**Interfaces and Types in .NET**

**1. ICor (Typo? Possibly intended to reference a specific type or namespace)**

* No direct reference to ICor exists in .NET. Please verify the intended term. You may be referring to ICorRuntimeHost (used in the .NET Framework) or another custom interface.
* If you meant something specific, please clarify for detailed information.

**2. int i**

* **Type**: int is a value type that represents a 32-bit signed integer.
* **Usage**:

int i = 42;

Console.WriteLine(i); // Output: 42

* **Common Operations**: Arithmetic, comparison, and bitwise operations.

**3. object i**

* **Type**: object is the base type from which all types derive in .NET.
* **Usage**:

object i = 42; // Boxing int into object

Console.WriteLine(i); // Output: 42

* **Characteristics**: Any type can be assigned to object because all types inherit from it.

**4. IEnumerable**

* **Namespace**: System.Collections
* **Definition**: Represents a collection of objects that can be iterated over.
* **Key Members**:
  + IEnumerator GetEnumerator()
* **Usage**:

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

foreach (int number in numbers)

{

Console.WriteLine(number);

}

* **Use Case**: Provides forward-only, read-only access to a collection.

**5. IQueryable**

* **Namespace**: System.Linq
* **Definition**: Extends IEnumerable to support LINQ queries that are executed against a data source.
* **Key Members**:
  + Expression Expression { get; }
  + IQueryProvider Provider { get; }
* **Usage**:

IQueryable<int> query = context.Numbers.Where(n => n > 10);

foreach (int number in query)

{

Console.WriteLine(number);

}

* **Use Case**: Enables deferred execution and efficient querying of data.

**6. ICollection**

* **Namespace**: System.Collections
* **Definition**: Represents a collection of objects that can be individually accessed.
* **Key Members**:
  + int Count { get; }
  + bool IsReadOnly { get; }
  + void Add(T item);
  + void Remove(T item);
* **Usage**:

ICollection<int> numbers = new List<int>();

numbers.Add(1);

numbers.Add(2);

Console.WriteLine(numbers.Count); // Output: 2

* **Use Case**: Adds, removes, or checks objects in the collection.

**7. IComparable**

* **Namespace**: System
* **Definition**: Defines a generalized comparison method for objects.
* **Key Members**:
  + int CompareTo(object other);
* **Usage**:

public class Person : IComparable

{

public string Name { get; set; }

public int Age { get; set; }

public int CompareTo(object obj)

{

if (obj == null) return 1;

Person other = obj as Person;

return this.Age.CompareTo(other.Age);

}

}

* **Use Case**: Used for sorting and comparisons in collections like List<T>.

**Summary Table**

| **Interface/Type** | **Namespace** | **Key Features** | **Example Use Case** |
| --- | --- | --- | --- |
| int | - | 32-bit signed integer | Numeric operations |
| object | System | Base type for all types | Polymorphism |
| IEnumerable | System.Collections | Iteration over a collection | Read-only traversal |
| IQueryable | System.Linq | LINQ queries with deferred execution | Efficient database queries |
| ICollection | System.Collections | Add, remove, and check items in a collection | Managing modifiable collections |
| IComparable | System | Defines object comparison | Sorting and ordering |