C# Extension Methods

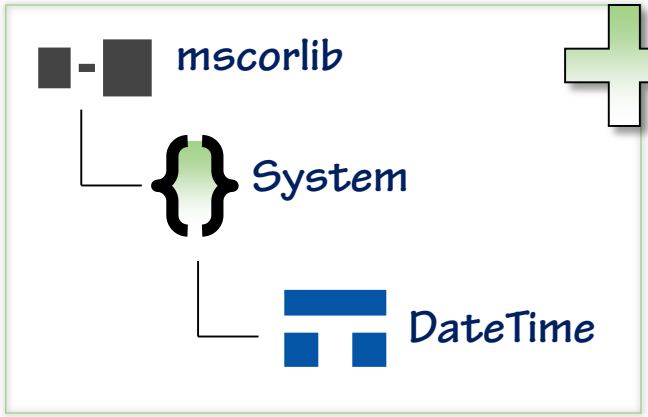
Module 1: Introducing Extension Methods

Elton Stoneman
geekswithblogs.net/eltonstoneman
elton@sixeyed.com

pluralsight
hardcore developer training

Introducing Extension Methods

- Extend a codebase by adding methods




```
public static string
 ToLegacyFormat(this DateTime dateTime)
{
    return dateTime.Year > 1930 ?
        dateTime.ToString("1yyMMdd") :
        dateTime.ToString("0yyMMdd");
}
```

```
var date = new DateTime(2013, 01, 01);
date = date.AddDays(1);
string legacyDate = date.ToLegacyFormat();
Assert.AreEqual("1130102", legacyDate);
```

Introducing Extension Methods

- **Extend types**
  - Classes & Structs
  - Interfaces
- **Extend generics**
  - Classes – List<T>
  - Interfaces – IEnumerable<T>
- **Enabling technology**
  - Adding to a third-party codebase
  - Adding to a class hierarchy
    - Without inheritance or composition
- **Straightforward to use**
  - But powerful

```
List<string>
List<Product>
List<IRepository>
```

```
IEnumerable<string>
IEnumerable<Product>
IEnumerable<IRepository>
```

About the Course

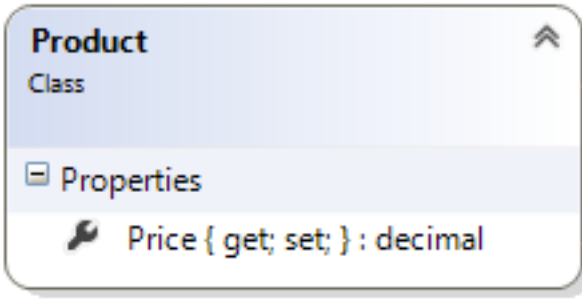
- **Aims**
  - Learn everything about extension methods in C#
  - What you can and can’t do
  - How extension methods work
  - Extension method library for (almost) any project
- **Pre-requisites**
  - Familiarity with C#
  - Classes, objects, interfaces
- **Toolset**
  - Visual Studio 2013
  - ILDasm
  - dotPeek (JetBrains)

Course Outline

- **Module 1: Introducing Extension Methods**
  - Writing and using
    - With types, interfaces & collections
  - What scenarios they enable
- **Module 2: Advanced Extension Methods**
  - Restrictions with extension methods
  - How .NET implements extensions
  - Building with different extension methods
- **Module 3: Extension Method Library (part 1)**
  - Core, Reflection, Entity Framework
- **Module 4: Extension Method Library (part 2)**
  - WCF, WebAPI, ASP.NET MVC

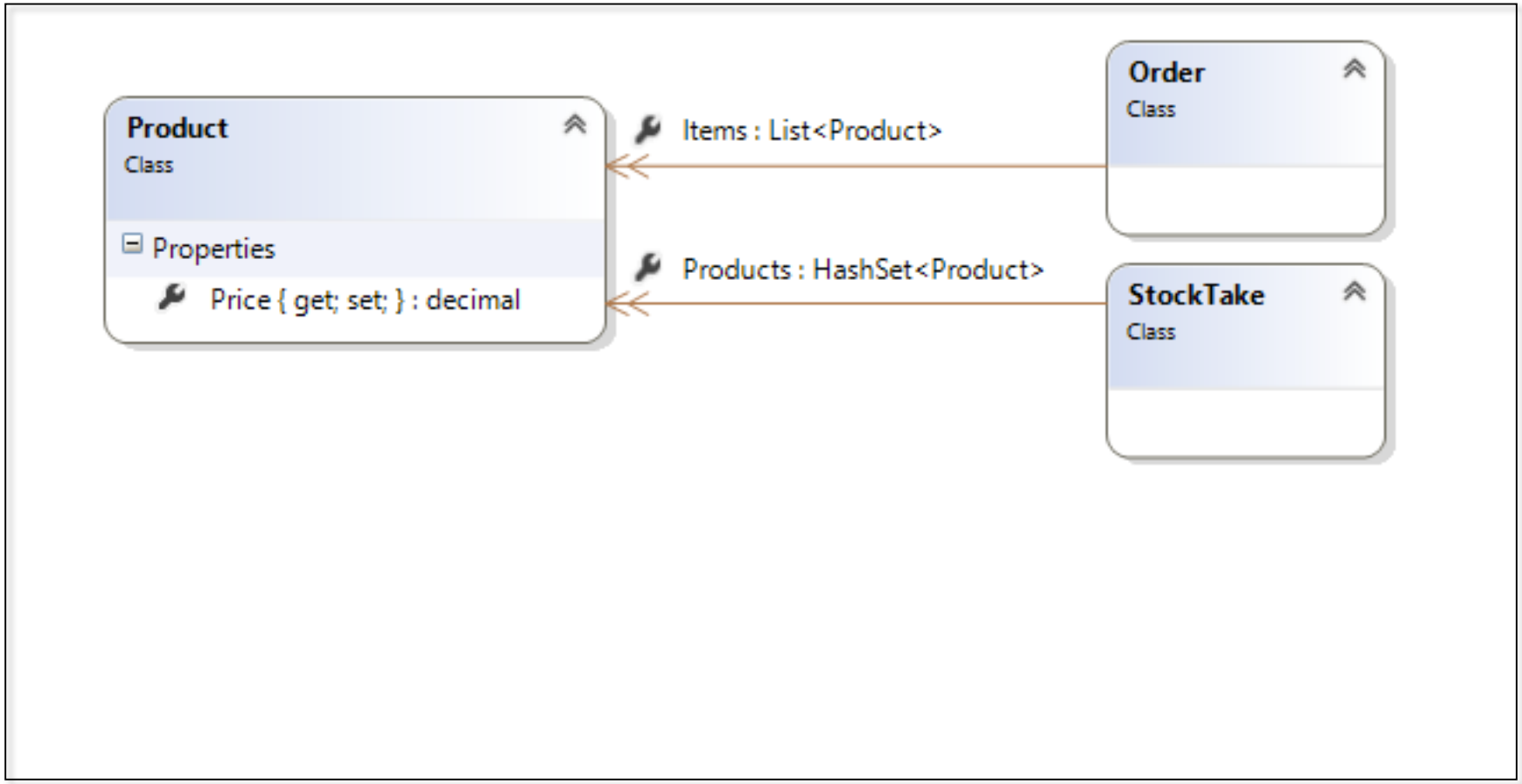
Introducing Extension Methods

- Centralize common code



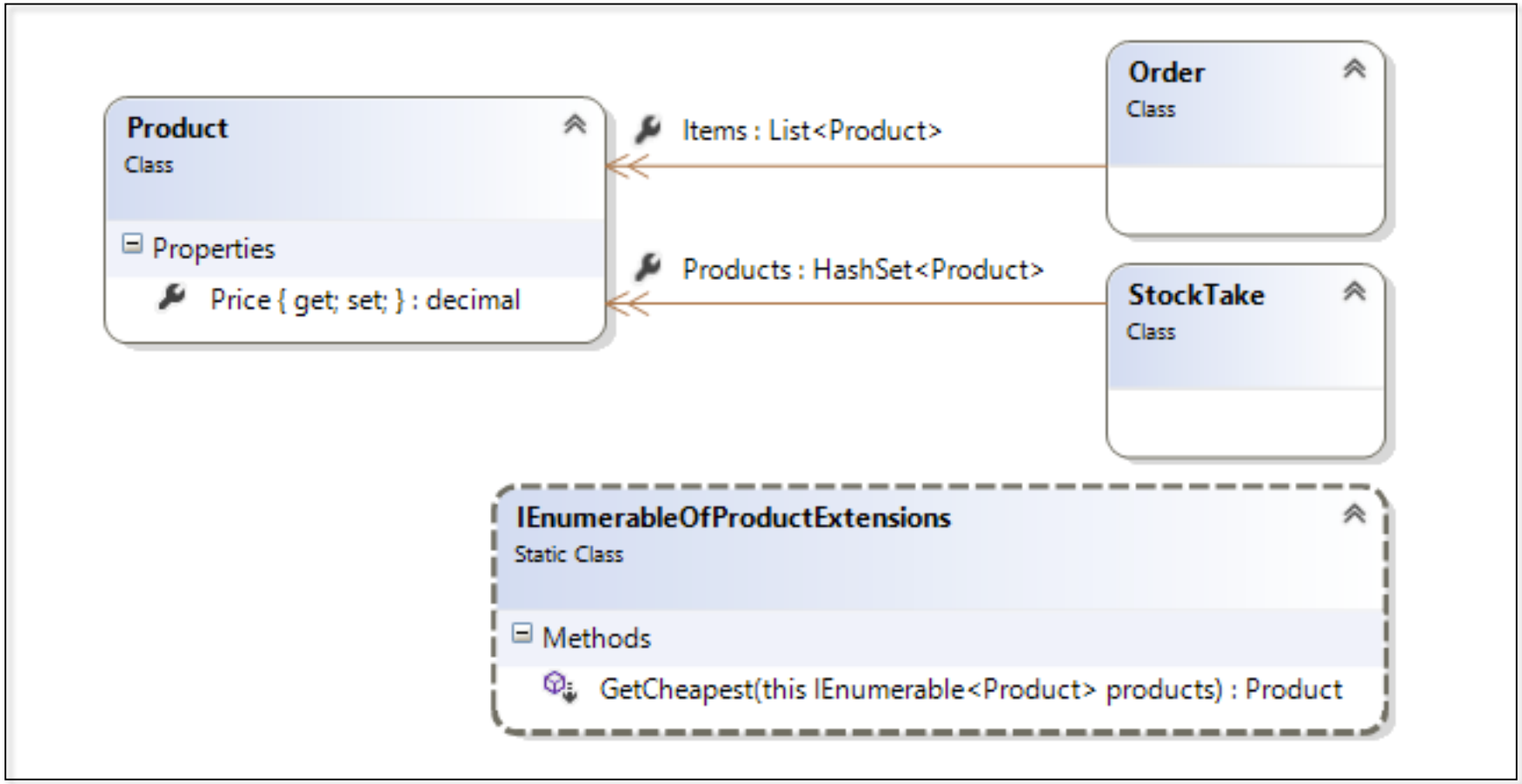
Introducing Extension Methods

- Centralize common code



Introducing Extension Methods

- Centralize common code



Demo 1: Extending System Types

Feature

Convert data into legacy system format

Task

Convert dates into legacy format (CYYMMDD)

Task

Convert names into legacy format (LAST, FIRST)

Demo 1: Extending System Types

Demo 1: Extending System Types

- **Writing extension methods**
  - Can live in any assembly
  - With any namespace
  - And any class name – typically
    - {Type}Extensions
    - {Feature}Extensions
- **Declaring extension methods**
  - Type to extend in **this** argument

```
public static string ToLegacyFormat(this DateTime dateTime)
```

  - Static method in static class

```
public static class LegacyExtensions
{
  public static string ToLegacyFormat(this DateTime dateTime)
```

Demo 1: Extending System Types

- **Using extension methods**
  - Reference extension method assembly
  - Import extension method namespace

```
using Sixeyed.Extensions.Samples;
```

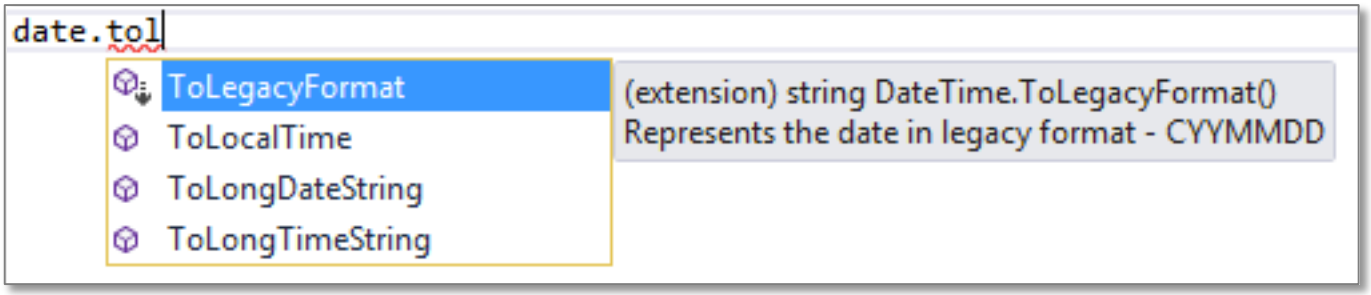
  - Call the method on an instance

```
var date = new DateTime(1920, 12, 31);
Assert.AreEqual("0201231", date.ToLegacyFormat());
```

```
var legacyName = "Elton Stoneman".ToLegacyFormat();
Assert.AreEqual("STONEMAN, ELTON", legacyName);
```

Tooling

- **Visual Studio**
  - IntelliSense for extension methods



  - But only when in scope
    - Referenced assemblies
    - Imported namespaces
- **Resharper**
  - Can import namespaces

Demo 2: Piggybacking Namespaces

Feature

Ouput dates in XML format (xsd:DateTime)

Task

Output dates in xsd:dateTime format - UTC

Task

Output dates in xsd:dateTime format – local time

Demo 2: Piggybacking Namespaces

Demo 2: Piggybacking Namespaces

- **Declaring extension methods**
  - Static class, static method
  - Type to extend in **this** argument
  - Use target type's namespace

```
namespace System
{
  public static class DateTimeExtensions
  {
    public static string ToXmlDateTime(this DateTime dateTime)
```

- **Using extension methods**
  - Directly

```
var dateTime = new DateTime(2013,10,24,13,10,15,951);
string xmlDateTime = DateTimeExtensions.ToXmlDateTime(dateTime);
```

Demo 2: Piggybacking Namespaces

- **Overloading extension methods**
  - Same name, different arguments
  - Type to extend in **this** argument

```
public static string ToXmlDateTime(this DateTime dateTime) { }

public static string ToXmlDateTime(this DateTime dateTime,
                XmlDateTimeSerializationMode serializationMode) { }
```

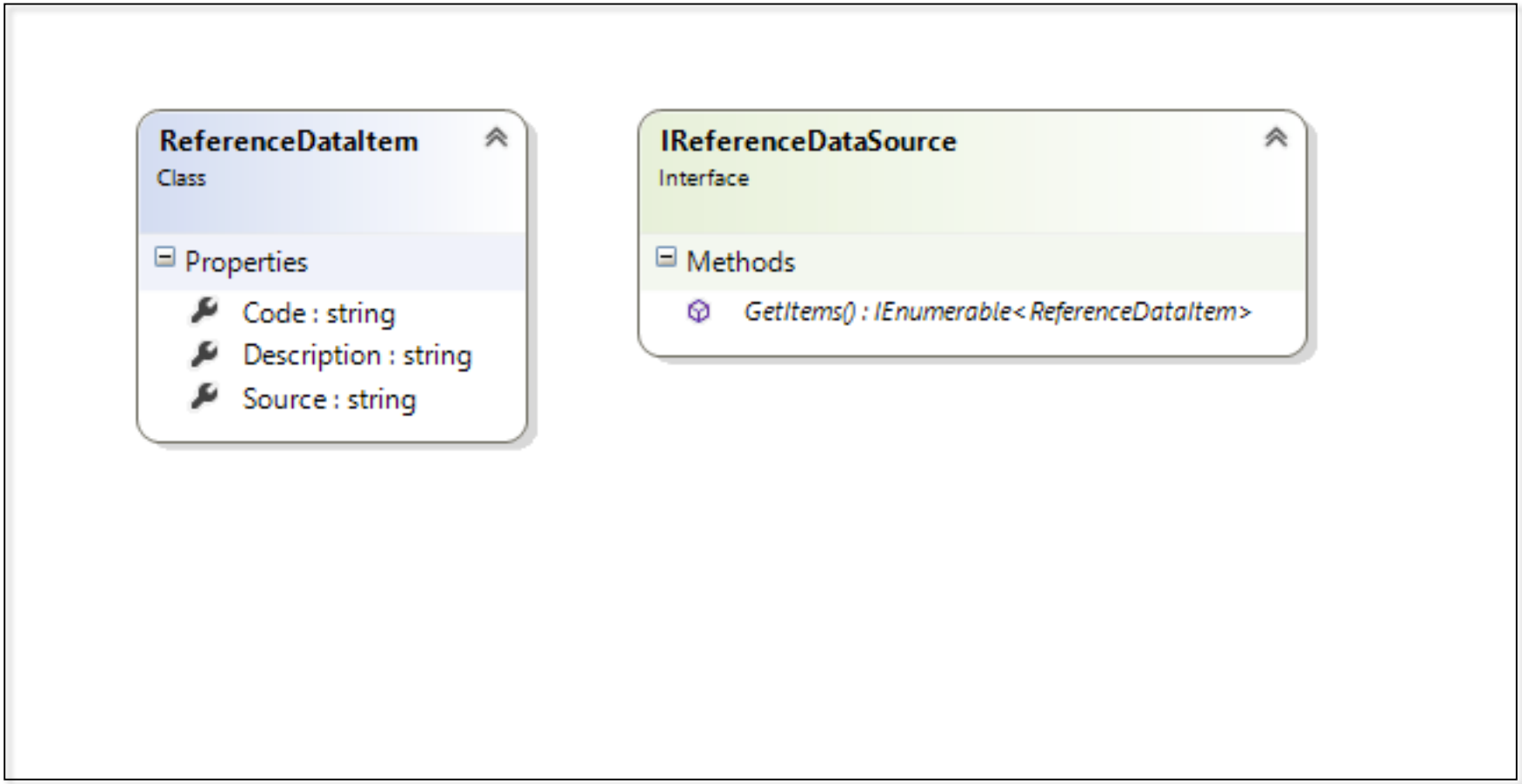
  - With optional parameters

```
public static string ToXmlDateTime(this DateTime dateTime) { }

public static string ToXmlDateTime(this DateTime dateTime,
          XmlDateTimeSerializationMode serializationMode =
          XmlDateTimeSerializationMode.Utc) { }
```

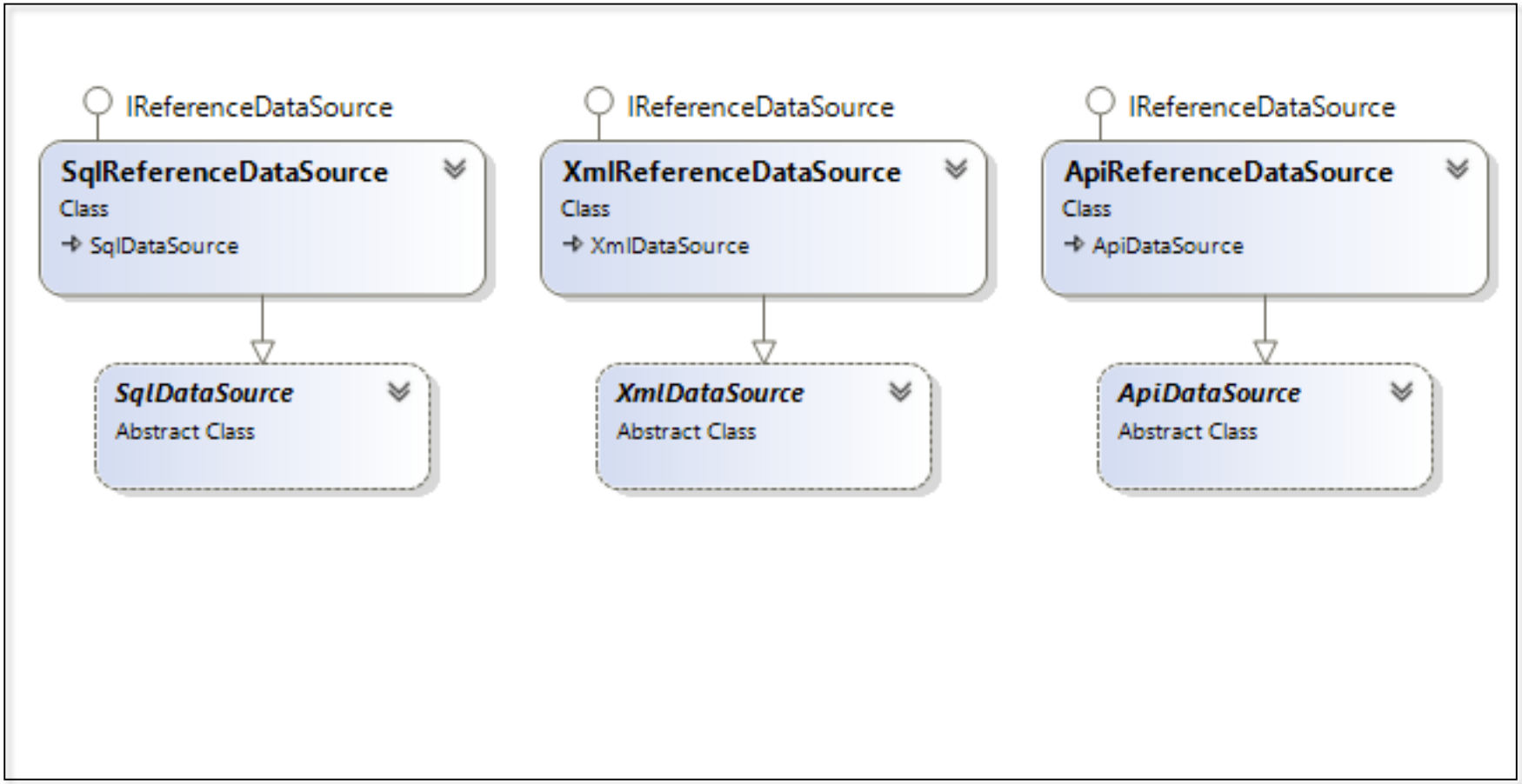
Simplifying the Codebase

- Extension instead of inheritance or composition



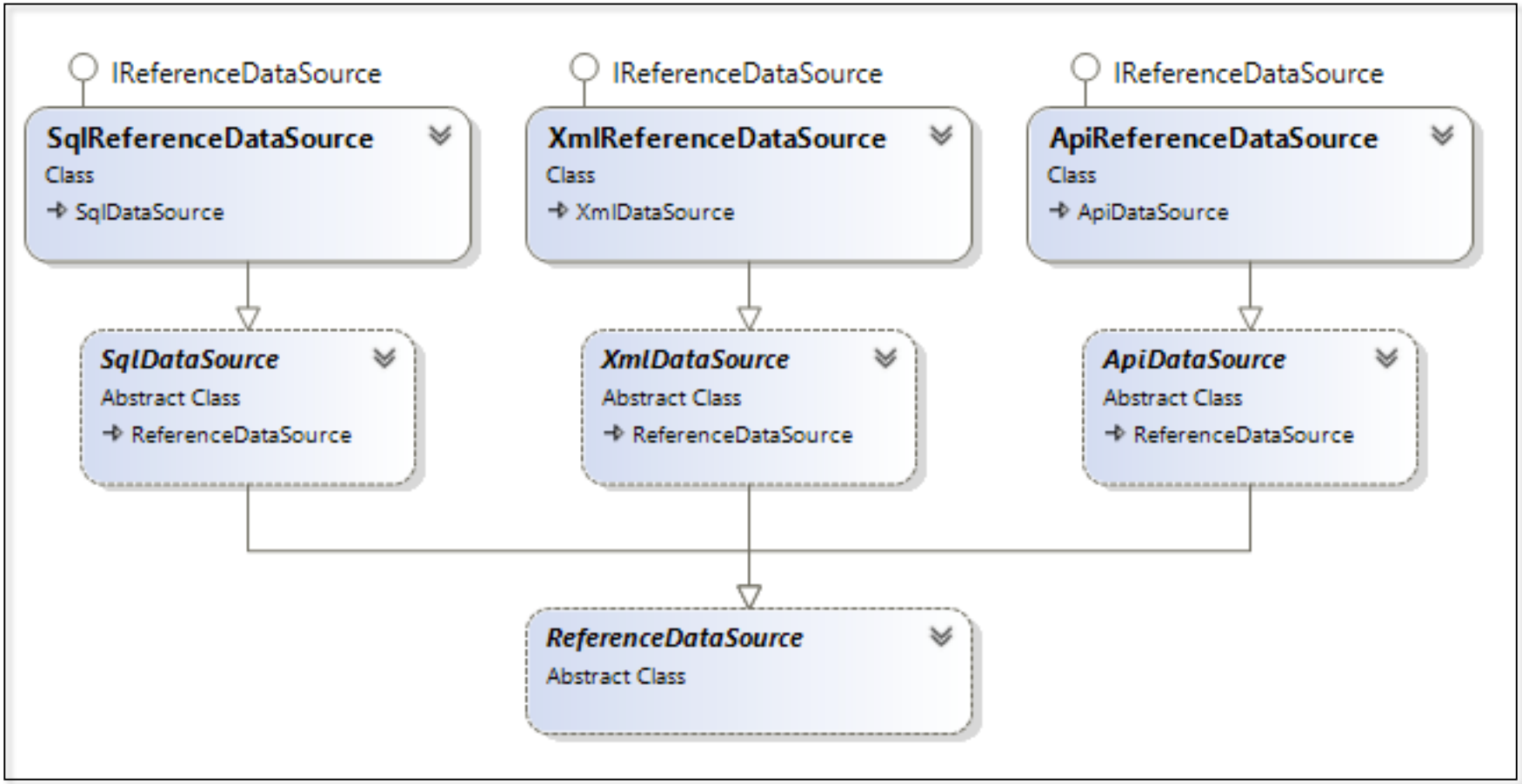
Simplifying the Codebase

- Extension instead of inheritance or composition



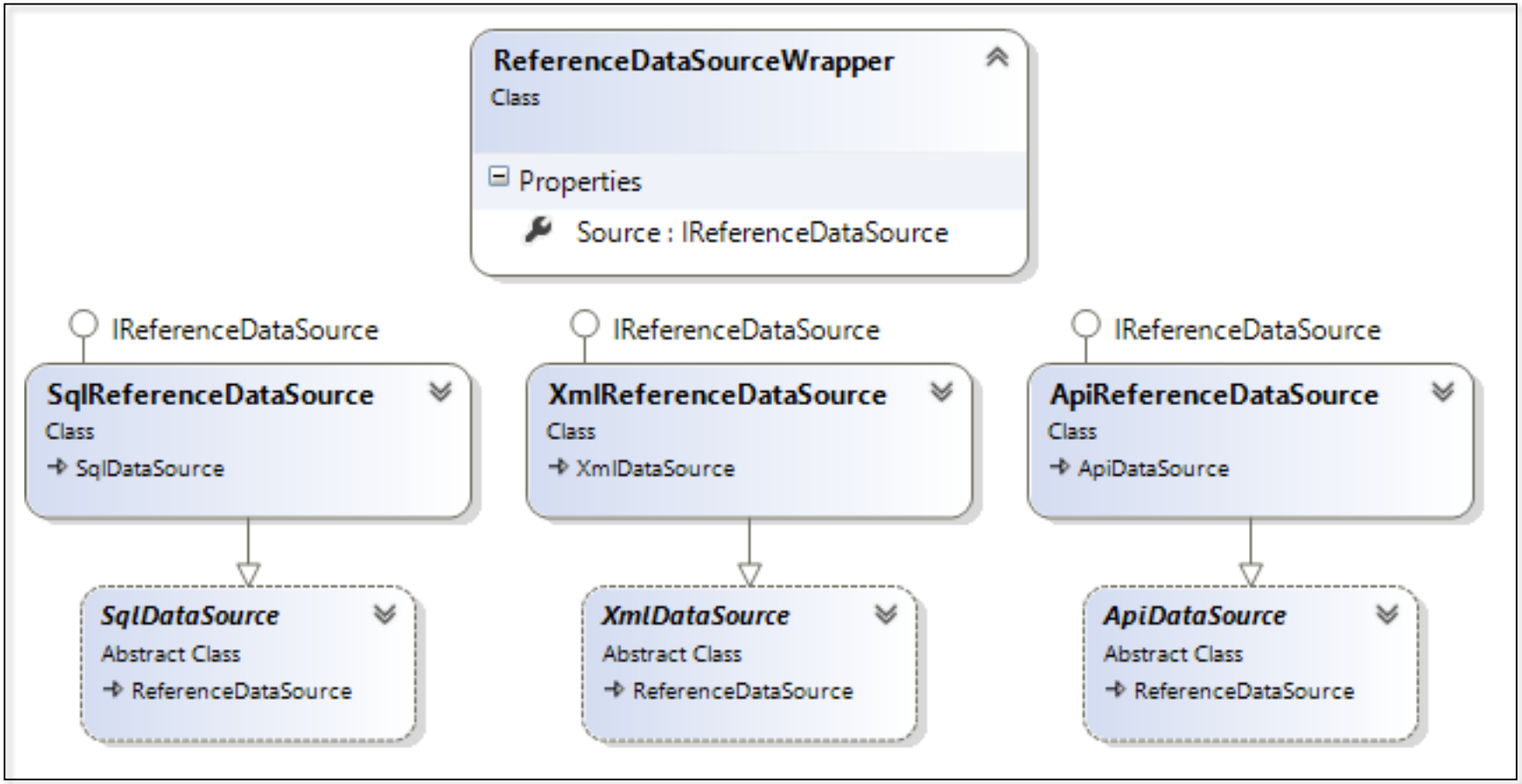
Simplifying the Codebase

- Extension instead of inheritance or composition



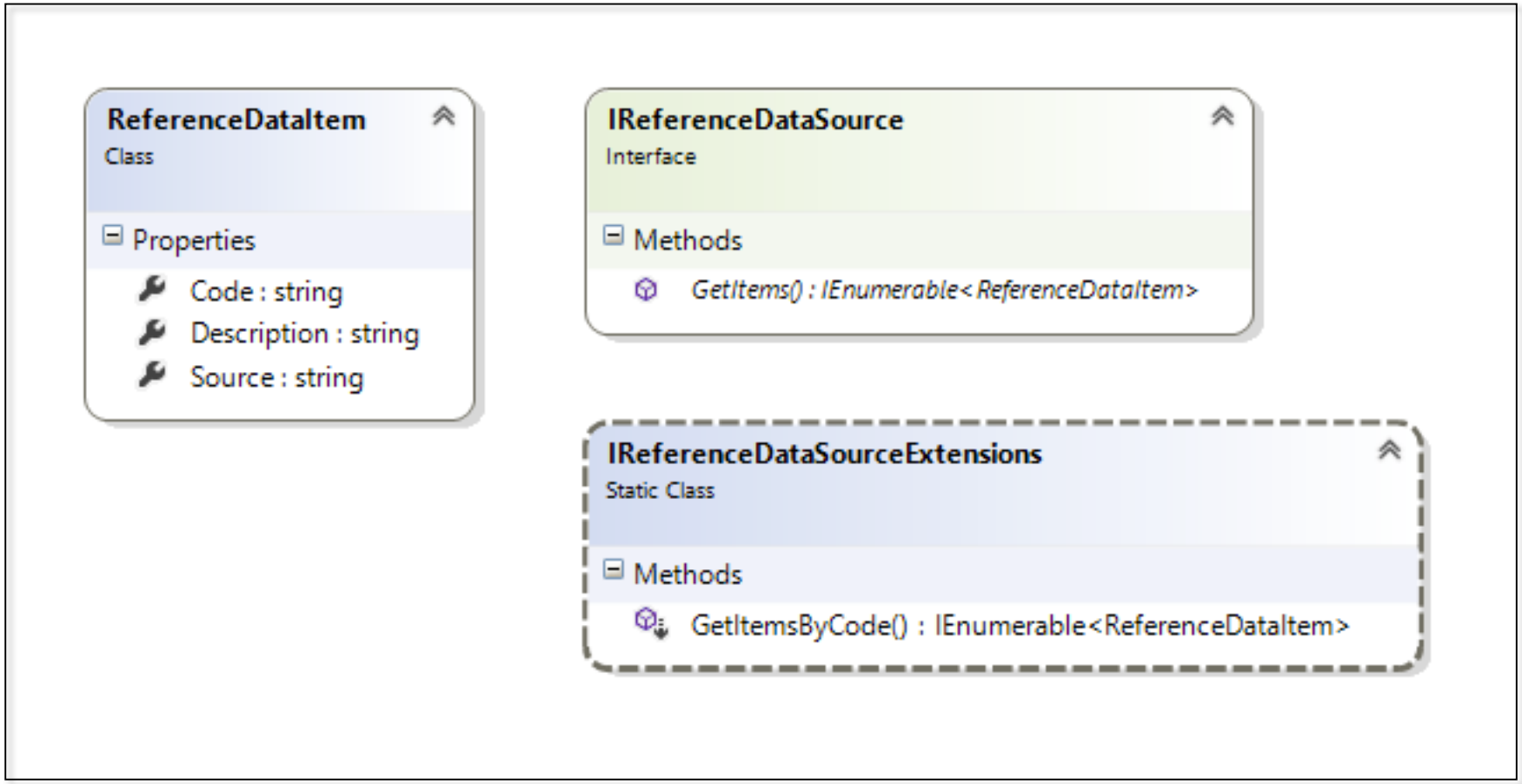
Simplifying the Codebase

- Extension instead of inheritance or composition



Simplifying the Codebase

- Extension instead of inheritance or composition



Demo 3: Extending Interfaces

Feature

Get reference data by type from any source

Task

Extend reference data source interface

Demo 3: Extending Interfaces

Demo 3: Extending Interfaces

- **Writing extension methods**
  - Static class, static method
  - Interface to extend in **this** argument

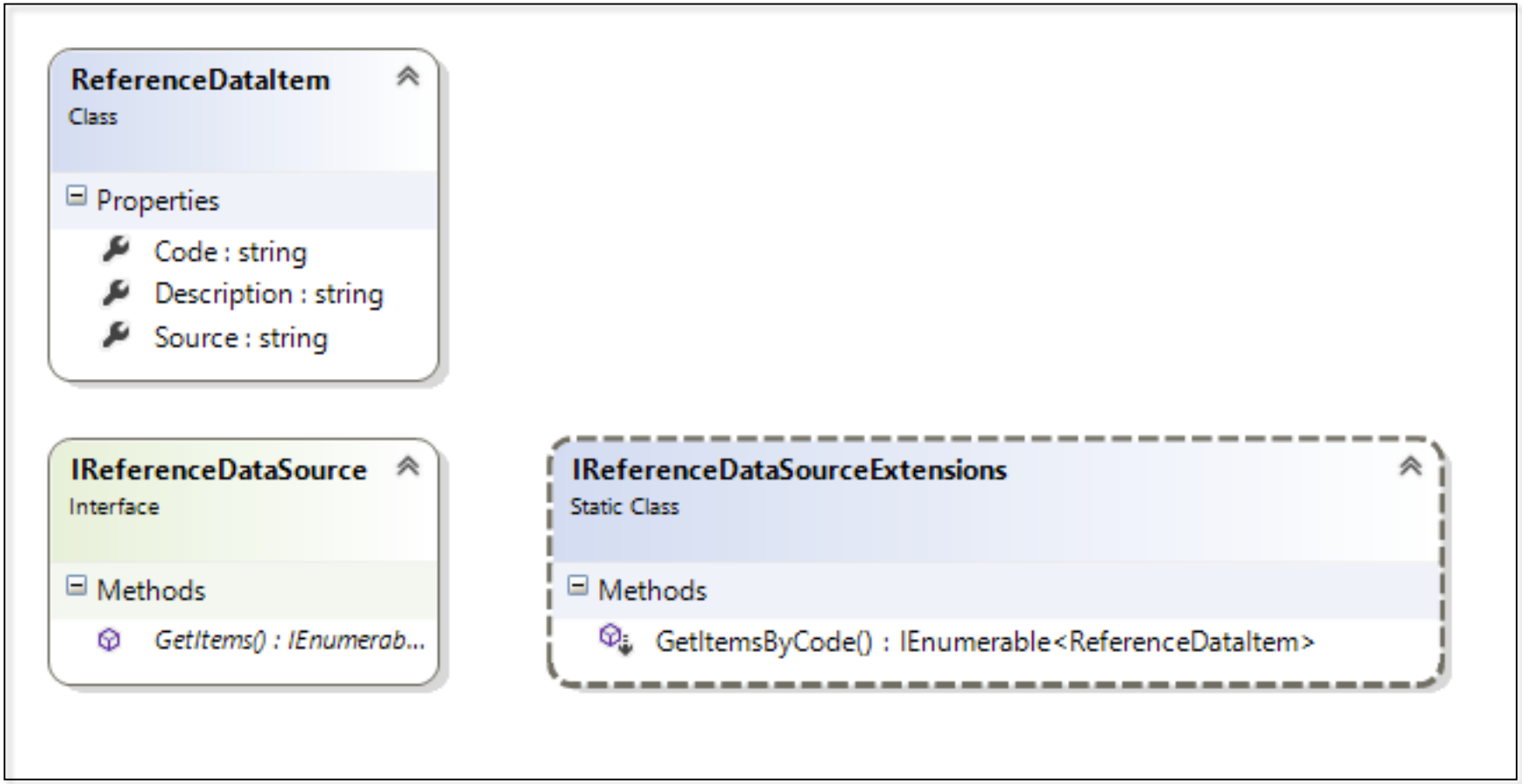
```
public static IEnumerable<ReferenceDataItem>
 GetItemsByCode(this IReferenceDataSource source, string code)
```

- **Using extension methods**
  - Apply to any interface implementation

```
 var items = new List<ReferenceDataItem>();
 var sqlSource = new SqlReferenceDataSource();
 items.AddRange(sqlSource.GetItemsByCode("xyz"));
 var xmlSource = new XmlReferenceDataSource();
 items.AddRange(xmlSource.GetItemsByCode("xyz"));
 var apiSource = new ApiReferenceDataSource();
 items.AddRange(apiSource.GetItemsByCode("xyz"));
```

Extending Collections

- Extensions over multiple objects



Demo 4: Extending Collections

Feature

Get aggregated reference data by type from multiple sources

Task

Extend array of reference data source interfaces

Task

Extend IEnumerable

Task

Extend IEnumerable<T>

Demo 4: Extending Collections

Demo 4: Extending Collections

- **Extending arrays**
  - Static class, static method
  - Array to extend in **this** argument

```
public static IEnumerable<ReferenceDataItem> GetAllItemsByCode
  (this IReferenceDataSource[] sources, string code) {}
```

- **Using extensions on arrays**
  - Apply to any implementation

```
var sqlSource = new SqlReferenceDataSource();
var xmlSource = new XmlReferenceDataSource();
var sources = new IReferenceDataSource[] { sqlSource, xmlSource };
var items = sources.GetAllItemsByCode("xyz");
```

Demo 4: Extending Collections

- **Extending collection interfaces**
  - Interface to extend in **this** argument

```
public static IEnumerable<ReferenceDataItem> GetAllItemsByCode
 (this IEnumerable sources, string code) {}
```

  - Implementation needs to cast entries

```
foreach (var source in sources)
{
  var refDataSource = source as IReferenceDataSource;
  if (refDataSource!= null) //etc.
```

Demo 4: Extending Collections

- **Using extensions on collection interfaces**
  - Apply to any interface implementation

```
var sqlSource = new SqlReferenceDataSource();
var xmlSource = new XmlReferenceDataSource();
var sources = new ArrayList();
sources.Add(sqlSource);
sources.Add(xmlSource);
sources.Add("i am not a reference data source");
sources.GetAllItemsByCode("xyz");
```

Demo 4: Extending Collections

- **Extending generic collection interfaces**
  - Static class, static method
  - Generic interface to extend in **this** argument

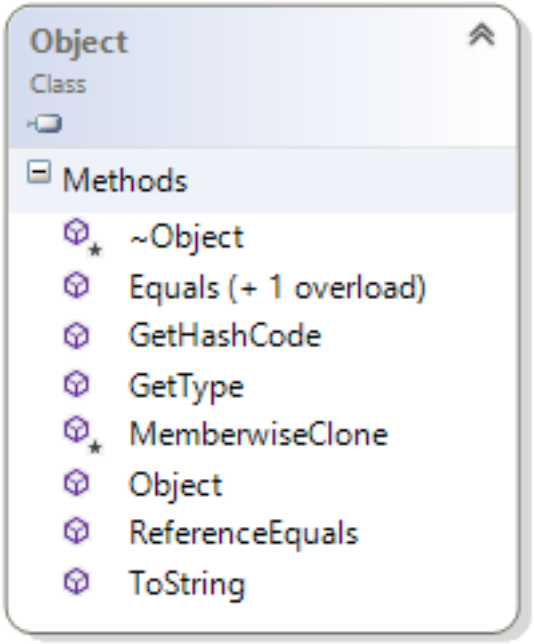
```
public static IEnumerable<ReferenceDataItem> GetAllItemsByCode
 (this IEnumerable<IReferenceDataSource> sources, string code) {}
```

- **Using extensions on generic collection interfaces**
  - Apply to any generic implementation

```
var s1 = new SqlReferenceDataSource();
var s2 = new XmlReferenceDataSource();
var list = new List<IReferenceDataSource> {s1, s2};
list.GetAllItemsByCode("xyz");
var hashSet = new HashSet<IReferenceDataSource> {s1, s2};
hashSet.GetAllItemsByCode("xyz");
```

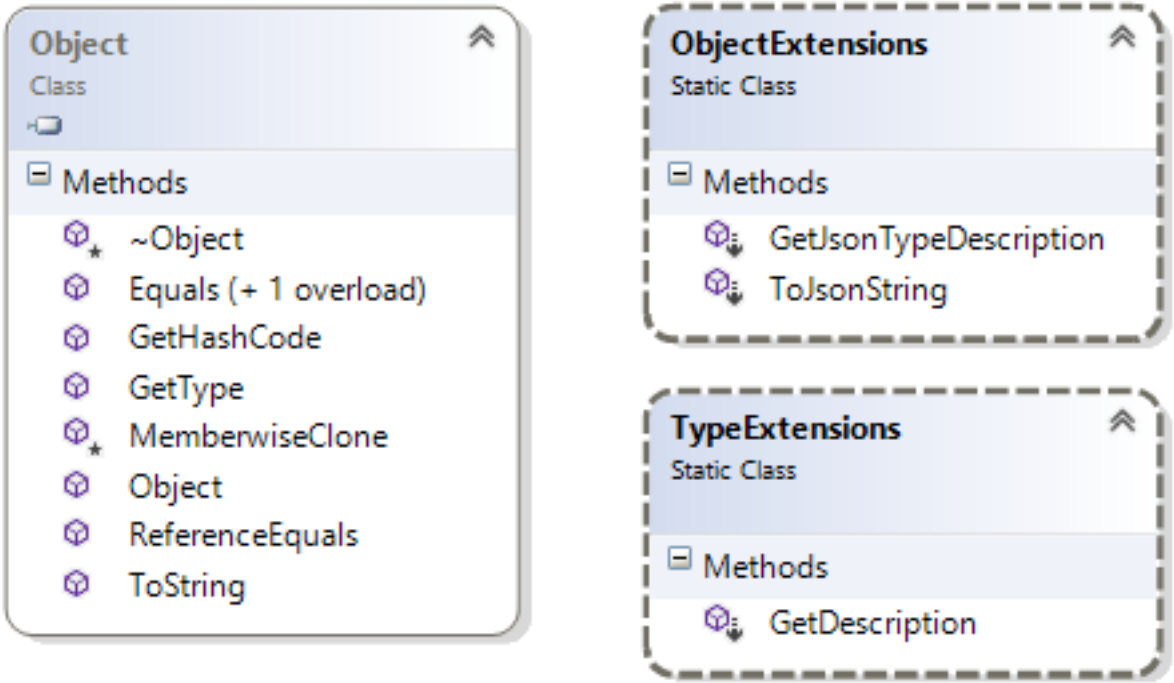
Extending Everything

- Extensions over any object



Extending Everything

- Extensions over any object



Demo 5: Extending Everything

Feature

Get formatted text version of any object for logging

Task

Extend Object to return a JSON representation

Task

Extend Type to return a structured description

Demo 5: Extending Everything

Demo 5: Extending Everything

- **Extending all objects**
  - Static class, static method
  - Extend **object** or **Type** in **this** argument

```
public static string ToJsonString(this object obj) {}
public static TypeDescription GetDescription(this Type type) {}
public static string GetJsonTypeDescription(this object obj) {}
```

- **Using extensions on objects**
  - Apply to anything

```
var obj1 = int.MaxValue;
Debug.WriteLine(obj1.ToJsonString());

var obj2 = new DateTime(2000, 12, 01);
Debug.WriteLine(obj2.ToJsonString());

var obj3 = new ReferenceDataItem { }; //etc.
Debug.WriteLine(obj3.ToJsonString());
```

Module Summary

- **How to write extension methods** ✓
  - Static class, static method
  - **this** keyword
- **How to extend** ✓
  - Classes & structs
  - Interfaces & collections
  - Object
- **Where to define extension methods** ✓
  - Any assembly, any class
  - Any namespace – including piggybacking
- **How to use extension methods** ✓
  - Add a reference to the assembly
  - Import the namespace
  - With IntelliSense

Key Scenarios ✓

Extending a
3rd-party codebase
*
Adding to a hierarchy
without inheritance
or composition
*
Adding aggregation
without collection classes
*
Extending every object