Inheritance

25 January 2022 09:31

Object Identity: Two objects are considered to be *identical* if they *refer* to the same instance in the memory. In .NET whether an object x is identical to object y is indicated by the expression:

```
System.Object.ReferenceEquals(x, y)
```

Object Equality: Two objects are considered to be *equal* if they are instances of same class with matching data in the memory. In .NET whether an object x is equal to an object y is indicated by the expression:

```
x.GetHashCode() == y.GetHashCode() && x.Equals(y)
```

C# Member Access Outside of Defining Type

Access Modifier	**Current Assembly**	**External Assembly**
private (default)	none	none
internal	all	none
protected	derived	derived
internal protected	all	derived
public	all	all

Abstract Class	**Interface**
It is a non-activatable class which can define *instance fields.*	It is a non-activatable reference type which cannot define *instance fields.*
It can define a pure (unimplemented) instance method using the *abstract* modifier.	Its instance method is implicitly pure unless it is defined with a specific implementation.
Members are private by default.	Members are public by default.

Supports an instance constructor which is called by derived classes.	Does not support an instance constructor.
It can extend exactly one other class which may or may not be abstract.	It can extend multiple other interfaces.
A class can inherit from a single abstract class and it must override pure methods of that class otherwise it must be declared abstract.	A class can inherit from multiple interfaces and it must either implement their pure methods or define them as abstract methods in which case this class must be declared abstract.
A struct cannot inherit from an abstract class.	A struct can inherit from multiple interfaces and it must implement their pure methods.
Generally defined for specifying common type of state supported by objects of different inheriting classes.	Generally defined for specifying common type of behavior supported by objects of different inheriting types.

Multiple Inheritance in .NET - The type-system of CLR does not allow a class to inherit from multiple classes because an instance of such a class will require a layout with multiple sub-objects out of which only first one can be referenced in a safe manner and without complicating runtime access to the type of that instance (for casting, reflection etc) which is essential in .NET. Since interface cannot define instance fields it does not require a sub-object within an instance of its inherited class and as such a class can inherit from multiple interfaces.