* Class
* Structures
* Class vs Structures
* Enumerations
* Constants
* Variables
* Readonly
* Methods
* Properties
* Events

# Class

The class is the *basic building block* of any object oriented programming (oops) language. The class is a user defined blueprint for creating objects (instances of the class). This class, which happens to be a blue print, may contain within it constructors, destructors, finalizers, constants, fields, methods, properties, indexers, events etc.

In the runtime, while creating the instances of the classes (also known as objects), you can initialize the member variables, properties etc through constructors and write custom logic for destructors / finalizers.

## Implementation

Fig.1 below shows a real world entity named Customer and its equivalent code representation in C#. The private prefix in front of each of these identifiers indicate that these members will not be accessed beyond this class.

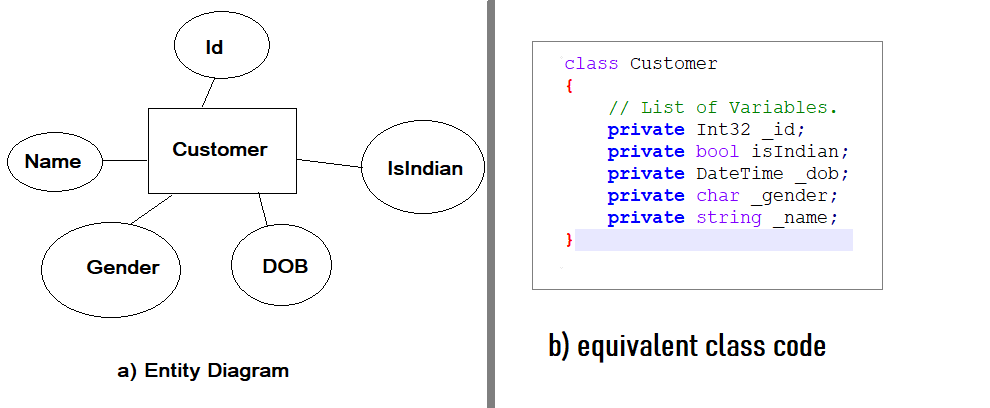


Fig.1. Real World Entity and an equivalent code.

## Characteristics of a class

1. The instances of all the classes are created in the managed heap and a reference placed in the stack. So, instances of classes are reference types.
2. All the variables created in the class should be private and methods be public / protected / others so that it adheres to the principle of encapsulation.
3. Classes are instantiated in the heap using the new keyword.  
    **example**:- Customer ganeshn = new Customer();
4. Instances of all the classes in the managed heap are memory managed by a component named Garbage Collector (GC).
5. In real world applications, a hierarchy of class templates are made using the Inheritance principle.

Download class samples from the below link:

Download Sample Class Files

# Struct

Structures (structs) are again a user defined blueprint, just like a class for creating objects and providing initial values for state - the difference being they are value types and gets created in the stack part of the Memory. Structure is defined using struct keyword. Using struct keyword one can define the structure consisting of different data types in it. A structure can also contain constructors, constants, fields, methods, properties, indexers and events etc.

# Class vs Structures

Classes and Structs both are User defined Data templates. That’s their commonality. Apart from that, the differences are listed in Table.1.

|  |  |  |
| --- | --- | --- |
| **Sl.No** | **Classes** | **Structs** |
| 1 | Classes are of reference types and hence are allocated in the heap. | Structs are of value types and are allocated in the stack. |
| 2 | Class templates are generally used in programs where the programmer has to make a lot of copies based on the template. | Structures are used in places where just one, two (or typically less than 10) instances of the template has to be used. |
| 3 | There is an indirection involved in classes so there is a negligible performance penalty involved. | As there is no indirection involved, structs are performance wise better |
| 4 | Classes can contain Constructors – Default, Parameterized, Copy constructors etc. | Structs cannot have Default constructors but can have parameterized constructors which have to mandatorily initialize all the members of the structs. |
| 5 | Class can have destructors within it. | Structures don’t have the concept of destructors. |
| 6 | A class can inherit from another class. | A Structure is not allowed to inherit from another structure or class. |
| 7 | The data member of a class can be protected. | The data member of structure can’t be protected. |
| 8 | Function member of the class can be virtual or abstract. | Function member of the structure cannot be virtual or abstract. |
| 9 | Two variable of class can contain the reference of the same object and any operation on one variable can affect another variable. | Each variable in struct contains its own copy of data (except in ref and out parameter variable) and any operation on one variable cannot effect another variable. |

**Table.1. Classes vs Structs.**

# Enumerations

Enumeration (enum) is used to declare a list of named integer constants and is a value data type. It can be defined using the enum keyword directly inside a namespace, class, or structure. The enum is used to give a name to each constant so that the constant integer can be referred using its name.

## **Example**

enum WeekDays

{

Monday = 0,

Tuesday =1,

Wednesday = 2,

Thursday = 3,

Friday = 4,

Saturday =5,

Sunday = 6

}

Code .1 Enumerations.

Download Enum Sample Files

# Constants

Constants are immutable values which are known at compile time and do not change for the life of the program. Constants are declared with the const modifier. Only the C# built-in types (excluding System.Object) may be declared as const.

Constants are used where you want a fixed parameter value to be used in multiple places. Change the value of the constant in the declaration part and the value changes in all the referred places.

Important: User-defined types, including classes, structs, and arrays, cannot be const. They can be only read-only.

For more information, Visit

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constants>

Download Constants Demo files

# Variables

A variable or scalar is a storage address (identified by a memory address) paired with an associated symbolic name, which contains some known or unknown quantity of information referred to as a value. The variable name is the usual way to reference the stored value, in addition to referring to the variable itself, depending on the context. This separation of name and content allows the name to be used independently of the exact information it represents. The identifier in computer source code can be bound to a value during run time, and the value of the variable may thus change during the course of program execution.

(Source: **wikipedia.com**)

Download variables demo files

# Readonly

In c#, readonly is a keyword which is useful to define read-only fields in our applications. The read-only field values need to be initialized either at the declaration or in a constructor of the same class unlike constant keyword in c#. If we use readonly keyword with fields, then those field values will be evaluated at the runtime.

To define read-only fields in c#, we need to use readonly keyword during the declaration of fields in our application and we can use readonly modifier with the numbers, boolean values, strings or with null references.

If we use readonly keyword to define the read-only field, then that field value cannot be changed once the constructor execution has finished so we should not use readonly keyword with the fields whose value will be changed at any time. It’s mandatory to initialize read-only field values either at the declaration or in a constructor otherwise we will get compile-time errors in our c# application.

(**Source**: tutlane.com - <https://www.tutlane.com/tutorial/csharp/csharp-readonly-property>)

Download ReadOnly demo files

# Methods

A method is a code block that contains a series of statements. A program causes the statements to be executed by calling the method and specifying any required method arguments. In C#, every executed instruction is performed in the context of a method. The Main() is the entry point for every C# application and it's called by the common language runtime (CLR) when the program is started.

Download Methods demo files

# Properties

A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field. Properties can be used as if they are public data members, but they are actually special methods called accessor s. This enables data to be accessed easily and still helps promote the safety and flexibility of methods.

1. Properties enable a class to expose a public way of getting and setting values, while hiding implementation or verification code.
2. A get property accessor is used to return the property value, and a set property accessor is used to assign a new value. These accessor s can have different access levels.
3. The value keyword is used to define the value being assigned by the set accessor
4. Properties can be read-write (they have both a get and a set accessor), read-only (they have a get accessor but no set accessor), or write-only (they have a set accessor, but no get accessor).

Important: Write-only properties are rare and are most commonly used to restrict access to sensitive data.

5) Simple properties that require no custom accessor code can be implemented either as expression body definitions or as auto-implemented properties.

(Source: csharpcorner.com)

Download Properties demo files

# Events

An event is an action or occurrence recognized by the operating system through the process that has invoked that event. This event (like *KeyUp*, *KeyDown*, *Keypress*, *MouseButtonClick*, *and MouseHover* etc) are associated with objects and any such interesting event is notified to the operating system through the process.

Events can be generated / triggered by the system, by the user, or in other ways. Typically, events are handled [synchronously](https://en.wikipedia.org/wiki/Synchronisation) with the [program flow](https://en.wikipedia.org/wiki/Control_flow); that is, the process may have one or more dedicated places where events are handled (event handler), frequently an [event loop](https://en.wikipedia.org/wiki/Event_loop).

A source of events includes the user, who may interact with the process through the computer's [peripherals](https://en.wikipedia.org/wiki/Peripheral) - for example, by typing on the [keyboard](https://en.wikipedia.org/wiki/Keyboard_(computing)). Another source is a [hardware](https://en.wikipedia.org/wiki/Computer_hardware) device such as a [timer](https://en.wikipedia.org/wiki/Timer#Software_timers). Software can also trigger its own set of events into the event loop, e.g. to communicate the completion of a task. Process that changes its behavior in response to events is said to be event-driven, often with the goal of being [interactive](https://en.wikipedia.org/wiki/Interactivity).

(**Reference**: ***Wikipedia***)

## Event Handlers

An event handler is a method that will handle the event – i.e. if the event occurs, what actions should happen is written in the event handler. Fig.2 shows the event and the event handler.

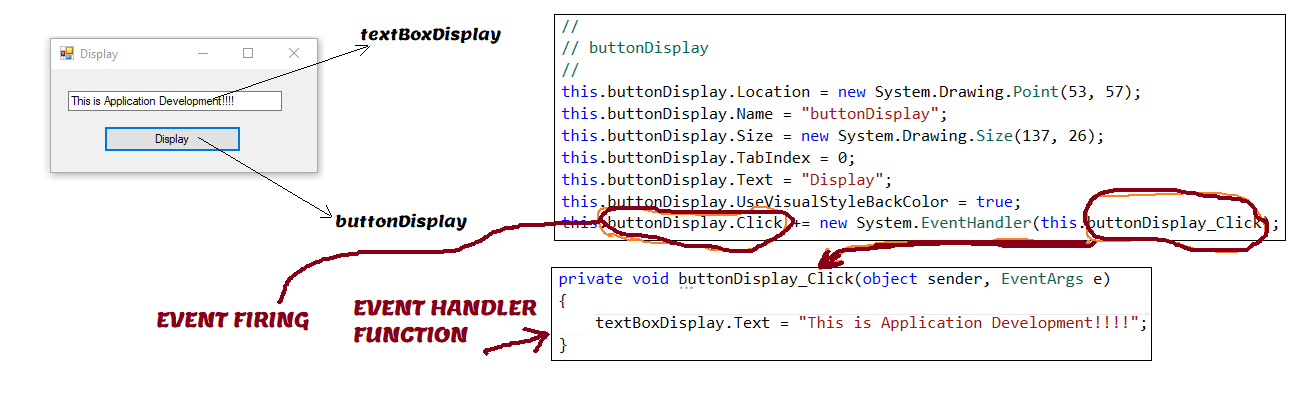


Fig.2 Events and Event handler

## Synchronous Events

Events, whose event handlers are immediately called after the event occurs are known as Synchronous events. Example – DocumentNew event - It gets called as soon as the user creates a new document.

## Asynchronous Events

Events, whose event handlers are called after a certain time lapse post the event occurrence are known as Asynchronous events. Example – A web service method getting called and the data submitted to the calling place.

Download Events Files