**Collection Interfaces in C#**

**1. IEnumerable**

**Description**

The IEnumerable interface is the base for all non-generic collections. It allows iteration over a collection using a foreach loop.

**Use Cases**

* When you only need to iterate over a collection without modifying it.
* Useful for LINQ queries since it provides deferred execution.

**Example**

using System;

using System.Collections;

class Program

{

static void Main()

{

ArrayList numbers = new ArrayList { 1, 2, 3, 4, 5 };

foreach (var number in numbers)

{

Console.WriteLine(number);

}

}

}

**2. ICollection**

**Description**

Extends IEnumerable and provides additional methods like Add, Remove, and Count.

**Use Cases**

* When you need to add, remove, or count items in a collection.
* Suitable for managing basic collection operations.

**Example**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

ICollection<int> numbers = new List<int> { 1, 2, 3 };

numbers.Add(4);

numbers.Remove(2);

Console.WriteLine("Count: " + numbers.Count);

}

}

**3. IList**

**Description**

Extends ICollection and adds methods for accessing elements by index, such as Insert and RemoveAt.

**Use Cases**

* When you need indexed access to elements or need to maintain insertion order.

**Example**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

IList<string> fruits = new List<string> { "Apple", "Banana" };

fruits.Insert(1, "Orange");

fruits.RemoveAt(0);

Console.WriteLine(string.Join(", ", fruits));

}

}

**4. IDictionary**

**Description**

Represents a collection of key-value pairs.

**Use Cases**

* When you need to store and retrieve data using keys.
* Useful for scenarios like caching or lookup tables.

**Example**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

IDictionary<int, string> employees = new Dictionary<int, string>

{

{ 1, "John" },

{ 2, "Doe" }

};

employees.Add(3, "Alice");

Console.WriteLine("Employee ID 2: " + employees[2]);

}

}

**5. ISet**

**Description**

Represents a collection of unique elements. It does not allow duplicate items.

**Use Cases**

* When you need to maintain a collection with no duplicates.
* Suitable for operations like union, intersection, and difference.

**Example**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

ISet<int> numbers = new HashSet<int> { 1, 2, 3 };

numbers.Add(3); // Won't be added as it's a duplicate

numbers.Add(4);

Console.WriteLine(string.Join(", ", numbers));

}

}

**Key Differences**

| **Interface** | **Key Features** | **Indexed Access** | **Allows Duplicates** | **Key-Value Pairs** |
| --- | --- | --- | --- | --- |
| IEnumerable | Iteration only | No | Yes | No |
| ICollection | Add, Remove, Count | No | Yes | No |
| IList | Indexed access, insertion/removal | Yes | Yes | No |
| IDictionary | Key-value pairs | No | No (keys) | Yes |
| ISet | Unique elements | No | No | No |