Collection classes implement interfaces. We can work with the interface of the collection to make more generalised code.

You CAN use an interface to define a property.

Interfaces can be generic. i.e. have generic param types.

Static methods can be appropriate when you wish to perform an operation on a collection of an instance. Define the static method in the class of the instances you want to act on.

IEnumerable<T> - gives a collection its ability to be enumerated using the foreach statement. It specifies one method, GetEnumerator(). This is not called directly, rather we use the foreach statement to iterate over the collection. This is the base interface for all generic collections.

ICollection<T> - extends IEnumerable of T. It specifies a property and a set of methods for basic collection operations; Count, Add, Remove

IList<T> - extends ICollection of T. It specifies methods for working with Lists by Index; IndexOf, Insert, RemoveAt…

IDictionary<T, V> - extends ICollection of T. It specifies properties and methods for dealing with collections of Key, Value pairs; Keys, Values, TryGetValue, ContainsKey…

Array implements… IEnumerable<T>, ICollection<T> and IList<T>(but in a non-standard way). From .NET 2 Array was modified to implemented the generic interfaces. However, these interfaces don’t appear in the Arrays interface so we cannot use these methods or properties directly.

IList of T implements… the same as array but is able to use the methods and properties provided.

IDictionary implements…. IEnumerable<T>, ICollection<T> and IDictionary<T> and can use all the methods and properties provided.

Interface as param best practices:

Do:

Consider interfaces rather than concrete types when passing collections.

Use a cast operation to cast return types to the desired collection type.

Use the most general interface to cover all considered types.

Consider returning IEnumerable<T> for an immutable collection.

Consider returning IEnumerable<T> if the use cases are unknown. The calling code can then decide on the use.

Use an iterator when returning 1 element at a time.

Use an iterator for deferred execution. It can greatly improve performance with large data sets.

Avoid:

Using any collections in System.Collections namespace.

Using IEnumerable<T> unless the only thing you’re doing is iterating through.

Returning IEnumerable<T> if the caller needs to edit the collection

Returning IEnumerable <T> if the caller needs additional info like Count.

When casting ICollection to Dictionary<T> the following syntax is used:

var dictionary = collection.ToDictionary(v=>v.ID);

Uses a lambda expression. It means for each element(v) use its company name as the key.

ICollection<T> can cast to non-generic ICollection types.

Returning IEnumerable<T> is a promise to return a sequence of typed elements. It also makes the collection read-only as it can only be iterated through.

To return one element at a time we define an iterator with a yield return statement.

The return type of an iterator must be IEnumerable<T> where T is the specific type.

When the **yield return** code is executed the place in the code is remembered, the method is returned to the calling code and processing is continued there. The next time the yield return method is called, the last point is remembered and continued from the save point. The iterator is consumed by a foreach statement or with a LINQ(Language Integrated Query) statement. When a yield return method is used, the code is only executed when used in a foreach loop and the first element accessed. This is called **Deferred execution**.

Deferred execution using yield return means execution is deferred until it is iterated.

Lazy evaluation – 1 element returned at a time.

FAQs

1. What is an interface?

A specification for a related set of properties and methods.

1. What does it mean when a class implements an interface?

It commits to supporting the interface by providing a concrete definition of the interface methods and properties.

1. What is the key benefit of using an interface as a data type?

We can define generalised code for properties, params and return values.

1. What does the IEnumerable<T> interface provide?

Method GetEnumerator() and the ability to iterate through a collection.

1. What does the ICollection<T> interface provide?

The ability to work with a collection.

1. What does the IList<T> interface provide?

Work with indexing. Arrays and Lists implement this interface.