.NET Technology (PS02CDCA34)

Unit – 3 : C# .NET-II

- Class fundamentals, OOPS concepts
- Arrays, Lists, Collections and iterating over them, Exception handling,
- Database programming – ADO.NET (architecture, connected and disconnected mode)
- Generating reports

Class Fundamentals

- **Class**
  - A blueprint/template that describes the behavior/state that the **object** of its type support.
  - **E.g.** A **class Car** that has
    - State (Attributes) :: Color, Brand, Weight, Model
    - Behaviors (Methods) :: Break, Accelerate, Slow Down, Gear Change
  - This is just a blueprint, it does not represent any Car, however using this we can create Car objects (or instances) that represents the car.
- **Object**
  - It is an instance of class.
  - When the objects are created, they inherit all the variables and methods from the class.

  E.g.
  - **Objects** : Audi, Toyota, Volvo, Maruti, Honda

Defining a Class

A class definition starts with the keyword class followed by the class name; and the class body enclosed by a pair of curly braces.

<access specifier> **class** class_name **{**

// member variables

<access specifier> <data type> **variable1;**

<access specifier> <data type> **variable2;**

...

<access specifier> <data type> **variableN;**

// member methods

<access specifier> <return type> **method1**(parameter_list) {

// method body }

<access specifier> <return type> **method2**(parameter_list) {

// method body }

...

<access specifier> <return type> **methodN**(parameter_list) {

// method body } **}**

OOPs Concepts

```
class Car {
  String color;
  String brand;
  int    weight;
  String model;
  void break() {
    //Write code here
  }
  void accelerate() {
    //Write code here
  }
  void slow_Down() {
    //Write code here
  }
  void gear_change() {
    //Write code here
  }
}
```

OOPs Concepts

- **Encapsulation**
  - Binding object state(fields / attributes) and behavior(methods) together.
  - **E.g.**
    - A **Class** is an example of encapsulation.
    - Inform what the object does instead of how it does.
    - Keep data and methods safe from outside interference or misuse.

OOPs Concepts

```
using System;
namespace RectangleApplication {
  class Rectangle {
    //member variables
    public int length;
    public int width;
public void Display() {
      Console.WriteLine("Length: {0}", length);
      Console.WriteLine("Width: {0}", width);    }
  }//end class Rectangle
class ExecuteRectangle {
    static void Main(string[] args) {
      Rectangle r = new Rectangle();
      r.length = 4;         r.width = 5;
      r.Display();         Console.ReadLine();     }  }     }
```

OOPs Concepts

- **Access Specifiers**

An **access specifier** defines the scope and visibility of a class member. C# supports the mainly three access specifiers −

  - Public
  - Private
  - Protected

- **Public** access specifier allows a class to expose its member variables and member functions to other functions and objects. Any public member **can be accessed from outside the class.**
- **Private** access specifier allows a class to hide its member variables and member functions from other functions and objects. **Only functions of the same class can access its private members.** Even an instance of a class cannot access its private members.
- **Protected** access specifier allows **a child class to access the member variables and member functions of its base class.** This way it helps in implementing inheritance.

OOPs Concepts

- **Inheritance**
  - The process by which one class acquires the properties and functionalities of another class is called inheritance.
  - Inheritance provides the idea of reusability of code and each sub class defines only those features that are unique to it, rest of the features can be inherited from the parent class.
  - **C# does not support multiple inheritance**. However, you can use interfaces to implement multiple inheritance.

Syntax:

<acess-specifier> **class <base_class>** {

... }

class <derived_class> : <base_class> {

... }

OOPs Concepts

```
using System;
namespace InheritanceApplication {
//Base class
  class Shape {
    protected int length;
    protected int breadth;
    public void setValues(int l, int b) {
      length = l; breadth = b;    }
  }
  // Derived class
  class Rectangle: Shape {
    public int getArea() {
      return (length * breadth);      }   }
```

OOPs Concepts

```
class RectangleTester {
    static void Main(string[] args) {

      Rectangle Rect = new Rectangle();
      Rect.setValues(5,4);

     // Print the area of the object.
      Console.WriteLine("Total area: {0}",  Rect.getArea());
      Console.ReadKey();
    }
  }
}
```

OOPs Concepts

- **Polymorphism**
  - It allows us to perform a single action in different ways.
  - In .NET, we use **method overloading and method overriding** to achieve polymorphism.
  - **Method overloading** is an example of static **polymorphism.**
  - **Method overriding** is an example of dynamic **polymorphism**.

  - **E.g. Method overloading : Addition of two numbers.**

    Both numbers may be int, float, double etc.

  - **E.g. Method overriding : Area of 2D shape :**

    Formula is different for Rectangle, Square, Triangle

OOPs Concepts

Method Overloading

Method overloading can be achieved by the following:

- By changing number of parameters in a method
- By changing the order of parameters in a method
- By using different data types for parameters

```
public class Methodoveloading
 {   public int add(int a, int b)  //two int type Parameters
  {       return a + b;    }
  public int add(int a, int b,int c)  //three int type Parameters
  {          return a + b + c;     }
  public float add(float a, float b)  //two float type Parameters
  {          return a + b ;     }
 }
```

OOPs Concepts

Method Overriding

–A base class have a methods.

–Multiple derived classes of Base class and they also have their own implementation of a methods

E.g.

```
class Base1
{  public void Display()
  {    System.Console.WriteLine("Base1::Good Morning");  }
}
class Derived1 : Base1
{  new public void Display()
  {    System.Console.WriteLine("Derived1::Good Afternoon");  }
}
class Derived1 : Base1
{  new public void Display()
  {    System.Console.WriteLine("Derived2::Good Evening");  }  }
```

OOPs Concepts

Method Overriding

```
class Demo
{  public static void Main()
  {        Base1 b;
      b = new Base1();
      b.Display();

      b = new Derived1();
      b.Display();

      b = new Derived2();
      b.Display();
  }
}
```

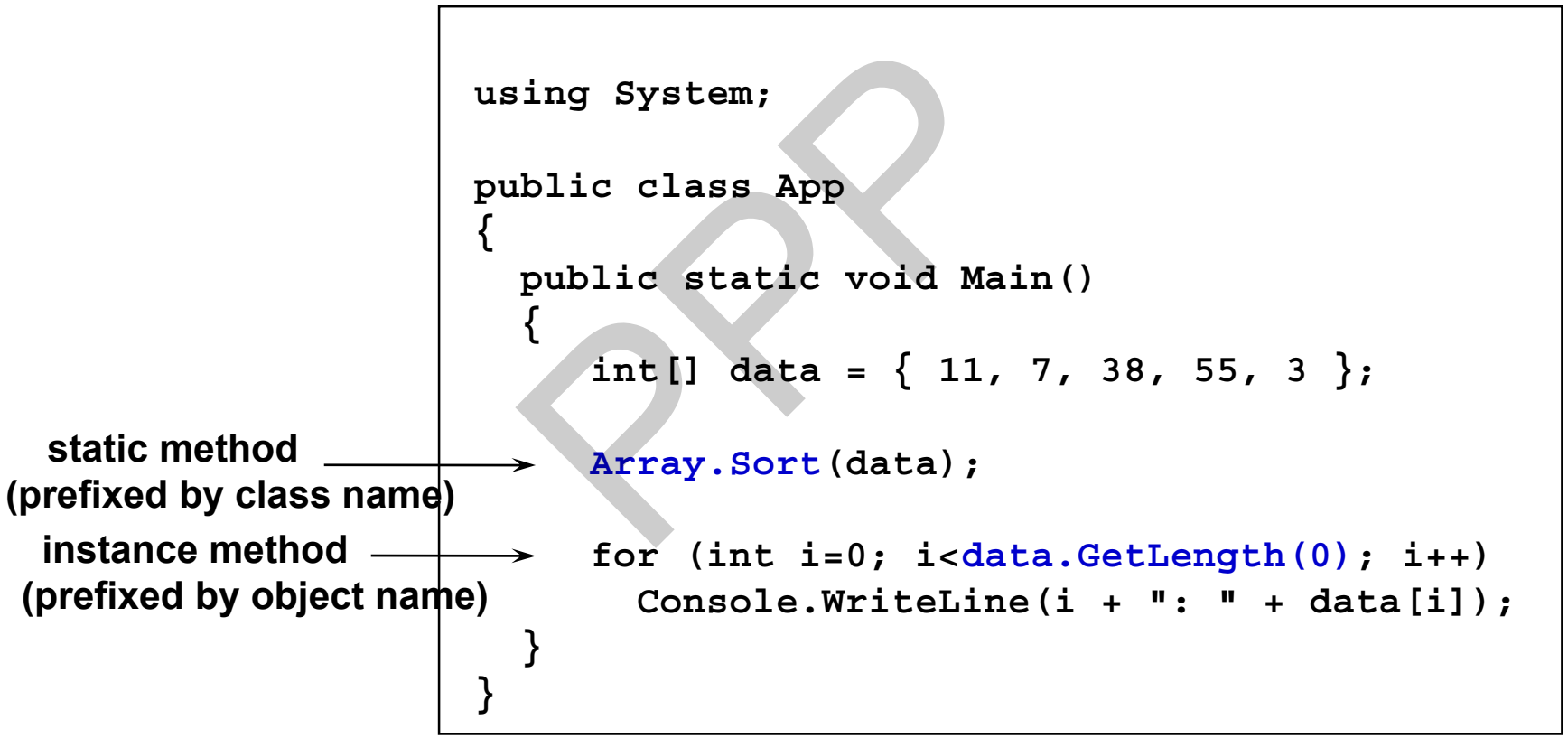
Types of methods

- Classes contain two types of methods:
  - Those with no return value (void)
  - Those with a return value (int, string, etc.)

- Methods may be:
  - instance
  - Static
- Instance methods require an object to call them
- Static methods are global and thus require only a class name

Calling methods

- **Here's an example of calling into the Array class:**




```
using System;

public class App
{
  public static void Main()
  {
    int[] data = { 11, 7, 38, 55, 3 };

    Array.Sort(data);

    for (int i=0; i<data.GetLength(0); i++)
      Console.WriteLine(i + ": " + data[i]);
  }
}
```

Arrays

- A collection of group of data variables of same type, sharing the same name for convenience

- Easy to search and manipulate

- Elements in an array are always numbered 0 through N-1, where N is the total number of elements, called the size or length of array

- Position of of an element in an array is called the element index or subscript

- Array of values are arranged in continuous memory locations

Arrays

One-Dimensional Array

Declaration of array

In C#, declaration of array is a reference declaration, no allocation of memory to hold array elements.

But no actual array created yet.

```
dataType[] arrayName;
```

Example

```
int[] data;
string[] sentence;
```

Arrays

Create an array using the `new` operator

```
arrayName = new dataType[size]
```

Allocate memory; hence an array

```
data = new [10];
```

Put together declaration and creation

```
int[] data = new int[100];
```

To access each element, use subscript or index, e.g. `data[i]`, where `i` should be in the range 0 through length-1

Arrays

C# has an easy way to initialize element values while declaring and creating the array

Example

```
int[] a = new int[2] {10, 20};
```

Or

```
int[] a = new int[] {10, 20};
```


`length`, a public method of `System.Array`, can be used to get the total number of elements, i.e. size of an array.

Lots of other methods in `System.Array`

Arrays

```
// Simple one dimensional array
using System;

class OneD {
public static void Main() {
int[] data = new int[5] { 1, 2, 3, 4, 5 };
int sum = 0;
Console.WriteLine("No. of elements = " + data.Length);
for (int i = 0; i < data.Length; ++i)
      Console.WriteLine(data[i]);
for (int i = 0; i < data.Length; ++i)
      sum = sum + data[i];
Console.WriteLine("sum of array elements: " + sum);
Console.ReadKey();
} }
```

Arrays

Two-dimensional arrays

Declaration of two-dimension arrays

`GetLength(0)` **and** `GetLength(1)` **respectively for the length of each dimension**

```
int[,] data = new int [3,5];
```

Arrays

```
// Simple two dimensional array
using System;
class TwoD
{
  public static void Main()
  {
    int[,] data = new int[3,5]; // row x column

    Console.WriteLine("No. of elements = " +
data.Length);

    for (int i = 0; i < data.GetLength(0); ++i) {
      Console.WriteLine("Row " + i + ": ");

      for (int j = 0; j < data.GetLength(1); ++j)
      {
        data[i,j] = i * j;
        Console.Write(data[i,j] + ", ");
      }
    Console.WriteLine();  }  }
```

Arrays

Two-dimensional array initialization

Different organizations

```
int[,] a = {{1,2}, {3,4}, {5,6}};

int[,] b = {{1,2,3}, {4,5,6}};

int[,] c = {{1,2,3,4,5,6}};
```

`a` has three rows of two elements

`b` has two rows of three elements

`c` has one row of six elements – a degenerate two-dimensional array!

List

- Requires a new `using` line at the top of your script

```
using System.Collections.Generic;
```

- List is a *generic collection*
  - Generic collections can work for any data type

```
List<string> sList;      // A List of strings
List<GameObject> goList;  // A List of GameObjects
```

- A List must be *defined* before it can be used (because Lists default to `null`)

```
sList = new List<string>();
```

- Elements are added to Lists using the Add() method

```
sList.Add("Hello");
sList.Add("World");
```

List

- List elements are accessed via *bracket access*
  - Bracket access is *zero indexed* (i.e., starts at `[0]`)

```
Console.WriteLine( sList[0] );        // Prints: "Hello"
 Console.WriteLine( sList[1] );       // Prints: "World"
```

- Lists have a Count of the number of elements

```
print( sList.Count );   // Prints: "2"
```

- Lists can be cleared of all elements

```
sList.Clear();          // Empties sList
```

List

- All these methods act on the List ["A","B","C","D"]
- Console.WriteLine(sList[0]); **// Prints first element : "A"**
- sList.Add("Apple"); **// ["A","B","C","D","Apple"]**
- sList.IndexOf("B"); **// 1 ("B" is the 1st element)**
- sList.IndexOf("Bob"); **// -1 ("Bob" is not in the List)**
- // sList.Insert(8, "X"); **// ERROR!!! Index out of range**
- sList.Insert(2, "X"); **// ["A","B","X","C","D","Apple"]**
- for (int i = 0; i < sList.Count; i++) **//print all elements of list**
- Console.WriteLine(sList[i]);
- sList.Remove("C"); **// ["A","B","X","D","Apple"]**
- sList.RemoveAt(1); **// ["A","X","D","Apple"]**
- sList.Clear(); **// [ ]**
- Lists can be converted to arrays
- string[] sArray = sList.ToArray();

List

```
using System.Collections.Generic;
class Test {
public static void Main() {
List<string> sList;     // A List of strings


//A List must be defined before it can be used(because Lists
    default to null)
sList = new List<string>();


sList.Add("A");
sList.Add("B");
sList.Add("C");
sList.Add("D");
```

List

```
Console.WriteLine(sList[0]);        // Prints first element : "A"
sList.Add("Apple");      // ["A","B","C","D","Apple"]
sList.IndexOf("B");      // 1   ("B" is the 1st element)
sList.IndexOf("Bob");        // -1   ("Bob" is not in the List)
// sList.Insert(8, "X");        // ERROR!!!   Index out of range
sList.Insert(2, "X");        // ["A","B","X","C","D","Apple"]
for (int i = 0; i < sList.Count; i++)  //print all elements of list
Console.WriteLine(sList[i]);
sList.Remove("C");       // ["A","B","X","D","Apple"]
sList.RemoveAt(1);       // ["A","X","D","Apple"]
sList.Clear();           // [ ]
Console.ReadKey();
}   }
```

When to Use List or Array

- Each have pros and cons:
  - List has flexible length, whereas array length is more difficult to change.
  - Array is very slightly faster.
  - Array allows multidimensional indices.
  - Array allows empty elements in the middle of the collection.

Collection

- Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables.

- Collection classes allocating memory dynamically to elements and accessing a list of items on the basis of an index etc.

- The following are the various commonly used classes of the **System.Collection namespace.**
  - ArrayList
  - Hashtable
  - Stack
  - Queue

Collection

ArrayList

–It represents ordered collection of an object that can be indexed individually.

–It is an alternative to an array. It can add and remove items from a list at a specified position using an index and the array resizes itself automatically. It also allows searching and sorting items in the list.

Hashtable

–It uses a key to access the elements in the collection.

–A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

Collection

Stack

–It represents a last-in, first out collection of object.

–It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.

Queue

–It represents a first-in, first out collection of object.

–It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called enqueue and when you remove an item, it is called deque.

Collection

```
Console.Write("Sorted Content: ");

        // Sort the elements in arraylist.
            al.Sort();

            // Show each element of array list
            foreach (int i in al)
            {
                Console.Write(i + " ");
            }
            Console.ReadKey();
        }
    }
```

Collection

```
using System;
using System.Collections;
namespace CollectionsApplication {
   class Program {
      static void Main(string[] args) {
         Stack st = new Stack();
 st.Push('A');  st.Push('B'); st.Push('C'); st.Push('D');

         Console.WriteLine("The next poppable value in stack: {0}", st.Peek());
         st.Push('E');
         Console.WriteLine();   Console.WriteLine("Removing values ");
         st.Pop();
         Console.WriteLine("Current stack: ");
         foreach (char c in st) {
            Console.Write(c + " ");
         }     }  }   }
```

Exception

It is a problem that arises during the execution of a program.

E.g. an attempt to divide by zero, array index out of bound etc.

Exceptions provide a way to transfer control from one part of a program to another.

C# exception handling is built upon four keywords:

try, catch, finally, and throw.

try − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.

catch − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

Exception

finally − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.

throw − A program throws an exception when a problem shows up. This is done using a throw keyword.

A try/catch block is placed around the code that might generate an exception.

Syntax

```
try {
   // statements causing exception
} catch( ExceptionName e1 ) {   // error handling code
} catch( ExceptionName e2 ) {   // error handling code
} catch( ExceptionName eN ) {   // error handling code
} finally {   // statements to be executed
}
```

Exception Classes in C#

Exception

Exception Classes in C#

The exception classes derived from the **System.Exception class.**

E.g. System.ApplicationException and System.SystemException classes.

The **System.SystemException class** is the base class for all predefined system exception.

The **System.ApplicationException class** supports exceptions generated by application programs.

The predefined exception classes derived from the Sytem.SystemException class −

System.IO.IOException

Handles I/O errors.

System.IndexOutOfRangeException

Handles errors generated when a method refers to an array index out of range.

Exception

System.ArrayTypeMismatchException

Handles errors generated when type is mismatched with the array type.

System.NullReferenceException

Handles errors generated from referencing a null object.

System.DivideByZeroException

Handles errors generated from dividing a dividend with zero.

System.InvalidCastException

Handles errors generated during typecasting.

System.OutOfMemoryException

Handles errors generated from insufficient free memory.

System.StackOverflowException

Handles errors generated from stack overflow.

Exception

```
class DivNumbers    {
        static void Main(string[] args) {
            int result = 0;
            int num1 = 50, num2 = 10;
            try {                result = num1 / num2;            }
            catch (DivideByZeroException e)  {
                Console.WriteLine("Exception caught: {0}", e);            }
            finally  {
                Console.WriteLine("Result: {0}", result);            }

            Console.ReadKey();
        }    }
```

Exception

Creating User-Defined Exceptions

You can also define your own exception.

User-defined exception classes are derived from the **Exception** class.

User Defined Exception

```
public class Temperature {
    int temperature = 0;
    // int temperature = 10;
    public void showTemp()  {
        if (temperature == 0) {
            throw (new TempIsZeroException("Zero Temperature found"));
        }
        else {
            Console.WriteLine("Temperature: {0}", temperature);
        }
    }
public class TempIsZeroException : Exception {
    public TempIsZeroException(string message) : base(message)
    {   }   }
```

User Defined Exception

```
class TestTemperature    {
        static void Main(string[] args){
            Temperature temp = new Temperature();
            try {
                temp.showTemp();            }
            catch (TempIsZeroException e) {
                Console.WriteLine("TempIsZeroException: {0}", e.Message);
            }
            Console.ReadKey();
        }    }    }
```

Various Files in Windows form Applicatoin

- An SLN file is a structure file used for organizing projects in Microsoft Visual Studio. It contains text-based information about the project environment and project state.

- In Solution explorer **.sln file** is automatically created when new project is created.

- When opened, the preSolution, Project, and postSolution information is read from the SLN file. This data is used to load the solution, projects within the solution, and any persisted information attached to the solution.

Various Files in Windows form Applicatoin

- **A Form-name.cs & program.cs file is created automatically in the Visual Designer.**
- **Program.cs file** is the entry point of the application. This will be executed first when the application runs, there is a public static void Main method, Whatever code you write inside that method will be executed in that same order
- **The code for Form-name is divided between the Form-name.cs file and the Form-name.Designer.cs file.**
- **Form-name.cs** is the **code-behind file** of the windows form.
- It is the class file of the windows form where the necessary methods, functions also **event driven methods and codes are written. One of the Method InitializeComponent** to initialize.
- **Form-name.designer.cs** is the designer file where form elements (Normally Drag and Drop controls) are initialized. elements will be automatically initialized in this class. **One of the method dispose** to clear the resources.

Various Files in Windows form Applicatoin

Form-name.cs file

- In the source code view for **Form-name.cs**

  Class called Form1 is defined. Form1 has a **base class called Form** (actually it is **System.Windows.Forms.Form**).

- It also has a new keyword called **partial** in its definition. This indicates to the compiler that one part of the class is defined in this file (currently only the constructor).

- After drag and drop controls in form you may set Dock and Anchor property of it. The **Dock and Anchor properties** dictate how the control will react when its parent control is resized

Various Files in Windows form Applicatoin

Form-name.resx file

- It's the resources file.
- It can contain strings for localization, images, icons...
- Try double-clicking on it in the Solution Explorer.
- The resources editor will appear.
- At the top left of the window, you can select the type of resources you want to examine (strings, images, audio...).
- In general, for each application resource that is selected as translatable, VisualStudio will create one .resx file. However the developer also needs to specify the languages this resource will be translated into! And for each language a new .resx file is also created by VisualStudio .NET.
- **E.g.** suppose a developer selects a WinForm for localization into French and German, VisualStudio will create three .resx files.

  Base Language : Form1.resx French Language: Form1.fr.resx

  German Language: Form1.de.resx

Assembly in .NET

- Assemblies are the core building blocks of .NET applications. In simple terms, **assembly is the .dll or .exe file** that is created while building any .NET application.
- You can find many definitions for assembly over the internet also. In my point of view, assembly is the building block that is created after a successful build operation of your application.
- **After the compilation of each project, a .exe or .dll file is generated, known as Assembly.**
- The name of an assembly file is the same as the project name.
- **The assembly of a project present under the bin\Debug folder of that project folder.**
- After the successful build of each project, an assembly is generated inside that folder.
- We may directly take the assembly file and install it into the client machine if it is **.exe type** or we can consume the file in another project if the assembly is **dll type**.

Assembly in .NET

- Class Library and Windows Forms Control Library projects generate a **.dll assembly** whereas Windows Forms Applications, Console Applications, WPF Applications, and Windows Services projects generate a **.exe assembly**.

- **.Exe type assembly** is run by their own as well as they provide support to others in their execution process so these assemblies are known as an **in-process component**

- **.dll type assembly** is called as **out process component** because they are only developed for the purpose to consume inside .exe type of application.

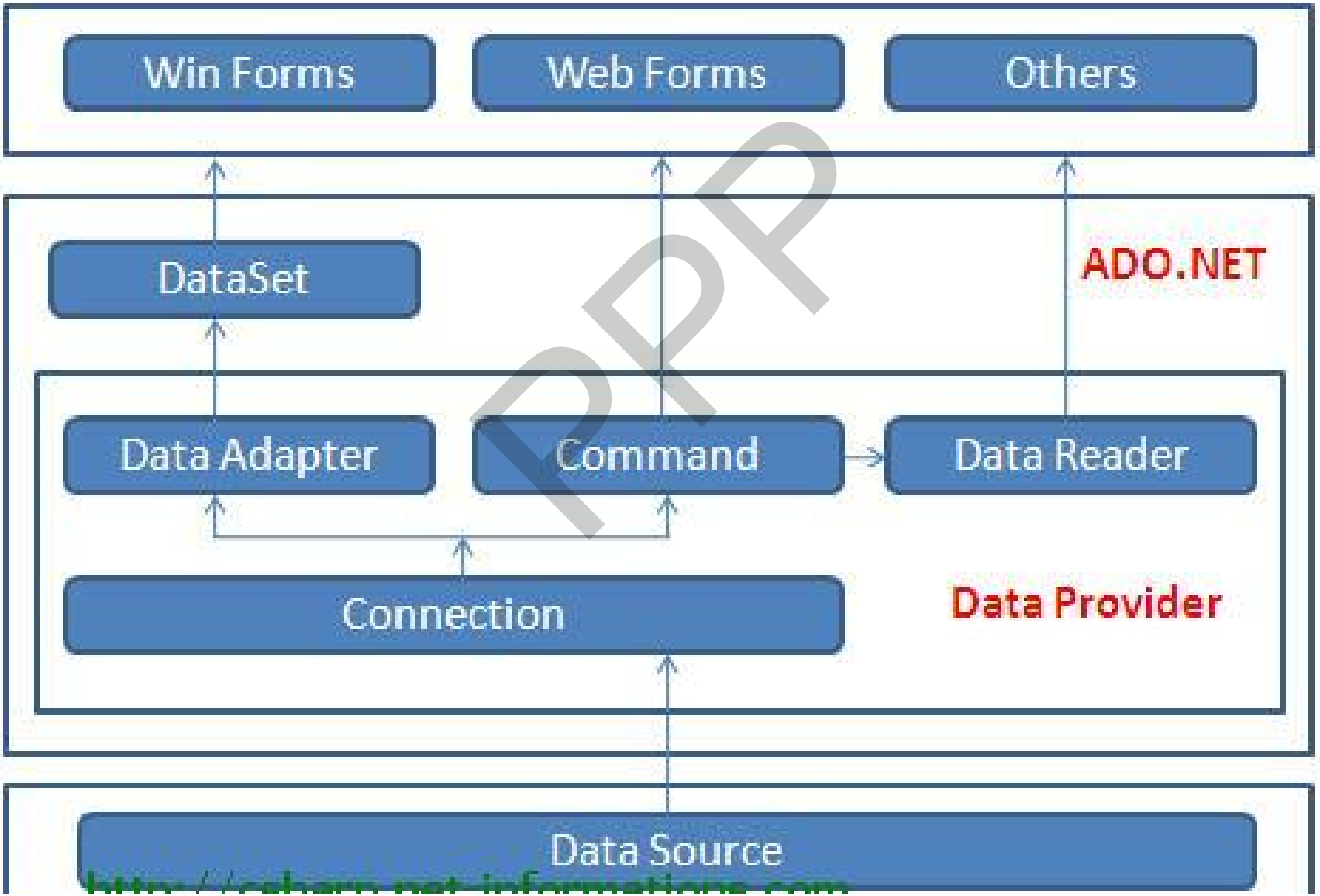
(Database Programming : ADO.NET)

What is ADO?

- ADO stands for **A**ctiveX **D**ata **O**bjects
- ADO is a Microsoft Active-X component
- ADO is automatically installed with Microsoft IIS
- ADO is a programming interface to access data in a database

(Database Programming : ADO.NET)

ADO.NET Architecture



(Database Programming : ADO.NET)

1. Connected Architecture

- The connection is established for the full time between the database and application.
- **E.g.** We make a program in C# that is connected with the database for the full time, so that will be connected architecture.
- Connected architecture is forward only and read-only. This means the connected mode will work only in one particular direction i.e. forward and that too for read-only purpose.
- Application issues query then read back results and process them.
- For connected architecture, use the object of the **DataReader class.**
- DataReader is used to retrieve the data from the database and it also ensures that the connection is maintained for the complete interval of time.
- In connected architecture, the application is directly linked with the Database.

(Database Programming : ADO.NET)

- To make an object of the DataReader class, we never use the new keyword instead we call the ExecuteReader() of the command object. **For e.g.**

SqlCommand cmd = new SqlCommand("Select * from Table");

SqlDatareader rdr = cmd.ExecuteReader(cmd);

- Here **cmd.ExecuteReader()** executes the command and creates the instance of DataReader class and loads the instance with data. Therefore, we do not use the 'new' keyword.

(Database Programming : ADO.NET)

2. Disconnected Architecture

- Disconnected architecture refers to the mode of architecture in Ado.net where the connectivity between the database and application is not maintained for the full time.
- Connectivity within this mode is established only to read the data from the database and finally to update the data within the database.
- This means during the processing of the application, we need data so that data is fetched from the database and kept in temporary tables.
- After that whenever data is required, it is fetched from the temporary tables. And finally, when the operations were completed, the connection was established to update the data within the database from the temporary tables.

(Database Programming : ADO.NET)

2. Disconnected Architecture

- In this mode, application issues query then retrieves and store results for processing.

- For this purpose, we use objects of **SqlDataAdapter and DataSet classes**.

- In disconnected architecture, a Dataset is used for retrieving data from the database.

- This way there is no need to establish a connection for the full time because DataSet acts as temporary storage.

- All the operations can be performed on the data using the Dataset and finally modified at the database.

(Database Programming : ADO.NET)

DataAdapter in Disconnected architecture

- DataAdapter class acts as an interface between application and database. It provides the data to the Dataset which helps the user to perform the operations and finally the modifications are done in the Dataset which is passed to the DataAdapter which updates the database.

- DataAdapter takes the decision for the establishment and termination of the connection.

- DataAdapter is required for connectivity with the database. DataAdapter established a connection with the database and fetches the data from the database and fill it into the Dataset.
- And finally, when the task is completed it takes the data from the DataSet and updates it into the database by again establishing the connection.

(Database Programming : ADO.NET)

DataAdapter in Disconnected architecture

- It can be said that DataAdapter acts as a mediator between the application and database which allows the interaction in disconnected architecture.

```
public DataTable GetTable(string query)        {
SqlDataAdapter adapter = new SqlDataAdapter(query, ConnectionString);
        DataTable Empl = new DataTable();
        adapter.Fill(Empl);
        return Empl;    }
```

- Here, the object of the SqlDataAdapter is responsible for establishing the connection.
- It takes query and ConnectionString as a parameter.
- The query is issued on the database to fetch the data.
- ConnectionString allows connectivity with the database.
- The fill() of the SqlDataAdapter class adds the Table.

(Database Programming : ADO.NET)

- **SqlConnection** and **SqlCommand** are classes of a connected architecture and found in the **System.Data.SqlClient namespace**.

- The **SqlConnection class** makes a connection with the database.

- Database connection is used by the **SqlCommand** to work with that database.
- The **SqlCommand class** is used to execute the SQL statements.

- **SqlCommand Method**

1. **ExecuteNonQuery() :** The ExecuteNonQuery() method does not return any record. Which means we use ExecuteNonQuery() in all operations with databases except retrieving records from a database. It only returns the number of affected rows.

2. **ExecuteReader( ) :** It executes SQL statements and returns zero or more rows. So when there is need to show records from database in connected architecture

(Database Programming : ADO.NET)

SqlDataReader :

- This is the class of connected architecture in .NET framework.
- The SqlDataReader is used to read a row of record at a time which is got using SqlCommand.
- It is read only, which means we can only read the record; it can not be edited.
- It is used with the ExecuteReader method of the SqlCommand class.

(Database Programming : ADO.NET)

Create a New Database

- From main menu view -> SQL Server Object Explorer to open databases.
- Select databases and right click on it and select Add New Database
- Give new database name e.g. temp.
- Select temp database.

Create table in database

- Right click on database temp then select new query option and entered following command to create a new table.

```
CREATE TABLE Product (
pid   INT          NOT NULL,
pname VARCHAR (1) NOT NULL,
PRIMARY KEY (pid ASC)  );
```

(Database Programming : ADO.NET)

Insert records in tables

- Now again selected temp Database then select refresh option to see created table product in list.

- Now select product table in temp database then right click and select view data and entered the actual data for product table.

(Database Programming : ADO.NET)

Namespaces for database programming.

==========

Basic Namespaces

----

using System;
using System.Windows.Forms;
using System.Data.SqlClient;

Optional Namespaces

----

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;

(Database Programming : ADO.NET)

Declare variables in form

```
SqlConnection   conn;
SqlCommand      comm;
SqlDataReader   dreader;
```

```
//Connection string for database connectivity
string connstring = "Data Source =(LocalDB)\\MSSQLLocalDB;
Integrated Security=True;
Connect Timeout=30;
AttachDbFilename=C:\Users\Administrator\AppData\Local\Microsoft\Microsoft SQL Server Local DB\Instances\MSSQLLocalDB\FirstDatabase.mdf";
```

(Database Programming : ADO.NET)

//When the form load we set focus on pid field of product table.

```
private void formdatabase_Load(object sender, EventArgs e)
        {
            txtpid.Focus();
        }
```

(Database Programming : ADO.NET)

Write following code on save button click event

```
private void btnsave_Click(object sender, EventArgs e)
        {
//Database connection object
        conn = new SqlConnection(connstring);

//Open database connection
        conn.Open();

//Command object to execute query on database
        SqlCommand comm = new SqlCommand("insert into
product values(" + txtpid.Text + "'" + txtpname.Text + "')", conn);
```

(Database Programming : ADO.NET)

```
try
    {
    // Execute Insert Query
     comm.ExecuteNonQuery();
    // If insert successful display message “Saved”
     MessageBox.Show(“ Record is save..");
    }
catch (Exception)
    {
        MessageBox.Show(“Record does not save");
    }
finally
    {
        conn.Close();
    }
}
```

(Database Programming : ADO.NET)

```
// To clear the content of a form fields

private void btnclear_Click(object sender, EventArgs e)
        {
            txtpid.Clear();
           txtpname.Clear();       }
```

(Database Programming : ADO.NET)

```
// Delete record from product table in database
private void btndelete_Click(object sender, EventArgs e)
        {   conn = new SqlConnection(connstring);
            conn.Open();
            comm = new SqlCommand("delete from product where pid
= " + txtpid.Text + " ", conn);
 try  {
                comm.ExecuteNonQuery();
                MessageBox.Show("Deleted...");
                txtpid.Clear();
                txtpname.Clear();
                txtpid.Focus();            }
            catch (Exception x) {
                  MessageBox.Show(" Not Deleted" + x.Message);   }
            finally
            {               conn.Close();           }       }
```

(Database Programming : ADO.NET)

// To update the product table data in database.

```
private void btnupdate_Click(object sender, EventArgs e)
        {  conn = new SqlConnection(connstring);
           conn.Open();
           comm = new SqlCommand("update product set pname = '"
+ txtpname.Text + "' where pid = " + txtpid.Text + "", conn);
           try
           {
              comm.ExecuteNonQuery();
              MessageBox.Show("Updated..");
           }
           catch (Exception)
           {
              MessageBox.Show(" Not Updated");
           }
           finally
           {                conn.Close();          } }
```

(Database Programming : ADO.NET)

```
// Search record in product table with pid
private void btnsearch_Click(object sender, EventArgs e) {
        conn = new SqlConnection(connstring);
        conn.Open();
        comm = new SqlCommand("select * from product where
pid = " + txtpid.Text + " ", conn);
        try {
            dreader = comm.ExecuteReader();
            if (dreader.Read()) {
                txtpname.Text = dreader[1].ToString();   }
            else {
                MessageBox.Show(" No Record");  }
            dreader.Close();  }
        catch (Exception) {
            MessageBox.Show(" No Record"); }
        finally {
            conn.Close();  }       }
```