Advanced Interface Topics

WHERE TO GO NEXT

Jeremy Clark
AUTHOR TITLE

@authortwitter www.jeremybytes.com

Overview

Best Practices

Interface Segregation Principle

Choosing Between Abstract Class and Interface

Updating Interfaces

Overview

Advanced Topics

Dependency Injection

Mocking

Interface Segregation Principle

```
public class List<T> : IList<T>,
      ICollection<T>, IList, ICollection,
      IReadOnlyList<T>, IReadOnlyCollection<T>,
      IEnumerable<T>, IEnumerable
```

“Clients should not be forced to depend upon methods that they do not use. Interfaces belong to clients, not hierarchies”

Martin & Martin. *Agile Principles, Patterns, and Practices in C#*. Pearson Education, 2006

We should have granular interfaces that only include the members that a particular function needs.

List<T> Interfaces

```
public class List<T> : IList<T>,
          ICollection<T>, IList, ICollection,
          IReadOnlyList<T>, IReadOnlyCollection<T>,
          IEnumerable<T>, IEnumerable
```

IEnumerable	GetEnumerator()
IEnumerable<T>	GetEnumerator()

List<T> Interfaces

```
public class List<T> : IList<T>,
            ICollection<T>, IList, ICollection,
            IReadOnlyList<T>, IReadOnlyCollection<T>,
            IEnumerable<T>, IEnumerable
```

ICollection<T>

- Count
- IsReadOnly
- Add()
- Clear()
- Contains()
- CopyTo()
- Remove()

Plus, everything in

- IEnumerable<T>
- IEnumerable

List<T> Interfaces

```
public class List<T> : IList<T>,
          ICollection<T>, IList, ICollection,
          IReadOnlyList<T>, IReadOnlyCollection<T>,
          IEnumerable<T>, IEnumerable
```

IList<T>

- Item / Indexer
- IndexOf()
- Insert()
- RemoveAt()

Plus, everything in

- ICollection<T>
- IEnumerable<T>
- IEnumerable

Granular Interfaces

`IEnumerable<T>`

If We Need to

- Iterate over a Collection / Sequence
- Data Bind to a List Control
- Use LINQ functions

Granular Interfaces

`ICollection<T>`

If We Need to

- Add/Remove Items in a Collection
- Count Items in a Collection
- Clear a Collection

Granular Interfaces

`IList<T>`

If We Need to

- Control the Order Items in a Collection
- Get an Item by the Index

IEnumerable Implementations

List<T>

Array

ArrayList

SortedList<TKey, TValue>

HashTable

Queue / Queue<T>

Stack / Stack<T>

Dictionary<TKey, TValue>

ObservableCollection<T>

+ Custom Types

IEnumerable<T> Implementations

List<T>

Array

SortedList<TKey, TValue>

Queue<T>

Stack<T>

Dictionary<TKey, TValue>

ObservableCollection<T>

+ Custom Types

ICollection<T> Implementations

List<T>

SortedList<TKey, TValue>

Dictionary<TKey, TValue>

+ Custom Types

IList<T> Implementations

List<T>

+ Custom Types

Program at the Right Level

`IEnumerable<T>`

If We Need to

- Iterate over a Collection / Sequence
- Data Bind to a List Control

`ICollection<T>`

If We Need to

- Add/Remove Items in a Collection
- Count Items in a Collection
- Clear a Collection

`IList<T>`

If We Need to

- Control the Order Items in a Collection
- Get an Item by the Index

IPersonRepository

```
public interface IPersonRepository
{
    IEnumerable<Person> GetPeople();

    Person GetPerson(string lastName);

    void AddPerson(Person newPerson);

    void UpdatePerson(string lastName, Person updatedPerson);

    void DeletePerson(string lastName);

    void UpdatePeople(IEnumerable<Person> updatedPeople);
}
```

Better Segregation

```
public interface IPersonRepository
{
    IEnumerable<Person> GetPeople();

    Person GetPerson(string lastName);
}
```

Better Segregation

```
public interface IPersonRepository
{
    void AddPerson(Person newPerson);

    void UpdatePerson(string lastName, Person updatedPerson);

    void DeletePerson(string lastName);

    void UpdatePeople(IEnumerable<Person> updatedPeople);
}
```

Comparison Summary

	Abstract Classes	Interfaces	
👍	May contain implementation code	May not contain implementation code	👎
👎	A class may inherit from a single base class	A class may implement any number of interfaces	👍
	Members have access modifiers	Members are automatically public	
	May contain fields, properties, constructors, destructors, methods, events and indexers	May only contain properties, methods, events, and indexers	

Regular Polygon

```
public abstract class AbstractRegularPolygon
{
    public int NumberOfSides { get; set; }
    public int SideLength { get; set; }
    public AbstractRegularPolygon(int sides, int length)
    {
        NumberOfSides = sides;
        SideLength = length;
    }
    public double GetPerimeter()
    {
        return NumberOfSides * SideLength;
    }
    public abstract double GetArea();
}
```

Abstract Class

Lots of Shared Code

Person Repository

CSV Repository

```
public IEnumerable<Person> GetPeople()
{
    var people = new List<Person>();
    if (File.Exists(path))
        using (var sr = new StreamReader(path))
        {
          string line;
          while ((line = sr.ReadLine()) != null)...
              people.Add(per);
        }
          }
          return people;
      }
```

Person Repository

SQL Repository

```
public IEnumerable<Person> GetPeople()
{
    using (var ctx = new PeopleEntities())
    {
        var people = from p in ctx.DataPersons
                     select new Person...

          return people.ToList();
      }
 }
```

Person Repository

Service Repository

```
public IEnumerable<Person> GetPeople()
{
     return serviceProxy.GetPeople();
}
```

Interface

No Shared Implementation Code

Interfaces & Abstract Classes in the .NET BCL

Abstract Classes with Shared Implementation

MembershipProvider, RoleProvider

CollectionBase

Interfaces & Abstract Classes in the .NET BCL

Interfaces to Add Pieces of Functionality

IDisposable

INotifyPropertyChanged, INotifyCollectionChanged

IEquatable<T>, IComparable<T>

IObservable<T>

IQueryable<T>, IEnumerable<T>

Interfaces & Abstract Classes in the .NET BCL

Base Classes that Implement Interfaces / Inherit from Abstract Classes

SqlMembershipProvider

SqlConnection, OdbcConnection, EntityConnection

List<T>, ObservableCollection<T>

Updating Interfaces

Interfaces are a Contract

- No changes after Contract is signed

Adding Members Breaks Implementation

Removing Members Breaks Usage

Inheritance is a Good Way to Add to an Interface

Adding Members with Inheritance

```
public interface ISaveable
{
    string Save();
}
```

```
public interface ISaveable
{
    string Save();
    string Save(string name);
}
```

Breaks Existing Implementers

```
public interface INamedSaveable :
    ISaveable
{
    string Save(string name);
}
```

Existing ISaveable Still Works

Dependency Injection

Loosely Coupled Code

Make “Something Else” Responsible for Dependent Objects

Design Patterns

- Constructor Injection
- Property Injection
- Method Injection
- Service Locator

Dependency Injection Containers

- Unity, StructureMap, Autofac, Ninject, Castle Windsor, and many others

Mocking

Create "Placeholder" Objects

- In-Memory
- Only Implement Behavior we care about

Great for Unit Testing

Mocking Frameworks

- RhinoMocks
- Microsoft Fakes
- Moq

Why Interfaces?

Maintainable

Extensible

Easily testable

Goals

Learn the ‘Why”

- Maintainability
- Extensibility

Implement Interfaces

- .NET Framework Interfaces
- Custom Interfaces

Goals

Create Interfaces

- Add Abstraction

Peek at Advanced Topics

- Mocking
- Unit Testing
- Dependency Injection

Summary

Contract

The “What” of Interfaces

Summary

Best Practice

Program to an abstraction rather than a concrete type

Summary

Program to an interface rather than a concrete class

Summary

Create Maintainable Code

Summary

Create & Implement a Custom Interface

- Use Abstraction to add Extensibility

Summary

Dynamic Loading & Unit Testing

- Fake Repository for Testability

Summary

Advanced Topics

Interface Segregation Principle

Dependency Injection

Mocking

Further Courses:

- Data Dependency Injection
- Solid Design Principles
- Model View / View Model Pattern
- Unit Testing
- Test-Driven Development
- Mocking