Default Implementation & Advanced Topics

Jeremy Clark
Developer Betterer

@jeremybytes www.jeremybytes.com

How

Updating interfaces

Default interface implementation

Interface inheritance

Other interface features

- Access modifiers
- Static members
- Plus, a few others

Interfaces vs. abstract classes

An interface is a contract

Adding Members Breaks Implementers

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

```
public class Catalog : ISaveable
{
  public void Save()
  {
    Console.Write("Saved (catalog)");
  }
}
```

Adding Members Breaks Implementers

```
public interface ISaveable {
  void Save();
  void Save(string message); // Added Member
}
```

```
public class Catalog : ISaveable
{
  public void Save()
  {
    Console.Write("Saved (catalog)");
  }
}
*** ERROR Save(string) is missing ***
```

Removing Members Breaks Clients

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

```
public class InventoryItem
{

  ISaveable saver = new SQLSaver();
  saver.Save("Added inventory");

}
```

Removing Members Breaks Clients

```
public interface ISaveable {
  void Save();
  // void Save(string message) REMOVED
}
```

```
public class InventoryItem
{

  ISaveable saver = new SQLSaver();
  saver.Save("Added inventory"); *** ERROR ***

}
```

An interface is a contract

Adding Interface Members

Default Implementation

Interface Inheritance

Default Implementation

An interface may include an implementation (default) for some or all of its members.

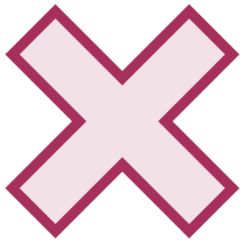
A class may provide its own implementation that will be used instead of the default.

If a class does not provide its own implementation, the default will be used.

Availability of Default Implementation

C# 8 (and later)

.NET 5.0 (and later)
.NET Core 3.0 & 3.1
.NET Standard 2.1



.NET Framework (all versions)
.NET Core 2.2 (and earlier)
.NET Standard 2.0 (and earlier)

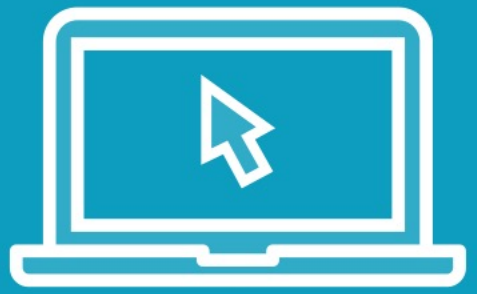
Existing Interface

```
interface ILogger
{
    void Log(LogLevel level, string message);
}

class ConsoleLogger : ILogger
{
    public void Log(LogLevel level, string message) { ... }
}
```

Default Implementation

```
interface ILogger
{
    void Log(LogLevel level, string message);

    void LogException(Exception ex) {           // New method
        Log(LogLevel.Error, ex.ToString());
    }
}

class ConsoleLogger : ILogger
{
    public void Log(LogLevel level, string message) { ... }
    // LogException(ex) uses default implementation
}
```

Demo



Change an existing interface with default implementation

Provide an implementation in the class that replaces the default

Calling an interface member with default implementation is similar to calling an interface member that has been explicitly implemented.

Default Implementation

```
interface ILogger
{
    void Log(LogLevel level, string message);

    void LogException(Exception ex) {           // New method
        Log(LogLevel.Error, ex.ToString());
    }
}

class ConsoleLogger : ILogger
{
    public void Log(LogLevel level, string message) { ... }
    // LogException(ex) uses default implementation
}
```

```
ILogger logger = new ConsoleLogger();
logger.LogException(ex);
// Exception logged successfully
```

◀ Interface type

```
ConsoleLogger consoleLogger = new ConsoleLogger();
consoleLogger.LogException(ex);
// COMPILER ERROR: default implementation not accessible
```

◀ Concrete type

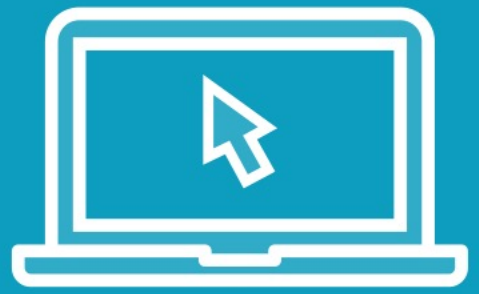
```
var varLogger = new ConsoleLogger();
varLogger.LogException(ex);
// COMPILER ERROR: default implementation not visible
```

◀ Concrete type
(same as “ConsoleLogger”)

```
((ILogger)consoleLogger).LogException(ex);
// Exception logged successfully
```

◀ Interface type

Demo



Calling an interface member with default implementation

Beware of making assumptions about how classes will implement the interface

Bad Assumption: Unintended Behavior

```
public interface IPeopleLogger
{
    public void Log(PeopleLogLevel level, string message);

    public void LogException(Exception ex)
    {
        Console.WriteLine($"{this.GetType()} - Exception: {ex.Message}");
    }
}
```

Assumes that all loggers will use the console

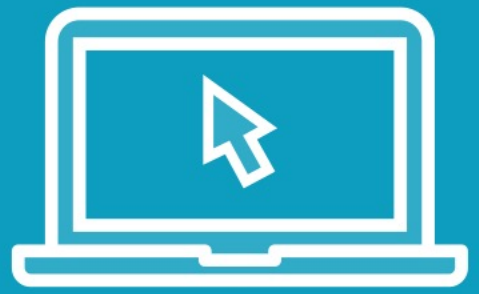
Advice: Use Existing Members

```
public interface IPeopleLogger
{
    public void Log(PeopleLogLevel level, string message);

    public void LogException(Exception ex)
    {
        Log(PeopleLogLevel.Error, ex.Message);
    }
}
```

Limited Usefulness

```
public interface IPersonRepository
{
    IReadOnlyCollection<Person> GetPeople();
    Person GetPerson(int id);

    void SavePerson(Person person)
    {
        // ???
    }
}
```

Demo



Making bad assumptions

Interface Inheritance

When an interface inherits another interface, the interface includes all members that it defines as well as all of the members in the parent interface.

```
public interface IEnumerable<T> : IEnumerable
```

Interface Inheritance

IEnumerable<T> includes all members from IEnumerable

Read-only Repository (Data Reader)

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

    Person GetPerson(int id);
}
```

Read-write Repository

```
public interface IPersonRepository : IPersonReader
{
    void AddPerson(Person newPerson);
    void UpdatePerson(int id, Person updated);
    void DeletePerson(int id);
}
```

Implementing Inherited Interfaces

```
public interface IPersonReader
{
  IEnumerable<Person> GetPeople();
  Person GetPerson(int id);
}

public interface IPersonRepository :
  IPersonReader
{
  void AddPerson(...);
  void UpdatePerson(...);
  void DeletePerson(...);
}
```

```
public class SQLRepository: IPersonRepository
{
    IEnumerable<Person> GetPeople(){
        // Implementation here
    }
    Person GetPerson(int id) {
        // Implementation here
    }
    void AddPerson(Person newPerson) {
        // Implementation here
    }
    void UpdatePerson(int id, Person updated) {
        // Implementation here
    }
    void DeletePerson(int id) {
        // Implementation here
    }
}
```

Using interface inheritance keeps our interfaces focused

Focused Implementation

```
public class ServiceReader: IPersonReader
{
    IEnumerable<Person> GetPeople(){...}
    Person GetPerson(int id) {...}
}


public class SQLRepository: IPersonReader
{
    IEnumerable<Person> GetPeople(){...}
    Person GetPerson(int id) {...}
}
```

Focused Implementation

```
public class ServiceReader: IPersonReader
{
    IEnumerable<Person> GetPeople(){...}
    Person GetPerson(int id) {...}
}


public class SQLRepository: IPersonRepository
{
    IEnumerable<Person> GetPeople(){...}
    Person GetPerson(int id) {...}
    void AddPerson(Person newPerson) {...}
    void UpdatePerson(int id, Person updated) {...}
    void DeletePerson(int id) {...}
}
```

Properties and Default Implementation

Default implementation can be used for properties,
but it is really only useful for read-only or calculated properties.

Automatic Properties

```
public class ConcreteRegularPolygon
{
    public int NumberOfSides { get; set; }
    public int SideLength { get; set; }
    //...
}
```

Automatic Properties

```
public interface IRegularPolygon
{
    int NumberOfSides { get; set; }
    int SideLength { get; set; }
    //...
}
```

Property Declarations

There is no way to use default implementation to create automatic properties in an interface

Interface Member Access Modifiers

public
- Default
- Visible to everyone

private
- Only visible in the interface
- Must have implementation

protected / internal
- Special cases

Further Study

Default implementation of properties
- Useful for calculated properties
- Cannot be used for automatic properties

Access modifiers
- public
- private
- protected
- internal

Static members
- Fields
- Methods
- Constructors
- Destructors

Additional Resources

https://bit.ly/3tYeAee

https://github.com/jeremybytes/csharp-interfaces-resources

Comparing Interfaces and Abstract Classes

Interface	Abstract Class
Defines a contract	Shared implementation
Implement any number of interfaces	Inherit from a single base class
Limited implementation code	Unconstrained implementation code
No automatic properties	Can have automatic properties
Properties Methods Events Indexers	Properties Methods Events Indexers Fields Constructors Destructors

How much implementation code is shared?

```
// Polygon

public int NumberOfSides {…}
public int SideLength {…}
public double GetPerimeter()
public double GetArea()
```

◀ Shared

◀ Shared

◀ Shared

◀ Not shared

Abstract Class

Data Readers

```
public IEnumerable<Person> GetPeople() {
    var address = $"{baseUri}/people";
    string reply = client.DownloadString(address);
    return JsonSerializer.Deserialize<List<Person>>(reply, options);
}

public IEnumerable<Person> GetPeople() {
    var fileData = FileLoader.LoadFile();
    var people = ParseData(fileData);
    return people;
}

public IEnumerable<Person> GetPeople() {
    using var context = new PersonContext(options);
    return context.People!.ToList();
}
```

No shared code

Interface

How

Updating interfaces

Default interface implementation

Interface inheritance

Other interface features

- Access modifiers
- Static members
- Plus, a few others

Interfaces vs. abstract classes