Phase-3

Name : Kotadiya Amisha K.

College : Government Engineering College, Rajkot

Branch : Computer Engineering

Understanding

Module-14

- **Understanding Classes**
    - Static, sealed, abstract

1) static, sealed, abstract.

Static

In c#, a static class can be created by using static modifier and the static class can contain only static members.

Generally, the static class is same as the non-static class, but the only difference is the static class cannot be instantiated. Suppose if we apply static modifier to a class, we don't require to use the new keyword to create a class type variable.

Another difference is the static class will contain only static members, but the non-static class can contain both static and non-static members.

when accessing static class members and functions directly with the class name because we cannot instantiate the static class.

C# Static Class Features

- The static class in c# will contain only static members.
- In c#, the static classes cannot be instantiated.
- C# static classes are sealed, so they cannot be inherited.

Sealed

In c#, sealed is a keyword used to stop inheriting the particular class from other classes. We can also prevent overriding the particular properties or methods based on our requirements.

Generally, when we create a particular class we can inherit all the properties and methods in any class. If you want to restrict access to a defined class and its members, then by using a sealed keyword, we can prevent other classes from inheriting the defined class.

we defined various classes with a sealed keyword to ensure that the defined classes are not inheritable to any class. In c#, we can use a sealed keyword before or after the access modifier to define sealed classes.

C# Sealed Keyword Features

- In c#, to apply a sealed keyword on a method or property, it must always use with override.
- In c#, we should not use an abstract modifier with a sealed class because an abstract class must be inherited by a class that provides an implementation of the abstract methods or properties.
- In c#, the local variables cannot be sealed.
- In c#, structs are implicitly sealed; they cannot be inherited.

Abstract

In c#, we can use abstract modifiers with classes, methods, properties, events, and indexers based on our requirements. The members we defined as abstract or included in an abstract class must be implemented by classes derived from an abstract class.

In c#, abstract class is a class that is declared with a abstract modifier. If we define a class with abstract modifier, then that class is intended only to be used as a base class for other classes.

The abstract class cannot instantiate, and it can contain both abstract and non-abstract members. The class that is derived from the abstract class must implement all the inherited abstract methods and accessors.

If we define a method with abstract modifier, then that method implementation must be done in a derived class.

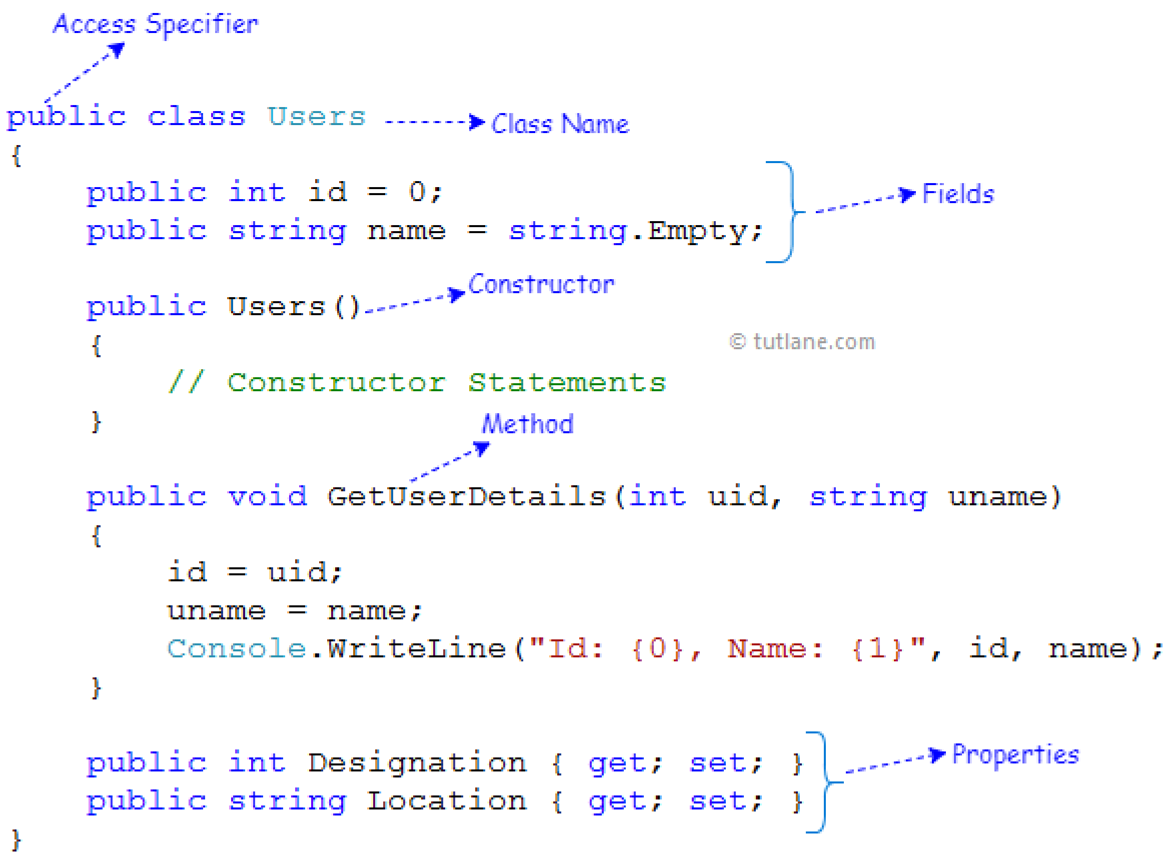
C# Abstract Class Features

- In c#, abstract classes cannot be instantiated.
- The abstract classes can contain both abstract and non-abstract methods and accessors.
- In c#, we should not use a sealed keyword with abstract class because the sealed keyword will make a class not inheritable, but abstract modifier requires a class to be inherited.
- A class that is derived from an abstract class must include all the implementations of inherited abstract methods and accessors.

Module-15

- **Depth in Classes**
  - Object, Properties, Methods, Events etc.

1) Object, Properties, Methods, Events etc.



```
public class Users
{
    public int id = 0;
    public string name = string.Empty;

    public Users()
    {
        // Constructor Statements
    }

    public void GetUserDetails(int uid, string uname)
    {
        id = uid;
        uname = name;
        Console.WriteLine("Id: {0}, Name: {1}", id, name);
    }

    public int Designation { get; set; }
    public string Location { get; set; }
}
```

Object

- In C#, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.
- In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.
- Object is a runtime entity, it is created at runtime.
- Object is an instance of a class. All the members of the class can be accessed through object.

Let's see an example to create object using new keyword.

Student s1 = new Student();//creating an object of Student

Properties

In c#, Property is an extension of the class variable. It provides a mechanism to read, write, or change the class variable's value without affecting the external way of accessing it in our applications.

In c#, properties can contain one or two code blocks called accessors, and those are called a get accessor and set accessor. By using get and set accessors, we can change the internal implementation of class variables and expose it without affecting the external way of accessing it based on our requirements.

Generally, in object-oriented programming languages like c# you need to define fields as private and then use properties to access their values in a public way with get and set accessors.

Following is the syntax of defining a property with get and set accessor in c# programming language.

```
<access_modifier> <return_type> <property_name>
{
    get
    {
       // return property value
    }
    set
    {
      // set a new value
    }
}
```

If you observe the above syntax, we used an access modifier and return type to define a property along with get and set accessors to make required modifications to the class variables based on our requirements.

In c#, the properties are categorized into three types, those are.

Type	Description
Read-Write	A property that contains a both get and set accessors, then we will call it a read-write property.
Read-Only property.	A property that contains only get accessor, then we will call it a read-only
Write-Only property.	A property that contains only set accessor, then we will call it a write-only

Methods

- A method is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as functions.
- Why use methods? To reuse code: define the code once, and use it many times.
- To call (execute) a method, write the method's name followed by two parentheses () and a semicolon;

Events

The following are the important points that we need to remember about events in the c# programming language.

- In c#, events are used to enable a class or object to notify other classes or objects about the action that is going to happen.
- To declare an event, we need to use event keyword with delegate type.
- Before raising an event, we need to check whether an event is subscribed or not.
- By using += operator, we can subscribe to an event, and by using -= operator, we can unsubscribe from an event.
- To raise an event, we need to invoke the event delegate.
- To respond to an event, we can define an event handler method in the event receiver, and the handler method must have the same signature of delegate in the event.
- To raise events, there must be subscribers; otherwise, they won’t be raised.
- In c#, an event can have multiple subscribers, and a subscriber can handle multiple events from multiple publishers.
- In case an event has multiple subscribers, then event handlers are invoked synchronously when an event is raised.

- In c#, the **publisher** determines when an event is raised, and the **subscriber** determines what action is taken in response to the event.
- In .NET Framework, events are based on **EventHandler** delegate and an **EventArgs** base class.

Module-16

- **Scope & Accessibility Modifiers**
  - Public, private, protected

1) Public, private, protected

Public

In c#, the public modifier is used to specify that access is not restricted, so the defined type or member can be accessed by any other code in the current assembly or another assembly that references it.

Private

In c#, the private modifier is used to specify that access is limited to the containing type, so the defined type or member can only be accessed by the code in the same class or structure.

Protected

In c#, the protected modifier is used to specify that access is limited to the containing type or types derived from the containing class, so the type or member can only be accessed by code in the same class or in a derived class.

Internal

In c#, the internal modifier is used to specify that access is limited to the current assembly. The type or member can be accessed by any code in the same assembly but not from another assembly.

Access Modifier	Description
public	It is used to specifies that access is not restricted.
private	It is used to specifies that access is limited to the containing type.
protected	It is used to specifies that access is limited to the containing type or types derived from the containing class.
internal	It is used to specifies that access is limited to the current assembly.
protected internal	It specifies that access is limited to the current assembly or types derived from the containing class.
private protected	It is used to specifies that access is limited to the containing class or types derived from the containing class within the current assembly.

Module-17

- **Namespace & .Net Library**

What is Namespace?

- namespaces are used to logically arrange classes, structs, interfaces, enums and delegates.
- The namespaces can be nested. That means one namespace can contain other namespaces also.
- The .NET framework already contains number of standard namespaces like System, System.Net, System.IO etc.
- In addition to these standard namespaces the user can define their own namespaces.

Declaring a Namespace

Syntax:

```
namespace <namespace_name>

{

// Classes and/or structs and/or enums etc.

}
```

- Note: namespace keyword is used to define namespace
- It is not possible to use any access specifiers like private, public etc with a namespace declaration.
- The namespaces in C# are implicitly have public access and this is not modifiable.
- Default it provide internal access.

Creating Aliases

Developers can create Aliases of the namespace.

Example:

```
using con = System.Console; // Create an alias

class MyClient

{

    public static void Main()

    {

        con.WriteLine("namespace demo");

    }

}
```

Standard Namespaces in .NET

- System: Contain classes that implement basic functionalities like mathematical operations, data conversions etc.
- System.IO: Contains classes used for file I/O operations.
- System.Net: Contains class wrappers around underlying network protocols.
- System.Collections: Contains classes that implement collections of objects such as lists, hashtable etc.
- System.Data: Contains classes that make up ADO.NET data access architecture.
- System,Drawing: Contains classes that implement GUI functionalities.
- System.Threading: Contains classes that are used for multithreading programming.
- System.Web: Classes that implement HTTP protocol to access web pages
- System.Xml: Classes that are used for processing XML data.

Module-18

- Creating and adding ref. to assemblies

- An Assembly is a basic building block of .Net Framework applications.
- It is basically a compiled code that can be executed by the CLR.
- An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality.
- An Assembly can be a DLL or exe depending upon the project that we choose.

Assemblies are basically the following two types:

1. Private Assembly

2. Shared Assembly

Private Assembly

- It is an assembly that is being used by a single application only.
- Suppose we have a project in which we refer to a DLL so when we build that project that DLL will be copied to the bin folder of our project.
- That DLL becomes a private assembly within our project. Generally, the DLLs that are meant for a specific project are private assemblies.

Shared Assembly

- Assemblies that can be used in more than one project are known to be a shared assembly.
- Shared assemblies are generally installed in the GAC. Assemblies that are installed in the
- GAC are made available to all the .Net applications on that machine

Module-19

- **Working with collections**

C# collection types are designed to store, manage and manipulate similar data more efficiently.

- Data manipulation includes adding, removing, finding, and inserting data in the collection.
- Adding and inserting items to a collection
- Removing items from a collection
- Finding, sorting, searching items
- Replacing items
- Copy and clone collections and items
- Capacity and Count properties to find the capacity of the collection and number of items in the collection

NET supports two types of collections.

1. Generic Collection

2. Non-Generic Collection

Generic collections with work generic data type.

In non-generic collections, each element can represent a value of a different type. The collection size is not fixed. Items from the collection can be added or removed at runtime.

Non-generic:

- ArrayList
- HashTable
- SortedList
- Stack
- Queue

Generic:

- List
- Dictionary
- SortedList
- Stack
- Queue