Collection in C

Collection classes are special classes for storing and retrieving data. These classes provide support for Stack, Queue, List, and Hash Table. Most Collection classes in C # deploy the same Interface.

Collection classes are special classes for storing and retrieving data. These classes provide support for Stack, Queue, List, and Hash Table. Most Collection classes in C # deploy the same Interface.

In C #, Collection classes serve diverse purposes, such as dynamically allocating memory for elements and accessing a list of items based on an index . These classes create a collection of objects of Object class, which is the base class for all data types in C #.

Note : Stack: stack, push: add new node to top (top) stack, pop: operation takes 1 element from top of stack.

Main content of the lesson:

Collection classes and their usage in C

The table below lists the commonly used classes of System.Collection namespace.

Description class and usage of ArrayList in C #

It represents a sorted set of an object that can be **indexed** for each item individually.

Basically, it is an alternative to an array. However, unlike in arrays, you can add and remove items from a list at a specified location using an index and the array itself can automatically resize itself. It also allows to allocate dynamic memory, add, search and sort items in a list.

Hashtable in C #
It uses a **key-value** pair to access elements in this collection.

A Hash Table is used when you need to access elements using a key, and you can identify a useful key value. Each item in the Hash Table has a pair of **key / value** . Key is used to access items in this collection form.

SortedList in C #
It uses a **key** as well as an **index** to access items in a list.

A sorted list is a combination of an array and a Hash Table. It contains a list of items that can be accessed using a key or an index. If you access the item using an index, it is an ArrayList, and if you access the item using a key, it is a HashTable. The set of items is always sorted by key value

Stack in C #
It represents a set of **Last-in, First-out** of objects. It is used when you need access to Last-in, First-out items. When you add an item to the list, it is called **pushing** and when you remove an item, it is called **popping** .

Queue in C #

It represents a First-in, First-out set of objects.

It is used when you need access to First-in, First-out items. When you add an item to the list, it is called enqueue and when you remove an item, it is called deque.

BitArray in C #
It represents an array in **binary representation** using values ??1 and 0. It is used when you need to store the Bit but not know the number of Bit. You can access items from the BitArray collection using an integer index, which starts at 0.

ArrayList in C

ArrayList in C # represents a sorted set of an object that can be indexed for each item separately. Basically, it is an alternative to an array. However, unlike in arrays, you can add and remove items from a list at a specified location using an index and an array that automatically resizes. It also allows to allocate dynamic memory, add, search and sort items in a list.

Methods and Properties of the ArrayList class in C #:

Here are the commonly used properties of the ArrayList class in C #:

Attribute DescriptionCapacity Retrieve or set the number of elements that ArrayList can contain. Count Get the actual number of elements contained in the ArrayList. IsFixedSize Get a value indicating whether or not ArrayList is a fixed size. IsReadOnly Get a value indicating that whether or not ArrayList is read-only. Item Retrieves or sets the element at the specified index.
Here are the commonly used **methods** of the **ArrayList** class in C #:

1. **public virtual int Add (object value);** : Add an object to the end of the ArrayList.
2. **public virtual void AddRange (ICollection c);** : Add elements of an ICollection to the end of the ArrayList.
3. **public virtual void Clear ();** : Remove elements from that ArrayList.

4. **public virtual bool Contains (object item);** : Determine whether or not an element is in the ArrayList.
5. **public virtual ArrayList GetRange (index int, int count);** : Returns an ArrayList that represents a subset of the elements in the source ArrayList.
6. **public virtual int IndexOf (object);** : Returns the index (based on 0) of the first occurrence of a value in the ArrayList or part of it.
7. **public virtual void Insert (int index, object value);** : Insert an element into ArrayList at the specified index.
8. **public virtual void InsertRange (int index, ICollection c);** : Insert an element of a collection into ArrayList at the specified index.
9. **public virtual void Remove (object obj);** : Remove the first occurrence of a specified object at the specified index.
10. **public virtual void RemoveAt (int index);** : Removing the element at the specified index of ArrayList.
11. **public virtual void RemoveRange (index int, int count);** : Remove an array of elements from ArrayList.
12. **public virtual void Reverse ();** : Reverse the element order in ArrayList.
13. **public virtual void SetRange (int index, ICollection c);** : Copy the elements of a collection through a sequence of elements in the ArrayList.
14. **public virtual void Sort ();** : Arrange elements in ArrayList.
15. **public virtual void TrimToSize ();** : Set capacity to the actual number of elements in ArrayList.

For example:

```
 using System ; using System . Collections ; namespace QTMCsharp { class Tester {
? minh h?
a ArrayList trong C#" ); Console . WriteLine ( "--------------------------------
?t vài ph?n t?
 vào ArrayList: " );  al . Add (50 );  al . Add ( 93 );  al . Add ( 85 );  al . A
 al . Add ( 99 ); Console . WriteLine ( "nDung l??ng c?
a ArrayList: {0} " ,  al . Capacity ); Console . WriteLine ( "S? ph?n t?
 trong ArrayList: {0}" ,  al . Count ); Console . WriteLine ( "nHi?n th?
 các ph?n t?
 có trong ArrayList: " ); foreach ( int  i  in  al ) { Console . Write ( i  + " "
?p x?p ArrayList r?
i in: " );  al . Sort (); foreach ( int  i  in  al ) { Console . Write ( i  + " "
```

When running the above code, you get the following result:

```
  Example illustrating ArrayList in C #
  ---------------------------------

  Insert some elements into the ArrayList:

  Capacity of ArrayList: 8
  Element number in ArrayList: 8

  Show elements contained in ArrayList:
  50 93 85 96 82 25 90 99

  Sort ArrayList and print:
  25 50 82 85 90 93 96 99
```

Hashtable in C

The Hashtable class in C # uses a key-value pair to access elements in this collection. Hash Table is used when you need access to elements using a key, and you can identify a useful key value. Each item in Hash Table has a key and value pair. Key is used to access items in this collection form.

Methods and properties of the Hashtable class in C #:

The table below lists the commonly used properties of the Hashtable class in C #:

Attribute DescriptionCount Get the number of key pairs and values ??contained in the hashtable. IsFixedSize Get a value indicating whether or not Hashtable has a fixed size. IsReadOnly Get a value indicating whether or not Hashtable is read-only. Item Retrieves or sets the value associated with the specified key. Keys Get an ICollection containing the key in Hashtable. Values ??Get an ICollection containing the values ??in the Hashtable.
List of commonly used methods of HashTablet class in C #:

1. **public virtual void Add (object key, object value);** Add an element with the key and value specified in the hashtable.
2. **public virtual void Clear ();** Remove all elements from Hashtable.
3. **public virtual bool ContainsKey (object key);** Determining whether or not the Hashtable contains a specific key.
4. **public virtual bool ContainsValue (object value);** Determining whether or not Hashtable is containing a specific value.
5. **public virtual void Remove (object key);** Remove the element with the key identified from Hashtable.

Example of Hashtable in C #:

```
 using System ; using System . Collections ; namespace QTMCSharp { class Tester 
?
c Anh" );  ht . Add ( "003" , "Mai Anh" );  ht . Add ( "004" , "Trâm Anh" );  ht
?
 Anh" );  ht . Add ( "006" , "Lan Anh" );  ht . Add ( "007" , "Minh Anh" ); ht .
 if ( ht . ContainsValue ( "Ngân Anh" )) { Console . WriteLine ( "Tên h?
c sinh này ?
ã có trong danh sách" ); } else {  ht . Add ( "009" , "Ngân Anh" ); } // L?
y t?p h?
p các key. ICollection  key  =  ht . Keys ; foreach ( string  k  in  key ) { Cons
```

Running the above program you will get the following result:

```
  007: Minh Anh
  004: Tram Anh
  005: The Anh
  008: Linh Anh
  009: Ngan Anh
  002: Ngoc Anh
  003: Mai Anh
  001: Hoang Anh
  006: Lan Anh
```

SortedList in C

The SortedList class in C # uses a key as well as an index to access items in a list.

A sorted list is a combination of an array and a Hash Table. It contains a list of items that can be accessed using a key or an index. If you access the item with the index, it is an ArrayList, and if you access the item by key, it is a HashTable. The set of items is always sorted by key value.

Methods and properties of SortedList class in C #:

These are the commonly used properties of SortedList class in C #:

1. Capacity: Get or set the capacity of SortedList.
2. Count: Get the number of elements contained in SortedList.
3. IsFixedSize: Get a value indicating whether or not SortedList has a fixed size.
4. IsReadOnly: Get a value indicating whether or not SortedList is read-only.
5. Item: Get and set the value associated with a specific key in SortedList.
6. Keys: Get the keys in SortedList.
7. Values: Get the values ??in SortedList.

Commonly used methods of SortedList class in C #:

1. **public virtual void Add (object key, object value);** Add an element with the specified key and value in SortedList.
2. **public virtual void Clear ();** Remove all elements from SortedList.
3. **public virtual bool ContainsKey (object key);** Determine whether or not SortedList contains a specific key.
4. **public virtual bool ContainsValue (object value);** Determine whether or not SortedList contains a specific value.
5. **public virtual object GetByIndex (int index);** Get the value at the specified index of SortedList.
6. **public virtual object GetKey (int index);** Get the key at the specified index of SortedList.
7. **public virtual IList GetKeyList ();** Get the keys in SortedList.
8. **public virtual IList GetValueList ();** Get the values ??in SortedList.
9. **public virtual int IndexOfKey (object key);** Returns the index (based on 0) of the specified key in SortedList.
10. **public virtual int IndexOfValue (object value);** Returns the index (based on 0) of the first occurrence of the specified value in SortedList.
11. **public virtual void Remove (object key);** Remove the element with the specified key from SortedList.
12. **public virtual void RemoveAt (int index);** Remove the element at the specified index of SortedList.
13. **public virtual void TrimToSize ();** Set capacity to the actual number of elements in SortedList.

Example of SortedList in C #:

```
 using System ; using System . Collections ; namespace CollectionsApplication { c
?
c Anh" );  sl . Add ( "003" , "Mai Anh" );  sl . Add ( "004" , "Trâm Anh" );  sl
?
 Anh" );  sl . Add ( "006" , "Lan Anh" );  sl . Add ( "007" , "Minh Anh" ); sl .
 if ( sl . ContainsValue ( "Ngân Anh" )) { Console . WriteLine ( "Tên h?
c sinh này ?
ã có trong danh sách" ); } else {  sl . Add ( "009" , "Ngân Anh" ); } // L?
y t?p h?
```

```
p các key.  ICollection  key  =  sl . Keys ; foreach ( string  k  in  key ) { Co
```

Running the above code we get the result:

```
001: Hoang Anh
002: Ngoc Anh
003: Mai Anh
004: Tram Anh
005: The Anh
006: Lan Anh
007: Minh Anh
008: Linh Anh
009: Ngan Anh
```

Stack in C

The Stack class in C # represents a Last-in, First-out set of objects. It is used when you need access to Last-in, First-out items. When you add an item to the list, it is called pushing and when you remove an item, it is called popping.

Methods and properties of the Stack class in C #:

Stack feature in C #:

1. **Count:** Get the number of elements in the Stack.

Commonly used methods of the Stack class in C #:

1. **public virtual void Clear ();** Remove all elements from Stack.
2. **public virtual bool Contains (object obj);** Remove all elements from Stack.
3. **public virtual object Peek ();** Returns an object at the top of the Stack without removing it.
4. **public virtual object Pop ();** Remove and return the object at the top of the Stack.
5. **public virtual void Push (object obj);** Insert an object at the top of the Stack.
6. **public virtual object [] ToArray ();** Copy Stack to a new array.

Example of Stack in C #:

```
 using System ; using System . Collections ; namespace CollectionsApplication { c
?n t?
i: " ); foreach ( char  c  in  st ) { Console . Write ( c  + " " ); } Console . W
? có th? pop ti?
p theo trong stack: {0}" ,  st . Peek ()); Console . WriteLine ( "2 - Stack hi
?n t?
i: " ); foreach ( char  c  in  st ) { Console . Write ( c  + " " ); } Console . W
?
." );  st . Pop ();  st . Pop ();  st . Pop (); Console . WriteLine ( "3 - Stack
?n t?
i: " ); foreach ( char  c  in  st ) { Console . Write ( c  + " " ); } } } }
```

Results when running the above code:

```
1 - Stack current:
WQTM
The value can pop next in the stack: H
2 - Stack current:
HVWQTM
Delete values.
3 - Stack current:
QTM
```

Queue in C

The Queue class in C # represents a First-in, First-out collection of objects. It is used when you need access to First-in, First-out items. When you add an item to the list, it is called enqueue and when you remove an item, it is called deque.

Method and properties of Queue class in C #:

The commonly used feature of the Queue class in C # is count, which takes the number of elements in the queue.

Commonly used methods of Queue class in C # include:

1. **public virtual void Clear ();** Remove all elements from Queue.
2. **public virtual bool Contains (object obj);** Determine whether or not an element is in Queue.
3. **public virtual object Dequeue ();** Remove and return the object at the start of Queue.
4. **public virtual void Enqueue (object obj);** Add an element to the end of Queue.
5. **public virtual object [] ToArray ();** Copy Queue to a new array.
6. **public virtual void TrimToSize ();** Set capacity to the actual number of elements in Queue.

Example of Queue in C #:

```
 using System ; using System . Collections ; namespace CollectionsApplication { c
?n t?
i: " ); foreach ( char  c  in  q ) Console . Write ( c  + " " ); Console . WriteL
?n t?
i: " ); foreach ( char  c  in  q ) Console . Write ( c  + " " ); Console . WriteL
?!" ); char  ch  = ( char ) q . Dequeue (); Console . WriteLine ( "Giá tr?
?
ã xóa: {0}" ,  ch );  ch  = ( char ) q . Dequeue (); Console . WriteLine ( "Giá t
? ?ã xóa: {0}" ,  ch ); Console . ReadKey (); } } }
```

The result when running the above code will be:

```
1 - Current Queue:
QTMW
2 - Current Queue:
QTMWVH
Delete some values!
Deleted value: Q
Deleted value: T
```

BitArray in C

The BitArray class in C # manages a small array of bit values, represented in binary form, with true that the bit is turned on (1) and false indicating the bit is turned off (0). It is used when you need to store the Bit but does not know how many Bit. You can access items from the BitArray collection using an integer index, starting at 0.

Methods and Properties of the BitArray class in C #:

List of commonly used properties of BitArray class in C #:

1. **Count:** Get the number of elements contained in BitArray.
2. **IsReadOnly:** Get a value indicating whether or not BitArray is read-only.
3. **Item:** Get or set the value of the bit at the specified location in BitArray.
4. **Length:** Get or set the number of elements in BitArray.

The table below lists the commonly used methods of the BitArray class in C #:

1. **public BitArray And (BitArray value);** Perform bitwise AND operation on the elements in the current BitArray with the corresponding elements in the specified BitArray.
2. **public bool Get (int index);** Take the value of the bit at a specified location in BitArray.
3. **public BitArray Not ();** Reverse all bit values ??in the current BitArray, so that the elements set to true are converted to false, and the elements set to false are converted to true.
4. **public BitArray Or (BitArray value);** Perform bitwise OR operation on the elements in the current BitArray with the corresponding elements in the BitArray defined.
5. **public void Set (index int, bool value);** Set the bit at a specified position in BitArray to the given value.
6. **public void SetAll (bool value);** Set all bits in BitArray to the given value.
7. **public BitArray Xor (BitArray value);** Perform bitwise operation OR on the elements in the current BitArray with the corresponding elements in the specified BitArray.

Example of BitArray in C #:

```
 using System ; using System . Collections ; namespace Quantrimangcom { class Pr
?o 2 m?ng bit có kích th??
c là 8 BitArray  ba1  = new BitArray ( 8 ); BitArray  ba2  = new BitArray ( 8 );
?u tr? giá tr? 60 và 13 trong các m?
ng  ba1  = new BitArray ( a );  ba2  = new BitArray ( b ); //Hi?n th? n?
i dung c?a m?ng ba1 Console . WriteLine ( "Ph?n t? c?
a Bit Array ba1: 60" ); for ( int  i  = 0 ;  i    ba1 . Count ;  i ++) { Console
?n th? n?i dung c?a m?ng ba2 Console . WriteLine ( "Ph?n t? c?
a Bit Array ba2: 13" ); for ( int  i  = 0 ;  i    ba2 . Count ;  i ++) { Console
?n th? n?i dung c?a m?
ng ba3 Console . WriteLine ( "Bit Array ba3 sau khi th?c hi?
n AND: 12" ); for ( int  i  = 0 ;  i    ba3 . Count ;  i ++) { Console . Write (
?c hi?
n OR: 61" ); for ( int  i  = 0 ;  i    ba3 . Count ;  i ++) { Console . Write ( "
```

After running the above code we can:

```
  Element of Bit Array three: 60
  False False True True True True False False
  Element of Bit Array three: 13
  False True False True False False False
  Bit Array ba3 after executing AND: 12
  False False True False False False True False
```

```
Bit Array ba3 after executing OR: 61
False True False True False False False
```

According to Tutorialspoint

Previous article: Event (Event) in C #

Next article: Generic in C #