Composing Functions into Larger Behavior

Zoran Horvat
CEO AT CODING HELMET
@zoranh75 http://csharpmentor.com

Understanding Virtual Functions

Object-oriented method

Functional style function

```
obj.f()
```

```
f(this)  <----------------------------->  f(obj)
```

Virtual functions

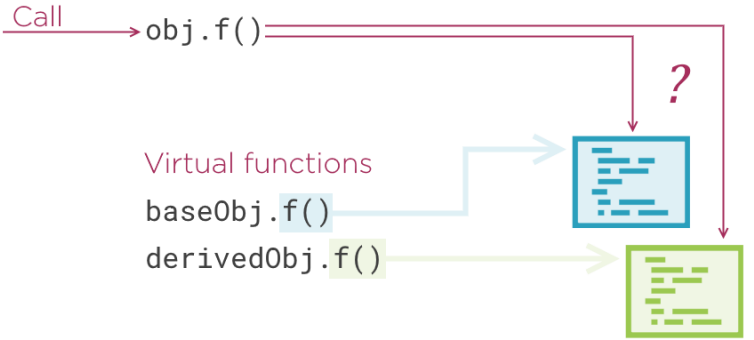
```
baseObj.f()
```

```
derivedObj.f()
```

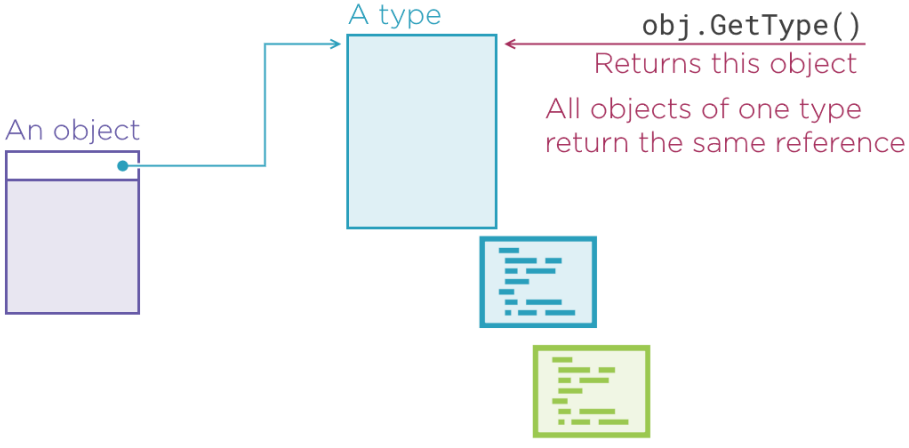
Base implementation

Override

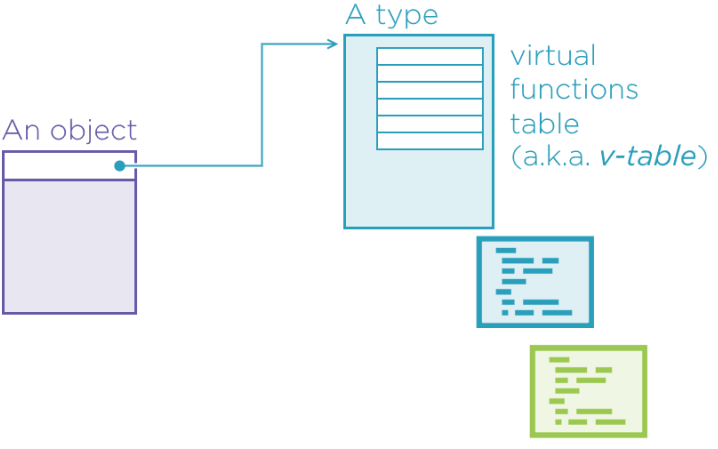
Understanding Virtual Functions



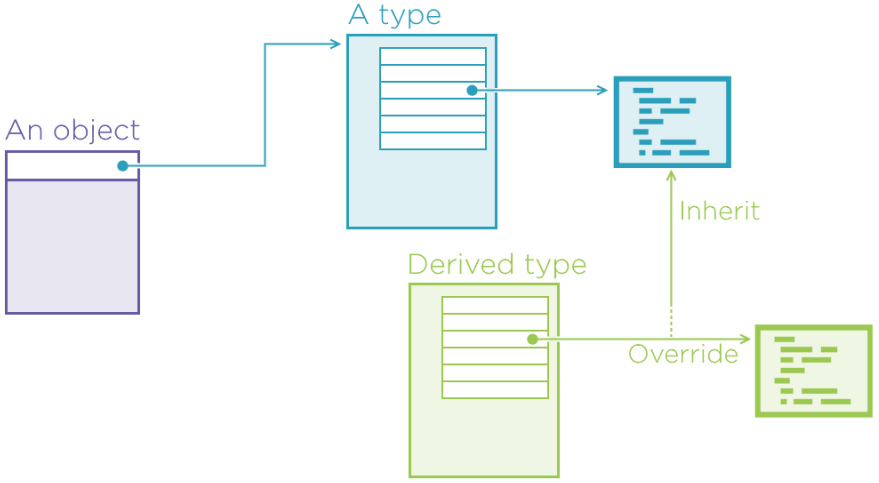
Understanding Virtual Functions



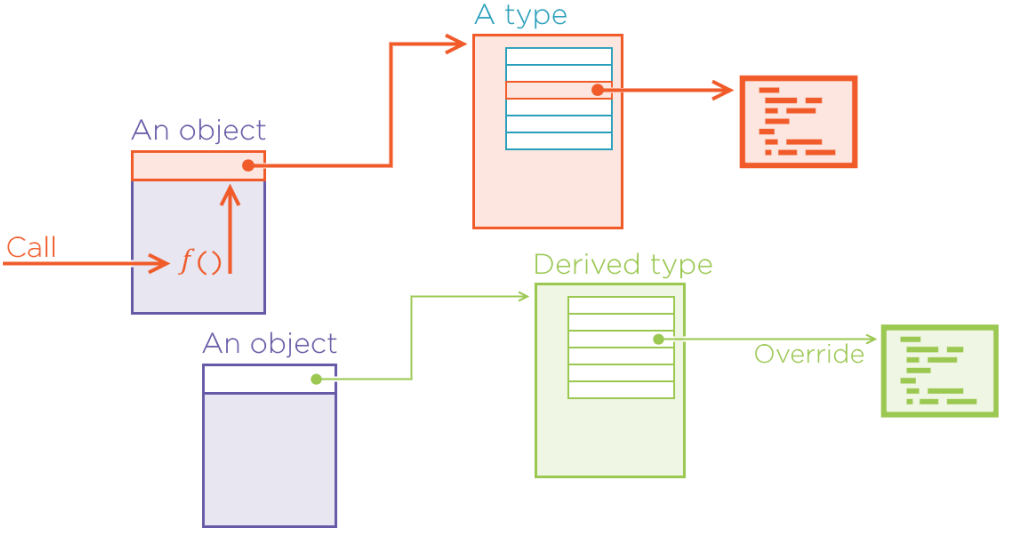
Understanding Virtual Functions



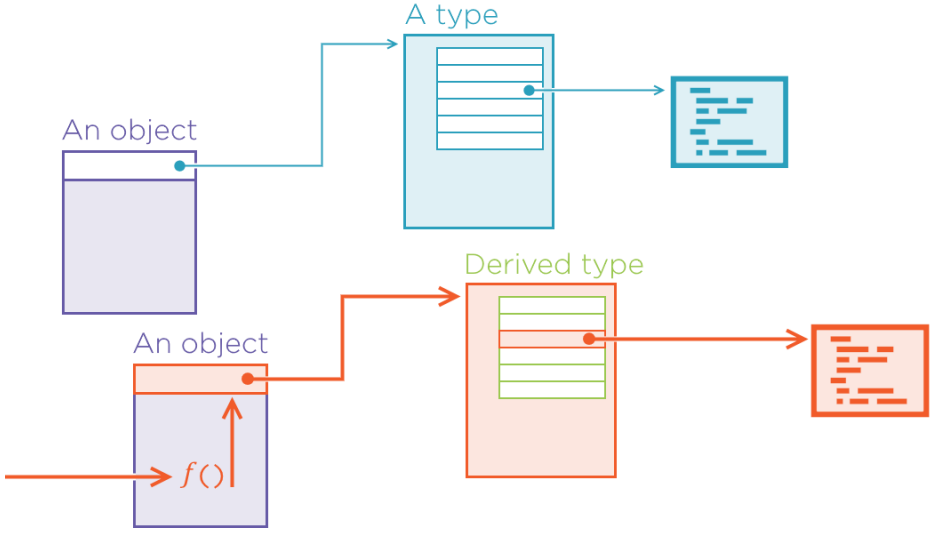
Understanding Virtual Functions



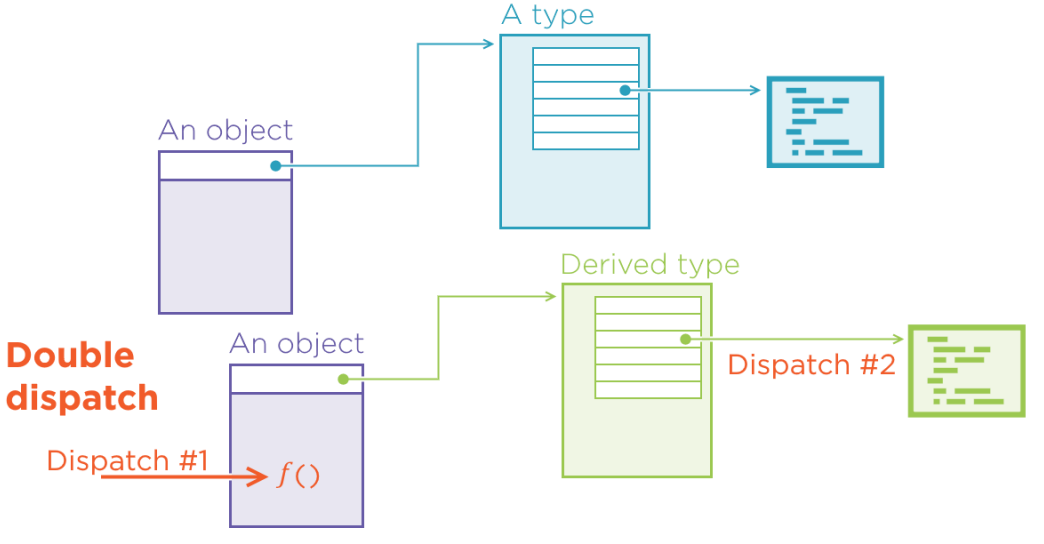
Understanding Virtual Functions



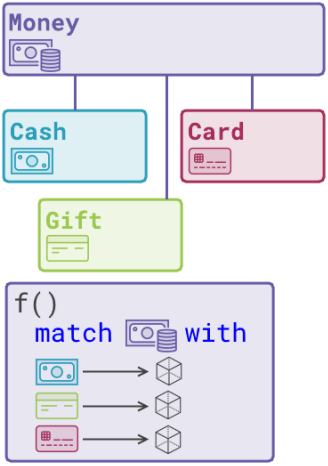
Understanding Virtual Functions

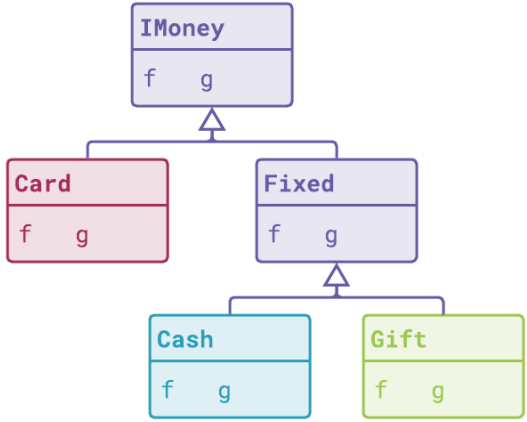


Understanding Virtual Functions

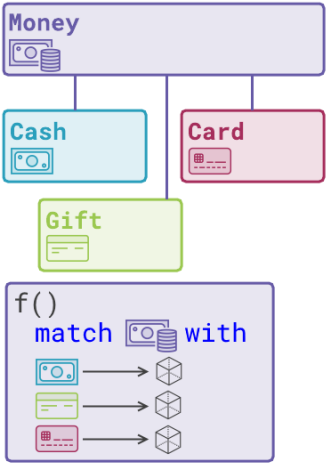


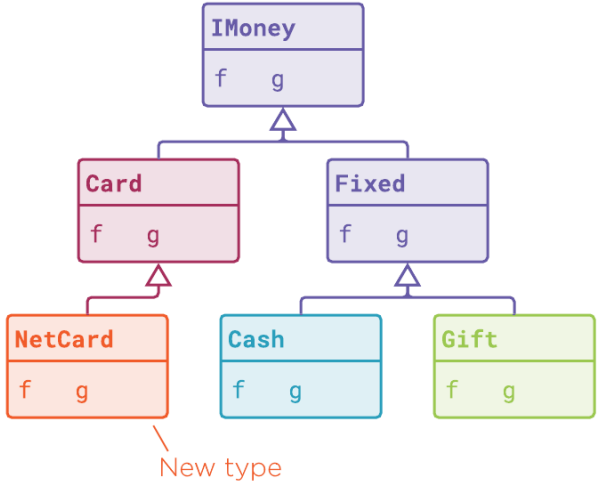
Pattern Matching vs. Virtual Functions



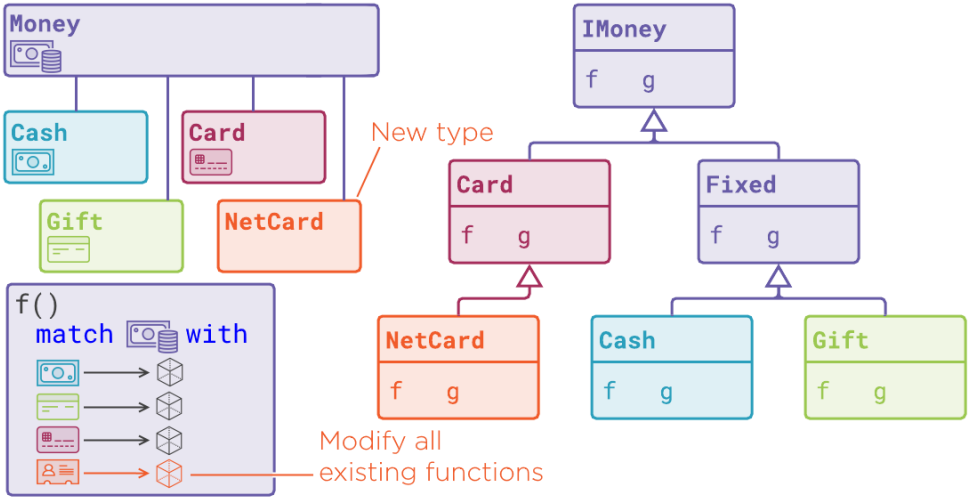


Pattern Matching vs. Virtual Functions

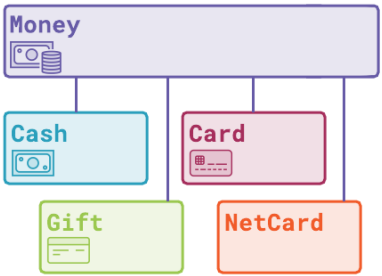




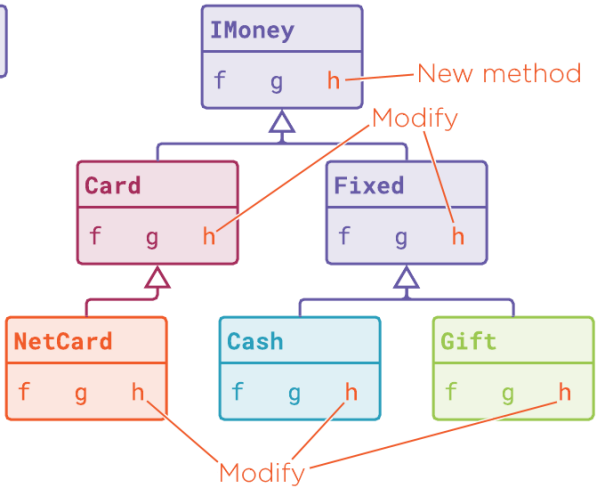
Pattern Matching vs. Virtual Functions



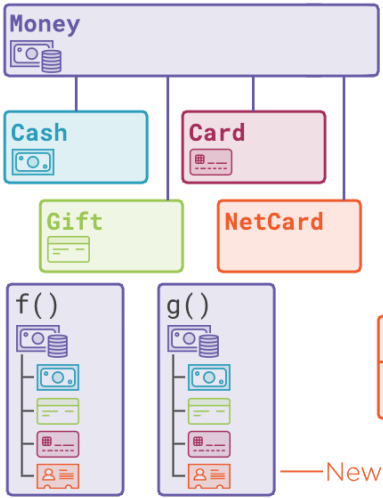
Pattern Matching vs. Virtual Functions

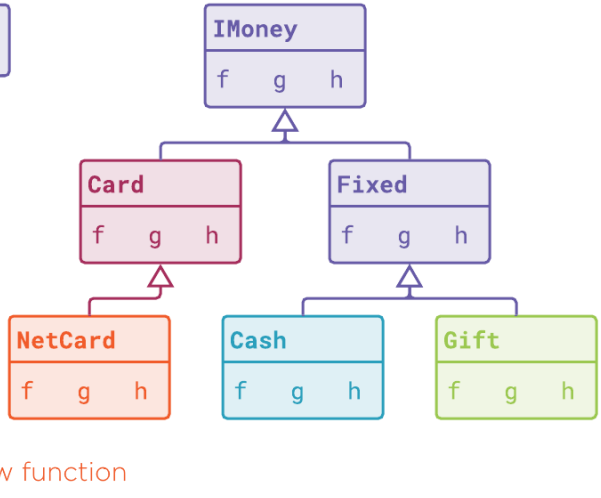



```
f()
  match  with
```



Pattern Matching vs. Virtual Functions

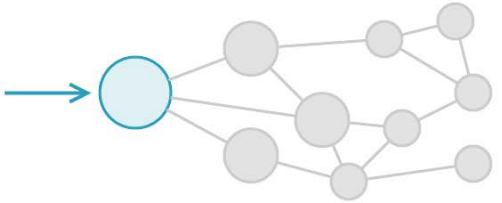




New function

Object Composition vs. Function Composition

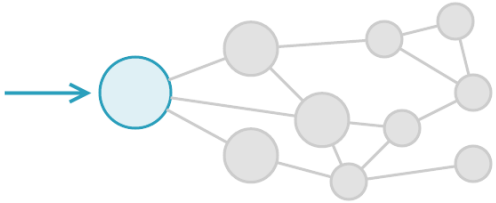
Object Composition vs. Function Composition



Strong points of
object-oriented design

- Object composition
- Polymorphic execution

Object Composition vs. Function Composition



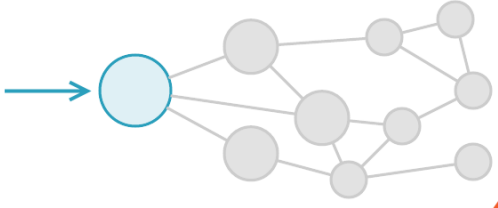
Strong points of
object-oriented design
- Object composition
- Polymorphic execution

Strengths of
functional programming
- Function composition

$$(f \circ g)(x) = f\big(g(x)\big)$$

Composition operator
"f after g"

Object Composition vs. Function Composition



Strong points of
object-oriented design

- Object composition
- Polymorphic execution

Strengths of
functional programming

- Function composition
- Partial function application

$$f(x, y)$$

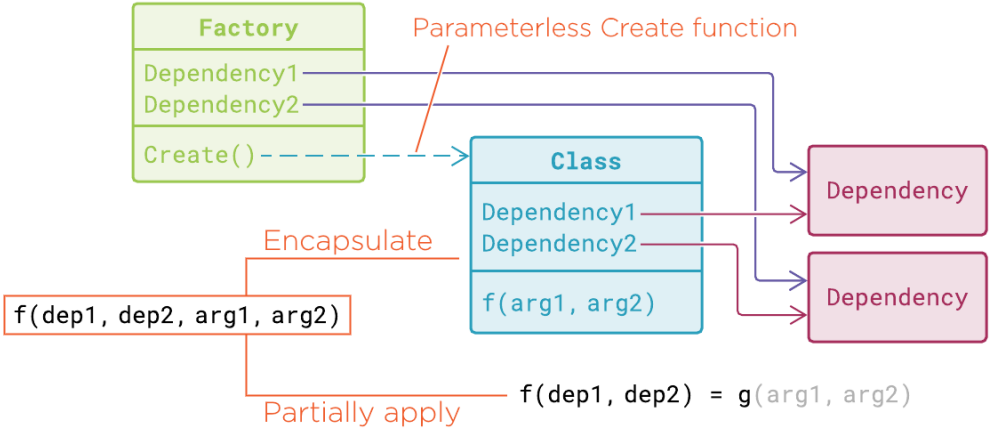
$$\downarrow$$

$$f(x) \longrightarrow g(y)$$

$$\downarrow$$

$$h \circ g$$

Dependencies vs. Partial Application



What Follows in This Module

Demonstrating function composition in action

Examining how long the argument list should be

Applying function composition to the payment model

Solving Quadratic Equation

$$ax^2 + bx + c = 0 \quad , a \neq 0$$

$$(p + q)^2 = p^2 + 2pq + q^2$$

Solving Quadratic Equation

$$ax^2 + bx + c = 0 \ , a \neq 0$$

$$(p + q)^2 = p^2 + 2pq + q^2$$

$$p^2 + 2pq + q^2 = (p + q)^2$$

$$ax^2 + \ bx \ + c$$

$$ax^2 + bx + c = 0 \qquad /\cdot a$$
$$a^2x^2 + abx + ac = 0$$
$$(ax)^2 + 2(ax)\frac{b}{2} + \left(\frac{b}{2}\right)^2 + ac - \left(\frac{b}{2}\right)^2 = 0$$
$$\left(ax + \frac{b}{2}\right)^2 = \frac{b^2 - 4ac}{4}$$
$$ax + \frac{b}{2} = \frac{\pm\sqrt{b^2 - 4ac}}{2} \qquad /\cdot 2$$
$$2ax + b = \pm\sqrt{b^2 - 4ac}$$
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$ax^2 + bx + c = 0 \qquad /\cdot a$$

$$a^2x^2 + abx + ac = 0$$

$$(ax)^2 + 2(ax)\frac{b}{2} + \left(\frac{b}{2}\right)^2 + ac - \left(\frac{b}{2}\right)^2 = 0$$

$$\left(ax + \frac{b}{2}\right)^2 = \frac{b^2 - 4ac}{4}$$

$$ax + \frac{b}{2} = \frac{\pm\sqrt{b^2 - 4ac}}{2} \qquad /\cdot 2$$

$$2ax + b = \pm\sqrt{b^2 - 4ac}$$

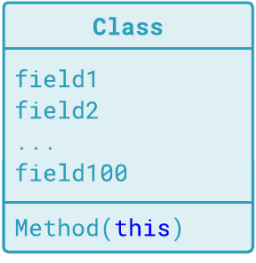
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

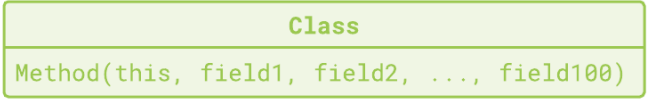
Quadratic equation roots

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$ ——— discriminant

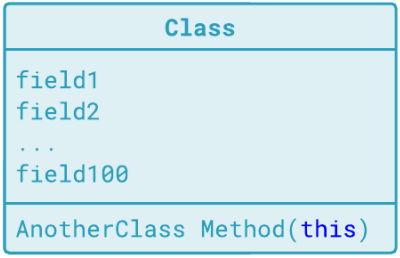
$$S = \begin{cases} \left\{-\frac{b}{2a}\right\} & , b^2 - 4ac = 0 \\ \left\{\frac{-b - \sqrt{b^2 - 4ac}}{2a}, \frac{-b + \sqrt{b^2 - 4ac}}{2a}\right\} & , b^2 - 4ac > 0 \\ \emptyset & , b^2 - 4ac < 0 \end{cases}$$

Introducing Fluent Interface Design





Introducing Fluent Interface Design




```
obj.Chaining().Calls().LikeA().Pro();
```

Summary

Function composition

- Compose small functions into larger ones
- Apply a function to the result of a previous function

F# composition operators

- Pipe-forward – |>
- Forward composition – >>
- Backward pipe and composition <|, <<

Summary

Function composition in C#

- Fluent interface design
- An object is passed to the next method
- LINQ is based on function composition

Benefits of function composition

- Shorter functions
- Cleaner code

Course Summary

Defining functions

- Function is not a process
- Function is a mapping

Functional programming in C#

- Avoid side effects
- Use higher-order functions
- Make types immutable
- Control flow with pattern matching

Course Summary

Functional sequences

- Treat sequences as objects
- Turn sequences immutable, too
- Rely on `IEnumerable<T>` and `ImmutableList<T>` whenever possible

Function composition

- Write small, isolated functions
- Compose them into larger functions

Course Summary

Why functional programming?

- To reduce code length
- To reduce number of bugs
- To easily extend the design