GENERIC COLLECTIONS

Generic collections are collections that are type-safe and can only hold objects of a specifictype.

They are defined in the System.Collections.Generic namespace and include the following classes:

List<T>

A dynamic-sized list that can hold objects of type T.

Dictionary<Tkey,TValue>

A collection of key/value pairs that maps keys of type TKey to values of type TValue.

SortedList<TKey,TValue>

A collection of key/value pairs that are sorted by the keys of type TKey.

Stack<T>

A collection of objects that supports adding and removing objects in a LIFO (last-in, first-out) manner.

Queue<T>

A collection of objects that supports adding and removing objects in a FIFO (first-in, first-out) manner.

LIST<T>

The **List<T>** in *C#* is a *generic* collection that represents a dynamic-sized list of objects of type **T**.

It is defined in the ***System.Collections.Generic*** namespace and provides a flexible way to store and manipulate collections of objects.

The List<T> class is the *generic* equivalent of the ***ArrayList*** class.

Some of the main features of List<T> include:

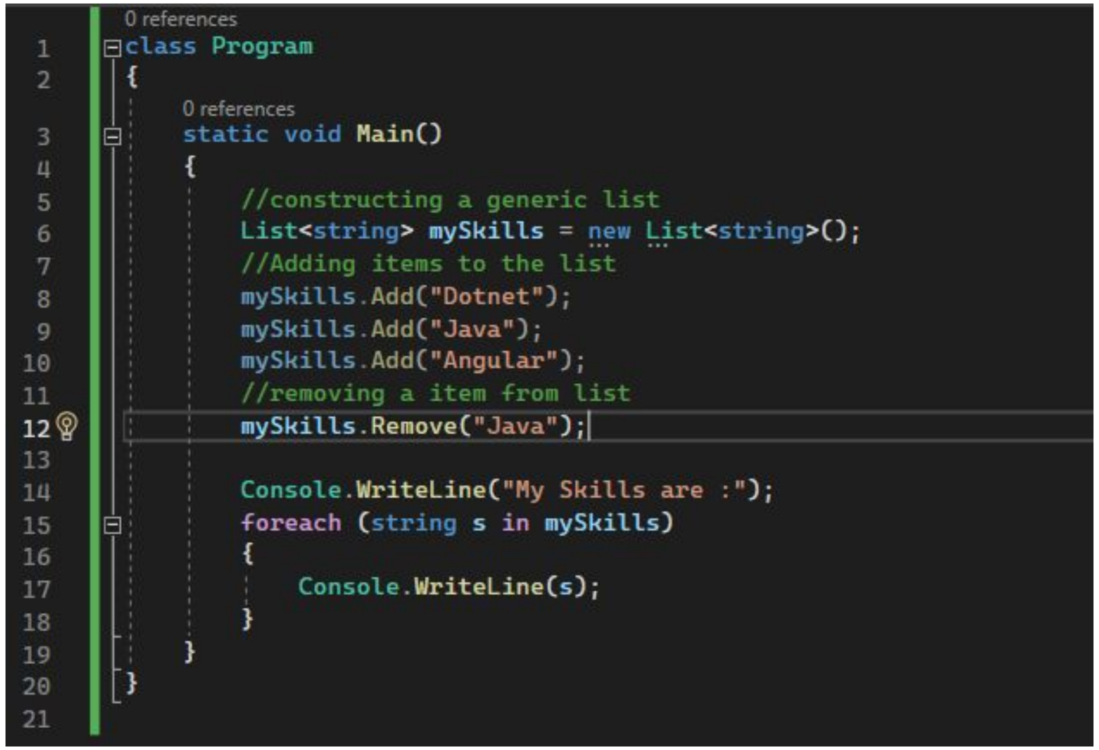
Dynamic sizing: The size of a **List<T>** can be modified at runtime, allowing you to add or remove items as needed.

Type safety: Since **List<T>** is a generic collection, it can only hold objects of type **T**, which provides type safety and prevents runtime errors.

Index-based access: Items in a **List<T>** can be accessed by their index, making it easy to retrieve and manipulate specific items in the list.

Built-in functionality : **List<T>** provides a range of methods for adding, removing, searching, and sorting items in the list.

Example



```
0 references
class Program
{
    0 references
    static void Main()
    {
        //constructing a generic list
        List<string> mySkills = new List<string>();
        //Adding items to the list
        mySkills.Add("Dotnet");
        mySkills.Add("Java");
        mySkills.Add("Angular");
        //removing a item from list
        mySkills.Remove("Java");

        Console.WriteLine("My Skills are :");
        foreach (string s in mySkills)
        {
            Console.WriteLine(s);
        }
    }
}
```

DICTIONARY<TKEY, TVALUE>

It is a generic collection that represents a collection of key/value pairs.

It is defined in the System.Collections.Generic namespace and provides a flexible way to store and manipulate collections of key/value pairs.

Key/value pairs: A **Dictionary<TKey, TValue>** stores a collection of key/value pairs. Each key in the dictionary is unique.

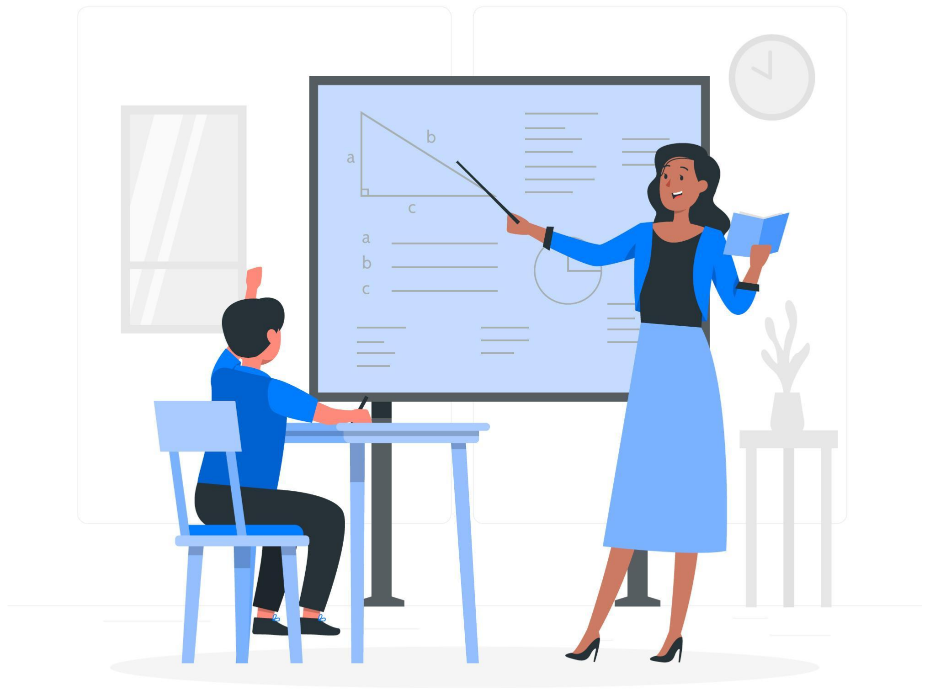
Fast lookup: Because **Dictionary<TKey, TValue>** is implemented as a hash table, accessing an element in the dictionary by key is very fast.

Dynamic sizing: The size of a **Dictionary<TKey, TValue>** can be modified at runtime, allowing you to add or remove key/value pairs as needed.

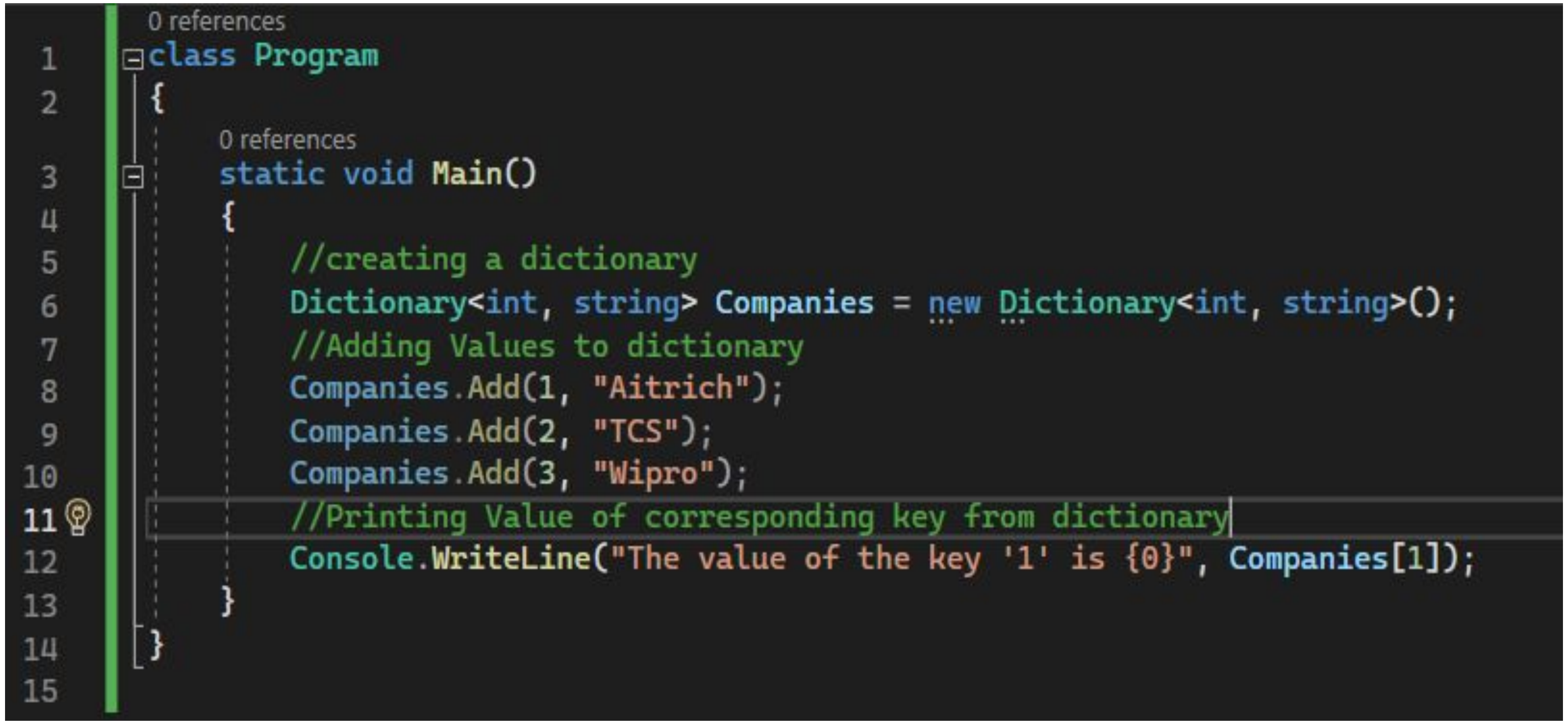
Type safety: Since **Dictionary<TKey, TValue>** is a generic collection, it can only hold objects of type **TKey** for keys and **TValue** for values, which provides type safety and prevents runtime errors.

It provides methods such as

- Add(TKey key, TValue value)
- ContainsKey(TKey key),
- ContainsValue(TValue value)
- Remove(TKey key)
- Clear()



Example

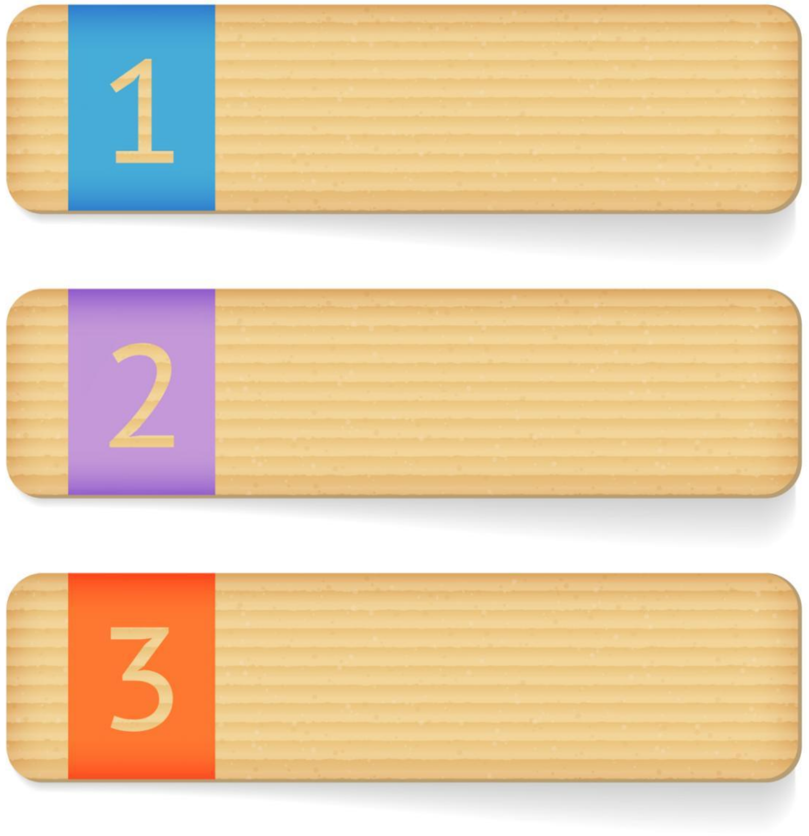


```
0 references
class Program
{
    0 references
    static void Main()
    {
        //creating a dictionary
        Dictionary<int, string> Companies = new Dictionary<int, string>();
        //Adding Values to dictionary
        Companies.Add(1, "Aitrich");
        Companies.Add(2, "TCS");
        Companies.Add(3, "Wipro");
        //Printing Value of corresponding key from dictionary
        Console.WriteLine("The value of the key '1' is {0}", Companies[1]);
    }
}
```

SORTEDLIST<TKEY, TVALUE>

SortedList<TKey, TValue> is a generic collection that represents a collection of key/value pairs that are sorted by key.

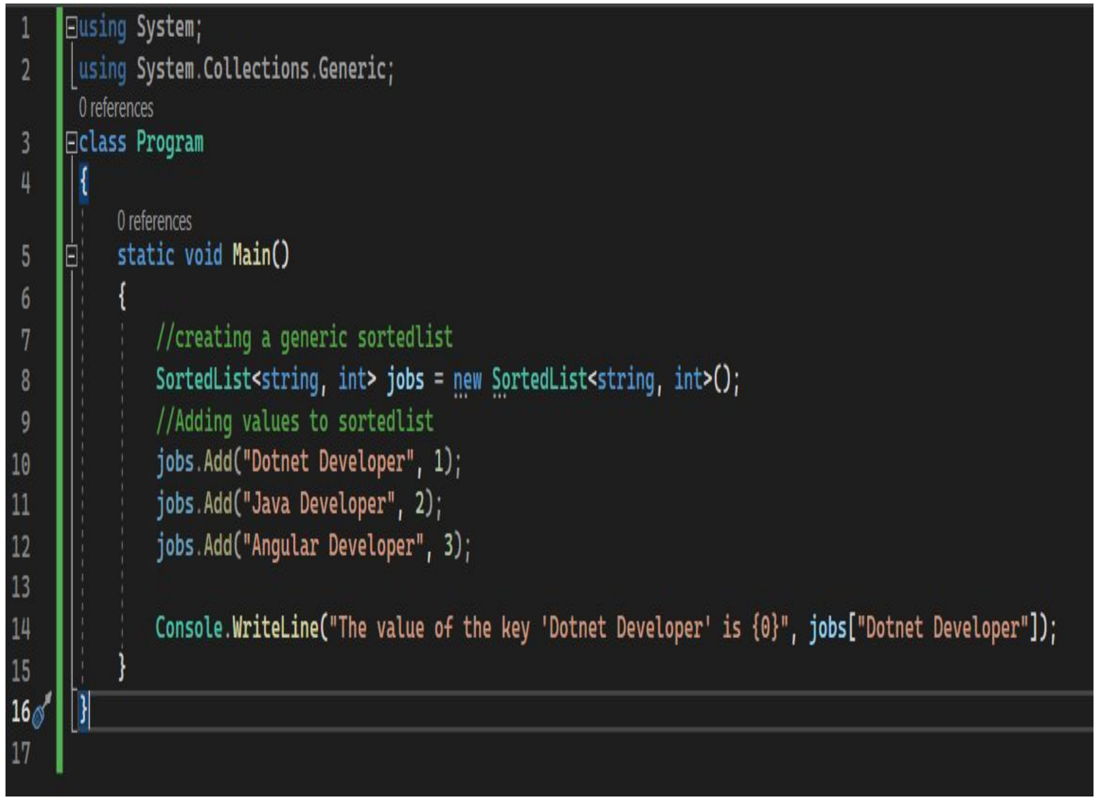
It is defined in the System.Collections.Generic namespace and provides a way to store and manipulate collections of key/value pairs in a sorted order.



It provides methods such as

- Add(TKey key, TValue value)
- ContainsKey(TKey key),
- ContainsValue(TValue value)
- Remove(TKey key)
- Clear()

Example



```
using System;
using System.Collections.Generic;
0 references
class Program
{
    0 references
    static void Main()
    {
        //creating a generic sortedlist
        SortedList<string, int> jobs = new SortedList<string, int>();
        //Adding values to sortedlist
        jobs.Add("Dotnet Developer", 1);
        jobs.Add("Java Developer", 2);
        jobs.Add("Angular Developer", 3);

        Console.WriteLine("The value of the key 'Dotnet Developer' is {0}", jobs["Dotnet Developer"]);
    }
}
```

QUEUE<T>

Queue<T> is a generic collection in C# that represents a first-in, first-out (FIFO) data structure.

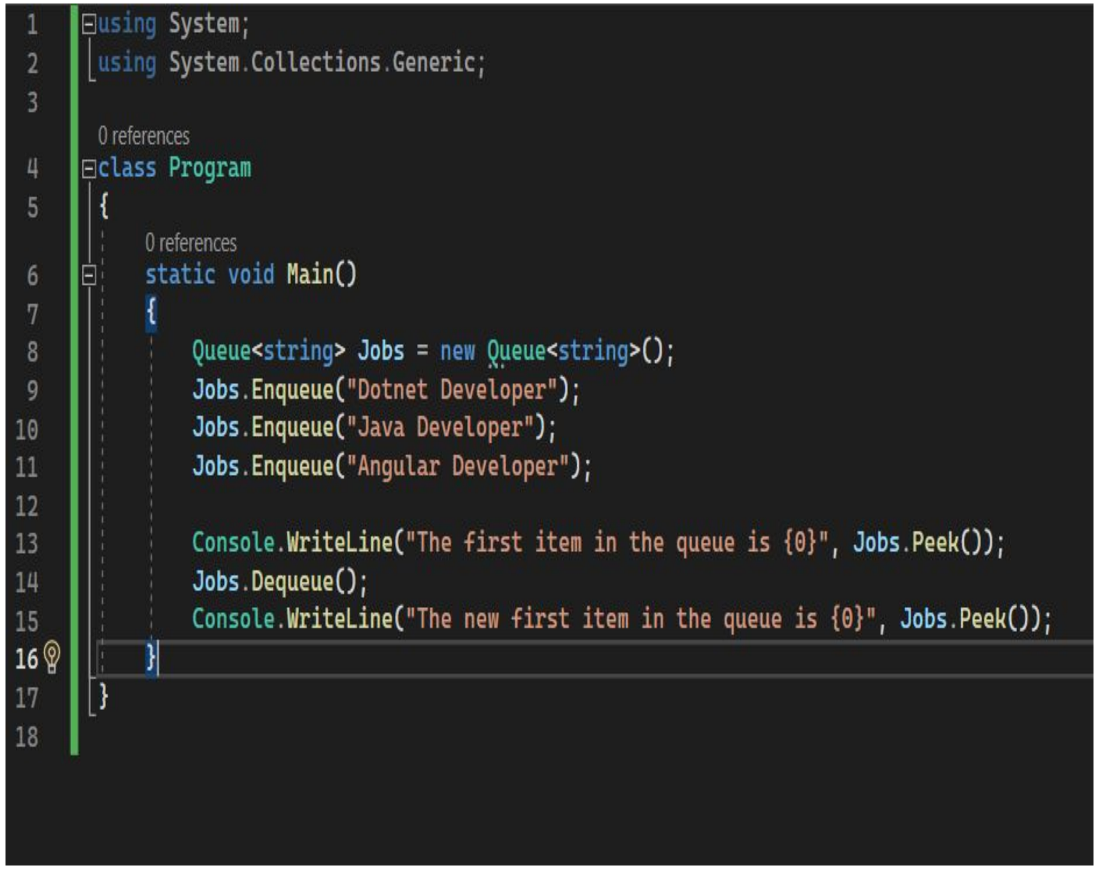
It is defined in the **System.Collections.Generic** namespace and provides a way to store and manipulate a collection of objects in the order in which they were added.

It provides methods such as

- Enqueue
- Dequeue,
- Peek
- Count.

Aitrich

Example



```
using System;
using System.Collections.Generic;

0 references
class Program
{
    0 references
    static void Main()
    {
        Queue<string> Jobs = new Queue<string>();
        Jobs.Enqueue("Dotnet Developer");
        Jobs.Enqueue("Java Developer");
        Jobs.Enqueue("Angular Developer");

        Console.WriteLine("The first item in the queue is {0}", Jobs.Peek());
        Jobs.Dequeue();
        Console.WriteLine("The new first item in the queue is {0}", Jobs.Peek());
    }
}
```

STACK<T>

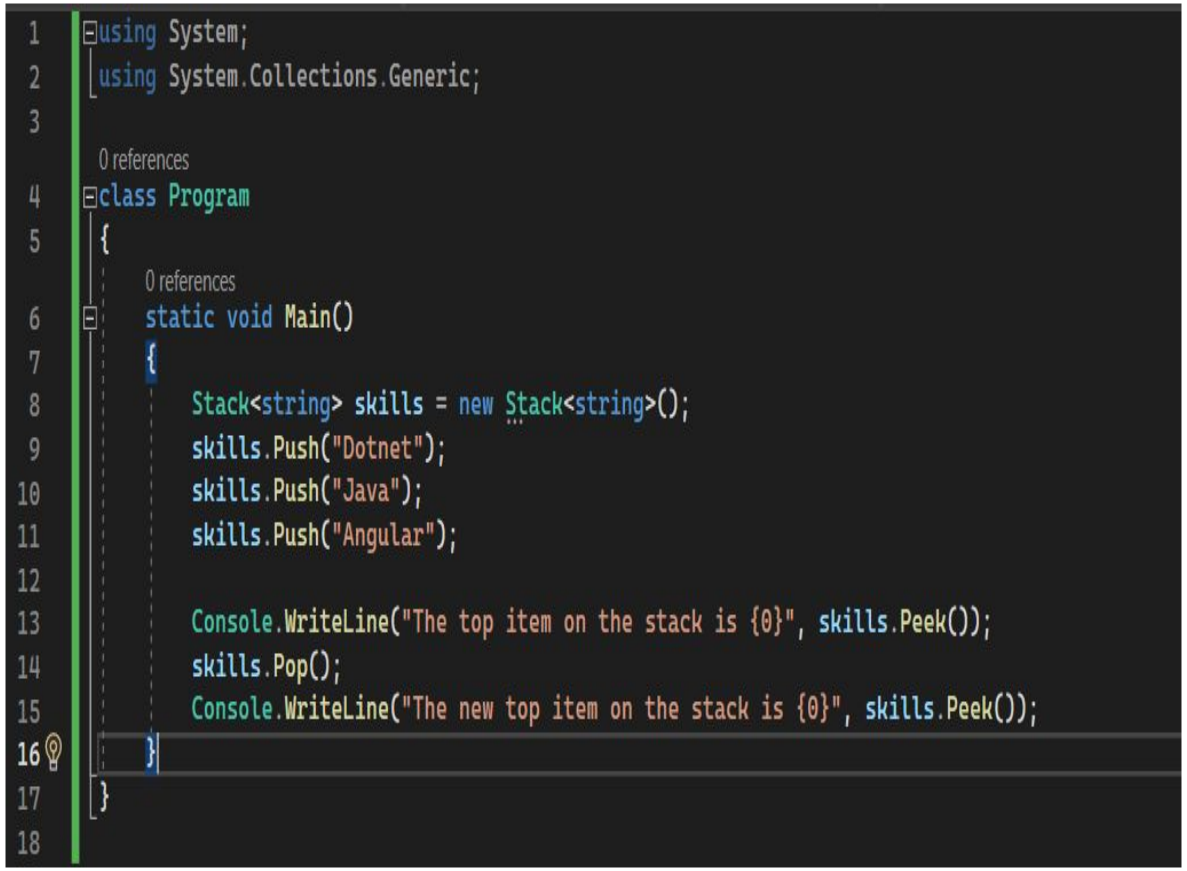
Stack<T> is a generic collection that represents a last-in, first-out (LIFO) data structure.

It is defined in the **System.Collections.Generic** namespace and provides a way to store and manipulate a collection of objects in the specific datatype order in which they were added.

It provides methods such as

- Push
- Pop
- Peek
- Count.
- Contains

Example

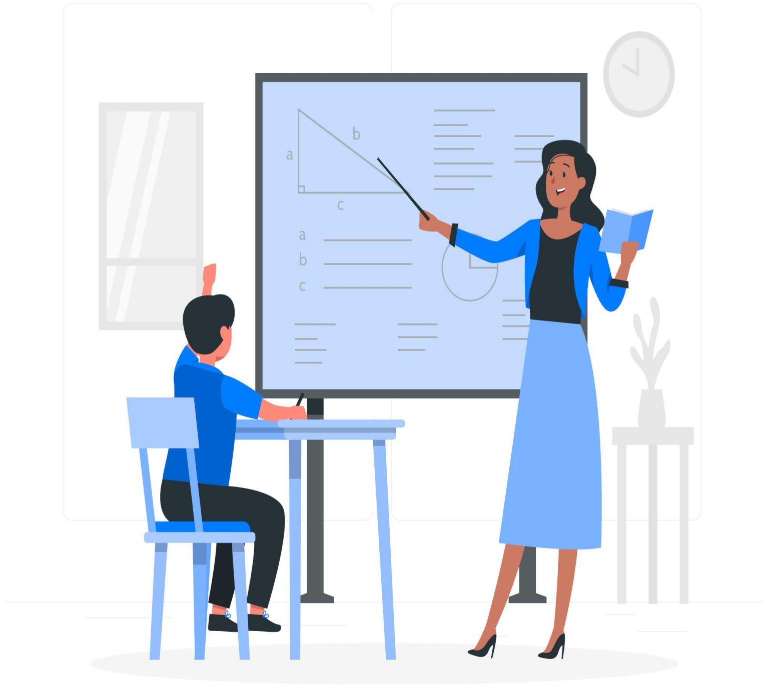


```
using System;
using System.Collections.Generic;

0 references
class Program
{
    0 references
    static void Main()
    {
        Stack<string> skills = new Stack<string>();
        skills.Push("Dotnet");
        skills.Push("Java");
        skills.Push("Angular");

        Console.WriteLine("The top item on the stack is {0}", skills.Peek());
        skills.Pop();
        Console.WriteLine("The new top item on the stack is {0}", skills.Peek());
    }
}
```

LINQ-Language Integrated Query

It is a powerful feature in C# that provides a unified way to query and manipulate data from different data sources. It allows you to write queries against collections, databases, XML documents, and other data sources using a consistent syntax.

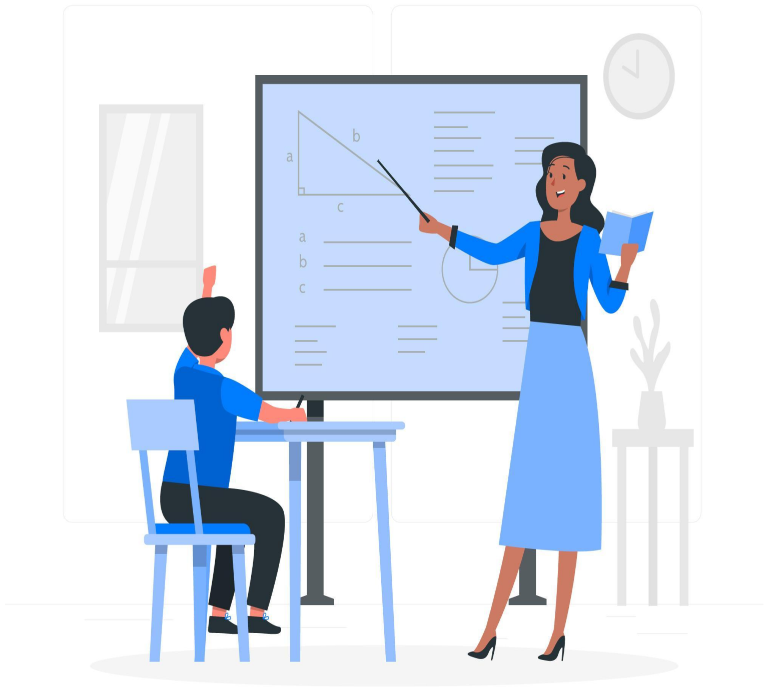
LINQ introduces a set of standard query operators that can be used with any data source that implements the IEnumerable<T> interface. These operators are defined as extension methods in the System.Linq namespace.



Aitrich

LINQ - Example

```
static void Main()
{/ Create a list of integers
    List<string> memberList = new List<string>
{"JobSeeker","admin","JobProvider"};// Query the list using
LINQ
   var allMembers = from member in memberList
               select member;// Iterate over the query results
   foreach (var member in allMembers)
   {
      Console.WriteLine(member);
   }
   Console.ReadLine();
 }
```



thank you