.NET Collections and Generics

PRATIAN
TECHNOLOGIES

PRATIAN TECHNOLOGIES

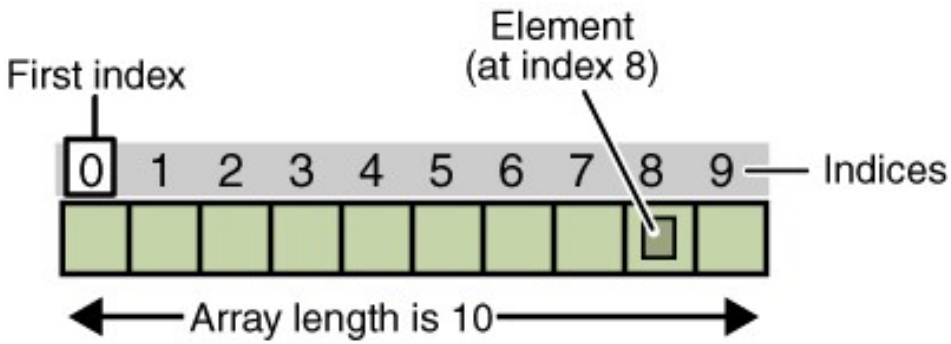
Objectives

- After the end of the Session the participants should be able to:
  - Understand Problems with Array.
  - Understand what are Collections.
    - What?
    - Why?
    - How?
  - Understand Generics- GenX Collections.
  - What are the built-in Classes of Generics
  - What are the Interfaces of Generics
  - Create Custom Generics
  - Implementing Custom generics.

PRATIAN TECHNOLOGIES

Current Scenario

- Arrays.
  - The Most simplest way of storing data as a Single Unit.
  - Fixed size is not always the Real time scenario.
  - How Could you achieve dynamic features in an Array?



PRATIAN TECHNOLOGIES

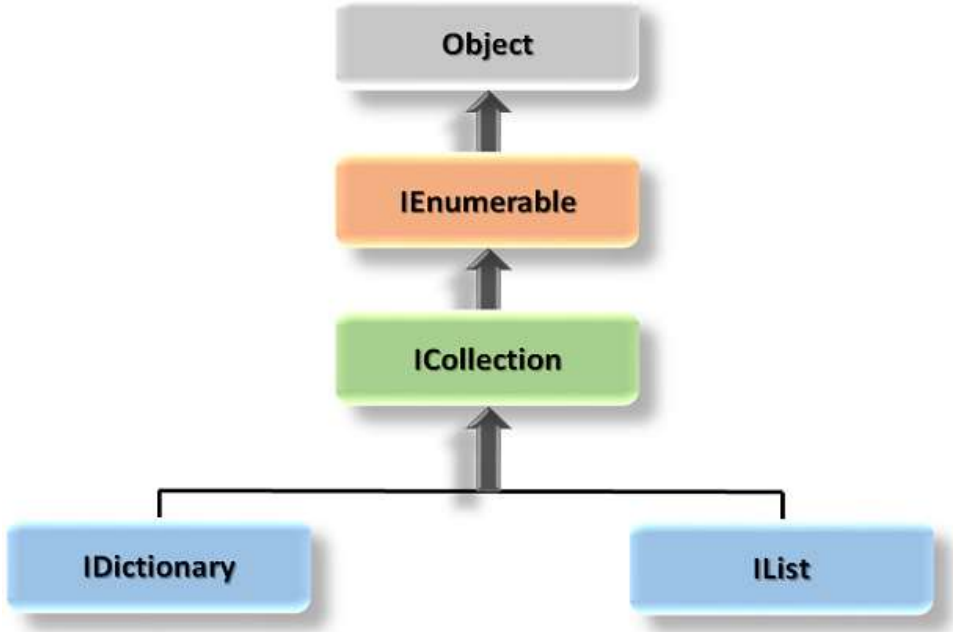
Collections

- What?
  - An object that stores a group of elements together as one unit.
  - A group of data of similar type semantically grouped for a purpose.
  - Arrays are the most primitive type of Collections available in all Languages.
- Why?
  - Traditional Arrays are fixed in Size.
  - Real time Applications needs an ability to append, Remove or modify the existing Data as per their needs.
  - An Example without an Idea of Dynamic Data.
- How?
  - .NET framework provides a huge sets of classes which fulfill the issues of Collections.
  - Grouped under the namespace **System.Collections**.

PRATIAN TECHNOLOGIES

Collections under .NET Terminology

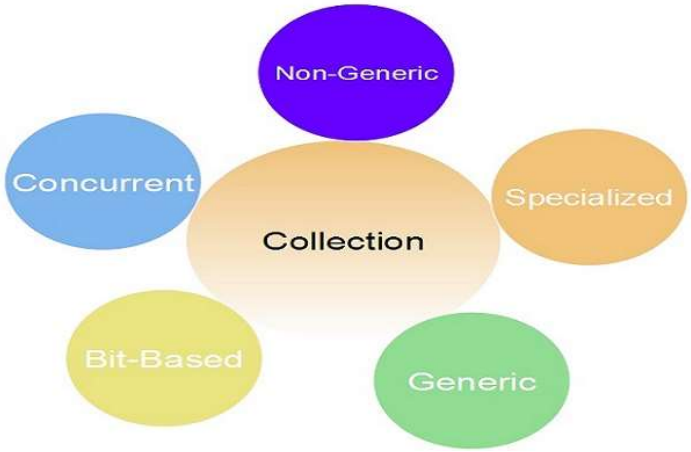
- Anything that allows to refer each element within it is called Collection.
- Simply put, It's a class that implements an interface called **IEnumerable**.



PRATIAN TECHNOLOGIES

System.Collections

- The .NET framework has a variety of built in Classes which provide a puzzling selection of collection objects, each with a somewhat specialized purpose.
- Supports 5 General types of Collections
  - Non Generic
  - Specialized
  - Bit based.
  - Generic
  - Concurrent



PRATIAN TECHNOLOGIES

Non-Generic Collections

Collection Classes	Description
ArrayList	A dynamically sized array that contain any sort of object.
Stack	A collection class that works on the Last in First out (Lifo) pattern. Means the element inserted Last will delete First.
Queue	A collection class that works on the First in First out (Fifo) pattern. Means the element inserted first will delete first.
Hashtable	A hash table that maps given keys to value allowing for bucket of segment data, A hash table maintains a sorted list of key/value pairs that can be accessed by a key value
SortedList	Sorted List maintains a sorted list of key/value pairs that can be accessed either key or index.

PRATIAN TECHNOLOGIES

Non-Generic Collections

- Demo

PRATIAN TECHNOLOGIES

Generic vs. Non Generic

- Non Generic stores the data as Object
  - Not type safe
  - Available in v1.0 and v1.1
- Generic are type safe way of storing the objects as Data structures
  - Available since v2.0
  - Easy way of storing objects.

Generic Collections (System.Collections.Generic)	**Non Generic Collections** (System.Collections)
List<T>	ArrayList
Dictionary<Tkey, Tvalue>	Hashtable
Stack<T>	Stack
Queue<T>	Queue

PRATIAN TECHNOLOGIES

Using Non Generic Class

```
ArrayList list = new ArrayList();
list.Add(123);//Stores the data as object(Boxed Value)
list.Add("SomeName");//Accepts this, but Wrong interpretation.
//Compiler does not throw Error.
```

Using Generic Class

```
List<int> list = new List<int>();
list.Add(123);//Stores the data as integer.
list.Add("SomeName");/*Does not Accepts this and
Compiler tells about this Error.*/
```

PRATIAN TECHNOLOGIES

Retrieval Issues

- **Using Non Generics:**

```
foreach(object element in list)
{
    if (element is int)//Check first for type safety.
    {
        //Integer Operations
    }
    else
    {
        //Non Integer Operations
    }
}
```

- **Using Generics:**

```
foreach(int element in list)//No  Need to check for Integer
{
    //Integer Operations
}
```

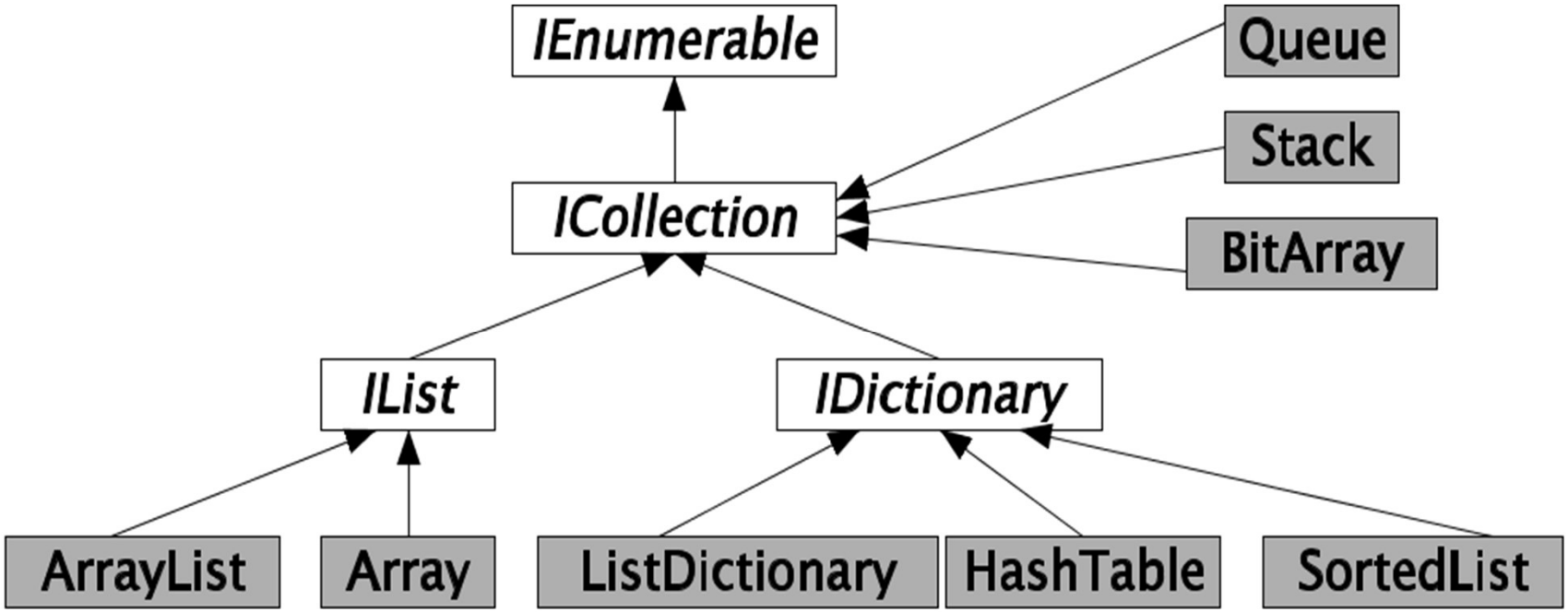
PRATIAN TECHNOLOGIES

Generics

- GenX Collections.
- The whole idea is to avoid accidental Type mismatches.
- Similar to Templates of C++.
- Have Classes to perform type safe way of storing the data.

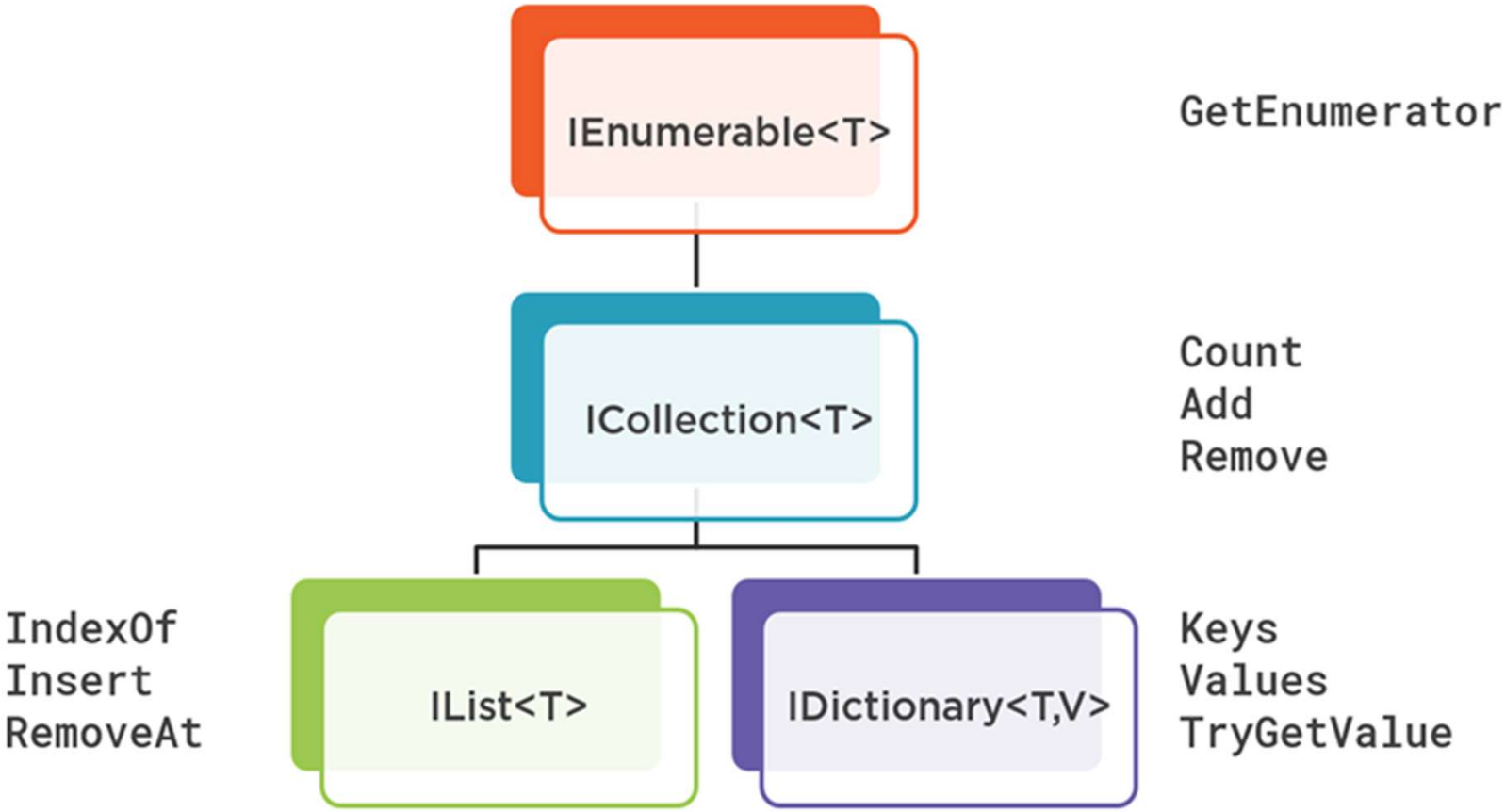
PRATIAN TECHNOLOGIES

The Classes and Interfaces of Generics.



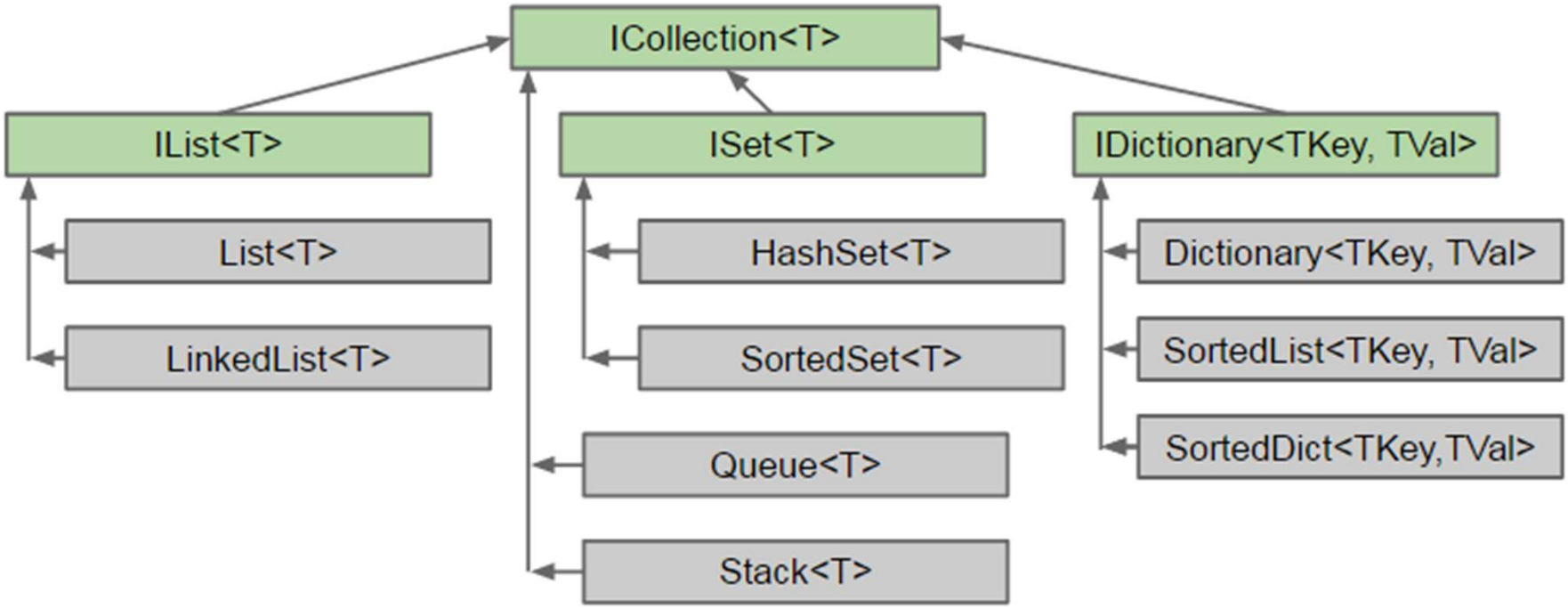
PRATIAN TECHNOLOGIES

Generic Collection Interfaces



PRATIAN TECHNOLOGIES

Complete Hierarchy



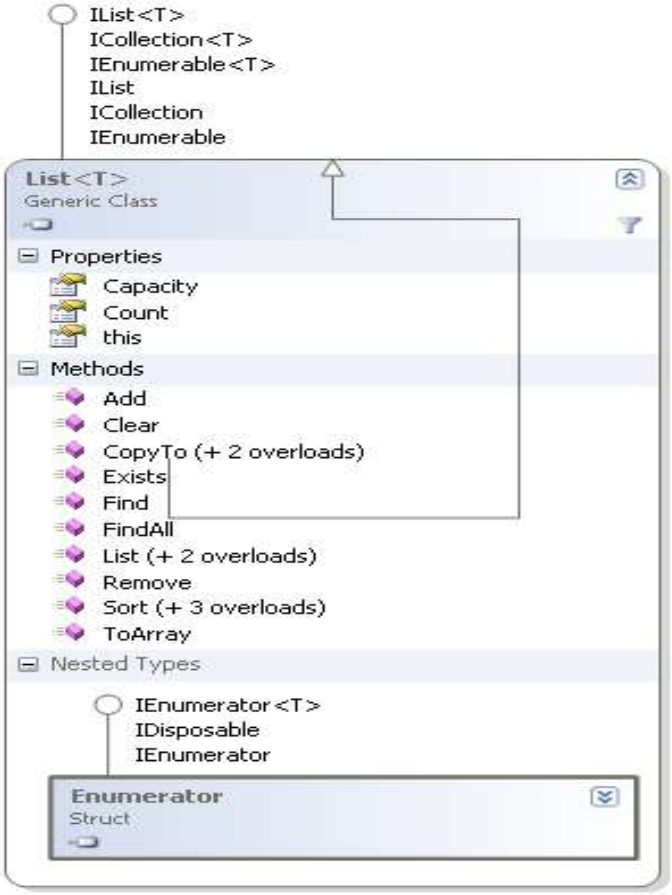
PRATIAN TECHNOLOGIES

Classes under Generics

Class	Description
List<T>	A dynamic array. Provides functionality similar to that found in the non-generic ArrayList class.
Dictionary<TK, TV>	Stores key/value pairs. Provides functionality similar to that found in the non-generic Hashtable class.
HashSet<T>	Represents a set of values.
LinkedList<T>	Stores elements in a doubly linked list.
Queue<T>	A first-in, first-out list. Provides functionality similar to that found in the non-generic Queue class.
SortedDictionary<TK, TV>	A sorted list of key/value pairs.
SortedList<TK, TV>	A sorted list of key/value pairs. Provides functionality similar to that found in the non-generic SortedList class.
Stack<T>	A first-in, last-out list. Provides functionality similar to that found in the non-generic Stack class.

PRATIAN TECHNOLOGIES

The List<T> Design

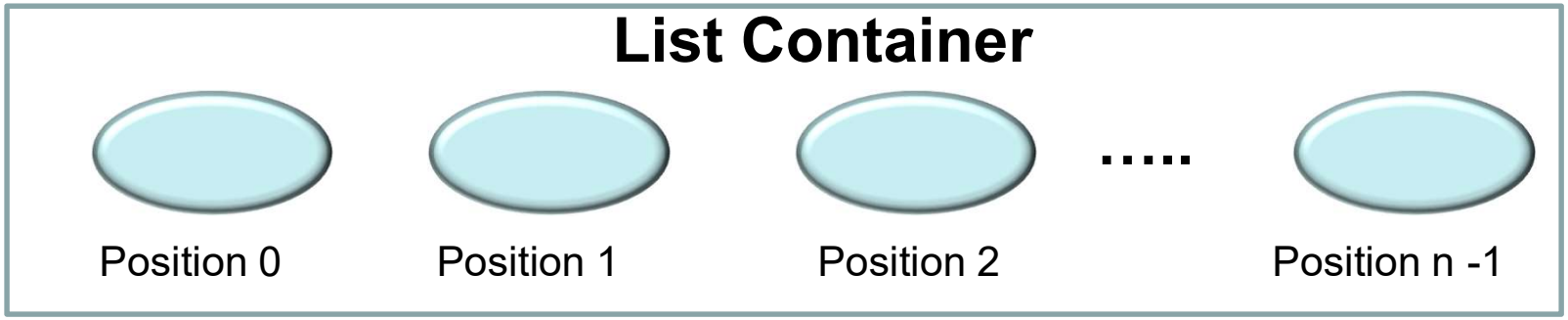


- Provides a Generic Dynamic Array kind of Data storage in it.
- Stores the data as First In Last Out.
- Simplest form of storing the data where we can modify the structure at any point of time.

PRATIAN TECHNOLOGIES

List<T> Usage

- Lists are Dynamic Array.
- Element is added to the Last of the list.
- The Count gives the number of elements within the List.



PRATIAN TECHNOLOGIES

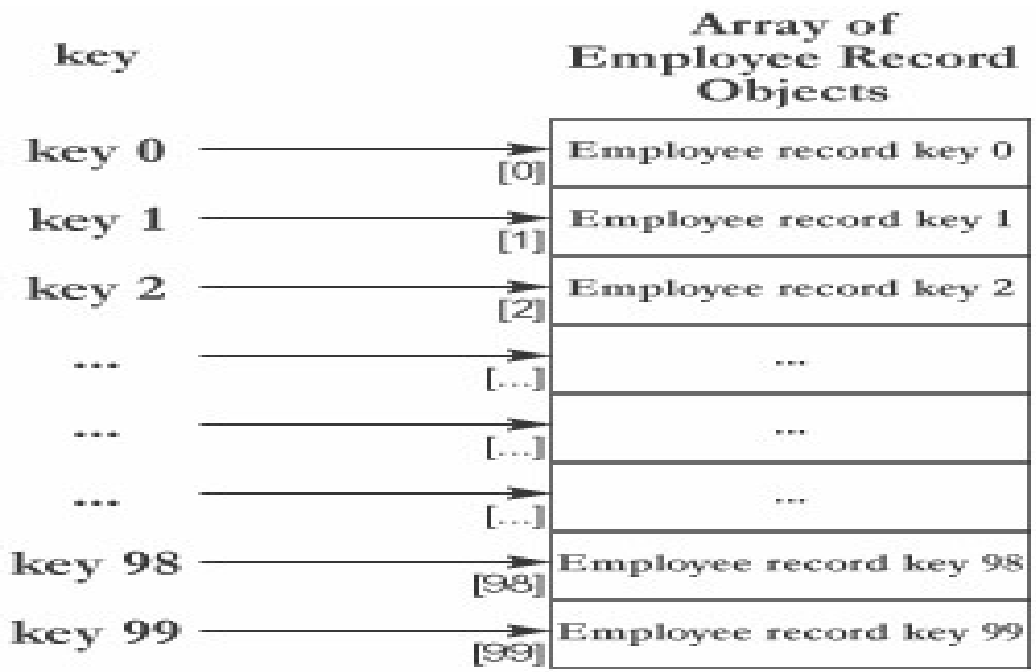
Code Snippet

```
class Phone...

class Program
{
    static void Main(string[] args)
    {
        List<Phone> mobiles = new List<Phone>();//Initially no Phones will be available
        mobiles.Add(new Phone { PhoneID = 111, PhoneModel = "N73", PhoneCost = 13200 });
        mobiles.Add(new Phone { PhoneID = 112, PhoneModel = "xPhoria", PhoneCost = 18000 });
        mobiles.Add(new Phone { PhoneID = 113, PhoneModel = "5800Xpress", PhoneCost = 12200 });
        //Phones are added using Add Method...

        mobiles.Remove(mobiles[2]);
    }
}
```

PRATIAN TECHNOLOGIES

Dictionary<TK,TV>

- Stores the data as Key-Value Pairs
- Provides a mapping from a set of keys to a set of values.
- Data accessed by the Keys.
- Dictionaries are Dynamic, grows if needed.



PRATIAN TECHNOLOGIES

The Class Structure

IDictionary<TKey, TValue>
ICollection<KeyValuePair<TKey, TValue>>
IEnumerable<KeyValuePair<TKey, TValue>>
IDictionary
ICollection
IEnumerable
ISerializable
IDeserializationCallback

Dictionary<TKey, TValue>
Generic Class

Properties
- Comparer : IEqualityComparer<TKey>
- Count : int
- Keys : KeyCollection
- this : TValue
- Values : ValueCollection

Methods
- Add() : void
- Clear() : void
- ContainsKey() : bool
- ContainsValue() : bool
- Remove() : bool

Nested Types

PRATIAN TECHNOLOGIES

Important Members

Method	Description
Add(TK k, TV v)	Adds the key/value pair specified by k and v to the dictionary. If the k is already in the dictionary, then its value is unchanged and an ArgumentException is thrown.
ContainsKey(TK k)	Returns true if k is a key in the invoking dictionary. Returns false otherwise.
ContainsValue(TV v)	Returns true if v is a value in the invoking dictionary. Returns false otherwise.
GetEnumerator( )	Returns an enumerator for the invoking dictionary.
Remove(TK k)	Removes k from the dictionary. Returns true if successful. Returns false if k was not in the dictionary.

PRATIAN TECHNOLOGIES

Code snippet.

```
Dictionary<string, string> Users = new Dictionary<string, string>();
/*Here the users is a collection which stores the
 * userid and password pairs of Each Employee*/
Users.Add("userID1", "pwd2");
Users.Add("userID2", "pwd12");
Users.Add("userID3", "pwd123");
Users.Add("userID3", "pwd123");
/*Throws arguement exception saying that
the key called userID3 already exists in the Collection.
 This Implies Only Unique Keys should be available.*/

if (Users.ContainsKey("userID3"))//Check if the Key already exists....
{
    //Code to tell that the User already exists...
}
else
    Users.Add("userID3", "pwd123");
//Similarly U can check for Value also....
```

PRATIAN TECHNOLOGIES

HashSet<T>

- Provides unordered list of Distinct Items.
- A set is a collection that contains no duplicate elements, and whose elements are in no particular order.
- Use set only when you don't want Duplicate entries into your Collection.
- HashSet is almost the same as your Algebra's set functionality.

PRATIAN TECHNOLOGIES

Important Members

Method	Description
Add(T element)	Adds the specified element to a set. Returns false if element already exists.
UnionWith	Modifies the current HashSet<T> object to contain all elements that are present in itself, the specified collection, or both.
IntersectWith	Changes the set to include only elements that are part of both the collection that is passed and the set
RemoveWhere	This method removes all elements on the condition that it match. The Condition is given as function thro a Delegate object called Predicate.
ExceptWith	Receives a collection as argument and removes all the elements from this collection from the set.

PRATIAN TECHNOLOGIES

Example

```
HashSet<string> IPLTeams = new HashSet<string>
{
    "Kolkatta Knight Riders",
    "Mumbai Indians",
    "Royal Chalengers",
    "Deccan Chargers",
    "Chennai Super Kings",
    "KingsXI Punjab"
};

HashSet<string> NewTeams = new HashSet<string>
{
    "Pune Warriors",
    "Kochi Tigers",
    "Mysore Kings",
    "Gujarat Patels"
};

if (NewTeams.Add("Oddisi Devils"))
    Console.WriteLine("New Team Added");
if (!NewTeams.Add("Kochi Tigers"))
    Console.WriteLine("They are already Included");
```

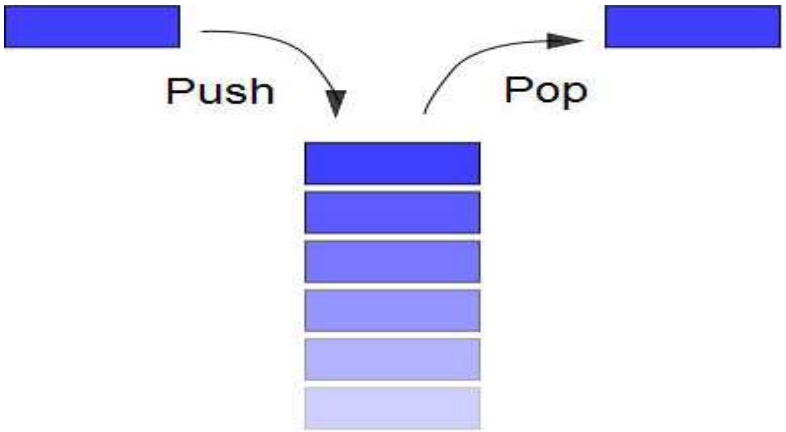
PRATIAN TECHNOLOGIES

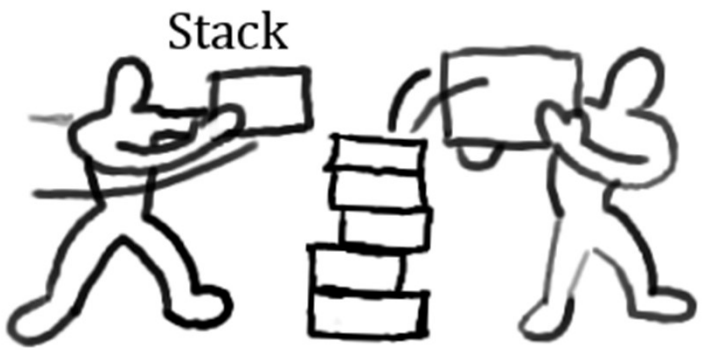
Working with Multiple sets

```
HashSet<string> CompleteTeams = new HashSet<string>();
CompleteTeams.UnionWith(IPLTeams);
CompleteTeams.UnionWith(NewTeams);
foreach (var team in CompleteTeams)
    Console.WriteLine(team);//Displays all the Teams with no Duplicates.

CompleteTeams.ExceptWith(NewTeams);
//Removes all NewTeams from the Current Set.
CompleteTeams.IntersectWith(NewTeams);
//Removes all Other Teams that are not the part of NewTeams
```

Stack<T>

- Stack<T>.
  - Stores the data as First In- Last Out manner.

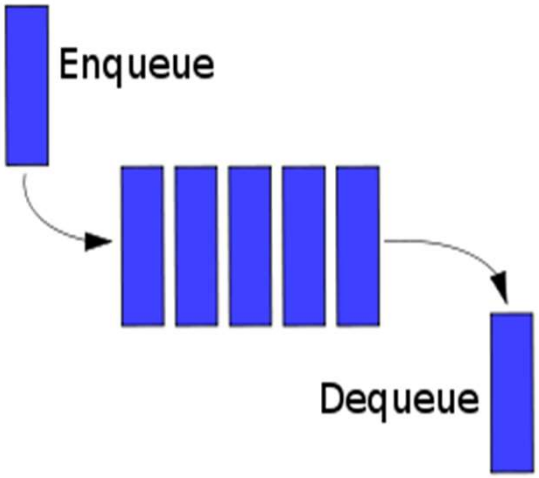




PRATIAN TECHNOLOGIES

Queue<T>

- Queue<T>.
  - Stores the Data as First-In- First Out manner.



PRATIAN TECHNOLOGIES

Generic Collections

- Demo

PRATIAN TECHNOLOGIES

Custom Collections

- What?
  - Classes created to make its object usable in a simple iteration like a for each statement.
- Why?
  - Creating a Collection that suits by my business needs without unnecessary functions.
- How?
  - Implement the interfaces of the Collections to suit your business needs.

PRATIAN TECHNOLOGIES

Collection Comparision

Data Type			Inherits from		
Array	Index Based	Generic Fixed Length	IEnumerable<T> Icomparable<T> Iequatable<T>	String[] strarr=new string[10]; It is strongly data type with fixed sized, so it is fast	makes limitation by fixed size
ArrayList	Index Based	Non Generic	Ilist Icollection IEnumerable	Arraylist objarr=new Arraylist No Data Type + No Dimension	Difficult to find item just via Index ,No Key
HashTable	Key Value Pair	Non Generic	Idictionary Icollection IEnumerable Iserializable	Determine an index for each item Hashtable objhash=new Hashatable(); Objhash.add(1001,"Mahsa")	Arraylist is faster due to hashtable must do key conversion and it consume time
IDictionary<T>	Key Value Pair	Generic	Icollection IEnumerable	Similar to Hashtable but it is generic	
List<T>	Index Based	Generic	Ilist<T> Icollection<T> IEnumerable<T>	Like Array is strong type Like ArrayList has No Dimension Modify(Add,Remove)	
Ilist<T>	Index Based	Generic	Icollection<T> IEnumerable<T>	Modify(Add,Remove) ***Interface helps to future changes***	Ilist can find the indexof item , **IList[i] √**
ICollection<T>	Randomly	Generic	IEnumerable<T>	Modify(Add,Remove), countable	Randomly,~~**ICollection[i]**~~
IEnumerable<T>	Index Based	Generic	IEnumerable	IEnumerable is read only , suitable to iterate through collection and cannot modify	bring all data from server to client and then filter them
IQueryable<T>	Index Based	Generic	IEnumerable<T>	Iqueryable is suitable for runtime query and reduce overhead from memory, improve performance	Iqueryable bring filtered data from server to client NOT all of them
Stack	Prioritized		Icollection IEnumerable	Non Generic Stack: System.Collections.Stack Generic Stack: System.Collections.Generic.Stack<int>	
Queue	Prioritized		Icollection IEnumerable	Non Generic Queue: System.Collections.Queue Generic Queue: System.Collections.Generic.Queue<int>	

PRATIAN TECHNOLOGIES

Interfaces of Generics

Interfaces	Decription
IEnumerable<T>	Exposes the enumerator, which supports a simple iteration over a collection of a specified type.
ICollection<T>	Defines methods to manipulate generic collections
IList<T>	Represents a collection of objects that can be individually accessed by index
IEnumerator<T>	Supports a simple iteration over a generic collection
IComparer<T>	Defines a method that a type implements to compare two objects
IDictionary<TK,TV>	Represents a generic collection of key/value pairs.
IEqualityComparer<T>	Defines methods to support the comparison of objects for equality

PRATIAN TECHNOLOGIES

IEnumerable<T>

- Fundamental feature of any Collection
- Provides an Ability to Enumerate.
- Returns an IEnumerator thro which you can iterate.
- Any Class which implements this interface, could be used to iterate using a for each statement.

PRATIAN TECHNOLOGIES

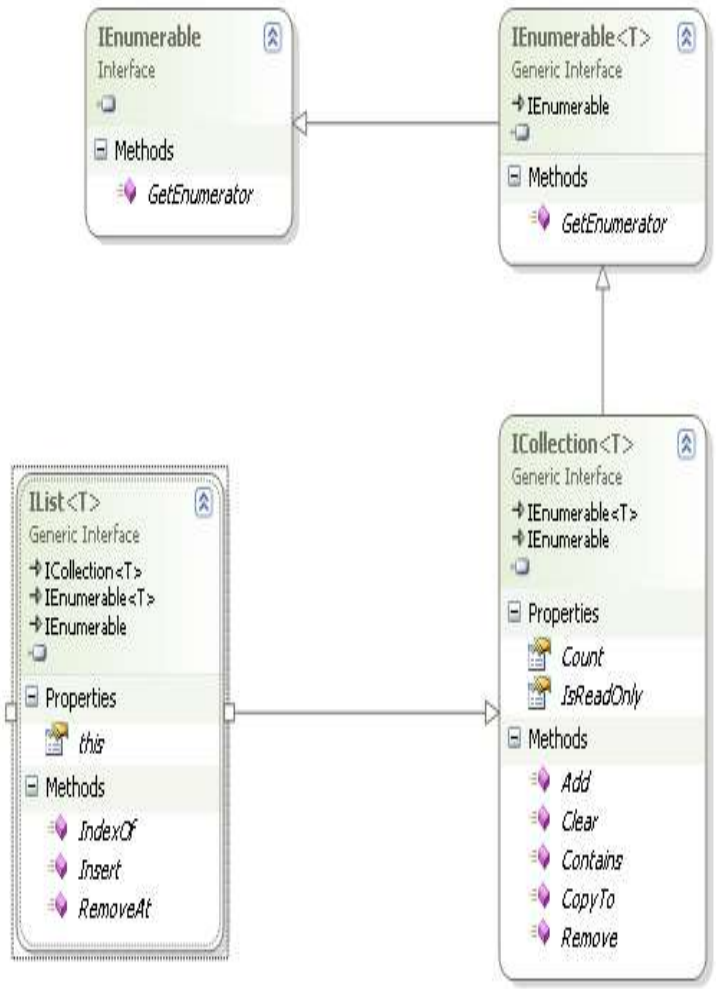
yield

- Used in an iterator block to provide a value to the enumerator object or to signal the end of iteration.
- The yield return statement returns the next object in the collection.

```
public IEnumerator<Employee> GetEnumerator()
{
    foreach (var emp in employees)
        yield return emp;
}
```

PRATIAN TECHNOLOGIES

IList<T>



- Represents a collection of objects that can be individually accessed by index.
- Functions
  - Insert
  - RemoveAt

PRATIAN TECHNOLOGIES

IComparable

- Implement this interface for comparing the current object with another object.
- This provides an ability to compare 2 objects on a certain condition.
- Classes that implement IList uses IComparable to sort the Data by calling Sort Method.
- All Built In types of .NET implement IComparable.
- Using its method, we could provide the functionality of Sorting of our Data on a certain Condition.
- You can make any class work with IList's built-in Sort Function by having it implement IComparable.

- Example.

```
public interface IComparable
{
    int CompareTo(object obj);
}
```

PRATIAN TECHNOLOGIES

IComparer Interface

- If you want the List to Compare your object on a multiple Conditions, then create a Class that implements IComparer interface.

```
public interface IComparer<T>
{
    int Compare(T x, T y);
}
```

- Implementing this Interface allows 2 independent objects to be compared on Multiple Conditions.
- This leads to apply this logic on Sorting further with multiple conditions.
- Example that Sorts on Name and if 2 Names are same, it Sorts by Address for those Names.

PRATIAN TECHNOLOGIES

Collections Comparison

- Demo

PRATIAN TECHNOLOGIES

PRATIAN TECHNOLOGIES

Collections and Generics

1. Which among the following are the ordered collection class?

a) BitArray

b) Queue

c) HashTable

d) Stack

PRATIAN TECHNOLOGIES

Collections and Generics

2. Which among the following is **not** an interface declared in System.Collection namespace?

a) IDictionaryComparer

b) IEnumerable

c) IEnumerator

d) IComparer

PRATIAN TECHNOLOGIES

Collections and Generics

3. Which among the following is the correct way to find out the number of elements currently present in an ArrayList Collection called arr?

a) arr.Capacity

b) arr.Count

c) arr.MaxIndex

d) arr.UpperBound

PRATIAN TECHNOLOGIES

Collections and Generics

4. Which statement is correct about the C#.NET code snippet given below?

```
1. Stack st = new Stack();
2. st.Push("Csharp");
3. st.Push(7.3);
4. st.Push(8);
5. st.Push('b');
6. st.Push(true);
```

a) Un-similar elements like “Csharp”,7.3,8 cannot be stored in the same stack collection.
b) Boolean values can never be stored in Stack collection
c) Perfectly workable code
d) All of the mentioned

PRATIAN TECHNOLOGIES

Collections and Generics

5. Which among the following is the correct way to access all the elements of the Stack collection created using the C#.NET code snippet given below?

```
Stack st = new Stack();
st.Push(10);
st.Push(20);
st.Push(-5);
st.Push(30);
st.Push(6);
```

a) IEnumerable e;
e = st.GetEnumerator();
while (e.MoveNext())
Console.WriteLine(e.Current);

b) IEnumerator e;
e = st.GetEnumerator();
while(e.MoveNext())
Console.WriteLine(e.Current);

c) IEnumerable e;
e = st.GetEnumerable();
while(e.MoveNext())
Console.WriteLine(e.Current);

d) None of the mentioned

PRATIAN TECHNOLOGIES

Collections and Generics

6. Which statements among the following are correct about the Collection Classes available in Framework Class Library?

a) Elements of a collection cannot be transmitted over a network

b) Elements stored in a collection can be modified only if all the elements are of similar types

c) Elements stored in a Collection can be retrieved but cannot be modified

d) Collection classes make use of efficient algorithms to manage the collection, hence improving performance of the program

PRATIAN TECHNOLOGIES

Collections and Generics

7. Among the given collections which one is I/O index based?

a) ArrayList

b) BitArray

c) Stack

d) Queue

PRATIAN TECHNOLOGIES

Collections and Generics

8. What is meant by the term generics?

a) parameterized types

b) class

c) structure

d) interface

PRATIAN TECHNOLOGIES

Collections and Generics

9. Choose the advantages of using generics?

a) Generics facilitate type safety

b) Generics facilitate improved performance and reduced code

c) Generics promote the usage of parameterized types

d) All of the mentioned

PRATIAN TECHNOLOGIES

Collections and Generics

9. What does the following code block defines?

```
1.  class Gen<T> {
2.      T ob;
3.  }
```

a) Generics class declaration

b) Declaration of variable

c) a simple class declaration

d) All of the mentioned

PRATIAN TECHNOLOGIES

Collections and Generics

10. What will be the output of the given code snippet?

```
1. public class Generic<T>
2. {
3.     Stack<T> stk = new Stack<T>();
4.     public void push(T obj)
5.     {
6.         stk.Push(obj);
7.     }
8.     public T pop()
9.     {
10.         T obj = stk.Pop();
11.         return obj;
12.     }
13. }
14. class Program
15. {
16.     static void Main(string[] args)
17.     {
18.         Generic<string> g = new Generic<string>();
19.         g.push(40);
20.         Console.WriteLine(g.pop());
21.         Console.ReadLine();
22.     }
23. }
```

a) 0

b) Runtime Error

c) 40

d) Compile time Error