Week 3 - C# NOTES

Introduction

.NET Framework 1.0 (2002)

- Introduced C#, VB.NET as primary languages
- Introduced ASP.NET for web apps, windows forms for desktop apps
- Introduced CLR

.NET Framework 2.0 (2005)

- Visual studio 2005
- Generics, ASP.NET 2.0, ADO.NET

.NET Framework 3.0 (2006)

- WPF(Windows Presentation Foundation), WCF (Windows communication foundation), WF(Workflow foundation), and cardSpace

.NET Framework 3.5 (2008)

- Visual studio 2008
- LINQ, ASP.NET AJAX

.NET Framework 4.0 (2010)

- Visual Studio 2010
- Parallel programming, MEF(Managed Extensibility Framework), Entity Framework
- ASP.NET MVC

.NET Framework 4.5 (2012)

- Visual Studio 2012
- async/await

.NET Framework 4.6 (2015)

- Visual Studio 2015

RyuJIT compiler for 64-bit, better performance, and security updates

.NET Framework 4.7 & 4.8 (2016-2019)

Latest major release

.NET Core(2016)

cross platform, open-source

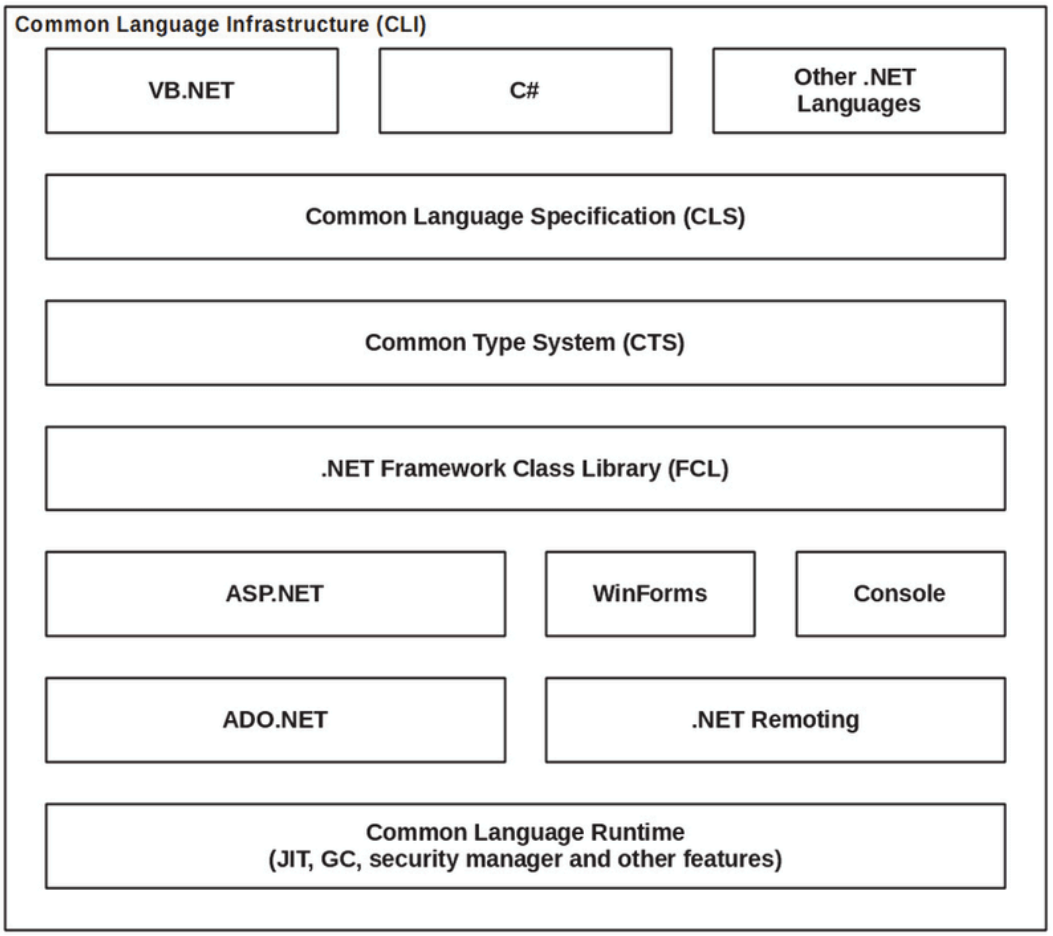
.NET 5(2020)

Unified merging .NET Core, .NET Framework, Xamarin, Mono

.NET 6, .NET 7, .NET 8 and .NET 9

Latest with LTS(Long term support)

Architecture of .NET Framework



CLR: Common Language Runtime

It is a execution engine that manages applications while they are running.

It include:

- memory management: Use Garbage Collector(GC) to manage memory
- Just-In-Time (JIT) : Complier that converts MSIL(Microsoft Intermediate Language) code to machine code
- Type safety and security: It verifies the type safety of the code before execution, preventing operations that could compromise memory or stability
- Exception Handling: It provide structure for errors and exception handling

FCL: Framework Class Library

It is collection of important classes

It provide class for file handling, Database connectivity(ADO.NET), string manipulation, text processing, XML operation, Web application development(ASP.NET), Desktop development(WPF, Windows forms)

CTS: Common type system

It is important for language interoperability

It like a dictionary for types

It have two kinds of types

- value types: int, double, bool, struct
- reference types: class, interface, delegate, string

CLS: Common Language Specifications

CLS is a specific set of rules and constraints that developers are encouraged to follow when creating public APIs

It is like a guideline to ensure language interoperability

Folder structure

/bin folder:

binary folder

Contains compiled output(executables and libraries) of project after build

/obj folder:

Used for incremental builds → speed up builds by not recompiling everything each time

.csproj file:

C# project file

XML file containing project configuration and metadata

It stores target framework(net6.0, net 8.0), nuget package references, build settings(Debug/release), output typ (Console app, web app, library)

.sln file:

Solution fule, used by visual studio to organize one or more projects into a single workspace

Datatype

Value type: byte, sbyte, short, ushort, int, uint, long, ulong, char, float, double, decimal, bool, enum, struct

Reference type: string, object, class, interface, delegate

Float and double is not recommanded for financial calculations

```
float a = 0.1f;
double b = 0.1;
// Not recommended for financial calculations
Console.WriteLine(a + b); // 0.20000000149011612

// Use decimal for financial calculations
decimal c = 0.1m;
```

```
decimal d = 0.2m;
Console.WriteLine(c + d); // 0.3
```

Access Modifiers

Access modifiers are keywords used to specify the declared accessibility of a member or a type.

- `public`: Access isn't restricted.
- `protected`: Access is limited to the containing class or types derived from the containing class.
- `internal`: Access is limited to the current assembly.
- `protected internal`: Access is limited to the current assembly or types derived from the containing class.
- `private`: Access is limited to the containing type.
- `private protected`: Access is limited to the containing class or types derived from the containing class within the current assembly.
- `file`: The declared type is only visible in the current source file.

The accessibility of member is always restricted by its container (Ex: A public member of a private class cannot be accessed outside)

▼ Ex:

```
namespace AccessModifier
{
    public class T1
    {
        public static int publicInt;
        internal static int internalInt;
        private static int privateInt = 0;

        static T1()
        {
            // T1 can access public or internal members
            // in a public or private (or internal) nested class.
```

```
        M1.publicInt = 1;
        M1.internalInt = 2;
        M2.publicInt = 3;
        M2.internalInt = 4;
        publicInt = 1;
        internalInt = 2;
        privateInt = 3;

        // Cannot access the private member privateInt
        // in either class:
        // M1.privateInt = 2; //CS0122
    }

    public class M1
    {
        public static int publicInt;
        internal static int internalInt;
        private static int privateInt = 0;
    }

    private class M2
    {
        public static int publicInt = 0;
        internal static int internalInt = 0;
        private static int privateInt = 0;
    }
}

public class MainClass
{
    public static void Main()
    {
        // Access is unlimited.
        T1.publicInt = 1;

        // Accessible only in current assembly.
        T1.internalInt = 2;
```

```
            // Error CS0122: inaccessible outside T1.
            //T1.privateInt = 3;

            // Access is unlimited.
            T1.M1.publicInt = 1;

            // Accessible only in current assembly.
            T1.M1.internalInt = 2;

            // Error CS0122: inaccessible outside M1.
            //    T1.M1.privateInt = 3;

            // Error CS0122: inaccessible outside T1.
            //    T1.M2.publicInt = 1;

            // Error CS0122: inaccessible outside T1.
            //    T1.M2.internalInt = 2;

            // Error CS0122: inaccessible outside M2.
            //    T1.M2.privateInt = 3;
        }
    }
}
```

The direct base class must be at least as accessible as the derived class.

```
class BaseClass {...}
public class MyClass: BaseClass {...} // Error
```

> Top level types like class, struct, enum, interface, delegate defined directly inside namespace has default accessibility level internal and does not allow to have private, protected, protected internal, private protected

Accessibility of members

enum: default and allowed is public

class : default is private and allowed is public, protected, internal, private, protected internal, private protected

interface: default is public and allowed is public protected internal, private, protected internal, private protected

struct: default is private and allowed is public, internal, private

Using namespace

The `namespace` keyword is used to declare a scope that contains a set of related objects. You can use a namespace to organize code elements and to create globally unique types.

A namespace can contain class, interface, struct, enum, delegate, or other namespace

A namespace is a logical unit of code which help to avoid name conflict

A namespace can span across multiple files means same namespace name can be used in two files and it will be combined

namespacename.something means it is nested namespace

▼ syntax:

```
namespace SampleNamespace
{
   class SampleClass { }

   interface ISampleInterface { }

   struct SampleStruct { }

   enum SampleEnum { a, b }

   delegate void SampleDelegate(int i);
```

```
    namespace Nested
    {
        class SampleClass2 { }
    }
}
```

```
// file scoped

using System;

namespace SampleFileScopedNamespace;

class SampleClass { }

interface ISampleInterface { }

struct SampleStruct { }

enum SampleEnum { a, b }

delegate void SampleDelegate(int i);
```

File scoped namespace ends at end of file.

If no namespace is provided it will consider it as global namespace

global namespace is available to all files without using directive

Namespace have implicitly public access

The `using` directive allows you to use types defined in a namespace without specifying the fully qualified namespace of that type.
A `using` directive doesn't give you access to any namespaces that are nested in the namespace you specify.

Adding the `global` modifier to a `using` directive means that using is applied to all files in the compilation (typically a project):

Ex:

```
global using static System.Math;
```

A global using must be before any other using

A using must be before any declaration of namespace

The `using static` directive names a type whose static members and nested types you can access without specifying a type name.

Create a `using` alias directive to make it easier to qualify an identifier to a namespace or type

```
using s = System.Text;
```

```
using s = System.Text;
using s.RegularExpressions; // Generates a compiler error.
```

A using alias directive can't have an open generic type on the right-hand side. For example, you can't create a using alias for a `List<T>`, but you can create one for a `List<int>`.

ArrayList

Dynamically resizing array that can hold items of any type because it stores them as object

Non-generic

To add element

```
ArrayList arrayList = new ArrayList();
arrayList.Add(1);
arrayList.Add("string");
arrayList.Remove(1); // Remove item with value 1
arrayList.RemoveAt(0); // Remove item at index 0
arrayList.Insert(2, 10); // index, value
Console.WriteLine("Size: " + arrayList.Count);
foreach (var item in arrayList)
{
    Console.WriteLine(item);
}
```

Hashtable

Hashtable is a collection of key-value pairs

Keys and values are stored as object

```
Hashtable hashtable = new Hashtable();
hashtable.Add("one", 1);
hashtable.Add("two", 2);
hashtable["three"] = 3; // Using indexer to add item
Console.WriteLine("Size: " + hashtable.Count);
Console.WriteLine(hashtable.ContainsKey("two")); // True
Console.WriteLine(hashtable.ContainsValue(4)); // False
Console.WriteLine(hashtable["three"]); // 3
foreach (string item in hashtable.Keys)
{
    Console.WriteLine(item);
}
```

List

Type-safe, flexibal replacement of ArrayList, use generic

```
List<int> list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);
Console.WriteLine("Size: " + list.Count); // 3
//list.Remove(2); // Remove item with value 2
//list.RemoveAt(0); // Remove item at index 0
list.Insert(1, 10); // Insert 10 at index 1
Console.WriteLine(list.Contains(3)); // True
Console.WriteLine(list.Find(x ⇒ x > 2)); // 10
foreach (var item in list.FindAll(x ⇒ x > 1))
{
    Console.WriteLine(item); // 10, 2, 3
}
```

Dictionary

modern, type safe replacement for hashtable

It store unique keys and their associated values

```
Dictionary<string, int> dictionary = new Dictionary<string, int>();
dictionary.Add("one", 1);
dictionary.Add("two", 2);
dictionary.Add("two", 22); // throw exception as key already exists
dictionary["three"] = 3; // Using indexer to add item
Console.WriteLine("Size: " + dictionary.Count); // 3
Console.WriteLine(dictionary.ContainsKey("two")); // True
Console.WriteLine(dictionary.ContainsValue(4)); // False
Console.WriteLine(dictionary["three"]); // 3
if (dictionary.TryGetValue("three", out int value))
{
    Console.WriteLine(value);
}
else
{
    Console.WriteLine("Key not found" + value);
```

```
}
foreach (string item in dictionary.Keys)
{
    Console.WriteLine(item);
}
```

Stack

Generic stack

```
Stack<int> stack = new Stack<int>();
stack.Push(1);
stack.Push(2);
stack.Push(3);
Console.WriteLine("Size: " + stack.Count); // 3
Console.WriteLine(stack.Pop()); // 3 → use with try-catch as it throws exce
ption if stack is empty
Console.WriteLine(stack.Peek()); // 2
```

Queue

Generic queue

```
Queue<int> queue = new Queue<int>();
queue.Enqueue(1);
queue.Enqueue(2);
queue.Enqueue(3);
Console.WriteLine("Size: " + queue.Count); // 3
Console.WriteLine(queue.Dequeue()); // 1 → use with try-catch as it throws
exception if queue is empty
Console.WriteLine(queue.Peek()); // 2
```

ENUM

An *enumeration type* (or *enum type*) is a value type defined by a set of named constants of the underlying integral numeric type.

```
enum Season
{
    Spring,
    Summer,
    Autumn,
    Winter
}
```

By default it associate a constant int value with each type starting with zero

We can give custom value to it

```
enum ErrorCode : ushort
{
    None = 0,
    Unknown = 1,
    ConnectionLost = 100,
    OutlierReading = 200
}
```

To verify variable is valid or not

```
bool isValid = Enum.IsDefined(typeof(enum_name), variable);
```

If you have not defined an enumeration member whose value is 0, consider creating a `None` enumerated constant. By default, the memory used for the enumeration is initialized to zero by the common language runtime

Enumeration using flags

We define enum with [Flags] attribute and give value in power of two

So one variable can hold multiple values

And we can use bitwise operator to compare

Ex:

```
[Flags]
public enum FileAccess
{
   None = 0,
   Read = 1,
   Write = 2,
   Execute = 4
}
```

```
FileAccess permission = FileAccess.Read | FileAccess.Write;
Console.WriteLine($"Permission: {permission} value is ");

if((permission & FileAccess.Write) == FileAccess.Write)
{
   Console.WriteLine("Write permission granted.");
}
else
{
   Console.WriteLine("Write permission denied.");
}
```

Typecasting

We can type cast like (int)variable

```
public enum ArrivalStatus { Unknown=-3, Late=-1, OnTime=0, Early=1 };
```

```
int value3 = 2;
ArrivalStatus status3 = (ArrivalStatus)value3;

int value4 = (int)status3;
```

Parse Enumeration value

```
ArrivalStatus status1 = (ArrivalStatus)Enum.Parse(typeof(ArrivalStatus), -1);
```

Date

All are in namespace (System)

Struct DateTime for manipulation

Struct Timespan for time difference

Struct DateTimeOffset for DateTime with time zone

Struct DateOnly to clear representation of date (after .net 6)

Struct TimeOnly to clear representation of time (after .net 6)

```
Console.WriteLine(DateTime.Now); // Current date and time  → 22/09/2025 3:06:11 pm
Console.WriteLine(DateTime.Today.Month); // Current month → 9
Console.WriteLine(DateTime.Now.DayOfWeek); // Current day of the week Monday, Tuesday, etc. → Monday
Console.WriteLine(DateTime.Today.AddDays(5).AddHours(3).AddMinutes(30)); // Date and time 5 days, 3 hours, and 30 minutes from now → 27/09/2025 3:30:00 am

DateTime dt = new DateTime(2026, 1, 1);
Console.WriteLine(dt.Subtract(DateTime.Now));  // Timespan → 100.08:53:11.3445369
Console.WriteLine(dt.ToString("d")); // 01/01/2026
Console.WriteLine(dt.ToString("D")); // Wednesday, 01 January 2026
Console.WriteLine(DateTime.Now.ToString("dd-mm-yyyy HH:mm:ss tt")); // Custom format → 22-07-2025 15:07:06 pm

bool isValid = DateTime.TryParse("01/01/01 15:01:20", out DateTime parsedDate);
```

```
if (isValid)
{
   Console.WriteLine(parsedDate); // 01/01/2001 3:01:20 PM
}

DateTime diwali = new DateTime(2025, 10, 20);
Console.WriteLine(diwali.Subtract(DateTime.Now));  // diwali - currentTime
→ 27.09:05:38.7503654    27 days 9 hours 5 minutes 38 seconds
Console.WriteLine(diwali.Subtract(DateTime.Now).TotalDays); // 27.378125
72086574  total days as fraction
Console.WriteLine(diwali.Subtract(DateTime.Now).TotalHours); // 657.0662
035800833
Console.WriteLine(diwali.Subtract(DateTime.Now).Hours); // 9 hours part o
nly

TimeSpan duration = diwali - DateTime.Now;
Console.WriteLine($"Time until Diwali: {duration.Days} days, {duration.Hou
rs} hours, {duration.Minutes} minutes, {duration.Seconds} seconds");

DateTime now = DateTime.Now;
TimeSpan fiveHour = TimeSpan.FromHours(5);
Console.WriteLine(now + fiveHour); // current time + 5 hours → 22/09/202
5 8:07:54 pm
Console.WriteLine(now.Add(fiveHour)); // current time + 5 hours → 22/09/2
025 8:07:54 pm

DateTimeOffset local = DateTimeOffset.Now;
Console.WriteLine(local); // current date and time with offset  22/09/2025
3:04:26 pm +05:30
Console.WriteLine(local.Offset); // 05:30:00

DateTimeOffset utc = DateTimeOffset.UtcNow;
Console.WriteLine(utc); // 22/09/2025 9:34:43 am +00:00



Console.WriteLine(DateOnly.FromDateTime(DateTime.Now)); // Current dat
e → 22/09/2025
Console.WriteLine(DateOnly.FromDateTime(DateTime.Now).AddDays(10));
```

```
// Date 10 days from now → 02/10/2025
Console.WriteLine(DateOnly.Parse("01/01/2026").Year); // 2026

Console.WriteLine(TimeOnly.FromDateTime(DateTime.Now)); // Current tim
e → 3:11 pm

TimeOnly start = new TimeOnly(9, 0, 0); // 9 AM
TimeOnly end = new TimeOnly(17, 0, 0); // 5 PM
TimeOnly currTime = TimeOnly.FromDateTime(DateTime.Now);
Console.WriteLine(currTime.IsBetween(start, end) ? "Working hours" : "Off
hours"); // Working hours
```

String

String class → Immutable string (Namespace: System)

```
string message = "Hello, World! Welcome to C# programming.";
Console.WriteLine(message);
Console.WriteLine(message.Length); // Length of the string → 40
Console.WriteLine(message.IndexOf("C#"));  // Position of substring → 25
Console.WriteLine(message.Substring(25)); // start index → C# programmi
ng.
Console.WriteLine(message.Substring(25, 2)); // start index, len → C#
Console.WriteLine(message.Replace("C#", "CSharp")); // Hello, World! Wel
come to CSharp programming.
Console.WriteLine(message.ToUpper()); // HELLO, WORLD! WELCOME TO
C# PROGRAMMING.

string m2 = "     hi     ";
Console.WriteLine(m2.Trim()); // "hi"
Console.WriteLine(m2.TrimStart()); // "hi     "
Console.WriteLine(m2.TrimEnd()); // "     hi"

string[] words = message.Split(' ');
foreach (var word in words)
{
```

```
    Console.WriteLine(word);
}

Console.WriteLine(string.Join(" ", words));  // separator, IEnumerable<strin
g> → Hello, World! Welcome to C# programming.

Console.WriteLine(string.IsNullOrEmpty(""));  // True
Console.WriteLine(string.IsNullOrEmpty(" ")); // False
Console.WriteLine(string.IsNullOrEmpty(" ABC ")); // False

string name = "dakshil";
Console.WriteLine(string.Format("Hello {0}", name));  // Hello dakshil

Console.WriteLine(message.Contains("C#") ? "Found" : "Not Found"); // Fo
und
Console.WriteLine(message.StartsWith("Hello") ? "Yes" : "No"); // Yes
Console.WriteLine("abc" + "def"); // abcdef
Console.WriteLine("abc" == "abc"); // True
```

StringBuilder class → Mutable string (Namespace: System.Text)

```
StringBuilder stringBuilder = new StringBuilder("Hello ");
stringBuilder.Append("Dakshil");
Console.WriteLine(stringBuilder); // Hello Daskhil

stringBuilder.Insert(0, "HI"); // start index, string
Console.WriteLine(stringBuilder);  // HiHello Dakshil

stringBuilder.Remove(0, 2); // start index, len
Console.WriteLine(stringBuilder);  // Hello Dakshil

stringBuilder.Replace("Dakshil", "World");
//stringBuilder[5] = 'H';   // no error
Console.WriteLine(stringBuilder);  // Hello World

string final = stringBuilder.ToString(); // Converts to immutable string
//final[5] = ','; // error: cannot modify but can read
```

```
Console.WriteLine(final);  // Hello World
Console.WriteLine(final[1]);
```

Regex class → Use to manipulate string and find string using regular expression (Namespace: System.Text.RegularExpressions)

```
string pattern = @"^\w+@\w+\.\w{2,4}$";
Console.WriteLine(Regex.IsMatch("test@gmail.com", pattern));

string patternNumber = @"\d";
Match result = Regex.Match("ab32b32n2234nj", patternNumber);
if (result.Success)
{
   Console.WriteLine(result.Value); // 3
}

MatchCollection result2 = Regex.Matches("ab32b32n2234nj", patternNum
ber);
foreach (Match m in result2)
{
   Console.Write(m.Value + " "); // 3 2 3 2 2 2 3 4
}
Console.WriteLine();

string replaced = Regex.Replace("ab32b32n2234nj", patternNumber, "#");
Console.WriteLine(replaced); // ab##b##n####nj
```

Math

Math class (Static class, Namespace : System)

```
Console.WriteLine("PI : " + Math.PI); // 3.1415926535897793
Console.WriteLine("E : " + Math.E); // 2.718281828459045

double num = 5.67;
```

```
Console.WriteLine("Abs : " + Math.Abs(-num)); // 5.67
Console.WriteLine("Ceiling : " + Math.Ceiling(num)); // 6
Console.WriteLine("Floor : " + Math.Floor(num)); // 5
Console.WriteLine("Round : " + Math.Round(num)); // 6
Console.WriteLine("Truncate : " + Math.Truncate(num)); // 5

Console.WriteLine(Math.Pow(2, 8)); // 256
Console.WriteLine(Math.Sqrt(64)); // 8

Console.WriteLine(Math.Max(10, 20)); // 20
Console.WriteLine(Math.Min(10, 20)); // 10

Console.WriteLine(Math.Sign(-55)); // -1
Console.WriteLine(Math.Sign(0)); // 0
Console.WriteLine(Math.Sign(55)); // 1

Console.WriteLine(Math.Sin(15)); // 0.6502878401571168

Console.WriteLine(Math.Exp(1)); // 2.718281828459045
Console.WriteLine(Math.Log(Math.E)); // natural log → 1
Console.WriteLine(Math.Log10(100)); // base 10 log → 2
Console.WriteLine(Math.Log(256, 2)); // base 2 log → 8
```

Data Tables

DataTable is a c# object that represent a single, in-memory table of data.

It is fundamental part of ADO.NET

Namespace: System.Data;

DataTable contains three main components: DataColumn, DataRow, Constraint

To create DataTable

```
DataTable StudentTable = new DataTable("Student");
StudentTable.Columns.Add("ID", typeof(int));
StudentTable.Columns.Add("Name", typeof(string));
StudentTable.Columns.Add("Age", typeof(int));
StudentTable.Columns.Add("Department", typeof(int));
StudentTable.PrimaryKey = new DataColumn[] { StudentTable.Columns["I
D"] };
```

To add foreign key constaint

```
ForeignKeyConstraint fk = new ForeignKeyConstraint(
    "FKStudentDepartment",
    DepartmentTable.Columns["ID"],             // Parent Column
    StudentTable.Columns["Department"]      // Child Column
);

fk.DeleteRule = Rule.Cascade;
fk.UpdateRule = Rule.Cascade;
StudentTable.Constraints.Add(fk);
```

Foreign key doesn’t give navigation like Relation

To add foreign key using relation

```
DataRelation relation = dataSet.Relations.Add("StudentDepartment",
    DepartmentTable.Columns["ID"],             // Parent Column
    StudentTable.Columns["Department"]);       // Child Column

relation.ChildKeyConstraint.DeleteRule = Rule.Cascade;
relation.ChildKeyConstraint.UpdateRule = Rule.Cascade;
```

To add rows

```
DepartmentTable.Rows.Add(1, "CSE");
DepartmentTable.Rows.Add(2, "IT");
```

Select method

```
DataRow[] result = StudentTable.Select("Age > 22", "Age DESC");
Console.WriteLine("Student with Age > 22:");
foreach (DataRow row in result)
{
    Console.WriteLine($"Student: {row["Name"]}, Age: {row["Age"]}");
}
```

We can use basic operation (=, <>, <, >, ≤, ≥)

To use string enclose it with ‘’

```
.Select("Name='dakshil'")
```

To use date enclose with ##

```
.Select("Date=#2025-01-15#");
```

We can use logical operator (AND, OR, NOT)

We can use LIKE, IN, IS NULL

File read write

File class:

- Static class
- System.IO
- Provide high level access for reading, writing file with simple functions

FileStream class:

Namespace: System.IO

Give low level control over file

StreamReader, StreamWriter:

Namespace: System.IO

Used to read and write data Line by Line without reading entire file and storing in memory

To read using File

```
string content = File.ReadAllText(filePath);
```

To read using StreamReader

```
using (StreamReader reader = new StreamReader(filePath))
{
    while (!reader.EndOfStream)
    {
        string line = reader.ReadLine();
        Console.WriteLine(line);
    }
}
```

To read using FileStream

```
using (FileStream fs = new FileStream(filePath, FileMode.Open, FileAccess.
Read))
{
    fs.Seek(10, SeekOrigin.Begin); // Move to the 10th byte from the start
    byte[] buffer = new byte[50];
    fs.Read(buffer, 0, buffer.Length);
```

```
    string result = System.Text.Encoding.UTF8.GetString(buffer);
    Console.WriteLine(result);
}
```

To write using File

```
File.WriteAllText(filePath, content);
```

To write using StreamWriter

```
using (StreamWriter writer = new StreamWriter(filePath))
{
    writer.WriteLine(content);
}
```

To append using File

```
File.AppendAllText(filePath, content);
```

To append using StreamWriter

```
using (StreamWriter writer = new StreamWriter(filePath, append: true))
{
    writer.WriteLine(content);
}
```

To append using FileStream

```
using (FileStream fs = new FileStream(filePath, FileMode.Append, FileAcce
ss.Write))
{
    byte[] buffer = System.Text.Encoding.UTF8.GetBytes(content);
```

```
    fs.Write(buffer, 0, buffer.Length);
}
```

Other method

```
File.Delete(path);
File.Exists(path);
File.Copy(source, dest);
```

ADO.NET

Used to connect with database and perform operations

To use MySql add package MySql.Data;

To create connection

```
MySqlConnection connection = new MySqlConnection(_connectionString);
```

To create command

```
MySqlCommand command = new MySqlCommand(query, connection);
```

To read data using DataReader

```
string query = "SELECT T02F02, T02F03, T01F02 FROM T01 INNER JOIN T
02 ON T01F01 = T02F08;";

MySqlCommand command = new MySqlCommand(query, connection);

connection.Open();
```

```
MySqlDataReader reader = command.ExecuteReader();

if (reader.HasRows)
{
    while (reader.Read())
    {
        Console.WriteLine($"{reader.GetString(0)} {reader.GetString(1)} {read
er.GetString(2)}");
    }
}
else
{
    Console.WriteLine("No rows found.");
}
```

To create DataAdapter

```
MySqlDataAdapter adapter = new MySqlDataAdapter(query, connection);
```

To fill DataTable from adapter

```
 adapter.Fill(table);
```

To update Database from DataTable

```
adapter.Update(table);
```

To use parameter write query with @variable_name

```
string query = "INSERT INTO T01 (T01F01, T01F02, T01F03) VALUES (@id,
```

```
@name, @location);";
```

To add value of variable

```
command.Parameters.AddWithValue("@id", 6);
```

To ExecuteNonQuery

```
int rowsAffected = command.ExecuteNonQuery();
```

To call procedure set

```
command.CommandType = CommandType.StoredProcedure;
```

and provide value of parameter using
command.Parameter.AddWithValue(variable_name, value);

If procedure return table use DataReader with ExecuteReader()

If procedure perform insert, update, delete use ExecuteNonQuery()

To use Out parameter of procedure

```
command.Parameters.Add("@p_count", MySqlDbType.Int32);
command.Parameters["@p_count"].Direction = ParameterDirection.Output;
int empCount = Convert.ToInt32(command.Parameters["@p_count"].Valu
e);
```