IEnumerable, ISaveable, IDontGetIt

Understanding C# Interfaces

Jeremy Clark

jeremybytes.com

github.com/jeremybytes/sdd-2024

An interface contains definitions for a group of related functionalities that a non-abstract class or struct must implement.

https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/interfaces

An interface describes
a set of capabilities
on an object.

“I have these functions.”

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	Fields
Methods	Constructors
Events	Destructors
Indexers	

Recommendation

Program to an abstraction rather than a concrete type.

Recommendation

Program to an interface rather than a concrete class.

Various Data Sources

Microsoft SQL Server

MongoDB

CSV

WebAPI

Amazon RDS

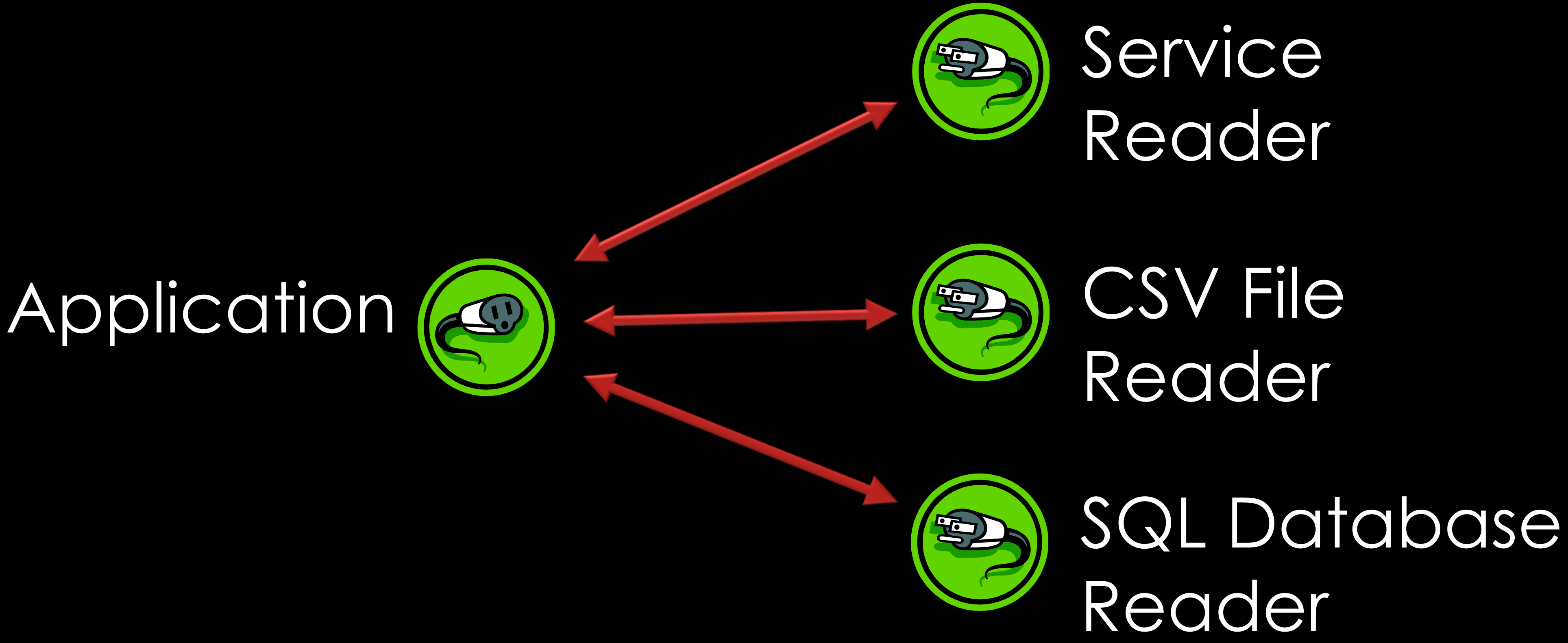
Oracle

JSON

Hadoop

Azure Cosmos DB

Pluggable Data Readers



Data Reader Interface

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

Interfaces and Flexible Code

Resilience in the face of change

Insulation from implementation details

Dynamic Factory

- Check configuration
- Create an assembly load context
- Load the assembly
- Look for the type
- Create the data reader
- Return the data reader

Other Benefits

Interfaces help us isolate code for easier unit testing.

Other Benefits

Interfaces can make dependency injection easier.

Interface Segregation Principle

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

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

Interface Segregation Principle

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

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

Interface Inheritance

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

- IEnumerable<T> inherits IEnumerable
- When a class implements IEnumerable<T>, it must also implement IEnumerable

IEnumerable<T> / IEnumerable

```
public interface IEnumerable<T> : IEnumerable
{
    IEnumerator<T> GetEnumerator();
}
public interface IEnumerable
{
    IEnumerator GetEnumerator();
}
```

When a class implements IEnumerable<T>,
it must also implement IEnumerable

List<T> Interfaces

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

IEnumerable<T>

GetEnumerator()

IEnumerable

GetEnumerator()

List<T> Interfaces

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

ICollection<T>

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

Plus
Everything in
IEnumerable<T>
and
IEnumerable

List<T> Interfaces

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

IList<T>

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

Plus
Everything in
ICollection<T>,
IEnumerable<T>,
and
IEnumerable

Granular Interfaces

- **If We Need to**
  - Iterate over a Collection / Sequence
  - Data Bind to a List Control
  - Use LINQ functions

→ **IEnumerable<T>**

- **If We Need To**
  - Add/Remove Items in a Collection
  - Count Items in a Collection
  - Clear a Collection

→ **ICollection<T>**

- **If We Need To**
  - Control the Order Items in a Collection
  - Get an Item by the Index

→ **IList<T>**

Summary

- An interface describes a set of capabilities of an object.
- Program to an abstraction (interface) rather than a concrete type (class).
- Resilience in the face of change.
- Insulation from implementation details.
- Easier unit testing.
- Easier dependency injection.

Thank You!

Jeremy Clark

- jeremybytes.com
- youtube.com/jeremybytes

- github.com/jeremybytes/sdd-2024