1. LINQ Last Method in C#:

The LINQ Last Method in C# returns the last element from a data source or a collection. If the data source or collection is empty, or if we specified a condition and with that condition, no matching element is found in the data source. The LINQ Last method will throw an InvalidOperationException. If the Data Source is Null, then it will throw ArgumentNullException. Two overloaded versions are available for the LINQ Last method, as shown in the image below.

As you can see, the first overloaded version of the Last method does not take any parameter. It simply returns the last element from the data source, and if it is empty, it will throw an InvalidOperationException.

The second overloaded version of the Last method takes a predicate as a parameter. Using this predicate, we can specify a condition, and then it returns the last matching element that satisfies the specified condition. In this case, if no element satisfies the specified condition, it will throw InvalidOperationException.

1. Purpose, Behavior, and Usage of LINQ Last Method:

Purpose: Returns the last element of a sequence.

Behavior: If the sequence contains no elements, Last throws an InvalidOperationException.

Usage: Use Last when you are confident that the sequence contains at least one element and that an empty sequence represents an exceptional scenario in the context of your application.

This is Like First and FirstOrDefault

When to Use Last and LastOrDefault in LINQ?

Use the Last Method in LINQ

Use Case: When you are certain the sequence contains at least one element, and an empty sequence is considered exceptional.

Appropriate Scenario: After filtering a sequence where at least one element is expected to satisfy the condition, such as retrieving the most recent transaction from a list of transactions.

Example: var latestTransaction = transactions.Where(t => t.Date < DateTime.Now).Last();

Use the LastOrDefault Method in LINQ

Use Case: When the sequence may be empty, and you want to handle that case without an exception, usually by checking if the returned value is the default for the type.

Appropriate Scenario: When querying data that may not exist, such as looking for a record that matches a condition that might not be met, you need to handle a ‘not found’ case gracefully.

Example: var lastSale = sales.LastOrDefault(s => s.Date < endOfFiscalYear);

When Not to Use Them

Performance Consideration: On non-indexed sequences (e.g., linked lists, non-list IEnumerable), using Last or LastOrDefault can be inefficient because the entire sequence must be enumerated to find the last element. In such cases, if you frequently access the last element, consider using a data structure with more efficient end access, like an array or a list.

Alternative Approach: If you’re working with a sequence that allows for efficient indexing from the end (such as an array or a List<T>), and you want to avoid the potential cost of enumerating the sequence, you might directly index the last element (e.g., myList[myList.Count – 1]).

Important Usage Notes:

Use LastOrDefault to avoid exceptions when working with potentially empty sequences. The default value (null for reference types, 0 for numeric types, false for bool, etc.) can be checked to determine if the sequence was empty.

Always ensure these methods align with your data’s expected size and structure. If an operation assumes a non-empty sequence, but the sequence can be empty, this could lead to logical errors in your program.

For performance-critical code, avoid using Last or LastOrDefault on large sequences that are not inherently indexed.

What is the Difference Between the Last and LastOrDefault Methods in LINQ?

Both Last and LastOrDefault methods in LINQ are used to return the last element from a data source. The Last and LastOrDefault methods in LINQ are used to retrieve the last element of a sequence, but they differ in how they handle sequences that contain no elements:

Last

Behavior: It returns the last element of a sequence.

Exception Handling: If the sequence contains no elements, Last will throw an InvalidOperationException.

Usage Consideration: Use Last when you are certain that the sequence will contain at least one element, and an empty sequence would be considered an exceptional case in the logic of your application.

LastOrDefault

Behavior: It returns the last element of a sequence or the default value for the type if the sequence contains no elements.

Exception Handling: LastOrDefault does not throw an exception for an empty sequence; it returns the default value instead (e.g., null for reference types, 0 for integers).

Usage Consideration: Use LastOrDefault when the sequence might be empty, and you want to handle this case without the overhead of exception handling. It allows for a more graceful degradation when dealing with potentially empty sequences.

So, choose between Last and LastOrDefault based on your expectations regarding the presence of a matching element and your preference for handling exceptional or empty sequence situations. Last is more suitable when you expect a match and want to handle exceptions, while LastOrDefault is useful when you want to handle default values gracefully when no match is found, or the sequence is empty.