

Refactoring to Bind It All Together



Zoran Horvat
CEO AT CODING HELMET

@zoranh75 <http://codinghelmet.com>



Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

Demo.java

```
1: Project
2
3     public class Demo {
4         @ private int getControlDigit(int number) {
5             int sum = 0;
6             boolean isOddPos = true;
7
8                 while (number > 0) {
9                     int digit = (int) (number%10);
10
11                     if (isOddPos) {
12                         sum += 3 * digit;
13                     }
14                     else {
15                         sum += digit;
16                     }
17
18                     number /= 10;
19                     isOddPos = !isOddPos;
20                 }
21
22                 int modulo = sum%11;
23                 if (modulo > 9)
24                     modulo = 0;
25
26                 return modulo;
27             }
28
29         public void run() {
```

2: Favorites

2: I: Structure

2: Favorites

Document number
(typed in by a man)

123456-1

Control digit
(added by software)

Event Log

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

Demo.java

```
1: Project
2
3     public class Demo {
4     @ private int getControlDigit(int number) {
5         int sum = 0;
6         boolean isOddPos = true;
7
8             while (number > 0) {
9                 int digit = (int) (number%10);
10
11                 if (isOddPos) {
12                     sum += 3 * digit;
13                 }
14                 else {
15                     sum += digit;
16                 }
17
18                 number /= 10;
19                 isOddPos = !isOddPos;
20             }
21
22             int modulo = sum%11;
23             if (modulo > 9)
24                 modulo = 0;
25
26             return modulo;
27         }
28
29     public void run() {
```

2: Favorites

2: I: Structure

2: Favorites

Ant Build

Maven

123456

Event Log

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

Demo.java

```
1: Project
2
3     public class Demo {
4     @ private int getControlDigit(int number) {
5         int sum = 0;
6         boolean isOddPos = true;
7
8         while (number > 0) {
9             int digit = (int) (number%10);
10
11            if (isOddPos) {
12                sum += 3 * digit;
13            }
14            else {
15                sum += digit;
16            }
17
18            number /= 10;
19            isOddPos = !isOddPos;
20        }
21
22        int modulo = sum%11;
23        if (modulo > 9)
24            modulo = 0;
25
26        return modulo;
27    }
28
29    public void run() {
```

2: Favorites

2: I: Structure

2: Favorites

Ant Build

Maven

1 2 3 4 5 6

Event Log

TODO Terminal Messages

The screenshot shows a Java code editor interface. The main window displays the `Demo.java` file. A specific line of code, `int digit = (int) (number%10);`, is highlighted with a green rectangular selection. To the right of the code, there is a visual representation of the digit extraction process: six blue-outlined boxes containing the digits 1, 2, 3, 4, 5, and 6, arranged horizontally. A green bracket on the right side of the boxes spans from the fifth box to the sixth, indicating the range of digits being processed by the highlighted line of code. The code itself is a method named `getControlDigit` that calculates the control digit for a given number using a weighted sum of its digits.

1: Project

2: Favorites

3: I: Structure

4: Demo.java

```
2
3     public class Demo {
4     @ private int getControlDigit(int number) {
5         int sum = 0;
6         boolean isOddPos = true;
7
8         while (number > 0) {
9             int digit = (int) (number%10);
10
11            if (isOddPos) {
12                sum += 3 * digit;
13            }
14            else {
15                sum += digit;
16            }
17
18            number /= 10;
19            isOddPos = !isOddPos;
20        }
21
22        int modulo = sum%11;
23        if (modulo > 9)
24            modulo = 0;
25
26        return modulo;
27    }
28
29    public void run() {
```

The diagram illustrates the calculation of the control digit for the number 123456. The digits are arranged in a row: 1, 2, 3, 4, 5, 6. The 1st, 3rd, and 5th digits (1, 3, 5) are multiplied by 3, while the 2nd, 4th, and 6th digits (2, 4, 6) are added directly. The result is then taken modulo 11.

1: Project

2: Favorites

3: I: Structure

4: Favorites

```
public class Demo {  
    private int getControlDigit(int number) {  
        int sum = 0;  
        boolean isOddPos = true;  
  
        while (number > 0) {  
            int digit = (int) (number%10);  
  
            if (isOddPos) {  
                sum += 3 * digit;  
            }  
            else {  
                sum += digit;  
            }  
  
            number /= 10;  
            isOddPos = !isOddPos;  
        }  
  
        int modulo = sum%11;  
        if (modulo > 9)  
            modulo = 0;  
  
        return modulo;  
    }  
  
    public void run() {  
    }  
}
```

Ant Build

m Maven

The diagram illustrates the step-by-step calculation of the control digit for the number 123456. It shows the digits grouped by position:

- 1, 2, 3, 4, 5, 6

The digits at odd positions (1, 3, 5) are multiplied by 3:

- 1 → 3
- 3 → 12
- 5 → 15

The digits at even positions (2, 4, 6) are added directly:

- 2 → 2
- 4 → 4
- 6 → 6

The results are then summed:

- 3 + 12 + 15 + 2 + 4 + 6 = 46

The final result is 4, which is the control digit for the number 123456.

1: Project

2: Favorites

3: I: Structure

4: Favorites

```
public class Demo {  
    private int getControlDigit(int number) {  
        int sum = 0;  
        boolean isOddPos = true;  
  
        while (number > 0) {  
            int digit = (int) (number%10);  
  
            if (isOddPos) {  
                sum += 3 * digit;  
            }  
            else {  
                sum += digit;  
            }  
  
            number /= 10;  
            isOddPos = !isOddPos;  
        }  
  
        int modulo = sum%11;  
        if (modulo > 9)  
            modulo = 0;  
  
        return modulo;  
    }  
  
    public void run() {  
        Demo  
    }  
}
```

Ant Build

m Maven

The diagram illustrates the step-by-step calculation of the control digit for the number 123456. It shows the digits grouped into pairs: (1, 2), (3, 4), (5, 6). The first pair (1, 2) is multiplied by 3, resulting in 12. This is then added to the second pair (3, 4), resulting in 12 + 34 = 46. Finally, the third pair (5, 6) is added, resulting in 46 + 56 = 102. The remainder of 102 divided by 11 is 1, which is the control digit.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

Demo.java

```
1: Project
2
3     public class Demo {
4     @ private int getControlDigit(int number) {
5         int sum = 0;
6         boolean isOddPos = true;
7
8             while (number > 0) {
9                 int digit = (int) (number%10);
10
11                 if (isOddPos) {
12                     sum += 3 * digit;
13                 }
14                 else {
15                     sum += digit;
16                 }
17
18                 number /= 10;
19                 isOddPos = !isOddPos;
20             }
21
22             int modulo = sum%11;
23             if (modulo > 9)
24                 modulo = 0;
25
26             return modulo;
27         }
28
29     public void run() {
```

2: Favorites

1: Structure

Ant Build

Maven

6

5

4

3

2

1

Event Log

```
1: Project Demo.java x  
2  
3     public class Demo {  
4         @ private int getControlDigit(int number) {  
5             int sum = 0;  
6             boolean isOddPos = true;  
7  
8             while (number > 0) {  
9                 int digit = (int) (number%10);  
10  
11                 if (isOddPos) {  
12                     sum += 3 * digit; // This line is highlighted with a green box.  
13                 }  
14                 else {  
15                     sum += digit;  
16                 }  
17  
18                 number /= 10;  
19                 isOddPos = !isOddPos;  
20             }  
21  
22             int modulo = sum%11;  
23             if (modulo > 9)  
24                 modulo = 0;  
25  
26             return modulo;  
27         }  
28  
29     public void run() {  
  
 1: I-Structure  
 2: Favorites Demo
```

The diagram illustrates the calculation of a control digit for the number 654321. The digits are processed from right to left. The digit 6 is multiplied by 3, resulting in 18, which is then added to the sum of the other digits (5+4+3+2+1=18).

*3 → 6
5
4
3
2
1

```
1: Project Demo.java x  
2  
3     public class Demo {  
4     @ private int getControlDigit(int number) {  
5         int sum = 0;  
6         boolean isOddPos = true;  
7  
8             while (number > 0) {  
9                 int digit = (int) (number%10);  
10  
11                 if (isOddPos) {  
12                     sum += 3 * digit;  
13                 }  
14                 else {  
15                     sum += digit;  
16                 }  
17  
18                 number /= 10;  
19                 isOddPos = !isOddPos;  
20             }  
21  
22             int modulo = sum%11;  
23             if (modulo > 9)  
24                 modulo = 0;  
25  
26             return modulo;  
27     }  
28  
29     public void run() {  
  
1: Structure I  
2: Favorites I  
3: Project Demo
```



Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

Demo.java

```
1: Project
2
3     public class Demo {
4         @ private int getControlDigit(int number) {
5             int sum = 0;
6             boolean isOddPos = true;
7
8                 while (number > 0) {
9                     int digit = (int) (number%10);
10
11                     if (isOddPos) {
12                         sum += 3 * digit;
13                     }
14                     else {
15                         sum += digit;
16                     }
17
18                     number /= 10;
19                     isOddPos = !isOddPos;
20                 }
21
22                 int modulo = sum%11;
23                 if (modulo > 9)
24                     modulo = 0;
25
26                 return modulo;
27             }
28
29         public void run() {
```

2: Favorites

2: I-Structure

2: Favorites

Ant Build

Maven

*3 6

*3 5

*3 4

3

2

1

Demo

Event Log

The code calculates a control digit for a given number. It iterates through each digit of the number, starting from the least significant digit (1). If the digit is at an odd position, it is multiplied by 3. If it is at an even position, it is multiplied by 1. The results are then summed to get the final modulo 11 value.

```
1: Project Demo.java x  
2  
3     public class Demo {  
4     @ private int getControlDigit(int number) {  
5         int sum = 0;  
6         boolean isOddPos = true;  
7  
8         while (number > 0) {  
9             int digit = (int) (number%10);  
10  
11             if (isOddPos) {  
12                 sum += 3 * digit;  
13             }  
14             else {  
15                 sum += digit;  
16             }  
17  
18             number /= 10;  
19             isOddPos = !isOddPos;  
20         }  
21  
22         int modulo = sum%11;  
23         if (modulo > 9)  
24             modulo = 0;  
25  
26         return modulo;  
27     }  
28  
29     public void run() {  
  
 1: I-Structure  
 2: Favorites Demo
```



Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

Demo.java

```
1: Project
2
3     public class Demo {
4         @ private int getControlDigit(int number) {
5             int sum = 0;
6             boolean isOddPos = true;
7
8                 while (number > 0) {
9                     int digit = (int) (number%10);
10
11                     if (isOddPos) {
12                         sum += 3 * digit;
13                     }
14                     else {
15                         sum += digit;
16                     }
17
18                     number /= 10;
19                     isOddPos = !isOddPos;
20                 }
21
22                 int modulo = sum%11;
23                 if (modulo > 9)
24                     modulo = 0;
25
26                 return modulo;
27             }
28
29         public void run() {
```

2: Favorites

1: Structure

2: Favorites

Ant Build

Maven

Event Log

The diagram shows the calculation of a control digit for the number 654321. The digits are processed from right to left. An if-block multiplies odd-position digits by 3 (6, 4, 2) and adds them to the sum. Even-position digits (5, 3, 1) are added directly to the sum. The final sum is 37, which is then taken modulo 11 to yield the control digit 6.

Digit Position	Digit Value	Calculation	Sum
1 (Even)	1	$1 \times 1 = 1$	1
2 (Odd)	2	$2 \times 3 = 6$	6
3 (Even)	4	$4 \times 1 = 4$	4
4 (Odd)	6	$6 \times 3 = 18$	18
5 (Even)	5	$5 \times 1 = 5$	5
6 (Odd)	3	$3 \times 3 = 9$	9
7 (Even)	2	$2 \times 1 = 2$	2
8 (Odd)	1	$1 \times 3 = 3$	3
Total Sum		37	37
Modulo 11		37 % 11 = 6	6

1: Project

2: Favorites

3: I: Structure

4: Demo.java

```
public class Demo {  
    private int getControlDigit(int number) {  
        int sum = 0;  
        boolean isOddPos = true;  
  
        while (number > 0) {  
            int digit = (int) (number%10);  
  
            if (isOddPos) {  
                sum += 3 * digit;  
            }  
            else {  
                sum += digit;  
            }  
  
            number /= 10;  
            isOddPos = !isOddPos;  
        }  
  
        int modulo = sum%11;  
        if (modulo > 9)  
            modulo = 0;  
  
        return modulo;  
    }  
  
    public void run() {  
    }  
}
```

5: Moreoojava

6: Demo

7: *3 6

8: *3 5

9: *3 4

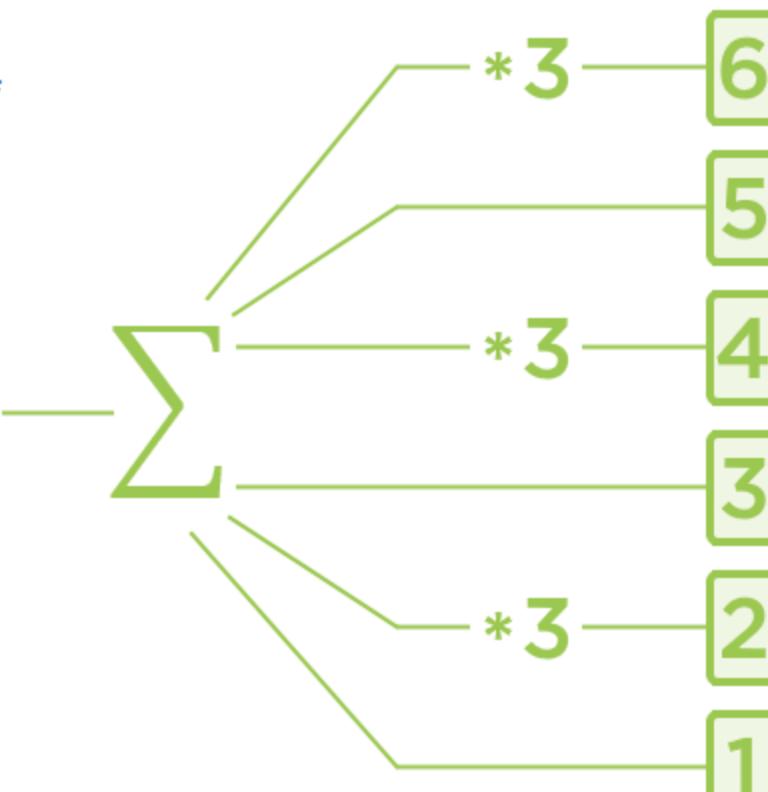
10: *3 3

11: *3 2

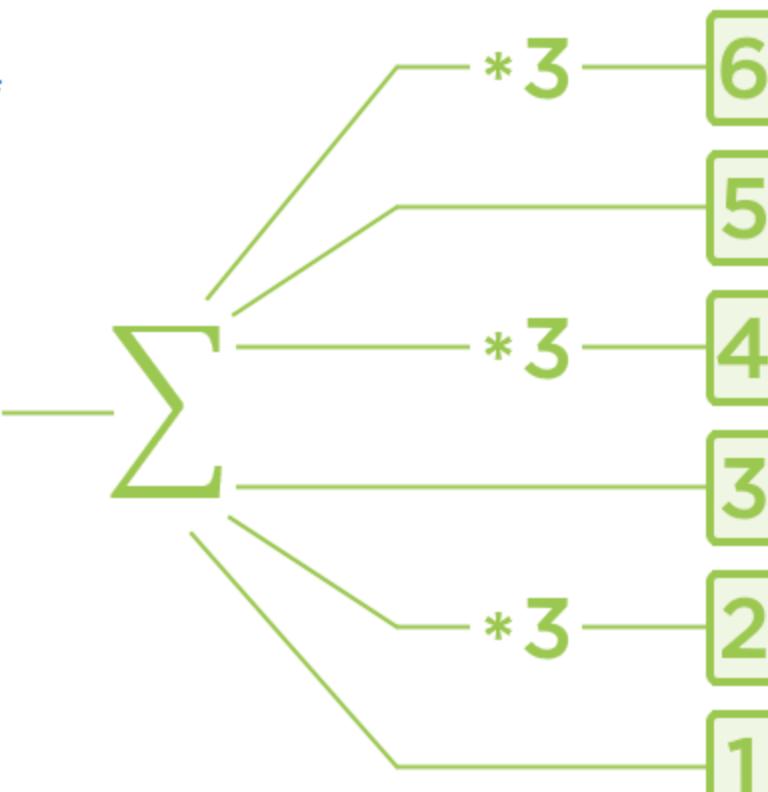
12: *3 1



```
1: Project  
2  
3     public class Demo {  
4         @ private int getControlDigit(int number) {  
5             int sum = 0;  
6             boolean isOddPos = true;  
7  
8                 while (number > 0) {  
9                     int digit = (int) (number%10);  
10  
11                     if (isOddPos) {  
12                         sum += 3 * digit;  
13                     }  
14                     else {  
15                         sum += digit;  
16                     }  
17  
18                     number /= 10;  
19                     isOddPos = !isOddPos;  
20                 }  
21  
22                 int modulo = sum%11;  
23                 if (modulo > 9)  
24                     modulo = 0;  
25  
26                 return modulo;  
27             }  
28  
29         public void run() {  
  
Demo
```

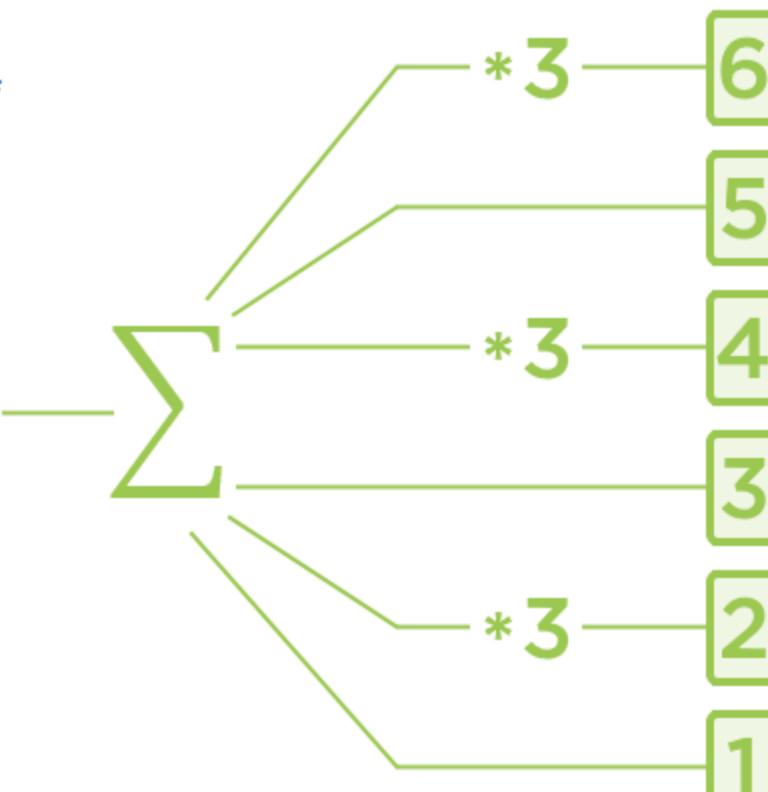


```
1: Project  
2  
3     public class Demo {  
4         @private int getControlDigit(int number) {  
5             int sum = 0;  
6             boolean isOddPos = true;  
7  
8             while (number > 0) {  
9                 int digit = (int) (number%10);  
10  
11                 if (isOddPos) {  
12                     sum += 3 * digit;  
13                 }  
14                 else {  
15                     sum += digit;  
16                 }  
17  
18                 number /= 10;  
19                 isOddPos = !isOddPos;  
20             }  
21  
22             int modulo = sum%11;  
23             if (modulo > 9)  
24                 modulo = 0;  
25  
26             return modulo;  
27         }  
28  
29     public void run() {  
  
Demo
```



The diagram illustrates the calculation of a control digit. It shows a tree structure where digits are multiplied by 3 and summed. The digits are 6, 5, 4, 3, 2, and 1, each multiplied by 3 and then summed to produce the final result.

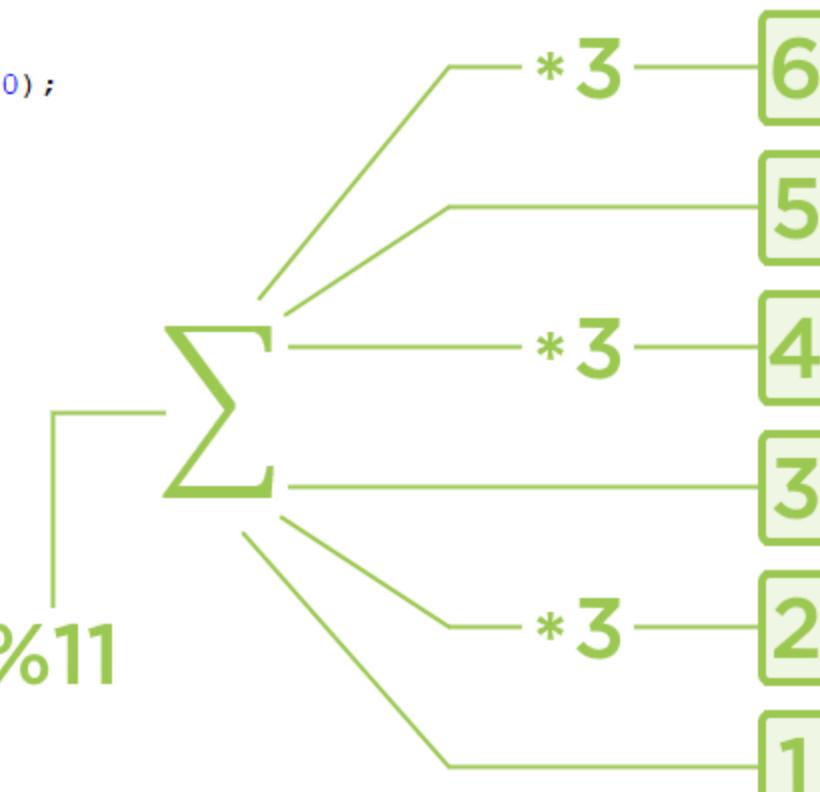
```
1: Project  
2  
3     public class Demo {  
4         @ private int getControlDigit(int number) {  
5             int sum = 0;  
6             boolean isOddPos = true;  
7  
8                 while (number > 0) {  
9                     int digit = (int) (number%10);  
10  
11                     if (isOddPos) {  
12                         sum += 3 * digit;  
13                     }  
14                     else {  
15                         sum += digit;  
16                     }  
17  
18                     number /= 10;  
19                     isOddPos = !isOddPos;  
20                 }  
21  
22                 int modulo = sum%11;  
23                 if (modulo > 9)  
24                     modulo = 0;  
25  
26                 return modulo;  
27             }  
28  
29         public void run() {  
  
Demo
```



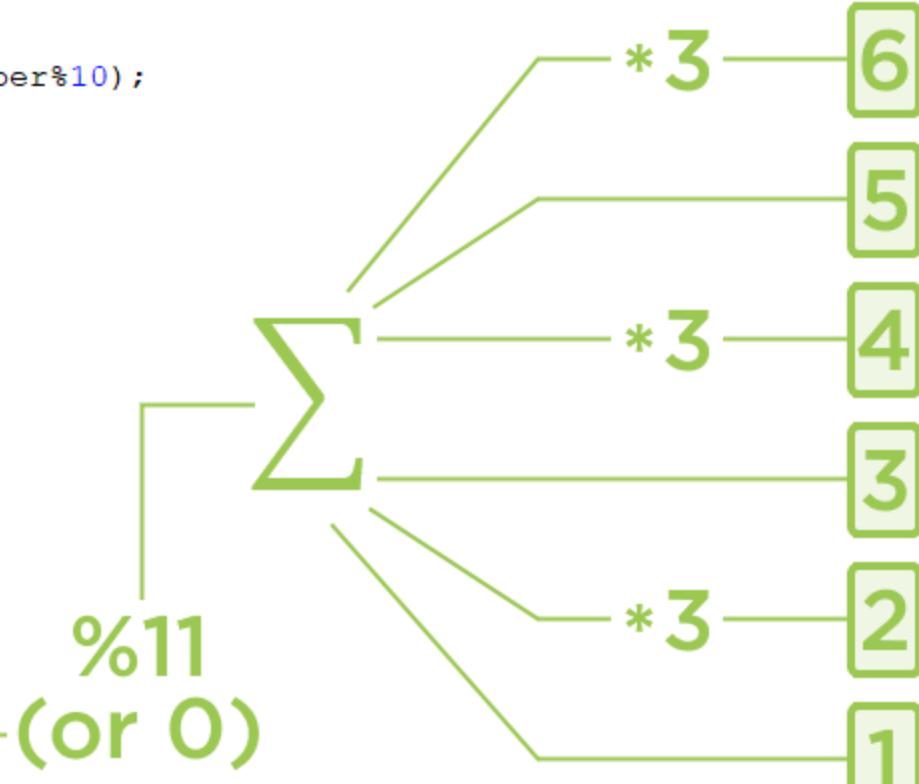
The diagram illustrates the calculation of a control digit. It shows a tree structure where digits are multiplied by 3 and summed. The digits 6, 5, 4, 3, 2, and 1 are shown in green boxes, each with a multiplier of 3 next to it.

```
1: Project  
2  
3     public class Demo {  
4         @ private int getControlDigit(int number) {  
5             int sum = 0;  
6             boolean isOddPos = true;  
7  
8                 while (number > 0) {  
9                     int digit = (int) (number%10);  
10  
11                     if (isOddPos) {  
12                         sum += 3 * digit;  
13                     }  
14                     else {  
15                         sum += digit;  
16                     }  
17  
18                     number /= 10;  
19                     isOddPos = !isOddPos;  
20                 }  
21  
22                 int modulo = sum%11;  
23                 if (modulo > 9)  
24                     modulo = 0;  
25  
26                 return modulo;  
27             }  
28  
29     public void run() {  
  
Demo
```

The diagram illustrates the calculation of a control digit. It shows a series of digits (6, 5, 4, 3, 2, 1) being multiplied by 3 and summed to get a result. The digits are arranged vertically, and green lines connect them to the multiplication factor '3' and then to a summation symbol. The final result is shown as a green box.

```
1: Project  
2  
3     public class Demo {  
4         @ private int getControlDigit(int number) {  
5             int sum = 0;  
6             boolean isOddPos = true;  
7  
8             while (number > 0) {  
9                 int digit = (int) (number%10);  
10  
11                 if (isOddPos) {  
12                     sum += 3 * digit;  
13                 }  
14                 else {  
15                     sum += digit;  
16                 }  
17  
18                 number /= 10;  
19                 isOddPos = !isOddPos;  
20             }  
21  
22             int modulo = sum%11;  
23             if (modulo > 9)  
24                 modulo = 0;  
25  
26             return modulo;  
27         }  
28  
29     public void run() {  
  
    Demo  
  
      
    *3 6  
    *3 5  
    *3 4  
    *3 3  
    *3 2  
    *3 1
```

```
1: Project  
2  
3     public class Demo {  
4         @private int getControlDigit(int number) {  
5             int sum = 0;  
6             boolean isOddPos = true;  
7  
8             while (number > 0) {  
9                 int digit = (int) (number%10);  
10  
11                 if (isOddPos) {  
12                     sum += 3 * digit;  
13                 }  
14                 else {  
15                     sum += digit;  
16                 }  
17  
18                 number /= 10;  
19                 isOddPos = !isOddPos;  
20             }  
21  
22             int modulo = sum%11;  
23             if (modulo > 9)  
24                 modulo = 0;  
25  
26             return modulo;  
27         }  
28  
29     public void run() {  
  
    Demo  
  
    1: Project  
    2: I-Structure  
    3: Favorites  
    4: Favorites  
    5: Favorites  
    6: Favorites  
    7: Favorites  
    8: Favorites  
    9: Favorites  
   10: Favorites  
   11: Favorites  
   12: Favorites  
   13: Favorites  
   14: Favorites  
   15: Favorites  
   16: Favorites  
   17: Favorites  
   18: Favorites  
   19: Favorites  
   20: Favorites  
   21: Favorites  
   22: Favorites  
   23: Favorites  
   24: Favorites  
   25: Favorites  
   26: Favorites  
   27: Favorites  
   28: Favorites  
   29: Favorites  
  
    Ant Build  
    Maven
```



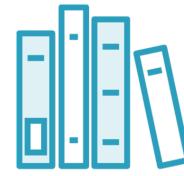
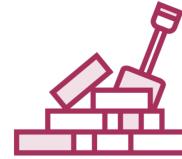
```
1: Project  
2  
3     public class Demo {  
4         @ private int getControlDigit(int number) {  
5             int sum = 0;  
6             boolean isOddPos = true;  
7  
8                 while (number > 0) {  
9                     int digit = (int) (number%10);  
10  
11                     if (isOddPos) {  
12                         sum += 3 * digit;  
13                     }  
14                     else {  
15                         sum += digit;  
16                     }  
17  
18                     number /= 10;  
19                     isOddPos = !isOddPos;  
20                 }  
21  
22                 int modulo = sum%11;  
23                 if (modulo > 9)  
24                     modulo = 0;  
25  
26                 return modulo;  
27             }  
28  
29         public void run() {  
  
    Diagram illustrating the calculation of the control digit:  
  
    The code calculates the sum of digits at odd positions (multiplied by 3) and even positions.  
  
    The result is then taken modulo 11 (or 0).  
  
    The diagram shows the mapping of digits to their values after being multiplied by 3:  
  
    * 6 → 18 (6 * 3)  
    * 5 → 15 (5 * 3)  
    * 4 → 12 (4 * 3)  
    * 3 → 9 (3 * 3)  
    * 2 → 6 (2 * 3)  
    * 1 → 3 (1 * 3)  
  
    These values are then summed up to get the final result.
```

Assessing Code Quality

```
private int getControlDigit(int number) {  
    int sum = 0;  
    boolean isOddPos = true;  
  
    while (number > 0) {  
        int digit = (int) (number%10);  
  
        if (isOddPos) { sum += 3 * digit; }  
        else { sum += digit; }  
  
        number /= 10;  
        isOddPos = !isOddPos;  
    }  
  
    int modulo = sum%11;  
    if (modulo > 9)  
        modulo = 0;  
  
    return modulo;  
}
```



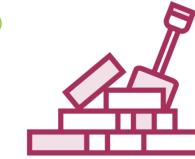
Nice?



Short?



Flexible?



Maintainable?

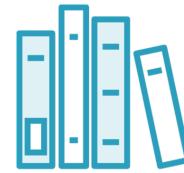


Assessing Code Quality

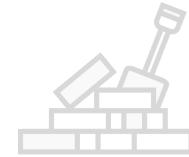
```
private int getControlDigit(int number) {  
    int sum = 0;  
    boolean isOddPos = true;  
  
    while (number > 0) {  
        int digit = (int) (number%10);  
  
        if (isOddPos) { sum += 3 * digit; }  
        else { sum += digit; }  
  
        number /= 10;  
        isOddPos = !isOddPos;  
    }  
  
    int modulo = sum%11;  
    if (modulo > 9)  
        modulo = 0;  
  
    return modulo;  
}
```



Nice?



Short?



Maintainable?



Flexible?

Complex requirements lead to longer implementation



Assessing Code Quality

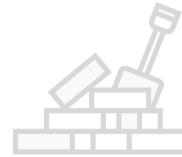
```
private int getControlDigit(int number) {  
    int sum = 0;  
    boolean isOddPos = true;  
  
    while (number > 0) {  
        int digit = (int) (number%10);  
  
        if (isOddPos) { sum += 3 * digit; }  
        else { sum += digit; }  
  
        number /= 10;  
        isOddPos = !isOddPos;  
    }  
  
    int modulo = sum%11;  
    if (modulo > 9)  
        modulo = 0;  
  
    return modulo;  
}
```



Nice?



Short?



Maintainable?



Flexible?

Requirements:

Multiply every other digit by three
starting with the first digit on the right
sum the results up
take modulo eleven of the sum
or substitute zero

Implementation:

Loop while not zero, take modulo 10, etc.



Assessing Code Quality

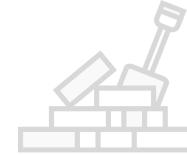
```
private int getControlDigit(int number) {  
    int sum = 0;  
    boolean isOddPos = true;  
  
    while (number > 0) {  
        int digit = (int) (number%10);  
  
        if (isOddPos) { sum += 3 * digit; }  
        else { sum += digit; }  
  
        number /= 10;  
        isOddPos = !isOddPos;  
    }  
  
    int modulo = sum%11;  
    if (modulo > 9)  
        modulo = 0;  
  
    return modulo;  
}
```



Nice?



Short?



Maintainable?



Flexible?



Cognitive mapping

Procedural code doesn't match the requirements in a spoken language

What shall we do when requirements change?

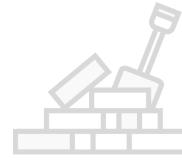


Assessing Code Quality

```
private int getControlDigit(int number) {  
    int sum = 0;  
    boolean isOddPos = true;  
  
    while (number > 0) {  
        int digit = (int) (number%10);  
  
        if (isOddPos) { sum += 3 * digit; }  
        else { sum += digit; }  
  
        number /= 10;  
        isOddPos = !isOddPos;  
    }  
  
    int modulo = sum%11;  
    if (modulo > 9)  
        modulo = 0;  
  
    return modulo;  
}
```



Nice?



Short?



Flexible?



Cognitive mapping

Requirements follow logic of a man

Implementation follows logic of a *programming language*



Assessing Code Quality

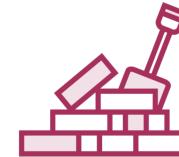
```
private int getControlDigit(int number) {  
    int sum = 0;  
    boolean isOddPos = true;  
  
    while (number > 0) {  
        int digit = (int) (number%10);  
  
        if (isOddPos) { sum += 3 * digit; }  
        else { sum += digit; }  
  
        number /= 10;  
        isOddPos = !isOddPos;  
    }  
  
    int modulo = sum%11;  
    if (modulo > 9)  
        modulo = 0;  
  
    return modulo;  
}
```



Nice?



Short?



Maintainable?



Flexible?



Cognitive mapping

Added requirement:
Read numbers left-to-right

Implementation:
No substitute for the % operator!



Assessing Code Quality

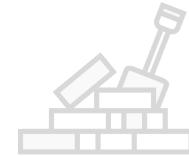
```
private int getControlDigit(int number) {  
    int sum = 0;  
    boolean isOddPos = true;  
  
    while (number > 0) {  
        int digit = (int) (number%10);  
        if (isOddPos) { sum += 3 * digit; }  
        else { sum += digit; }  
  
        number /= 10;  
        isOddPos = !isOddPos;  
    }  
  
    int modulo = sum%11;  
    if (modulo > 9)  
        modulo = 0;  
  
    return modulo;  
}
```



Nice?



Short?



Maintainable?



Flexible?



Cognitive
mapping

Right-to-left or left-to-right
Multiplier depends on date

One modulo or the other

With or without substitute

...

...

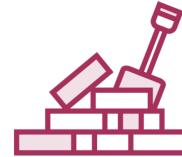


Assessing Code Quality

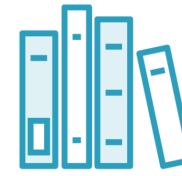
```
private int getControlDigit(int number) {  
    int sum = 0;  
    boolean isOddPos = true;  
  
    while (number > 0) {  
        int digit = (int) (number%10);  
  
        if (isOddPos) { sum += 3 * digit; }  
        else { sum += digit; }  
  
        number /= 10;  
        isOddPos = !isOddPos;  
    }  
  
    int modulo = sum%11;  
    if (modulo > 9)  
        modulo = 0;  
  
    return modulo;  
}
```



Nice?



Maintainable?



Short?



Flexible?



Cognitive mapping



Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

Demo.java

1: Project

2

3 public class Demo {

4 @ private int getControlDigit(int number) {

5 int sum = 0;

6 boolean isOddPos = true;

7

8 while (number > 0) {

9 int digit = (int) (number%10);

10

11 if (isOddPos) {

12 sum += 3 * digit;

13 }

14 else {

15 sum += digit;

16 }

17

18 number /= 10;

19 isOddPos = !isOddPos;

20 }

21

22 int modulo = sum%11;

23 if (modulo > 9)

24 modulo = 0;

25

26 return modulo;

27 }

28

29 public void run() {

Demo

2: Favorites

3: I: Structure

4: Favorites

Ant Build

Maven

Infrastructural

Substantial

The diagram illustrates the static analysis of the 'getControlDigit' method. It uses color-coded boxes to identify different components of the code:

- Red boxes:** Surround the method parameters ('number'), the loop condition ('number > 0'), the loop variable ('digit'), the assignment to 'sum' inside the loop, the assignment to 'isOddPos' inside the loop, and the assignment to 'modulo'.
- Green boxes:** Surround the multiplication ('3 * digit'), the addition to 'sum', the assignment to 'digit' inside the loop, the division assignment ('number /= 10'), the assignment to 'isOddPos', the modulo operation ('sum%11'), the conditional check ('modulo > 9'), and the assignment to 'modulo'.
- Large green box:** Surrounds the entire modulo calculation ('int modulo = sum%11;').
- Red box:** Surrounds the assignment to 'modulo' ('modulo = 0;').

A vertical line connects the 'Infrastructural' and 'Substantial' labels to the code, indicating the scope of each category.

Pluralsight > src > com > codinghelmet > moreoojava > DigitsStream

Main

Demo.java StraightNumber.java DigitsStream.java

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5: 1: Project

6

7 public class DigitsStream implements ForwardingStream<Integer> {

8 private Stream<Integer> stream;

9

10 @Override

11 public Stream<Integer> getStream() { return this.stream; }

12

13 private DigitsStream(Stream<Integer> stream) {

14 this.stream = stream;

15 }

16

17 @ ...

18 public static DigitsStream of(Stream<Integer> stream) {

19 return new DigitsStream(stream);

20 }

21

22 public int multiplyWith(int... factors) {

23 return this.getStream().zip(this.repeatEndlessly(factors))

24 }

25

26 private Stream<Integer> repeatEndlessly(int... factors) {

27 return Stream

28 .iterate(factors, UnaryOperator.identity())

29 .flatMapToInt(ToIntFunction.toIntFunction(

30)}

31 }

32

33

DigitsStream > multiplyWith()

6 TODO 0 Messages 4 Run Event Log

zip(p, q, f)

Ant Build

Maven

Pluralsight > src > com > codinghelmet > moreoojava > DigitsStream

Main

Demo.java StraightNumber.java DigitsStream.java

Project 1: Project

Ant Build m Maven

```
6
7     public class DigitsStream implements ForwardingStream<Integer> {
8         private Stream<Integer> stream;
9
10        @Override
11        public Stream<Integer> getStream() { return this.stream; }
12
13        private DigitsStream(Stream<Integer> stream) {
14            this.stream = stream;
15        }
16
17        @
18        public static DigitsStream of(Stream<Integer> stream) {
19            return new DigitsStream(stream);
20        }
21
22        public int multiplyWith(int... factors) {
23            return this.getStream().zip(this.repeatEndlessly(factors));
24        }
25
26        private Stream<Integer> repeatEndlessly(int... factors) {
27            return Stream
28                .iterate(factors, UnaryOperator.identity())
29                .flatMapToInt(ToIntFunction::stream)
30                .boxed();
31        }
32
33    }
```

DigitsStream > multiplyWith()

zip(p, q, f)

Diagram illustrating the zip operation. Two vertical stacks of boxes represent streams p and q . A third vertical stack of boxes represents the result of the zip operation. An arrow labeled f points from the bottom of stream p to the top of the result stack, indicating that elements from both streams are paired together by the function f .

Project 2: Favorites

Event Log

Pluralsight > src > com > codinghelmet > moreoojava > DigitsStream

Main

Demo.java StraightNumber.java DigitsStream.java

Project 1: Project

Ant Build m Maven

```
6
7     public class DigitsStream implements ForwardingStream<Integer> {
8         private Stream<Integer> stream;
9
10        @Override
11        public Stream<Integer> getStream() { return this.stream; }
12
13        private DigitsStream(Stream<Integer> stream) {
14            this.stream = stream;
15        }
16
17        @
18        public static DigitsStream of(Stream<Integer> stream) {
19            return new DigitsStream(stream);
20        }
21
22        public int multiplyWith(int... factors) {
23            return this.getStream().zip(this.repeatEndlessly(factors));
24        }
25
26        private Stream<Integer> repeatEndlessly(int... factors) {
27            return Stream
28                .iterate(factors, UnaryOperator.identity())
29                .flatMapToInt(ToIntFunction::stream)
30                .boxed();
31        }
32
33    }
```

DigitsStream > multiplyWith()

zip(p, q, f)

The diagram illustrates the `zip` operation. It shows two input streams, `p` and `q`, represented by vertical columns of blue rectangles. These streams are combined via a function `f` into a single output stream, represented by a vertical column of green rectangles.

Pluralsight > src > com > codinghelmet > moreoojava > DigitsStream

Main

Demo.java StraightNumber.java DigitsStream.java

Project

1: Project

2: Favorites

3: Structure

4: Favorites

5: I: Structure

6

```
public class DigitsStream implements ForwardingStream<Integer> {
    private Stream<Integer> stream;

    @Override
    public Stream<Integer> getStream() { return this.stream; }

    private DigitsStream(Stream<Integer> stream) {
        this.stream = stream;
    }

    @
    public static DigitsStream of(Stream<Integer> stream) {
        return new DigitsStream(stream);
    }

    public int multiplyWith(int... factors) {
        return this.getStream().zip(this.repeatEndlessly(factors));
    }

    private Stream<Integer> repeatEndlessly(int... factors) {
        return Stream
            .iterate(factors, UnaryOperator.identity())
            .flatMapToInt(ToIntFunction.toIntFunction(Objects::stream))
            .boxed();
    }
}
```

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

DigitsStream > multiplyWith()

zip(p, q, f)

Ant Build

Maven

I

Event Log

Pluralsight > src > com > codinghelmet > moreoojava > DigitsStream

Main

Demo.java StraightNumber.java DigitsStream.java

Project 1: Project

Ant Build m Maven

```
6
7     public class DigitsStream implements ForwardingStream<Integer> {
8         private Stream<Integer> stream;
9
10        @Override
11        public Stream<Integer> getStream() { return this.stream; }
12
13        private DigitsStream(Stream<Integer> stream) {
14            this.stream = stream;
15        }
16
17        @
18        public static DigitsStream of(Stream<Integer> stream) {
19            return new DigitsStream(stream);
20        }
21
22        public int multiplyWith(int... factors) {
23            return this.getStream().zip(this.repeatEndlessly(factors));
24        }
25
26        private Stream<Integer> repeatEndlessly(int... factors) {
27            return Stream
28                .iterate(factors, UnaryOperator.identity())
29                .flatMapToInt(ToIntFunction::stream)
30                .boxed();
31        }
32
33    }
```

DigitsStream > multiplyWith()

zip(p, q, f)

The diagram illustrates the `zip` operation. It shows two input streams, `p` and `q`, represented by vertical columns of blue rectangles. These streams are combined via a function `f` into a single output stream, represented by a vertical column of green rectangles.

Pluralsight > src > com > codinghelmet > moreoojava > DigitsStream

Main

Demo.java StraightNumber.java DigitsStream.java

Project 1: Project

Ant Build m Maven

```
6
7     public class DigitsStream implements ForwardingStream<Integer> {
8         private Stream<Integer> stream;
9
10        @Override
11        public Stream<Integer> getStream() { return this.stream; }
12
13        private DigitsStream(Stream<Integer> stream) {
14            this.stream = stream;
15        }
16
17        @
18        public static DigitsStream of(Stream<Integer> stream) {
19            return new DigitsStream(stream);
20        }
21
22        public int multiplyWith(int... factors) {
23            return this.getStream().zip(this.repeatEndlessly(factors));
24        }
25
26        private Stream<Integer> repeatEndlessly(int... factors) {
27            return Stream
28                .iterate(factors, UnaryOperator.identity())
29                .flatMapToInt(ToIntFunction::stream)
30                .boxed();
31        }
32
33    }
```

DigitsStream > multiplyWith()

zip(p, q, f)

Diagram illustrating the zip operation. It shows two streams, p and q, represented by vertical columns of blue and green rectangles respectively. Arrows from both streams point to a third column, f, represented by a vertical column of green rectangles. This indicates that each element from stream p is paired with the corresponding element from stream q, and the result is collected into stream f.

TODO Terminal Messages Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > DigitsStream

Main

Demo.java StraightNumber.java DigitsStream.java

Project 1: Project

Ant Build m Maven

```
6
7     public class DigitsStream implements ForwardingStream<Integer> {
8         private Stream<Integer> stream;
9
10        @Override
11        public Stream<Integer> getStream() { return this.stream; }
12
13        private DigitsStream(Stream<Integer> stream) {
14            this.stream = stream;
15        }
16
17        @
18        public static DigitsStream of(Stream<Integer> stream) {
19            return new DigitsStream(stream);
20        }
21
22        public int multiplyWith(int... factors) {
23            return this.getStream().zip(this.repeatEndlessly(factors));
24        }
25
26        private Stream<Integer> repeatEndlessly(int... factors) {
27            return Stream
28                .iterate(factors, UnaryOperator.identity())
29                .flatMapToInt(ToIntFunction::stream)
30                .boxed();
31        }
32
33    }
```

DigitsStream > multiplyWith()

zip(p, q, f)

Pluralsight > src > com > codinghelmet > moreoojava > ControlDigit

1: Project Demo.java ControlDigit.java ControlDigitAlgorithm.java PonderingModuloAlgorithm.java StraightNumber.java DocumentNumber.java DigitsStream.java

2

```
3     public interface ControlDigit {  
4         @ static ControlDigitAlgorithm accountingAlgorithm() {  
5             return PonderingModuloAlgorithm.multipleDigitsModulo(  
6                 StraightNumber::digitsFromLeastSignificant, divisor: 11, substitute: 0, ...factors: 3, 1);  
7         }  
8  
9         @ static ControlDigitAlgorithm salesAlgorithmLegacy() {  
10             return PonderingModuloAlgorithm.singleDigitModulo(  
11                 StraightNumber::digitsFromMostSignificant, divisor: 7, ...factors: 7, 3, 2);  
12         }  
13     }  
14  
15     Hey mate!  
16  
17     We call it May 2017  
18  
19     Yup! We definitely  
20     call it May 2017  
21  
22     Hi guys!  
23  
24     What do you call  
25     that new algorithm?  
26  
27     What?  
28     No, that's a date  
29
```

2: Favorites I: Structure

ControlDigit > salesAlgorithmLegacy()

6: TODO 0: Messages 4: Run Event Log

The image shows a Java code editor with a sidebar featuring two cartoon characters. The left character, wearing glasses and a green shirt, says "Hey mate!", "We call it May 2017", and "Yup! We definitely call it May 2017". The right character, also wearing glasses and a blue shirt, responds with "Hi guys!", "What do you call that new algorithm?", "What?", and "No, that's a date". The code editor displays Java code related to control digit algorithms.

Summary



Implementing an object-oriented algorithm

- Separate infrastructural operations
- Focus on domain-related operations
- Wrap loops into objects (e.g. streams)
- Wrap transformations into meaningful methods



Summary



Exercising the algorithm's flexibility

- Easy to adapt to changing requirements
- Parameterize data and behavior
- Inject behavior as dependencies
- Give names to prominent sets of parameters
- Produce a flexible and maintainable algorithm implementation



Course Summary



Removing loops

- Turn their subjects into streams
- Stream encapsulates the loop
- Implement domain-specific streams
- Expose domain-specific methods
- Implement composite objects
- Composite behaves like a single object



Course Summary



Design a Domain-specific Language (DSL)

- Let complex transform read like a sentence
- Develop maintainable domain code



Course Summary



Removing multiway branching

- Organize rule objects into a graph
- Encapsulate selection and execution of domain logic
- Use rules to manage volatile business rules and flows



Course Summary



Turning algorithms into objects

- Make them maintainable
in the long run



Course Summary



Enforcing the object-oriented method

- Improve coding skills
- Use design patterns
- Apply refactoring
- Incorporate functional design,
because...
- THE BEST OBJECT-ORIENTED CODE
IS FUNCTIONAL CODE

