Basics

21 January 2022 09:06

.NET: It is a software environment for building and executing *runtime managed applications* on multiple platforms. It offers support for

1. **Common Language Runtime** (CLR) - It specifies how data-types are implemented and how code is compiled for .NET applications and handles execution of such compiled (managed) code on top of services offered by its host. It includes support for
   (a) **Common Type System** - A .NET data-type is either a *value type* which provides direct access to the data or a *reference type* which provides access to the data through an indirection. The CLR's type system provides common implementations for *primitive value types* and a *unified object oriented model* for implementing user-defined reference and value types.
   (b) **Virtual Execution System** - A set of related .NET data-types are compiled together into a deployment unit called an *assembly* which contains *meta-data* (machine-readable description) for those types and *intermediate language* (IL) *opcodes* (machine neutral instructions) for their implemented methods. The CLR's execution system loads the assemblies required by the application at runtime and translates the IL opcodes of a method to their equivalent native machine instruction just-in-time of its invocation.
2. **Base Class Library** (BCL) - It is a framework (Microsoft.NETCore.App) of assemblies containing types required by a .NET application for consuming services offered by the following in a portable manner
   (a) **Runtime** which includes support for built-in data types, reflection and interoperation
   (b) **Platform** which includes support for concurrency, file I/O and communication.
3. **C# Programming Language** - It is a high-level programming language (pronounced as See Sharp) designed specifically for implementing applications which target the CLR. It has following important characteristics
   (a) It offers C++ like but more expressive syntax based on the CLR's

type system with opt-in support (unsafe blocks) for pointers.
(b) It is primarily object oriented based on common root single class inheritance model with added support for generic and functional programming.

C# Built-in Types

Type	**Data**
bool	Primitive *true*/*false* option value
char	Primitive character Unicode value
byte/sbyte	Primitive 8-bit unsigned/signed integer value
short/ushort	Primitive 16-bit signed/unsigned integer value
int/uint	Primitive 32-bit signed/unsigned integer value
long/ulong	Primitive 64-bit signed/unsigned integer value
nint/unint	Primitive native-size signed/unsigned integer value
float	Primitive 32-bit single-precision floating-point real value
double	Primitive 64-bit single-precision floating-point real value
decimal	128-bit high-precision fixed-point real value
string	reference to an immutable sequence of characters
object	reference to an instance of any type

struct	**class**
Defines a *non-nullable* type which is implicitly passed by *value*.	Defines a *nullable* type which is always passed by *reference*.
Memory is *automatically* allocated for its instance and *new* operator may be used for its initialization.	Memory is *explicitly* allocated for its instance using *new* operator which also performs its initialization.
Memory is assigned to its locally identified instance on the *stack* (fast) and it is automatically reclaimed after the identifying method of this instance returns.	Memory is assigned for any instance on the *heap* (slow) and it is automatically reclaimed during *garbage collection* that occurs after this instance is no longer reachable from any method.

Instance fields can be accessed using *direct addressing* (fast)	Instance fields can only be accessed through an *indirection* (slow)
Parameter-less constructor is always supported and the implicitly defined one initializes instance fields to their default values.	Parameter-less constructor is supported if it is either defined explicitly or in absence of any explicitly defined constructor in which case the implicitly defined one initializes instance fields to their default values.
Cannot explicitly extend any other type because it always extends System.ValueType which itself extends System.Object.	Can explicitly extend any one other class type otherwise it implicitly extends System.Object.
Implicit conversion to a compatible type requires boxing (copying) of instance data on the heap.	Implicit conversion to a compatible type does not require any copying of instance data.
Suitable for abstraction of complex data which is small and requires high performance data access.	Suitable for abstraction of complex data which is large or requires extensibility through subtyping.