Identifying the Components of Fluent APIs

Floyd May
SOFTWARE CRAFTSMAN

@softwarefloyd https://medium.com/@floyd.may

Overview

- Characteristics
- Components
- History
- Notation

Method Chaining

```
return input
    .Where(x => x.Color == "Blue")
    .OrderBy(x => x.Width)
    .ThenBy(x => x.Height);
```

Method Chaining

```
testee.ShouldNotifyFor(t => t.FirstName)
    .And(t => t.FullName)
    .When(() => testee.FirstName = "John");
```

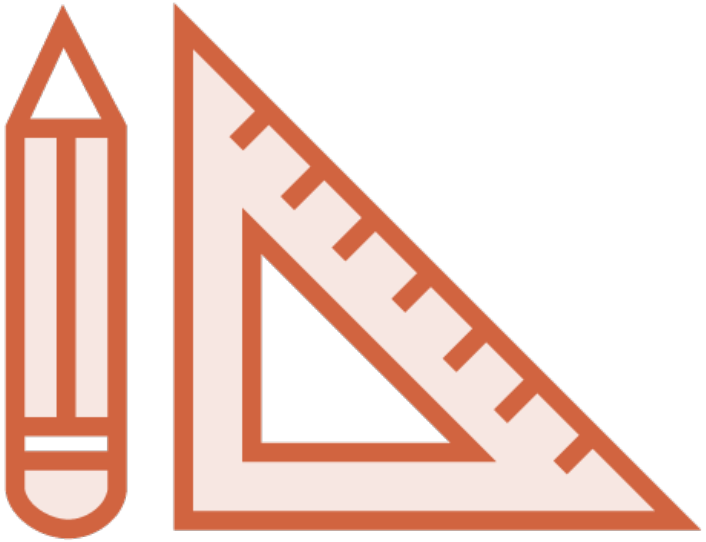
Method Chaining

```
var mapper = FluentMapper
    .ThatMaps<AddressDTO>().From<Address>()
        .ThatSets(tgt => tgt.Address1).From(src => src.Street1)
        .ThatSets(tgt => tgt.Address2).From(src => src.Street2)
        .ThatSets(tgt => tgt.Zip).From(src => src.PostalCode)
        .IgnoringSourceProperty(x => x.Notes)
    .Create();
```

Fluent APIs use method chaining

- Weave together related calls
- Compose complex concepts

Object-Oriented Design



Class names

- Nouns

Method names

- Verbs
- Verb phrases

```
return input
    .Where(x => x.Color == "Blue")
    .OrderBy(x => x.Width)
    .ThenBy(x => x.Height);
```

- Preposition
- Verb
- Preposition

```
return input
    .Where(x => x.Color == "Blue")
    .OrderBy(x => x.Width)
    .ThenBy(x => x.Height);
```

“Return elements from the input where the color property is equal to the string ‘Blue’; order the elements by the Width property, then by the Height property.”

```
testee.ShouldNotifyFor(t => t.FirstName)
    .And(t => t.FullName)
    .When(() => testee.FirstName = "John");
```

- Verb
- Conjunction
- Adverb

```
testee.ShouldNotifyFor(t => t.FirstName)
     .And(t => t.FullName)
     .When(() => testee.FirstName = "John");
```

“The testee should notify for the FirstName and FullName properties when the testee’s FirstName property is set to the string ‘John’.”

```
var mapper = FluentMapper
    .ThatMaps<AddressDTO>()
        .From<Address>()
        .ThatSets(tgt => tgt.Address1)
            .From(src => src.Street1)
        .ThatSets(tgt => tgt.Address2)
            .From(src => src.Street2)
        .ThatSets(tgt => tgt.Zip)
            .From(src => src.PostalCode)
        .IgnoringSourceProperty(x => x.Notes)
    .Create();
```

- Adjective
- preposition
- Adjective
- Preposition
- Adjective
- Preposition
- Adjective
- Preposition
- Gerund (Adverb)
- Verb

```
var mapper = FluentMapper.ThatMaps<AddressDTO>().From<Address>()
        .ThatSets(tgt => tgt.Address1).From(src => src.Street1)
        .ThatSets(tgt => tgt.Address2).From(src => src.Street2)
        .ThatSets(tgt => tgt.Zip).From(src => src.PostalCode)
        .IgnoringSourceProperty(x => x.Notes)
    .Create();
```

“Create a mapper that maps the AddressDTO type from the Address type, that sets Address1 from Street1, Address2 from Street2, Zip from PostalCode, ignoring the source property Notes.”

Fluent APIs are meant to read like prose


https://commons.wikimedia.org/wiki/File:Donald_Ervin_Knuth.jpg

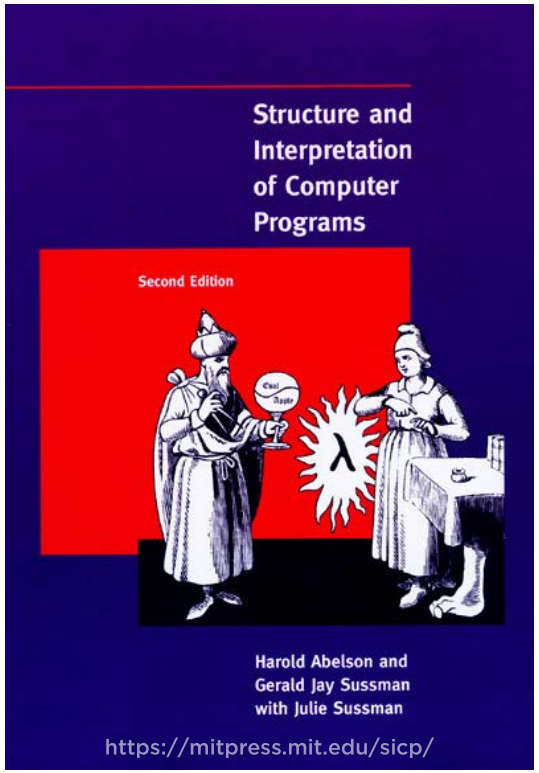
Donald Knuth

Literate Programming

- Stanford University
- 1984

Source code as essays

- Expository text
- Written for (human) reader



Structure and Interpretation of Computer Programs

Massachusetts Institute of Technology (MIT)

- Hal Abelson and Gerald Sussman
- 1985

Highly-regarded

- Manage complexity
- Large, non-trivial codebases

“...programs must be written for people to read, and only incidentally for machines to execute.”

Abelson and Sussman,
Structure and Interpretation of Computer Programs

Language features

- Lambdas
- Expression Trees
- Generics

Enhance design

- Improve readability
- Drive functionality
- Prevent errors

Embedded DSL

Functionality and Readability

Lambdas

```
return input
    .Where(x => x.Color == "Blue")
    .OrderBy(x => x.Width)
    .ThenBy(x => x.Height);
```

Delegate

```
private int CompareShapes(Shape x, Shape y)
{
  var result = x.Width.CompareTo(y.Width);

  if (result != 0)
      return result;

  return x.Height.CompareTo(y.Height);
}
```

Lambdas Prevent Errors

Original Sample

```
testee.ShouldNotifyFor(t => t.FirstName)
   .And(t => t.FullName)
   .When(() => testee.FirstName = "John");
```

Without Lambdas

```
testee.ShouldNotifyFor("FristName")
   .And("FullName")
   .When(() => testee.FirstName
        = "John");
```

Characteristics of Fluent APIs

Method chaining

- Compose complex concepts from simple elements

Written like prose

- Flex object-oriented naming idioms
- Building blocks of English sentences

Harness language features

- Lambdas, expressions, generics
- Improved API usability

English Language Constructs in Fluent APIs

Verb

```
...
  .OrderBy(x => x.Width)
```

Conjunction

```
...
  .And(x => x.LastName)
```

Gerund

```
...
  .IgnoringProperty(x =>
x.Notes)
```

Preposition

```
...
  .With(x => x.Boots)
```

Adjective

```
...
  .NotNull()
```

Adverb

```
...
  .ThenBy(x => x.Height)
```

Context

An object in a fluent API that:

facilitates method chaining

and

retains information from previous calls

```
public static IEnumerable<TSource> Skip<TSource>(
    this IEnumerable<TSource> source,
    int count
)
```

LINQ's Skip Method

Methods Similar to Skip

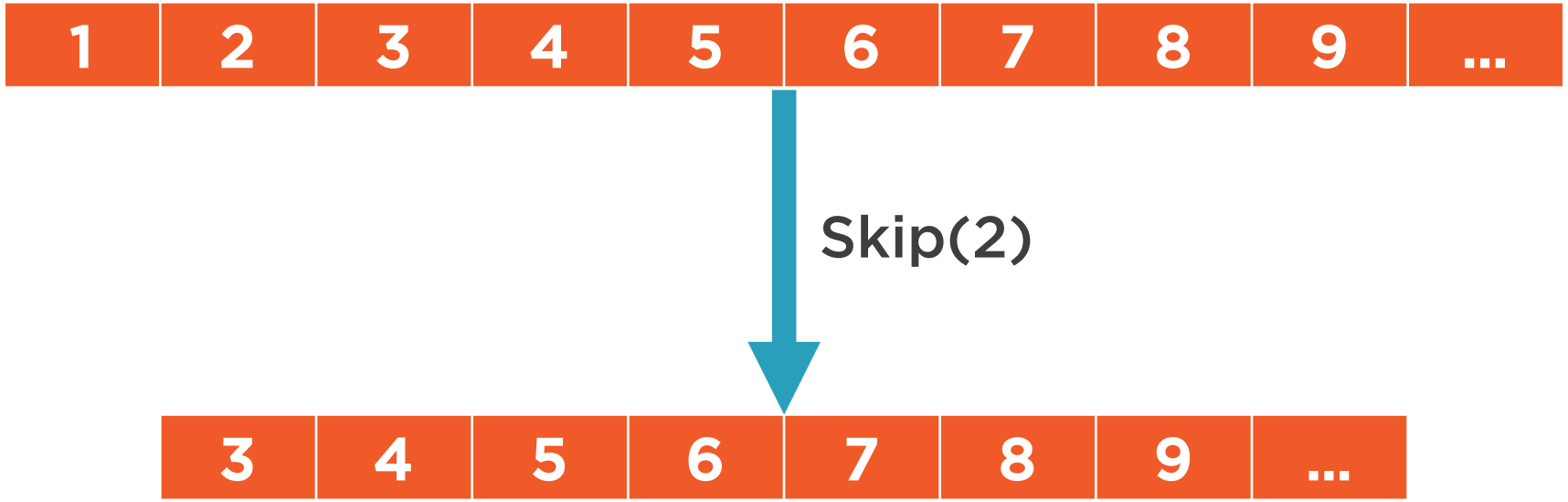
Where

SkipWhile

Take

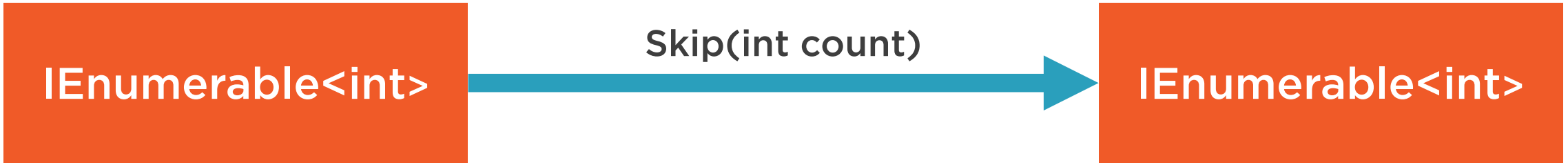
TakeWhile

One Context to the Next



The central design activity of building fluent APIs is crafting paths from one context to another.

Context Graph of Skip



Context Graph of Skip

Where(Func<int,bool> predicate)

TakeWhile(Func<int,bool> predicate)

SkipWhile(Func<int,bool> predicate)

Take(int count)

Skip(int count)

Simple Fluent Call

A fluent method call that returns a context that is the same type as the object upon which the call was made.

```
return new CarBuilder
    .With(Engines.V8)
    .With(Wheels.Alloy | Wheels.Mag)
    .WithInterior(InteriorMaterial.Leather, InteriorColors.Gray)
    .With(PaintColors.CobaltBlue);
```

Fluent Builder Example

Many implementations of the builder pattern use fluent syntax

```
public CarBuilder With(Engines engineType);
public CarBuilder With(Wheels wheelType);
public CarBuilder WithInterior(
        InteriorMaterial material,
        InteriorColor color
        );
```

CarBuilder Method Signatures

Same return type for each method

- Simple fluent calls

```
public static IEnumerable<TSource> Skip<TSource>(
    this IEnumerable<TSource> source,
    int count
)
```

LINQ's Skip Method – Again

Notice the use of the generic type parameter, TSource

- Similar signatures for SkipWhile, Take, TakeWhile, Where

Many Different Methods, Same Context Type

Where(Func<int,bool> predicate)
TakeWhile(Func<int,bool> predicate)
SkipWhile(Func<int,bool> predicate)
Take(int count)
Skip(int count)

Generic Context

A fluent context represented by a generic type whose generic type parameters are pertinent to the fluent API it participates in.

```
public static IEnumerable<TSource> OrderBy<TSource, TKey>(
    this IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector
)
```

LINQ's OrderBy Method

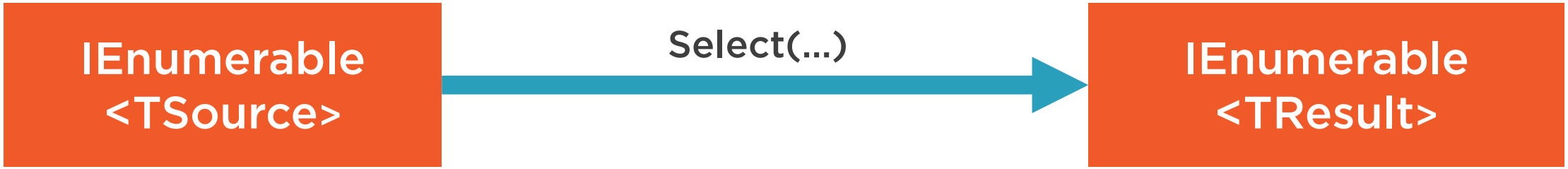
This is still a simple fluent call

- TKey helps to identify details of the keySelector, but doesn't affect the return type

```
public static IEnumerable<TResult> Select<TSource, TResult>(
    this IEnumerable<TSource> source,
    Func<TSource, TResult> selector
)
```

LINQ's Select Method

Context Graph of Select



Type Inference

C# compiler feature

Omit generic type parameters

- Only if there's enough information to deduce what types are needed

```
Enumerable.Range(0, 100)
    .Select<int, string>(x => x.ToString("0000"))
```

Harness Type Inference

Less code

Less typing

More purpose

A Sample from FluentMapper

```
public static TypeMappingSpec<TTarget, TSource>
    WithMappingAction<TTarget, TSource>(
        this TypeMappingSpec<TTarget, TSource> spec,
        Expression<Action<TTarget, TSource>> mappingExpression
    )
    where TTarget : class
    where TSource : class
{
    /* implementation */
}
```

Method signatures in fluent APIs tend to be...

Verbose

Less code and less typing for our users

- Type inference

Type safety

- Convert run-time errors to compile-time
- Harness Intellisense

Are These The Same?

Fluent APIs

Extension Methods

```
public static IEnumerable<TSource> Skip<TSource>(
    this IEnumerable<TSource> source,
    int count
)
```

LINQ Is Driven by Extension Methods

Often the first exposure to fluent APIs in C#

```
public static string FormatAsCurrency(this decimal value)
{
    /* implementation */
}
```

Is This Fluent?

The return type, string, likely doesn't facilitate chaining

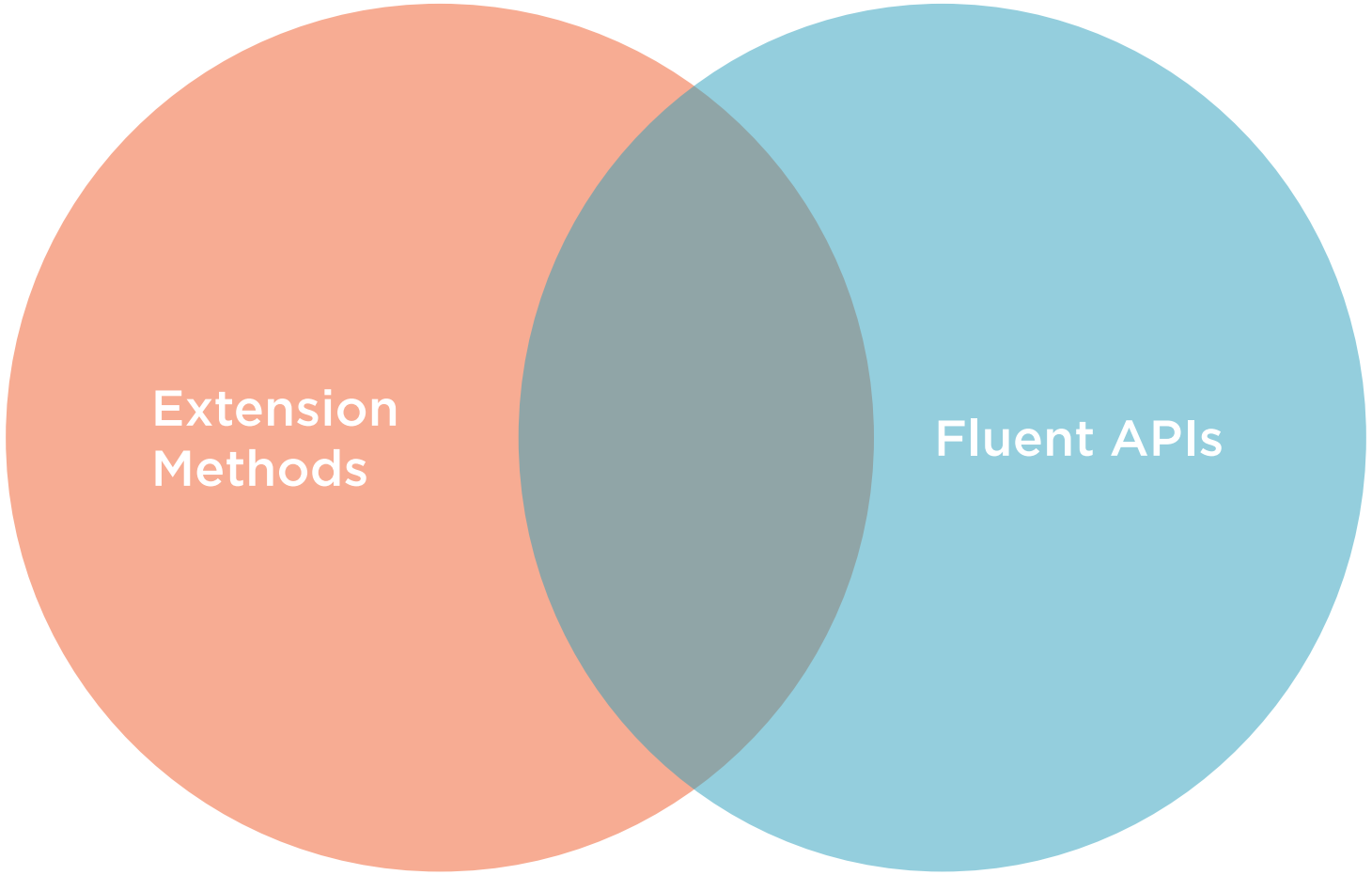
String is poorly-suited for retaining information about previous calls

```
public static void LogToConsole(this object value)
{
    /* implementation */
}
```

Is This Fluent?

A void return type can’t facilitate method chaining

Extension Methods and Fluent APIs



Characteristics of Fluent APIs

Method chaining

Expressed like prose

Uses C# language features

Silly Computer, Source Code Is for Humans!

- **Donald Knuth**
  - Stanford
  - Literate Programming

- **Abelson and Sussman**
  - Massachusetts Institute of Technology
  - *"... programs must be written for people to read..."*

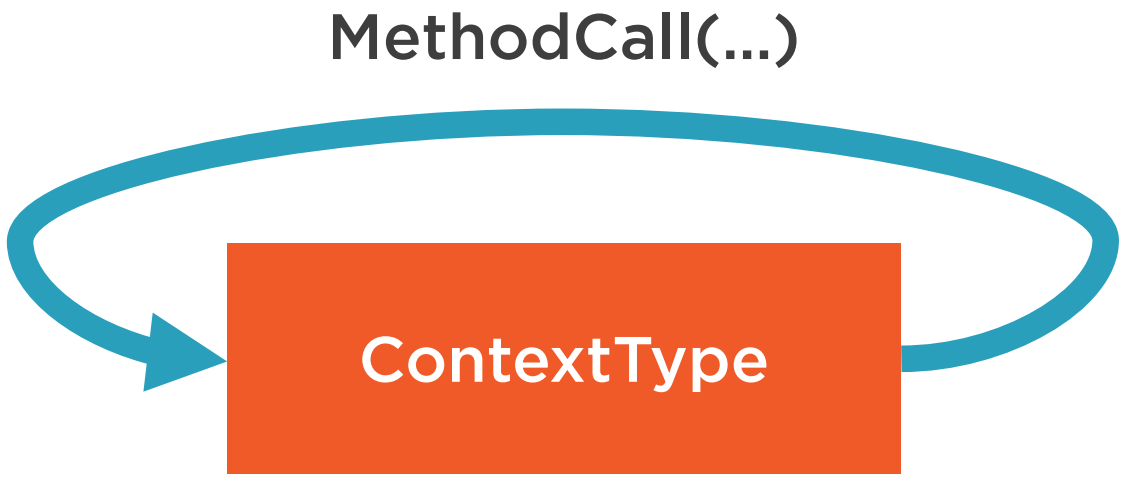
Context

Retains information from previous calls

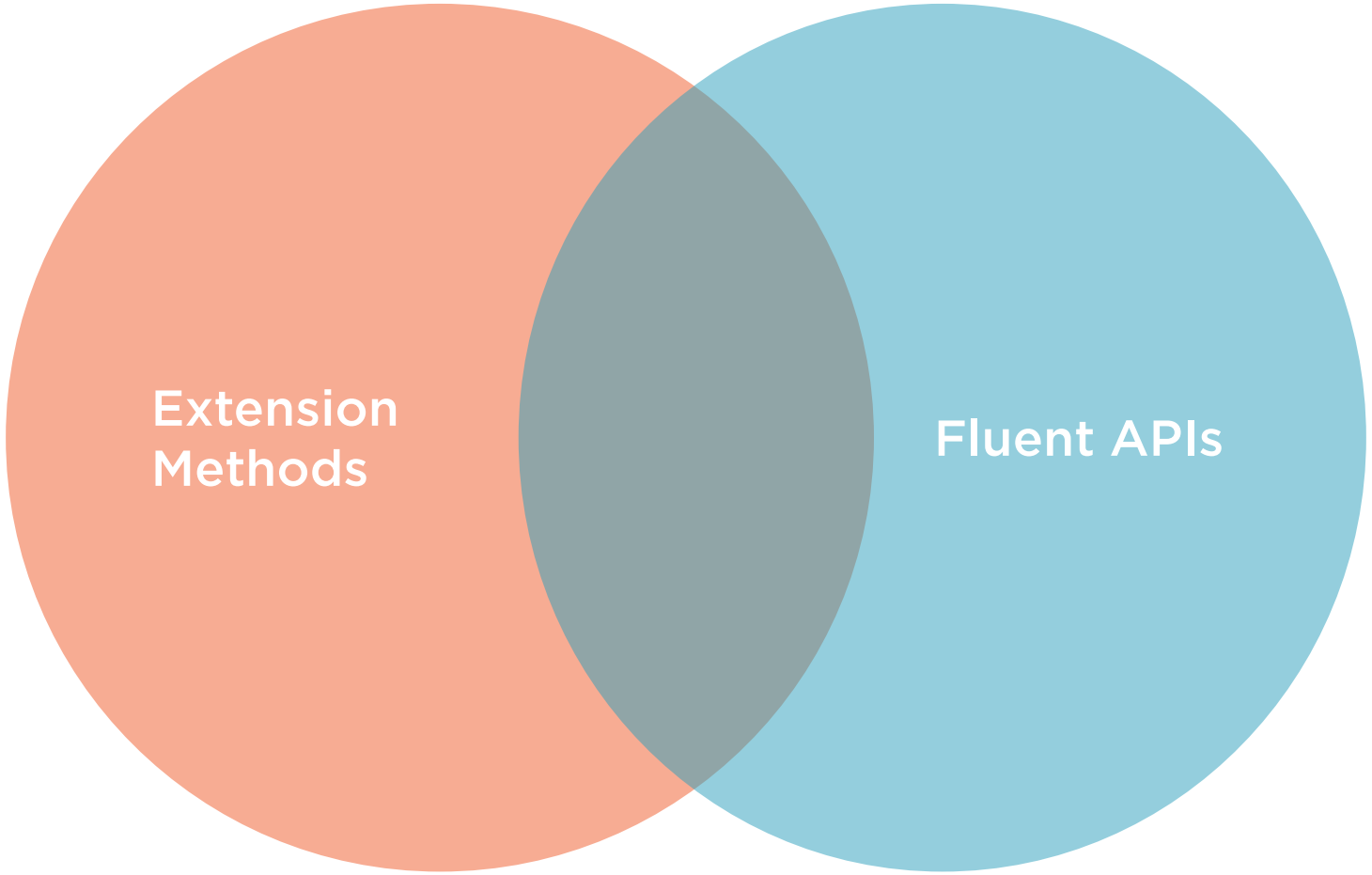
Facilitates further method chaining

Crafting Paths

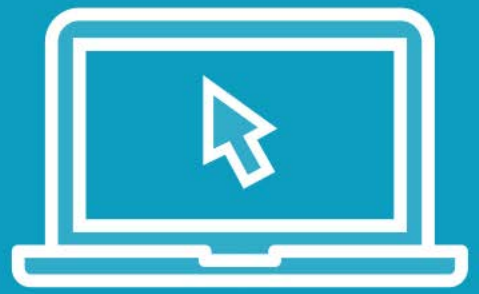
Simple Fluent Call



Extension Methods and Fluent APIs



Demo



Coming up:

- Seeman's INotifyPropertyChanged API
- FluentMapper

Thanks!