Generic Collections in C#

The non-generic collection classes such as `ArrayList`, `Stack`, `Queue`, `Hashtable`, etc operate on the object data type. As they operate on object data type hence they are loosely typed. Loosely typed means you can store any type of values into the collection. Because of this loosely typed nature, we may get runtime errors. Not only we get run time errors because of the loosely-typed nature, but it also affects the performance of the application due to boxing and unboxing. The problems of non-generic collections are overcome by the Generic collections in C#.

The .NET framework has re-implemented all the existing collection classes such as `ArrayList`, `Stack`, and `Queue`, etc. in Generic Collections such as `List<T>`, `Stack<T>` and `Queue<T>`. Here `T` is nothing but the type of values that we want to store into the collection. So, the most important point that you need to remember is while creating the objects of Generic Collection classes, you need to explicitly provide the type of values that you are going to store into the collection.

The Generic Collection classes are implemented under the `System.Collections.Generic` namespace. The classes which are present in this namespace are as follows.

```
Stack<T>, Queue<T>, LinkedList<T>, SortedList<T>, List<T>,
Dictionary<TKey, TValue>
```

Note: Here the `<T>` refers to the type of values we want to store under them.

The Generic Collections in C# are strongly typed. The strongly typed nature allows these collection classes to store only one type of value into it. This not only eliminates the type mismatch at runtime but also we will get better performance as they don't require boxing and unboxing while they store value type data.

So, it is always a preferable and a good programming choice to use the generic collection classes rather than using the non-generic collection classes.

List<T>

The Generic List in C# is a collection class which is present in `System.Collections.Generic` namespace. The List Collection class is one of the most widely used generic collection classes in real-time applications. This Generic List collection class represents a strongly typed list of objects which can be accessed by using the index. It also provides methods that can be used for search, sort and manipulate the list items.

Example:

```
public class Program
{
    public static void DisplayList(List<Employee> employeesList)
    {
        foreach (Employee employee in employeesList)
        {
            Console.WriteLine("ID = {0}, Name = {1}, Gender = {2}, Salary = {3}",
                    employee.ID, employee.Name, employee.Gender, employee.Salary);
```

```
        }
        Console.WriteLine();
    }
    public static void Main(string[] args)
    {
        Employee emp1 = new Employee() { ID = 101, Name = "Dhaval", Gender =
                                         "Male", Salary = 5000 };
        Employee emp2 = new Employee() { ID = 102, Name = "Prachi", Gender =
                                         "Female", Salary = 7000 };
        Employee emp3 = new Employee() { ID = 103, Name = "Yash", Gender =
                                         "Male", Salary = 5500 };
        Employee emp4 = new Employee() { ID = 104, Name = "Nirav", Gender =
                                         "Male", Salary = 6500 };
        Employee emp5 = new Employee() { ID = 105, Name = "Divya", Gender =
                                         "Female", Salary = 7500 };
        Employee emp6 = new Employee() { ID = 106, Name = "Chirag", Gender =
                                         "Male", Salary = 8500 };
        List<Employee> listEmployees = new List<Employee>();
        Console.WriteLine("Add and AddRange Method");
        //Use Add() method to add one item at a time to the end of the list
        listEmployees.Add(emp1);
        listEmployees.Add(emp2);
        listEmployees.Add(emp3);
        //Create another list
        List<Employee> AnotherlistEmployees = new List<Employee>();
        AnotherlistEmployees.Add(emp4);
        AnotherlistEmployees.Add(emp5);
        AnotherlistEmployees.Add(emp6);
        //Use AddRange() method to add another list of items
        listEmployees.AddRange(AnotherlistEmployees);
        DisplayList(listEmployees);
        Console.WriteLine("GetRange Method");
        //Use GetRange() method to returns a range of items from the list.
        List<Employee> ListOfNewEmployees = listEmployees.GetRange(3, 3);
        DisplayList(ListOfNewEmployees);
        Console.WriteLine("Remove RemoveAt and RemoveAll");
        //Use Remove() method to removes only the first matching item from list.
        listEmployees.Remove(emp1);
        //Use RemoveAt() method to remove an item from a specified index.
        // listEmployees.RemoveAt(0);
        //Use RemoveAll() method to removes all the items from a collection
        // that matches the specified condition.
        listEmployees.RemoveAll(x => x.Gender == "Female");
        DisplayList(listEmployees);
        Console.WriteLine("RemoveRange Method");
        // Use RemoveRange() method to removes a range of elements from the list.
        listEmployees.RemoveRange(0, 2);
        DisplayList(listEmployees);
        Console.WriteLine("Insert and InsertRange Method");
        //Use Insert() method to insert a single item at a specfic position
        listEmployees.Insert(0, emp1);
        listEmployees.Insert(1, emp2);
```

```
        //Use InsertRange() to insert another list at a specified position
        listEmployees.InsertRange(0, AnotherlistEmployees);
        DisplayList(listEmployees);
        Console.WriteLine("Clear Method");
        // Use Clear method to remove all the items from the list collection
        listEmployees.Clear();
        Console.WriteLine("Total Items in the List After Clear function = " +
                                    listEmployees.Count);
        Console.ReadKey();
    }
}
public class Employee
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Gender { get; set; }
    public int Salary { get; set; }
}
```

Dictionary<TKey, TValue>

The Dictionary in C# is a Collection class same as HashTable i.e. used to store the data in the form of Key Value Pairs, but here while creating the dictionary object you need to specify the type for the keys as well as the type for values also. A dictionary is a collection of (key, value) pairs. The Dictionary Generic Collection class is present in System.Collections.Generic namespace. When creating a dictionary, we need to specify the type for the key and as well as for the value. The fastest way to find a value in a dictionary is by using the keys. Keys in a dictionary must be unique.

Example:

```
public class Program
{
    static void Main(string[] args)
    {
        //Create Employee object
        Employee emp1 = new Employee() { ID = 101, Name = "Dhaval", Gender =
                                            "Male", Salary = 20000 };
        Employee emp2 = new Employee() { ID = 102, Name = "Divya", Gender =
                                            "Female", Salary = 30000 };
        Employee emp3 = new Employee() { ID = 103, Name = "Yash", Gender =
                                            "Male", Salary = 40000 };
        // Create a Dictionary collection where Employee ID is the key and the
        // key Type is int Employee object is the value and the value Type is
        // Employee
        Dictionary<int, Employee> dictionaryEmployees = new Dictionary<int,
                                                                Employee>();
        // Add Employee objects to the dictionary collection Employee ID is the
        // key and the employee object is the value
        dictionaryEmployees.Add(emp1.ID, emp1);
        dictionaryEmployees.Add(emp2.ID, emp2);
        dictionaryEmployees.Add(emp3.ID, emp3);
```

```
        // Retrieve the value (i.e. Employee object) from the dictionary using
        // the key (i.e. Employee ID). The fastest way to get a value from the
        // dictionary is by using its key
        Employee employee101 = dictionaryEmployees[101];
        Console.WriteLine("Employee 101 in employee dictionary");
        Console.WriteLine("ID = {0}, Name = {1}, Gender = {2}, Salary = {3}",
                          employee101.ID, employee101.Name, employee101.Gender,
                             employee101.Salary);
        Console.WriteLine();
        // It is also possible to loop thru each key/value pair in a dictionary
        Console.WriteLine("All Employees keys and values in employee
                             dictionary");
        foreach (KeyValuePair<int, Employee> employeeKeyValuePair in
                             dictionaryEmployees)
        {
            Console.WriteLine("Key = " + employeeKeyValuePair.Key);
            Employee emp = employeeKeyValuePair.Value;
            Console.WriteLine("ID = {0}, Name = {1}, Gender ={2}, Salary = {3}",
                             emp.ID, emp.Name, emp.Gender, emp.Salary);
        }
        Console.WriteLine();
        // We can also use implicitly typed variable var to loop thru each
        // key/value pair in a dictionary. But try to avoid using var, as this
        // makes our code less readable
        Console.WriteLine("All Employees keys and values in employee
                             dictionary");
        foreach (var employeeKeyValuePair in dictionaryEmployees)
        {
            Console.WriteLine("Key = " + employeeKeyValuePair.Key);
            Employee emp = employeeKeyValuePair.Value;
            Console.WriteLine("ID = {0}, Name = {1}, Gender ={2}, Salary = {3}",
                             emp.ID, emp.Name, emp.Gender, emp.Salary);
        }
        Console.WriteLine();
        // To get all the keys in the dictionary we have to use the keys
        // properties of dictionaryCustomers object as shown below
        Console.WriteLine("All Keys in Employee Dictionary");
        foreach (int key in dictionaryEmployees.Keys)
        {
            Console.WriteLine(key + " ");
        }
        Console.WriteLine();
        // First get the keys, then get the values using the keys
        Console.WriteLine("All Keys and values in Employee Dictionary");
        foreach (int key in dictionaryEmployees.Keys)
        {
            Console.WriteLine(key + " ");
            Employee emp = dictionaryEmployees[key];
            Console.WriteLine("ID = {0}, Name = {1}, Gender ={2}, Salary = {3}",
```

```
                                emp.ID, emp.Name, emp.Gender, emp.Salary);
        }
        Console.WriteLine();
        //To get all the values in the dictionary use Values property
        Console.WriteLine("All employees objects in Employee Dictionary");
        foreach (Employee emp in dictionaryEmployees.Values)
        {
            Console.WriteLine("ID = {0}, Name = {1}, Gender ={2}, Salary = {3}",
                                emp.ID, emp.Name, emp.Gender, emp.Salary);
        }
        // If we try to add a key that already exists in the dictionary we will
        // get an exception - An item with the same key has already been added.
        // So, check if the key already exists
        if (!dictionaryEmployees.ContainsKey(101))
        {
            dictionaryEmployees.Add(101, emp1);
        }
        Console.WriteLine();
        // When accessing a dictionary value by key, make sure the dictionary
        // contains the key,otherwise we will get KeyNotFound exception.
        if (dictionaryEmployees.ContainsKey(110))
        {
            Employee emp = dictionaryEmployees[110];
        }
        else
        {
            Console.WriteLine("Key does not exist in the dictionary");
        }
        Console.ReadKey();
    }
}
public class Employee
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Gender { get; set; }
    public int Salary { get; set; }
}
```

Collection Initializers

This is a new feature added to C# 3.0 which allows initializing a collection directly at the time of declaration like an array.

A `Dictionary<TKey,TValue>` contains a collection of key/value pairs. Its Add method takes two parameters, one for the key and one for the value. To initialize a `Dictionary<TKey,TValue>`, or any collection whose Add method takes multiple parameters, enclose each set of parameters in braces.

Example:

```
public class Program
{
    static void Main(string[] args)
    {
        // Collection initializer
        // Initializing the collection directly at the time of declaration
        Dictionary<int, Employee> dictionaryEmployees = new Dictionary<int,
                                                                Employee>()
        {
            { 101, new Employee {ID=101, Name="Dhaval", Gender="Male", Salary =
                  20000}},
            { 102, new Employee {ID=101, Name="Divya", Gender="Female", Salary
                  = 30000}},
            { 103, new Employee {ID=101, Name="Yash", Gender="Male", Salary =
                  40000}}
        };
        //To get all the values in the dictionary use Values property
        Console.WriteLine("All employees objects in Employee Dictionary");
        foreach (Employee emp in dictionaryEmployees.Values)
        {
            Console.WriteLine("ID = {0}, Name = {1}, Gender ={2}, Salary = {3}",
                              emp.ID, emp.Name, emp.Gender, emp.Salary);
        }
        Console.ReadKey();
    }
}
public class Employee
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Gender { get; set; }
    public int Salary { get; set; }
}
```