

Chapter 5: C# Collections

In our previous chapter, we have learned about how we can use arrays in C#. Let's have a quick overview of it, Arrays in programming are used to group a set of related objects. So one could create an array or a set of Integers, which could be accessed via one variable name.

Collections are similar to Arrays, it provide a more flexible way of working with a group of objects.

In arrays, you would have noticed that you need to define the number of elements in an array beforehand. This had to be done when the array was declared.

But in a collection, you don't need to define the size of the collection beforehand. You can add elements or even remove elements from the collection at any point of time. This chapter will focus on how we can work with the different collections available in C#.

C# ArrayList

The ArrayList collection is similar to the Arrays data type in C#. The biggest difference is the dynamic nature of the array list collection.

For arrays, you need to define the number of elements that the array can hold at the time of array declaration. But in the case of the Array List collection, this does not need to be done beforehand. Elements can actually be added or removed from the Array List collection at any point of time. Let's look at the operations available for the array list collection in more detail.

1. Declaration of an Array List – The declaration of an ArrayList is provided below. An array list is created with the help of the ArrayList Data type. The “new” keyword is used to create an object of an

ArrayList. The object is then assigned to the variable a1. So now the variable a1 will be used to access the different elements of the array list.

ArrayList a1 = new ArrayList()

2. Adding elements to an array – The add method is used to add an element to the ArrayList. The add method can be used to add any sort of data type element to the array list. So you can add an Integer, or a string, or even a Boolean value to the array list. The general syntax of the addition method is given below

ArrayList.add(element) Below are some examples of how the “add” method can be used. The add method can be used to add various data types to the Array List collection.

Below you can see examples of how we can add Integer’s Strings and even Boolean values to the Array List collection.

a1.add(1) – This will add an Integer value to the collection

a1.add(“Example”) – This will add a String value to the collection

a1.add(true) – This will add a Boolean value to the collection

Now let’s see this working at a code level. All of the below-mentioned code will be written to our Console application. The code will be written to our Program.cs file. In the program below, we will write the code to create a new array list. We will also show to add elements and to display the elements of the Array list.

```
static void Main(string[] args)
{
    ArrayList a1 = new ArrayList();
    a1.Add(1);
    a1.Add("Example");
    a1.Add(true);
    Console.WriteLine(a1[0]);
    Console.WriteLine(a1[1]);
    Console.WriteLine(a1[2]);
    Console.ReadKey();
}
```

The image shows a code editor with a C# program. Three callout boxes with arrows point to specific lines of code:

- Box 1 (top right):** "Defining an array list" points to the line `ArrayList a1 = new ArrayList();`.
- Box 2 (middle left):** "Adding elements to the array list" points to the three lines `a1.Add(1);`, `a1.Add("Example");`, and `a1.Add(true);`.
- Box 3 (bottom right):** "Displaying the elements of the array list" points to the three lines `Console.WriteLine(a1[0]);`, `Console.WriteLine(a1[1]);`, and `Console.WriteLine(a1[2]);`.

Code Explanation:

1. The first step is used to declare our Array List. Here we are declaring a1 as a variable to hold the elements of our array list.
2. We then use the add keyword to add the number 1 , the String “Example” and the Boolean value ‘true’ to the array list.
3. We then use the Console.WriteLine method to display the value of each array lists element to the console. You will notice that just like arrays, we can access the elements via their index positions. So to access the first position of the Array List, we use the [0] index position. And so on and so forth.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/DemoApplication/bin/Debu
1
Example
True
```

The elements of the array list displayed in the output

From the output, you can clearly see that all of the elements from the array list are sent to the console.

Let's look at some more methods which are available as part of the ArrayList.

Count – This method is used to get the number of items in the ArrayList collection. Below is the general syntax of this statement.
ArrayList.Count() – This method will return the number of elements that the array list contains. **Contains** - This method is used to see if an element is present in the ArrayList collection. Below is the general syntax of this statement

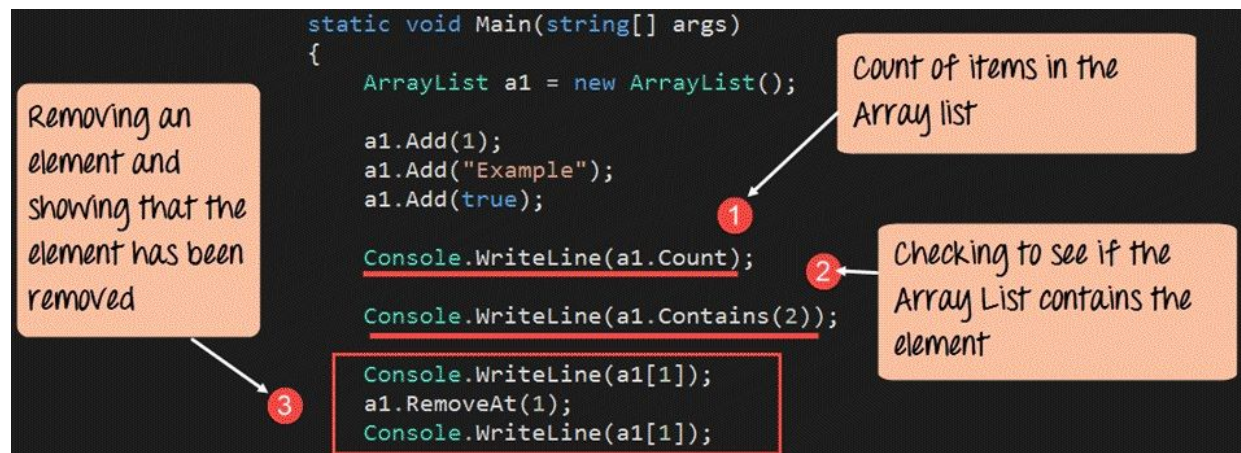
ArrayList.Contains(element) – This method will return true if the element is present in the list , else it will return false.

RemoveAt - This method is used to remove an element at a specific

position in the ArrayList collection. Below is the general syntax of this statement

`ArrayList.RemoveAt(index)` – This method will remove an element from a specific position of the Array List.

Now let's see this working at a code level. All of the below-mentioned code will be written to our Console application. The code will be written to our Program.cs file. In the below program, we will write the code to see how we can use the abovementioned methods.



Code Explanation:

1. So the first property we are seeing is the Count property. We are getting the

Count property of the array list a1 and then writing it to the Console.

2. In the second part, we are using the Contains method to see if the arraylist a1

contains the element 2. We then write the result to the Console via the

Writeline command.

3. Finally to showcase the Remove element method , we are performing the below steps,

a. First, we write the value of the element at Index position 1 of the array list to the console.

b. Then we remove the element at Index position 1 of the array list. c.

Finally, we again write the value of the element at Index position 1 of the

array list to the console. This set of steps will give a fair idea whether

the
remove method will work as it should be.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/DemoApplication/bin/Debug/Der
3
False
Example
True
```

output of the various
Array List functions

Why the last
value is true?

If you see the sequence of events , the element Example is removed from the array because this is at position 1. Position 1 of the array then gets replaced by what was in position 2 earlier which the value 'true'

C# Stack

The stack is a special case collection which represents a last in first out (LIFO) concept. To first understand LIFO, let's take an example. Imagine a stack of books with each book kept on top of each other.

The concept of last in first out in the case of books means that only the top most book can be removed from the stack of books. It is not possible to remove a book from between, because then that would disturb the setting of the stack.

Hence in C#, the stack also works in the same way. Elements are added to the stack, one on the top of each other. The process of

adding an element to the stack is called a push operation. To remove an element from a stack, you can also remove the top most element of the stack. This operation is known as pop.

Let's look at the operations available for the Stack collection in more detail.

Declaration of the stack – A stack is created with the help of the Stack Data type. The keyword “new” is used to create an object of a Stack. The object is then assigned to the

variable st. Stack st = new Stack()

Adding elements to the stack – The push method is used to add an element onto the stack. The general syntax of the statement is given below. **Stack.push(element)**

Removing elements from the stack – The pop method is used to remove an element from the stack. The pop operation will return the topmost element of the stack. The general syntax of the statement is given below

Stack.pop()

Count – This property is used to get the number of items in the Stack. Below is the general syntax of this statement.

Stack.Count

Contains - This method is used to see if an element is present in the Stack. Below is the general syntax of this statement. The statement will return true if the element exists, else it will return the value false.

Stack.Contains(element)

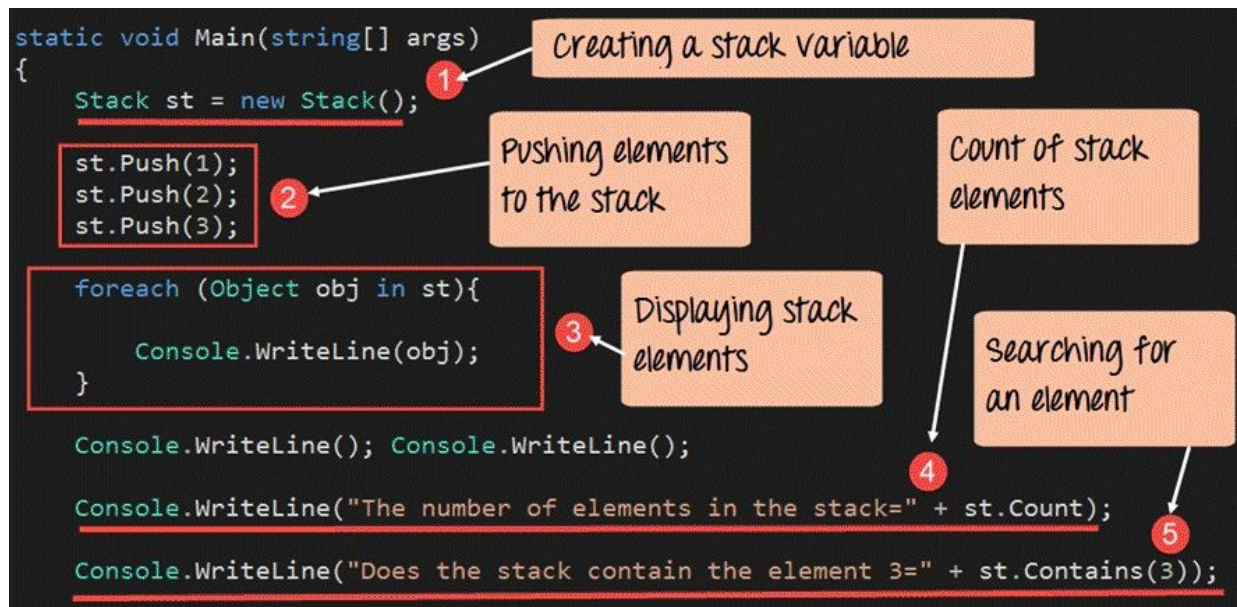
Now let's see this working at a code level. All of the below-mentioned code will be written to our Console application. The code will be written to our Program.cs file. In the below program, we will write the code to see how we can use the abovementioned methods.

In this example, we will see

How a stack gets created.

How to display the elements of the stack , and use the Count and

Contain methods.

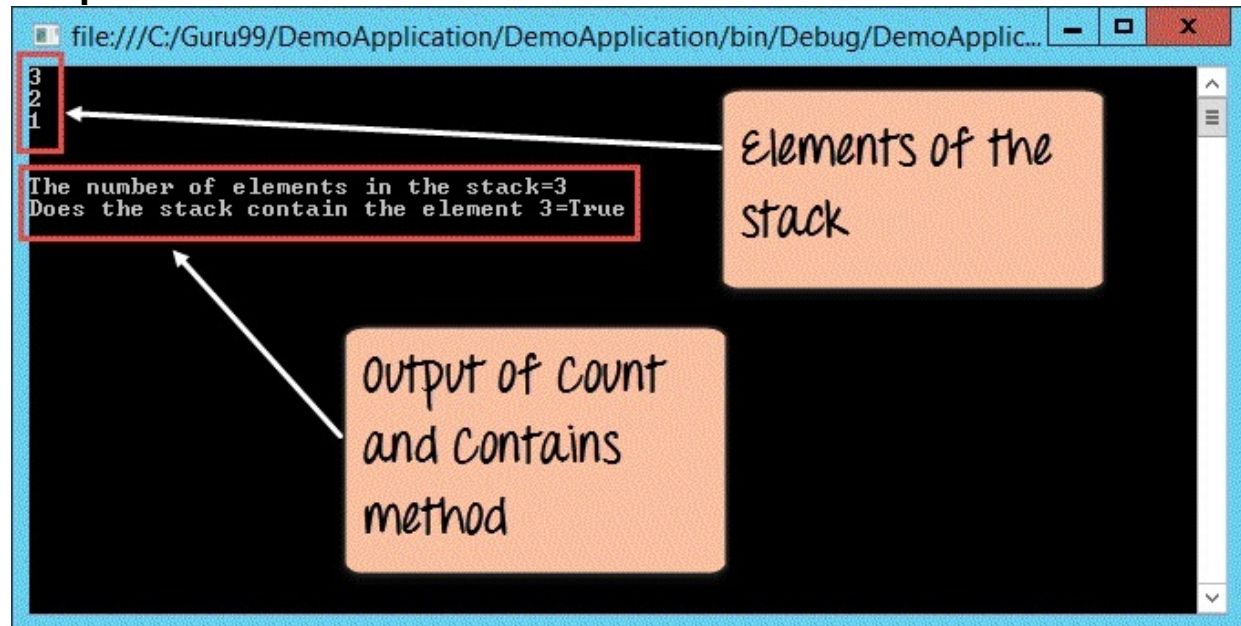


Code Explanation:

1. The first step is used to declare the Stack. Here we are declaring "st" as a variable to hold the elements of our stack.
2. Next, we add 3 elements to our stack. Each element is added via the Push method.
3. Now since the stack elements cannot be accessed via the index position like the array list, we need to use a different approach to display the elements of the stack. The Object (obj) is a temporary variable, which is declared for holding each element of the stack. We then use the foreach statement to go through each element of the stack. For each stack element, the value is assigned to the obj variable. We then use the Console.Writeline command to display the value to the console.
4. We are using the Count property (**st.count**) to get the number of items in the stack. This property will return a number. We then display this value to the console.
5. We then use the Contains method to see if the value of 3 is present in our stack. This will return either a true or false value. We then display this return value to the console.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



The screenshot shows a program window with the title bar "file:///C:/Guru99/DemoApplication/DemoApplication/bin/Debug/DemoApplic...". The output area contains the following text:

```
3  
2  
1  
The number of elements in the stack=3  
Does the stack contain the element 3=True
```

Handwritten annotations in orange boxes with arrows pointing to the output:

- A box labeled "Elements of the stack" with an arrow pointing to the numbers 3, 2, and 1.
- A box labeled "Output of count and contains method" with an arrow pointing to the two lines of text below the numbers.

From the output, we can clearly see that the elements of the stack are displayed. Also, the value of True is displayed to say that the value of 3 is defined on the stack.

Note : You have noticed that the last element pushed onto the stack is displayed first. This is the topmost element of the stack. The count of stack elements is also shown in the output.

Now let's look at the "remove" functionality. We will see the code required to remove the topmost element from the stack.


```
static void Main(string[] args)
{
    Stack st = new Stack();

    st.Push(1);
    st.Push(2);
    st.Push(3);

    st.Pop(); 1

    foreach (Object obj in st){

        Console.WriteLine(obj);

    }
}
```

Popping an
element from
the stack

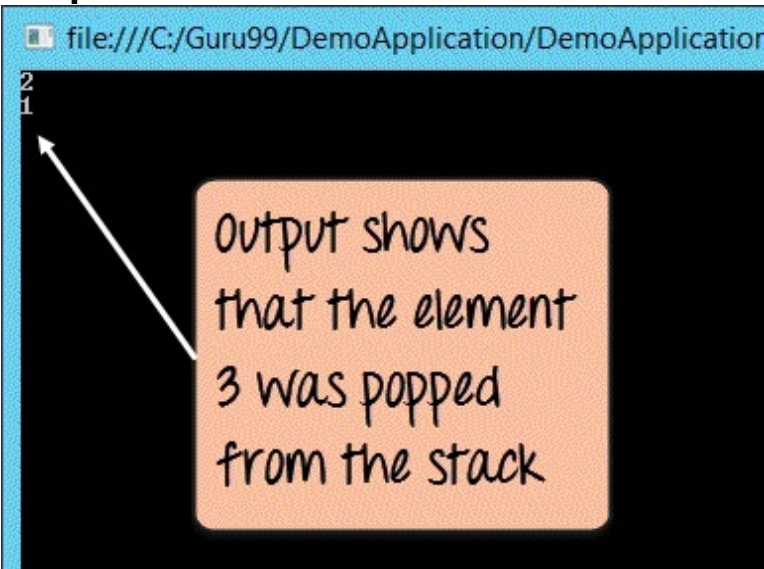
Code

Explanation:

1. Here we just issue the pop method which is used to remove an element from the stack.

If the above code is entered properly and the program is run, the following output will be displayed.

Output:



file:///C:/Guru99/DemoApplication/DemoApplication

2
1

output shows
that the element
3 was popped
from the stack

We can see that the element 3 was removed from the stack.

C# Queue

The Queue is a special case collection which represents a first in first out concept. Imagine a queue of people waiting for the bus. Normally, the first person who enters the queue will be the first person to enter the bus. Similarly, the last person to enter the queue will be the last person to enter into the bus. Elements are added to the stack, one on the top of each other.

The process of adding an element to the queue is the enqueue operation. To remove an element from a queue, you can use the dequeue operation. The operation in queues are similar to stack we saw previously.

Let's look at the operations available for the Queue collection in more detail.

Declaration of the Queue – The declaration of a Queue is provided below. A Queue is created with the help of the Queue Data type. The “new” keyword is used to create an object of a Queue. The object is then assigned to the variable qt.

Queue qt = new Queue()

Adding elements to the Queue – The enqueue method is used to add an element onto the queue. The general syntax of the statement is given below. **Queue.enqueue(element)**

Removing elements from the queue – The dequeue method is used to remove an element from the queue. The dequeue operation will return the last element of the queue. The general syntax of the statement is given below

Queue.pop()

Count – This property is used to get the number of items in the queue. Below is the general syntax of this statement.

Queue.Count

Contains - This method is used to see if an element is present in the Queue. Below is the general syntax of this statement. The statement will return true if the element exists, else it will return the value false.

Queue.Contains(element)

Now, let's see this working at a code level. All of the below-mentioned code will be written to our Console application.

The code will be written to our Program.cs file. In the below program, we will write the code to see how we can use the above-mentioned methods.

In this example, we will see how a queue gets created. Next, we will see how to display the elements of the queue, and use the Count and Contain methods.

```
static void Main(string[] args)
{
    Queue qt = new Queue();
    qt.Enqueue(1);
    qt.Enqueue(2);
    qt.Enqueue(3);
    foreach (Object obj in qt){
        Console.WriteLine(obj);
    }
    Console.WriteLine(); Console.WriteLine();
    Console.WriteLine("The number of elements in the queue" + qt.Count);
    Console.WriteLine("Does the queue contain" + qt.Contains(3));
}
```

1. Creating a queue variable

2. adding elements to the queue

3. Displaying queue elements

4. Count of queue elements

5. Searching for an element

Code Explanation:

1. The first step is used to declare the Queue. Here we are declaring qt as a variable to hold the elements of our Queue.
2. Next, we add 3 elements to our Queue. Each element is added via the "enqueue" method.
3. Now one thing that needs to be noted about Queues is that the elements cannot be accessed via the index position like the array list. We need to use a different approach to display the elements of the

Queue. So here's how we go about displaying the elements of a queue.

We first declare a temporary variable called obj. This will be used to hold each element of the Queue.

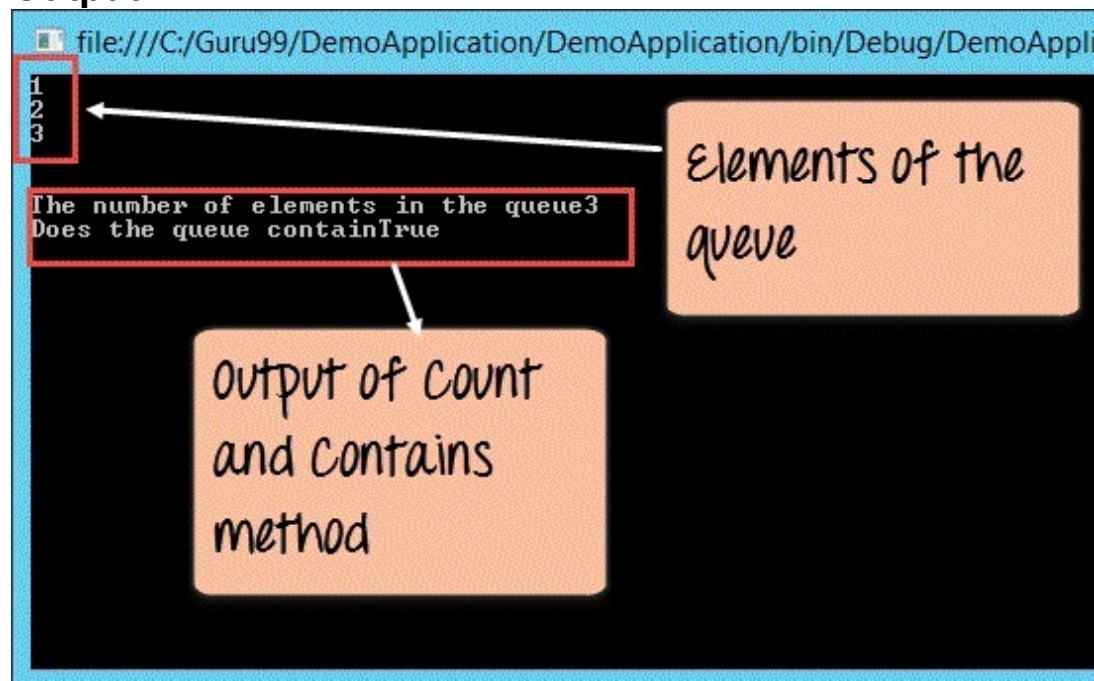
We then use the foreach statement to go through each element of the Queue. For each Queue element, the value is assigned to the obj variable. We then use the Console.WriteLine command to display the value to the console.

4. We are using the "Count" property to get the number of items in the Queue. This property will return a number. We then display this value to the console.

5. We then use the "Contains" method to see if the value of 3 is present in our Queue. This will return either a true or false value. We then display this return value to the console.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



The screenshot shows a console window with the following output:

```
1  
2  
3  
The number of elements in the queue3  
Does the queue containTrue
```

Handwritten annotations in orange boxes with arrows pointing to the output:

- A box labeled "Elements of the queue" points to the first three lines of output (1, 2, 3).
- A box labeled "Output of Count and Contains method" points to the last two lines of output ("The number of elements in the queue3" and "Does the queue containTrue").

From the output, we can clearly see that the elements of the Queue are displayed. Note that, unlike "stack" in "queue" the first element


pushed on to the queue is displayed first. The count of queue elements is also shown in the output. Also, the value of True is displayed to say that the value of 3 is defined on the queue.

Now let's look at the remove functionality. We will see the code required to remove the last element from the queue.

```
static void Main(string[] args)
{
    Queue qt = new Queue();

    qt.Enqueue(1);
    qt.Enqueue(2);
    qt.Enqueue(3);

    qt.Dequeue(); 1
    foreach (Object obj in qt){
        Console.WriteLine(obj);
    }
}
```

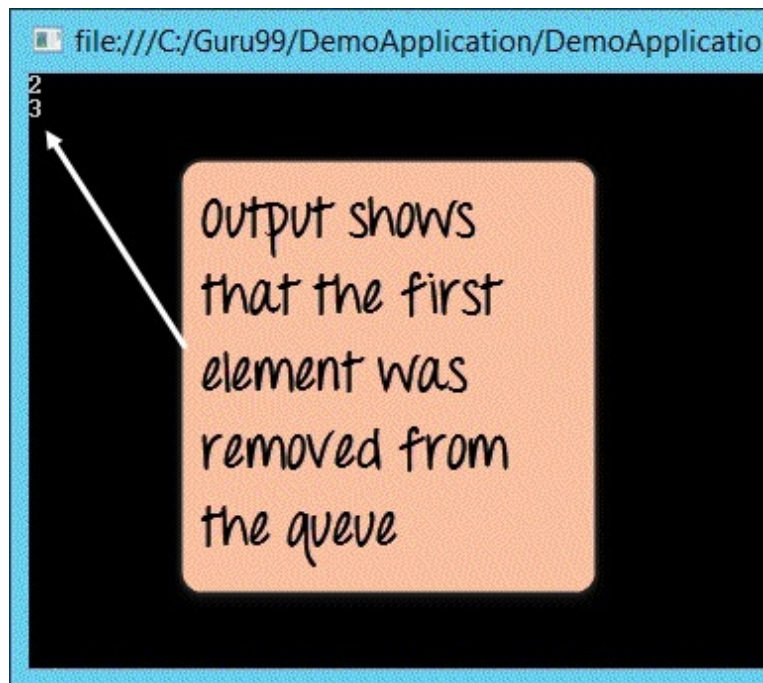


Code Explanation:

1. Here we just issue the “dequeue” method, which is used to remove an element from the queue. This method will remove the first element of the queue.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



From the output, we can see that the first element which was added to the queue, which was the element 1, was removed from the queue.

C# Hashtable

A hash table is a special collection that is used to store key-value items. So instead of storing just one value like the stack, array list and queue, the hash table stores 2 values. These 2 values form an element of the hash table.

Below are some example of how values of a hash table might look like. { "001", ".Net" }
{ "002", ".C#" }
{ "003", "ASP.Net" }

Above we have 3 key value pairs. The keys of each element are 001, 002 and 003 respectively. The values of each key value pair are ".Net", "C#" and "ASP.Net" respectively.

Let's look at the operations available for the Hashtable collection in more detail.

Declaration of the Hashtable – The declaration of a Hashtable is shown below. A Hashtable is created with the help of the Hashtable Data type. The “new” keyword is used to create an object of a Hashtable. The object is then assigned to the variable ht.

Hashtable ht = new Hashtable()

Adding elements to the Hashtable – The Add method is used to add an element on to the queue. The general syntax of the statement is given below **HashTable.add(“key”, “value”)**

Remember that each element of the hash table comprises of 2 values, one is the key, and the other is the value.

Now, let’s see this working at a code level. All of the below-mentioned code will be written to our Console application.

The code will be written to our Program.cs file. In the below program, we will write the code to see how we can use the above-mentioned methods. For now in our example, we will just look at how we can create a hashtable , add elements to the hashtable and display them accordingly.

1. First, we declare the hashtable variable using the Hashtable data type by using keyword “New.” The name of the variable defines is ‘ht’.
2. We then add elements to the hash table using the Add method. Remember that we need to add both a key and value element when adding something to the hashtable.
3. There is no direct way to display the elements of a hash table.

In order to display the hashtable , we first need to get the list of keys (001, 002 and 003) from the hash table.

This is done via the ICollection interface. This is a special data type which can be used to store the keys of a hashtable collections. We then assign the keys of the hashtable collection to the variable ‘keys’.

4. Next for each key value, we get the associated value in the hashtable by using the statement ht[k].

If the above code is entered properly and the program is run the following output will be displayed.

Let’s look at some more methods available for hash tables.

ContainsKey - This method is used to see if a key is present in the Hashtable. Below is the general syntax of this statement. The statement will return true if the key exists, else it will return the value false.

Hashtable.Containskey(key)

ContainsValue - This method is used to see if a Value is present in the Hashtable. Below is the general syntax of this statement. The statement will return true if the Value exists, else it will return the value false.

Hashtable.ContainsValue(key) Let's change the code in our Console application to showcase how we can use the "Containskey" and "ContainsValue" method.

1. First, we use the ContainsKey method to see if the key is present in the hashtable. This method will return true if the key is present in the hashtable. This method should return true since the key does exist in the hashtable.

2. We then use the ContainsValue method to see if the value is present in the hashtable. This method will return 'true' since the Value does exist in the hashtable.

If the above code is entered properly and the program is run the following output will be displayed.

From the output, you can clearly see that both the key and value being searched are present in the hash table.

Summary

The Array List collection is used to store a group of elements. The advantage of the Array list collection is that it is dynamic in nature. You can add and remove elements on the fly to the array list collection.

A Stack is based on the last in first out concept. The operation of adding an element to the stack is called the push operation. The operation of removing an element to the stack is called the pop

operation.

A Queue is based on the first in first out concept. The operation of adding an element to the queue is called the enqueue operation. The operation of removing an element to the queue is called the dequeue operation. A Hashtable is used to store elements which comprises of key values pairs. To access the value of an element , you need to know the key of the element.