1.What is Event ? Explain its properties , advantages and example.

An event is a mechanism that enables a class or object to notify other classes or objects when something significant happens. Events are a part of the C# language's support for the observer design pattern, where an object, known as the subject or publisher, maintains a list of dependents, known as observers or subscribers, and notifies them of state changes.

Example of Event is :

```
using System;

public class MyEventArgs : EventArgs
{
    public string Message { get; }

    public MyEventArgs(string message)
    {
        Message = message;
    }
}

public class Publisher
{

    public event EventHandler<MyEventArgs> MyEvent;

    public void RaiseEvent(string message)
    {

        if (MyEvent != null)
        {
            MyEventArgs args = new MyEventArgs(message);

            MyEvent(this, args);
        }
    }
}

public class Subscriber
```

```
{
    public void HandleEvent(object sender, MyEventArgs e)
    {
        Console.WriteLine($"Event handled by Subscriber:
{e.Message}");
    }
}

class Program
{
    static void Main()
    {

        Publisher publisher = new Publisher();
        Subscriber subscriber = new Subscriber();


        publisher.MyEvent += subscriber.HandleEvent;


        publisher.RaiseEvent("Hello from the event!");


        publisher.MyEvent -= subscriber.HandleEvent;


        publisher.RaiseEvent("Event with no subscribers.");

        Console.ReadLine();
    }
}
```

Properties of Event :

- Event Declaration:

  Events are declared using the event keyword followed by the delegate type that defines the method signature for event handlers.

- Event Handler Type:

The delegate type used to define the event handler methods is known as the event handler type. It specifies the method signature that event subscribers must adhere to.

- Event Access Modifiers:

  Events can have access modifiers, such as public, private, protected, or internal, indicating their visibility.

- Event Invocation Pattern:

  The ?. null-conditional operator is commonly used when invoking events to avoid null reference exceptions if there are no subscribers. This operator ensures that the event is invoked only if there are subscribers.

- Custom Event Arguments:

  Events can use custom event argument classes derived from EventArgs to pass additional information to event subscribers.

2. What is File handling?

File handling refers to the ability to read from and write to files using various classes and methods provided by the .NET Framework. File handling is crucial for tasks such as reading configuration files, logging data, or working with any kind of persistent storage.

The System.IO namespace provides classes and methods for working with files and directories. Here are some of the key classes for file handling:

File Class:

The File class provides static methods for creating, copying, deleting, moving, and opening files. It also includes methods for reading and writing text or binary data.

FileStream Class:

The FileStream class allows for reading from or writing to a file as a stream of bytes. This class is useful for more advanced scenarios where direct control over the stream is required.

StreamReader and StreamWriter Classes:

These classes provide a convenient way to read and write text data from and to files, respectively.

Example of file handling is:

```
using System;

using System.IO;

class Program

{

   static void Main()

   {

     ReadFromFile("input.txt");

     WriteToFile("output.txt", "Hello, File Handling!");

    AppendToFile("output.txt", "\nAppending more content.");

    ReadFromFile("output.txt");

   }

   static void ReadFromFile(string filePath)

  {

    string content = File.ReadAllText(filePath);

   Console.WriteLine($"Content of {filePath}:\n{content}\n");

   }

   static void WriteToFile(string filePath, string content)
```

```
    {
        File.WriteAllText(filePath, content);
       Console.WriteLine($"Content written to {filePath}\n");
    }
     static void AppendToFile(string filePath, string content)
    {
    File.AppendAllText(filePath, content);
    Console.WriteLine($"Content appended to {filePath}\n");
    }
}
```

Advantages of file handling:

Data Import/Export:

File handling facilitates the import and export of data. Applications can read data from files in various formats (e.g., CSV, XML) and write data to files for sharing or backup purposes.

Error Handling and Reporting:

Log files can be used to store information about errors and exceptions, aiding developers in diagnosing and fixing issues. This is especially valuable for applications running in production environments.

Serialization and Deserialization:

File handling supports serialization, allowing objects to be converted into a format that can be stored in a file. This is useful for saving and loading complex data structures.

3. What is Assembly ?

An assembly is a compiled code library that contains Intermediate Language (IL) code and metadata needed for the execution of a program. Assemblies are the fundamental units of deployment and versioning in .NET applications. They can take the form of executable (.exe) files, which are applications, or dynamic link libraries (.dll), which are code libraries.

Properties of Assembly :

Name:

An assembly has a name that includes the assembly's simple name, version number, culture, and public key token (if the assembly is strong-named).

Versioning:

Assemblies support versioning to manage changes over time. The version number is typically specified in the format Major.Minor.Build.Revision.

Strong Naming:

A strong-named assembly is signed with a cryptographic key pair, providing a unique identity and ensuring the integrity of the assembly. This is important for security and versioning.

Manifest:

An assembly includes a manifest, which is a block of metadata that contains information about the assembly, including the names and versions of all the assembly's constituent files.

Type Metadata:

Assemblies contain metadata that describes the types (classes, interfaces, etc.) defined in the assembly, including information about their members, methods, properties, etc.

Example of Assembly are:

- MainAssembly (Executable):

```
using System;

namespace MainAssembly
{
    class Program
    {
        static void Main()
        {
            Console.WriteLine("Hello from Main Assembly!");


            SubAssembly.UtilityClass.DisplayMessage();


            Console.WriteLine($"MainAssembly Version:
{typeof(Program).Assembly.GetName().Version}");
        }
    }
}
```

- SubAssembly (Class Library):

```
using System;

namespace SubAssembly
{
    public static class UtilityClass
    {
        public static void DisplayMessage()
        {
            Console.WriteLine("Hello from Sub Assembly!");
        }
    }
}
```

- AssemblyInfo.cs for SubAssembly (Assembly-level Attributes):

```
using System.Reflection;
using System.Runtime.InteropServices;

[assembly: AssemblyTitle("SubAssembly")]
[assembly: AssemblyDescription("A sample class library
assembly")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Your Company")]
[assembly: AssemblyProduct("SubAssembly")]
[assembly: AssemblyCopyright("Copyright © 2023")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

[assembly: ComVisible(false)]

[assembly: Guid("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")]

[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

4.What is Web Form ?

A Web Form refers to a type of web application page that is created using the ASP.NET Web Forms framework. ASP.NET Web Forms is a part of the ASP.NET web application development stack provided by Microsoft. It is designed to simplify the development of dynamic, data-driven web applications by providing a set of server-side controls and an event-driven programming model.

Some features are:

- ASP.NET Web Forms Framework:

  ASP.NET Web Forms is a framework for building web applications. It abstracts the complexities of web development by offering a component-based programming model similar to traditional desktop applications.

- Server-Side Controls:

Web Forms use server-side controls, which are ASP.NET components that encapsulate common HTML elements and provide additional functionality. Examples include TextBox, Button, GridView, and DropDownList.

- Event-Driven Programming Model:

  Web Forms follow an event-driven programming model. Server-side controls raise events, and developers can write event handler methods to respond to these events. This model makes it easy to manage user interactions.

- ViewState:

  ViewState is a mechanism in Web Forms that allows the state of server-side controls to be persisted across postbacks. It helps in maintaining the state of controls between requests, providing a stateful experience similar to desktop applications.

- Page Life Cycle:

  Web Forms have a defined life cycle that includes events like Page_Init, Page_Load, and Page_PreRender. Developers can write code in these events to perform initialization, handle user input, and prepare the page for rendering.

Example is :

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="WebForm1.aspx.cs"
Inherits="WebApplication1.WebForm1" %>



<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

   <title>Simple Web Form Example</title>

</head>

<body>
```

```
    <form id="form1" runat="server">
        <div>
            <h2>Simple Web Form</h2>
            <asp:Label ID="lblMessage" runat="server"
Text=""></asp:Label>
            <br />
            <asp:TextBox ID="txtName"
runat="server"></asp:TextBox>
            <br />
            <asp:Button ID="btnGreet" runat="server" Text="Greet"
OnClick="btnGreet_Click" />
        </div>
    </form>
</body>
</html>
using System;


namespace WebApplication1
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {


        }
```

```
        protected void btnGreet_Click(object sender, EventArgs e)
        {
            string userName = txtName.Text.Trim();


            if (!string.IsNullOrEmpty(userName))
            {
                lblMessage.Text = $"Hello, {userName}! Welcome to the
Web Form.";
            }
            else
            {
                lblMessage.Text = "Please enter your name.";
            }
        }
    }
}
```

5.Explain Web Services ?

a web service is a software component designed to communicate over the web using standard protocols. It provides a way for

applications to exchange information and functionality over the internet or an intranet. There are different types of web services, but one common approach is to create web services using the ASP.NET framework.

- ASP.NET Web Services:

  ASP.NET provides a technology for building web services called ASP.NET Web Services or ASMX services. These services allow communication using protocols like SOAP (Simple Object Access Protocol).

- Web API and WCF:

  While ASMX services were popular in earlier versions of ASP.NET, newer technologies like ASP.NET Web API and Windows Communication Foundation (WCF) are often used for building web services in modern ASP.NET applications. ASP.NET Web API is particularly well-suited for building RESTful services.

- SOAP and REST:

  Web services can use different protocols, and two common ones are SOAP and REST. SOAP is a protocol for exchanging structured information in web services, while REST (Representational State Transfer) is an architectural style that uses standard HTTP methods.

- Consuming Web Services:

  Clients can consume web services using various tools and libraries, such as HttpClient in C# for RESTful services or by adding a service reference for SOAP services.

- Hosting and Consuming:

  Web services need to be hosted on a server to be accessible. ASP.NET web services can be hosted in IIS, and WCF services can be hosted in various environments. Clients can

then consume these services by making HTTP requests and receiving responses.