1. What are LINQ Element Methods or Operators?

The LINQ Element Methods or Operators are used to return a single element from a data source using the element’s index or a predicate, i.e., a condition. These Element Operators can be used with a single data source or on a query of multiple data sources.

In LINQ, the ElementAt and ElementAtOrDefault methods are used to retrieve an element from a specific index in a sequence. They behave similarly to array indexing but can be used with any type that implements IEnumerable<T>.

1. LINQ ElementAt Method:

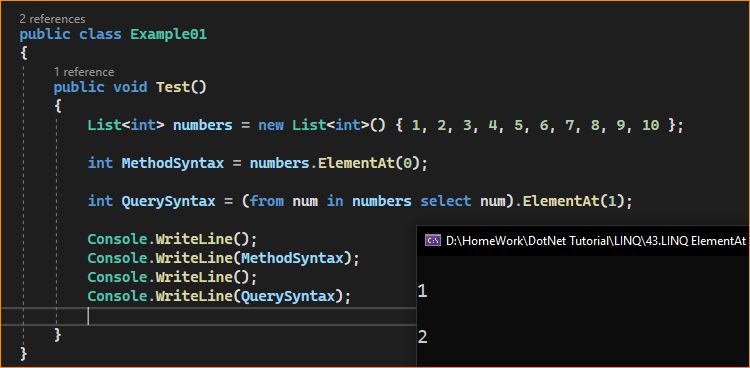
The LINQ ElementAt Method in C# returns the element at a specified index in a sequence. If the data source is empty or the provided index value is out of range, we will get ArgumentOutOfRangeException. If the Data Source is Null, then it will throw ArgumentNullException. If you go to the definition of the ElementAt method, then you will see the following signature.

As you can see, this method takes one parameter, i.e., the index position. Then, it will return the element present in that index position of the data source. There is no overloaded version available for this method.

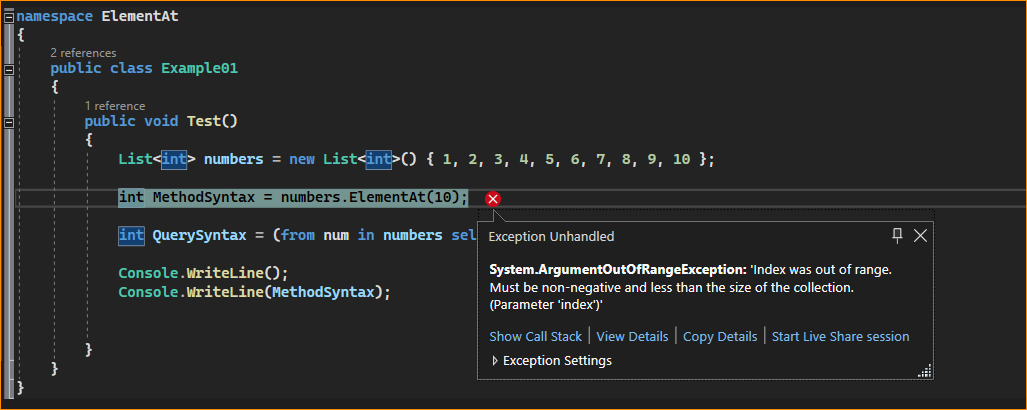
* **Purpose**: Returns the element at a specified index in the sequence.
* **Usage**: You use ElementAt when you are confident that the index you specify is within the bounds of the collection. If the specified index is out of range, it will throw an ArgumentOutOfRangeException.

1. Example to Understand LINQ ElementAt Method in C#.

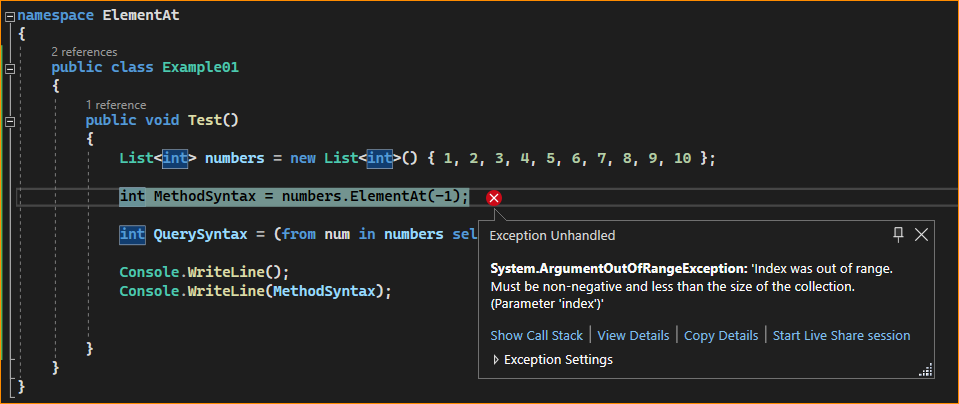
Let us see an example to Understand the LINQ ElementAt Method in C# with both Method and Query Syntax. Our requirement is to fetch the Element Present in Index Position 1 using the ElementAt Method. For a better understanding, please have a look at the following example. Here, we have created one data source that contains integer numbers. Then, we fetch the element present in index position 1 by using the ElementAt method, and to that method, we pass the value 1. There is no such operator called ElementAt available to write the Query Syntax. If you want, you can combine the method syntax and query syntax to write the code, as shown in the example below.



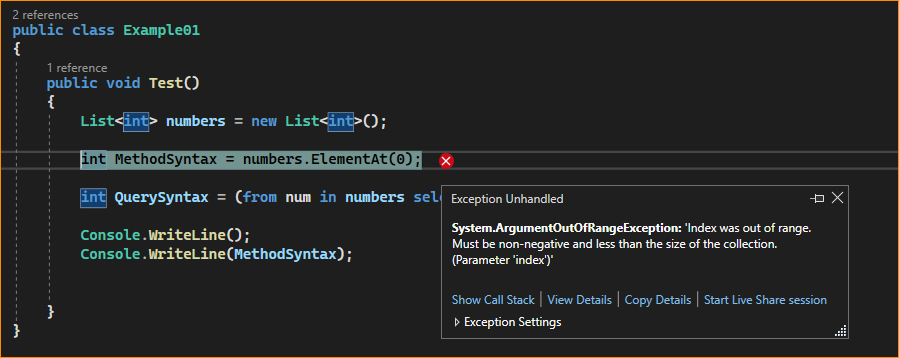
What happen when index not in List or array-



What happen enter negative values –

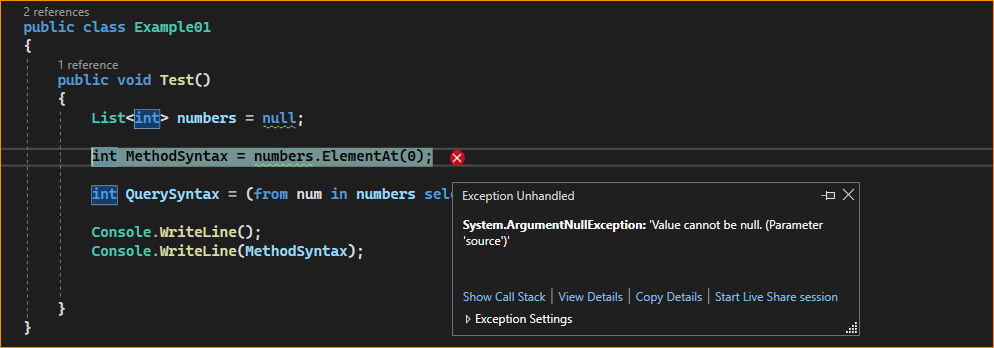


What happen When data source empty-



What happens If we call the LINQ ElementAt method on a Data Source, which is Null?

If we call the LINQ ElementAt method on a Data Source that is Null, then it will throw ArgumentNullException. Let us understand this with an example. Please have a look at the following example. Here, the data source or collection is Null; on the Null Data Source, we call the LINQ ElementAt Method.



**Points to Remember:**

While working with the LINQ ElementAt Method in C#, we will get runtime ArgumentOutOfRangeException exceptions in the following scenarios.

1. If the Data Source is empty. ArgumentOutOfRangeException
2. If you specify a negative value for the index position. ArgumentOutOfRangeException
3. If you specify the index position which is out of range. ArgumentOutOfRangeException

If you don’t want the Runtime ArgumentOutOfRangeException Exception, instead you want to return a default value, then you need to use the LINQ ElementAtOrDefault method.

1. LINQ ElementAtOrDefault Method in C#:

The LINQ **ElementAtOrDefault** method in C# exactly does the same thing as the LINQ ElementAt method except that this method does not throw an **ArgumentOutOfRangeException** exception when the data source is empty or when the supplied index value is out of range or when you specify a negative value for the index position. In such cases, it will return the default value based on the data type of the element the data source contains. If the Data Source is Null, then it will throw ArgumentNullException.

That means this method returns the default value (based on the data source data type) if the index is outside the bounds of the source sequence; otherwise, the element is at the specified position in the source sequence. If you go to the definition of the ElementAtOrDefault method, then you will see the following signature.

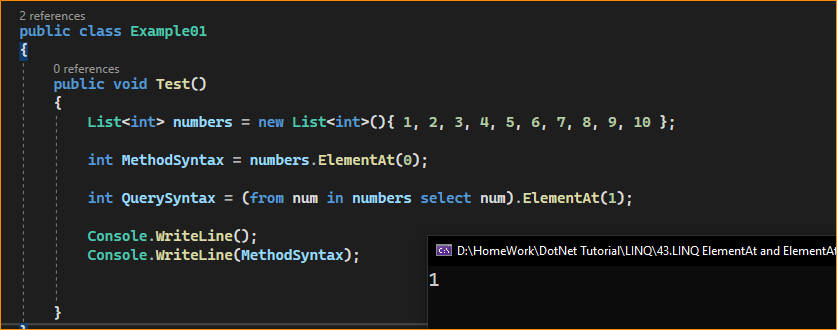


* Purpose: Returns the element at a specified index in the sequence or the default value for the type if the index is out of range.
* Usage: You use ElementAtOrDefault when the index might be outside the bounds of the collection and you want to avoid throwing an exception. The default value for reference types is null, and for value types, it’s the default constructor value (for example, 0 for int, false for bool, etc.).

Note: Like the LINQ ElementAt method, the ElementAtOrDefault method also does not have an overloaded version. Let us understand this method with Examples.

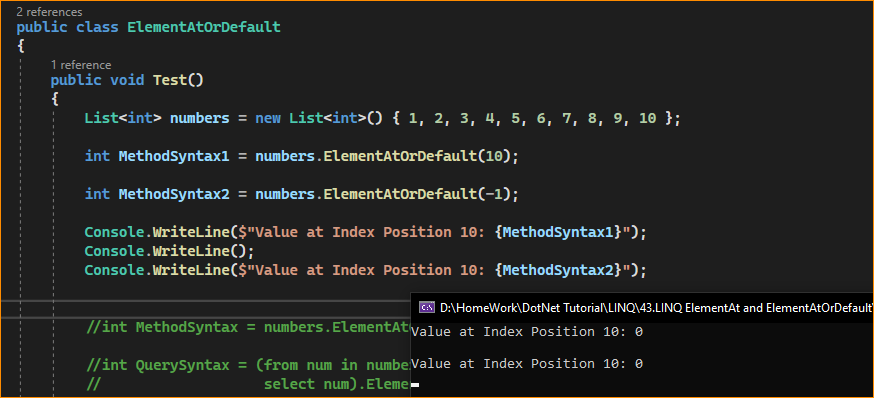
1. Example to Understand LINQ ElementAtOrDefault Method in C#.

Let us see an example to Understand the LINQ ElementAtOrDefault Method in C# with both Method and Query Syntax. Our requirement is to fetch the Element Present in Index Position 1 using the ElementAtOrDefault Method. For a better understanding, please have a look at the following example. Here, we have created one data source which contains integer numbers. Then, we fetch the element present in index position 1 by using the ElementAtOrDefault method, and to that method, we pass the value 1. There is no such operator called ElementAtOrDefault available to write the Query Syntax. If you want, you can combine the method syntax and query syntax to write the code, as shown in the example below.



1. What happens if the Index Value is out of the Range of the Collection?

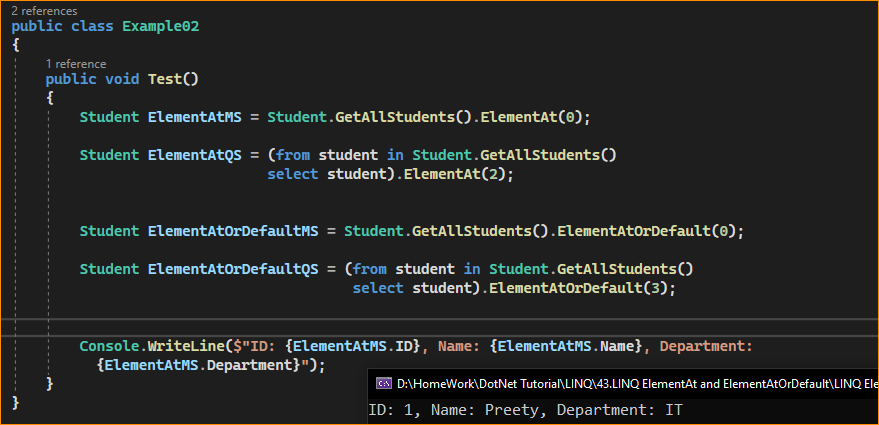
Please have a look at the following example for a better understanding. The data source or collection contains 10 elements so that the index will start from 0 to 9. Let’s see what happens when we try to fetch the element from index position 10 or pass a negative index value, as shown in the example below. In this case, as we are using the ElementAtOrDefault method, it will not throw any exception. Instead, it will return the default value based on the data type of the data source. Here, the collection data type is Integer, and the default value of Integer is 0. So, in the below example, it will return 0.



1. LINQ ElementAt and ElementAtOrDefault Methods in C# with Complex Type:

Let us Understand LINQ ElementAt and ElementAtOrDefault Methods in C# with Complex Type. For this, we are going to use the following Student class. So, first, create a class file with the name Student.cs and then copy and paste the following code into it. It is a very simple class having 3 properties and one method, which is to return a collection of students.

Next, modify the Main method of the Program class as follows. In the example below, I show how to use both LINQ ElementAt and ElementAtOrDefault Methods in C# with Complex Type using Method and Query Syntax.



1. What is the Difference Between the LINQ ElementAt and ElementAtOrDefault Methods in C#?

Both methods are used to return an element from the specified index of a data source. But if the element is not available at the specified index position, if the data source is empty, or if we specified a negative value for the Index position, then the ElementAt method will throw an ArgumentOutOfRangeException exception. At the same time, the ElementAtOrDefault method will not throw an exception. Instead, it returns a default value based on the data source elements’ data type.

When to Use Them

**ElementAt**: When working with a sequence where you’re sure the index you’re querying will be within bounds, similar to accessing an array by index. This might be when you’ve already checked the collection size or when dealing with fixed-size collections.

**ElementAtOrDefault**: When dealing with dynamic collections where you’re unsure if the index you’re querying will be within bounds, you want to handle cases that aren’t safe. This is particularly useful when you don’t want to catch an exception for performance reasons or when you’re iterating through indices that could be out of bounds.

Use ElementAt when you expect the index to be valid and you want to retrieve the element at that index. If the index is out of range, it will throw an exception, which can help you catch unexpected issues.

Use ElementAtOrDefault when uncertain about the index’s validity or you want to handle out-of-range scenarios gracefully without throwing exceptions. It returns the default value instead of raising an exception.

It’s important to note that using these methods on sequences that do not have an indexer and require enumeration (like IEnumerable sequences that are not lists or arrays) can result in poor performance if used repeatedly, as the sequence must be enumerated up to the specified index each time. If multiple index-based retrievals are needed for such sequences, it might be more efficient to first convert the sequence to a list or array.