1. What is LINQ GroupBy Method in C#?

In C#, LINQ (Language Integrated Query) provides a powerful way to query and manipulate collections of data. The GroupBy method in LINQ is used to group elements in a collection based on a specified key and create groups of elements that share the same key.

The LINQ GroupBy Method in C# belongs to the Grouping Operators Category. This method exactly does the same thing as the Group By clause does in SQL Query. This method takes a flat sequence of elements and then organizes the elements into groups (i.e., IGrouping<TKey, TSource>) based on a given key. That means that based on the given key, it will group the elements.

If you go to the definition of the GroupBy method, then you will see that it returns an IEnumerable<IGrouping<TKey, TSource>> where TKey is nothing but the Key value on which the grouping has been formed, and TSource is the collection of elements that matches the grouping key value. If this is not clear at the moment, then don’t worry. Let us try to understand this with some examples.

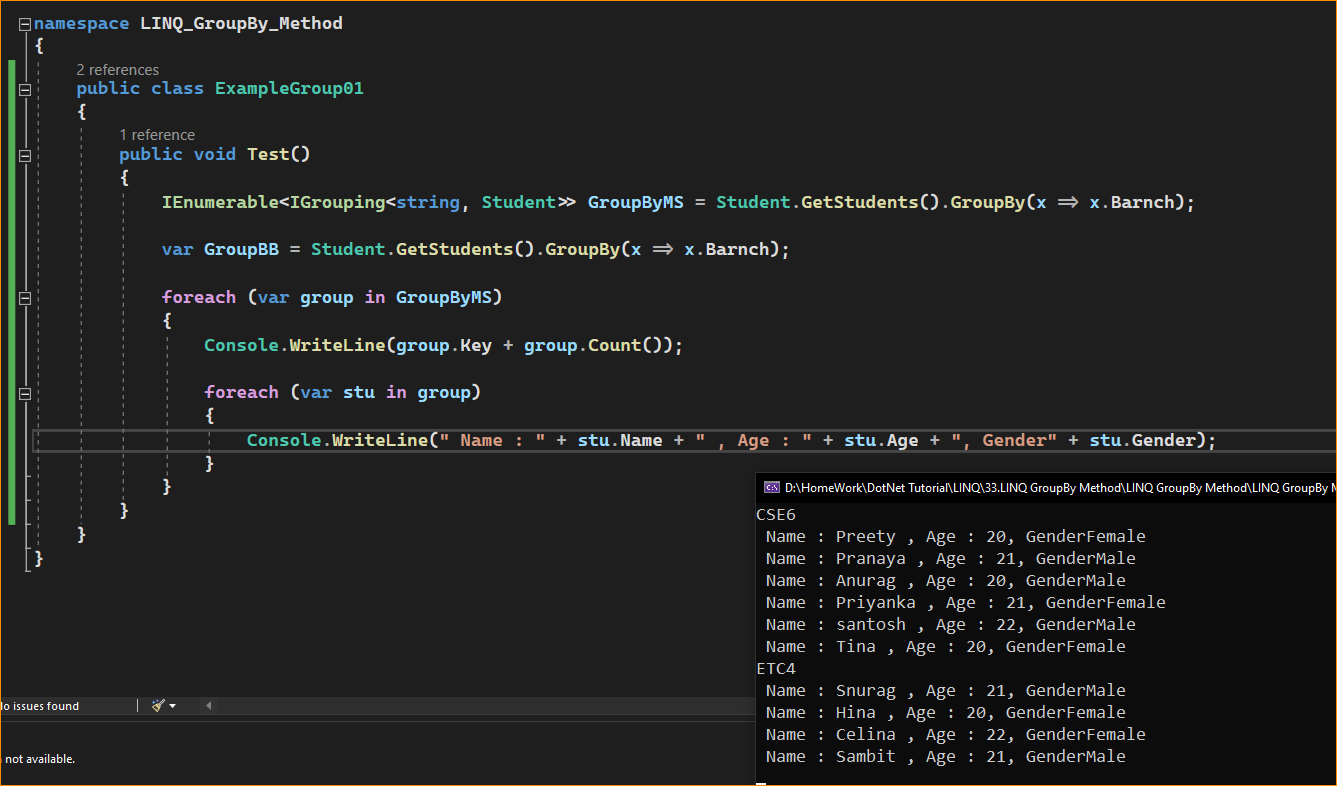
1. Examples to Understand LINQ GroupBy Method in C#

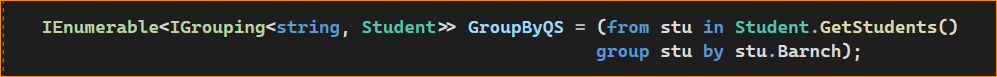
Let us understand how to use the LINQ GroupBy Method in C# using both Method and Query Syntax with some examples. We will use the following Student class to understand the GroupBy Method. So, create a class file named Student.cs and copy and paste the following code. This is a very simple class having five properties such as ID, Name, Gender, Branch, and Age. This class also has one method called GetStudents(), which returns a list of all students, and this is going to be our data source.

Grouping the Students Based on the Branch

* Now, our requirement is to group the students based on Branch.

For a better understanding, please look at the following example, which exactly does the same thing. The following example organizes the students into groups based on their branch (i.e., **the branch will act as the key**). It also means that students with the same branch will be stored in the same group, where each group has a key and the corresponding student collection. Here, the key will be the Branch, and the collection will be the student belonging to that branch. The following example code is self-explained, so please go through the comment lines. In the below example, I am showing how to use the GroupBy Method using both Method Syntax and Query Syntax.





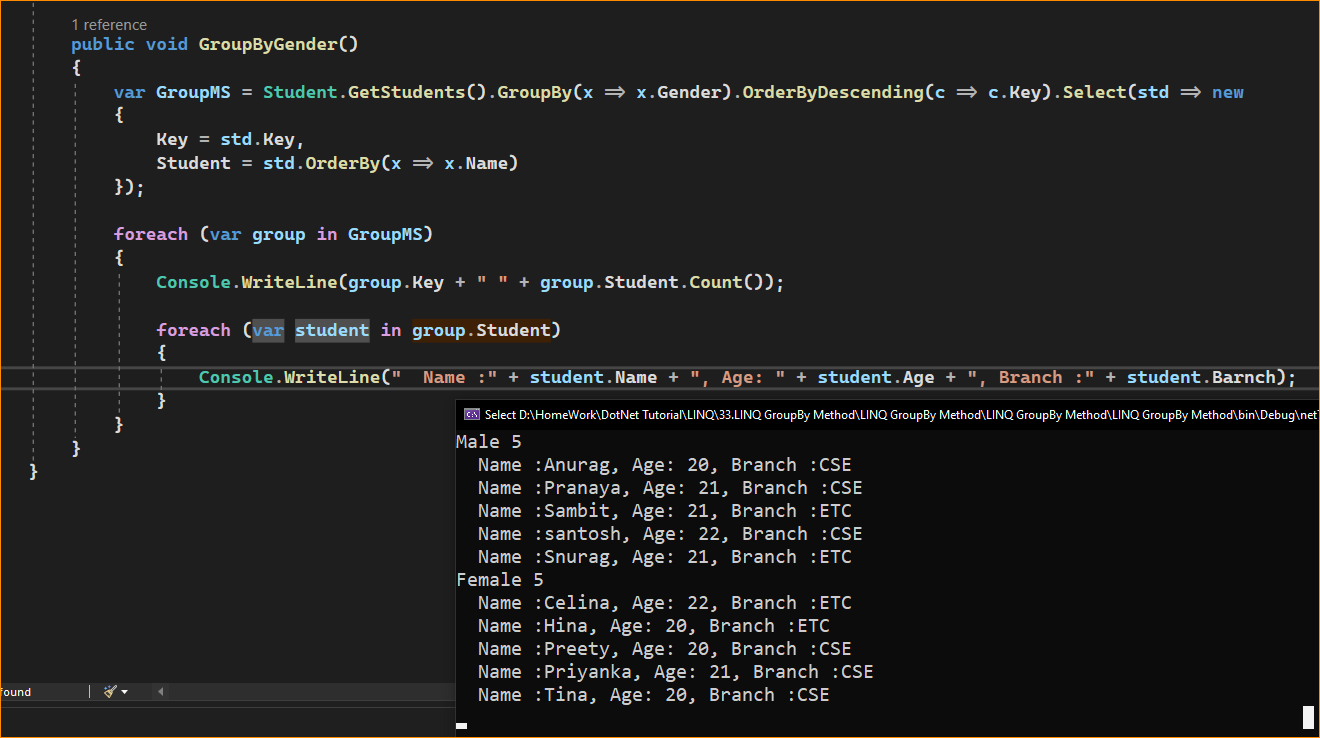
Note: Each group has a key, and you can access the value of the key by using the key property. Along the same line, you can use the count property to check how many elements are in that group. Again, using a for each loop, you can access all the elements of a group, as shown in the above example.

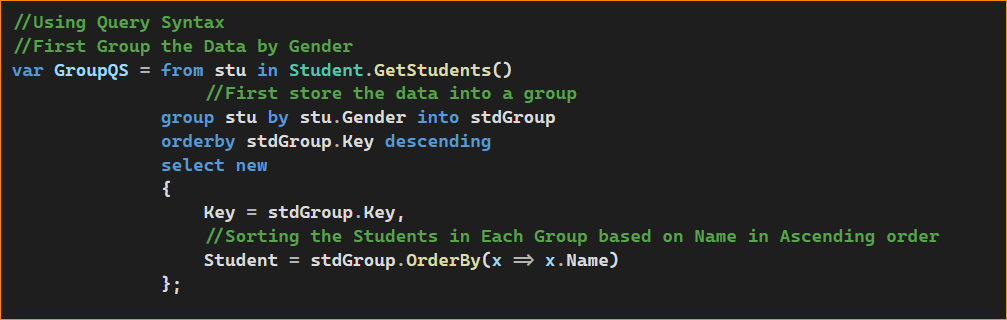
1. Grouping Students by Gender in Descending Order, Names in Ascending Order in Each Group

Let us see an example to Understand the LINQ GroupBy Method with the following Requirements.

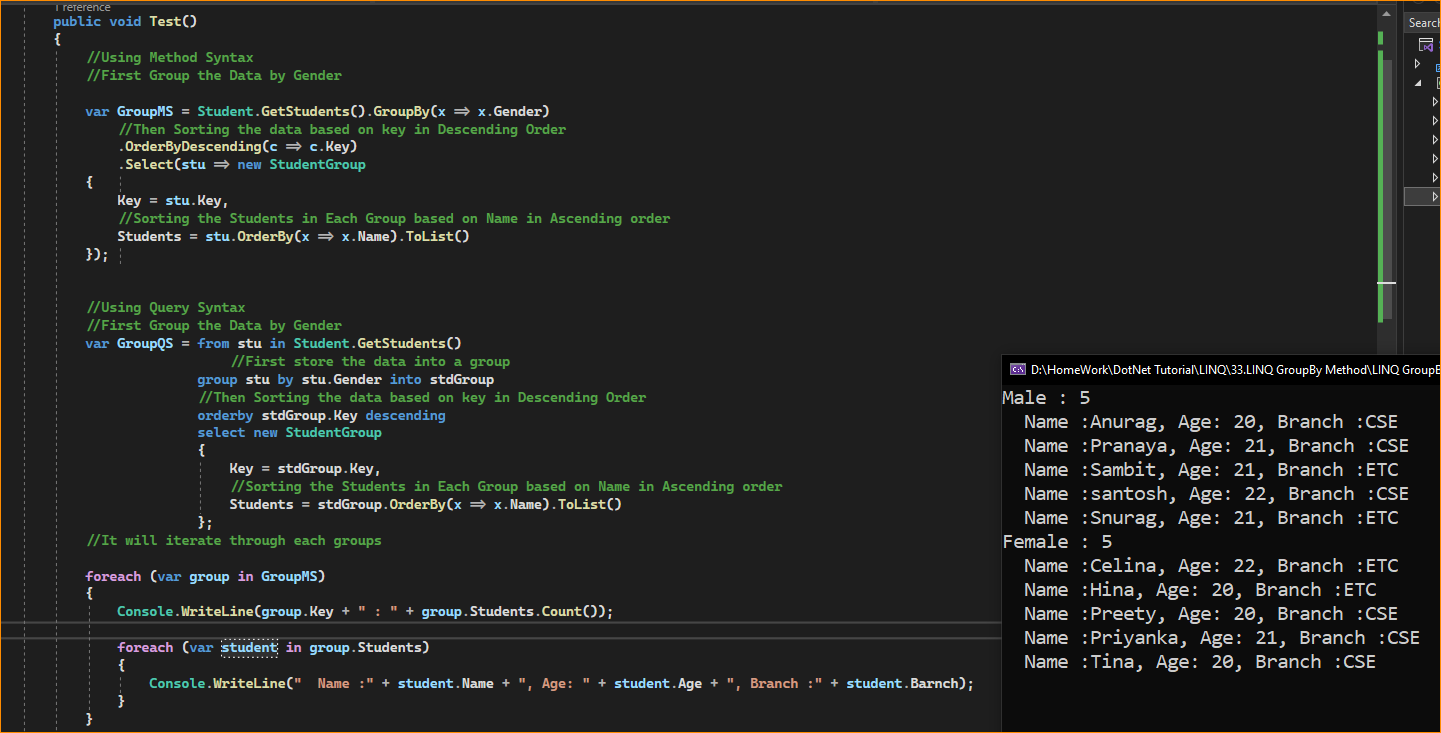
* First, Grouping the Students by Gender.
* Sort the Groups in Descending Order, i.e., soring the gender in Descending Order.
* Finally, Sort the Student Names in each group in Ascending Order.

In the following example, we group the students by Gender using the LINQ GroupBy Method. Then, we sort the data by Gender in descending order, and finally, we sort the students in each group by their name in Ascending Order. The following example code is self-explained, so please go through the comment lines.





In the above example, the result is projected to an anonymous type. If you want, you create a new type with the required properties, and then you can project the result to that new type. Let us understand this. First, create a class file named StudentGroup.cs and copy and paste the following code.



1. When to use the LINQ GroupBy Method in C#?

The LINQ GroupBy method in C# groups elements in a collection based on a specified key or property. It is used for performing grouping and aggregation operations on data. You should consider using the GroupBy method in the following situations:

* Grouping Data: When you have a collection of data, such as a list or an array, and want to group elements based on a common attribute or property. For example, you can group a list of employees by their department, customers by their country, or orders by their date.
* Aggregation: When you need to perform aggregate operations, such as calculating sums, averages, counts, or other statistics for each group of elements. After using GroupBy to group the data, you can use aggregate functions like Sum, Average, Count, Min, Max, etc., to calculate values for each group.
* Data Transformation: When you want to transform the structure of your data by grouping and nesting elements. For instance, you can use GroupBy to create hierarchical data structures, like nested dictionaries or objects, where each group becomes a key, and the elements in the group are stored as values.
* Query Composition: When building complex queries by chaining multiple LINQ operations together, including filtering, sorting, and projecting, GroupBy can be a key part of your query to organize data before applying further operations.
* Duplicates Identification: If you want to identify duplicates in a collection based on a certain key, GroupBy can group elements by that key and filter groups with more than one element.
* Relationship Data Handling: In scenarios where you have parent-child relationships, you need to group children by their parent entity.
* Remember that GroupBy is executed as a deferred execution, which means the grouping is not performed until you iterate over the groups. This is part of LINQ’s design to promote efficient querying and to avoid unnecessary work if the results are never used.