Visual Basic.NET Object-Oriented Programming

Implementing Business Objects & Components

Part (I of III)

(Lecture Notes 3A)

Professor: A. Rodriguez
3.1 Business Objects Components (Review)

3.1.1 Business Objects Basic Key Components

- In our last lectured we summarized the key properties of an objects:
  - **Data:** The private variables
  - **Interface:** Property Let & Get/Methods, Event-handlers
  - **Behavior:** The services provided by the object’s methods or what it can do.

- In summary, we can see that Objects have three key items: 1) **Data**, 2) an **interface**, & 3) **behavior**.

3.1.2 Business Objects Requirements

- OK, now that we understand the theory of classes and Objects, let understand Business Objects.
- **Business Objects** are Objects that model and simulate the real world and business processes.
- **Business Objects** needs to contain the following business rules:
  1. **Business Object must contain all Business Logic & Rules** – contain the necessary Business Logic & Rules to perform their business process or functions
  2. **Validation** – Business Objects should contain the logic to verify that the data being set by the user is valid, correct data type, length etc. Business Objects should protect itself from unauthorized or unwanted, harmful access:

- **Business Objects** needs to contain the following characteristics:
  1. **User Interface Support**:
     - Contain all the **features** and **functionality** the UI-Developer will need to make communication between the User & the application effective.
     - The **Business Objects** are the **core** of the application and must be designed in a way that is very easy to change the **UI Layer** without risking the business logic stored in the business objects.
  2. **They manage their own data & database access**:
     - The Business Object should contain all the code to manage the data access or interact with the database. Operations such as searching, inserting, updating, deleting the database should be done by the business objects.
     - Database access should NOT be performed in the User Interface Layer but from the Business Object Layer.
  3. **Scalable & Reusable**:
     - Can **evolve** & gain new data, properties & methods to support more functionality.
  4. **Distributed Business Objects**:
     - Business Objects should contain the technology to allow them to be distributed across processes and application.
     - Distributed Objects are about sending the object (smart data) from one machine to another, rather than sending raw data and hoping that the business logic on each machine is being kept in sync.
3.2 Distributed 5-Tier Application Architecture & Client/Server Architecture (Review)

### 3.2.1 N-Tier Distributed Application Architecture

- In our last lecture, we also introduced an n-tier **Scalable** Application Architecture that addresses the necessary components to develop powerful distributed applications.
- This architecture is designed to work well whether it’s placed entirely on a single workstation or whether it is spread across a number of machines or servers on a network.
- This logical architecture breaks the application development process into five layers or tiers:

  - **Presentation or User-Interface Layer**: Front-end or client that users will interact with. It can be Forms, Web Forms or any type of Graphical Interface.
  - **Business Object Layer**: Contains all the Business Objects of the application. All the business processing, logic, rules, validation, data access etc. should be in this layer. Works more closely with the UI and performs the business processing.
  - **Data Access Business Objects Layer**: Components or Objects that work more closely with accessing & storing the data to the database system. These objects perform the actual data access from the Database Storage Layer.
  - **Database Storage and Management**: Database Management system (DBMS) that will actually store and manage the data. This layer actually represents the database itself such as Oracle, MS SQL and MS Access.

- Using this architecture, there are generally two categories of application development:
  1. **Distributed Application (Non-Web)** - These are the standard traditional client/server applications in which the client process runs on the workstation and the server process on another machine or server. For these applications, we use the four-tier architecture in which the presentation and User Interface layer are the same:

![Four-tier architecture diagram]

  - Presentation Layer/UI
  - Business Logic Objects
  - Data Access BO
  - Database Layer (DBMS)

  2. **Web-Based Distributed Applications** - These are applications that follow the Web-Base Client/Server architecture. These applications use the five-tier architecture where the presentation layer is the browser and the UI is on the web server:

![Five-tier architecture diagram]

  - Presentation Layer
  - User Interface Layer (UI)
  - Business Logic Objects
  - Data Access BO
  - Database Layer (DBMS)
3.2.2 Business Objects and the Client/Server Architecture

Introduction

- Client/Server Technology is the standard that has been used when creating network applications and systems.
- The Client/Server definition is as follows:
  - **Client** – Process or executable program that makes requests to another process or executable
  - **Server** – Process or executable program that services or complies to the request made by a client process, thus the name server.

- The **Client/Server Architecture** is composed of server architectures. They are as follows:
  - Single-Tier
  - Two-Tier
  - Three-Tier
  - Web-Based

- Our goal is to design distribute applications using our 5-tier Application Architecture that fit the Client/Server Architecture in use today.
- Therefore we need to determine how to place the individual layers of the 5-tier Application Architecture in the various components that make up the Client/Server Architecture.

Business Objects Placement in a Single-Tier

- As for In a Single-Tier, the point is that the Presentation, User-Interface (UI), Business Logic or Objects, Data Access Layer and Data processing are all on the same computer:

![Client Workstation Diagram](image-url)
Business Objects Placement in a Two-Tier

- Two-tier consist of a client machine, and a separate machine or Database Server such as Microsoft SQL Server or Oracle Server.
- Placement of Business Object really depends on which type of Two-tier architecture we are designing:

1. **Fat-Client/Thin-Server:** Client does most of the processing. Presentation, User-Interface (UI), Business Logic or Objects, Data Access Layer are done on client. The Data Service Layer is the actual Database Server, which handles the storage or services the data.

   ![Diagram of Fat-Client/Thin-Server architecture]

2. **Thin-Client/Fat-Server:** Client has less processing; therefore the Presentation, User-Interface (UI) and Business Logic are on the client. The Data-Centric Business Objects, which handles the data access code, resides on the server with the Database Processing Layer.

   ![Diagram of Thin-Client/Fat-Server architecture]
Business Objects Placement in a Three-Tier

- Usually in a Three-Tier configuration, the Presentation Layer, User-Interface (UI) and usually the Business Objects are kept in the client, the Data-Service Objects that handle database access resided in the Application Server, and the processing of data or Database Access layer resides on the Database Server.

![Diagram of a Three-Tier architecture]

Business Objects Placement in Web-Based Client/Server

- In a Web-Based configuration, the Presentation Layer is the Web Browser on the client, and the User-Interface (UI), which handles the User-Interface code is the HTML/ASP code in the Web Server. In addition, the Data-Access Objects that handle database access resided in the Web Server. As usual, the Data Service layer or database itself resides on the Database Server.

![Diagram of a Web-Based architecture]
### 3.3 Implementing Business Objects (IMPORTANT)

#### 3.3.1 .NET Key Technologies to Implement the 5-Tier Application Architecture

- In order to implement our architecture, the Business Objects will need to be distributed; they need to possess the technology to allow them to be sent from one machine to another throughout the network.
- The .NET key technologies that will allow us to implement *Distributed Business Objects* are:

  1. **Class Library Project (DLL)** – Business Objects need to be packaged as a Class Library or DLL (Dynamic-Link-Library) in order for them to be portable. This is the first step in creating distributed objects. You can package one or several classes into one DLL container.

  2. **Remoting** – Subsystem of the .NET Framework that allows .NET applications to interact with each other whether in one computer or computer to computer communication.

  3. **Serialization** – Process of converting a complex set of data, such as that contained in an object, into a single data or byte stream. The byte stream can be a stream of byte, text, XML stream etc. **Deserialization**, is the process of unpacking the byte stream to recreate the original complex data.

- We will be covering these technologies in the next several lectures.

### 3.3 Single-Process & Cross-Process Communication

#### 3.3.1 Introduction

- Objects are normally part of an application or process. Usually running on a single machine or single process. This is known as **Single-Process Communication**. In other words, the Presentation/User Interface and Business Objects are all within the same process.

- But, as we saw earlier, the 5-Tier Application Architecture was distributed or spread out across the Client/Server architecture.

- Therefore, in a distributed application, some of our objects might run on a single process or may run on different individual machines within different processes. This is known as **Cross-Process Communication**.

#### 3.3.2 In-Process/Single-Process Communication

- An In-Process or Single-Process *Objects runs inside the client's process or memory space.*

- An In-Process Object can only be accessed by other members of the process. CANNOT be access outside the process.

- **Single-Process communication** has the following characteristics:

<table>
<thead>
<tr>
<th>Advantages/Characteristics</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ <em>Fastest</em> way to interact with the objects since objects resides in the same Process.</td>
<td>▪ Can NOT serve multiple applications</td>
</tr>
<tr>
<td>▪ Less Overhead</td>
<td>▪ Objects cannot stand-alone in one machine or without the client.</td>
</tr>
</tbody>
</table>

![Single-Process Communication](image-url)
3.3.3 Cross-Process/Out-of-Process Communication

- Out-of-Process communication means that Objects running in different processes can interact with each other.
- Keep in mind that each processes effectively a separate **EXECUTABLE** program.
- It runs in its own process or memory space and thus independent from any other programs.
- Out-of-Process communication has the following characteristics:

<table>
<thead>
<tr>
<th>Advantages/Characteristics</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Cross-process components allows communication with applications running on a single computer as well as a remote computers</td>
<td>▪ <strong>Cross-Process communication</strong> between the client and the objects means that there is more <strong>overhead</strong></td>
</tr>
<tr>
<td>▪ Allows for implementation of distributed applications and network communication.</td>
<td>▪ Slower Communication link due to cross-process communication</td>
</tr>
<tr>
<td>▪ Our 5-Layer application architecture can be implemented and the layers can spread out across the network.</td>
<td></td>
</tr>
</tbody>
</table>

3.3.4 Summary

- Obviously from the two diagrams above we see that we need both for **Single-Process Communication**, which is the norm, and **Cross-Process Communication** in order to implement distributed applications across a network.
- Because of these requirements, Business Objects in .NET are classified as **Local, Anchored** or **Unanchored**.
3.4 Local, Anchored, and Unanchored Objects

3.4.1 Local Objects (.NET Default)
- Local objects are what we have been working with so far in CS608 and up to this point of CS708.
- This is the default in .NET. By default when we create and object within an application, this object is a local object.
- Local objects mean that the objects we create are local to our application and can only be accessed by the process in which they were created.
- Local Objects running in one application or process CANNOT be accessed by another process.
- The UI and other components of the application communicate with the object via Single-Process Communication.

Implementing Local Objects
- To implement Local Objects we don’t need to do anything special. Since this is the default, when we create objects re what we have been working with so far in CS608 and up to this point of CS708.
- This is the default in .NET. By default when we create and object within an application, this object is a local object.

3.4.2 Anchored Objects - Cross-Platform Communication (Pass-by-Reference)
- Now we look at cross-process communication between our objects. But, just from the name of these objects we can deduct that these objects are anchored or don’t move from one location.
- This is a new classification for objects. Anchored objects that reside in one place. You may be confused because since we want to spread our objects around, why would we want them anchored in one place?
- Actually, we will be able to spread these objects around but by the concept of Pass-by-Reference.
- Anchored objects stay in one location, but other process can access them by Reference or a pointer to the object.
- This means that when we pass objects from computer to computer what we pass is a reference or pointer and not the original object.
- This type of object is stuck or anchored on the machine or process in which it was created. It never moves, but can be accessed by other processes by reference.
- Calling a property or method is done across the network and processed by the object via the reference, and the result also sent across the network.
- IMPORTANT! Note that if need to call an anchored object in another machine and we need to make quite a number of property or method calls, this will involve a large overhead. Because of this, getting information from this type of object is done with one call, where you get everything in one shot, to avoid repeated calls to properties or methods.
The following diagram illustrated anchored objects:

Process 1 Executable

When to use Anchored Objects
- Anchored objects are important, because we can guarantee that they will always run on a specific machine only.
- For example, in a Two-Tier Thin Client/Fat Server, and the Three-tier client/server architectures, we note that the Data Access Business Objects Layer needs to run on a specific server or Application Server. This particular server needs to have access to the database server, therefore objects in this layer should be anchored to that specific machine.

Implementing Anchored Objects
- To implement Anchored Objects we need the following .NET technologies:
  - Class Library Project (DLL) – Business Objects need to be packaged as a Class Library or DLL (Dynamic-Link-Library).
  - Inherit from .NET Framework class MarshalByRefObject:

```csharp
Public Class MyAnchoredClass
    Inherits MarshalByRefObject
End Class
```
  - Remoting – .NET Subsystem that handles communication between objects across a Network. In this case By-Reference.

From this point the .NET Framework takes care of the rest.
3.4.3 UnAnchored or Distributed Objects - Cross-Platform Communication (Pass-by-Value)

- **Unanchored Objects** are known as **Distributed Objects**. These type of objects can be passed from one process to another process or from one machine to another, **By-Value**. By value means that a copy of the original object is placed on the target machine.
- Since a copy of an object is passed to another machine, the other components of that machine can interact with locally or within the process or **In-Process communication**.
- Note that the overhead of this type of communication is during the copy of the object across the network. But once the object is copied, since it now becomes **In-process communication** there is practically no overhead when making multiple calls to properties and methods.
- The diagram below illustrates this concept:

- **When to use Unanchored or Distributed Objects**
  - Unanchored or Distributed objects need to move from process to process or from machine to machine.
  - In the Two-Tier, Three-tier and Web-based client/server architectures, the Business Objects Layer is a good candidate for classification as unanchored since these objects perform the processing and need to be sent through the network in order to get populated and distribute the Business Logic thought-out the network.
Implementing Unanchored Objects

- To implement *Unanchored Objects* we need the following .NET technologies:
  - **Class Library Project (DLL)** – Business Objects need to be packaged as a Class Library or DLL (Dynamic-Link-Library).
  - Insert the `<Serializable()>` _Attribute statement and an underscore before the class declaration:

    ```csharp
    <Serializable()> 
    Public Class MyUnAnchoredClass 
    End Class
    ```

- **Remoting** – .NET Subsystem that handles communication between objects across a Network. In this case *By-Value*.

- From this point the .NET Framework takes care of the rest.

3.4.4 Summary

- **Local objects**:  
  - Can only be accessed by components within its process.
  - This is the default in .NET, therefore nothing needs to be done to implement.

- **Anchored objects**:  
  - These objects are stuck on the process or machine in which they were created and are important, because we can guarantee that they will always run on a specific machine only.
  - Communication with these types of objects is via *Pass-By-Reference* or a *pointer* is passed to other processes that wish to communicate with the Anchored Objects.
  - *Data Access BO Layer* will be created as Anchored Objects since they need to run on a specific machine with access to the Database Layer.
  - To implement we need to Inherit our classes from the `MarshalByRefObject` class.

- **Unanchored Objects or Distributed Objects**:  
  - Distributed Objects can be passed from one process to another process or from one machine to another, *By-Value*. By value means that a *copy* of the original object is placed on the target machine.
  - The *Business Objects Layer* is a candidate as *Distributed or Unanchored Objects*.
  - To implement, you need to use the `<Serializable()>` _attribute statement.

- The *Anchored* and *Unanchored* Objects require the following technologies:
  - **Class Library Project (DLL)** – Business Objects need to be packaged as a Class Library or DLL (Dynamic-Link-Library).
  - **Remoting** – .NET Subsystem that handles communication between objects across a Network. Either *Pass-By-Reference* or *Pass-By-Value*. 
3.5 Creating Components - Dynamic-Link-Library (DLL) - Creating Class Library

3.5.1 Introduction to Components or Dynamic-Link Libraries (DLL)

- A dynamic Link Library is a container or block of code containing procedures, functions, classes etc. that are available to any program that want to use it during run time.
- The code inside the DLL container can be written in any computer language, but it must be compiled into machine language, thus accessible any application regardless of the language used to create the application.
- It’s called dynamically linked since the procedures & functions inside the DLL are not statically linked to any particular executable program.
- What is meant by statically or dynamically linked?
  - First let understand what are we talking about here. When you create method (Function/Procedures) you take two steps:
    1) Create the Method
    2) Call the Method from your code.
  - Well, when you compile your program, the compiler statically links or binds the Method Call to the actual method
  - This means that a copy of the method call is actually added to the location where the Call was made.
  - This process applies to a Static Library file with Methods & Classes that you purchase and add to your program. The methods or classes in the library are statically linked or bind during the compilation.
  - This means your program is now larger
- Note that Dynamic Link Libraries files have extension .dll.
- Note that this topic of DLL’s is general to any programming language, whether is C++, C#, VB.NET, Java etc
- FORMS, FORM CONTROLS OR ANY TYPE OF USER-INTERFACE CODE CANNOT EXIST INSIDE A DLL!!!!
- DLL’s extend the concept of code reusability since Classes are packaged within the DLL container and can be distributed.
- DLL’s have the following advantages:
  - Reduced Size: The size of executable programs can be small since we are keeping common functions and procedures in a DLL.
  - Minimal Maintenance: Easier to maintain, since we can make changes to the DLL while not having to touch the executable programs.
  - Consistency: We can write consistent programs since we can use the code in the DLL across a variety of applications, giving those applications similar functionality.
  - Using .NET creating a DLL is a snap. In the past, DLL were difficult to program.

- DLL’s have the following disadvantage:
  - Complex to distribute & Manage: DLL’s required that you keep track of version number, type etc. when upgraded and distributed among applications.
  - CANNOT EXECUTE ON THEIR OWN! THEY NEED TO BE HOSTED BY A PROGRAM OR PROCESS!!!!

Dynamic Link Library & .NET(OOP)

- In the next few lectures, and the remainder of this course, we will place our Class Objects inside DLL’s.
- By placing objects inside DLL’s, the class objects don’t have to be tied to any particular program. The DLL and call be called by many applications, distributed or sold as re-usable packages etc. Thus extending the OOP concept reusability to another level!
- In .NET we create DLLs by creating a Class Library Project.
3.5.4 DLL’s in .NET- Class Library Project

- We create DLL in .NET by creating a CLASS LIBRARY PROJECT.
- The Class Library Project will host the Classes of our programs.
- CLASSES CANNOT CONTAIN ANY TYPE OF FORMS OR USER-INTERFACE CODE!!!!
- We will place our Anchored & Unanchored Class Objects inside DLL’s and spread them through our 5-Tier Architecture, primarily in the Business Objects & Data Access BO layers:

![Business Logic Objects](image1)

![Data Access BO](image2)

- Actually each of these two layers can be in different DLL, which makes sense so they can be scalable within the client/server architecture. But that depends on the client architecture you are implementing.

3.5.5 DLL’s and In-Process Communication (UnAnchored Objects)

- A DLL runs inside the client's process or memory space, in other words In-Process or Single Process Communication.
- The DLL is uploaded by the client process or executable and runs in the same memory space.
- DLL’s CANNOT RUN on their own, they are uploaded only during the execution of the client program and live in the same memory space as the client. They need a host.
- In this course, Unanchored or Distributed Objects will run In-Process to the client or User-Interface Layer. Only copies of these objects will be sent to other processes etc.
- The following diagram illustrates this concept:

![Executable Process - In-Process DLL](image3)

- In-Process DLL’s have the following characteristics:

<table>
<thead>
<tr>
<th>Advantages/Characteristics</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Fastest way for classes to interact with the objects since In-Process communication. Less Overhead</td>
<td></td>
</tr>
<tr>
<td>- Code easily shared among applications</td>
<td></td>
</tr>
<tr>
<td>- Easy to evolve, simply distribute new .DLL to all application using them</td>
<td></td>
</tr>
<tr>
<td>- Can NOT serve multiple applications</td>
<td></td>
</tr>
<tr>
<td>- Other process cannot access the DLL that is hosted within the client’s memory space</td>
<td></td>
</tr>
<tr>
<td>- Cannot stand-alone or as an application</td>
<td></td>
</tr>
<tr>
<td>- An updated DLL is not compatible with the previous version</td>
<td></td>
</tr>
</tbody>
</table>
3.5.6 DLL’s and Cross-Process Communication (Anchored Objects)

- For Cross-Process Communication, DLL’s will be used to store the Anchored Objects as well.
- The DLL is uploaded by an executable and runs in the same memory space as the executable.
- In reality, this is the same configuration as the previous Unanchored Object. This is because it is the same. A DLL will always need a HOST or an executable.
- The point is, that if we wish to have an object or objects run on their own in a particular machine, for example in the Three-Tier or Web Based Client/Server architectures the Data Access Layer is placed on the Application Server or in a Web Server. The same applies to the Business Object layer in the Web-Based Client/Server. This means these layers needs to be executables or run on their own as shown in the figure below:

- We also stated earlier that the Data Access Layer will be composed of Anchored Objects
- But In order to be able to place the Data Access BO layer in the Application Server, we need to place the Data Access BO Objects in a DLL.
- But a DLL’s CANNOT RUN on its own, it must have a host executable.
- This means that we need to CREATE AN EXECUTABLE PROCESS TO HOST THE DLL.
- The difference with this project or executable compared to what you have done in class so far, is that this client process will not contain User Interface code or any other UI related code, unlike the Unanchored Object DLL configuration. In this case, it is simply an EMPY executable just to host the DLL and contain any code required to communicate with the other objects in other layers or DLL.
- With this in mind, we have two options available to us in .NET:

  - **Option I** – Create an executable process with the logic to host the DLL, this way the executable can run, hosting the DLL for other processes to access the Anchored Object.
    - This option will require creating a client program with no UI, but will require some additional logic to program the complexities to make the DLL available for other processes.
- **Option II (Easy Option)** – Use Microsoft’s Internet Information Server (IIS) or Web Server to host the DLL that store the Anchored Objects in the Data Access Layer.
  - The IIS Web Server is only being used to serve as an executable for the DLL. It is not being used as an actual Web Server. It is simply fulfilling the necessity of the DLL to have a executable host.
  - This option has the advantage that we don’t have to create a special executable from scratch and the complexities that come with such configuration. IIS takes care of all the complexity.
  - The IIS Service is standard on all Windows 2000 & XP computers.
  - In order to implement this option in .NET, we create an **Empty Web Project**, and add the DLL Class Library project inside the **Empty Web Project**, is that easy!

- So to summarize, Anchored Objects DLL’s will run In-Process to with an IIS Web Server.
- Once we place the Anchored Objects inside a DLL running inside a Web Project, this object is available to other process who wish to communicate with it.
- We now have *Cross-Process communication*.
- The following diagram illustrates this concept:

**Diagram:**

- **Client Process Memory Space**
  - Client/UI Program
  - User Interface Layer (UI)
  - Business Logic Objects
  - Dynamic-Link-Library
  - In-Process Communication

- **Web Server Process Memory**
  - DLL
  - Data Access BO
  - IIS Web Server Service
  - IIS Web Server Executable
  - Cross-Process Communication

**Executable Process - In-Process DLL**
# Chapter 4  Creating DLL’s (Class Library Projects)

## 4.1 Implementing a Component or Dynamic-Link-Libraries

### 4.1.1 DLL’s in .NET (Class Library Project)
- We create DLL in .NET by creating a **Class Library Project**.
- The Class Library Project will host the Classes of our programs.
- We will place our *Anchored & Unanchored* Class Objects inside DLL’s and spread them through our 5-Tier Architecture, primarily in the Business Objects & Data Access BO layers.

### 4.1.2 Placing or Packaging DLLs
- Based on our 5-tier Application Architecture, packaging and placement of the Business Objects in DLL’s will depend on which architecture we are implementing. Let’s look at the possible scenarios:

1. **Placing the UnAnchored & Anchored Objects within their own DLL**

   - This is best suited architectures were both type of objects reside on the same computer.
     - For example **Two-Tier Fat-Client/Thin Server** architectures where all the Business Objects & Data Access Business Objects are **HOSTED** by the **Client** (Suited for our current course projects)
     - **Web-Based Client/Server** where the Business Objects & Data Access Business Objects are **HOSTED** by the **Web Server**:
Note that you can also simply use this packaging method or both tiers in one DLL and distribute them to all the client/server architectures. If a particular component does not required any of the Business Objects Layers they simply just don’t use them.

2. A flexible method is to place the UnAnchored Objects & Anchored Objects in individual DLLs

- This is best suited architectures were both type of objects reside in different computers. Examples of these are:
  - Two-Tier Thin-Client/Fat-Server architectures where Business Logic Objects are HOSTED by the Client and Data Access Objects are HOSTED by the Database Server.
  - Three-Tier Client/Server architectures where Business Logic Objects are HOSTED by the Client and Data Access Objects are HOSTED by the Application Server
- **Web-Based N-Tier Client/Server** where Business Objects are **HOSTED** by the **Web Server** and Data Access Objects are **HOSTED** by the **Application Server**:

![Diagram of Web-Based N-Tier Client/Server](image-url)
4.2 Creating DLL’s - EXAMPLE OF CREATING A GENERAL DLL PROJECT

4.2.1 Creating DLL in .NET

- We create DLL in .NET by creating a Class Library Project.
- The Class Library Project will host the Classes of our programs.
- Creating and using a DLL in .NET involves the following steps:

  I. Creating the DLL Class Library Project & Adding the Classes
  II. Referencing the DLL in client programs or process which want to communicate with the DLL members.
  III. Use the following Syntax from the client to reference the classes in the DLL:
      
      ```
      DLLName.Class
      ```
      Example: BusinessObjectDLL.clsCustomer

- In the following sections we will create three projects to demonstrate the user of DLL’s as follows:

  1. Sample Program 1 – Creating a DLL only. We can create a DLL project that we can distribute at a later time.
  2. Sample Program 2 – Use the DLL project from Sample Program 1 and add a client process to manage it.
  3. Sample Program 3 – Upgrade the previous DictionaryBase Customer Management Application by encapsulating all classes in a DLL.

4.2.2 Sample Program #1 – Create DLL project to host Person Class

- In this example we will create a Class Library Project and create a person class in the DLL.

Example 4.1 – Person Class DLL

Problem statement:
- Create a DLL or Class Library Project. Place inside the DLL a person class. The class has the following requirements:

**DLL & Class Requirements**

- The DLL project should host the following class, data, properties & methods members

  **Class Person Member Data:**
  - Name: Type String
  - IDNumber: Type Integer
  - BirthDate: Date
  - Address: Type String
  - Phone: Type String

  **Class Member Properties & Methods:**
  - Let & Get Properties for each data member.
  - Constructor methods (Default & Parameterized)
  - The Method **Sub Print()**, which prints the Persons data to a text file.

**Business Object requirements:**
- Encapsulating this class in a COMPONENT or DLL will satisfy the first requirements for creating the Business Object layer.
- We will have the following scenario.

![Business Logic Objects and DLL](image-url)
HOW IT’S DONE:

Part I – Create The Class Library Project:

- Before we create the Class Library project, we will take the following approach:
  - Since we are creating a standalone DLL or COMPONENT for DISTRIBUTION, there is no need to create a regular solution, so we will create an **Empty Solution**, just for the purpose of creating the DLL Class Library Project.
  - Once the DLL or Class Library Project is created the DLL file is available for distribution or use in other programs.

- The steps and code are as follows:

**Step 1: Open the Visual Studio IDE and invoke the Start Page & Select New Project**

**Step 2: In the New Project Dialog select Visual Studio Solutions**

- Select Visual Studio Solutions, Select Blank Solution, Name of Solution and browse to location to store Solution:

  - **Step a – Solution**
  - **Step b – Blank Solution**
  - **Step c – Solution Name**
  - **Step d – Project Path**

- This step will create a blank solution as show in the **Solution Explore Window**: 
Step 3: Add a New Project, in Project Dialog Screen select Class Library

- In the Main Menu, select File|Add Project|New Project… In the Project Screen, select Visual Basic Project, Class Library, Name of DLL. Note that the PATH or location will automatically point to the Empty Solution we just created:

Step 4: IDE is invoked

- The IDE is invoked and note that a default class, Class1 is already created for you.
- You can begin to enter data onto this class, or delete it and add classes from previous projects etc.
Step 5: File Structure Created

- In your computer, you should now have a folder for the EMPTY SOLUTION WE CREATED, in this case we named it Component Solution:

![Component Solution folder](image1)

- Inside the Solution, you will find the actual SOLUTION FILES, and Folders for EVERY PROJECT IN THE SOLUTION, in this case we created one project, the BUSINESSOBJECTS DLL or COMPONENT project:

![BusinessObjects folder](image2)

- Inside the BUSINESSOBJECTS project folder you will find all the CLASSES inside the COMPONENT or DLL:
Step 6: Add Code to the Class Module – Business Object Layer

- In the Class Module, begin adding the code for the Business Object.

Step 1: In the Solution Explore, Right-Click and Rename Class1 to clsPerson

```vba
Option Explicit On
Option Strict On

'File Access Class library
Imports System.IO

Public Class clsPerson
```

Step 2: In the Class Module code window enter the code for the private data:

```vba
#Region "Private Data Declarations"
'*******************************************************************************
'Class Data or Variable declarations
Private m_Name As String
Private m_IDNumber As Integer
Private m_BirthDate As Date
Private m_Address As String
Private m_Phone As String
#End Region
```
Step 3: Add the Class Property Procedures:

```vbnet
#Region "Public Properties Declarations"
'*****************************************************************************' 'Property Procedures

Public Property Name() As String
    Get
        Return m_Name
    End Get
    Set(ByVal Value As String)
        m_Name = Value
    End Set
End Property

Public Property IDNumber() As Integer
    Get
        Return m_IDNumber
    End Get
    Set(ByVal Value As Integer)
        m_IDNumber = Value
    End Set
End Property

Public Property BirthDate() As Date
    Get
        Return m_BirthDate
    End Get
    Set(ByVal Value As Date)
        m_BirthDate = Value
    End Set
End Property

Public Property Address() As String
    Get
        Return m_Address
    End Get
    Set(ByVal Value As String)
        m_Address = Value
    End Set
End Property

Public Property Phone() As String
    Get
        Return m_Phone
    End Get
    Set(ByVal Value As String)
        m_Phone = Value
    End Set
End Property
#End Region
```
Step 4: Enter code for Default & Parameterized Constructor Methods:

```vbnet
#Region "Constructor Methods"
'*********************************************************************
'Class Constructor Methods

'Default Constructor
Public Sub New()
'Note that private data members are being initialized
  m_Name = ""
  m_IDNumber = 0
  m_BirthDate = #1/1/1900#
  m_Address = ""
  m_Phone = "(000)-000-0000"
End Sub

'Parameterized Constructor
Public Sub New(ByVal Name As String, ByVal IDNum As Integer, ByVal BDate As Date, _
  ByVal Address As String, ByVal Phone As String)
  Me.Name = Name
  Me.IDNumber = IDNum
  Me.BirthDate = BDate
  Me.Address = Address
  Me.Phone = Phone
End Sub
#End Region

Step 5: Add PrintPerson() Method as follows:

#Region "Regular Methods"
'*********************************************************************
'Class Methods

'*********************************************************************

''''<summary>
''''Prints Person's data to Text File
''''Uses StreamWriter class from System.IO library
''''</summary>
''''<remarks></remarks>
Public Overridable Sub Print()
  'Create StreamWriter Object for append to file listed
  Dim objPrinter As New StreamWriter("PersonPrinter.txt", True)

  'Call StreamWriter Object WriteLine method to write the string to file
  objPrinter.WriteLine(m_Name & "," & m_IDNumber & "," & _
    m_BirthDate & "," & m_Address & "," & m_Phone)

  'Close StreamWriter Object
  objPrinter.Close()
End Sub
#End Region
End Class
```
Step 6: Compile and Build the project. Click Build|Build Solution

Step 7: Build Solution

Step 7: DLL is ready for user

- DLL IS NOW CREATED AND READY TO BE USED.
- WE CAN NOW DISTRIBUTE THIS DLL IS DESIRED. IF YOU LOOK IN THE BIN/DEBUG FOLDER OF THE PROJECT YOU WILL SEE THE DLL FILE:

  ANY CLIENTS OR PROCESS THAT WISH TO COMMUNICATE WITH THE DLL MUST DO THE FOLLOWING:

  1. ADD THE DLL TO THAT SOLUTION OR PROCESS – Copy the DLL File to the project folder
  2. MUST CREATE A REFERENCE TO THE DLL IN ORDER TO BE ABLE TO USE IT

- NOTE THAT AN XML VERSION IS ALSO AVAILABLE
4.2.2 Sample Program #2 – CONVERTING THE Customer Retail Management Application TO USE A DLL (NEEDED FOR CLASS PROJECT)

- In this example upgrade the DictionaryBase Customer Management Application we created in Lecture 2B Sample Program 1.
- We will now upgrade this application by CREATING A NEW SOLUTION and PACKAGING ALL CLASSES IN A COMPONENT or DLL.
- THIS IS THE FIRST STEP IN CONVERTING OUR APPLICATION INTO A TRUE N-TIERED CLIENT/SERVER APPLICATION.

Example 4.2 – DLL Customer Management Application

Problem statement:
- Upgrade the Customer Management application by taking the first step into a true n-tier Client/Server application by placing all classes in a COMPONENT or DLL.

Business Object requirements:
- Use the current classes in the application:
  1. Creating a new Solution to host our application.
  2. ADD/create a Component or DLL project
  3. Copy/paste all classes from your previous Customer Manager project to this new version. ADD, and encapsulate clsPerson, clsCustomer & clsCustomerList within the DLL Project:

Client Process requirements (Presentation/UI Layer):
- The client project is a Module-Driven Windows Application which contains a Customer Management application from previous lecture, which via a Form enables users to retrieve, edit, remove, insert, list, print & print all customers.
- The client project will represent the Presentation/User-Interface Layer of our 5-tier architecture

The Client process is to reference and use the COMPONENT or DLL project with the classes. As expected the DLL will run in the same memory space as the client or In-Process Communication as shown in diagram below:
The high-level steps are as follows:

1. Copy/paste the entire Customer Management application folder under the NEW EMPTY SOLUTION FOLDER.
2. Now the solution contains TWO PROJECTS, THE DLL & EXECUTABLE PROJECT THAT WILL HOST THE DLL.

Implementation Details:
- Before upgrade the application, we need to make some decision on how to implement
- There are two options available as follows:

**OPTION 1 – USE A NEW OR BLANK SOLUTION. MIGRATE ALL PROJECT FILES TO NEW SOLUTION:**

1. Create a NEW BLANK Solution
2. Create and Add a NEW Class Library project
3. COPY/PASTE the entire FOLDER of the existing Customer Management Program to the BLANK solution
4. After, migrate the classes from the Customer Management project to the DLL. Objectives are to have the classes contained within the DLL.

Benefits of this approach are that we have a Generic or BLANK Solution with multiple projects within. The Solution will be the top FOLDER, with two child folders, one for the DLL, & the Client Customer Management project. These Folders are within the Solution Folder as separate projects. Here is an example or view of the folder structure:

- Note how the Solution is in the root directory of the application while each individual project is within its own folder. The Solution is generic and not tied to any particular project.
- Also note how the FOLDER STRUCTURE match the architecture of a CLIENT using a DLL as shown in diagram below:

![Diagram showing client process memory space with client UI program, dynamic-link-library, in-process communication, and executable process - in-process DLL.](image-url)
OPTION 2 – USE EXITING SOLUTION. ADD DLL PROJECT TO EXITING SOLUTION & PROJECT:

1. Use the exiting Solution of the Customer Management application.
2. Add a NEW Class Library project to the EXITING SOLUTION
3. After, migrate the classes from the Customer Management project to the DLL.

- Since the current Customer Management Application is already contained within its Solution named Customer Management, we can simply add the Class Library project or DLL to this solution and migrate all classes from their current location to the Class Library Project
- This approach is OK and valid, but the Solution FILES are within the CLIENT PROJECT. In addition, the DLL project folder will reside INSIDE THE CLIENT PROJECT folder structure as shown in the diagram below:

- This option may seem appealing for your class projects and homework since you already have the client application solution already created from your previous assignments.
- Nevertheless, this is not an elegant way of doing it and does not represent the interaction between the CLIENT PROJECT & DLL.
- Also it is not suitable for the Client/Server architecture model.

In this course, we will use Option 1. It will require some additional work, but is not something difficult, simply some copy/paste operations etc.
HOW IT'S DONE:

Business Object Layer

Part I – Create The EMPTY SOLUTION and Class Library Project:

The steps and code are as follows:

Step 1: Open the Visual Studio IDE and Select New Project

Step 2: In the New Project Dialog select Other Project Types | Visual Studio Solutions

Select Visual Studio Solutions, Select Blank Solution, Name of Solution and browse to location to store Solution:

This step will create a blank solution as show in the Solution Explore Window:

Step a – Solution

Step b – Blank Solution

Step c – Solution Name

Step d – Project Path
Step 3: Add a New Project, in Project Dialog Screen select Class Library

- In the Main Menu, select File|Add Project|New Project… In the Project Screen, select Visual Basic Project, Class Library, Name of DLL. Note that the PATH or location will automatically point to the Empty Solution we just created:

  **Step a – Project type**

  **Step b – Class Library**

  **Step c – Project Name**

  **Step d – Project Path**

  Will automatically point to existing solution

Step 4: IDE is invoked

- The IDE in invoked with Class1 is already created for you.
Step 5: File Structure Created up to NOW

- In your computer, you should now have a folder for the SMALLBUSINESSAPP EMPTY SOLUTION WE CREATED and within the solution, the DLL or COMPONENT PROJECT we just added:

![Image of file structure]

Step 6: Delete Class1

- Since we are going to re-use the classes already created in the Customer Management Application, we can delete the default CLASS1
- The IDE screen looks as follows:

![Image of IDE screen]

- The file structure now looks as follows:

![Image of file structure after deleting Class1]
Part II – Add the CLIENT PROJECT:

Step 7: COPY/PASTE to the Customer Management Client Application from Previous Lecture

- Locate and copy your DictionaryBase Customer Management application project to the NEW SOLUTION:

1. Using your Windows Explore, navigate to the location where you kept your DictionaryBase Customer Management Application used in the Collection Part II notes.
2. Right-click on the project folder and select COPY.
3. Now navigate back to your NEW SOLUTION folder.
4. Right-Click and select PASTE.
5. Now your Customer Management program is now located in the same location as your solution & DLL project. This should look similar to the figure below:

6. I will also rename this folder to CustomerManagementClient, so the name reflects the role of this application:

7. Another view of the FOLDER STRUCTURE:
Step 8: Add the CustomerManagementClient Project to the Current Solution

- Now we need to add this project to the Solution. Perform the following steps:

  1. In the Main Menu, select File|Add Project|Exiting Project… this will invoke the Add Exiting Project dialog.
  2. Navigate to the NEW SOLUTION and the CustomerManagementClient project and select it:

Step 9: The IDE Now has the Client & DLL projects

- The IDE now looks as follows:
Step 10: Migrate the Classes from the Client Project to the DLL.

- Now we need to move the classes to the DLL. **But first, Make sure the default Class1 is removed.** We don’t need it.
- Moving the classes to the DLL there are two methods available:

**Method 1 – Cut/Paste Class Files from Client Project to DLL Project**
1. Cut/Paste or Move the actual class FILES form the CustomerManagementClient program to the DLL project folder
2. Using the IDE, add them to the DLL project.

**Method 2 – SELECT/SHIFT/Drag Class object from Client Project to DLL Project**
- Fortunately, with the .NET IDE, we can simply Click & Drag the classes to the DLL project. The IDE will take care of moving the files for you. So, do the following:
  1. **USING THE IDE/SOLUTION EXPLORER, HOLDING THE CONTROL KEY, CLICK & SELECT EACH CLASS**
  2. **HOLD THE SHIFT KEY DOWN, NOW DRAG** the classes from the Client project and **DROP** them in the DLL project.
  3. **Note that the FILES WILL BE AUTOMATICALLY MOVED AS WELL.**

- The IDE will now look as follows:
Step 11: Select Client Application as the Startup Project

- If you have noticed that in the Solution Explorer, the DLL BusinessObjects Project is in bold typeface. This is because to the Solution, this project is the **Startup Project** since it was created first.
- The Startup Project is the project designated in the solution to execute and control the application.
- But as we know from the lectures in this notes that DLLs CANNOT execute on their own. They need a host to run them.
- Therefore the DLL CANNOT start the project or be the startup object. Only the client can be the Startup.
- You need to set the **Client Application** as the Startup Project which will control this Solution. The client project needs to be the startup since it is the one controlling the DLL. DLL’s cannot execute.
- In the Solution explorer do the following:
  
  a. In the Solution Explorer Window, Right-Click on the client Project, in this case the **CustomerManagementClient** and select **Set as Startup Object**
  
  b. You will now notice that the Client project is highlighted in Bold
In order for the Client Project to see the classes in the DLL, it must set a reference or pointer to the DLL. This is done as follows:

1. In solution explorer, SELECT the Client **Customer Management** Project section. Now in the **Solution Explore Tool Bar**, select the **SHOW ALL FILES BUTTON**. This will show all the files associated with the project and in addition, a folder named **REFERENCES**:

2. **RIGHT-CLICK** on the References Folder & SELECT **Add Reference**... you will invoke the **“Add Reference”** dialog box:
3. Select the *Projects Tab*
4. Select the *Business Objects Project.*
5. Click OK

6. In the Solution Explore, if you expand the Reference Entry, you will see the reference to the Business Object DLL
Part III – Modify the Client Code:

- Now we need to do some modification to the Client program. So far we moved all classes to reside inside the DLL.
- Therefore the current references to the classes from within the Client program need to change, since the classes are no longer in the Client project but in the DLL.
- This means that now when we USE THE OBJECTS in our code we need to access the classes VIA THE DLL. Syntax is as follows:

```
DLLCOMPONENT.Class.Property
DLLCOMPONENT.Class.Metho()
```

Step 13: Modify Module to References to the Class via COMPONENT

- In the module, modification will be required in the declaration section & Initialization method.

Step 1: In the Module Modify the Collection Class Object Declaration

- The Collection Class is stored in the DLL, therefore we need to modify this reference when creating the Collection Class Object
- Currently we have the following:

```
'Declare Public Array of Person Objects
Public CustomerList As New clsCustomerListManager
```

- Based on the syntax, we need the following:

```
'Declare Public Array of Person Objects
Public CustomerList As New BusinessObjects.clsCustomerListManager
```

- Let’s look at the Module Declaration and the Sub Main() method:

```
Option Explicit On
Option Strict On
Module modMainModule

'Declare Public Array of Person Objects
Public objCustomerList As New BusinessObjects.clsCustomerListManager

Dim objMainForm As frnMain = New frnMain

Public Sub Main()

'Perform initialization
InitializeList()

'Display Customer Form
objMainForm.ShowDialog()

End Sub
```
Step 2: Modify the InitializeList Method:

In this example we don’t have any code that references the objects in the Initialize method. But if we did, we would have to call BusinessObjects.Class in order to create the objects:

```vbnet
'************************************************************************************
''' <summary>
''' Name: InitializeList() Method
''' Purpose: Nothing is required for this example
''' </summary>
''' <remarks></remarks>
Public Sub InitializeList()
    'No objects are added to Customer Collection from initialize
    'Since we are storing our Customers in a File, we don’t really
    'want to add Customer object from here! If we do
    'these objects will be stored in the file via Save() and then
    'we will have duplicate objects during the load(), and since we cannot have
    'two objects with the same key we will raise and Exception.

    End Sub

End Module
```
Step 14: Modify the Code to the User Interface Form to Reference the DLL

- Now we make the required modifications on the Form code, where ever objects are created
- Currently we have the following:

**frmMain Form:**
- No reference to classes are done here

**frmCustomerManagement Form:**
- In this form we create class objects, therefore we need to reference the DLL or COMPONENT

![Customers Form](image)

- The required changes will involve the code within the Form to communicate with the classes which are now residing within the DLL.
- In the original client project, the classes were part of the Client process. Now things have changed, the classes are no longer in the client; they are within the DLL and require that we reference them via the DLL using the syntax:
  
  `COMPONENT.Class`

- For example in the Form we may have the following code:

  ```vbscript
  'Declare Form Level POINTER
  Private objCustomer As clsCustomer
  ```

- But now that we have the classes encapsulated in a DLL, we need to use the syntax: DLLName.Class in order to access the class, for example:

  ```vbscript
  'Declare Form Level POINTER
  Private objCustomer As BusinessObjects.clsCustomer
  ```

- So, now we need to search through our Form code and modify all object references to reflect the **BusinessObjects DLL**
Step 1: Open the Form Code Editor and modify the Declaration Section and Form Load() event

- In the Form, there is only two locations where the classes are being referenced in the declaration section & List Event.
- In the Form declaration section. Modify all object references to reflect the BusinessObjects DLL.

```vbnet
Option Explicit On
Option Strict On

Public Class frmCustomerManagement

    'Declare Form Level POINTER
    Private objCustomer As BusinessObjects.clsCustomer

End Class
```

Step 2: Modify the List Section and Form Load() event

- In the LIST EVENT, modify all object references to reflect the BusinessObjects DLL.

```vbnet
Private Sub btnList_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnList.Click
    'Step A- Begin Error trapping
    Try
        'Step 1-Clear the list
        lstCustomers.Items.Clear()

        'Step 2-Create Temporary Person and Dictionary object POINTERS
        Dim objDictionaryEntry As DictionaryEntry
        Dim objItem As BusinessObjects.clsCustomer

        'Step 3-Use For..Each loop to iterate through Collection
        For Each objDictionaryEntry In objCustomerList
            'Step 4-Convert DictionaryEntry pointer returned to Type Person
            objItem = CType(objDictionaryEntry.Value, BusinessObjects.clsCustomer)

            'Step 5-Create the string to list
            Dim strLine As String = objItem.CustomerID & "," & _
            objItem.Name & "," & _
            objItem.SocialSecurity & "," & _
            objItem.BirthDate & "," & _
            objItem.Address & "," & _
            objItem.Phone

            'Step 6-Add string to ListBox
            lstCustomers.Items.Add(strLine)
        Next

        'Step B-Traps for general exceptions.
        Catch objE As Exception
            'Step C-Inform User
            MessageBox.Show(objE.Message)
        End Try
    End Try
End Sub
```
frmRetailManagement Form:
- In this form we also create class objects, therefore we need to reference the DLL or COMPONENT

Step 1: Open the Form Code Editor and modify the Declaration Section and Form Load() event
- In the Form, there is only two locations where the classes are being referenced.
- In the Form declaration section and Form_Load() event. Modify all object references to reflect the BusinessObjects DLL

```vbnet
Option Explicit On
Option Strict On

Public Class frmRetailManagement

' FORM-LEVEL VARIABLES & OBJECT DECLARATIONS SECTION

Private WithEvents objCustomer As BusinessObjects.clsCustomer
```
Step 2: Modify the Form Load() event

In the Form_Load() event, modify all object references to reflect the BusinessObjects DLL.
Step 1: Compile and Build the project.

When you build and compile the program, a COPY of the BusinessObjects DLL is placed in the CLIENT Customer Management project BIN FOLDER. SINCE A REFERENCE WAS SETUP:

![Image of BusinessObjects DLL]

Step 2: Execute the application.

Form Output:

![Image of Customers management form]

![Image of Manager information form]
File Output:

- Screen shots of Customer Data & Customer Printer Files:

![CustomerData.txt - Notepad](image1)

B33, Sam Franks, 333-22-3333, 3/12/1967, 333 Jay Street, 718 260-5333, 0
111, Joe Smith, 111-11-1111, 1/23/1971, 333 Jay Street, 718 260-5000, 97
222, Angel Rod, 222-22-2222, 3/12/1967, 222 Jay Street, 718 260-5000, 0

![CustomerPrinter.txt - Notepad](image2)

Joe Smith, 111-11-1111, 1/23/1971, 333 Jay Street, 718 260-5000, 111
Joe Smith, 111-11-1111, 1/23/1971, 333 Jay Street, 718 260-5000, 111
Angel Rod, 222-22-2222, 3/12/1967, 222 Jay Street, 718 260-5000, 222
4.2.5 Conclusion & Summary of Sample Program 2

In this example we did the following:

1. Created a Generic (Blank) Solution
2. Created a DLL Project within the Solution
3. Added our previous Customer Management Application to the Solution and named it Customer Management Client.
4. We migrated all classes from the Client project to the DLL. In effect we implemented the following DLL packaging:

![Diagram of DLL containing Business Objects Layer only](image)

5. We then proceeded to modify the Client Process to reference the DLL when accessing the classes

The main point is that we have accomplished the following:

1. We meet A VERY IMPORTANT REQUIREMENT for Business Objects; we PACKAGED ALL CLASSESS in a DLL COMPONENT.

2. We HOSTED the DLL using the Customer Management client process therefore in-process communication is taking place:

![Diagram of Client Process Memory Space](image)

3. WE IMPLEMENTED THE FIRST TWO LAYERS of the Three-Layer Application Architecture:
- Overall, we implemented a **Single-Tier Client/Server Application** since we only have ONE PROCESS running which is the Client Process hosting the DL:

![Diagram showing Single-Tier Client/Server Application](image)

- Note that we have not yet created the **Data Access Code in our Business Objects**. In addition, for this type of architecture, the **Database Layer** must be a flat file, or a flat file desktop database such as MS Access, Fox Pro, etc.