Tactical Design Patterns in .NET: Control Flow

UNDERSTANDING CONTROL FLOW

Zoran Horvat
CTO at InterVenture GmbH

@zoranh75 www.codinghelmet.com

About This Course

Previous course:

- Tactical Design Patterns in .NET: Managing Responsibilities

In this course:

- Dealing with control flow in code

When control flow is done right…

- Code is simpler and shorter
- Methods are shorter
- Coordination between classes is easier to follow
- Classes are simpler and shorter when flow decisions are made differently

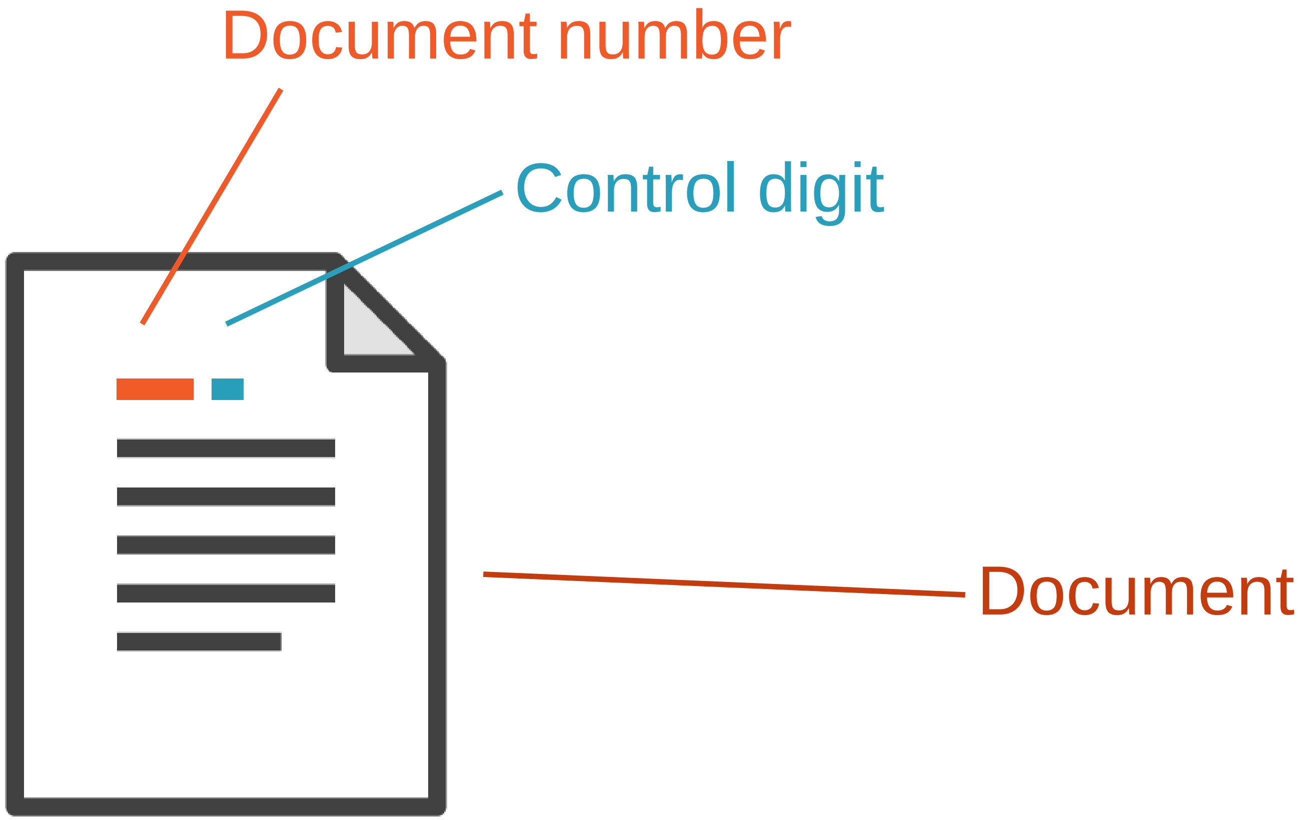
Understanding Control Flow

More complex the code, harder to manage control flow

A few examples before we start with design patterns

- Demonstrate basic techniques of simplifying the control flow
- Prepare the for advanced techniques that incorporate design patterns

Example



Document number

Control digit

Document

82712476 X

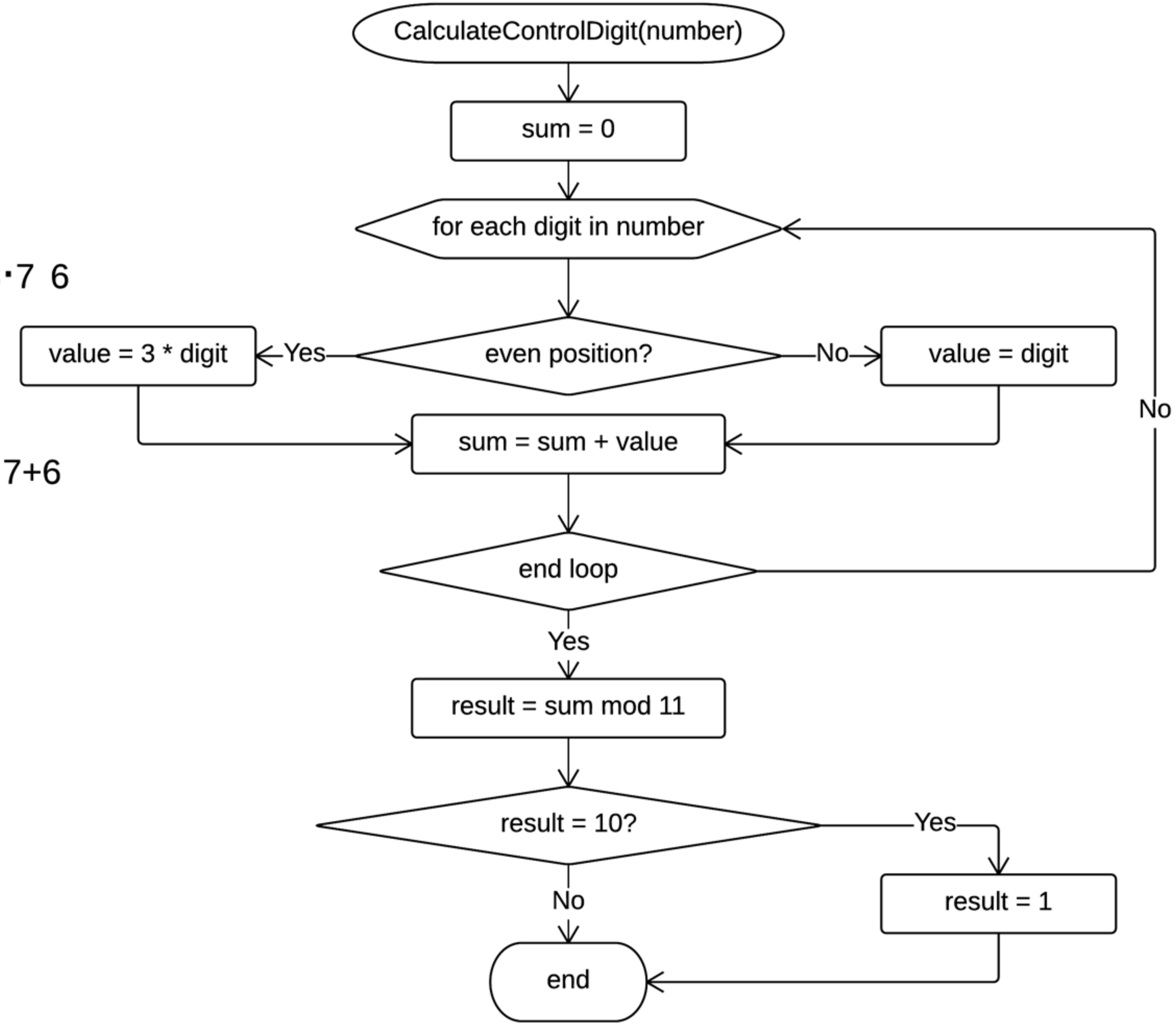
8 2 7 1 2 4 7 6

$3 \cdot 8 \quad 2 \quad 3 \cdot 7 \quad 1 \quad 3 \cdot 2 \quad 4 \quad 3 \cdot 7 \quad 6$

$3 \cdot 8+2+3 \cdot 7+1+3 \cdot 2+4+3 \cdot 7+6$

85 mod 11

8



CalculateControlDigit(number)

sum = 0

for each digit in number

even position?

Yes

value = 3 * digit

No

value = digit

sum = sum + value

end loop

No

Yes

result = sum mod 11

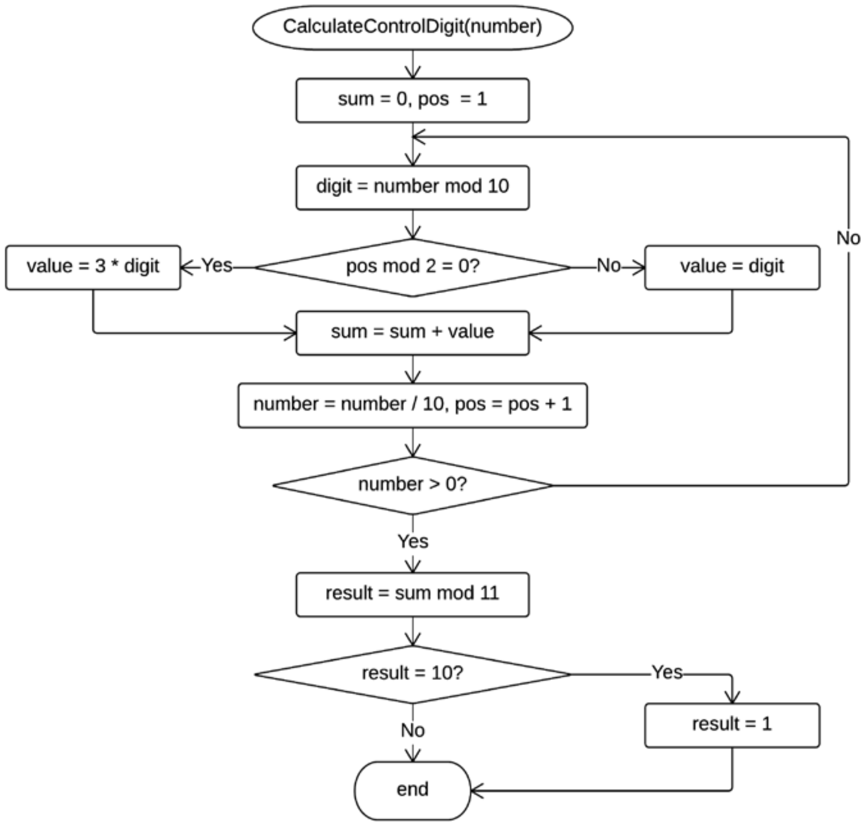
result = 10?

Yes

result = 1

No

end



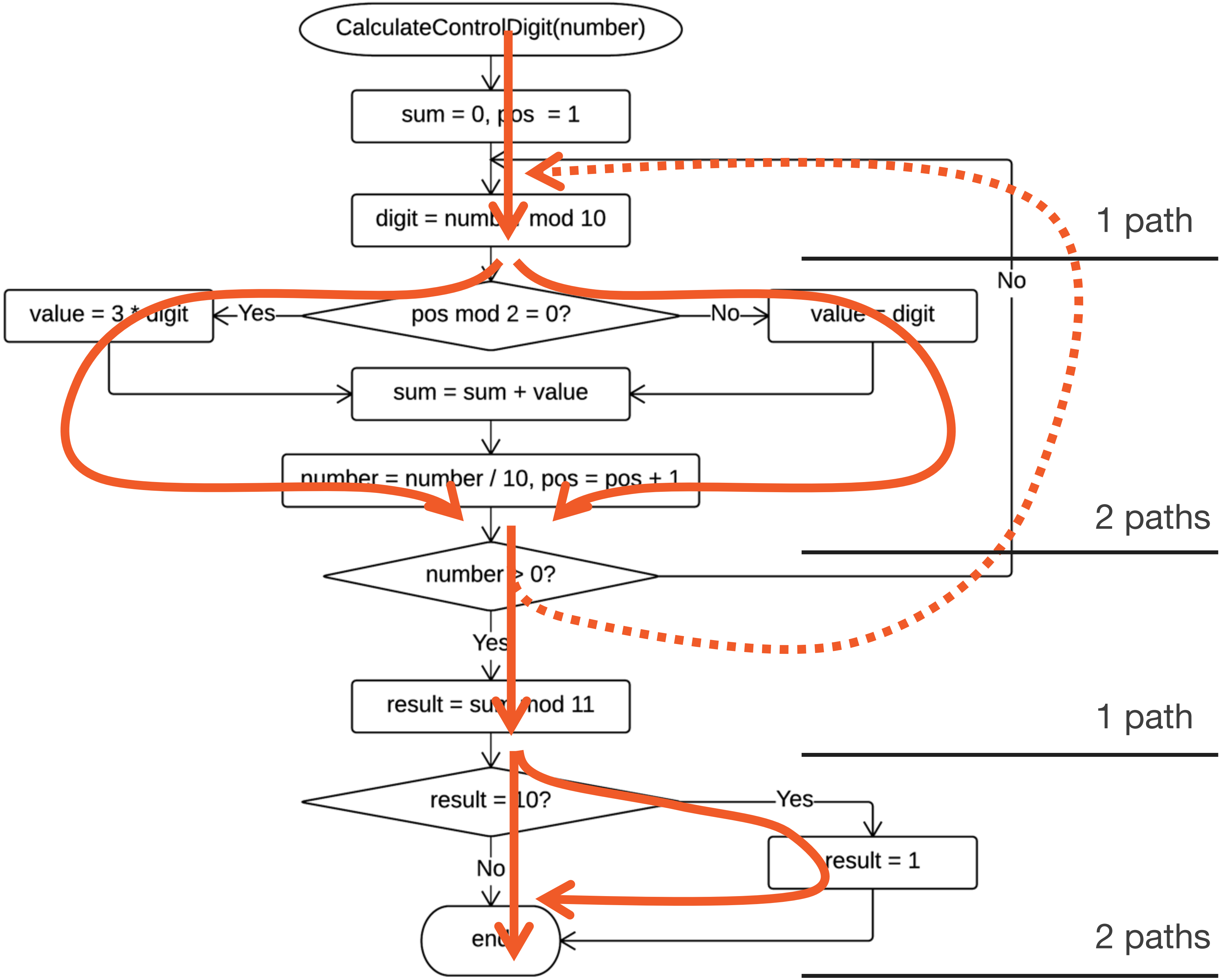
Complexity…

- Number of mutually independent execution paths

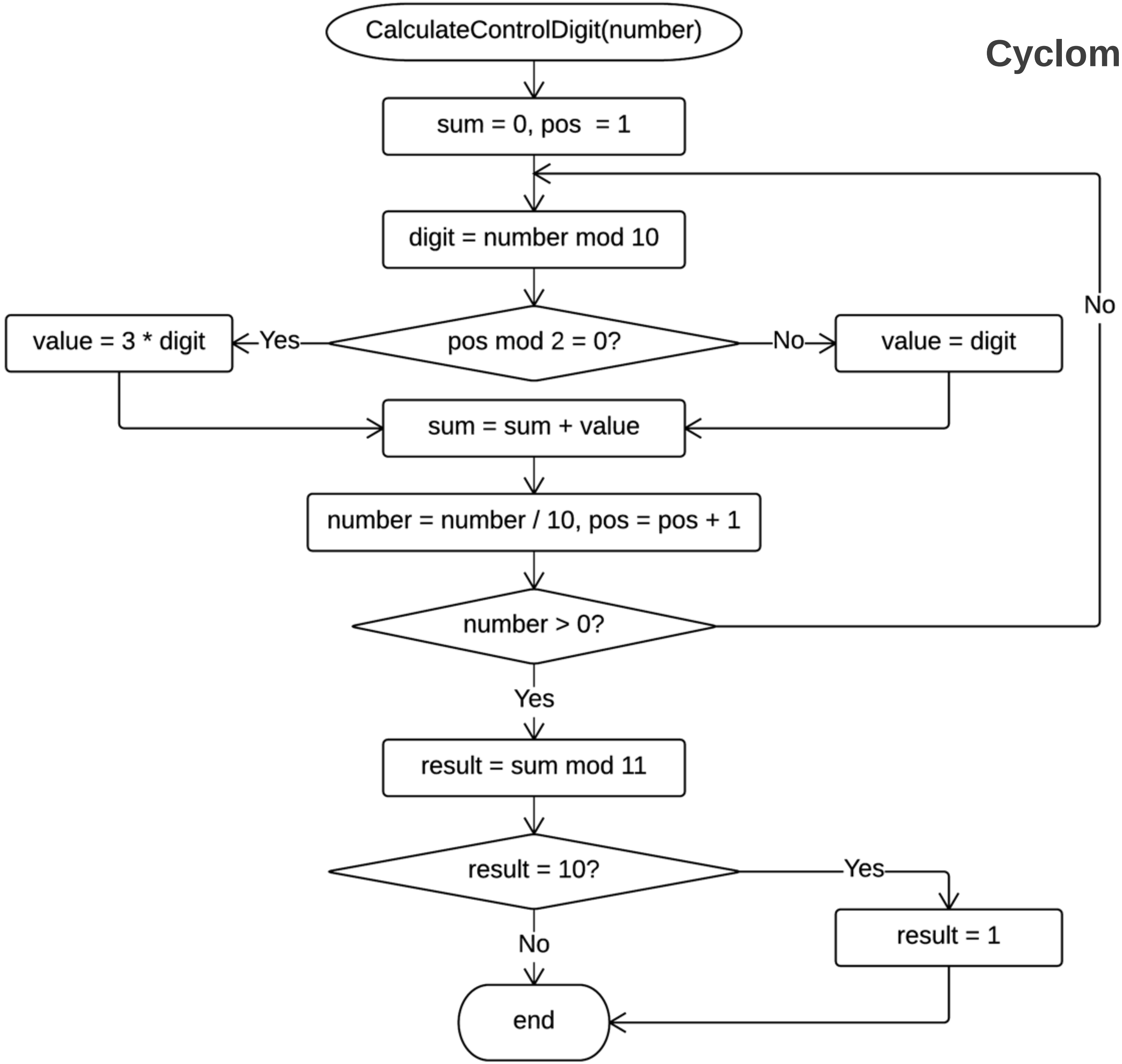
Code complexity not the same as asymptotic complexity

We will only analyze code complexity

- First analysis – by hand
- Later on – code metrics
  - Cyclomatic complexity metric



Total: 1x2x1x2 = 4 paths

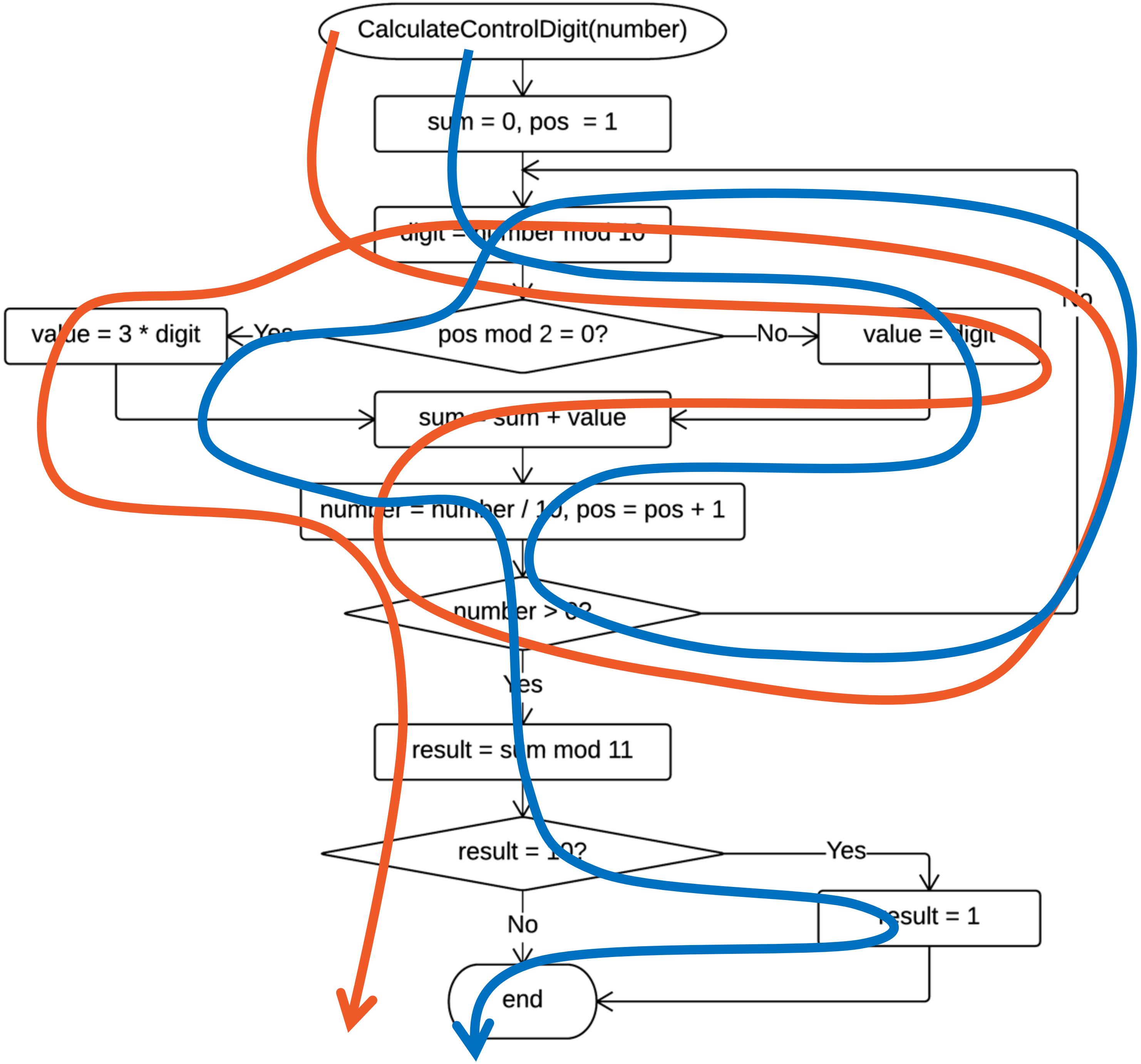


Cyclomatic complexity =
Number of edges – Number of nodes + 2

Number of edges = 15

Number of nodes = 13

Cyclomatic complexity = 15 – 13 + 2 = 4



Testing strategy

- Path coverage
- Branch coverage

Unit Tests – Path Coverage

Input	Output
x	x
3	3
31	1
42	3

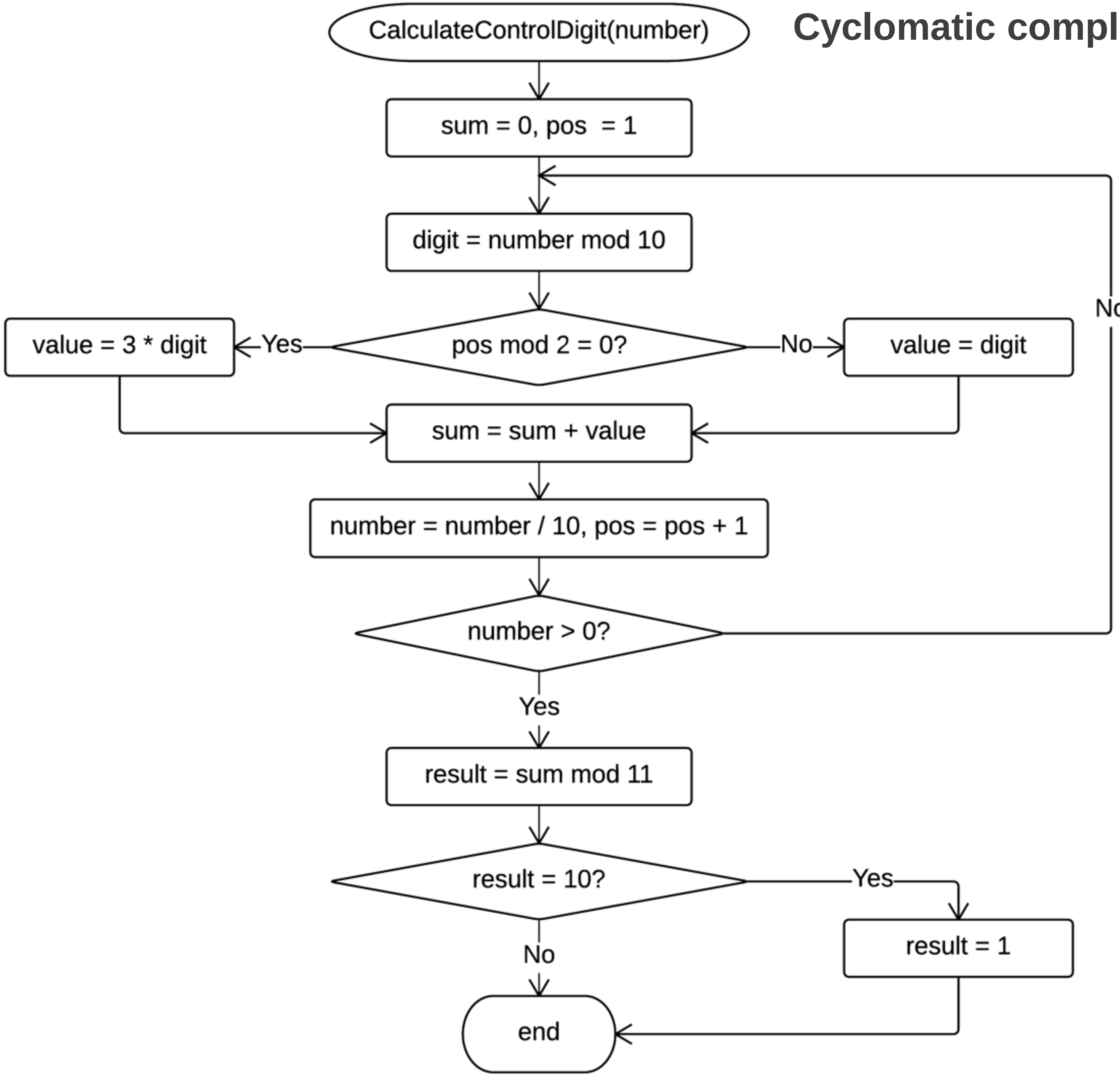
Unit Tests – Branch Coverage

Input	Output
31	1
42	3

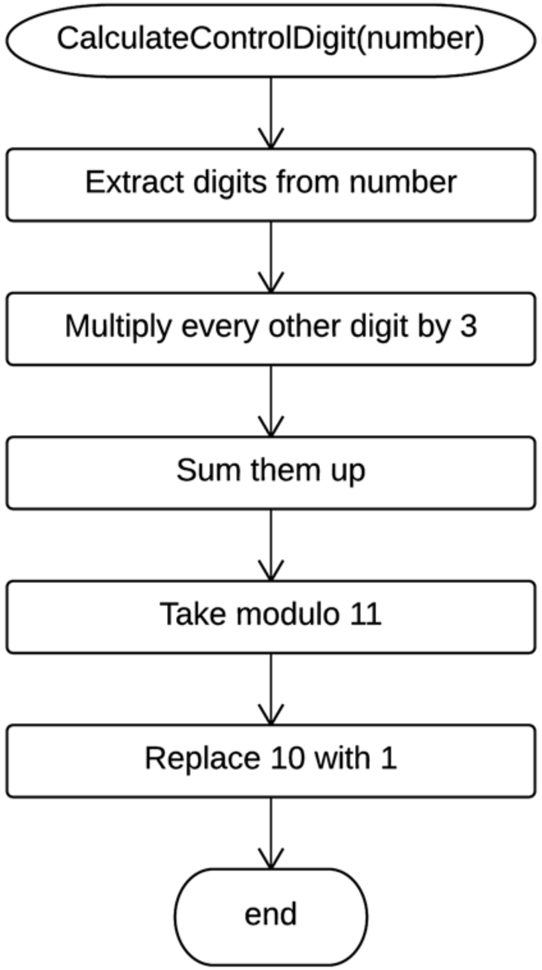
Cyclomatic complexity =

Number of edges – Number of nodes + 2

Cyclomatic complexity = 4



Cyclomatic complexity = 1


CalculateControlDigit(number)
Extract digits from number
Multiply every other digit by 3
Sum them up
Take modulo 11
Replace 10 with 1
end

Module 4:
Map-Reduce Design Pattern

The Following Modules

Module 2 – Coping With Null References

- Receiving null arguments
- Returning null from methods
- Testing if variable is null before using it

Module 3 – Null Object and Special Case Patterns

- Remove null references from a demo application

Module 4 – Map-Reduce pattern

- Rewrite the control digit calculation example
- Rely on collection operations and functional logic

The Following Modules

Module 5 – Iterator pattern and sequences

- Sequences offer great opportunities in design
- Help remove loops and iterations
- Replace them with generic operations on sequences

Module 6 – Option<T> functional type

- Collection of zero or one elements
- Can be used in places where there is no valid object to produce

The Following Modules

Module 7 – Service Locator pattern

- Often referred to as anti-pattern
- But that is not necessarily true
- Service locator can be used without causing negative effects

Module 8 – Guard clause and If-Then-Throw pattern

- Guard clauses do not belong to the class where they are implemented

Summary

Analyzing code complexity

- Counting independent paths
- Counting branches

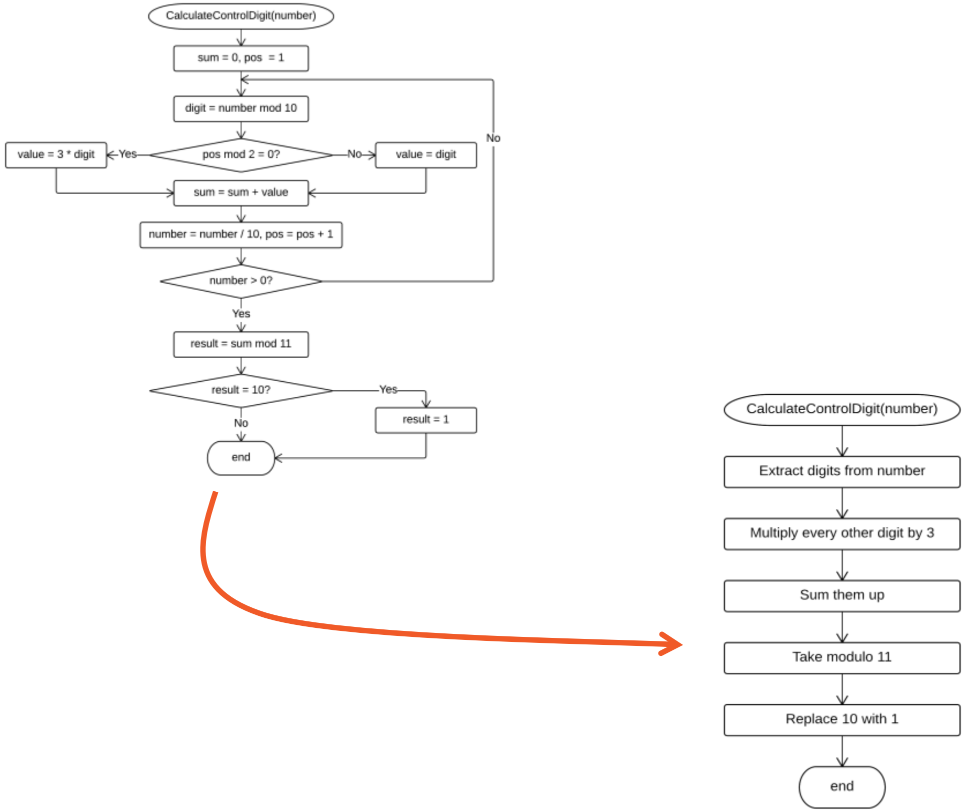
An example of complex implementation

An example of optional method arguments

Design patterns in this course simplify control flow

Suggestion for reading examples

- Observe how design patterns are causing branching and loops to be removed



CalculateControlDigit(number)

sum = 0, pos = 1

digit = number mod 10

pos mod 2 = 0?

Yes

value = 3 * digit

No

value = digit

sum = sum + value

number = number / 10, pos = pos + 1

number > 0?

No

Yes

result = sum mod 11

result = 10?

Yes

result = 1

No

end

CalculateControlDigit(number)

Extract digits from number

Multiply every other digit by 3

Sum them up

Take modulo 11

Replace 10 with 1

end

Coming next:
Coping with null references
Null Object and Special Case design patterns