**Interface Segregation Principle**

The ‘I’ in SOLID design.

“Clients should not be forced to depend upon methods that they do not use. Interfaces belong to clients, not hierarchies” – we should have granular interfaces that only include the members that a particular function needs.

IList<T> – an indexer (allows insertion of data at specific points).

By implementing interfaces that inherit from other interfaces you’re inheriting from then you future proof the code as a class can be coded to either of the interfaces and allowed to be assigned to either type of variable. This adds coding flexibility. Useful with the collection type that might want different levels of functionality (indexing, iteration).

Interface vs Abstract Class

Shared code should become an abstract class.

No shared code = interface.

**MembershipProvider & RoleProvider** - .NET classes for access rights. Have shared implementation from a parent class.

**CollectioinBase** – good abstract class for making strongly typed collections.

**IDisposable** – implemented to free unmanaged resources

**INotifyPropertyChanged**, **INotifyCollectionChanged** – implemented to let the UI know the underlying object has been changed.

**IEquatable**<T>, **IComparable**<T> - give us methods to check if objects can be considered equal.

**IObservable**<T> - very basic way to give us publish/subscribe pattern.

**IQueryable**<T>, **IEnumerable**<T> - give us access to a variety of link functions.

**Updating Existing Interfaces**

Interfaces are contract. Once it’s made, it can’t be changed. Adding members from an interface could break any class that implements the interface. Removing risks breaking the functionality of existing code that might rely on the member.

Inheritance is a good way to add new members to an existing interface. Directly inherit from existing interface and add the new member. This won’t break existing implementers.

**Dependency Injection**

The idea is to create loosely coupled code. Come back to it…..

**Mocking**

The idea is to create place holder objects where we don’t implement an entire class but only the methods we plan on using. Exists in-memory only

Great for unit testing. Examples include RhinoMocks, Microsoft Fakes, Moq

Basically uses a framework to create an object that can be told what to do so it can implement the functionality being tested.

**What do we want from our code?**

Maintainable, Extensible, easily testable.

Interfaces are a good way.

Interface – collection of public members that form a contract.

Program to abstraction rather than concrete type.