Collections in C#

Back to: C#.NET Tutorials For Beginners and Professionals

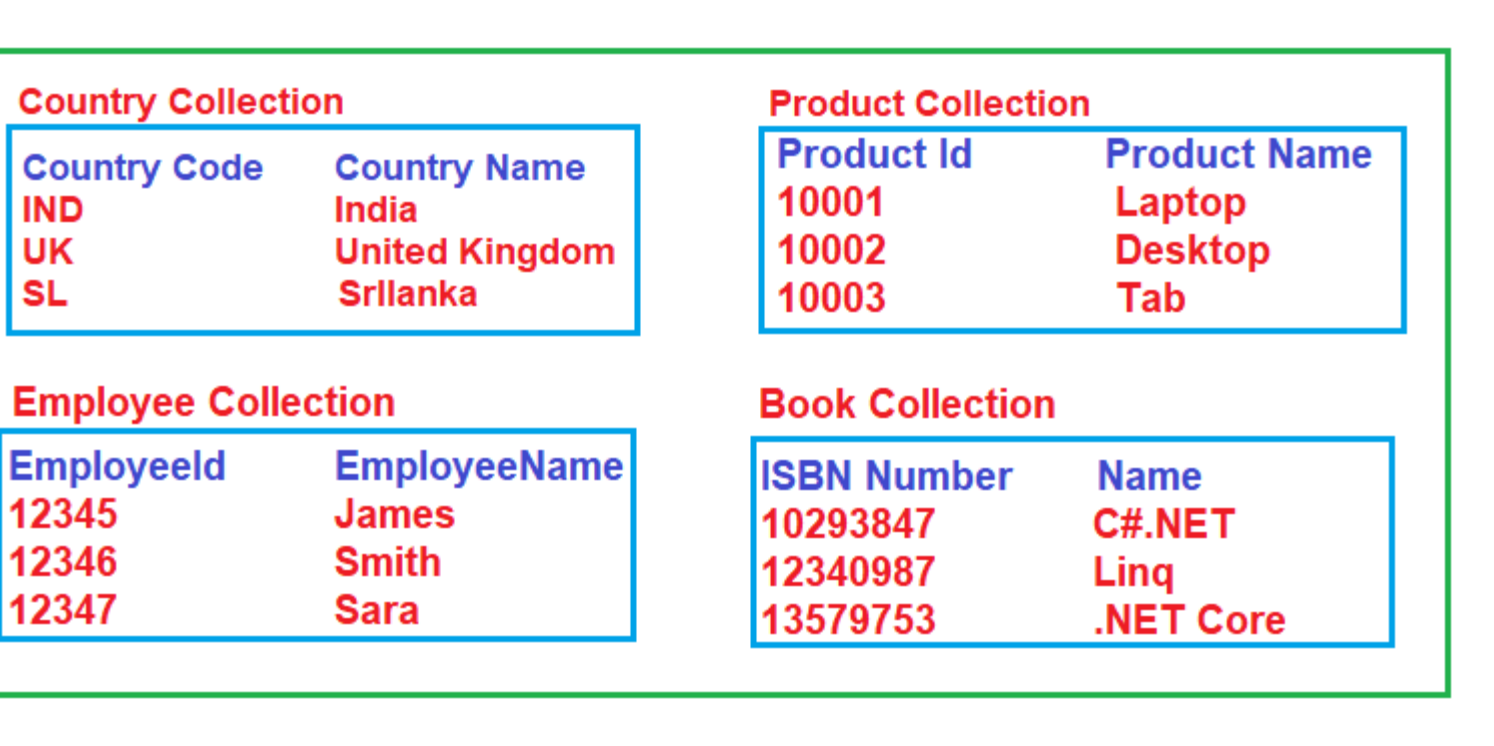
Introduction to Collections in C#

In this article, I am going to give you a brief Introduction to **Collections in C#**. Please read our previous article where we discussed the **Advantages and Disadvantages of Arrays in C#** with Examples. Collections are similar to Arrays, it provides a more flexible way of working with a group of objects. As part of this article, we are going to discuss the following pointers in detail.

1. **Introduction to Collections?**
2. **General Categories of Collections?**
3. **What are Array and their Disadvantages in C#?**
4. **What are collections in C#?**
5. **How do collections overcome the problems of Array in C#?**
6. **What are the different types of Collections available in C#?**

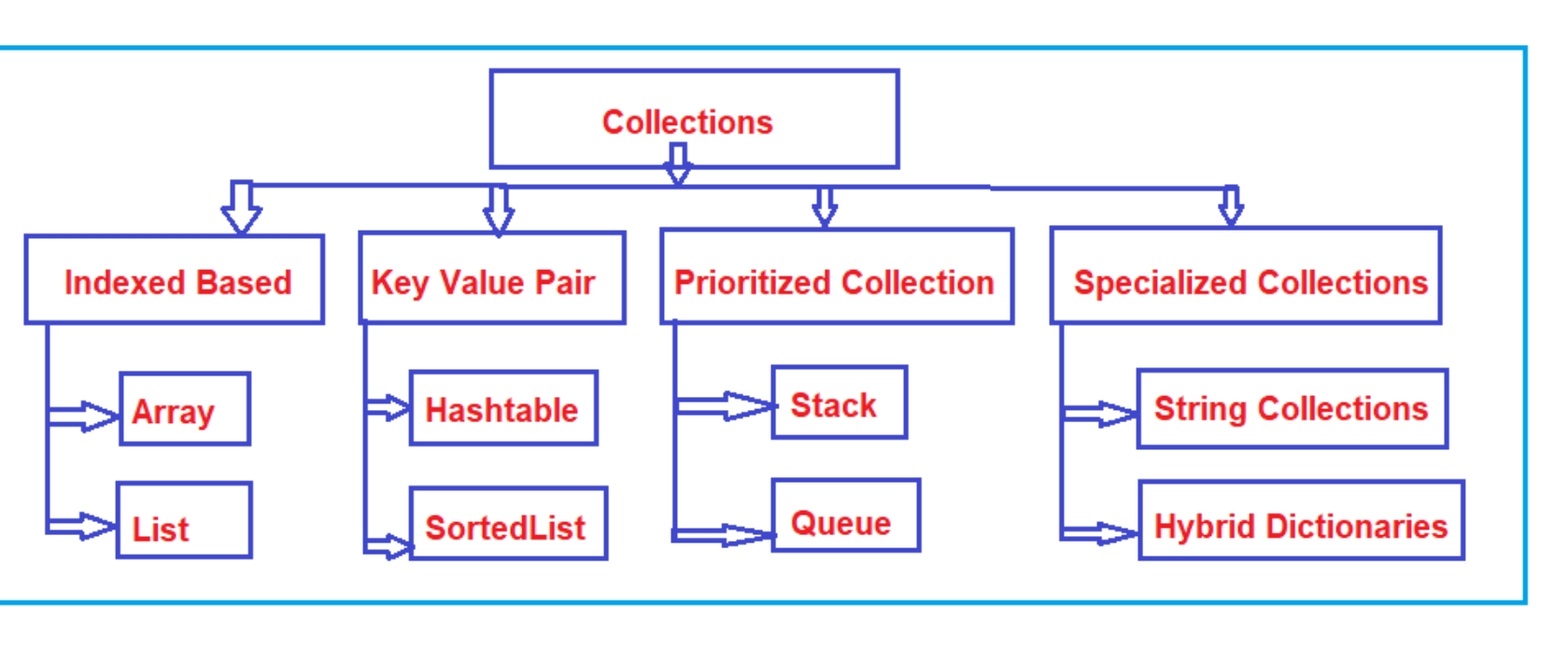
Introduction to Collections:

Collections are nothing but groups of records that can be treated as one logical unit. For example, you can have a country collection that can have a country code and country name. You can have a product collection that has the Product Id and Product Name. You can have an employee collection having the employee's name and employee id. You can have a book collection having an ISBN number and book name. For a better understanding, please have a look at the below image.



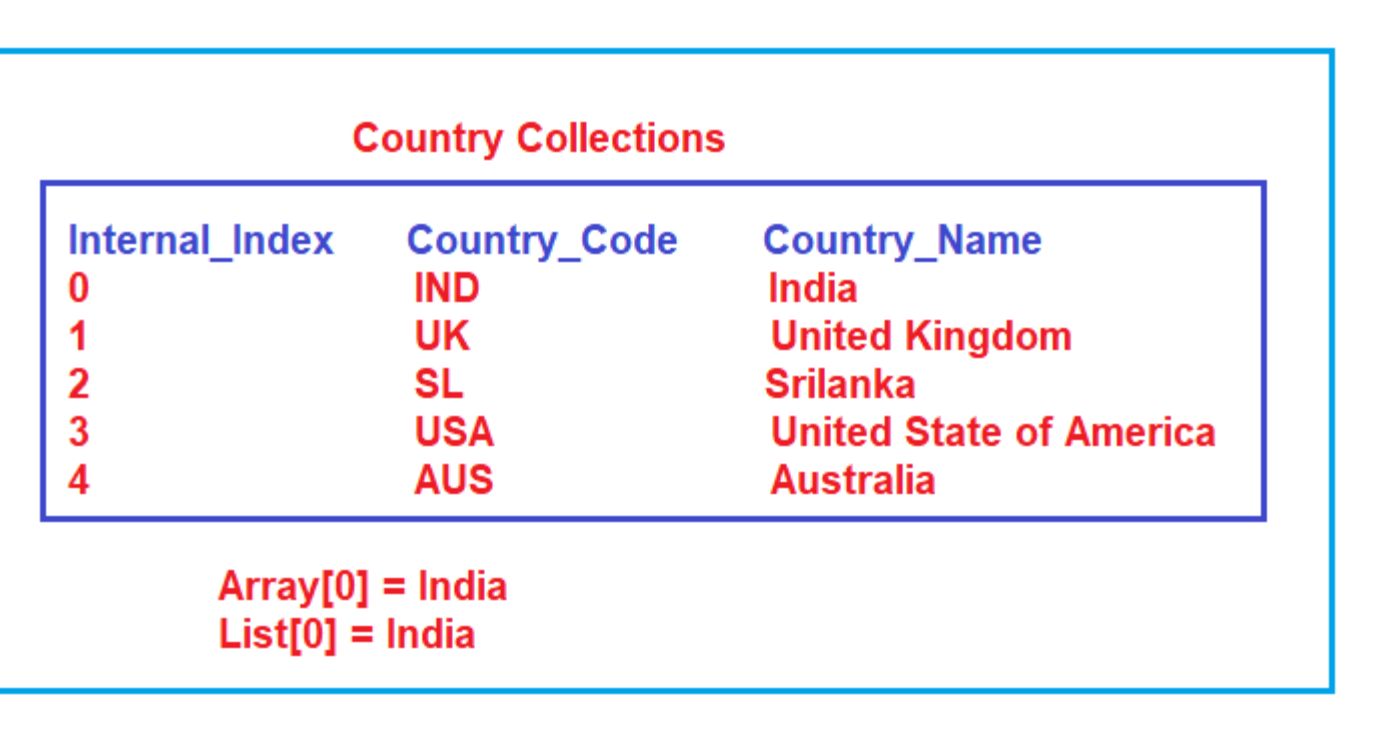
General Categories of Collections:

Collections are classified into 4 types such as Indexed Based, Key-Value Pair, Prioritized Collection, and Specialized Collections. For a better understanding, please have a look at the below diagram.



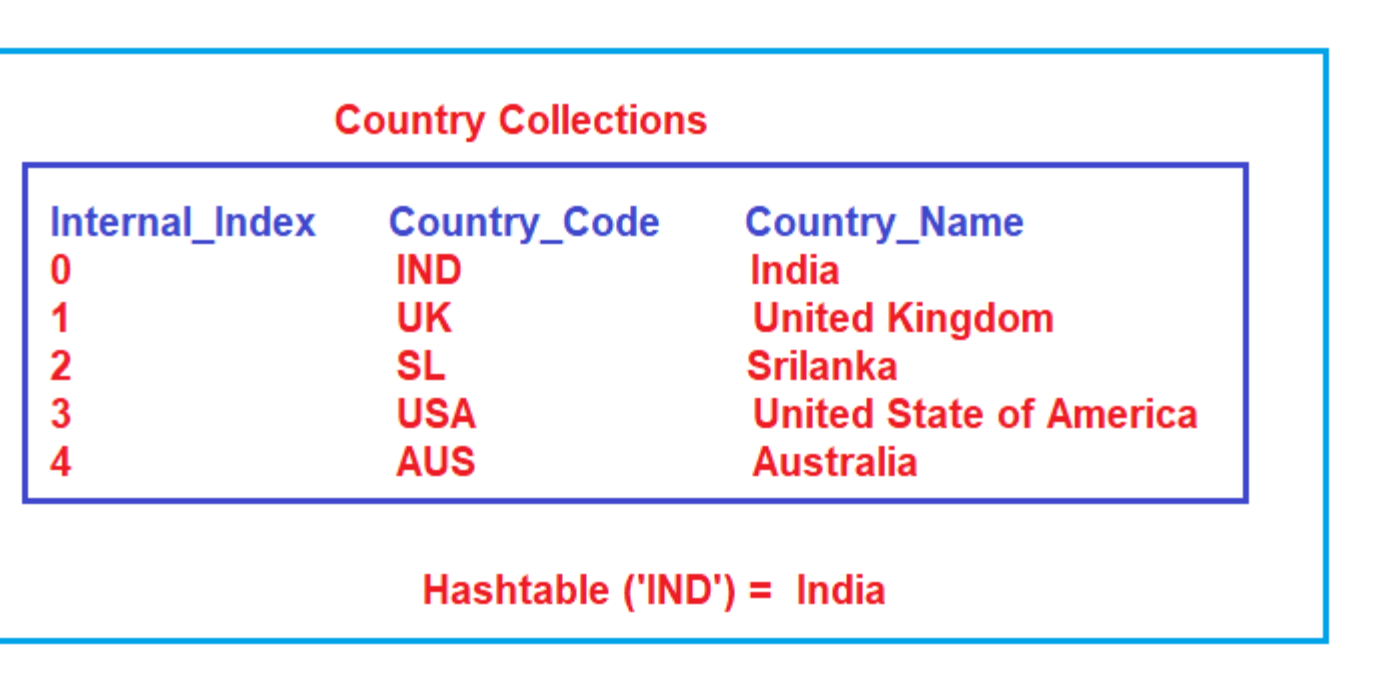
Indexed Base Collections:

In Indexed Based, we have two kinds of collections i.e. Array and List. To understand the Indexed Based collection, please have a look at the following country collection. So, when we add any elements to .NET Collections like Array, List, or ArrayList, it maintains its own internal index number. This internal index number is auto-generated by the framework and using this index number we can identify the records.



Key-Value Pair Collections

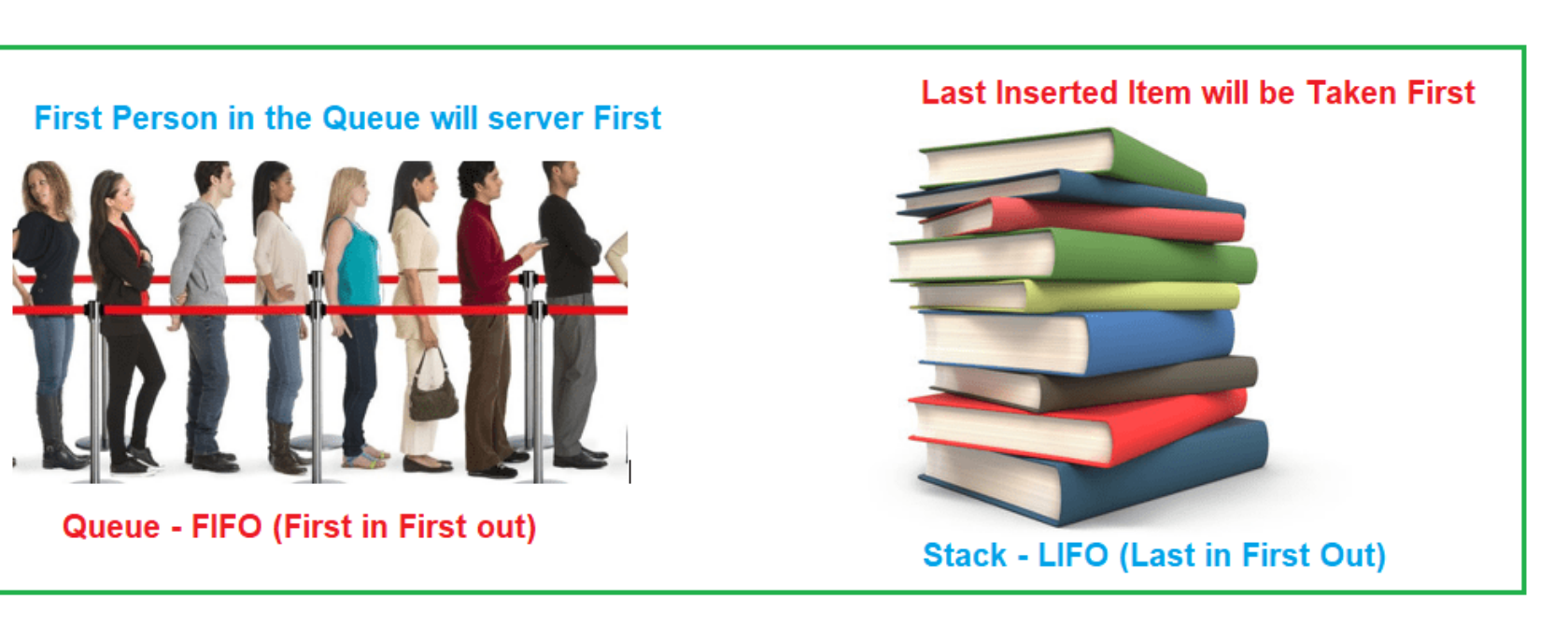
In the Key-Value Pair collection, we have Hashtable, Dictionary, and SortedList. In real-time projects, we rarely accessed the records using the internal index numbers. We generally use some kind of keys to identify the records and there we use the Key-Value Pair collections like Hashtable, Dictionary, and SortedList. For a better understanding, please have a look at the below diagram.



So, basically, if we want to fetch the records based on a key, then we need to use Key-Value Pair collections such as Hashtable, Dictionary, and SortedList.

Prioritized Collections:

The Prioritized Collections help you to access the elements in a particular sequence. The Stack and Queue collections belong to the Prioritized Collections category. If you want First in First Out (FIFO) access to the elements of a collection, then you need to use Queue collection. On the other hand, if you want Last in First Out (LIFO) access to the elements of a collection, then you need to use the Stack collection. For a better understanding, please have a look at the below image.



Specialized Collections:

The Specialized Collections are specifically designed for a specific purpose. For example, a Hybrid Dictionary starts as a list and then becomes a hashtable.

Now, let us understand what are the problems with the Traditional Array in C#, and how we can overcome such problems using collections in C#.

What are Array and Their disadvantages in C#?

In simple words, we can say that the Arrays in C# are the simple data structure that is used to store similar types of data items in sequential order. Although the arrays in C# are commonly used, they have some limitations.

For example, you need to specify the array's size while creating the array. If at execution time, you want to modify it that means if you want to increase or decrease the size of an array, then you need to do it manually by creating a new array or by using the Array class's Resize method, which internally creates a new array and copies the existing array element into the new array.

Following are the Limitations of Array in C#:

1. The array size is fixed. Once the array is created we can never increase the size of the array. If we want then we can do it manually by creating a new array and copying the old array elements into the new array or by using the Array class Resize method which will do the same thing means to create a new array and copy the old array elements into the new array and then destroy the old array.
2. We can never insert an element into the middle of an array
3. Deleting or removing elements from the middle of the array.

To overcome the above problems, the Collections are introduced in C# 1.0.

What is a Collection in C#?

The **Collections in C#** are a set of predefined classes that are present in the **System.Collections** namespace that provides greater capabilities and functionalities than the traditional arrays. The collections in C# are reusable, more powerful, and more efficient and most importantly they have been designed and tested to ensure quality and performance.

So in simple words, we can say a Collection in C# is a dynamic array**.** That means the collections in C# have the capability of storing multiple values but with the following features.

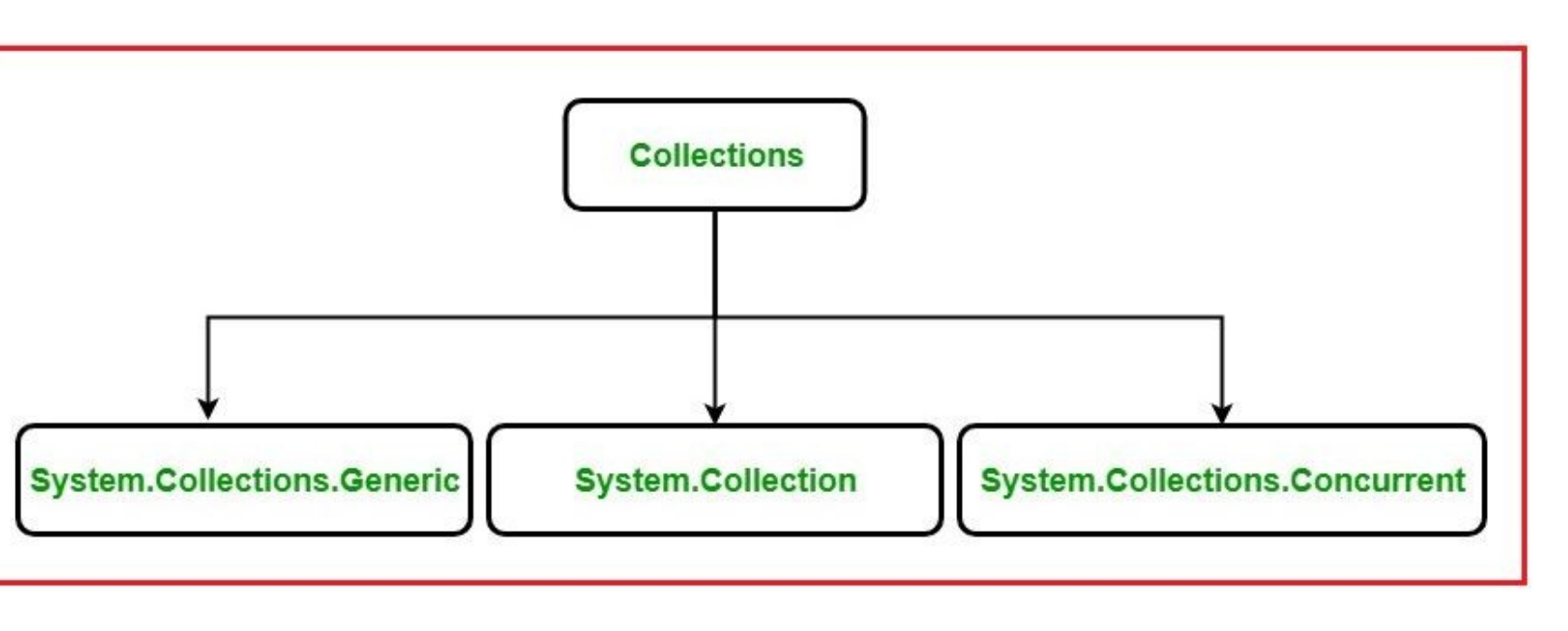
1. Size can be increased dynamically.
2. We can insert an element into the middle of a collection.
3. It also provides the facility to remove or delete elements from the middle of a collection.

The collections in C# are classes that represent a group of objects. With the help of C# Collections, we can perform different types of operations on objects such as Store, Update, Delete, Retrieve, Search, and Sort objects, etc. In short, all the data structure work can be performed by collections in C#. That means Collections standardize the way in which the objects are handled by our program.

Types of Collections in C#

There are 3 ways to work with collections. The three namespaces are given below:

1. System.Collections classes
2. System.Collections.Generic classes
3. System.Collections.Concurrent classes



Non-Generic Collections Classes in C#:

The Non-Generic Collection Classes come with C# 1.0 and are defined under the **System.Collections** namespace. The Non-Generic collection classes in C# operate on objects, and hence can handle any type of data, but not in a safe-type manner. The **System.Collections** namespace contains the following classes:

1. **ArrayList:** It Implements the System.Collections.IList interface using an array whose size is dynamically increased as required.
2. **Stack:** It represents a simple last-in-first-out (LIFO) non-generic collection of objects.
3. **Queue:** It represents a first-in, first-out collection of objects.
4. **Hashtable:** It represents a collection of key/value pairs that are organized based on the hash code of the key.
5. **SortedList:** It represents a collection of key/value pairs that are sorted by the keys and are accessible by key and by index.

Generic Collections Classes in C#:

The Generic Collection Classes come with C# 2.0 and are defined under the **System.Collection.Generic** namespace. It provides a generic implementation of standard data structures like linked lists, stacks, queues, and dictionaries. These collection classes are type-safe because they are generic means only those items that are type-compatible with the type of the collection can be stored in a generic collection, it eliminates accidental type mismatches. The System.Collections.Generic namespace has the following classes:

1. **List<T>:** It represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists.
2. **Stack<T>:** It represents a variable size last-in-first-out (LIFO) collection of instances of the same specified type.
3. **Queue<T>:** It represents a first-in, first-out collection of objects.
4. **HashSet<T>:** It represents a set of values. It removes duplicate elements from the collection.
5. **Dictionary<TKey, TValue>:** It represents a collection of keys and values.
6. **SortedList<TKey, TValue>:** It represents a collection of key/value pairs that are sorted by key based on the associated System.Collections.Generic.IComparer implementation.
7. **SortedSet<T>:** It represents a collection of objects that are maintained in sorted order.
8. **SortedDictionary<TKey, TValue>:** It represents a collection of key/value pairs that are sorted on the key.
9. **LinkedList<T>:** It represents a doubly linked list.

Concurrent Collection Classes in C#:

It came in .NET Framework Version 4 and onwards. It provides various threads-safe collection classes that are used in place of the corresponding types in the **System.Collections** and **System.Collections.Generic** namespaces, when multiple threads are accessing the collection simultaneously. The **System.Collections.Concurrent** namespace provides classes for thread-safe operations. Now multiple threads will not create problems for accessing the collection items. The System.Collections.Concurrent namespace has the following classes:

1. **BlockingCollection<T>:** It provides blocking and bounding capabilities for thread-safe collections that implement System.Collections.Concurrent.IProducerConsumerCollection.
2. **ConcurrentBag<T>:** It represents a thread-safe, unordered collection of objects.
3. **ConcurrentStack<T>:** It represents a thread-safe last-in-first-out (LIFO) collection.
4. **ConcurrentQueue<T>:** It represents a thread-safe first-in-first-out (FIFO) collection.
5. **ConcurrentDictionary<TKey, TValue>:** It represents a thread-safe collection of key/value pairs that can be accessed by multiple threads concurrently.

In the next article, I am going to discuss the **Non-Generic ArrayList Collection Class in C#** with Examples. Here, in this article, I give you a brief introduction to **Collections in C#**. I hope this article will help you with your needs. I would like to have your feedback. Please post your feedback, question, or comments about this article.

Dot Net Tutorials

About the Author: Pranaya Rout

Pranaya Rout has published more than 3,000 articles in his 11-year career. Pranaya Rout has very good experience with Microsoft Technologies, Including C#, VB, ASP.NET MVC, ASP.NET Web API, EF, EF Core, ADO.NET, LINQ, SQL Server, MYSQL, Oracle, ASP.NET Core, Cloud Computing, Microservices, Design Patterns and still learning new technologies.

Registration Open – Full-Stack .NET with Angular & ASP.NET Core

New Batch Starts: 15th September, 2025

Session Time: 8:30 PM – 10:00 PM IST

Advance your career with our expert-led, hands-on live training program. Get complete course details, the syllabus, and Zoom credentials for demo sessions via the links below.

View Course Details & Get Demo Credentials

Registration Form

Join Telegram Group

Contact: +91 70218 01173 (Call / WhatsApp)

Previous Lesson
Advantages and Disadvantages of Arrays in C#

Next Lesson
ArrayList in C#

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name*

Email*

Website

Post Comment

© Dot Net Tutorials | Website Design by Sunrise Pixel