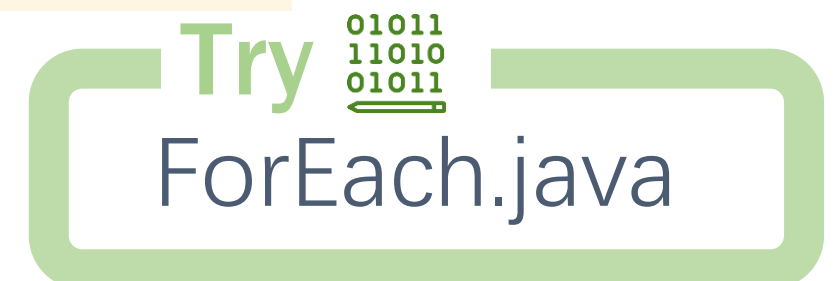
Try 
DoWhile.java

for-each 语句

- 先看句法：

```
for (type variableName : arrayName) {  
    // code block to be executed  
}
```
- **for-each** 语句专门用来遍历包含多个数据的列表或集合结构。比如：数组。

```
1 String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
2 for (String i : cars) {  
3     System.out.println(i);  
4 }
```



练习时间

- 分别使用 for、while、do-while 循环遍历以下数组：

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

- 思考：上述这些循环方式，分别适合在什么时候使用？你能举出具体的例子说明吗？

break 语句

- **break** 语句不仅可以用于跳出 switch 语句，还可用于跳出任何循环：

```
1 for (int i = 0; i < 10; i++) {  
2     System.out.println(i); // => 0 1 2 3 4  
3     if (i == 4) {  
4         break;  
5     }  
6 }
```

continue 语句

- **continue** 语句将跳过本次循环。

```
1 for (int i = 0; i < 5; i++) {  
2     if (i == 2) {  
3         continue;  
4     }  
5     System.out.println(i); // => 0 1 3 4  
6 }
```

Note



continue 语句将结束**本次**循环并开始条件判断，
而 break 语句将立刻结束剩余**全部**循环。

练习

- 以下两段代码将分别输出什么内容？

```
1 int i = 0
2 while (i < 5) {
3     System.out.println(i);
4     i++;
5     if (i == 2) {
6         break;
7     }
8 }
```

```
1 int i = 0
2 while (i < 5) {
3     if (i == 2) {
4         continue;
5     }
6     i++;
7     System.out.println(i);
8 }
```

Q & A

Question and answer

Coffee ☕ Break

嵌套循环和标签

当我们访问较复杂的数据结构，如多维数组时，可能需要在每次循环时都执行一个别的循环。比如访问一个 2 维数组，第一个循环（外层循环）负责遍历所有数组，而第二个循环（内层循环）负责遍历这些数组里的元素。这被称为循环的**嵌套**。

在嵌套循环中，`break` 和 `continue` 语句都只会考虑离它最近的那一层循环（内层循环）。如果要跳出外层循环，可以使用**标签**[ラベル]（`Label`）。

Try 01011
11010
01011
Label.java

目录

1

控制语句

2

循环语句

3

方法

方法

- 想一想：在之前的编程过程中，你有没有出现反复编写同一段或是类似代码的情况？
- 如果一个程序中经常反复使用到类似的功能，我们自然可以靠复制粘贴代码来节省开发时间。但这对代码提高可读性并无帮助，反而容易使代码冗长、杂乱。
- 我们可以使用**方法**[メソッド]来应对这个问题。方法定义了一段代码块，我们可以通过**调用**它来重复利用这段代码，以实现相同的功能。
- 比起直接复制粘贴，方法更为灵活：通过传入不同的**参数**，我们可以对所实现功能的具体细节进行调整。

接次页 

[↩ 接前页](#)

Example ✓

编写一个程序，计算 $2^3 + 5^4 + 8^5$ 的值。
我们当然可以写成这样：

```
1 public static void main(String[] args) {  
2     int a = 2 * 2 * 2 + 3 * 3 * 3 * 3 + 8 * 8 * 8 * 8 * 8;  
3     System.out.println(a); // => 32857  
4 }
```

但是，这个程序的 *可读性* 高吗？
这个程序的 *可扩展性* 又如何？

[接次页 ➞](#)

[↩ 接前页](#)

Example ✓

如果我们有一个可以计算“a 的 b 次方”的方法 `power(a, b)`，我们就可以使用下面的代码来计算同样的结果：

```
1 public static void main(String[] args) {  
2     int a = power(2, 3) + power(3, 4) + power(5, 8);  
3     System.out.println(a);  
4 }
```


这个程序的 *可读性*和 *可扩展性*又如何？

方法的定义

- Java 已经给我们提供了很多方便的方法，我们之前也已用到一些。但实际开发过程中，**定义**自己的方法仍是有必要的。

- 定义方法的基本句法：

```
1 static int power(int a, int b) {  
2     codes;  
3 }
```

- 其中，`power`是这个方法的**名称**；`int`是这个方法**返回值**的类型；`int a` 和 `int b` 是方法的**参数**列表；当方法被调用，`statement` 处的代码便会被执行。`static` 关键词我们会在之后（ § 2.1.3）讲解。
- 方法的命名规范和变量一致。

方法的参数列表

- 每个方法可以有 1 个、0 个（无参）或多个**参数**^[引数]，在创建方法时，需要标明方法所有参数的**类型**和**名称**。多个参数用逗号隔开：

Example ✓

接受 0 个参数：

```
static int answerToEverything()
```

接受 1 个参数：

```
static int sqrt(int a)
```

接受 2 个参数：

```
static int power(int a, int b)
```


方法的返回值

- 方法的**返回值**[戻り値]用来将方法里计算出的结果返回到调用它的地方。方法的返回值需要在定义时注明清楚。
- 在方法中，使用 `return` 语句将数据作为返回值返回（**这同时将立刻终止方法！**）：

```
1 static int answerToEverything() {  
2     return 42;  
3 }
```

- 如果方法不需要返回值，可以将返回值定义为 **void**：

```
1 static void sayHello() {  
2     System.out.println("Hello!");  
3 }
```

Tips

返回值为 `void` 的方法也可使用 `return` 语句，用于立刻终止方法。

方法的调用

- 在我们定义好方法后，就随时可以在任何地方**调用**[呼び出す]它，执行我们定义的代码。要调用一个方法，在其名称后写上括号“**()**”并将每个参数的值按顺序输入，以**逗号**隔开：

```
1 public static void main() {  
2     sayHello();           // 调用了一个无参方法  
3     sqrt();               // 调用了一个有 1 个参数的方法  
4     System.out.println(power(2, 3)); // 调用了一个多参数方法  
5 }
```

- 可以注意到，我们可以直接将方法的返回值用于其他命令或运算。



练习时间

- 写一个用来判断一个整数奇偶性的方法。
- 方法名：checkParity。
- 参数： 类型： int； 名称： num。
- 返回值： 类型： String：
 - 如果是奇数则返回 “odd”， 偶数则返回 “even”。
- 在 main 方法中调用它， 检验是否符合要求。

Q & A

Question and answer

方法的重载

- Java 支持创建具有同样名称的多个方法，只要保证方法的参数列表的**类型**不同。这被称为方法的**重载**[オーバーロード]：

```
1 static int power(int a, int b);  
2 static float power(float a, float b);  
3 static int power(int a);
```

- 除了可以统一功能类似，但参数类型不同的方法外，重载还能用于实现“默认参数”：

```
1 static int power(int a) {  
2     return power(a, 2);  
3 }
```

Try 

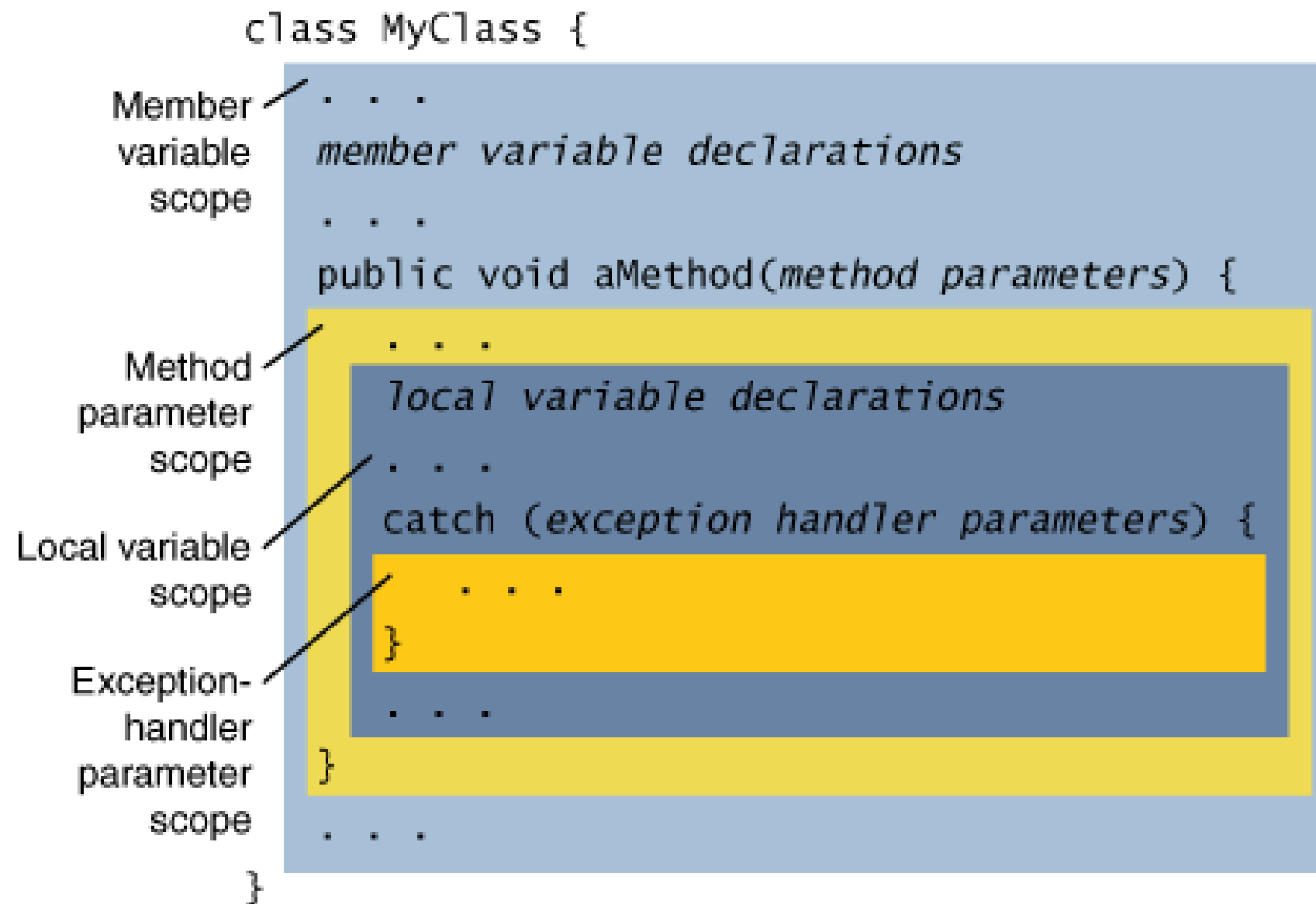
Overload.java

Note

重载的方法参数类型必须不同。只有返回类型不同的方法无法用于重载。

作用域

- 在 Java 中，变量只能在创建它们的区域内被访问。这称为变量的**作用域**[スコープ]。



局部变量

- 在方法内部声明的变量，其作用域便是方法的内部。因此它们无法在其他方法中被访问到。它们被称为**局部变量**[ローカル変数]：

```
1 static int answerToEverything() {  
2     // 这里不能用 answer  
3  
4     int answer = 42;  
5  
6     // 这里可以使用  
7     return answer;  
8 }  
9  
10 public static void main() {  
11     // 这里不能用 answer  
12     // answer = 0; // => 报错  
13 }
```

Tips

方法的参数也是方法的局部变量。

代码块

- **代码块**[ブロック]是指一组花括号“{}”之间的代码。
- 代码块里声明的变量，作用域便是该代码块。
- 作为特例，for（for-each）语句的小括号“（）”中声明的变量，也属于该 for 语句的代码块。
- 你也可以不使用控制命令而直接使用代码块。



方法的递归

- **递归**[再帰]是指在一个方法里面调用它自己。
- 如果方法需要解决的问题可以分解成几个规模更小的同类型问题（*子问题*），而利用这些子问题的解我们可以解决原本的问题，那就轮到递归出场了。

Example ✓

编写一个方法，计算斐波那契数列 $f(n)$ 的值。其中：

$$f(0) = 0;$$

$$f(1) = 1;$$

$$f(n) = f(n - 1) + f(n - 2), \text{ if } n \geq 2.$$

接次页 ➞

 接前页

Example ✓

编写一个方法，计算斐波那契数列 $f(n)$ 的值。其中：

$$f(0) = 0;$$

$$f(1) = 1;$$

$$f(n) = f(n - 1) + f(n - 2), \text{ if } n \geq 2.$$

- 在编写该方法 `int f(int n)` 时，我们可以直接使用 $f(n - 1)$ 和 $f(n - 2)$ 来计算这两个子问题的解，再通过对它们求和来解决原问题。

递归的停止条件

- 当然，我们不可能无限地让方法递归调用自己。因此，在编写递归方法的时候，一定要确保方法存在某些**停止条件**（Halting Condition）：当问题的规模足够小的时候，不使用递归调用而是直接求出答案：

```
1 static int f(int n) {  
2     if (n == 0) {  
3         return 0;  
4     } else if (n == 1) {  
5         return 1;  
6     } else {  
7         return f(n - 1) + f(n - 2);  
8     }  
9 }
```

Try 
Recursion.java

Q & A

Question and answer

总结

Sum Up

1. 分支语句: **if-else**、switch-case、?:。
2. 循环语句:
 - ① **for**、**while**、do-while、for-each。
 - ② break 和 continue。
3. 方法:
 - ① 创建和使用。
 - ② 重载。
 - ③ 作用域。
 - ④ 递归。

THANK YOU!