Why use them?

Future proofing our code.

Making the code maintainable so that when changes need to be made to a part of the application, we minimise the changes to other parts of the application. Interfaces are a big part of it with best practices.

Best practice – **program to an abstraction rather than to a concrete type.**

Program to an interface rather than a concrete type. Program to the contract, don’t worry about implementation yet.

List<T> - strongly typed collection of objects.

FIFO - QUEUE

LIFO – STACK

Interface segregation principle –

IEnumerable contains only one member; getEnumerator. This specifies the ability to iterate over a collection: Foreach, list boxes.

CRUD – Create, read, update, delete.

If our variable is an interface type, we don’t know the objects type, we just know it implements the interface contract. Therefore we are coding to the contract.

Why use interface over concrete?

Change may affect concrete classes and their use as local variables. Using an interface doesn’t care about the class that’s coming back, all it cares about is the contract.

This is future proofing by coding to an abstraction.

GetEnumerator() – method from the IEnumerable interface.

IEnumerable – generic version. Objects will be considered of type object. This means a variable of type IEnumerable will only have access to members of IEnumerable.

IEnumerable<Person> - specific version. Objects will eb considered of type Person. This means members of type Person will be exposed.

Use an interface variable if you only care about the interface methods.

Interfaces give resilience to change.