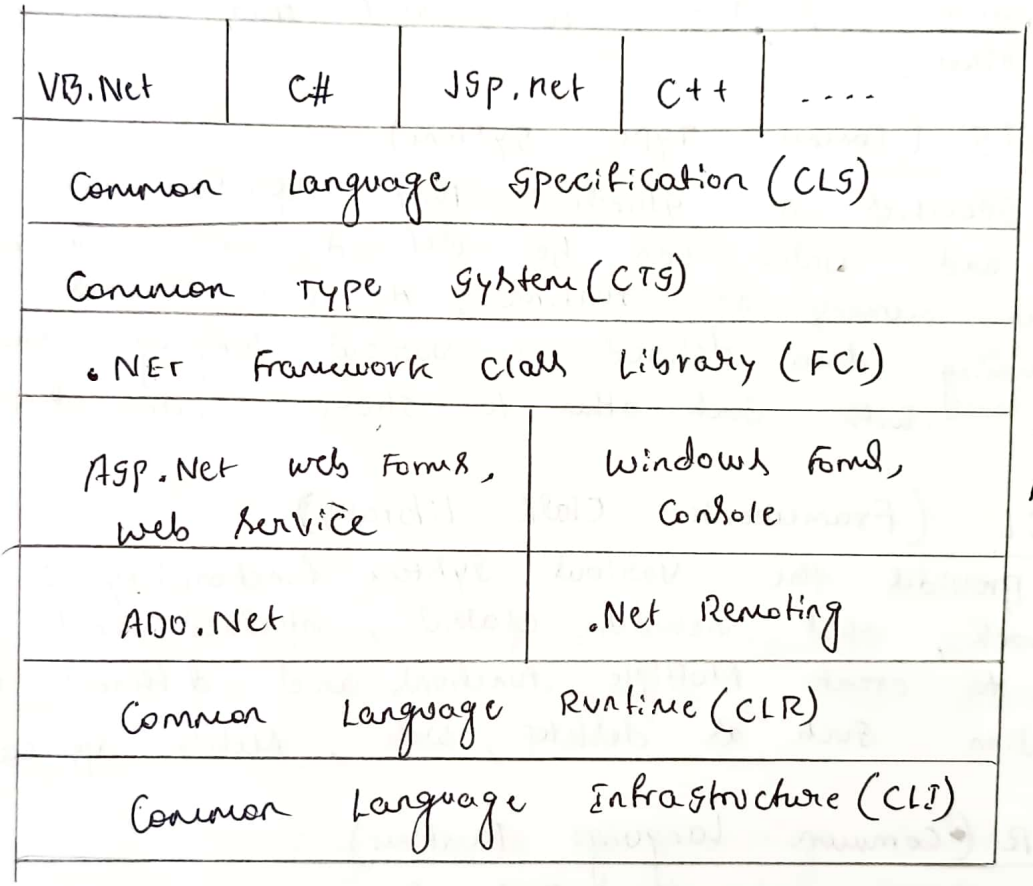
Assignment - 1

Advance programming

(1) With a diagram, explain the working of .Net framework 4.0.

:



.NET Framework Architecture

The .NET Framework is a software framework developed by Microsoft that provided a runtime environment for executing and managing applications. Version 4.0 introduced several enhancements and features.

There are following components of .NET framework

1. CLS (Common Language Specification)

It is a subset of common type system that defines a set of rules and regulations which should be followed by every language that comes under the .net framework.

In other words, a CLS language should be Cross-language integration or interoperability. For Example, in C# and VB.NET Language, the C# language terminate each Statement with SemiColon, where as in VB.NET it is not end with SemiColon, and when these Statement executes in .NET Framework, it provid a common platform to interact and share info. with each other.

2. CTS (Common Type System)

It Specifies a standart that represent what type of data and value can be defined and managed in Computer memory at runtime. A CTS ensures that Programming data defined in various language should be interact with each other to share information.

3. FCL (Framework Class Library)

It provides the various System functionality in the .NET Framework, that includs classes, interfaces and dat types etc. to create Multiple functions and different types of application such as desktop, Web, Mobile application etc.

4. CLR (Common Language Runtime)

It is a important part of a .NET Framework that works like a virtual Component of the .NET Framework to executes the different language program like C#, VB etc A CLR also helps to Convert a Source Code into the byte Code, and this byte Code is known as CIL (Common Intermediate Language) or MSIL. After Converting into a byte Code, a CLR used a JIT Compiler at run time that helps to Convert a CIL or MSIL Code into the machine or native Code.

5. Common Language Infrastructure

The language-independent nature of the CLI makes it easier to transition your app across different platforms. The CLI is an app development framework that is language-independent. Therefore, developers don't have to worry about changing the language or syntax of their source code when switching from one platform to another.

6. Microsoft .NET Assemblies

A .NET assembly is the main building block of the .NET framework. It is a small unit of code that contains a logical compiled code in the common language infrastructure, which is used for deployment, security and versioning.

(2) Explain the exception handling concept with all key words and program example. And discuss the usage of data types.

∴ Exception handling is a programming construct that allows developers to manage and respond to errors or exceptional situations that may occur during the execution of a program.

In C#, exception handling is implemented using the 'try', 'catch', 'finally' and 'throw' keywords.

1. try - The 'try' block contains the code that might raise an exception. It is followed by one or more 'catch' blocks. Used to define a try block.

2. Catch - Used to define a catch block. It is used to catch and handle a specific type of exception that might occur in the associated try block. Multiple 'catch' blocks can be used to handle different types of exceptions.

3. finally - The 'finally' block contains code that will be executed whether an exception is thrown or not. It is optional, but if used, it ensures that certain cleanup or resource release operations are performed.

4. Throw - The 'throw' keyword is used to explicitly throw an exception. This can be done in response to a specific condition that the program detects.

Ex :-

```
Using System;
Class ExceptionHandle.
{
  Static void Main()
  {
    try
    {
      Console.Write(" Enter a number");
      String input = Console.ReadLine();
      int number = int.parse(input);
      int result = 10/number;
      Console.WriteLine(" Result " + Result);
    }
    Catch (FormatException ex)
    {
      Console.WriteLine(" Error ; Please enter a valid number");
    }
    Catch (Exception e)
    {
      Console.WriteLine(" unexpected
    }
```

```
Catch (Exception e)
Catch (DivideByZeroException ex)
{
  Console.WriteLine ("Error ; Can't divide");
}
finally
{
  Console.WriteLine ("Code executed");
}
}
```

Data type

In C#, data types are used to define the type of data that a variable can store. Data types specify the size and layout of the variable's memory, as well as the range of values it can hold. C# provides a rich set of built-in data types that can be broadly categorized into the following:

1. Value Type
   - Integral types
     - byte : 8-bit unsigned integer.
     - sbyte : 8-bit signed integer.
     - short : 16-bit signed integer.
     - ushort : 16-bit unsigned integer.
     - int : 32-int signed integer.
     - long : 64-int signed integer.
   - Floating point types
     - float : 32-bit single-precision floating-point.
     - double : 64-bit double-precision floating-point.
     - decimal : 128-bit decimal.
   - Other value types
     - char : 16-bit Unicode character.
     - bool : Boolean (true/false)
     - struct : User-defined value type.

2. Reference Types.

- Object - The Ultimate base class for all types in C#.
- String - A sequence of characters.
- delegates - Reference to a method.
- array - A Collection of elements of the same type.

(3) Explain the following

(i) Sealed and runtime polymorphism

In C#, the 'sealed' keyword is used to prevent a class from being inherited, when a class is marked as sealed, it cannot be used as a base class for other classes.

Ex:-

```
Using System;
Sealed class BaseClass
{
    Public virtual void Display()
    {
        Console.WriteLine (" Base class Display ");
    }
}
class Program
{
    Static void Main()
    {
        BaseClass obj = new BaseClass();
        obj.Display();
    }
}
```

In this example, the "BaseClass" is marked as 'sealed' and attempting to inherit from it will result in a compilation error.

Runtime polymorphism in C# is achieved through method overriding. It allows a base class reference variable to refer to objects of its derived classes, and the method that gets executed is determined at runtime.

Ex :

```
Using System;
Class Animal
{
    public virtual void Sound()
    {
        Console.WriteLine("Animal makes a Sound");
    }
}
Class Dog : Animal
{
    public override void Sound()
    {
        Console.WriteLine("Dog barks");
    }
}
Class Cat : Animal
{
    public override void Sound()
    {
        Console.WriteLine("Cat meows");
    }
}
Class program
{
    static void Main()
    {
        Animal myAnimal;
        myAnimal = new Dog();
        myAnimal.Sound();
        myAnimal = new Cat();
        myAnimal.Sound();
    }
}
```

(11) Accessors and Mutators

Accessors

It often referred to as a "getter", is a method or property that allows you to retrieve the value of a private field. It provides read-only access to the underlying data. Accessors are declared using the 'get' keyword.

Mutators

A mutator, often referred to as a "setter", is a method or property that allows you to modify the value of a private field. It provides write-only access to underlying data. Mutators are declared as the 'set' keyword.

Ex :-

```
Using System;
Class person
{
    Private String name;
    Public String GetName()
    {
        return name;
    }
    public void SetName(String newName)
    {
        name = newName;
    }
}
Class program
{
    Static void Main()
    {
        Person person = new Person();
        String cur = Person.GetName();
        Console.WriteLine("Current Name:" + cur);
        Person.SetName("John Doe");
        String Update = person.GetName();
        Console.WriteLine("Updated Name:" + Update);
    }
}
```

(iii) WPF

WPF or windows presentation Foundation, is a graphical subsystem for creating rich and interactive user interface in widows - based applications. It is a part of the .NET framework and provides a powerfull and flexible way to build modern, visually appealing desktop application. WPF was introduced by microsoft as a replacement for windows forms, offering a more sophisticated and feature-rich approach to UI development.

Here are some key features and concepts.

1. XAML 2. Dependency properties 3. Data Binding
4. MVVM 5. Styles and Templates. 6. Graphics and Animation.

→ overall, WPF is a versatile framework that empowers developers to create modern and visually compelling windows application.

(v) WCF

WCF, or windows Communication Foundation, is a framework developed by Microsoft for building distributed and interoperable service-oriented applications. It provides a unified programming model for developing and connecting various distributed systems and services. WCF supports various communication protocols and allows developers to build applications that can communicate across different platforms and technologies.

key features and concepts associated with WCF include:

1. Service-oriented Architecture (SOA) 2. Interoperability
3. Contracts 4. Endpoints 5. Bindings 6. Hostings.

(V) NameSpace

In C#, a namespace is a way to organize and group related code elements. It helps prevent naming conflicts and provides a logical structure to the code. A namespace can contain classes, interfaces, structures, interfaces, structures, enumerations and other namespaces. This organization is particularly important in large projects where different developers may be working on different parts of the applications.

Ex :-

```
Using System;
namespace MyNameSpace
{
  class MyClass
  {
    public void Display()
    {
      Console.WriteLine("Hellow from MyNameSpace");
    }
  }
}
class Program
{
  Static void Main()
  {
    MyNameSpace.MyClass myobj = new MyNameSpace.MyClass();
    myObj.Display();
  }
}
```

(VI) Boxing and UnBoxing

Boxing - It is the process of converting value type (eg 'int', 'char', 'double') to the object type ("System.object") which is reference.

* When a value is boxed, a new object is created on the heap, and the value of the value type is copied into that
* This process allows value types to be treated as objects, which is necessary in scenarios where a value type needs to be used in a context that expects an object

Unboxing - It is the process of converting an object type back to its original value type.

* It involves extracting the value from the boxed object and aligning it back to a value type variable.
* Unboxing is necessary when you want to retrieve the original value stored in an object after it has been boxed.

EX

```
Using System;
Class Program
{
    Static void Main()
    {
        int org = 40;
        object boxe = org;
        int unboxed = (int) boxe;
        Console.WriteLine(" original value " + org);
        Console.WriteLine(" Boxed value " + boxe);
        Console.WriteLine(" Unboxed value" + unboxed);
    }
}
```

(4) Discuss the importance of CLR, CLS and CTS in a .NET framework.

In .NET framework, The CLR, CLS and CTS play crucial roles in ensuring interoperability, consistency and robustness of application written in various languages.

1. Common Language Runtime (CLR)

- CLR is the execution environment provided by the .NET framework. It manages the execution of code written in different language, providing features like memory management, security handling & thread management.

- Language independence - CLR enables the execution of code written in multiple programming languages, allowing developers to choose the language that best suits their needs. It facilitates language interoperability by providing a common runtime for different language.

- Just-in-Time Compilation (JIT) - It uses Jit Compilation to convert Intermediate Language code into native machine code at runtime. It allows for platform independence, as the same IL code can run on any system with the CLR.

(2) Common Language Specification.

Language interoperability - CLS defines a set of rules and guidelines that language compilers must adhere to in order to achieve interoperability. By following CLS, developers can create components in one language that can be used by applications written in other CLS-compliant languages.

Component Reusability - CLS encouraged the creation of components that are reusable across different language. This promotes the developers of modular and extendible software solutions.

Reduced Development time - Developers can leverage components created in d/f languages, saving time and effort by reusing existing code.

3. Common Type System (CTS)

Data type interoperability - CTS defines a common set of data types that are supported by all CLS-compliant languages. This ensures that data types used in one language can be seamlessly used by another language, promoting data type interoperability.

Object interoperability - CTS also defines rules for creating and using objects, ensuring that objects created in one language can be used by other CLS-compliant language.

Cross-Language Inheritence - CTS supports inheritence across languages, allowing classes defined in one language to inherit from classes defined in other language.

(5) List out the benefits of .net framework. Explain them in details.

∴ * Language Interoperability - .NET support multiple programming language such as C#, VB.NET, F# and more. This enables developers to choose the language that best fits their requirements and expertise while allowing components written in d/f language to seamlessly work together.

* Flexible development - It is one of the unavailable benefit of the .NET framework. Flexible deployment is one of the most important .NET core capabilities. It may either be integrated into your program or deployed separately.

* Reliability and scalability — when it come to developing commercial apps, the .NET framework has shown very strong and trustworthy application development platform. With dot net you can create reliable and scalable applications.

* Simple Caching System .NET Core — It is one of the most efficient advantages of .NET framework is memory management, It occupy more than the expected space in the dot net development of applications.

* open-source framework — It is based on an open-source environment, making it extremely versatile and user-friendly. you can develop all types of applications for windows or desktop different OS. and devices of any configuration.

* Code Reusing — when building new S/W, web pages or .NET application, code reuse is described as reusing code that already exists, either inside your company or elsewhere.

(6) Write the Syntax of abstract class. Explain the characteristics of abstract class and abstract methods.

: in C# an abstract class is declared using the 'abstract' keyword.

Syntax

```
abstract class MyBaseClass   // Abstract class
{
    public abstract void MyAbstractMethod();   // Abstract method
    public void MyConcreteMethod() { }   // Concrete method
    public abstract int MyAbstractProperty { get; set; }   // Abstract property
    public int MyConcreteProperty { get; set; }
}
```

Characteristics of Abstract Class

1. Cannot be instantiated — abstract classes cannot be instantiated on their own. They are meant to be inherited by other classes.

2. May contain Abstract and Concrete Members — Abstract classes can have both abstract and concrete methods, as well as abstract and concrete properties.

3. Can Contain Constructors - Abstract classes can have Constructor and these Constructors are called when a derived class object is instantiated.

4. Can have fields and events - Abstract classes can contain fields, and other members typical of a class.

Characteristics of Abstract methods

1. No Implementation - Abstract method do not have an implementation in the abstract class. They are meant to be implemented by the derived classes.

2. Marked with the 'abstract' keyword - Abstract methods are declared using the 'abstract' keyword in the abstract class.

3. Cannot be private or static - Abstract methods cannot be private or static. They must have at least protected or public visibility.

(7) With a program example explain inheritence, encapsulation and interface

:- Inheritence - It is a fundamental object-oriented programming concept that allows a new class to inherit properties and behaviours from an existing class (base class). The derived class can extend or override the functionalities of the base class.

Encapsulation - It is the bundling of data and methods that operate on the data into a single unit known as a class. It helps in controlling access to the data by providing access modifiers and facilitated data handling.

Interface - An interface defines a contract that classes can implement. It contains method signatures, properties, events and indexers. Classes that implement an interface must provide concrete implementations for all the members defined by that interface.

Example

```
using System;
public interface IShape                    // Interface
{
    double CalculateArea();
}
public class Shape                         // Base class with inheritance
{
    protected double _width;
    protected double _height;
    public double Width                    // Encapsulation using properties.
    {
        get { return _width; }
        set { _width = value; }
    }
    public double Height
    {
        get { return _height; }
        set { _height = value; }
    }
}
public class Rectangle : Shape, IShape
{
    public Rectangle (double width, double height)
    {
        Width = width;
        Height = height;
    }
    public double CalculateArea()
    {
        return _width * _height;
    }
}
class Program
{
    static void Main()
    {
        Rectangle rectangle = new Rectangle (5, 10);
        Console.WriteLine ("{" width : {rectangle.Width}, Height : {rectangle.
                            Height}");
        double area = rectangle.CalculateArea();
        Console.WriteLine ("{" Area : {area}");
    }
}
```

(9) Define properties and interface. Demonstrate with a program example.

Properties - In C#, properties provide a way to encapsulate the internal state of a class and control access to its fields. They consist of a get accessor and a set accessor. properties are defined using the 'get' and 'set' keywords.

Interface - An interface in C# is a contract that defines a set of members that a class must implement if it chooses to implement that interface. An interface is declared using the 'interface' keyword.

Ex:

```
Using System;
Public interface ILogger          // Interface
{
  Void LogMessage (String message);
}
Public class ConsoleLogger : ILogger
{
  Public string LoggerName{get ; set ; }
  Public void LogMessage (String message)
  {
    Console.WriteLine ($" {LoggerName} : {message} ");
  }
}
Class program
{
  Static void Main()
  {
    ConsoleLogger logger = new ConsoleLogger();
    logger.LoggerName = "ConsoleLogger";
    logger.LogMessage (" This is a log message ");
  }
}
```

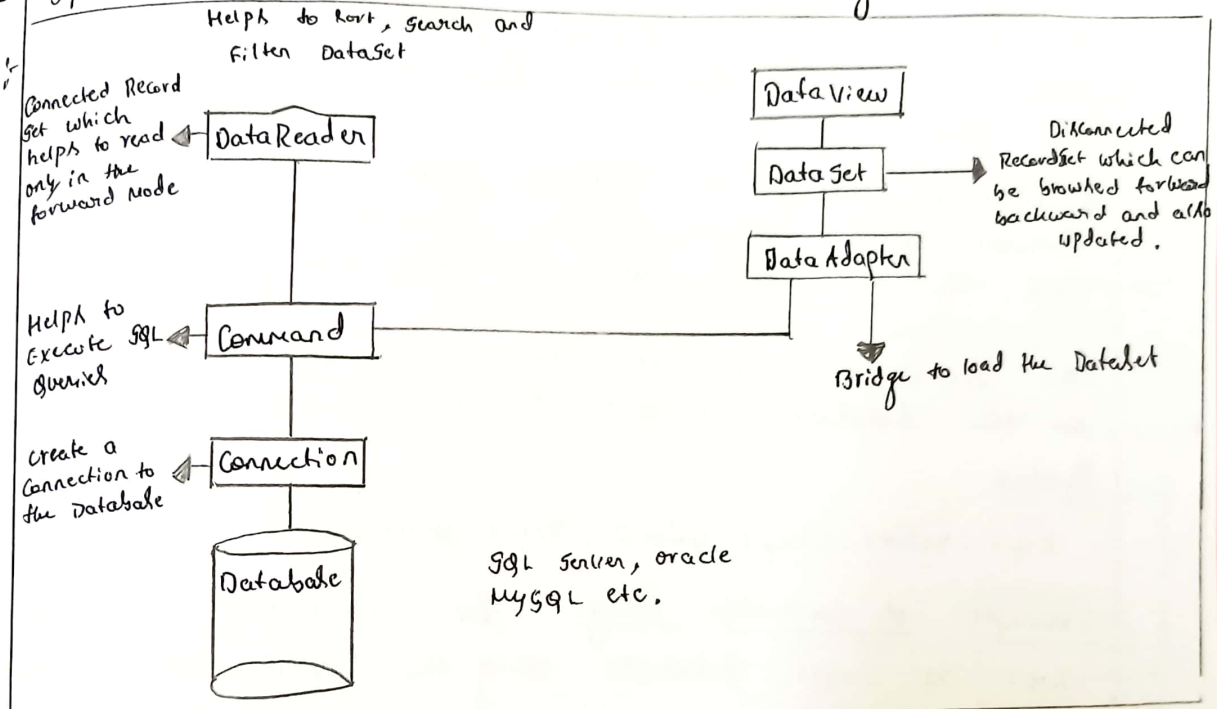
program example.

encapsulate
its fields.
properties
rds.

that defined a
it choosed to
d using the

18

(8) Explain the architecture of ADO.NET. Write the syntax and example for Connection string. 15



* Connection - The first important Component of ADO.NET Architecture is the Connection object. The Connection object is required to Connect with your backend database which Can be SQL Server, oracle, MySQL etc.

* Command - The Second important Component of ADO.NET Architecture is the Command object. When we talk about database like SQL Server, oracle, MySQL etc. We know one thing, these databases only understand SQL.

* Data Reader - It is a read-only Connection-oriented recordset that helps us to read the records only in the forward Mode. Here, you need to understand three things i.e read-only, connection-oriented and forward mode.

* DataSet - It is a Disconnected record set that Can be browsed in both i.e forward and backward mode. It is not read-only i.e you Can update the data present in the dataset.

* Data Adapter - The DataAdapter is one of the Component of ADO.NET which acts as a bridge between the Command object and dataset. What the ~~dataset~~ DataAdapter does it, it takes the data from the Command object and fills the data set.

* DataView Class - The DataView class enabled us to create different views of the data stored in a DataTable. This is most often used in data-binding applications.

The Syntax for a Connection String varies depending on the database or data source you are connecting to.

Syntax

key1 = value1 ; key2 = value2 ; key3 = value3 ;...

Example of Connection String for Connecting to a Micro Soft Sql Server database using the 'System.Data.SqlClient' provider.

Data Source = ServerName ; Initial Catalog = DatabaseName ;
User Id = UserName ; Password = SecretPassword ;

Ex:-

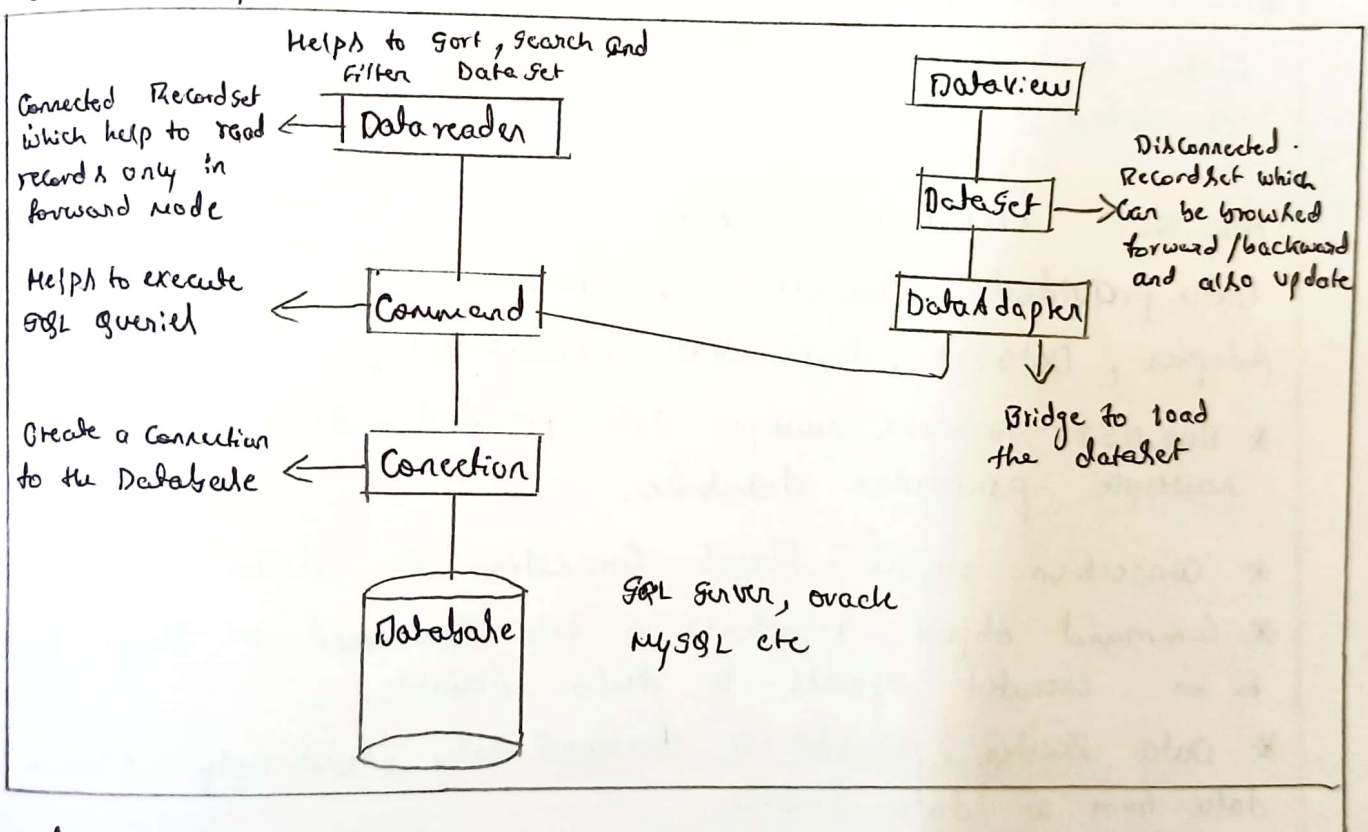
```
Using System;
Using System.Data.SqlClient;
Class Program
{ Static Void Main()
  {
    String con = " Data Source = ServerName ; Inial Catalog = DatabaseName
                 Userid = UserName ; Password = SecretPassword ;

    Using (SqlConnection Connect = new (SqlConnection(con))
    {
      try
      {
        Connect.open();
        Console.WriteLine (" connection open ");
        Connect.Close();
      }
```

```
catch (Exception e)
{
    Console.WriteLine ("Error: " + e.Message);
}
}
}
```

(10) Draw and explain the Architecture of ADO.NET

ADO.NET is a set of libraries in the .NET framework that provides a data access architecture for connecting to databases, retrieving, manipulating, updating data. The architecture of ADO.NET includes several key components and concepts.



ADO.NET Workflow

* Connect to the data source
  Establish the connection to the data source using a Connection object
* ~~to~~ Execute Command
  Use Command objects to execute SQL queries, Stored Procedures or other Commands against the data Sources.

* Retrieve data
  - Use a DataReader to efficiently retrieve and read data when a forward-only, read-only stream is sufficient.
  - Use a DataAdapter to fill a DataSet with data for more complex scenarios or disconnected scenarios.

* Work with data in memory
  Manipulate and analyze data in memory using DataTables, DataViews, and other data structures within a DataSet.

* Update data
  Use the DataAdapter to apply changes made to the in-memory DataSet back to the data source.

* Disconnect from the Data Source
  Close the connection when the data access operations are completed.

ADO.NET Architecture Contain
Data provider, Connection, Command, DataReader, Data Adapter, DataSet, DataTables, DataView.

* ADO.NET supports multiple data providers, each specific to particular database.
* Connection object shows Connection to data source.
* Command object represents a SQL Command or Stored procedure to be executed against a data source.
* Data Reader provides a forward-only, read-only stream of data from a data source.
* The DataAdapter acts as a bridge between a Dataset And DataSource.
* DataSet is an in-memory cache of data retrieved from a data source.
* DataTable represents a single table in memory.
* The DataView Provides a way to filter and sort data in the Data Table.