MASTER OF COMPUTER APPLICATION

Department of Computer Engineering & Applications

Session- 2025 – 26

Subject Name & Code:	.NET Framework using C# (MCAE0402)
Course:	MCA
Semester:	3rd
Section:	A
Class Roll No:	21
University Roll No:	2484200082

Submitted To	Submitted By
Dr. Sachendra Singh Chauhan	Janavi Agrawal
(Assistant Professor)	

Assignment 2 - .NET Framework using C#

1. Explain the key differences between Windows services and Web services in C#.

Windows Services are long-running background applications that start automatically when the system boots. They don’t require user interaction and are mainly used for system-level tasks such as logging, file monitoring, or scheduled tasks.

Web Services, on the other hand, are network-based services that allow communication between applications over the Internet using standard protocols like HTTP and SOAP. They are mainly used to share data and functionality between distributed systems.

Differences:

Feature	Windows Service	Web Service
Execution	Runs in background on Windows OS	Hosted on a web server
User Interface	No UI	Can be accessed via browser or client
Communication	Local machine	Over HTTP/SOAP
Usage	System-level automation	Application integration

2. Describe the Windows service and explain the purpose of OnStart and OnStop methods.

A Windows Service is a special type of application that runs continuously in the background. It is managed by the Service Control Manager (SCM) and can start automatically when the computer boots up.

- OnStart(): Executed when the service starts. It initializes resources and starts the main operation.
- OnStop(): Executed when the service stops. It releases resources and performs cleanup.

Example: If a service logs data to a file every hour, OnStart() starts the timer, and OnStop() stops it and closes the file.

3. Create a basic Windows service that logs the current date and time to a text file. Include the steps to install and start the service.

Steps:

1. Open Visual Studio → Create new Windows Service Project.
2. In Service1.cs, write:

```
protected override void OnStart(string[] args)
{
    File.AppendAllText(@"C:\log.txt", "Service started: " + DateTime.Now + "\n");
    timer = new Timer(WriteTime, null, 0, 60000);
}
private void WriteTime(object state)
{
    File.AppendAllText(@"C:\log.txt", "Time: " + DateTime.Now + "\n");
}
protected override void OnStop()
{
    File.AppendAllText(@"C:\log.txt", "Service stopped: " + DateTime.Now + "\n");
}
```

3. Build project → Run “installutil MyService.exe”.
4. Open Services.msc → Start the service.

4. Design a website to consume a SOAP-based Web service and display the result in a TextBox.

Example: Currency Conversion

Steps:
1. Create ASP.NET Web Application.
2. Add Service Reference → Enter SOAP service URL.
3. Add TextBox and Button in Default.aspx.
4. Write this code in Default.aspx.cs:

```
protected void btnConvert_Click(object sender, EventArgs e)
{
    CurrencyService.ServiceSoapClient client = new
CurrencyService.ServiceSoapClient();
    double result = client.ConvertUSDToINR(Convert.ToDouble(txtUSD.Text));
    txtINR.Text = result.ToString("F2");
}
```

5. What are attributes in C#? Explain their purpose and provide an example of a custom attribute.

Attributes in C# provide metadata that can be retrieved at runtime using reflection. They describe additional information about program elements.

Example:

```
[AttributeUsage(AttributeTargets.Class)]
public class AuthorAttribute : Attribute
{
    public string Name { get; set; }
    public double Version { get; set; }
    public AuthorAttribute(string name, double version)
    {
        Name = name; Version = version;
    }
}
[Author("Jaya", 1.0)]
public class SampleClass { }
```

6. Explain the use of the Obsolete attribute in C#. What is its impact during compilation?

The Obsolete attribute marks a program element as outdated.

Example:

```
[Obsolete("Use NewMethod instead", true)]
```

If true → Compile-time error.
If false → Compile-time warning.

Impact: Helps developers migrate from old APIs to newer ones safely.

7. Write a program in C# to create a custom attribute Author with properties like Name and Version.

```
[AttributeUsage(AttributeTargets.Class)]
public class AuthorAttribute : Attribute
{
    public string Name { get; set; }
    public double Version { get; set; }

    public AuthorAttribute(string name, double version)
    {
        Name = name; Version = version;
    }
}
```

```
[Author("Jaya Sharma", 1.2)]
public class Demo
{
    public void Display()
    {
        Console.WriteLine("Author Attribute Example");
    }
}
```

8. What is an assembly in C#? Explain the difference between private and shared assemblies.

An assembly is the compiled output of a .NET application (.exe or .dll).

Private Assembly: Used by one application, stored locally.
Shared Assembly: Used by multiple applications, stored in the Global Assembly Cache (GAC).

Differences:

Feature	Private Assembly	Shared Assembly
Scope	Single Application	Multiple Applications
Location	App Folder	GAC
Naming	Normal	Strong name required

9. Describe the structure of an assembly. Highlight the role of the manifest and metadata.

Structure:
1. Manifest – Contains assembly identity and references.
2. Metadata – Describes types and members.
3. MSIL Code – Intermediate code.
4. Resources – Images, icons, etc.

Role:
- Manifest: Helps .NET locate and load correct version.
- Metadata: Enables reflection and type safety.

10. Explain the steps to create a shared assembly and register it in the Global Assembly Cache (GAC).

Steps:
1. Create Class Library project.
2. Generate Strong Name: sn -k keypair.snk
3. Add in AssemblyInfo.cs: [assembly: AssemblyKeyFile("keypair.snk")]

4. Build project → .dll file created.
5. Register in GAC: gacutil -i MyLibrary.dll
6. Verify: gacutil -l MyLibrary

11. Create a class library in C# and compile it as a DLL. Demonstrate its use in a console application.

MathLibrary.cs:

```
namespace MathLibrary
{
    public class Calculator
    {
        public int Add(int a, int b)
        {
            return a + b;
        }
    }
}
```

Console App:

```
using MathLibrary;
class Program
{
    static void Main()
    {
        Calculator calc = new Calculator();
        Console.WriteLine("Sum: " + calc.Add(10, 20));
    }
}
```

12. How would you use ADO.NET to fetch employee records from SQL Server and display them in a DataGridView?

```
SqlConnection con = new SqlConnection("connection string");
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Employees", con);
DataTable dt = new DataTable();
da.Fill(dt);
dataGridView1.DataSource = dt;
```

13. How to enable offline functionality using DataSet and DataTable in ADO.NET?

DataSet stores data in memory and can be used offline.

```
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Employees", con);
SqlCommandBuilder cb = new SqlCommandBuilder(da);
```

```
DataSet ds = new DataSet();
da.Fill(ds, "Employees");

ds.Tables["Employees"].Rows[0]["Name"] = "Jaya";
da.Update(ds, "Employees");
```

14. How would you implement parameterized queries in ADO.NET to prevent SQL Injection?

```
SqlCommand cmd = new SqlCommand("SELECT * FROM Employees WHERE
ID=@ID", con);
cmd.Parameters.AddWithValue("@ID", txtID.Text);
SqlDataAdapter da = new SqlDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
dataGridView1.DataSource = dt;
```

15. Inventory Management System — Using DataSet and DataReader

1. Use DataSet to display and edit product data.

```
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Products", con);
DataSet ds = new DataSet();
da.Fill(ds, "Products");
dataGridView1.DataSource = ds.Tables["Products"];
```

2. Use DataReader for summary data.

```
SqlCommand cmd = new SqlCommand("SELECT COUNT(*), SUM(Price) FROM
Products", con);
SqlDataReader dr = cmd.ExecuteReader();
if (dr.Read())
    Console.WriteLine("Count: " + dr[0] + " Total Value: " + dr[1]);
```