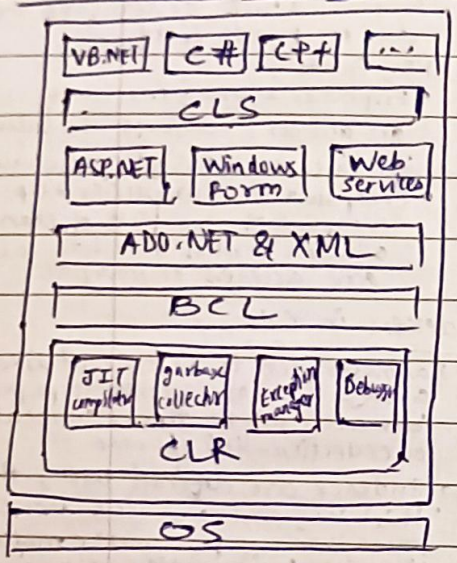
.NET

Architecture of .NET



CLR → execution engine
- → handle running app", memory mngt, garbage collection
- → provide runtime env. to handle execution of .NET
- → convert program into native code.

CLS → set of rules and guidline that .NET language follows to ensure interoperability
- → CLS compliant feature allows a library written in C# to be used by VB.NET application without compatibility issues.

CTS → define how types are declared, used and managed in .NET runtime
- → provide a framework for type safety and interoperability.

FCL → is a broader collection of libraries that include BCL and additional libraries for various application model, networking, data access, XML processing and more.

BCL → is a part of FCL
- → comprehensive collection of reuseable classes, interface & API that provide core functionalities

MSIL

- → CPU independent intermediate language that is generated by .NET compiler
- → serve as a bridge b/t high level code and machine code
- → How it works?
  - when you compile .NET application the high level code is first converted into MSIL which is stored in an assembly (DDL or EXE file)
  - At runtime MSIL is converted into native code by JIT compiler

This process ensure that code can be executed on different hardware architecture.

Assemblies

- → they are compiled code libraries used for deployment, versioning, security.
- → It can be either (.exe) or library (.dll)
  - Executable (exe) → assemblies that can be executed directly
  - Dynamic Link Library → simplifies deployment by bundling all necessary component.

Namespace

- → way to organize code into logical group helping to avoid name conflict and improve code readability.
- → use 'namespace' keyword to define namespace
- → we 'using' directive to include a namespace in your code.

How managed execution addresses the "DLL Hell" problem

- → Managed execution is how .NET code using CLR manages code execution, offering services like memory management, type safety, version control.
- → How it solve DLL Hell:
  1. Version control: Each assembly (DLL) has a version, so different apps can use different versions without conflict.
  2. Private Assemblies: App can keep their own DLL without affecting other
  3. No Registry Issue: DLL don't need to registered in windows, making thing simpler
  4. Global Assembly Cache (GAC): Shared DLL are stored in a central place, with versioning to avoid overwriting.
  5. Binding Redirects: Application can specify which assembly version to load.

Function Overloading in C#

→ there might be two or more method in a class with the same name but different number, types, and order of parameter, it is called method overloading.

```
class Program
{ static void Main()
    { Console.WriteLine(Add(5,3));
      Console.WriteLine(Add(5.5,3.3));
    }
  static int Add(int a, int b)
    { return a+b;
    }
  static double Add(double a, double b)
    { return a+b;
    }
}
```

Constructor & Destructor in C#

Constructor

→ is a special method that is automatically called an instance of class is created.
→ it is called explicitly when an object is created using 'new' keyword.
→ characteristic:
  → same name as class name
  → do not have return type not even void
  → we can have multiple constructor in a class, each with different parameter.

Destructor

→ is a special method that is automatically called when an object is destroyed or collected by the garbage collector.
→ characteristic:
  → same name as class name but with tilde (~) prefix
  → cannot have parameter or be overloaded
  → destructor are called automatically by garbage collector not manually by programmer.
  → uncommon: In most C# program, destructor is rarely needed bcz .NET framework provide automatic memory management.

Properties in C#

→ is a special types of members that provide a flexible mechanism to read, write, or compute the value of private field.
→ Key Points:
  • Properties usually consist of 'get' accessor: retrieves the value
    'set' accessor: assign a new value
  • Encapsulation: properties help encapsulate the field of class, allowing control over how values, are accessed or modified.

Indexers in C#

→ allows object to be indexed like arrays. They are similar to properties but are used to access elements in a collection-like manner
→ Indexer are defined using the 'this' keyword and can have more than one parameter making it possible to access element by index

```
public class Car
{ public string Make {get; set;}
  public string Model {get; set;}

  public Car(string make,
             string model)
    { Make = make;
      Model = model;
    }
  ~Car()
    { Console.WriteLine("Car
                destroyed");
    }
}

class Program
  { static void Main(string[]
                     args)
    { Car c1 = new Car
        ("Toyota", "A5");
    }
}
```

Operator overloading in C#

↳ give the ability to use the same operator to do various operation.

Syntax:

```
access specifier className operator operatorSymbol (parameter)
{ //code
}
```

Reflection API in C#

↳ let you look inside your code while its running

↳ You can:

- Inspect code : see details about classes, methods, properties at runtime
- Modify Behaviour : Create objects, call methods, or change properties dynamically
- Work with Attributes : Check and use custom tags (attribute) on your code

↳ It's like mirror for your program to analyze itself and even changes if needed.

↳ Is performed using System.Reflection namespace.

C#.NET features

- → support OOP concepts for modular & reusable code
- → ensure datatype consistency & reducing Runtime error
- → Automatic memory mngt
- → Asynchronous Programming
- → Rich standard Libraries → speed up development
- → Cross Platform Development
- → C# provide ~~event~~ event and delegate for implementing publisher subscriber pattern. These are used to handle events like button click in GUI application.

BCL support .NET App^n

- → provide core functionality for everyday program^n
- → include data structure such as list and dictionaries for data management.
- → support network communication using classes for sending HTTP request and handling network protocol.
- → include classes for connecting to dB and executing queries enabling efficient data manipulation
- → offer cryptographic service and mechanism for authentication & authorization.
- → provide threading and asynchronous programming support for writting concurrent and parallel code.

Eg. `Type type = typeof(MyClass); // Get type of class`

`object obj = Activator.CreateInstance(type); // ~~Get~~ create obj dynamically`

`MethodInfo method = type.GetMethod("SayHello") // Get Methods`

`method.Invoke(obj, null); // Call method dynamically`

~~obj : SayHello~~

String Manipulation techniques

- → Concatenation
- → Interpolation
- → Substring
- → Splitting
- → Joining
- → Trimming (removing whitespaces)
- → ToUpper & ToLower
- → Replacing
- → Checking for Substring (.contain)

ADO.NET → bridge b/t app^n & data source.

↳ Key technology for data access in .NET app^n & enabling you to interact with various dB, retrieve data & manipulate data.

Benefits

- Allow working with data offline
- provide tools for data manipulation
- improve performance by caching data and reducing repeated queries.
- ensure data integrity with reliable transaction handling.
- works with different database through a common UI
- Data Binding support : enable automatic updates of UI controls when data changes.

ADO.NET	ADO
• Uses dataset & DataTable to work with data offline	• maintain constant connection to dB
• It is CLR based library	• It is based on COM
• Data is stored in XML format.	• Data is stored in binary form
• It does not have feature of locking	• It has feature of locking
• require SQL JOINS and UNION to combine data	• ~~req~~ uses ~~Relational~~ DataRelational object to combine data
• It give us choice of using ~~one~~ client side cursor and server side cursor	• allow us to create client side cursor only.

Dataset → act as in-memory container for structured data ~~but~~ and exist within your app" memory.
- ↳ can hold one or more tables and relationship between them.
- ↳ offers : • Disconnected data Access
  • Data Manipulation
  • Data Consistency

Managed Provider → bridge b/t app" & data source
- ↳ act as a translator, understand specific comm" protocol of different databases and convert them into a common language that ADO.NET understand.
- ↳ functions : • Database Connectivity
  • Data conversion
  • SQL Execution.

SQLDataSource control
- ↳ allows us to access and manipulate data in ASP.NET page without using ADO.NET classes directly
- ↳ Connection to data is made through two important properties
  • ConnectionString
  • Provider Name
- ↳ Reading Data
  - ↳ ConnectionString and SelectCommand

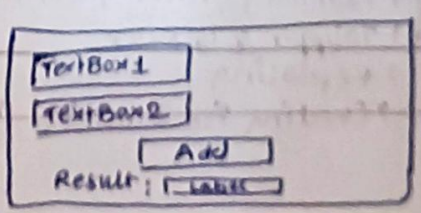
  Writing Data
  - ↳ 'InsertCommand', 'UpdateCommand', 'DeleteCommand' with parameter

```
<asp:SqlDataSource ID="..."
    runat="server" ConnectionString="<%$
    ConnectionStrings:YourConnectionString %>"
    SelectCommand="SELECT * FROM CUSTOMERS">
</asp:SqlDataSource>
```

Windows Form Control
- Button
- TextBox
- Label
- List Box
- PictureBox
- RadioButton
- TrackBar
- ProgressBar
- ComboBox

Window Form → UI framework used to create rich desktop based app".
- ↳ offering ready made controls that you simply drag and drop
- ↳ uses event driven model which means that program wait for the user to do something
- ↳ data you app" work with is
  • managed in models
  • the form display the data
  • and your code handle the logic

```
Using System;
using System.Windows.Forms;
namespace AddTwoNumbers
{ public partial class Form1 : Form
  { public Form1()
    { InitializeComponent();
    }
    private void button_Click(object sender,
                              EventArgs e)
    { int num1 = int.Parse
        (textBox1.Text);
      int num2 = int.Parse
        (textBox2.Text);
      int sum = num1 + num2;
      label1.Text = "Result:" +
        sum.ToString();
    }
  }
}
```



Page No.
Date

Menus → provide a structured way for users to access application features.
↳ always visible in the application
↳ help to organize feature
↳ common types include MenuStrip & ContextMenuStrip

① MenuStrip
- a menubar displayed at top of the form
- can have dropdown menus like File, Edit etc.

Eg. Drag a MenuStrip from toolbox
Add menu items (eg. "File" > "Open", "Save", "Exit")
Code for handle menu item
```
private void exitToolStripMenuItem_Click(object
                    sender, EventArgs e)
{ Application.Exit();
}
```

② ContextMenuStrip
- A right-click menu for specific control
- Add a contextMenuStrip from toolbox, define menuItems, and assign it to a control.

Eg. `textBox1.ContextMenuStrip = contextMenuStrip1;`

Dialogs → are pop-up windows used for user interaction like displaying messages or collecting input.
↳ simplify user interaction
↳ used for temporary interaction
↳ Eg. MessageBox, SaveFileDialog, OpenFileDialog etc.

MessageBox: for showing message

```
MessageBox.Show("Hello",
                "Title",
                MessageBoxButtons.OK);
```

Tooltips → provide a small pop-up hints or descriptive text that appears when a user hovers over a control or UI element.
↳ purpose is to provide additional info without cluttering the UI

Eg:
```
ToolTip toolTip = new
                  ToolTip();
toolTip.SetToolTip(button1,
                   "Click here to submit");
toolTip.SetToolTip(textBox1,
                   "Enter your name.");
```

Printing in Windows Form

- to handle printing in WF use PrintDocument which provide functionality to define what and how to print
- We can add an object of this class to the project and then handle event such as PrintPage
- optionally, use PrintDialog for user config print function.
- Finally, call print() function to start print process

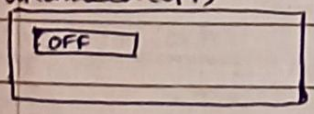
GDI+

- graphic library in windows that provide serive for 2D graphic rendering, image processing and typography
- offers base class that enable user to draw different custom shapes or drawing on screen
- key component
  - graphic object → provide method to draw shapes
  - Pen and Brushes → used to add color & style
  - Coordinate System → use X & Y point to place things on screen
  - Text Rendering → write text in diff. font & style.
  - Displaying img → using Graphic.DrawImage()
  - Matrix → manages transformation (scale, rotate)

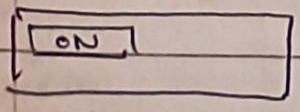
```
private void Form1_Paint(object sender, PaintEventArgs e)
{ Graphics g = e.Graphics;
  g.DrawRectangle(Pen, 50, 50, 100, 50);
  g.FillRectangle(Brush, 50, 150, 100, 50);
}
```

```
public static void Main()
{ Application.EnableVisualStyles();
  Application.Run(new CustomForm());
}
}
```

o/p: Unchecked (OFF)



Checked (ON)



Handle Multiple Events

Can be handle means reacting to different user actions like button click, text change etc.. in your application.

1. Add Controls to the Form → Drag & drop
2. Write Event Handler → create methods to handle what should happen when an event occurs.
3. Connect event to Handler → Link event to method you wrote. This is done automatically in the designer or in your code.
4. Handle Similar Events in One Method → If multiple control do similar things, you can use one method to handle them all and figure out which control triggered the event

Eg. If Multiple Button do same Thing:

```
private void Button_Click(object sender,
                          EventArgs e)
{ Button clickedButton = sender as Button;
  MessageBox.Show($"{clickedButton.Text}
                  was clicked");
}

Button1.Click += Button_Click;
Button2.Click += Button_Click;
```

Custom Windows form control

```
using System;
using System.Drawing;
using System.Windows.Forms;
public class ToggleSwitch : CheckBox
{ protected override void OnPaint(PaintEventArgs e)
  { //Background color
    e.Graphics.FillRectangle(Checked ? Brushes.
                LightGreen : Brushes.LightCoral,
                ClientRectangle);
    //Toggle circle
    int size = Height - 4;
    e.Graphics.FillEllipse(Brushes.White,
                Checked ? Width - size - 4 : 2, 2, size,
                size);
  }
}
public class CustomForm : Form
{ public CustomForm()
  { this.Text = "Minimal Toggle Switch";
    this.Size = new Size(300, 150);
    var toggle = new ToggleSwitch
    { Width = 100,
      Height = 40,
      Location = new Point(100, 50)
    };
    this.Controls.Add(toggle);
```

ASP.NET

- Web framework developed by Microsoft
- enable developer to create dynamic webpages, APIs, web services.
- is a part of .NET framework and provide robust platform for developing web app" with rich set of features
- ASP.NET Functionality :
  - Event Handling → execute server side logic, handle events when the button is clicked
  - Dynamic Web Page Creation → enable the creation of interactive and responsive web pages
  - Web API → facilitate creation of RestFul APIs for data exchange in JSON and XML format.
  - Security → Built in authentication & authorization features.
  - Data Access → connect to various data sources dB using ADO.NET or Entity Framework
  - Razor Pages → simplified way to create page-focused application using Razor syntax
  - WebSocket & Asynchronous programing → using asyn & await keyword
    - for two way comm" b/t server & client

In ASP.NET, WebControl & HTMLControls serve to create interactive UI for web app"

Both control allow developer to build forms and manage user input.

Web control → server side control

- provide rich functionality, automatic state mgnt, easy integration with server side logic.
- Common Web Controls :
  - Button • TextBox • Label
  - DropDownList • GridView

Eg. `<asp: TextBox ID = "txtName" runat = "server" />`

HTML control → client side processing

- do not have built in state mgnt.
- Standard HTML element wrapped in ASP.NET control that can be used on server side.
- Common HTML controls :
  - Input - basic i/p field for user input
  - Button - standard HTML Button
  - Select - dropdown list
  - Text area - multiline i/p field

Eg. `<input type = "text" id = "txtName" runat = "server" />`

Rich Server Controls

- are advanced, server side components used in ASP.NET to create interactive web app".
- Key Features :
  - Data Binding : easily connect to data source
  - State Mngt : automatically maintain state across postbacks using ViewState.
  - Event Handling : support various event that allow user to execute server side logic
  - Customizability : allow developer to modify appearance without extensive coding
  - Built In Functionality : comes with built in feature that simplify task such as validation, formating and paging.

Master Pages

- are parent pages which define overall layout of web Application and all other child pages are derived from these master pages
- contain markups, controls, banners, navigation menus and that you want to include in all pages on your website
- include standard and HTML controls
- you can define a master page by specifying the @Master directive at top of a web page.
- By default, a master page contain a single ContentPlaceHolder control which add dynamic content to a web Page.

Themes → help in providing similar overall look to all pages.

- allow you to apply consistent style and layout across entire website.
- To create consistent website using ASP.NET 2.0 Theme

① Create a Theme Folder
- In root of project create 'App_Theme Folder and add subfolder (eg. MyTheme)

② Add Skins
- create .skin file to define appearance of ASP.NET controls like Button, TextBox

```
<asp: Button runat = "server"
    BackColor = "LightBlue"
    ForeColor = "White" />
```

③ Add CSS & Image
- add css file (eg. style.css) to control layout & style

```
body {
    background-color : #f0f0f0;
}
```

④ Apply Theme
- to apply theme to all pages in your theme you can specify theme in web.config file

Globally, `<pages theme = "MyTheme" />`

on individual pages using @page directive
`<%@page Theme = "MyTheme" %>`

Login Controls

- simplifies user authentication by providing an out-of-the box UI for log-in, along with built-in logic for user verification.
- Pre built controls:
  - Login
  - LoginView
  - LoginStatus
  - CreateUserWizard
  - PasswordRecovery

Validation controls

- ensure that data entered by user on a webform is valid before it is processed.
- Built-in controls:
  - RequiredFieldValidator → ensure field not empty
  - RegularExpressionValidator
  - CompareValidator
  - RangeValidator → check values falls in specific range

WPF (Windows Presentation Foundation)

- UI framework for building Windows desktop Application.
- provide rich, modern, visually appealing UI.
- It seperate design and logic making it easier for developer and designers to collaborate.
- uses markup language XAML to define UI in declarative manner.
- Connect UI element directly to data sources, reducing boilerplate code.
- support MVVM pattern making app" more modulors and testable.
- every control in WPF can be re-templated, allowing complete customization.
- easily add animation, videos and 2D/3D graphics

WCF (Windows Communication foundation)

- framework developed for building ~~service~~ service oriented application.
- enable app" to communicate across diff. platform networks and protocol
- Service Oriented ~~Arch~~ Architecture → designed loosely coupled, reusable services that can be accessed independently.
- provide built in feature of encryption, authentication
- support synchronous, asynchronous, one-way and duplex messaging

Accessing Data using ADO.NET in ASP.NET

(1.) Add Connection String to web.config:

```
<connectionStrings>
  <add name="DbConnectionString"
       connectionString="Server=your_server;
                         Database=your_db;
                         User Id=your_user;
                         Password=your_pass;"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

(2.) Code to fetch data

```
using System;
using System.Data.SqlClient;
public partial class FetchData : System.Web.UI.Page
{ protected void Page_Load(object sender, EventArgs e)
  { string connString = System.Configuration.ConfigurationManager.
      ConnectionStrings["DbConnectionString"].ConnectionString;
    string query = "SELECT * FROM EMPLOYEES";
    using (SqlConnection conn = new SqlConnection(connString))
    using (SqlConnection cmd = new SqlCommand(query, conn))
    { conn.Open()
      GridViewEmployee.DataSource = cmd.ExecuteReader();
      GridViewEmployee.DataBind();
    }
  }
}
```

(3) Add Grid View to ASPX Page

```
<asp:GridView
    ID="GridViewEmployee"
    runat="server"
    AutoGenerateColumns="True" >
</asp:GridView>
```

GridView Control

↳ allow you to display, filter, and manage data in a tabular format.

- Displaying Data.

(1) Bind Data to GridView

```
<asp:GridView ID="" runat="Server" AutoGenerateColumn="false">
    <Columns>
        <asp:BoundField DataField="Num" HeaderText="Name" />
        <asp:BoundField DataField="Age" HeaderText="Age" />
    </Columns>
</asp:GridView>

GridView.DataSource = myDataSet.Tables[0]
GridView.DataBind();
```

- Filtering Data

↳ involve modifying the data source before binding.

```
String FilterExpression = "Age > 20"
DataView dv = new DataView(MyDataSet.Table[0]);
dv.RowFilter = FilterExpression;
GridView1.DataSource = dv;
GridView1.DataBind();
```

Cookies → Small piece of data stored on the client browser

Creating

```
HttpCookie cookie = new HttpCookie("UserPreference")
cookie["Theme"] = "Dark";
Cookie.Expire = DateTime.Now.AddDays(30);
Response.Cookie.Add(cookie)
```

Retrieving

```
HttpCookie cookie = Request.Cookies["User Preference"];
if (cookie != null)
{ String theme = cookie["Theme"];
}
```

Two main approach to allow users to select from a dropdown list in a GridView within ASP.NET

① Using Template

- Define data source
- Create Template
  - ↳ ItemTemplate
  - ↳ EditTemplate
- Add DropDownList control → within EditTemplate, add DropDownList control and bind it to data source from the dropdown items

```
<asp:TemplateField>
    <ItemTemplate><%# Eval("Category Name") %></ItemTemplate>
    <EditItemTemplate>
        <asp:DropDownList DataSourceID="" DataTextField="CategoryName"
            DataValueField="CategoryID"
            SetValues='<%# Bind("Category ID") %>' />
    </EditItemTemplate>
</asp:TemplateField>
```

② BoundField

- Define data source
- Add Bound field
- set dataTextfield and dataValuefield

```
<asp:BoundField DataField="Category ID"
    DataSourceID=""
    DataTextField="CategoryName"
    DataValueField="CategoryID" />
```

Adding HyperLink to GridView

- Define data source
- Create HyperLinkField or TemplateField
- Set HyperLink Properties: specify URL and text for hyperlink.
- Bind Data

```
<asp:TemplateField HeaderText="Link">
    <ItemTemplate>
        <asp:HyperLink ID="" runat="server"
            NavigateURL='<%# Bind("Text") %>' />
    </ItemTemplate>
</asp:TemplateField>
```

Session State

- allow you to store and retrieve user specific data for a duration of user session
- Key features:
  - User specific → each user have their own session data.
  - Serverside → data is stored on user not on user browser.
  - Temporary → The session last until the user closes the browser or the session time is out.
- configuration of session state

```
<configuration>
  <System.Web>
    <SessionState mode="InProc"
                  timeout="20" />
  </System.Web>
</configuration>
```

UDDI

- centralized directory where web servies offered by different organization are published in UDDI.
- provide a single location where client can search the web services offered by different organization.
- Role of UDDI:
  - Registry of web Services.
  - Discovery
    - client can search UDDI directories to find web services that meet their need based on keyword, service, type etc.
  - Standard Description
    - uses standardized meta data (such as WSDL) to describe how web services can be accessed and invoked.
  - B2B Integration
    - useful for B2B interaction which allows businesses to discover and integrate each other services without needing pre existing aggreements.

XML Web services (SOAP services)

Creating a XML Web Services

1. Create a New Project
2. Add WebServices: Project > Add New Item > WebService and provide name for services.
3. Define Method → create public method within a web service class that you want to expose as web service.

```
public int AddNumber (int a, int b)
{ return a+b;
}
```

4. Build and deploy

Press F5 to build and run the project.

Main motivation of using XML Web Servies

- Interoperability → allow diff. software system to communicate with each other via protocol
- Distributed Computing → Web Services provide mechanism for building distributed app^n
- Platform Independence → enable data sharing and running fun^n b/t diff. system
- Loose Coupling → make system less dependent on each other so that they can be updated or maintened more easily

Process of designing XML Web Services

1. Identify service Boundries
   - determine functionality your service will provide.
2. Choose comm^n protocol → such as SOAP or REST
3. Define operation & method
4. Design the Data contract
   - uses XML Schema (XSD) to define structure of the data that will be exchanged. This ensure both client and server can validate the data.
5. Create WSDL
   - XML document describe the methods your services offers, comm^n protocol, enabling client to understand how to interact with the services.
6. Handling exception and faults
7. Security and authentication: uses security measure to ensure secure comm^n b/t client and the web servies