**HashTable**

Both collections are used to maintain dictionary, where data is stored in the form of key, value pairs.

1. In HashTable keys are taken randomly where in Sorted List key will be in ascending order.
2. A Hashtable is a non-generic collection.
3. Hashtable is defined under System.Collections namespace.
4. In Hashtable, you can store key/value pairs of the same type or of the different type.
5. In Hashtable, there is no need to specify the type of the key and value.
6. In Hashtable, if you try to access a key that doesn’t present in the given Hashtable, then it will give null values.
7. The data retrieval is faster than Hashtable due to no boxing/ unboxing.
8. It doesn’t maintain the order of stored values.

**Dictionary**

1. A Dictionary is a generic collection.
2. Dictionary is defined under System.Collections.Generic namespace.
3. In Dictionary, you can store key/value pairs of same type.
4. The data retrieval is slower than Dictionary due to boxing/ unboxing.
5. In Dictionary, you must specify the type of key and value.
6. In Dictionary, if you try to access a key that doesn’t present in the given Dictionary, then it will give error.
7. It always maintain the order of stored values.

What are Attributes?

➢Attributes concept in .NET is a way to mark or store meta data about the code in assembly.

➢Often it is an instruction meant for the runtime.

➢The Runtime can change its behavior or course of action based on the attribute present.

➢In .NET framework there are many built in attributes

➢For e.g. : Serializable, NonSerlalized, XmlIgnore, WebMethod to name few

➢In .NET an Attribute is a sub class of System.Attribute.

➢Conventionally Attribute class names would end with “Attribute”

word.

➢Attributes can be applied to various code parts in .NET and are

called Targets for an attribute

➢For instance class, fields, methods, constructors can be targets for

an attribute.

➢We can apply multiple attributes to a target.

**Threads**

High on Memory

Run on Single Core

Context Switching

Fire and Forget

**Tasks**

Light Weight

Multi Core Aware

Asynchronous Operations

Can return a value

**Comparison Table Between IEnumerable and IEnumerator**

|  |  |  |
| --- | --- | --- |
| **Parameters of Comparison** | **IEnumerable** | **IEnumerator** |
| Type of Interface | Generic | Non-generic |
| Number of Methods | One | Two |
| Returns | IEnumerator | None |
| Cursor State | Cannot retain | Can retain |
| Product | Enumerated object | For each statement result |
| Way of Writing | Abstract | Concrete |

IComparable and IComparer interface is used when a class has a data member as an array of object of any other class.

For the ordered data types like numbers or strings, comparison can be done easily.

As there can be many different data membersfor the object, comparison of two objects cannot be done directly.

In that situation, that class, whose objects are stored in an array, should implement IComparable and/or IComparer interface.

While implementing methods of IComparable or IComparer, you can choose any one of the fields for comparison, as per business requirement.

The same implemented method will be used for sorting of the array or collection.

**IComparable Interface:**

This interface is used to sort elements and compare the current instance with another object of same type.

Method in IComparable interface is System.IComparable.CompareTo(System.Object)

The CompareTo method returns an int value that shows how two elements are related.

**IComparer Interface:**

This interface is used to sort elements

Method in IComparable interface is System.IComparer.Compare(System.Object,System.Object)

This method compare two objects and returns a value indicating whether one is less than, equal to or greater than other

Returns zero if both are same

Less than zero if first object is less than zero

Greater than zero if first object is greater than zero

**When to use IComparable and Comparer Interface?**

While searching and sorting elements in collection,comparision of elements is needed

The comparisions can be between elements of collection or between an element and specified value

Methods used for comparison are Equal, IComparable.CompareTo and IComparer.Compare

|  |  |  |
| --- | --- | --- |
| **Parameters of Comparison** | **XML** | **XAML** |
| Subset | All XML documents are not equal to XAML. | All XAML documents are XML documents. |
| Type | Markup Language | Declarative Language |
| Presence | Web applications. | Design and control window, other web apps. |
| Also define | Other existing markup languages. | Properties of objects, definition, and the relationship between them. |
| Derivation | Standard Generalized Markup Language | XML |
| Primary Use | Searching webs to automate certain web tasks, creating interactive pages. | Design, edit, create and control Windows and web apps, like Silverlight plug-ins. |
| Alternative | Can replace HTML. | Available for the Windows platform and other platforms as well. |

Sr. No. Panels & Description

1 **Stack Panel**

Stack panel is a simple and useful layout panel in XAML. In stack panel, child elements can be arranged in a single line, either horizontally or vertically, based on the orientation property.

2 **Wrap Panel**

In WrapPanel, child elements are positioned in sequential order, from left to right or from top to bottom based on the orientation property.

3 **Dock Panel**

DockPanel defines an area to arrange child elements relative to each other, either horizontally or vertically. With DockPanel you can easily dock child elements to top, bottom, right, left and center using the Dock property.

4 **Canvas Panel**

Canvas panel is the basic layout panel in which the child elements can be positioned explicitly using coordinates that are relative to the Canvas any side such as left, right, top and bottom.

5 **Grid Panel**

A Grid Panel provides a flexible area which consists of rows and columns. In a Grid, child elements can be arranged in tabular form.

**InnerException**

The InnerException is a property of an exception. When there are series of exceptions, the most current exception can obtain the prior exception in the InnerException property.

Let us say we have an exception inside a try block throwing an ArgumentException and the catch clause catches it and writes it to a file. However, if the file path is not found, FileNotFoundException is thrown. Let's say that the outside try block catches this exception, but how about the actual ArgumentException that was thrown? Is it lost? No, the InnerException property contains the actual exception. This is the reason for the existence of an InnerException property for any exception.

**Code First Approach**

It is very popular approach between MVC programmers. It has full control over the code rather than database activity. Using this approach we can do all the database activity from the code. So, we can say manual changes to database have been lost to use this and everything is depended on the code.

**Database First Approach**

If you have already a designed database and you don’t want to do extra effort then you can go with this approach. If your database is ready then Entity Framework will complete his duty and create POCO entities for you with T4 template.

You can modify the database manually and update model from database. So, we can say, entity framework is able to create your model classes based on tables and columns from relational database.

**Model First Approach**

In this approach, you need to draw your model first and let workflow to decide to generate your database. If you are going to work with small project then it will be best. You cannot make manual change into database because of you don’t have command on database, but you can made the changes into model classes.

An indexer is a special type of property that allows a class or a structure to be accessed like an array for its internal collection. C# allows us to define custom indexers, generic indexers, and also overload indexers.

https://www.c-sharpcorner.com/article/uses-of-int-parse-convert-toint-and-int-tryparse

https://www.keentpoint.com/tutorial/ADO-Dot-Net

**C# Partial Types**

C# provides a concept to write source code in separate files and compile it as a single unit. This feature is called partial types and included in C# 2.0. The partial keyword is used to create partial types.

It allows us to write partial class, interface, struct and method in two or more separate source files. All parts are combined when the application is compiled.

==================================================================================================

Unlike class, structs in C# are value type than reference type.

It is useful if you have data that is not intended to be modified after creation of struct.

Struct doesn't support inheritance. But it can implement interfaces.

Structs are used for lightweight objects such as Color, Rectangle, Point etc.

=====================================================================================================In C#, an enum (or enumeration type) is used to assign constant names to a group of numeric integer values. It makes constant values more readable, for example, WeekDays.Monday is more readable then number 0 when referring to the day in a week.

An enum is defined using the enum keyword, directly inside a namespace, class, or structure.

All the constant names can be declared inside the curly brackets and separated by a comma.

The following defines an enum for the weekdays.

If values are not assigned to enum members, then the compiler will assign integer values to each member starting with zero by default.

The first member of an enum will be 0, and the value of each successive enum member is increased by 1.

==================================================================================================C# - Anonymous Type

In C#, an anonymous type is a type (class) without any name that can contain public read-only properties only. It cannot contain other members, such as fields, methods, events, etc.

You create an anonymous type using the new operator with an object initializer syntax. The implicitly typed variable- var is used to hold the reference of anonymous types.

====================================================================================================Instance variables are non-static variables and are declared in a class but outside any method, constructor or block. As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed. Unlike local variables, we may use access specifiers for instance variables.

====================================================================================================

Static variables are also known as Class variables. If a variable is explicitly declared with the static modifier or if a variable is declared under any static block then these variables are known as static variables.

These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.

Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.

Static variables are created at the start of program execution and destroyed automatically when execution ends.

====================================================================================================

The Reference type variable is such type of variable in C# that holds the reference of memory address instead of value. class, interface, delegate, array are the reference type. When you create an object of the particular class with new keyword, space is created in the managed heap that holds the reference of classes.

====================================================================================================

An indexer is a special type of property that allows a class or a structure to be accessed like an array for its internal collection. C# allows us to define custom indexers, generic indexers, and also overload indexers.

An indexer can be defined the same way as property with this keyword and square brackets [].

======================================================================================================Extension methods, as the name suggests, are additional methods. Extension methods allow you to inject additional methods without modifying, deriving or recompiling the original class, struct or interface. Extension methods can be added to your own custom class, .NET framework classes, or third party classes or interfaces.

===================================================================================================CObject Initializer Syntax

C# 3.0 (.NET 3.5) introduced Object Initializer Syntax, a new way to initialize an object of a class or collection. Object initializers allow you to assign values to the fields or properties at the time of creating an object without invoking a constructor.

====================================================================================================

The Inner Exception in C# is a property of an exception class. When there is a series of exceptions, then the most current exception obtains the previous exception details in the InnerException property. In order words, we can say that the InnerException property returns the original exception that caused the current exception.

====================================================================================================

What is Mutex in C#?

Mutex also works likes a lock i.e. acquired an exclusive lock on a shared resource from concurrent access, but it works across multiple processes. As we already discussed exclusive locking is basically used to ensure that at any given point of time, only one thread can enter into the critical section.

The Mutex class provides the WaitOne() method which we need to call to lock the resource and similarly it provides ReleaseMutex() which is used to unlock the resource. Note that a Mutex can only be released from the same thread which obtained it.

====================================================================================================

What is Semaphore in C#?

The Semaphore in C# is used to limit the number of threads that can have access to a shared resource concurrently. In other words, we can say that Semaphore allows one or more threads to enter into the critical section and execute the task concurrently with thread safety. So, in real-time, we need to use Semaphore when we have a limited number of resources and we want to limit the number of threads that can use it.

=================================================================================================

What is Method Hiding in C#?

When we use the new keyword to hide a base class member, then it is called Method Hiding in C#. We will get a compiler warning if we miss the new keyword. This is also used for re-implementing a parent class method under child class. Re-implementing parent class methods under child classes can be done using two different approaches, such as

Method Overriding

Method Hiding

In the first case, we re-implement the parent class methods under child classes with the permission of parent class because here in parent class the method is declared as virtual giving permission to the child classes for overriding the methods using the override modifier.

In the 2nd approach, we re-implement the method of parent class even if those methods are not declared as virtual that is without parent permission we are re-implementing the methods.

==================================================================================================

C# 7 Features

What is Digit Separator in C#?

In reality, it’s very difficult to read a very large number. To overcome this problem, C# 7 comes with a new feature called digit separators “\_”. Now, it is possible to use one or more Underscore (\_) characters as digit separators in C# to represent a very big number. Sometimes, it is required when we are going to represent a very big number.

====================================================================================================

Why do we need Tuples in C#?

If you want to return more than one value from a method then you need to use Tuples in C#. And in the programming world, it is a very common thing to return multiple values from a method. Tuples are not completely new in C# 7. In .NET Framework 4.0, a set of Tuple classes has been introduced in the System namespace. Tuples in C# 7, provide a better mechanism to return multiple values from a method.

====================================================================================================

What are Local Functions in C#?

The Local Functions in C# are the special kind of inner function or you can say sub-function or function within a function that can be declared and defined by the parent function. These methods or functions are the private methods for their containing type and are only called by their parent method.

==================================================================================================

Func is built-in delegate type.

Func delegate type must return a value.

Func delegate type can have zero to 16 input parameters.

Action is a delegate type defined in the System namespace. An Action type delegate is the same as Func delegate except that the Action delegate doesn't return a value. In other words, an Action delegate can be used with a method that has a void return type.

====================================================================================================

Predicate delegate takes one input parameter and boolean return type.

Anonymous method and Lambda expression can be assigned to the predicate delegate.

==================================================================================================

File And FileInfo Class

The .NET Framework class library contains two similar classes for dealing with files — FileInfo and File.

The File class provides static methods for creating, deleting, and manipulating files, whereas the FileInfo class exposes instance members for files manipulation.

Like the Directory class, the File class only exposes static methods and does not contain any properties.

===================================================================================================

The Directory class is similar to DirectoryInfo class. The key difference between is that Directory exposes static members instead of instance members. The Directory class also exposes only methods — no properties.

StreamReader and StreamWriter classes are used when we dealing with text files.StreamReader is derived from the TextReader class, an abstract class that represents a reader that can read a sequential series of characters.

The StreamWriter class is derived from the abstract TextWriter class and is used for writing characters to a stream.

The BinaryReader class is then used to read the binary data from the FileStream, and the BinaryWriter is used to write the binary data to the file.

===================================================================================================

When to Use abstract class and Interface?

If you anticipate creating multiple versions of your component, create an abstract class.

If the functionality you are creating will be useful across a wide range of disparate objects, use an interface. Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited for providing common functionality to unrelated classes.

If you are designing small, concise bits of functionality, use interfaces. If you are designing large functional units, use an abstract class.

If you want to provide common, implemented functionality among all implementations of your component, use an abstract class. Abstract classes allow you to partially implement your class, whereas interfaces contain no implementation for any members.

https://www.c-sharpcorner.com/article/json-serialization-and-deserialization-in-c-sharp/

Serialization and deserialization in .NET

JSON data is a common format these days when passing data between applications. When building .NET application, JSON data format conversion to .NET objects and vice versa is very common.

What is JSON?

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write and easy for machines to parse and generate. JSON is a text format that is completely language independent.

We can implement JSON Serialization/Deserialization in the following three ways:

* Using JavaScriptSerializer class
* Using DataContractJsonSerializer class
* Using JSON.NET library

JSON supports the following two data structures,

Collection of name/value pairs - This Data Structure is supported by different programming languages.

Ordered list of values - It includes an array, list, vector or sequence, etc.

**Using DataContractJsonSerializer**

DataContractJsonSerializer class helps to serialize and deserialize JSON.

It is present in namespace System.Runtime.Serialization.Json which is available in assembly System.Runtime.Serialization.dll. Using the class we can serialize an object into JSON data and deserialize JSON data into an object.

**Deserialization**

In Deserialization, it does the opposite of Serialization, which means it converts JSON string to a custom .Net object. I