C# - Generics
C# - Collections

C# - 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.
- You write the specifications for the class or the method, with substitute parameters for data types.
- When the compiler encounters a constructor for the class or a function call for the method, it generates code to handle the specific data type.

C# - Generics

Features of Generics

- It helps you to maximize code reuse, type safety, and performance.
- You can create generic collection classes. The .NET Framework class library contains several new generic collection classes in the *System.Collections.Generic* namespace.
- You may use these generic collection classes instead of the collection classes in the *System.Collections* namespace.
- You can create your own generic interfaces, classes, methods, events and delegates.
- You may create generic classes constrained to enable access to methods on particular data types.
- You may get information on the types used in a generic data type at run-time by means of reflection.

C# - Generics

Features of Generics

- In the .NET Framework, the non-generic collections (ArrayList, Hashtable, SortedKist, Queue etc.) store elements internally
- in 'object' arrays which, can of course, store any type of data.
- This means that, in the case of value types (int, double, bool, char etc), they have to be 'boxed' first and
- then 'unboxed' when you retrieve them which is quite a slow operation.
- Whilst you don't have this problem with reference types, you still have to cast them to their actual types before you can use them.

C# - Generics

Features of Generics

- In contrast, generic collections (List<T>, Dictionary<T, U>, SortedList<T, U>,
- Queue<T> etc) store elements internally in arrays of their actual types and so no boxing or casting is ever required.
- This means that generic collections are faster than their non-generic counterparts when using value types and
- more convenient when using reference types. In short, the latter are now virtually redundant.

C# - Collections

- Collection classes are specialized classes for data storage and retrieval.
- These classes provide support for stacks, queues, lists, and hash tables.
- Most collection classes implement the same interfaces.
- Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc.
- These classes create collections of objects of the Object class, which is the base class for all data types in C#.

C# - Collections

- **Simple Example**

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

class listTest {
    // a function with the unsafe modifier is called from a normal function
        static void Main(string[] args)         {
            List<int> numbers = new List<int>();
            numbers.Add(1);
            numbers.Add(2);
            numbers.Add(3);

            int secondNumber = numbers[1];
            Console.WriteLine(numbers[1]);
            Console.WriteLine(secondNumber);
            Console.Read();
        }
}
```

C# - Collections

- .NET supports two types of collections, generic collections and non-generic collections.
- C# provides several built-in collection classes in the System.Collections and System.Collections.Generic namespaces.
- Prior to .NET 2.0, it was just collections and when generics were added to .NET, generics collections were added as well.
- Generic collections work with generic data type. The following table lists and matches these classes.

Non-generic		Generic
ArrayList	---------->	List
HashTable	---------->	Dictionary
SortedList	---------->	SortedList
Stack	---------->	Stack
Queue	---------->	Queue

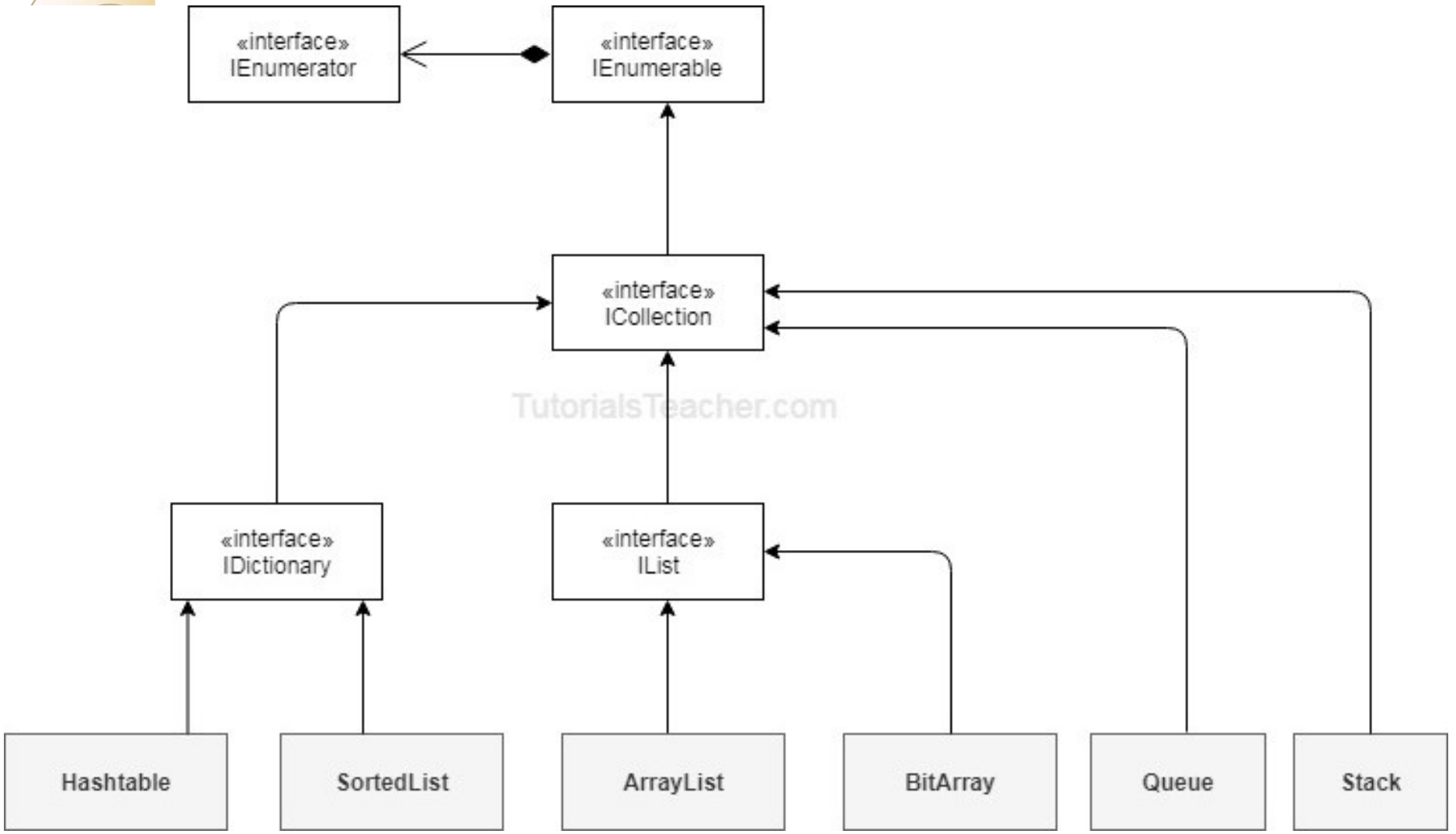
C# - Collections

- In non-generic collections, each element can represent a value of a different type.
  - The collection size is not fixed. Items from the collection can be added or removed at runtime.
- Generic Collections work on the specific type that is specified in the program whereas non-generic collections work on the object type.
  - Specific type
  - Array Size is not fixed
  - Elements can be added / removed at runtime.

C# - Collections

- The following are the various commonly used classes of the **System.Collection** namespace.
- ArrayList
- HashTable
- SortedList
- Stack
- Queue

Hierarchy of the interfaces and classes for the non-generic collections



C# - ArrayList Class

- It represents an ordered collection of an object that can be indexed individually.
- It is basically an alternative to an array.
- However unlike array you can add and remove items from a list at a specified position using an ***index*** and the array resizes itself automatically.
- It also allow dynamic memory allocation, adding, searching and sorting items in the list.

C# - ArrayList Class

Property	Description
Capacity	Gets or sets the number of elements that the ArrayList can contain.
Count	Gets the number of elements actually contained in the ArrayList.
IsFixedSize	Gets a value indicating whether the ArrayList has a fixed size.
IsReadOnly	Gets a value indicating whether the ArrayList is read-only.
Item	Gets or sets the element at the specified index.

C# - ArrayList Class

Method Name & Purpose
public virtual int Add(object value); Adds an object to the end of the ArrayList.
public virtual void AddRange(ICollection c); Adds the elements of an ICollection to the end of the ArrayList.
public virtual void Clear(); Removes all elements from the ArrayList.
public virtual bool Contains(object item); Determines whether an element is in the ArrayList.
public virtual ArrayList GetRange(int index, int count); Returns an ArrayList which represents a subset of the elements in the source ArrayList.
public virtual int IndexOf(object); Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it.

C# - ArrayList Class

Method Name & Purpose
public virtual void Insert(int index, object value); Inserts an element into the ArrayList at the specified index.
public virtual void InsertRange(int index, ICollection c); Inserts the elements of a collection into the ArrayList at the specified index.
public virtual void Remove(object obj); Removes the first occurrence of a specific object from the ArrayList.
public virtual void RemoveAt(int index); Removes the element at the specified index of the ArrayList.
public virtual void RemoveRange(int index, int count); Removes a range of elements from the ArrayList.
public virtual void Reverse(); Reverses the order of the elements in the ArrayList.

C# - ArrayList Class

Method Name & Purpose
public virtual void SetRange(int index, ICollection c); Copies the elements of a collection over a range of elements in the ArrayList.
public virtual void Sort(); Sorts the elements in the ArrayList.
public virtual void TrimToSize(); Sets the capacity to the actual number of elements in the ArrayList.

C# - Hashtable Class

- The Hashtable class represents a collection of **key-and-value pairs** that are organized based on the hash code of the key.
- It uses the key to access the elements in the collection.
- A hash table is used when you need to access elements by using **key**, and you can identify a useful key value.
- Each item in the hash table has a key/value pair.
- The key is used to access the items in the collection.

C# - Hashtable Class

Property	Description
Count	Gets the number of key-and-value pairs contained in the Hashtable.
IsFixedSize	Gets a value indicating whether the Hashtable has a fixed size.
IsReadOnly	Gets a value indicating whether the Hashtable is read-only.
Item	Gets or sets the value associated with the specified key.
Keys	Gets an ICollection containing the keys in the Hashtable.
Values	Gets an ICollection containing the values in the Hashtable.

C# - Hashtable Class

Method Name & Purpose
public virtual void Add(object key, object value); Adds an element with the specified key and value into the Hashtable.
public virtual void Clear(); Removes all elements from the Hashtable.
public virtual bool ContainsKey(object key); Determines whether the Hashtable contains a specific key.
public virtual bool ContainsValue(object value); Determines whether the Hashtable contains a specific value.
public virtual void Remove(object key); Removes the element with the specified key from the Hashtable.

C# - SortedList Class

- The SortedList class represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index.
- 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 items using an index, it is an ArrayList, and if you access items using key, it is a Hashtable.
- The collection of items is always sorted by the key value.

C# - SortedList Class

- The SortedList class implements the IEnumerable, ICollection, IDictionary and ICloneable interfaces.
- In SortedList, an element can be accessed by its key or by its index.
- A SortedList object internally maintains two arrays to store the elements of the list, i.e, one array for the keys and another array for the associated values.
- Here, a key cannot be null, but a value can be.
- The capacity of a SortedList object is the number of key/value pairs it can hold.

C# - SortedList Class

- In SortedList, duplicate keys are not allowed.
- In SortedList, you can store values of the same type and of the different types due to the non-generic collection. If you use a generic SortedList in your program, then it is necessary that the type of the values should be the same.
- In SortedList you cannot store keys of different data types in the same SortedList because the compiler will throw an exception. So, always add the key in your SortedList of the same type.
- You can also cast key/value pair of SortedList into DictionaryEntry.

C# - SortedList Class

Property	Description
Capacity	Gets or sets the capacity of the SortedList.
Count	Gets the number of elements contained in the SortedList.
IsFixedSize	Gets a value indicating whether the SortedList has a fixed size.
IsReadOnly	Gets a value indicating whether the SortedList is read-only.
Item	Gets and sets the value associated with a specific key in the SortedList.
Keys	Gets the keys in the SortedList.
Values	Gets the values in the SortedList.

C# - Stack Class

- It represents a last-in, first out collection of object.
- It is used when you need a last-in, first-out access of items.
- When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.

Property	Description
Count	Gets the number of elements contained in the Stack.

C# - Stack Class

Method Name & Purpose
public virtual void Clear(); Removes all elements from the Stack.
public virtual bool Contains(object obj); Determines whether an element is in the Stack.
public virtual object Peek(); Returns the object at the top of the Stack without removing it.
public virtual object Pop(); Removes and returns the object at the top of the Stack.
public virtual void Push(object obj); Inserts an object at the top of the Stack.
public virtual object[] ToArray(); Copies the Stack to a new array.

C# - Queue Class

- It represents a first-in, first out collection of object.
- It is used when you need a first-in, first-out access of items.
- When you add an item in the list, it is called **enqueue**, and when you remove an item, it is called **deque.**

Property	Description
Count	Gets the number of elements contained in the Queue.

C# - Queue Class

Method Name & Purpose
public virtual void Clear(); Removes all elements from the Queue.
public virtual bool Contains(object obj); Determines whether an element is in the Queue.
public virtual object Dequeue(); Removes and returns the object at the beginning of the Queue.
public virtual void Enqueue(object obj); Adds an object to the end of the Queue.
public virtual object[] ToArray(); Copies the Queue to a new array.
public virtual void TrimToSize(); Sets the capacity to the actual number of elements in the Queue.

C# - Dictionary Class

- A Dictionary<TKey, TValue> is a generic collection that consists of elements as key/value pairs that are not sorted in an order. For example,

```
Dictionary<int, string> country = new Dictionary<int, string>();
```

- Here, country is a dictionary that contains int type keys and string type values. Here is how we can create a dictionary in C#.

```
Dictionary<dataType1, dataType2> dictionaryName = new Dictionary<dataType1, dataType2>();
```

- Here,

dictionaryName - name of the dictionary
dataType1 - datatype of keys
dataType2 - datatype of values

C# - Dictionary Class

- In C#, we can perform different operations on a dictionary. We will look at some commonly used Dictionary<TKey, TValue> operations in this tutorial:
  - Add Elements
  - Access Elements
  - Change Elements
  - Remove Elements
- We can change the value of elements in dictionary and remove the elements inside the dictionary.