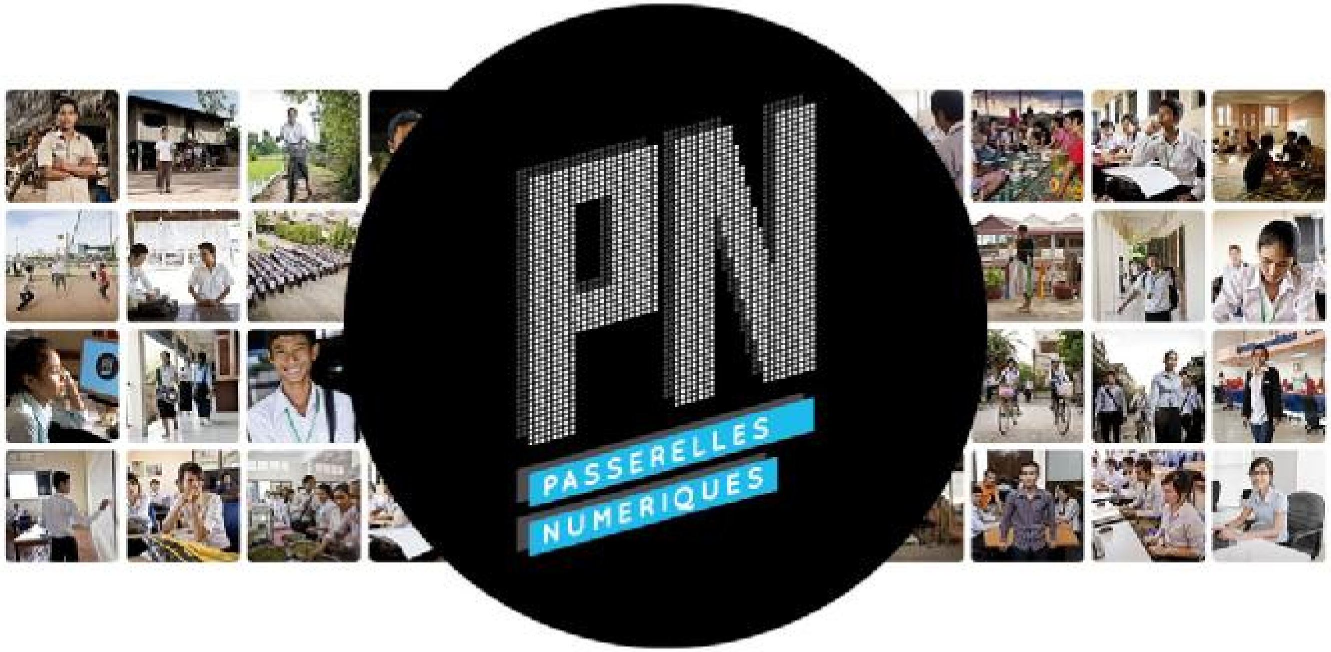


Microsoft®
.NET

UNIT 6. COLLECTIONS & GENERICS

- Lesson 1. ArrayList class
- Lesson 2. Hastable class
- Lesson 3. SortedList class

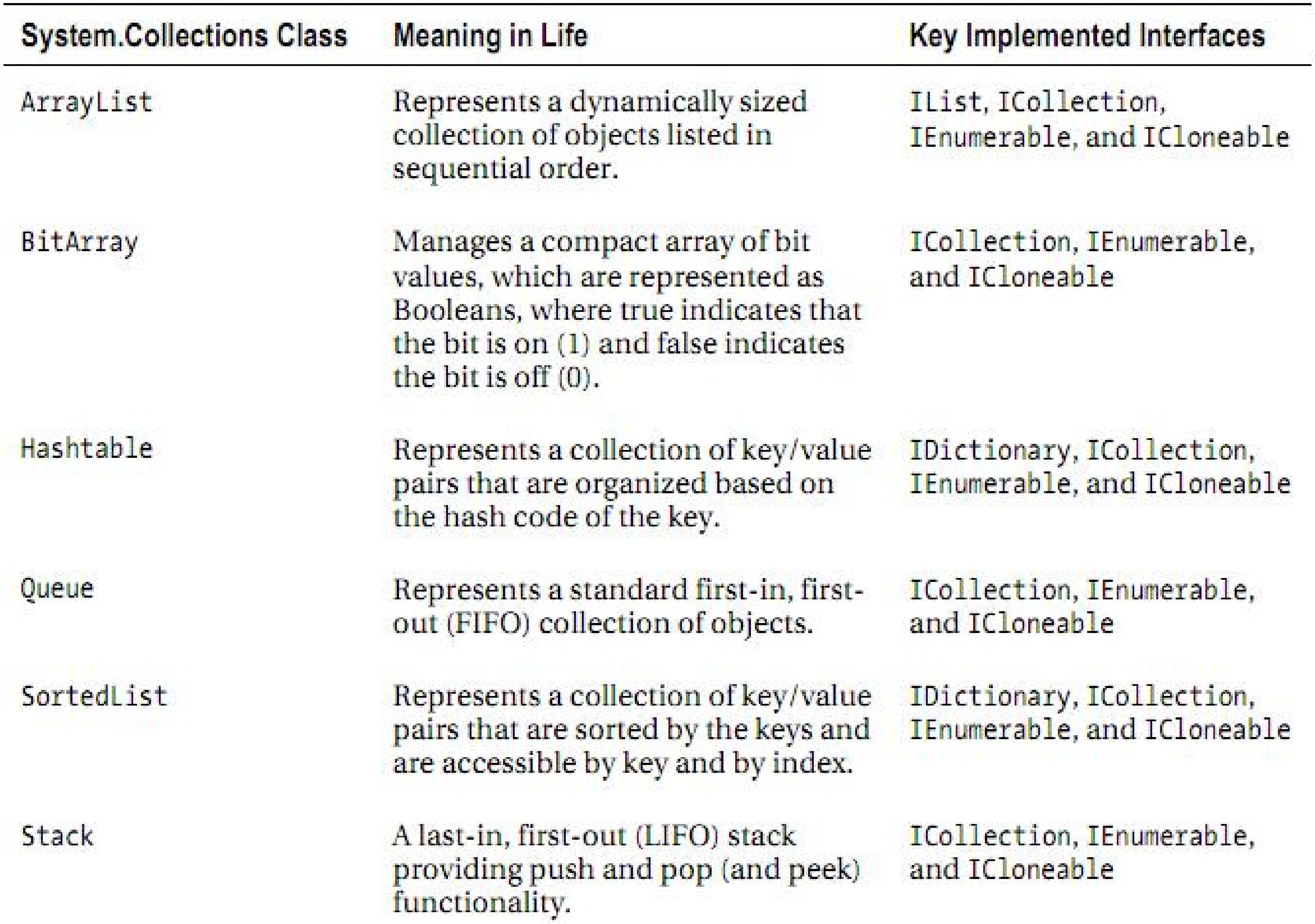
INTRODUCTION

- To help overcome the limitations of a simple array
- collection classes are built to dynamically resize themselves on the fly as you insert or remove item.
- Many of the collection classes offer increased type safety and are highly optimized to process the contained data in a memory-efficient manner.

INTRODUCTION

- A collection class can belong to one of two broad categories:
  - Nongeneric collections (primarily found in the System.Collections namespace)
  - Generic collections (primarily found in the System.Collections.Generic namespace)

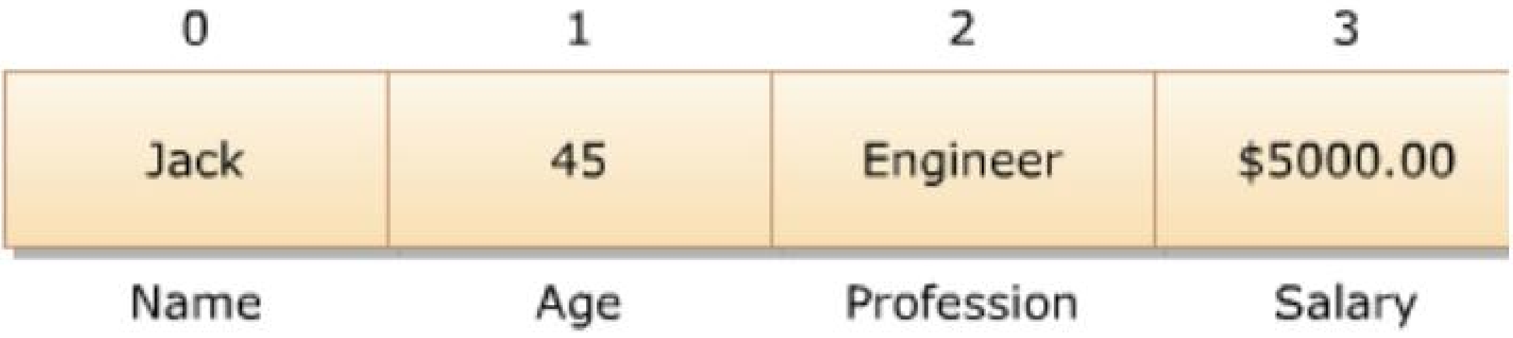
SOME USEFUL COLLECTIONS



System.Collections Class	Meaning in Life	Key Implemented Interfaces
ArrayList	Represents a dynamically sized collection of objects listed in sequential order.	IList, ICollection, IEnumerable, and ICloneable
BitArray	Manages a compact array of bit values, which are represented as Booleans, where true indicates that the bit is on (1) and false indicates the bit is off (0).	ICollection, IEnumerable, and ICloneable
Hashtable	Represents a collection of key/value pairs that are organized based on the hash code of the key.	IDictionary, ICollection, IEnumerable, and ICloneable
Queue	Represents a standard first-in, first-out (FIFO) collection of objects.	ICollection, IEnumerable, and ICloneable
SortedList	Represents a collection of key/value pairs that are sorted by the keys and are accessible by key and by index.	IDictionary, ICollection, IEnumerable, and ICloneable
Stack	A last-in, first-out (LIFO) stack providing push and pop (and peek) functionality.	ICollection, IEnumerable, and ICloneable

Introduction

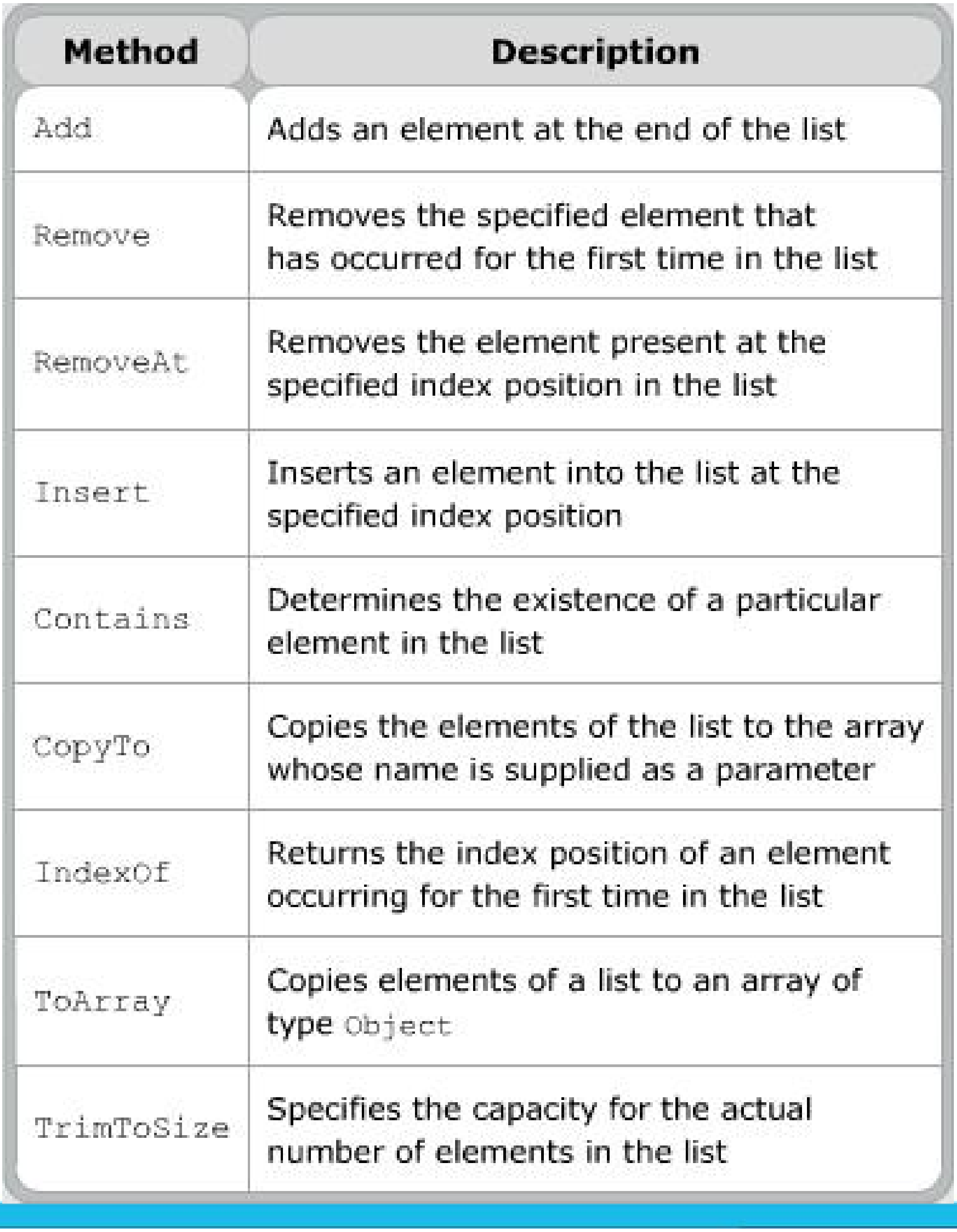
- The ArrayLisc class is a variable-length array that can dynamically increase or decrease in size.
- Unlike the Array class, this class can store elements of different data types.



0	1	2	3
Jack	45	Engineer	$5000.00
Name	Age	Profession	Salary

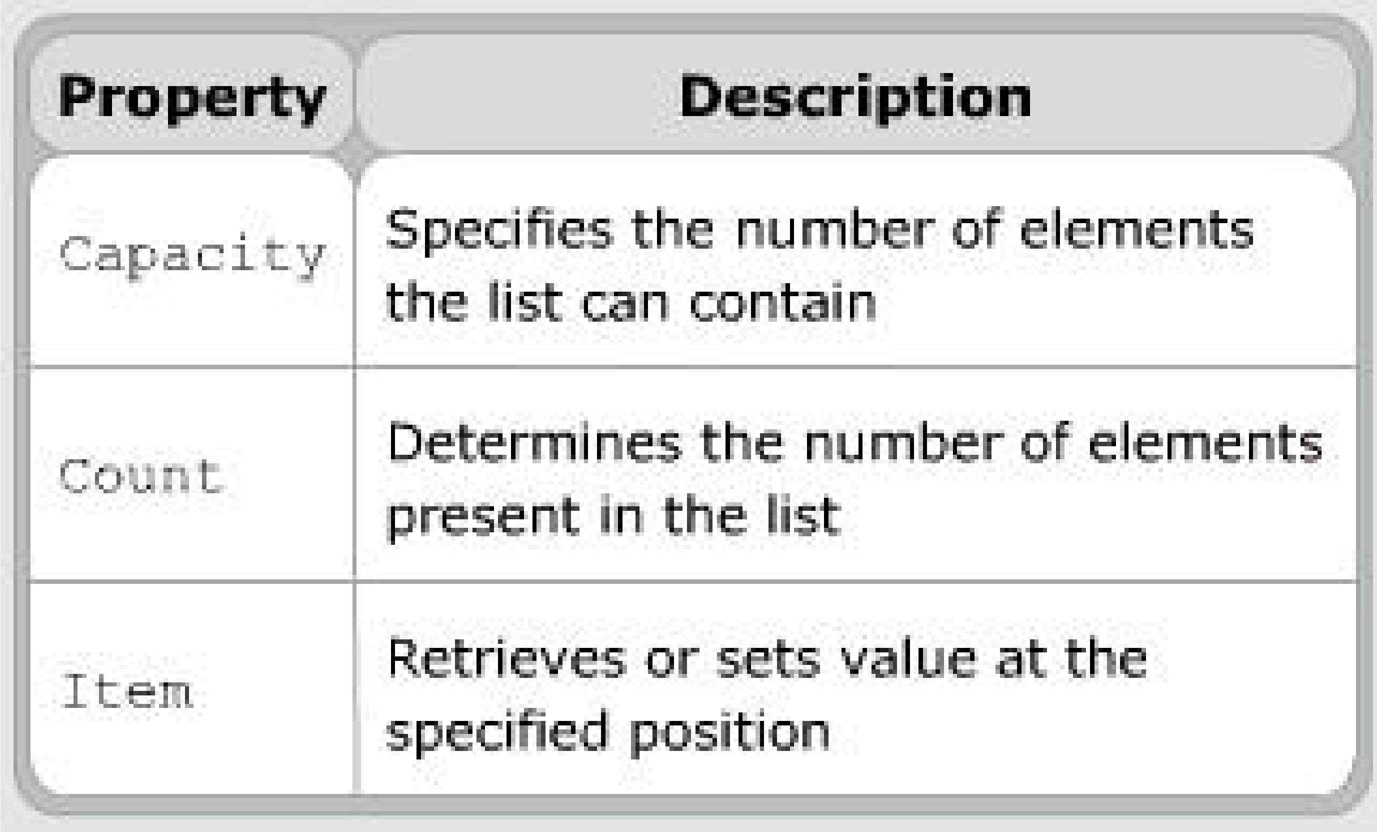
Array List

Methods



Method	Description
`Add`	Adds an element at the end of the list
`Remove`	Removes the specified element that has occurred for the first time in the list
`RemoveAt`	Removes the element present at the specified index position in the list
`Insert`	Inserts an element into the list at the specified index position
`Contains`	Determines the existence of a particular element in the list
`CopyTo`	Copies the elements of the list to the array whose name is supplied as a parameter
`IndexOf`	Returns the index position of an element occurring for the first time in the list
`ToArray`	Copies elements of a list to an array of type `Object`
`TrimToSize`	Specifies the capacity for the actual number of elements in the list

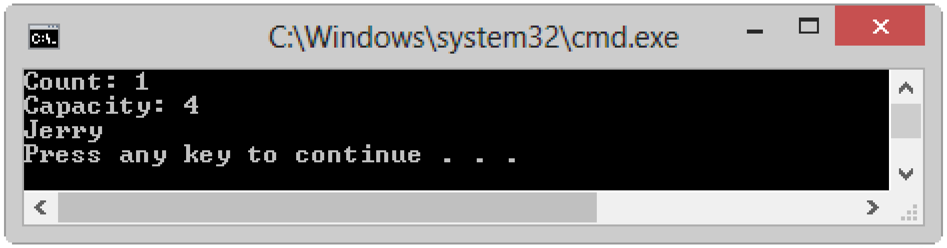
Properties



Property	Description
`Capacity`	Specifies the number of elements the list can contain
`Count`	Determines the number of elements present in the list
`Item`	Retrieves or sets value at the specified position

Snippet

```
static void Main(string[] args)
{
    ArrayList objArr=new ArrayList();
    objArr.Add("Tom");
    objArr.Add("Jerry");
    objArr.Add("Mickey");

    objArr.RemoveAt(0);
    objArr.RemoveAt(1);
    //objArr.Insert(0,"Scarat");

    Console.WriteLine("Count: "+objArr.Count);
    Console.WriteLine("Capacity: "+objArr.Capacity);

    foreach (string mem in objArr)
    {
        Console.WriteLine(mem);
    }
}
```

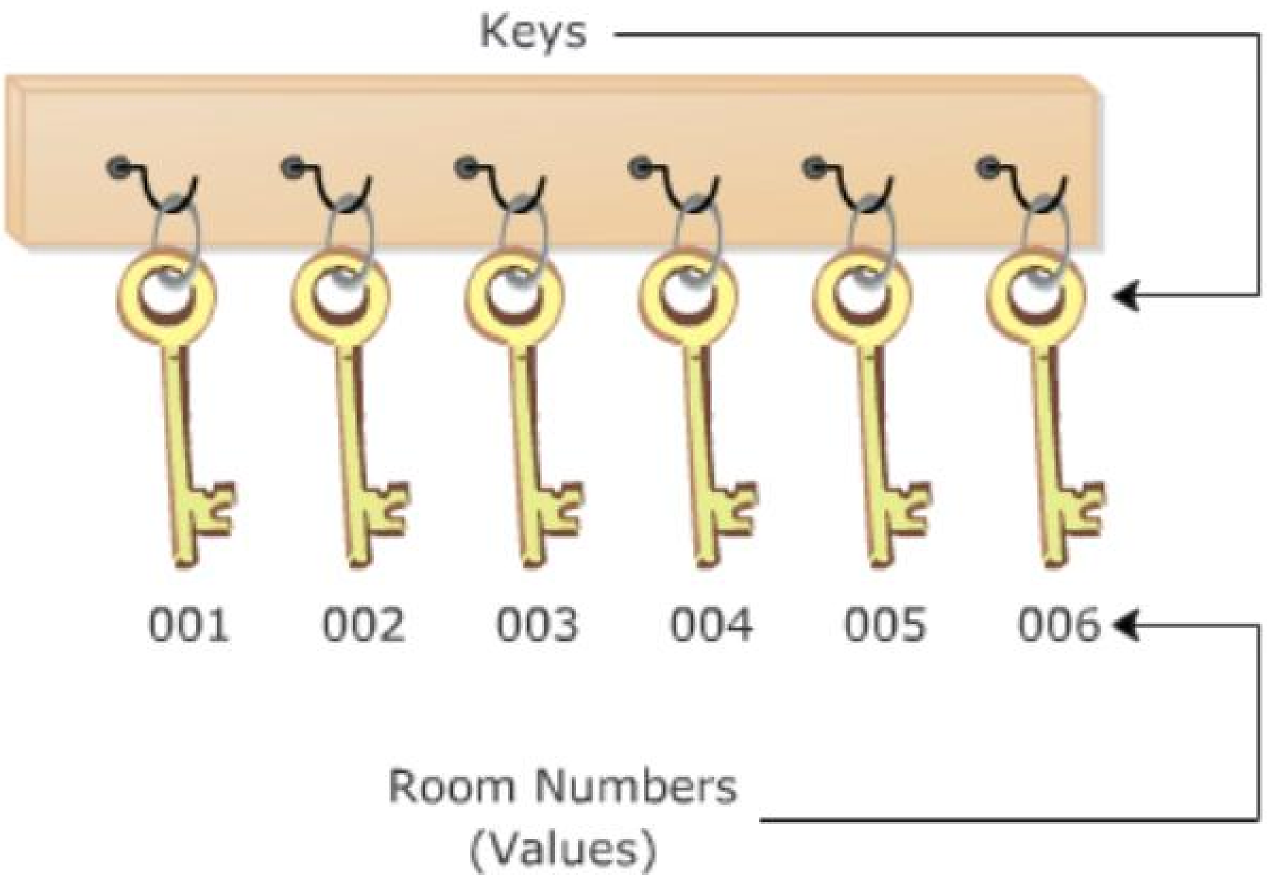


```
Count: 1
Capacity: 4
Jerry
Press any key to continue . . .
```

LESSON 4. HASTABLE CLASS

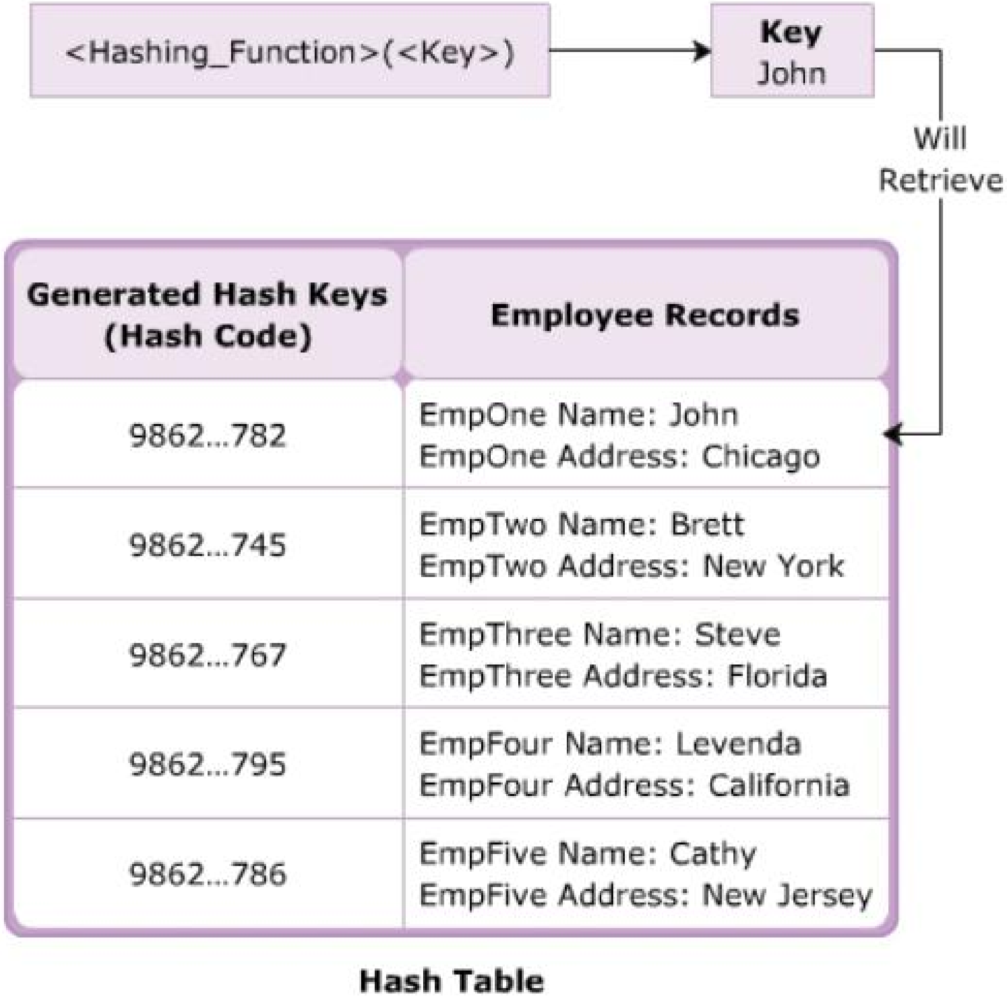
- Introduction
- Methods
- Properties
- Snippet

Introduction



Introduction

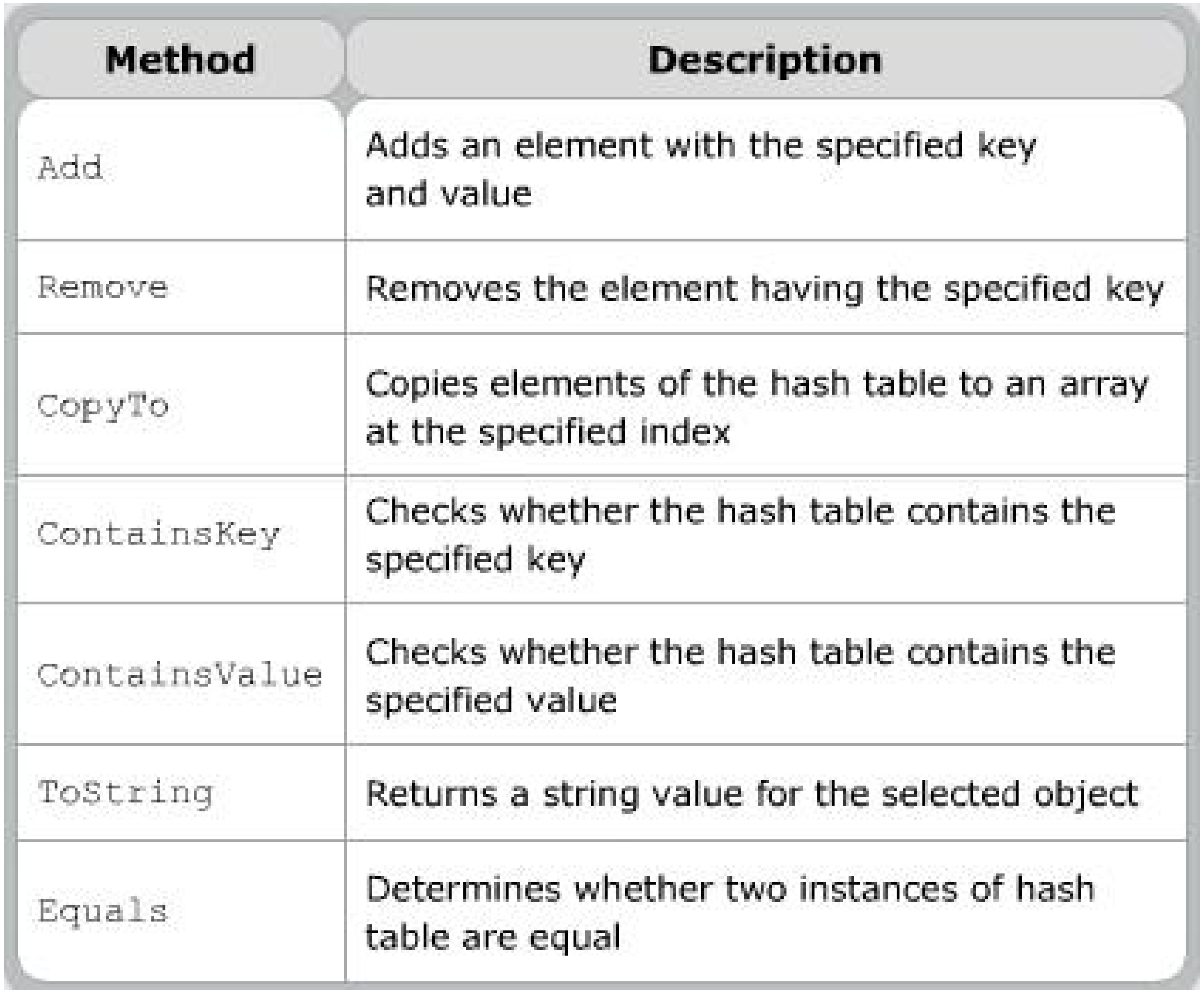
Similar to the keyholder, the Hashtable class in C# allows you to create collections in the form of keys and values.



Generated Hash Keys (Hash Code)	Employee Records
9862...782	EmpOne Name: John EmpOne Address: Chicago
9862...745	EmpTwo Name: Brett EmpTwo Address: New York
9862...767	EmpThree Name: Steve EmpThree Address: Florida
9862...795	EmpFour Name: Levenda EmpFour Address: California
9862...786	EmpFive Name: Cathy EmpFive Address: New Jersey

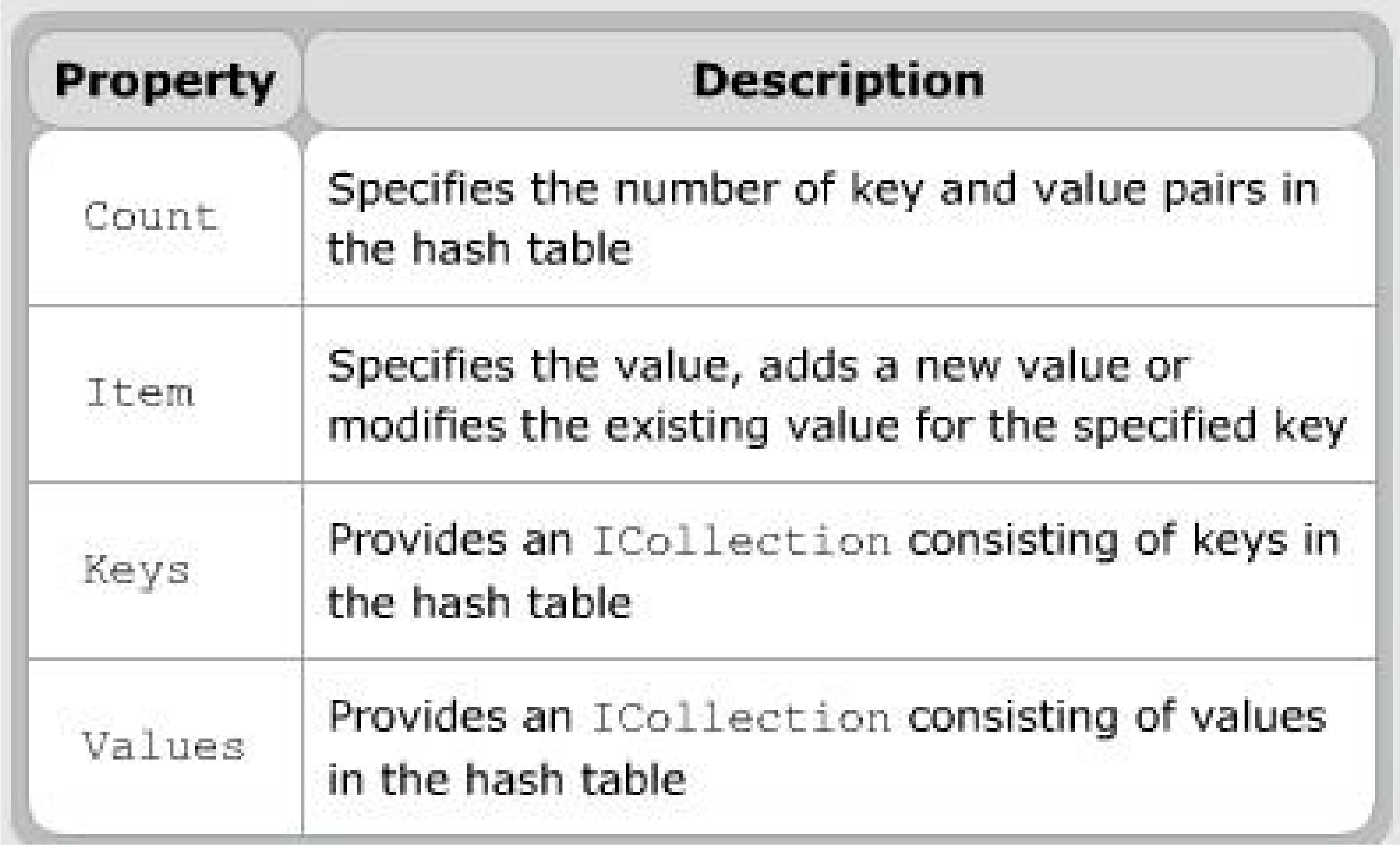
Hash Table

Methods



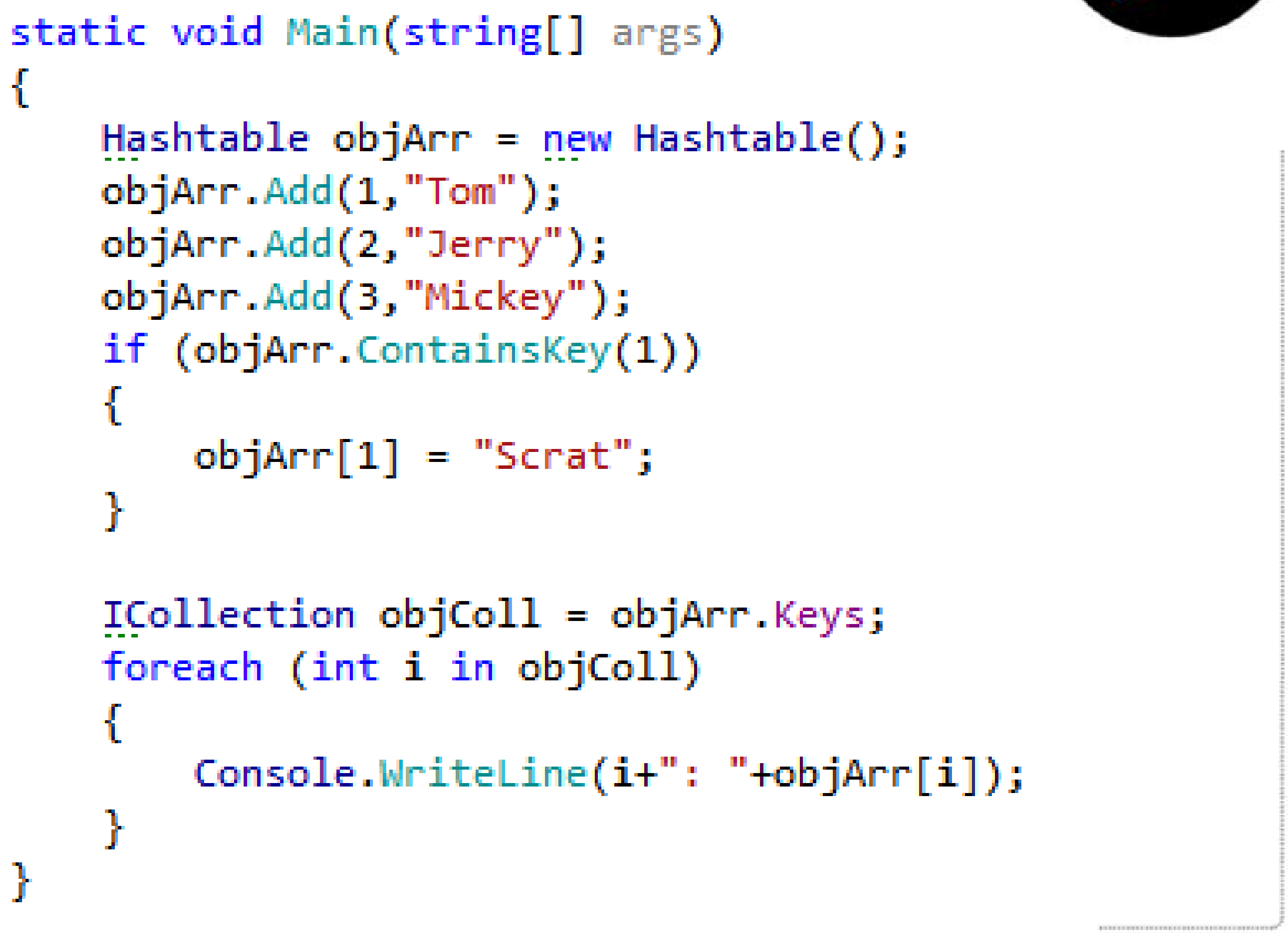
Method	Description
`Add`	Adds an element with the specified key and value
`Remove`	Removes the element having the specified key
`CopyTo`	Copies elements of the hash table to an array at the specified index
`ContainsKey`	Checks whether the hash table contains the specified key
`ContainsValue`	Checks whether the hash table contains the specified value
`ToString`	Returns a string value for the selected object
`Equals`	Determines whether two instances of hash table are equal

Properties



Property	Description
`Count`	Specifies the number of key and value pairs in the hash table
`Item`	Specifies the value, adds a new value or modifies the existing value for the specified key
`Keys`	Provides an `ICollection` consisting of keys in the hash table
`Values`	Provides an `ICollection` consisting of values in the hash table

Properties



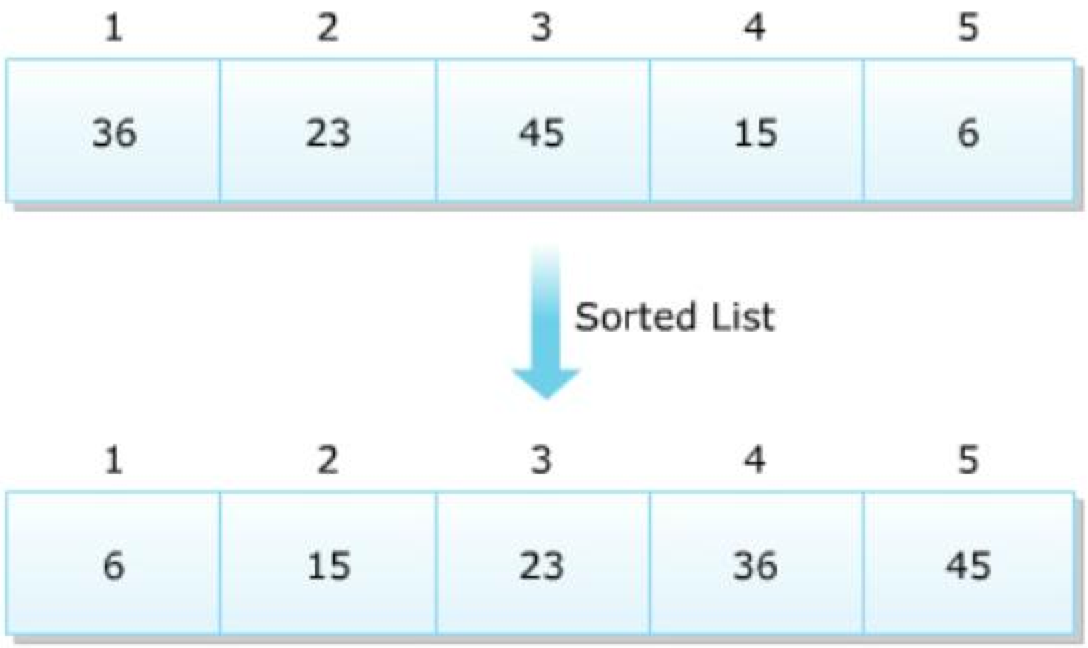
```
static void Main(string[] args)
{
    Hashtable objArr = new Hashtable();
    objArr.Add(1,"Tom");
    objArr.Add(2,"Jerry");
    objArr.Add(3,"Mickey");
    if (objArr.ContainsKey(1))
    {
        objArr[1] = "Scrat";
    }

    ICollection objColl = objArr.Keys;
    foreach (int i in objColl)
    {
        Console.WriteLine(i+": "+objArr[i]);
    }
}
```

LESSION 3. SORTED LIST CLASS

- Introdution
- Methods
- Properties

Introduction

- The SortedList class represents a collection of key and value pairs where elements are **sorted according to the key**.

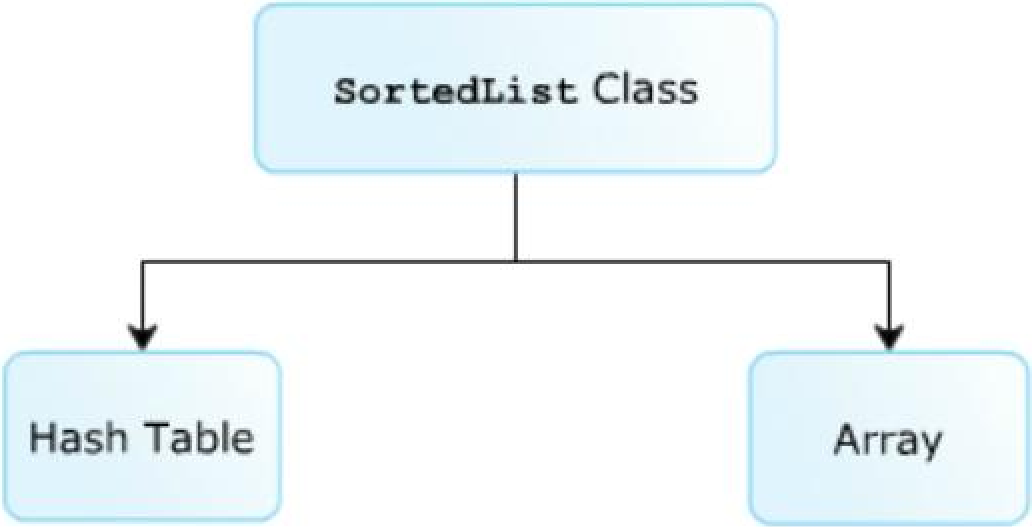


1	2	3	4	5
36	23	45	15	6

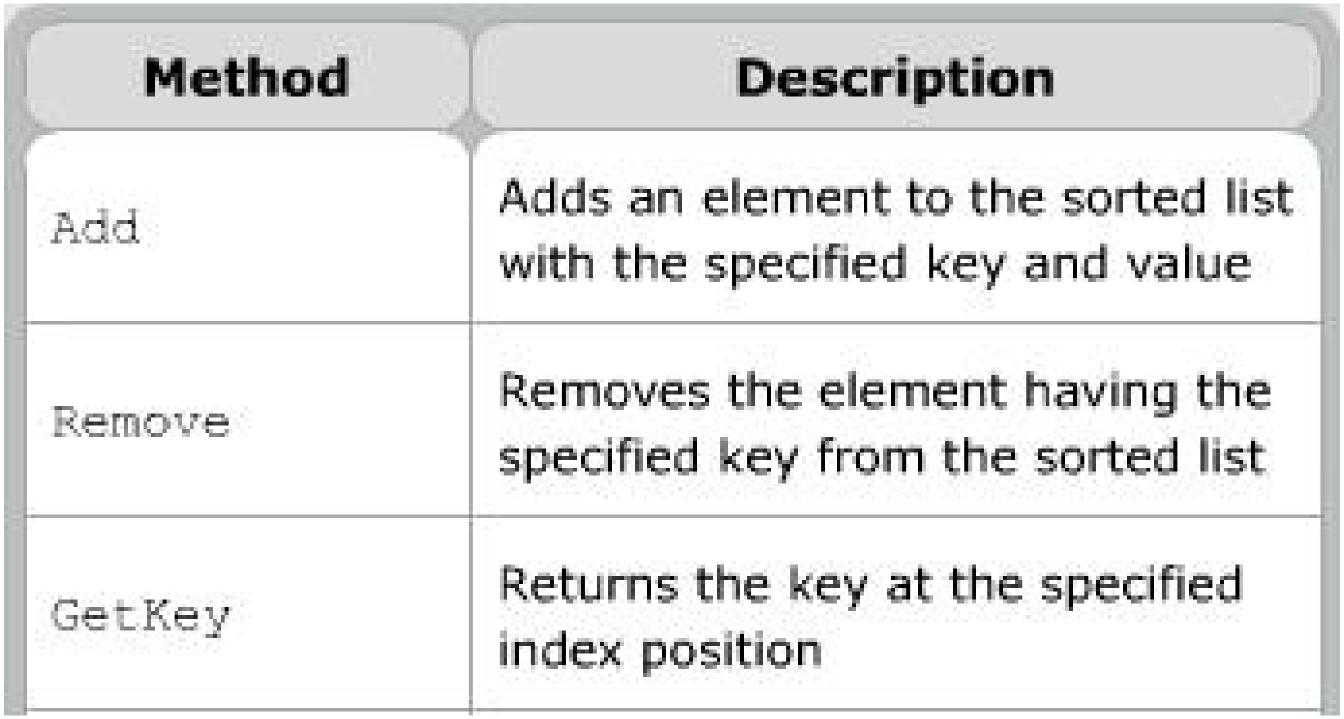
Sorted List

1	2	3	4	5
6	15	23	36	45

- The SortedList class is a **combination** of the Hashtable class and the ArrayList class.

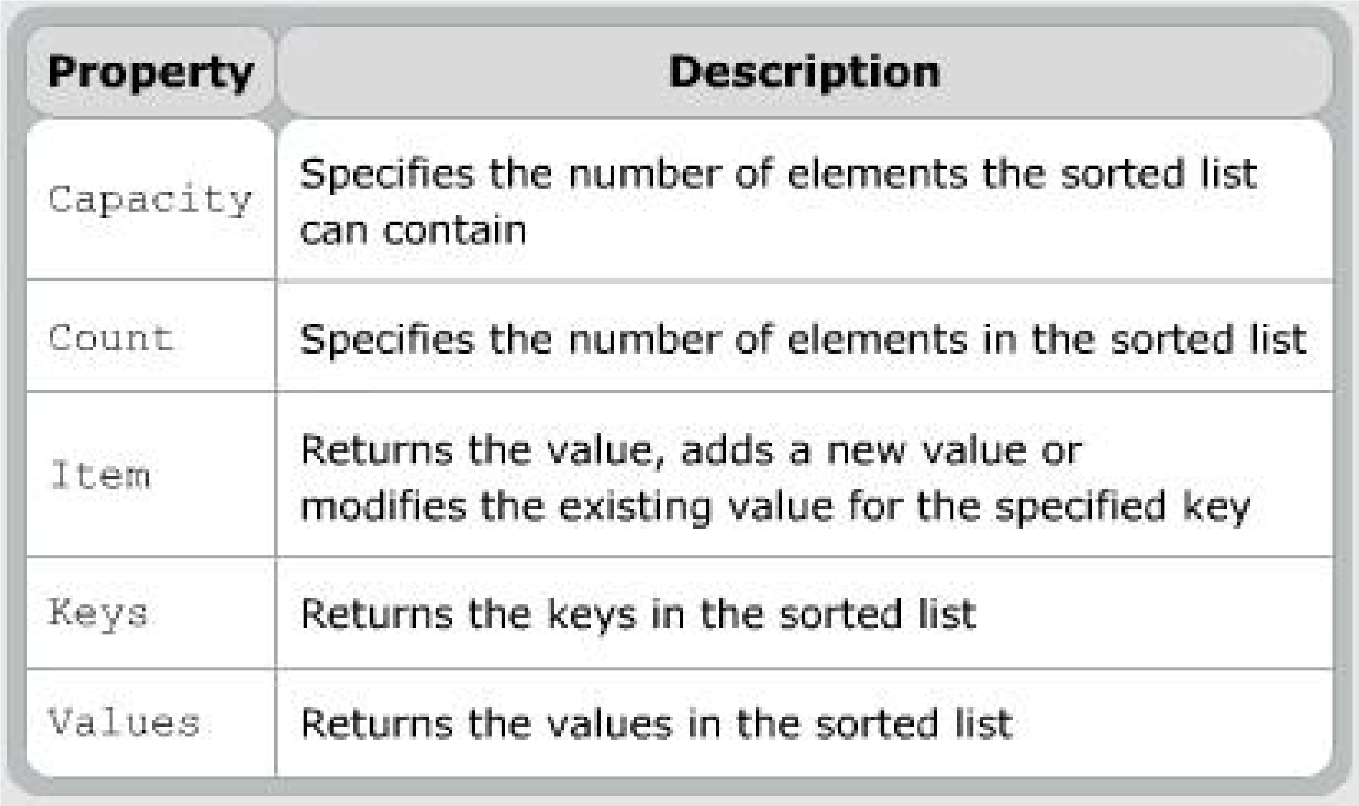


Methods



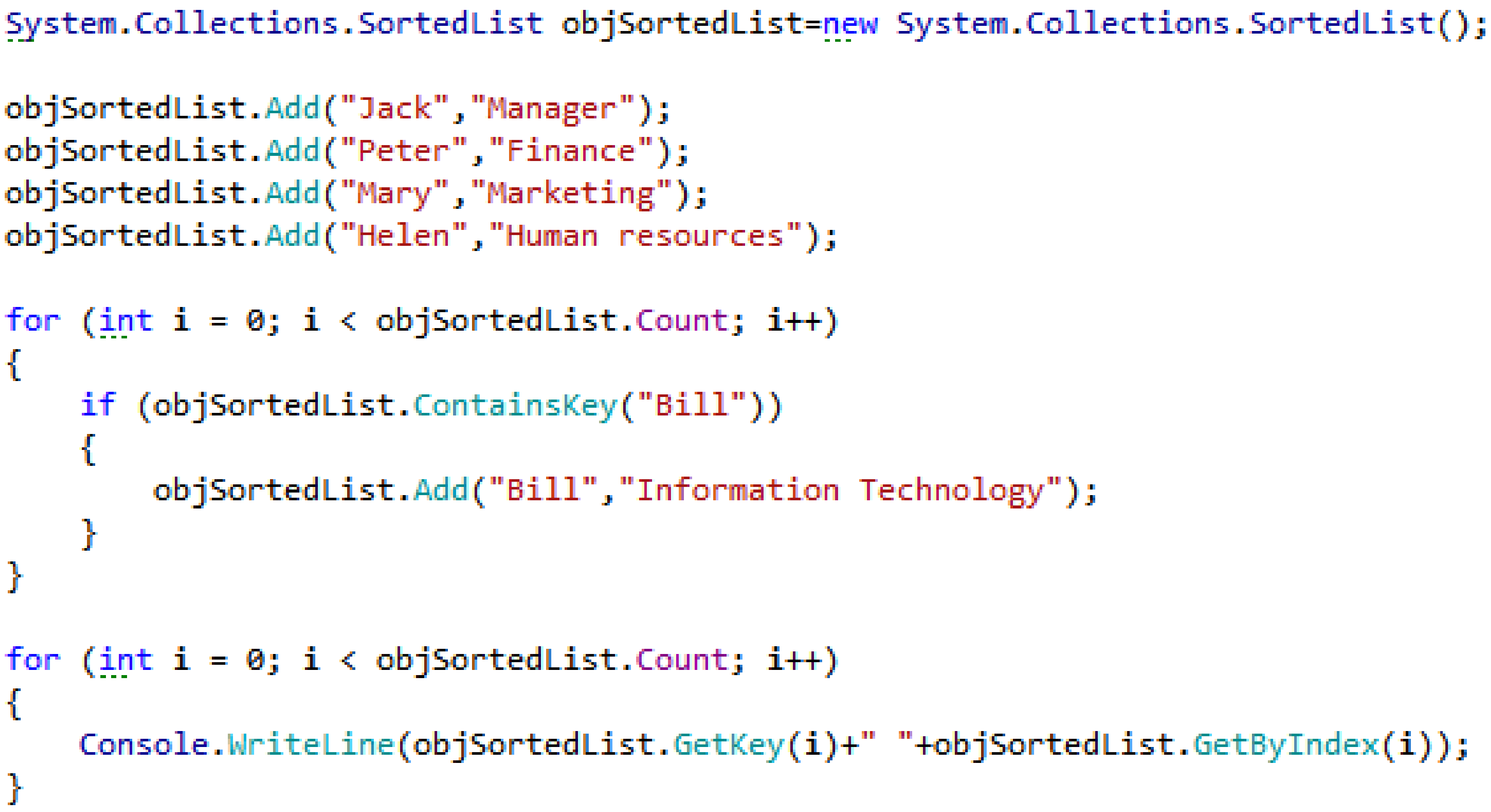
Method	Description
Add	Adds an element to the sorted list with the specified key and value
Remove	Removes the element having the specified key from the sorted list
GetKey	Returns the key at the specified index position

Properties



Property	Description
Capacity	Specifies the number of elements the sorted list can contain
Count	Specifies the number of elements in the sorted list
Item	Returns the value, adds a new value or modifies the existing value for the specified key
Keys	Returns the keys in the sorted list
Values	Returns the values in the sorted list

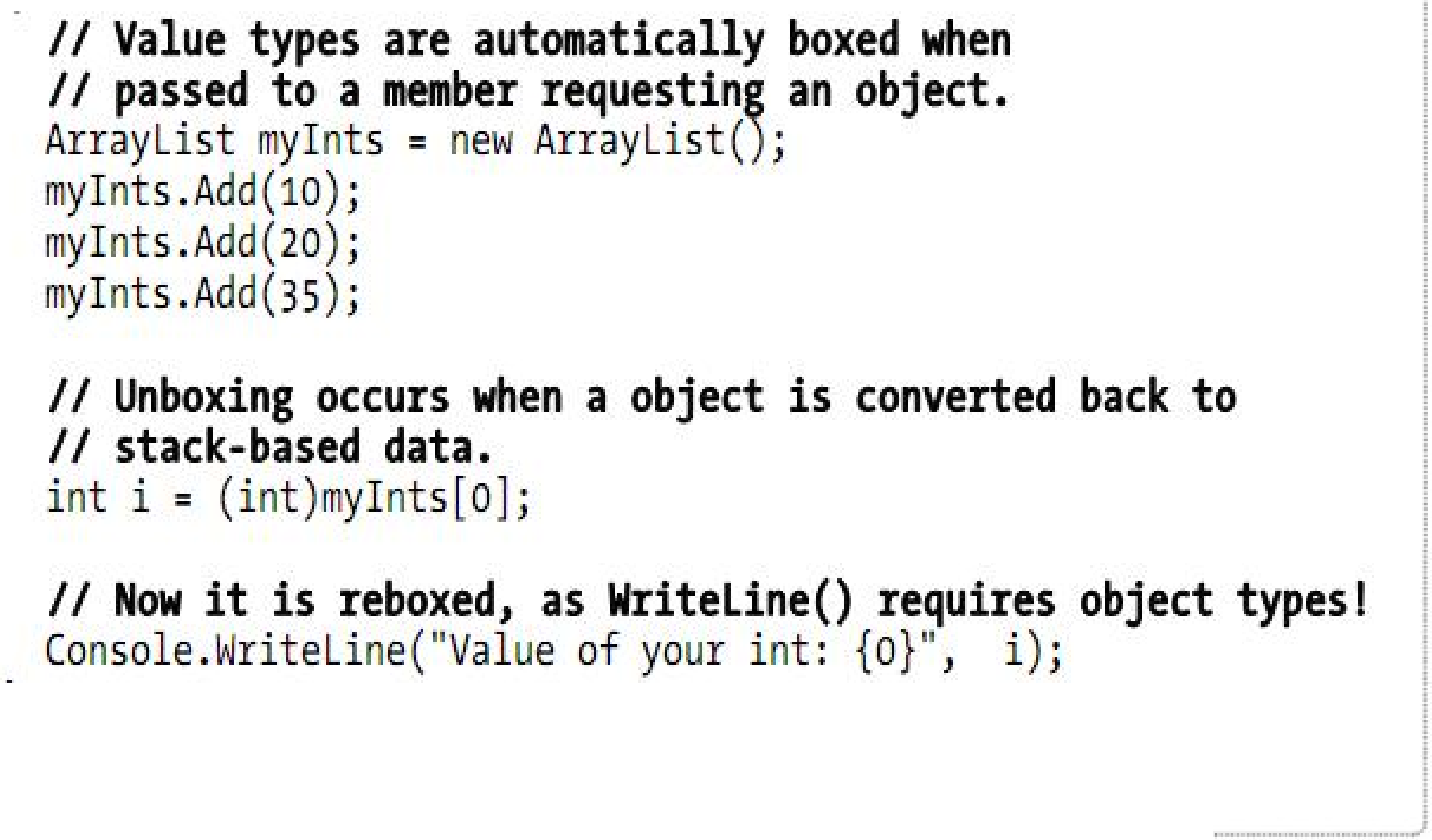
Snippet



```
System.Collections.SortedList objSortedList=new System.Collections.SortedList();

objSortedList.Add("Jack","Manager");
objSortedList.Add("Peter","Finance");
objSortedList.Add("Mary","Marketing");
objSortedList.Add("Helen","Human resources");

for (int i = 0; i < objSortedList.Count; i++)
{
    if (objSortedList.ContainsKey("Bill"))
    {
        objSortedList.Add("Bill","Information Technology");
    }
}

for (int i = 0; i < objSortedList.Count; i++)
{
    Console.WriteLine(objSortedList.GetKey(i)+" "+objSortedList.GetByIndex(i));
}
```

THE PROBLEMS OF NONGENERIC COLLECIONS

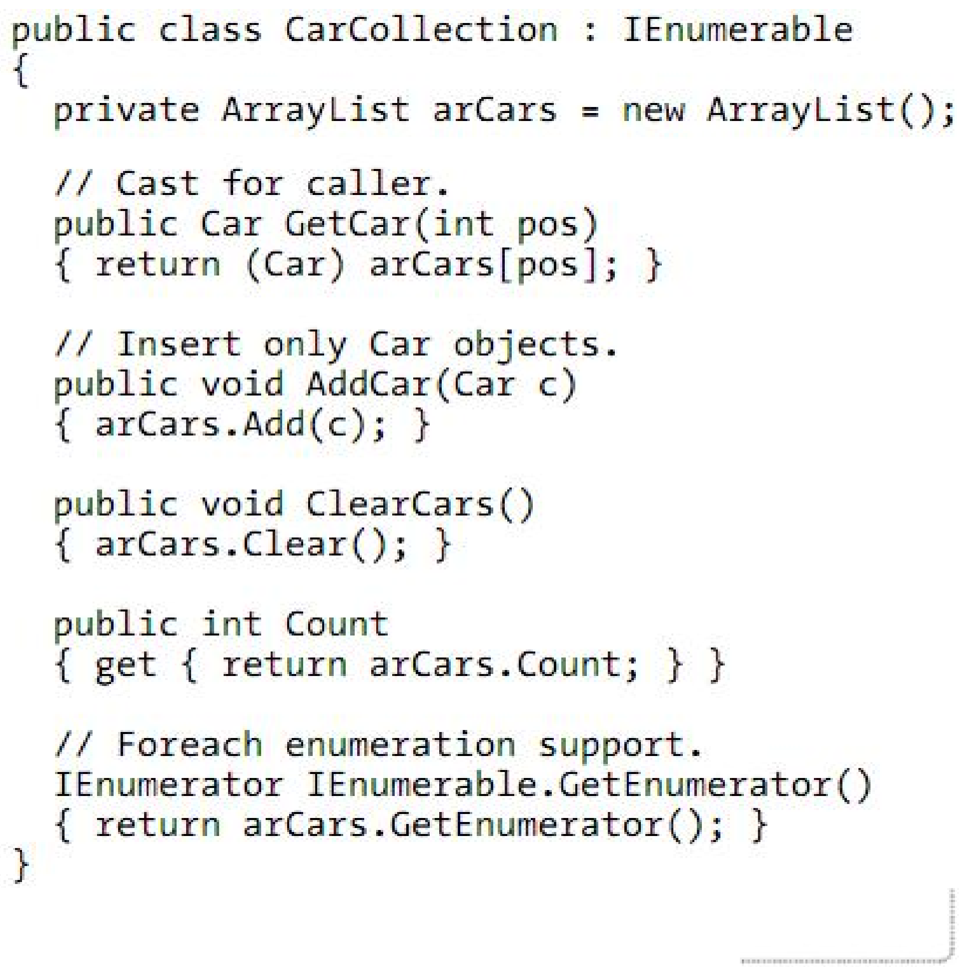
- **The issue of Performance**
  - **Problem with boxing and unboxing**



```
// Value types are automatically boxed when
// passed to a member requesting an object.
ArrayList myInts = new ArrayList();
myInts.Add(10);
myInts.Add(20);
myInts.Add(35);

// Unboxing occurs when a object is converted back to
// stack-based data.
int i = (int)myInts[0];

// Now it is reboxed, as WriteLine() requires object types!
Console.WriteLine("Value of your int: {0}",  i);
```

THE PROBLEMS OF NONGENERIC COLLECIONS

- The issue of Type Safety
  - Custom collections for each unique data type



```
public class CarCollection : IEnumerable
{
  private ArrayList arCars = new ArrayList();

  // Cast for caller.
  public Car GetCar(int pos)
  { return (Car) arCars[pos]; }

  // Insert only Car objects.
  public void AddCar(Car c)
  { arCars.Add(c); }

  public void ClearCars()
  { arCars.Clear(); }

  public int Count
  { get { return arCars.Count; } }

  // Foreach enumeration support.
  IEnumerator IEnumerable.GetEnumerator()
  { return arCars.GetEnumerator(); }
}
```

GENERICS

- Use a Generic collection class to provide many benefits:
  - Generics provide better performance because they do not result in boxing or unboxing penalties when storing value types.
  - Generics are type safe because they can contain only the type of type you specify.
  - Generics greatly reduce the need to build custom collection types because you specify the “type of type” when creating the generic container
- Use generic classes in System.Collection.Generic namespace
  - List<T> where T is of type

GENERICS

- Generics allow you to delay the specification of the data type of programming elements in a class or a method, until it is actually used in the program.
- In other words, generics allow you to write a class or method that can work with any data type.

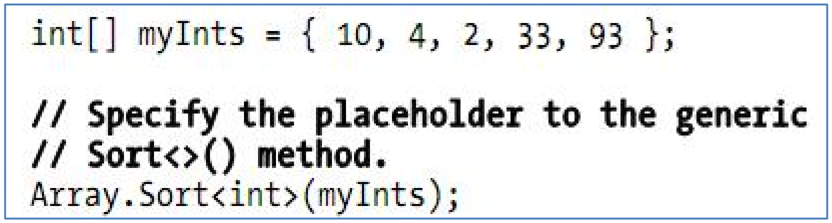
GENERICS

- **Ex:**

```
static void UseGenericList()
{
// This List<> can hold only Person objects.
  List<Person> morePeople = new List<Person>();
  morePeople.Add(new Person  ("Frank", "Black", 50));
  Console.WriteLine(morePeople[0]);

  // This List<> can hold only integers.
  List<int> moreInts = new List<int>();
  moreInts.Add(10);
  moreInts.Add(2);
  int sum = moreInts[0] + moreInts[1];

  // Compile-time error! Can't add Person object
  // to a list of ints!
  // moreInts.Add(new Person());
}
```

GENERICS

- Support a handful of generic members (methods and properties):

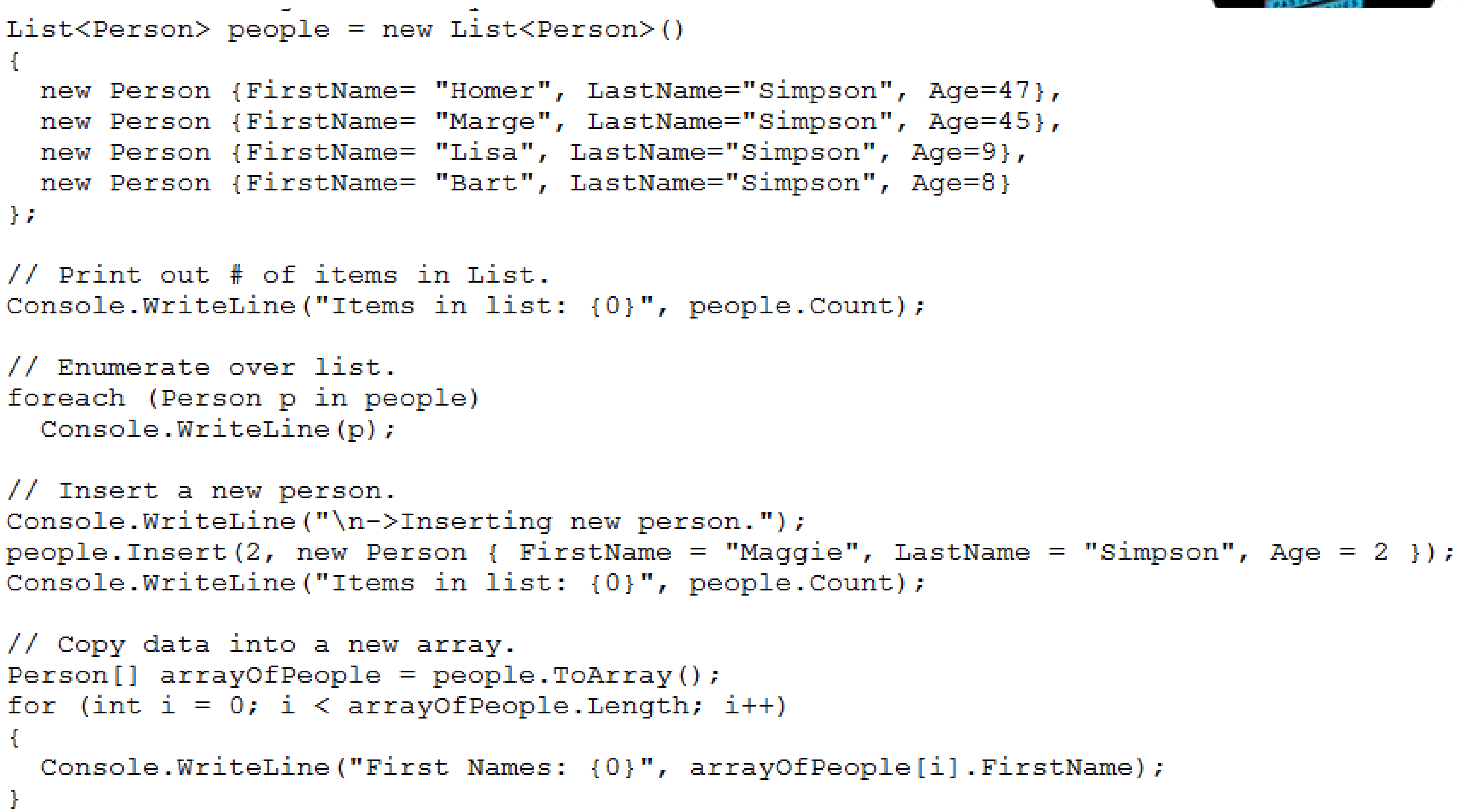


```
int[] myInts = { 10, 4, 2, 33, 93 };

// Specify the placeholder to the generic
// Sort<>() method.
Array.Sort<int>(myInts);
```

- Support various framework behaviors (cloning, sorting, and enumeration)

```
public class Car : IComparable<Car>
{
...
  // IComparable<T> implementation.
  int IComparable<Car>.CompareTo(Car obj)
  {
    if (this.CarID > obj.CarID)
      return 1;
 if (this.CarID < obj.CarID)
     return -1;
   else
     return 0;
 }
```

COLLECTION INITIALIZATION SYNTAX

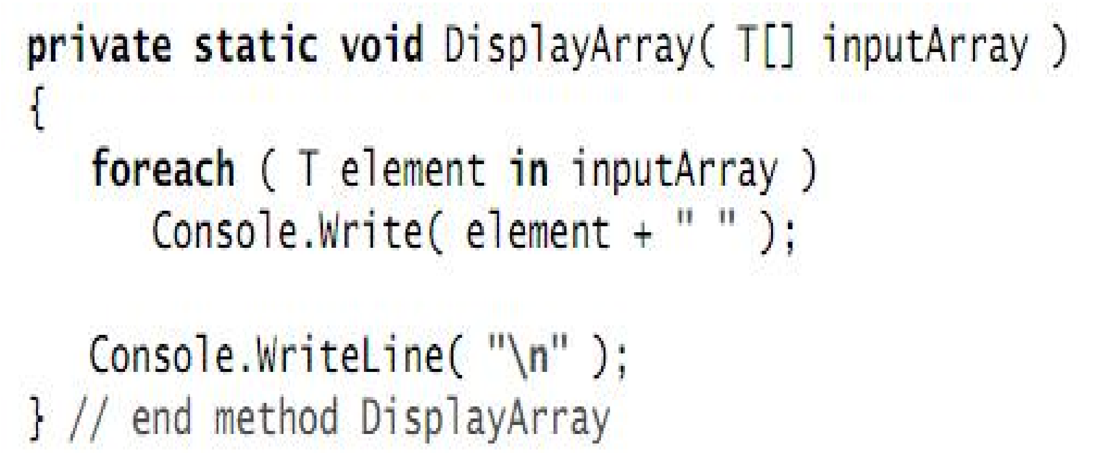


```
List<Person> people = new List<Person>()
{
  new Person {FirstName= "Homer", LastName="Simpson", Age=47},
  new Person {FirstName= "Marge", LastName="Simpson", Age=45},
  new Person {FirstName= "Lisa", LastName="Simpson", Age=9},
  new Person {FirstName= "Bart", LastName="Simpson", Age=8}
};

// Print out # of items in List.
Console.WriteLine("Items in list: {0}", people.Count);

// Enumerate over list.
foreach (Person p in people)
  Console.WriteLine(p);

// Insert a new person.
Console.WriteLine("\n->Inserting new person.");
people.Insert(2, new Person { FirstName = "Maggie", LastName = "Simpson", Age = 2 });
Console.WriteLine("Items in list: {0}", people.Count);

// Copy data into a new array.
Person[] arrayOfPeople = people.ToArray();
for (int i = 0; i < arrayOfPeople.Length; i++)
{
  Console.WriteLine("First Names: {0}", arrayOfPeople[i].FirstName);
}
```

- **Pratice above example with Stack, Queue, SortedList**

GENERIC METHODS

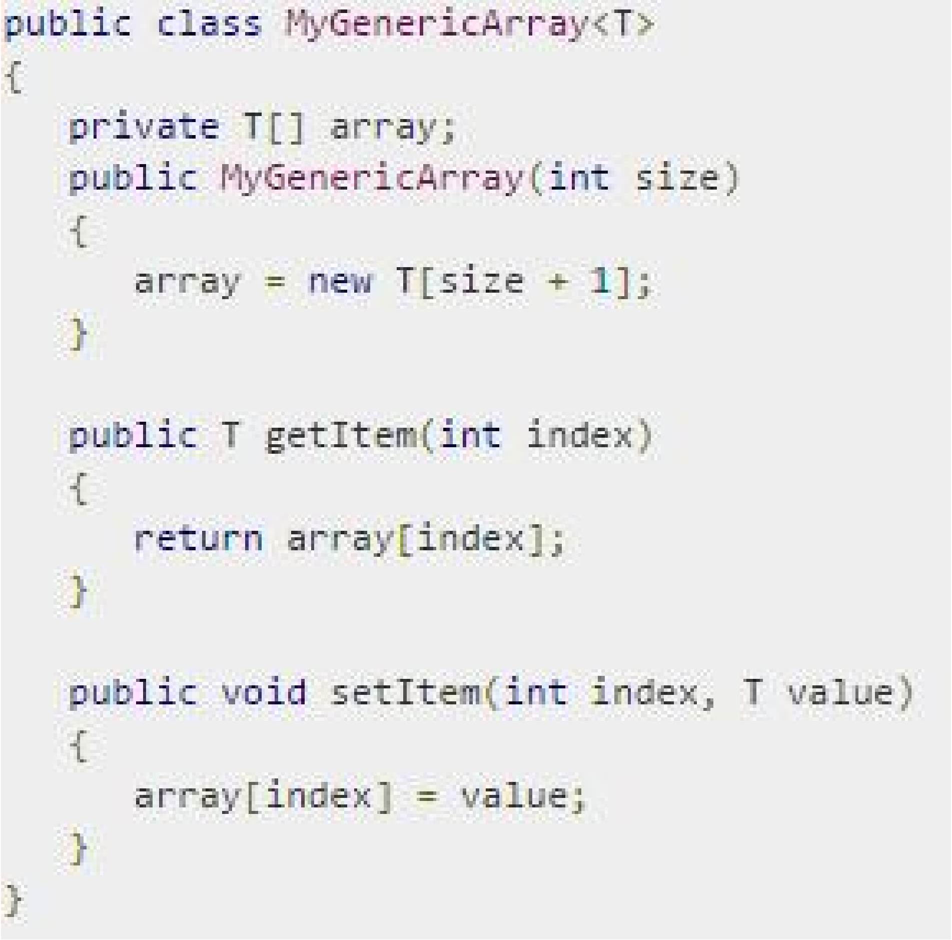
- **Custom Generic method**



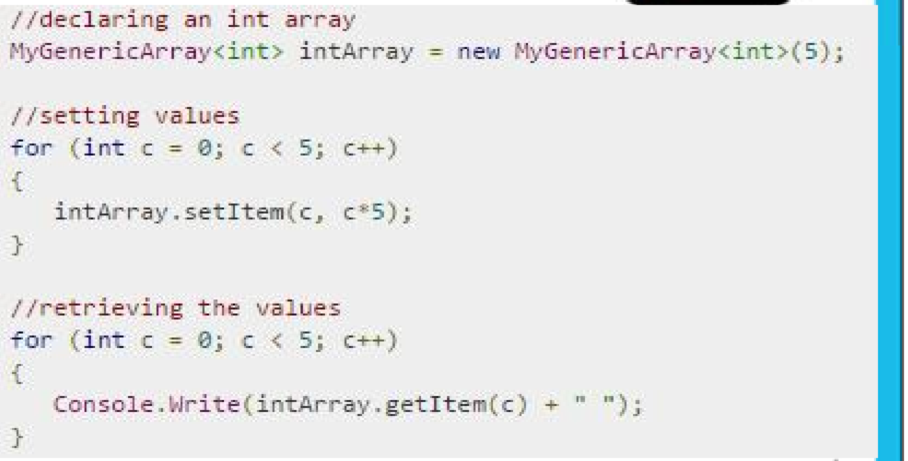
```
private static void DisplayArray( T[] inputArray )
{
   foreach ( T element in inputArray )
      Console.Write( element + " " );

   Console.WriteLine( "\n" );
} // end method DisplayArray
```

- **Ex: Write a swap generic method to swap 2 integer, 2 object of Person type.**

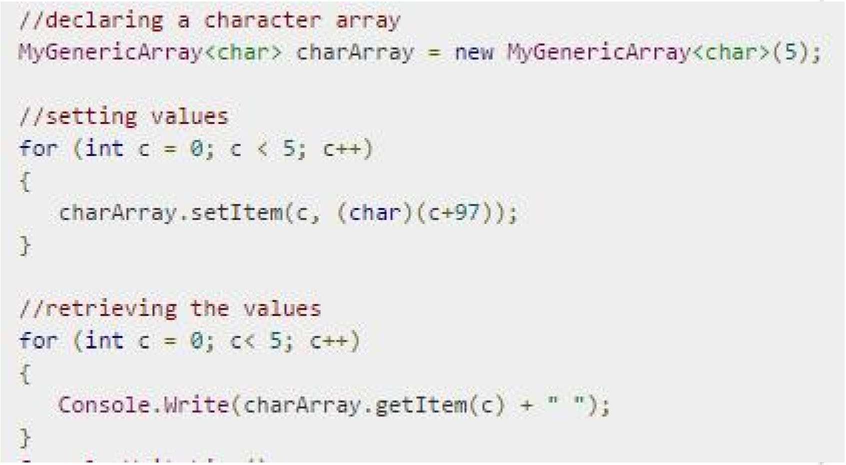
GENERIC CLASSES



```
public class MyGenericArray<T>
{
   private T[] array;
   public MyGenericArray(int size)
   {
      array = new T[size + 1];
   }

   public T getItem(int index)
   {
      return array[index];
   }

   public void setItem(int index, T value)
   {
      array[index] = value;
   }
}
```



```
//declaring an int array
MyGenericArray<int> intArray = new MyGenericArray<int>(5);

//setting values
for (int c = 0; c < 5; c++)
{
   intArray.setItem(c, c*5);
}

//retrieving the values
for (int c = 0; c < 5; c++)
{
   Console.Write(intArray.getItem(c) + " ");
}
```



```
//declaring a character array
MyGenericArray<char> charArray = new MyGenericArray<char>(5);

//setting values
for (int c = 0; c < 5; c++)
{
   charArray.setItem(c, (char)(c+97));
}

//retrieving the values
for (int c = 0; c< 5; c++)
{
   Console.Write(charArray.getItem(c) + " ");
}
```

- **Ex: Practice with Stack, Queue, SortedList**