C# Advance

C# and .Net Web API

Yash Lathiya

Table of Contents

Types of Class

There are 4 types of classes in .net

- Abstract Class

This class contains method with the only signature & not body.

- Sealed Class

This class doesn’t support inheritance.

- Partial Class

When class is too large then multiple different functionalities of class can be written / implemented separately but at compile time it will compile into once.

- Static Class

Static class is a global class.

It will not allow to create object.

Directly we can use method of the static class.

Generics

What is difference between generic & non generic ?

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
namespace GenericsExample {
    class Program {
      static void Main(string[] args) {
        //generic list
        List<int> ListGeneric = new List<int> { 5, 9, 1, 4 };
        //non-generic list
        ArrayList ListNonGeneric = new ArrayList { 5, 9, 1, 4 };
        // timer for generic list sort
        Stopwatch s = Stopwatch.StartNew();
        ListGeneric.Sort();
        s.Stop();
        Console.WriteLine($"Generic Sort: {ListGeneric}    \n Time taken:
{s.Elapsed.TotalMilliseconds}ms");

        //timer for non-generic list sort
        Stopwatch s2 = Stopwatch.StartNew();
        ListNonGeneric.Sort();
        s2.Stop();
        Console.WriteLine($"Non-Generic Sort: {ListNonGeneric}  \n Time taken:
{s2.Elapsed.TotalMilliseconds}ms");
        Console.ReadLine();
      }
    }
  }
```

Here, Generic list will be sorted faster than non generic as it is specified datatype in generic while in non generic list, each input is taken as object and then casted to relevant memory. It takes much time.

```
public class GenericClass<T>
{

    public GenericClass(T information)
    {
        System.Diagnostics.Debug.WriteLine(information);
    }

}
```

Here, Generic class is created. Which contains generic method. Any datatype can be used as here to define class, we can add generic method too.

```
public class GenericMethod
{
    public T GenericMethodOfClass<T>(T information)
    {
        return information;
    }
}
```

Here generic method is added where generic class is not created.

Generic controller can be also implemented, but it’s not a good programming practise to create it.

File System

File system is a category in which developer handles all file operations, File operations include operation of

- File Uploading
  `file.SaveAs(filePath)`

- File Downloading

- Creating File

  `File.WriteAllText(filePath, objFil01.l01f02)`

- Override content of file,
  `File.WriteAllText(filePath, objFil01.l01f02)`

- Reading File &
  `File.ReadAllText(filePath)`

- Deletion of the file.
  `File.Delete(filePath)`

`HttpContext.Current.Server.MapPath("~/DirectoryName")` – We can use current context of path by using this method.

To file download, we need to use IhttpResponse

```
HttpResponseMessage response = new HttpResponseMessage(HttpStatusCode.OK);
                response.Content = new ByteArrayContent(fileBytes);
                response.Content.Headers.ContentDisposition = new
System.Net.Http.Headers.ContentDispositionHeaderValue("attachment")
                {
                    FileName = fileName
                };
                 response.Content.Headers.ContentType = new
 System.Net.Http.Headers.MediaTypeHeaderValue("application/octet-stream")
```

Serialization & Deserialization

Serialization:

Serialization involves converting an object or data structure into a format that can be easily transmitted over a network or stored in a persistent storage medium like a file or a database.

This process is necessary when you need to transfer an object's state between different components of a distributed system or when persisting data for later use.

During serialization, the object's properties or fields are converted into a linear stream of bytes or a text-based format, which can then be sent or stored.

Deserialization:

Deserialization is the reverse process of serialization.

It involves taking a serialized data stream and reconstructing the original object or data structure from it.

This is crucial when receiving data over a network or retrieving data from a storage medium.

The deserialization process interprets the serialized data, extracts the relevant information, and recreates the object with its original state.

Base Class Library

In the .NET Framework, the Base Class Library (BCL) is a fundamental component that provides a comprehensive set of classes, types, and functions to support the development of a wide range of applications.

The BCL is part of the .NET Framework's core functionality and serves as the foundation for building applications across various domains, including web development, desktop applications, and more.

BCL includes following features

- Common Types and Collections:

The BCL includes fundamental types such as strings, integers, floating-point numbers, and collections like arrays, lists, dictionaries, and queues.

- Input/Output Operations:

The BCL facilitates input and output operations with classes for file handling & stream manipulation. Developers can read from and write to files, streams, and other data sources using the provided classes.

- Networking:

Networking functionality is supported in the BCL, allowing developers to create networked applications. Classes for working with sockets, protocols like HTTP, and other networking components are part of the library.

- Security:

Security-related features, such as cryptography and secure communication, are included in the BCL.

Developers can leverage classes for encryption, digital signatures, and secure communication protocols to enhance the security of their applications.

- Exception Handling:

Exception handling is a crucial aspect of robust software development, and the BCL includes classes for working with exceptions. Developers can catch and handle exceptions using classes like Exception and related types.

- Asynchronous Programming:

With the introduction of asynchronous programming patterns, the BCL includes classes to support asynchronous operations. This is crucial for developing responsive and scalable applications.

We can create our own base class library, & use it in another project by adding project reference to current project. And adding namespace of base library in current project, It’ll adapt all methods and features of base library.

Use of Lambda

Lambda Statements

```
lstMov01.ForEach(movie =>
           {
               if (movie.v01f04 == genre)
               {
                   lstMov01Genre.Add(movie.v01f01);
               }
            });
```

Allows to write block of statements

Lambda Expression

```
MOV01 objMov01 =
lstMov01.FirstOrDefault( movie => movie.v01f01 == movieName );
```

Selects items by lambda expression
(single line selecting condition)

Extension Method

Extension methods in C# provide a way to add new methods to existing types without modifying their source code. They enhance the expressiveness and readability of code by allowing developers to call custom methods on objects of types that they didn't originally define.

```
public static int
Count(this List<Mob01> lstMob01, string companyName)
   {
       int count = 0;

       foreach(Mob01 objMob01 in lstMob01)
       {
           if(
                objMob01.b01f03?.ToLower() ==
                companyName.ToLower()
           )
           {
               count++;
           }
       }

       return count;

   }
```

LINQ

LINQ, or Language Integrated Query, is a set of features in the Microsoft .NET framework that provides a unified and expressive way to query and manipulate data from various sources.

LINQ allows developers to use a consistent syntax for querying data, regardless of whether the data is stored in databases, XML files, collections, or other data sources.

Key components of LINQ include:

- Query Expressions:

LINQ introduces a declarative syntax that allows developers to write queries using familiar keywords such as from, where, select, group by, and order by. These queries resemble SQL statements and are used to filter, project, and sort data.

- Standard Query Operators:

LINQ provides a set of standard query operators, such as Where, Select, OrderBy, and GroupBy, which can be used to perform common operations on data collections. These operators are applicable to a wide range of data sources, promoting code reuse and consistency.

- Deferred Execution:

LINQ queries are lazily executed, meaning that the actual execution of the query is deferred until the results are explicitly requested.

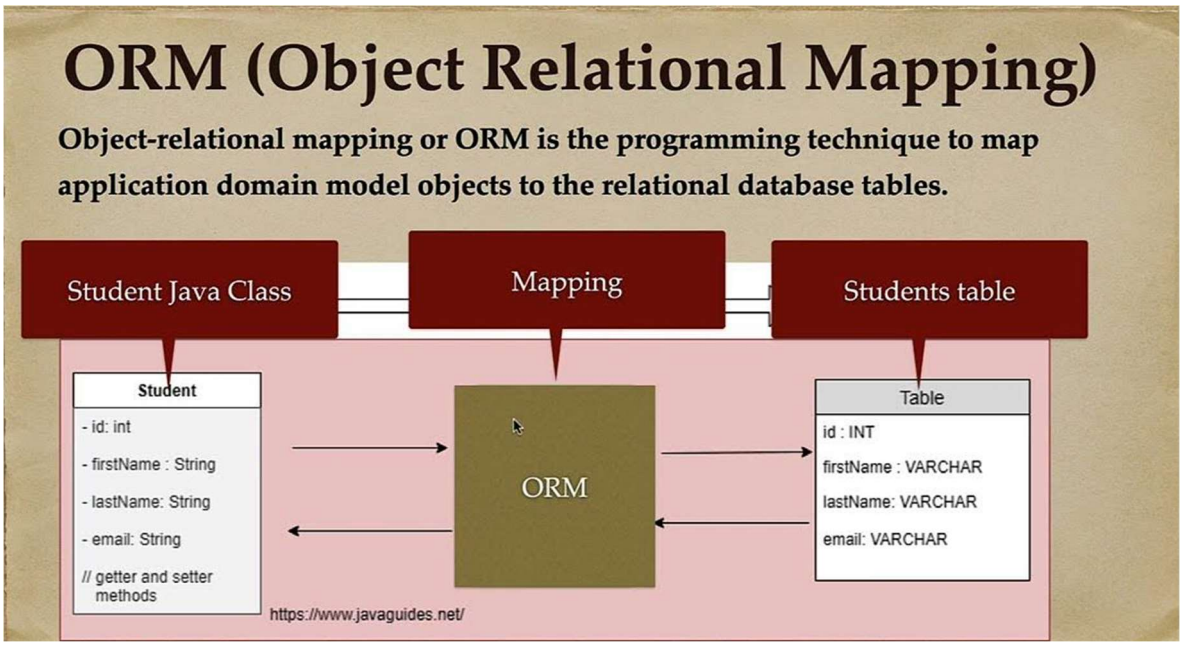
This allows for more efficient processing, especially when dealing with large datasets.

- Lambda Expressions:

LINQ heavily utilizes lambda expressions, which are concise, inline functions.

```
// Specify the data source.

int[] scores = { 97, 92, 81, 60 };

// Define the query expression.

IEnumerable<int> scoreQuery =
    from score in scores
    where score > 80
    select score;

// Execute the query.

foreach (var i in scoreQuery)
{
    Console.Write(i + " ");
}
```

ORM



To configure ORM (Object Relational Mapping) in .net web api, I have followed steps which are entitled below :

- Added Connection String in web.config file

```
<connectionStrings>

                <add
                name="MySqlConnection"
                connectionString="Server=127.0.0.1;

                        Port=3306;Database=employee_yashl;
                             User Id=Admin;
                             Password=gs@123;"
                providerName="MySql.Data.MySqlClient" />

</connectionStrings>
```

- In global.asax.cs file, used ConfigureOrmLite() method, implementation of that method is displayed below :

```
/// <summary>
/// Configures orm lite to connect mySql
/// </summary>

private void ConfigureOrmLite()
{

// Retrieve the connection string from the configuration file
(Web.config)

var connectionString =
System.Configuration.ConfigurationManager
.ConnectionStrings["MySqlConnection"]
.ConnectionString;


// Create an instance of OrmLiteConnectionFactory using the
retrieved connection string

var dbFactory = new OrmLiteConnectionFactory
                (connectionString, MySqlDialect.Provider);


// Set the default dialect provider for OrmLite to MySQL

OrmLiteConfig.DialectProvider = MySqlDialect.Provider;

// Store the IDbConnectionFactory instance in a static variable
for application-wide usage

MyAppDbConnectionFactory.Instance = dbFactory;

}
```

Here, MyAppDbConnectionFactory is a static class which is defined by me to configure DbFactory configuration globally.

Instance is static variable which is assigned instance of dbFactory.

Now, I am able to use ORM actually,

- Add alias name in object model, which is then name of respected table in MySql Database.

```
using ServiceStack.DataAnnotations;

namespace _9_ORM_Tool.Models
{

    [Alias("EMP01")]
    public class Emp01
    {

        public int p01f01 { get; set; }

        public string p01f02 { get; set; }

        public string p01f03 { get; set; }

        public int p01f04 { get; set;}

    }
 }
```

Here, we can also add attribute like primary key, unique key, Auto Increment etc .

(P.T.O.)

- I can manipulate database now as model is configured with class. I've created Employee Service class which is responsible for all operation.

```
using _9_ORM_Tool.Models;
using ServiceStack.Data;
using System;
using ServiceStack.OrmLite;

namespace _9_ORM_Tool.Connection
{
    public class EmployeeService
    {

        private readonly IDbConnectionFactory dbFactory;

        public EmployeeService(IDbConnectionFactory dbFactory)
        {
             this.dbFactory = dbFactory ?? throw new
             ArgumentNullException(nameof(dbFactory));
        }

        public void AddEmployee(Emp01 objEmp01)
        {
            using (var db = dbFactory.OpenDbConnection())
            {
                db.Insert(objEmp01);
            }
        }

        public Emp01 GetEmployeeById(int p01f01)
        {
            using (var db = dbFactory.OpenDbConnection())
            {
                return db.SingleById<Emp01>(p01f01);
            }
        }

        public void UpdateEmployee(Emp01 objEmp01)
        {
            using (var db = dbFactory.OpenDbConnection())
            {
                db.Update(objEmp01);
            }
        }

        public void DeleteEmployee(int p01f01)
        {
            using (var db = dbFactory.OpenDbConnection())
            {
                db.DeleteById<Emp01>(p01f01);
            }
        }

    }
}
```

Security & Cryptography

Security is all about authentication & authorization. We can use Cryptographic algorithms to ensure stable and efficient security.

There are algorithms like AES, DES, Triple DES, RSA etc.

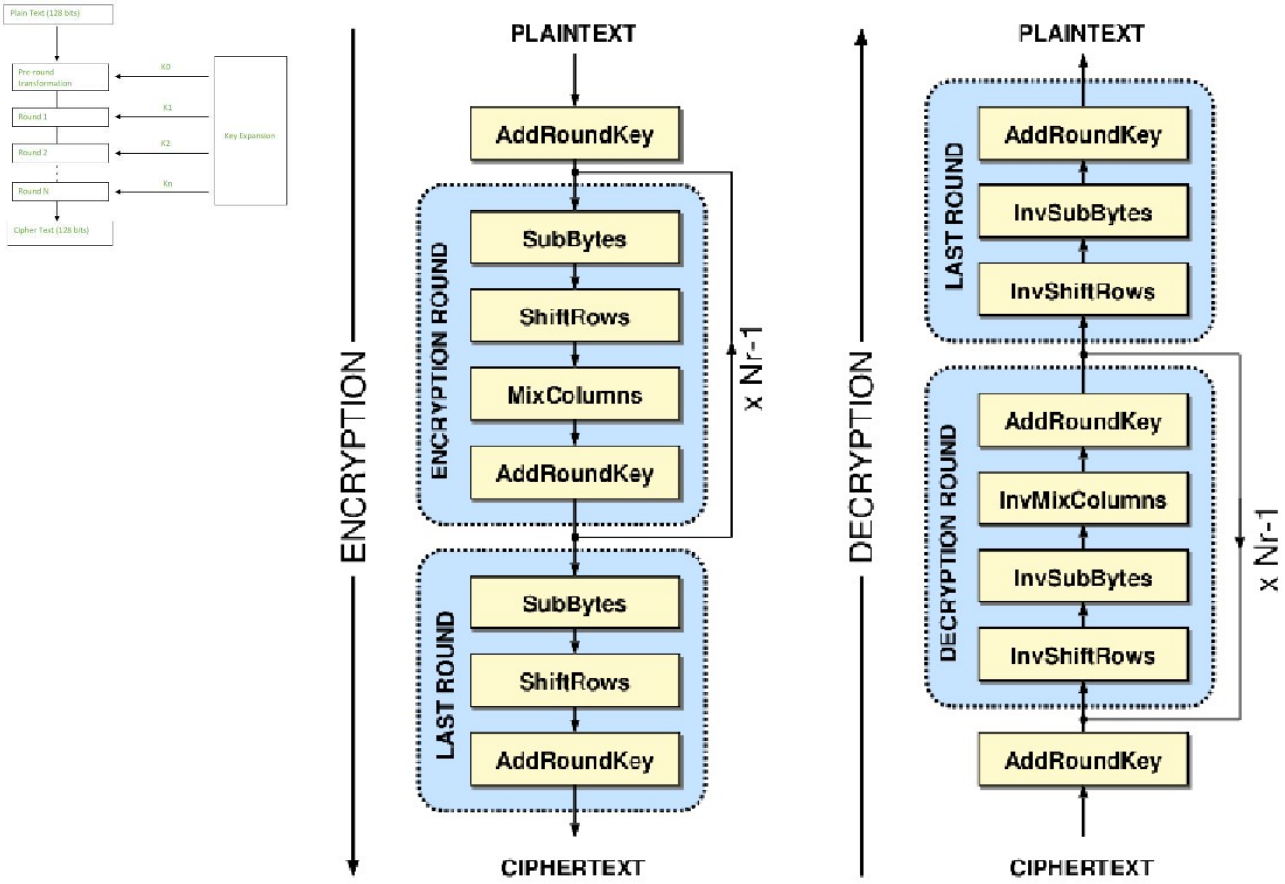
AES algorithm is used for advanced security requirements like banking transactions.

DES algorithm is generally not used as it’s not as secure it should be.

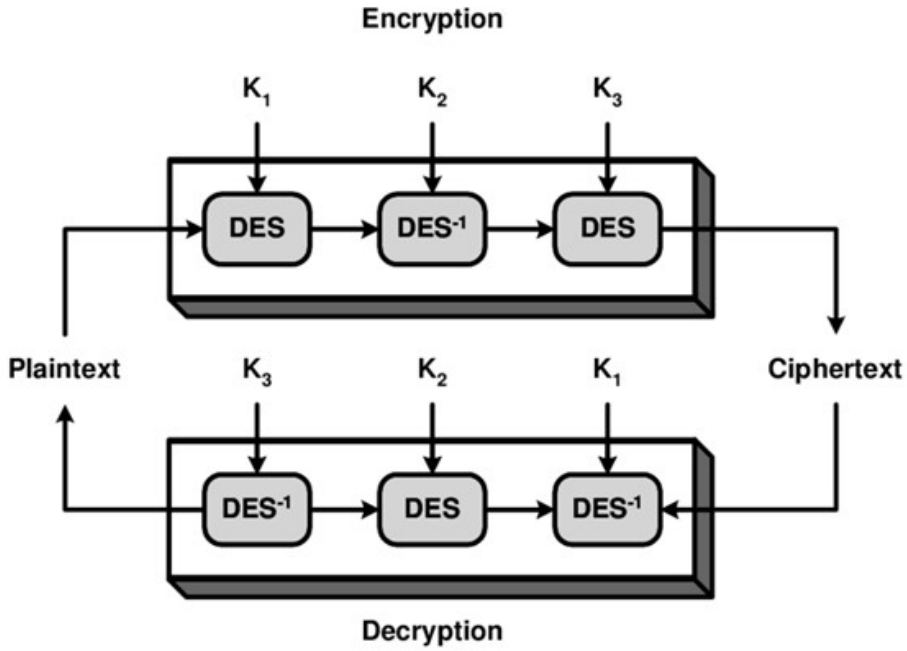
Triple DES is approach is used to fulfil security reqirements.

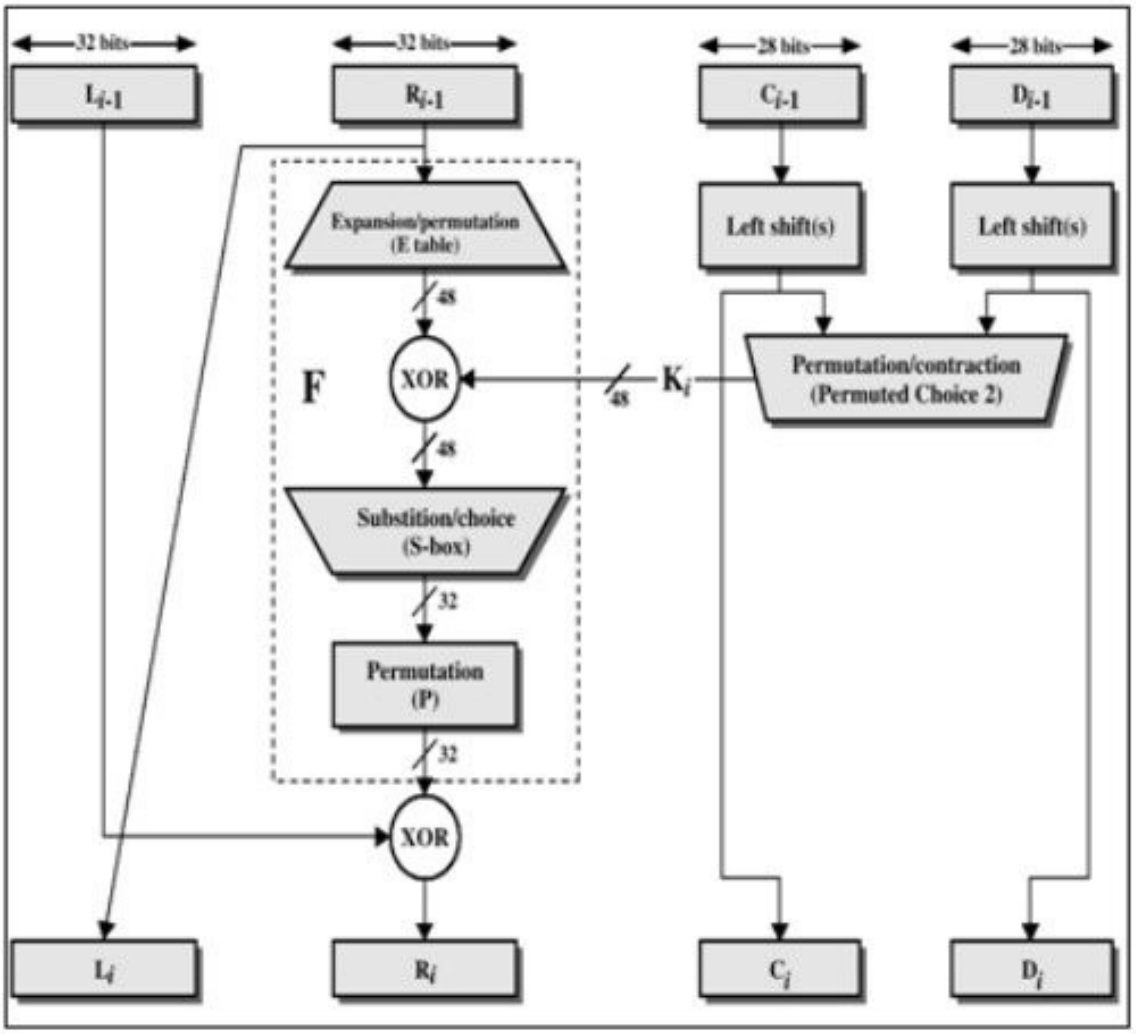
RSA is generally used for Digital Signature requirements.

AES

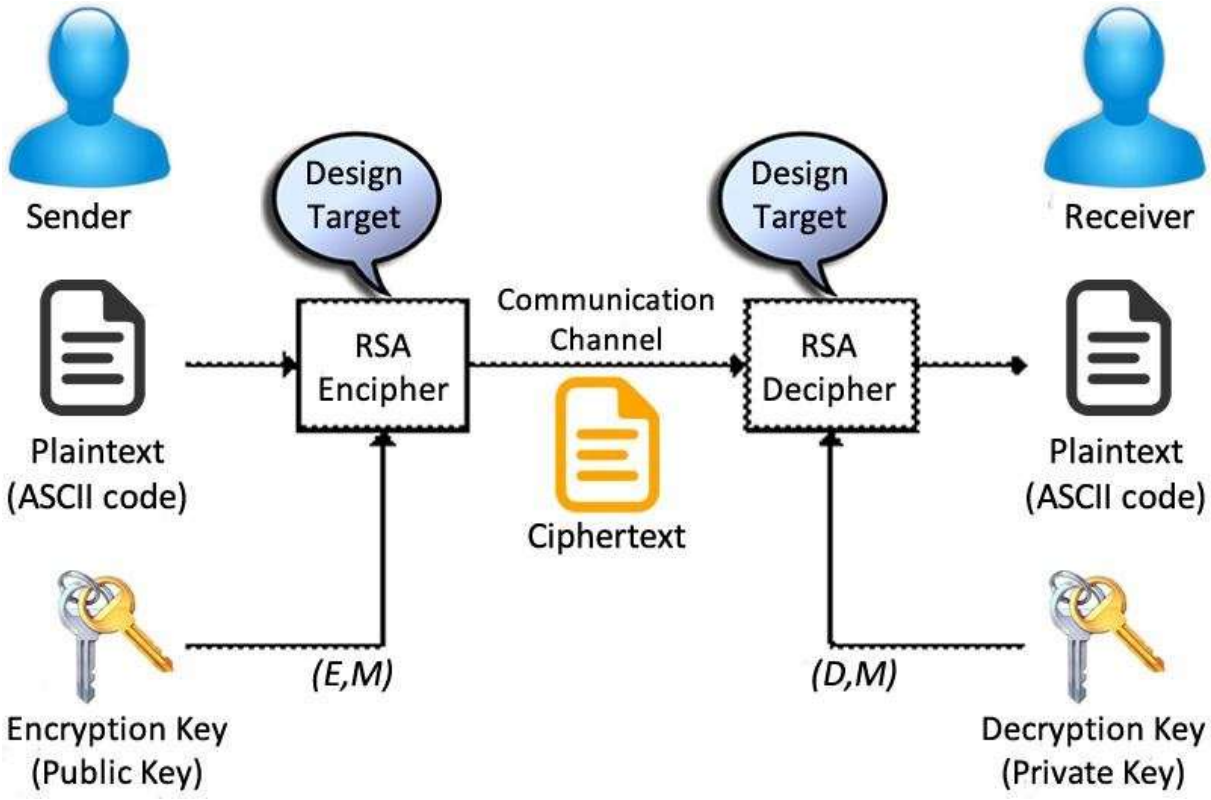


Triple DES


Encryption
K1
K2
K3
DES
DES-1
DES
Plaintext
K3
K2
K1
Ciphertext
DES-1
DES
DES-1
Decryption


32 bits
32 bits
28 bits
28 bits
Li-1
Ri-1
Ci-1
Di-1
Expansion/permutation
(E table)
Left shift(s)
Left shift(s)
48
F
XOR
48
Ki
Permutation/contraction
(Permuted Choice 2)
48
Substition/choice
(S-box)
32
Permutation
(P)
32
XOR
Li
Ri
Ci
Di

RSA



Dynamic

dynamic data type enables dynamic typing, allowing for late binding and type determination at runtime.

It supports interoperability with dynamic languages and scenarios where types are not known until runtime.

Dynamic variables lack compile-time type checking, providing flexibility but with the potential for runtime errors.

We should use dynamic very carefully, as it bypasses compile-time checks.

Database with C# (CRUD)

Here, We deal with database without ORM. We are witing queries to retrieve and insertion of data.

I've followed steps to use MySql database with my .net web api.

- I've created ConnectionStrings.json file which consists all Connection string within the api.

```
{
      "ConnectionStrings": {
            "name": "MySqlConnection",
            "connectionString": {
                  "Server": "127.0.0.1",
                  "Port": 3306,
                  "Database": "employee_yashl",
                  "User Id": "Admin",
                  "Passoword": "gs@123"
            }
      }
}
```

- I've created Connection class which configures MySql connection with the web api.

```
using System;
using System.IO;
using MySql.Data.MySqlClient;
using Newtonsoft.Json;


namespace _12_Database_With_CRUD.Connection
{
    public static class Connection
    {
        static MySqlConnection _mySqlConnection;

        static Connection()
        {
            try
            {
                // Extracting Connection String

                string jsonFilePath = @"C:\Users\yash.l\source\repos\Yash-
Lathiya\Demo\API Basic
Demo\5_C#_Advance\C#Advance\12_Database_With_CRUD\Connection\ConnectionString.jso
n";
                var connectionStrings = File.ReadAllText(jsonFilePath);

                dynamic connectionStringObject =
JsonConvert.DeserializeObject(connectionStrings);
                string server =
connectionStringObject.ConnectionStrings.connectionString.Server;
```

```
                int port =
connectionStringObject.ConnectionStrings.connectionString.Port;
                string database =
connectionStringObject.ConnectionStrings.connectionString.Database;
                string userId =
connectionStringObject.ConnectionStrings.connectionString["User Id"];
                string password =
connectionStringObject.ConnectionStrings.connectionString.Passoword;

                string mysqlConnectionString =
$"server={server};port={port};database={database};user={userId};password={passwor
d};";

                // Providing Connection string to mySqlConnection
                _mySqlConnection = new MySqlConnection(mysqlConnectionString);
                //_mySqlConnection.Open();

            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }

        }


        public static MySqlConnection GetMySqlConnection()
        {
            return _mySqlConnection;
        }
    }
}
```

- Now, I can use GetMySqlConnection() method whenever it is needed to go for connection. I've created EmployeeManager class which has business logic of all CRUD operation.

```
using _12_Database_With_CRUD.Models;
using MySql.Data.MySqlClient;
using System;

namespace _12_Database_With_CRUD.BL
{

    public static class EmployeeManager
    {
        private static MySqlConnection _mySqlConnection;

        static EmployeeManager()
        {
            try
            {
                // Get the MySqlConnection instance from the Connection class
                _mySqlConnection = Connection.Connection.GetMySqlConnection();
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }
        }
```

```
        public static void AddEmployee(Emp01 objEmp01)
        {
            using (MySqlCommand command = new MySqlCommand())
            {
                command.Connection = _mySqlConnection;
                command.CommandText = @"INSERT INTO
                                            EMP01
                                            (p01f01, p01f02, p01f03, p01f04)

                                             VALUES (@p01f01, @p01f02, @p01f03,
                                             @p01f04)";

                command.Parameters.AddWithValue("@p01f01", objEmp01.p01f01);
                command.Parameters.AddWithValue("@p01f02", objEmp01.p01f02);
                command.Parameters.AddWithValue("@p01f03", objEmp01.p01f03);
                command.Parameters.AddWithValue("@p01f04", objEmp01.p01f04);

                try
                {
                    _mySqlConnection.Open();
                    command.ExecuteNonQuery();
                }
                catch(Exception ex)
                {
                    throw new Exception(ex.Message);
                }
                finally
                {
                    _mySqlConnection.Close();
                }
            }
        }

        public static Emp01 GetEmployee(int p01f01)
        {
            using (MySqlCommand command = new MySqlCommand())
            {
                command.Connection = _mySqlConnection;

                command.CommandText = @"SELECT
                                            p01f01,
                                            p01f02,
                                            p01f03,
                                            p01f04
                                        FROM
                                            EMP01
                                        WHERE
                                            p01f01 = @p01f01";

                command.Parameters.AddWithValue("@p01f01", p01f01);

                try
                {
                    _mySqlConnection.Open();

                    using (MySqlDataReader reader = command.ExecuteReader())
                    {
                        if (reader.Read())
                        {
                            // Create and return an Emp01 object based on the
data from the database
                            Emp01 employee = new Emp01
```

```
                        {
                            // Map database columns to Emp01 properties
                            p01f01 = Convert.ToInt32(reader["p01f01"]),
                            p01f02 = Convert.ToString(reader["p01f02"]),
                            p01f03 = Convert.ToString(reader["p01f03"]),
                            p01f04 = Convert.ToInt32(reader["p01f04"]),
                        };

                        return employee;
                    }
                }
            }
            catch(Exception ex)
            {
                throw new Exception(ex.Message);
            }
            finally
            {
                _mySqlConnection.Close();
            }

            return null;
        }
    }
    public static void UpdateEmployee(Emp01 objEmp01)
    {
        using (MySqlCommand command = new MySqlCommand())
        {
            command.Connection = _mySqlConnection;

            command.CommandText = @"UPDATE
                                        EMP01
                                    SET
                                        p01f02 = @p01f02,
                                        p01f03 = @p01f03,
                                        p01f04 = @p01f04
                                    WHERE
                                        p01f01 = @p01f01";

            // Assuming the actual column names and values need to be
replaced in the query
            command.Parameters.AddWithValue("@p01f01", objEmp01.p01f01);
            command.Parameters.AddWithValue("@p01f02", objEmp01.p01f02);
            command.Parameters.AddWithValue("@p01f03", objEmp01.p01f03);
            command.Parameters.AddWithValue("@p01f04", objEmp01.p01f04);

            try
            {
                _mySqlConnection.Open();
                command.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }
            finally
            {
                _mySqlConnection.Close();
            }

        }
    }
```

```
        public static void DeleteEmployee(int p01f01)
        {
            using (MySqlCommand command = new MySqlCommand())
            {
                command.Connection = _mySqlConnection;

                command.CommandText = @"DELETE FROM
                                            EMP01
                                        WHERE
                                            p01f01 = @p01f01";

                command.Parameters.AddWithValue("@p01f01", p01f01);

                try
                {
                    _mySqlConnection.Open();
                    command.ExecuteNonQuery();
                }
                catch (Exception ex)
                {
                    throw new Exception(ex.Message);
                }
                finally
                {
                    _mySqlConnection.Close();
                }
            }
        }
    }
}
```

All detailed demos are published in github > Basic API > C# Advance