Collections

Like an array, a collection can hold one or more elements. Unlike arrays, collections don't have a fixed size. Instead, the size of a collection is increased automatically when elements are added to it. In addition, most types of collections provide methods that you can use to change the capacity of a collection. As a result, collections usually work better than arrays when you need to work a varying number of elements. There are mainly five types of collections: lists, sorted lists, queues, stacks, and array lists; apart from other collections provided by .NET Framework.

* A collection is an object that can hold one or more elements.
* The collection classes in the System.Collection.Generic namespace use a feature known as generics to allow you to create typed collections that can only store the specified type.
* The collection classes in the System.Collection namespace allow you to create untyped

Scanned by CamScanner

collections. With an untyped collection, you can store any type of object in the collection.

* The set of parantheses after a class name indicates that it is a typed collection

How arrays and collections are similar

* Both can store multiple elements, which can be value types or reference types.

How arrays and collections are different

* An array is a feature of the Visual Basic language that inherits the Array class. Collections are other classes in the .NET Framework.
* Collection classes provide methods to perform operations that arrays don't provide.
* Arrays are fixed in size. Collections are variable in size.

typed untyped

Commonly used collection classes

.NET 2.0 to 4.0	.NET 1.x	Description
List()	ArrayList	Uses an index to access each element. This class is efficient for accessing elements sequentially, but inefficient for inserting elements into a list.

Scanned by CamScanner

SortedList()	SortedList	Uses a key to access a value, which can be any type of object. This class can be inefficient for accecing elements sequentially, but it is efficient for inserting elements into a list.
Queue()	Queue	Uses special methods to add and remove elements.
Stack()	Stack	Uses special methods to add and remove elements.

Example of Untyped collections

```
Dim numbers As New ArrayList
numbers.Add(3)
numbers.Add(70)
numbers.Add("Test")  ' will compile but causes an exception
Dim sum As Integer = 0
Dim number As Integer
For i As Integer = 0 To numbers.Count-1
    number = CInt(numbers(i))  'cast is required
    sum += number
Next
```

Scanned by CamScanner

Example of Typed collections

```
Dim numbers As New List(Of Integers)
numbers.Add(3)
numbers.Add(70)
'numbers.Add("Test") ' won't compile - prevents runtime error
Dim sum As Integer = 0
Dim number As Integer
for i As Integer = 0 To numbers.Count - 1
    number = numbers(i)  'no cast is required
    sum += number
Next
```

* Typed collections have two advantages over untyped collections. First, they check the type of each element at compile-time and prevent runtime errors from occuring. Second, they reduce the amount of casting that's needed when retrieving objects.
* Untyped collections are part of the System.Collections namespace, while typed collections are part of the System.Collections.Generic namespace.

Scanned by CamScanner

List

* A list is a collection that automatically adjusts its capacity to accommodate new elements.
* The default capacity of a list is 16 elements, but you can specify a different capacity when you create a list. When the number of elements in a list exceeds its capacity, the capacity is automatically doubled.

```
'create a list of String elements
Dim titles As New List(Of String)

'create list of Decimal elements
Dim prices As New List(Of Decimal)

'create a list of strings with a capacity of 3
Dim lastNames As New List(Of String)(3)
```

Common properties and methods of the List() Class -

Property	Description
Item(index)	Gets or sets the element at the specified index. The index for the first item in a list is 0. Since Item is default property, its name can be omitted

Scanned by CamScanner

Capacity	Gets or sets the number of elements the list can hold.
Count	Gets the number of elements in the list.

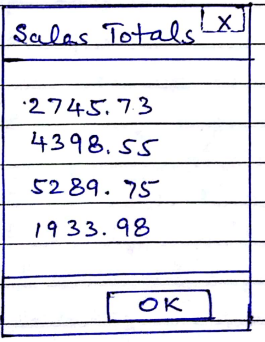
Method	Description
Add(Object)	Adds an element to the end of a list and returns the element's index.
Clear()	Removes all elements from the list and sets its Count property to zero.
Contains(Object)	Returns a Boolean value that indicates if the list contains the specified object.
Insert(index, object)	Inserts an element into a list at the specified index.
Remove(Object)	Removes the first object occurrence of the specified object
RemoveAt(index)	Removes the element at the specified index of a list.
BinarySearch(Object)	Searches a list for a specified object and returns the index for that object.
Sort()	Sorts the elements in a list into ascending order.

Scanned by CamScanner

```
' code that causes the size of a list of
' names to be increased.

Dim lastNames As New List (Of String) (3)
lastNames.Add ("Ram")
LastNames.Add ("Shyam")
lastNames.Add ("Mohan")
lastNames.Add("Sita")  ' capacity is doubled to 6 elements
lastNames.Add("Geeta")
lastNames.Add("Reeta")
lastNames.Add("Hari")  ' capacity is doubled
                          to 12 elements

' Syntax for retrieving a value from a list
  listName[.Item](index)

' Create a list that holds Decimal values using
  collection initializer to assign values to list
Dim salesTotal As New List (Of Decimal) From
  {3275.68D, 4398.55D, 5289.75D, 1933.98D}

' Retrieve the first value from the list
Dim sales1 As Decimal = salesTotals(0)
```

Scanned by CamScanner

```
'Insert and remove an element from the list
salesTotals.Insert(0, 2745.73D) 'Insert a new element
sales1 = salesTotals(0)                    '2745.73
Dim sales2 As Decimal = salesTotals(1)     '3275.68
salesTotals.RemoveAt(1)                    'remove 2nd element
sales2 = salesTotals(1)                    '4398.55

'Display the list in a message box
Dim salesTotalString As String = ""
For Each d As Decimal In salesTotals
    salesTotalString &= d.ToString & vbCrLf
Next
MessageBox.Show(salesTotalString, "Sales Totals")
```



Scanned by CamScanner

```
'Check for an element in the list and remove it if
 it exists
Dim x As Decimal = 2745.73D
If salesTotals.Contains(x) Then
    salesTotals.Remove(x)
End If

'Sort and search the list
                          'In Array sort is static method
salesTotals.Sort()        'Sort is instance method
Dim sales2Index As Integer = salesTotals.BinarySearch
                             (sales2)
```

Scanned by CamScanner

Sorted List

We can implement a collection by using the SortedList() class. A sorted list is useful when you need to look up values in the list based on a key value. Eg: A sorted list consists of item numbers and unit prices, the keys are the item numbers. Then, the list can be used to look up the unit price for any item number.

Each item in a sorted list is actually a KeyValuePair structure that consists of two properties: Key and Value. Here, the Value property can store value types or reference types.

Like a list, you can set the initial capacity of a sorted list by specifying the number of elements in the parantheses when the list is created. Then, if the number of elements in the list exceeds the capacity as the program executes, the capacity is doubled.

Since using a sorted list makes it easy to look up a key and return its corresponding value, this is the right type of collection to use when you need to do that type of lookup. A sorted list is also the right choice when you need to keep the elements in sequence key.

Scanned by CamScanner

Common properties and methods of the SortedList() class:

Property	Description
Item(key)	Gets or sets the value of the element with the specified key. Since Item is the default property, its name can be omitted.
Keys	Gets a collection that contains the keys in the list.
Values	Gets a collection that contains the values in the list.
Capacity	Gets or sets the number of elements the list can hold.
Count	Gets the number of elements in the list.

Method	Description
Add(key, value)	Adds an element with the specified key and value to the sorted list.
Clear()	Removes all elements from the sorted list.
ContainsKey(key)	Returns a Boolean value that indicates whether or not the sorted list contains the specified key.
ContainsValue(value)	Returns a Boolean value that indicates whether or not the sorted list contains the specified value.
Remove(key)	Removes the element with the specified key from the sorted list.
RemoveAt(index)	Removes the element at the specified index from the sorted list.

Scanned by CamScanner

Properties of the KeyValuePair structure

Property	Description
Key	The key for the SortedList item.
Value	The value associated with the key.

```
'Create and load a sorted list
Dim salesList As New SortedList(Of String, Decimal)
salesList.Add("Adams", 3274.68D)
salesList.Add("Finkle", 4398.55D)
salesList.Add("Lewis", 5289.75D)
salesList.Add("Potter", 1933.97D)

'Look up a value in the sorted list based on a key
Dim employeeKey As String = "Lewis"
Dim salesTotal As Decimal = salesList(employeeKey)

'Convert the sorted list to a tab-delimited string
Dim salesTableString As String = ""
For Each employeeSalesEntry As KeyValuePair(Of String, Decimal)
        In salesList
    salesTableString &= employeeSalesEntry.Key & vbTab &
        employeeSalesEntry.Value & vbCrLf
Next
MessageBox.Show(salesTableString, "Sorted List Totals")
```

Scanned by CamScanner

Queues and Stack

A queue is first-in, first-out (FIFO) collection because its items are retrieved in the same order in which they were added.

A stack is last-in, first-out (LIFO) collection because its items are retrieved in the reverse order from the order in which they were added.

Properties and methods of the Queue() class

Property	Description
Count	Gets the number of items in the queue.

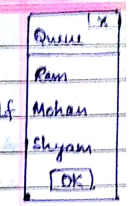
Method	Description
Enqueue(object)	Adds the specified object to the end of the queue
Dequeue()	Gets the object at the front of the queue and removes it from the queue.
Clear()	Removes all items from the queue.
Peek()	Retrieves the next item in the queue without deleting it.

Example of Queue

```
Dim nameQueue As New Queue (Of String)
nameQueue.Enqueue ("Ram")
nameQueue.Enqueue ("Mohan")
nameQueue.Enqueue ("Shyam")
```

Scanned by CamScanner

```
Dim nameQueueString As String = ""
Do While nameQueue.Count > 0
    nameQueueString &= nameQueue.Dequeue & vbCrLf
Loop
MessageBox.Show(nameQueueString, "Queue")
```

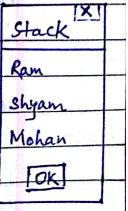


Properties and methods of the Stack() class

Property	Description
Count	Gets the number of items in the stack.

Method	Description
Push(Object)	Adds the specified object to the top of the stack.
Pop()	Gets the object at the top of the stack and removes it from the stack.
Clear()	Removes all items from the stack.
Peek()	Retrieves the next item in the stack without deleting it.

```
'Example of stack
Dim nameStack As New Stack(Of String)
nameStack.Push("Ram")
nameStack.Push("Shyam")
nameStack.Push("Mohan")
Dim nameStackString As String = ""
Do While nameStack.Count > 0
    nameStackString &= nameStack.Pop & vbCrLf
Loop
MessageBox.Show(nameStackString, "Stack")
```



Scanned by CamScanner

Array List

Array list is the most common untyped collection. An array list works like a list. However, since an array list defines an untyped collection, there are few differences.

When you declare an Arraylist class, you don't define the type within parantheses. Instead, each element in the array list is stored as an Object type. As a result, any value type that you store in the array list must be converted to a reference type. To do that, an object is created and the value is stored in that object. The process of putting a value in an object is known as **boxing**, and it's done automatically whenever a value type needs to be converted to a reference type.

When you retrieve an element from an array, you must cast the Object type to the appropriate data type. The process of getting a value out of the Object is known as **unboxing**.

```
'Create an array list that holds decimal values
.Dim salesTotals As New Arraylist from
    {3275.68D, 4398.55D, 5289.75D, 1933.98D}
'Retrieve the first value from the array list
Dim sales1 As Decimal = CDec(salesTotals(0))
```

Scanned by CamScanner

```
' Insert and remove an element from the array list
salesTotals.Insert(0, 2745.73D)
sales1 = CDec(salesTotals(0))
Dim sales2 As Decimal = CDec(salesTotals(1))
salesTotals.RemoveAt(1)
sales2 = CDec(salesTotals(1))

' Display the array list
Dim salesTotalString As String = ""
For Each d As Decimal In salesTotals
    salesTotalString &= d.ToString & vbCrLf
Next
MessageBox.Show(salesTotalString, "Sales Total")

' Check for an element in the array list and removes it if it exists
Dim x As Decimal = 2745.73D
If salesTotals.Contains(x) Then
    salesTotals.Remove(x)
End If

' Sort and search the array list
salesTotals.Sort()
Dim salesIndex As Integer = salesTotals.BinarySearch(sales2)
```

Scanned by CamScanner