

Advanced .NET

Module 2

String Handling

- In C#, string is an object of **System.String** class that represent sequence of characters.
- We can perform many operations on strings such as concatenation, comparision, getting substring, search, trim, replacement etc.
- In C#, *string* is keyword which is an alias for *System.String* class. That is why string and String are equivalent. We are free to use any naming convention.

Methods

Clone()

It is used to return a reference to this instance of String.

Compare(String, String)

It is used to compares two specified String objects. It returns an integer that indicates their relative position in the sort order.

CompareTo(String)

It is used to compare this instance with a specified String object. It indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified string.

Concat(String, String)

It is used to concatenate two specified instances of String.

Contains(String)

It is used to return a value indicating whether a specified substring occurs within this string.

IndexOf(String)

It is used to report the zero-based index of the first occurrence of the specified string in this instance.

ToString()

It is used to return instance of String.

Trim()


It is used to remove all leading and trailing white-space characters from the current String object.

clone()

- The Clone() method is used to clone a string object. It returns another copy of same data.
- The return type of Clone() method is object.
- It does not take any parameter.

Example

```
public static void Main(string[] args)
{
    string s1 = "Hello ";
    string s2 = (String)s1.Clone();
    Console.WriteLine(s1);
    Console.WriteLine(s2);
}
```



Hello
Hello

Quoted string literals

Quoted string literals start and end with a single or double quote character (") on the same line.

Escape sequence	Character name
\'	Single quote
\"	Double quote
\\	Backslash
\0	Null
\a	Alert
\b	Backspace

Exception Handling

- An exception is a problem that arises during the execution of a program.
- A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.
- Exceptions provide a way to transfer control from one part of a program to another.
- C# exception handling is built upon four keywords: try, catch, finally, and throw.

Exception Handling

- **try** – A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
- **catch** – A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **finally** – The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **throw** – A program throws an exception when a problem shows up. This is done using a throw keyword

Exception Classes

- **System.IO.IOException** : Handles I/O errors.
- **System.IndexOutOfRangeException** : Handles errors generated when a method refers to an array index out of range.
- **System.ArrayTypeMismatchException** : Handles errors generated when type is mismatched with the array type.
- **System.DivideByZeroException** : Handles errors generated from dividing a dividend with zero.

Multithreading

- A thread is a lightweight process, or in other words, a thread is a unit which executes the code under the program.
- So every program has logic and a thread is responsible for executing this logic.
- Every program by default carries one thread to executes the logic of the program and the thread is known as the Main Thread, so every program or application is by default single-threaded model.
- This single-threaded model has a drawback. The single thread runs all the process present in the program in synchronizing manner, means one after another. So, the second process waits until the first process completes its execution, it consumes more time in processing.

Multithreading

- Multi-threading is a process that contains multiple threads within a single process. Here each thread performs different activities.
- The major advantage of multithreading is it works simultaneously, which means multiple tasks execute at the same time.
- And also maximizing the utilization of the CPU because multithreading works on time-sharing concept, which means, each thread takes its own time for execution and does not affect the execution of another thread, this time interval is given by the operating system.

System.Threading Namespace

The System.Threading namespace contains classes and interfaces to provide the facility of multithreaded programming. It also provides classes to synchronize the thread resource. A list of commonly used classes are given below:

- Thread
- Mutex
- Timer
- Monitor
- Semaphore
- ThreadLocal
- ThreadPool

Advantages of Multithreading:

- It executes multiple process simultaneously.
- Maximize the utilization of CPU resources.
- Time sharing between multiple process.

System.Threading.Thread class

The following are the most common instance members of the System.Threading.Thread class:

- **Name**

A property of string type used to get/set the friendly name of the thread instance.

- **Priority**

A property of type System.Threading.ThreadPriority to schedule the priority of threads.

- **IsAlive**

A Boolean property indicating whether the thread is alive or terminated.

- **ThreadState**

A property of type System.Threading.ThreadState, used to get the value containing the state of the thread

System.Threading.Thread class

- **start()**
Starts the execution of the thread.
- **abort()**
Allows the current thread to stop the execution of the thread permanently.
- **suspend()**
Pauses the execution of the thread temporarily.
- **resume()**
Resumes the execution of a suspended thread.
- **join()**
Make the current thread wait for another thread to finish.
- **sleep()**
Temporarily suspend the current execution of the thread for specified milliseconds.

Thread Synchronization

- Synchronization is a technique that allows only one thread to access the resource for the particular time. No other thread can interrupt until the assigned thread finishes its task.
- In multithreading program, threads are allowed to access any resource for the required execution time. Threads share resources and executes asynchronously.
- Accessing shared resources (data) is critical task that sometimes may halt the system. We deal with it by making threads synchronized.
- It is mainly used in case of transactions like deposit, withdraw etc.

Advantage of Thread Synchronization

- Maintain consistency
- No Thread Interference

Lock

- We can use C# **lock keyword** to execute program synchronously.
- It is used to get lock for the current thread, execute the task and then release the lock.
- It ensures that other thread does not interrupt the execution until the execution finish.

FILE I/O

- A file is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a stream.
- The stream is basically the sequence of bytes passing through the communication path.

There are two main streams:

- **Input stream**
- **Output stream**
- The input stream is used for reading data from file (read operation) and the output stream is used for writing into the file (write operation).

I/O Classes

- The System.IO namespace has various classes that are used for performing numerous operations with files, such as creating and deleting files, reading from or writing to a file, closing a file etc.
- The following are the non-abstract classes in the System.IO namespace
 - - **BinaryReader** : Reads primitive data from a binary stream.
 - **BinaryWriter** : Writes primitive data in binary format.
 - **BufferedStream** : A temporary storage for a stream of bytes.
 - **Directory** : Helps in manipulating a directory structure.
 - **DirectoryInfo** : Used for performing operations on directories.
 - **DriveInfo** : Provides information for the drives.
 - **File** : Helps in manipulating files.
 - **StringReader** : Is used for reading from a string buffer.
 - **StringWriter** : Is used for writing into a string buffer.