CS708 Project 1 – Video Management System

(Version 3.0)

Take Home Project – Professor A. Rodriguez

Problem Statement

- You re-hired again as a consultant, by ***NYCTech Solutions Inc***. (Represented by Prof. Rodriguez) to **UPDATE** the **Video Management System** application to **VERSION 3.0**. Detailed requirements a listed in sections below.

- As in the previous employment contract, the *client* (Prof. Rodriguez) has the right to **fire you** (Fail you) at any time or reduce the amount of payment (Low Grade) during the development of the system, if the system does not work, you fall behind schedule (Late) and most important **NOT** following the program ***REQUIREMENTS***. **Warning!** NOT following the requirements can quickly lead to the **unemployment line**!

General Programming Requirements

New Features & Functionality

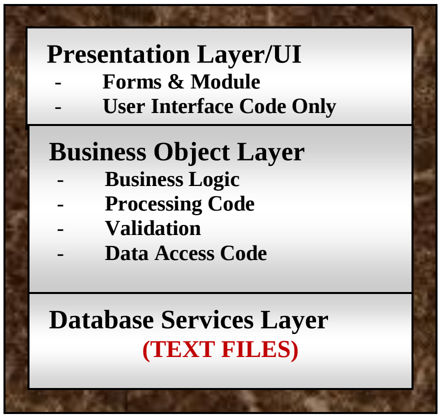
- In the previous **versions 1 & 2**, the objectives were to implement the following back-end management features and functionality only:
  - Back-end Management System
    - ***Customer Management***,
    - ***DVD Management***
    - ***Video Game Management***
    - ***Employee Management***

- Each of these modules or features leveraged text files to store each of the transaction data:
  - **EmployeeData.txt**. – Stores all ***Employee Management*** data.
  - **CustomerData.txt**. – Stores all ***Customer Management*** data.
  - **VideoGameData.txt**. – Stores all ***Video Game Management*** data.
  - **DVDData.txt**. – Stores all ***DVD Management*** data.

- In this version there will be NO NEW UPGRADE IN FEATURES AND FUNCTIONALITY.
- **Keep all features and functionality from the previous Version 2.0 as shown above.**

Application Architecture

- We will continue to use the ***3-tiered Client/Server Application Architecture*** as in Version 2.
- In version 1 & 2, the application was be design with FULL NETWORK expansion in mind; the design should use an application architecture using Object-Oriented-Programming Technology that will enable such growth:



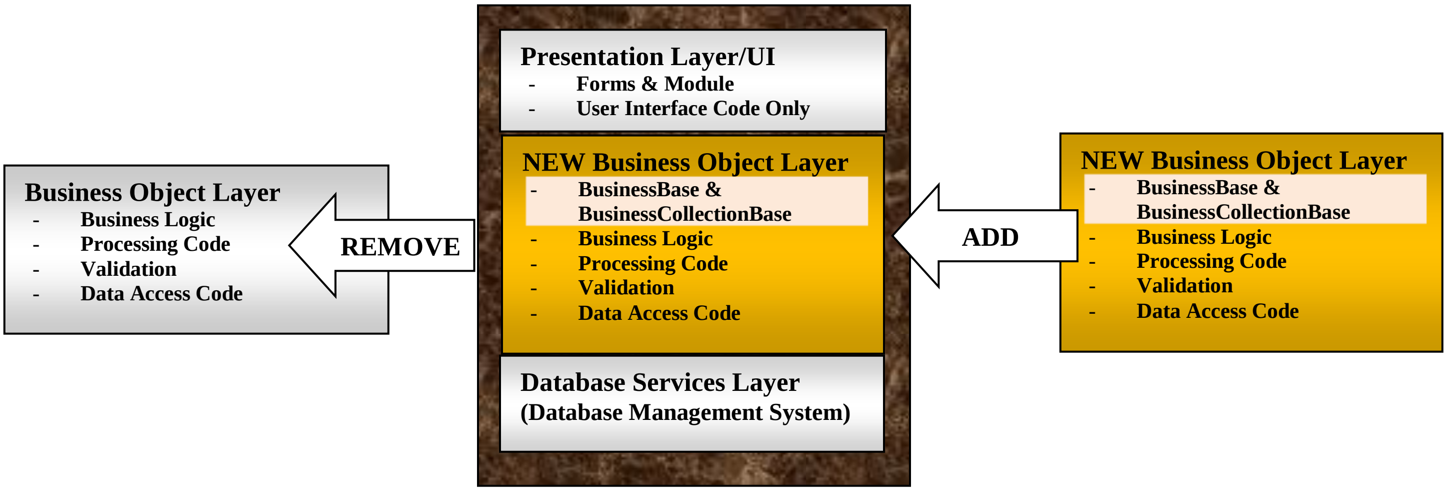
Business Object Layer Upgrade

- ❑ The **Class/Object** model has been UPGRADED to provide for **FULL DATABASE SUPPORT** in the BUSINESS OBJECT LAYER.
- ❑ Continue to implement the ***3-tiered Client/Server Application Architecture***, nevertheless UPGRADE as follows:

  - ▪ IMPLEMENT a NEW BUSINESS OBJECT LAYER as follows:

    - ○ Converting all regular classes to ***BUSINESS CLASSES*** by inheriting from **BUSINESSBASE** Class & modifying accordingly.
    - ○ Converting all Collection Classes to ***BUSINESS COLLECTION CLASSES*** by inheriting from **BUSINESSCOLLECTIONBASE** Class and modifying accordingly.

      - ➢ This is an example of why this architecture is so flexible. We are able to change the middle or ***Business Object Layer*** with a new one with little or no modification to the ***Presentation/User-Interface Layer*** or ***Database layers***



Conclusion

- ❑ You can add any functionality you required to application & the object model, **BUT you should NOT remove any of the current object model class specifications, without the approval of the contractor (Prof Rodriguez)!**
- ❑ **There are a total of 6 Detailed Requirements you must satisfy to get an FULL PAYMENT (A GRADE) in this project**.

Detailed Requirements

- The following sections list the individual detailed requirements of the application upgrade.

Requirement #1 – Upgrade Back-End Management System but Keep all Functionality of CST608 Project 2

- In this third version of the project (CST4708 PROJECT 1), you are to UPGRADE the following:
  - Back-end Management System
    - *Customer Management*,
    - *DVD Management*
    - *Video Game Management*
    - *Employee Management*

- All features and functionality of Project 1 should work in CST3608 Project 2.
- We are not taking away functionality but upgrading with better business object layer mechanism etc., and adding Data Access Code to retrieve and save data in a Database (MS Access).

Requirement #2 – No User-interface Code inside the Business Object Layer & No Processing Code inside the User-Interface Layer

- **ABSOLUTELY NO USER-INTERFACE CODE INSIDE THE BUSINESS OBJECT LAYER OR CLASSES**

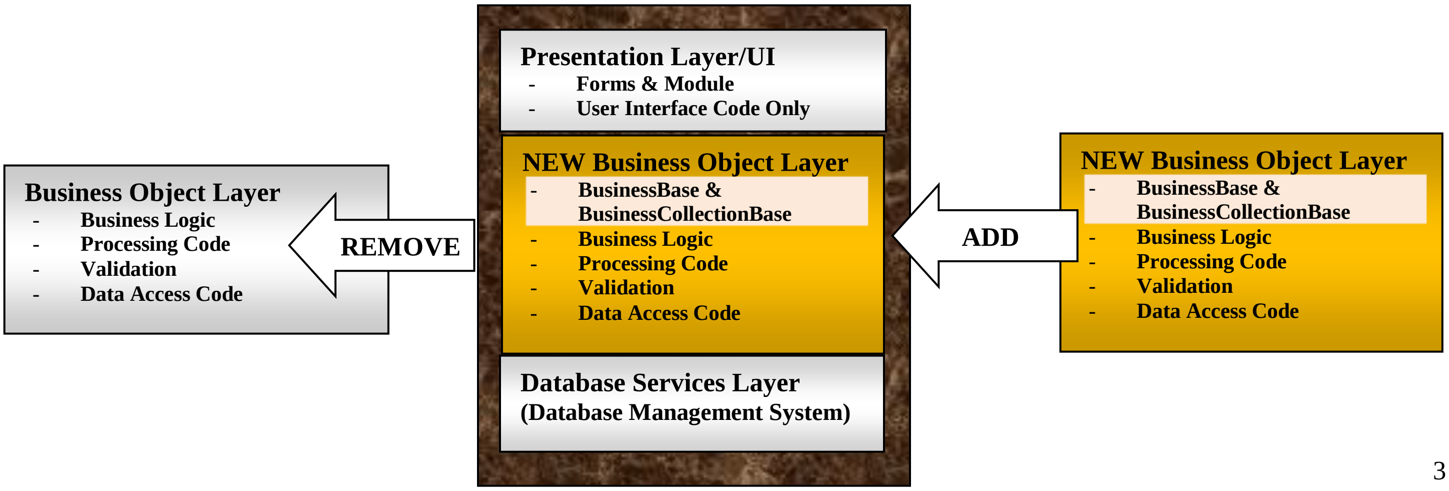
- **ABSOLUTELY NO PROCESSING-CODE INSIDE THE USER-INTERFACE LAYER (FORMS & MODULES)**

- THESE REQUIREMENTS ARE FROM PROJECT #1, BUT MUST BE FOLLOWED FOR ALL VERSION OF THE APPLICATION.
- The Business Objects Layer should handle only processing code and no user-interface code.
- In situation in which you need to flag an issue or communicate outside of the objects, `Throw` a specialized Exceptions such as an `ApplicationException` or `NotSupportedException`.
- The Forms and Module should only perform User-Interface code, that is get input request from user, call the Business Object layer to perform the processing and display data results to user.

Requirement #3 – In the Three-Layer Scalable Application Architecture, UPGRADE the BusinessObject Layer by Converting ALL CLASSES to Business Classes & Business Collection Classes

Upgrade The Application Architecture Business Object's Layer

- We are going to swap the Bussiness Object Layer with a new one by UPGRADING ALL CLASSES to ***BUSINESS CLASSES*** & ***BUSINESS COLLECTION CLASSES*** follows:

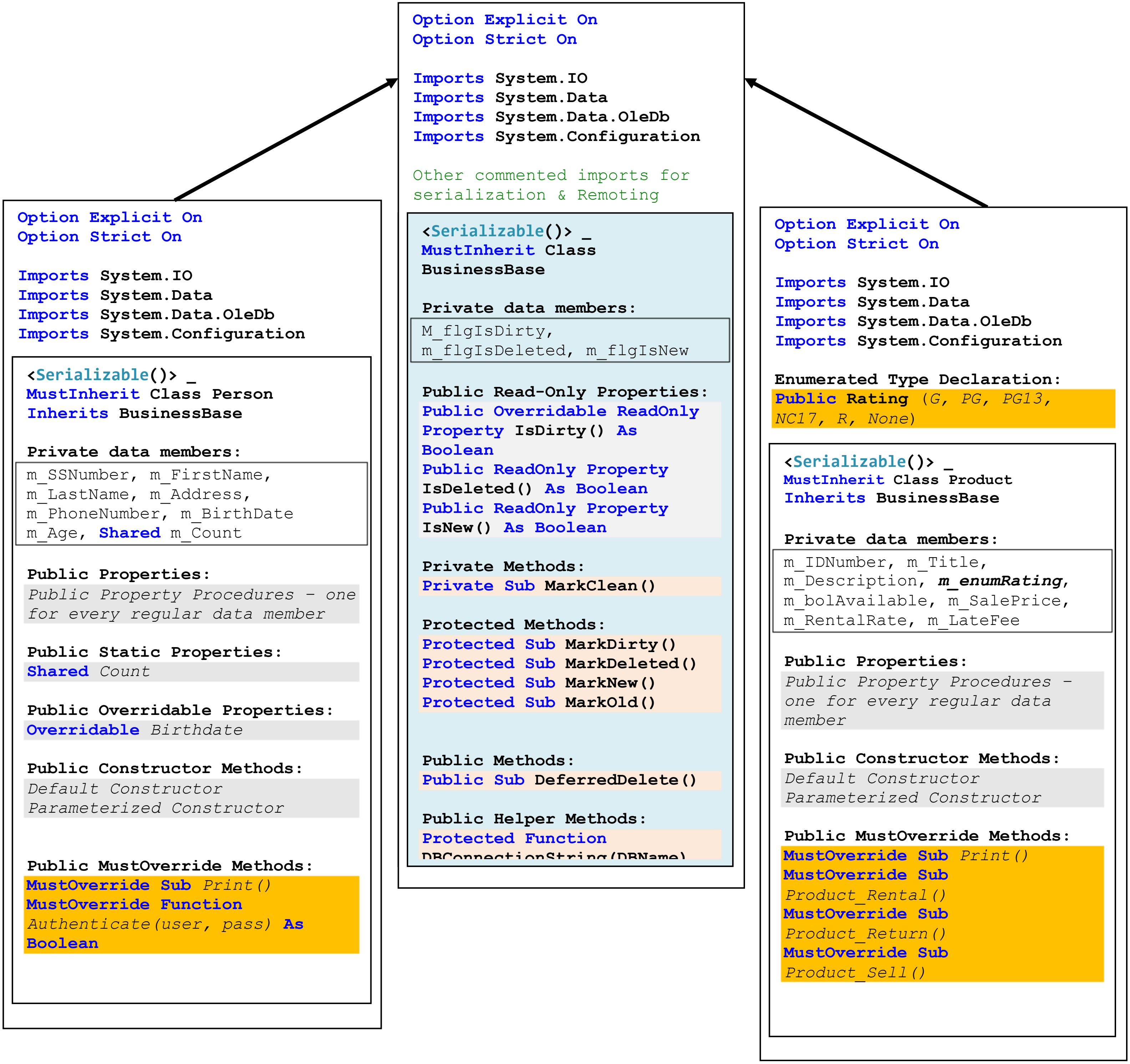


- ❑ This section lists the required classes needed to implement this phase of the project.

Business Object Layer

Business Object Architecture & Class Object Model:

1) **UPGRADE** The Object-Model with the following CLASS INHERITANCE hierarchy between ***BusinessBase***, ***Person*** & ***Product***:

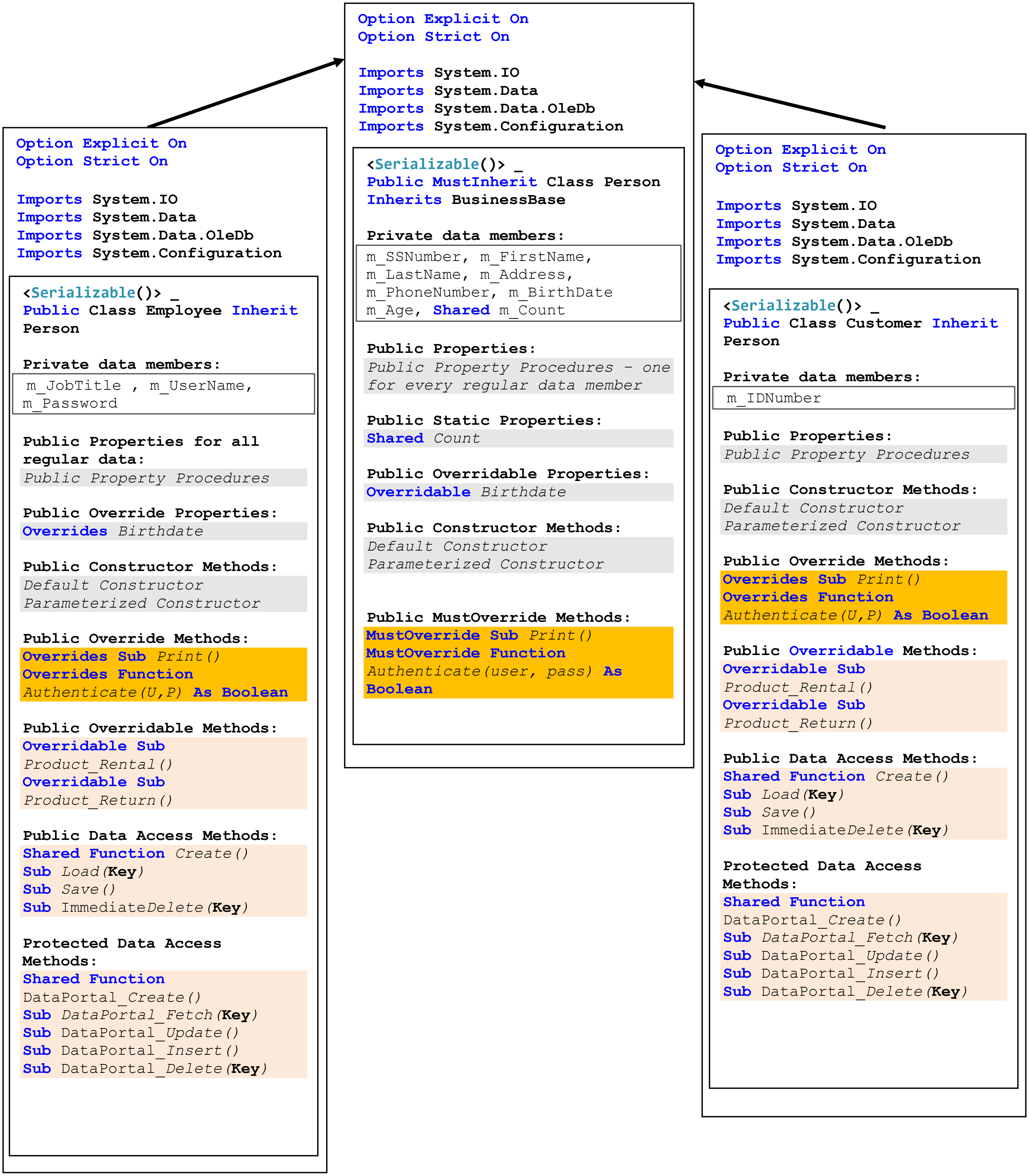


```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

Other commented imports for
serialization & Remoting

<Serializable()> _
MustInherit Class
BusinessBase

Private data members:
M_flgIsDirty,
m_flgIsDeleted, m_flgIsNew

Public Read-Only Properties:
Public Overridable ReadOnly
Property IsDirty() As
Boolean
Public ReadOnly Property
IsDeleted() As Boolean
Public ReadOnly Property
IsNew() As Boolean

Private Methods:
Private Sub MarkClean()

Protected Methods:
Protected Sub MarkDirty()
Protected Sub MarkDeleted()
Protected Sub MarkNew()
Protected Sub MarkOld()


Public Methods:
Public Sub DeferredDelete()

Public Helper Methods:
Protected Function
DBConnectionString(DBName)
```

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
MustInherit Class Person
Inherits BusinessBase

Private data members:
m_SSNumber, m_FirstName,
m_LastName, m_Address,
m_PhoneNumber, m_BirthDate
m_Age, Shared m_Count

Public Properties:
Public Property Procedures – one
for every regular data member

Public Static Properties:
Shared Count

Public Overridable Properties:
Overridable Birthdate

Public Constructor Methods:
Default Constructor
Parameterized Constructor


Public MustOverride Methods:
MustOverride Sub Print()
MustOverride Function
Authenticate(user, pass) As
Boolean
```

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

Enumerated Type Declaration:
Public Rating (G, PG, PG13,
NC17, R, None)

<Serializable()> _
MustInherit Class Product
Inherits BusinessBase

Private data members:
m_IDNumber, m_Title,
m_Description, m_enumRating,
m_bolAvailable, m_SalePrice,
m_RentalRate, m_LateFee

Public Properties:
Public Property Procedures –
one for every regular data
member

Public Constructor Methods:
Default Constructor
Parameterized Constructor

Public MustOverride Methods:
MustOverride Sub Print()
MustOverride Sub
Product_Rental()
MustOverride Sub
Product_Return()
MustOverride Sub
Product_Sell()
```

2) **UPGRADE** The Object-Model with the following CLASS INHERITANCE hierarchy between ***Person***, ***Employee*** & ***Customer***:

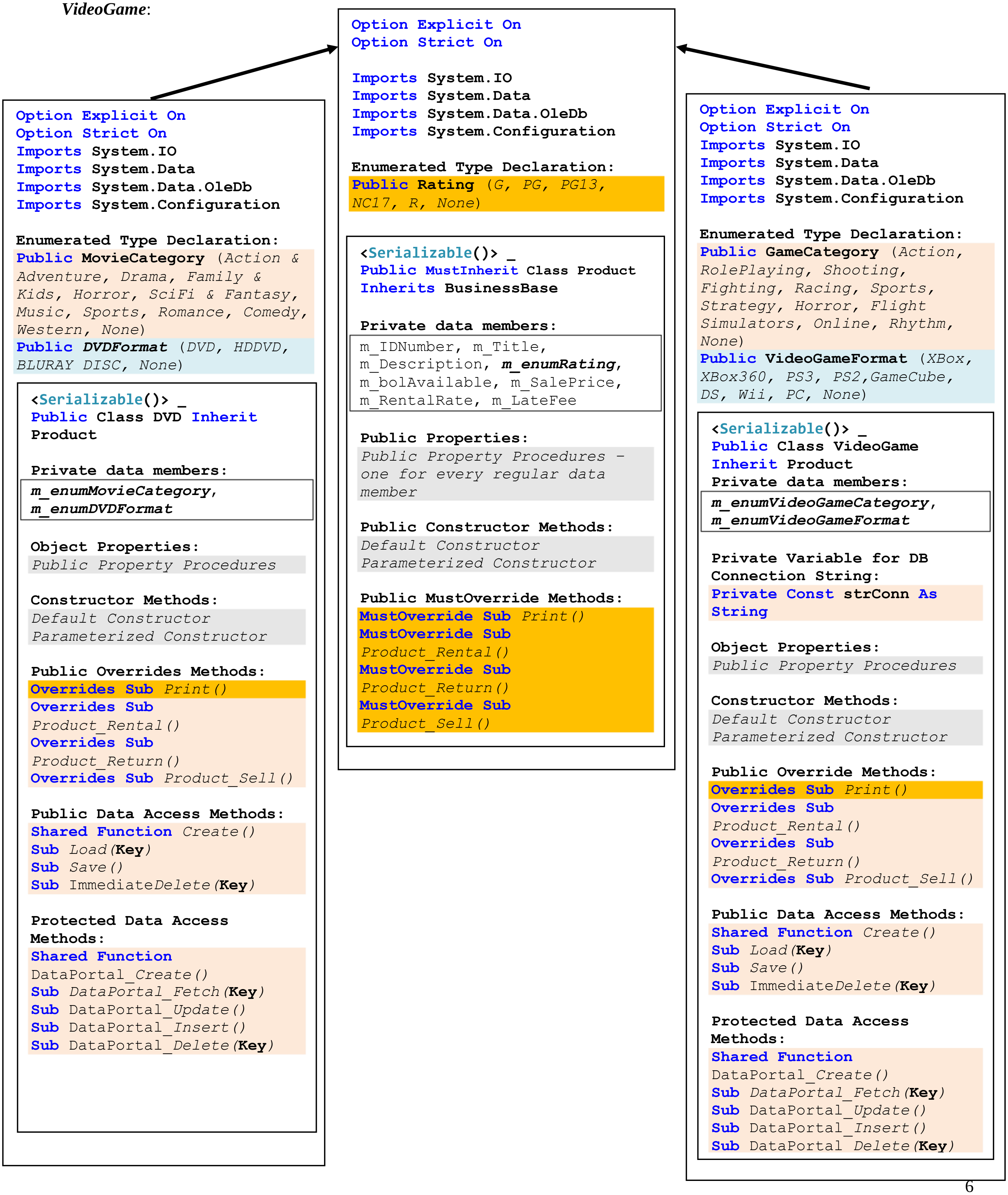


```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
Public Class Employee Inherit
Person

Private data members:
 m_JobTitle , m_UserName,
m_Password

Public Properties for all
regular data:
Public Property Procedures

Public Override Properties:
Overrides Birthdate

Public Constructor Methods:
Default Constructor
Parameterized Constructor

Public Override Methods:
Overrides Sub Print()
Overrides Function
Authenticate(U,P) As Boolean

Public Overridable Methods:
Overridable Sub
Product_Rental()
Overridable Sub
Product_Return()

Public Data Access Methods:
Shared Function Create()
Sub Load(Key)
Sub Save()
Sub ImmediateDelete(Key)

Protected Data Access
Methods:
Shared Function
DataPortal_Create()
Sub DataPortal_Fetch(Key)
Sub DataPortal_Update()
Sub DataPortal_Insert()
Sub DataPortal_Delete(Key)
```

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
Public MustInherit Class Person
Inherits BusinessBase

Private data members:
m_SSNumber, m_FirstName,
m_LastName, m_Address,
m_PhoneNumber, m_BirthDate
m_Age, Shared m_Count

Public Properties:
Public Property Procedures - one
for every regular data member

Public Static Properties:
Shared Count

Public Overridable Properties:
Overridable Birthdate

Public Constructor Methods:
Default Constructor
Parameterized Constructor


Public MustOverride Methods:
MustOverride Sub Print()
MustOverride Function
Authenticate(user, pass) As
Boolean
```

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
Public Class Customer Inherit
Person

Private data members:
 m_IDNumber

Public Properties:
Public Property Procedures

Public Constructor Methods:
Default Constructor
Parameterized Constructor

Public Override Methods:
Overrides Sub Print()
Overrides Function
Authenticate(U,P) As Boolean

Public Overridable Methods:
Overridable Sub
Product_Rental()
Overridable Sub
Product_Return()

Public Data Access Methods:
Shared Function Create()
Sub Load(Key)
Sub Save()
Sub ImmediateDelete(Key)

Protected Data Access
Methods:
Shared Function
DataPortal_Create()
Sub DataPortal_Fetch(Key)
Sub DataPortal_Update()
Sub DataPortal_Insert()
Sub DataPortal_Delete(Key)
```

3) **UPGRADE** The Object-Model with the following CLASS INHERITANCE hierarchy to represent the ***Product***, ***DVD*** and ***VideoGame***:

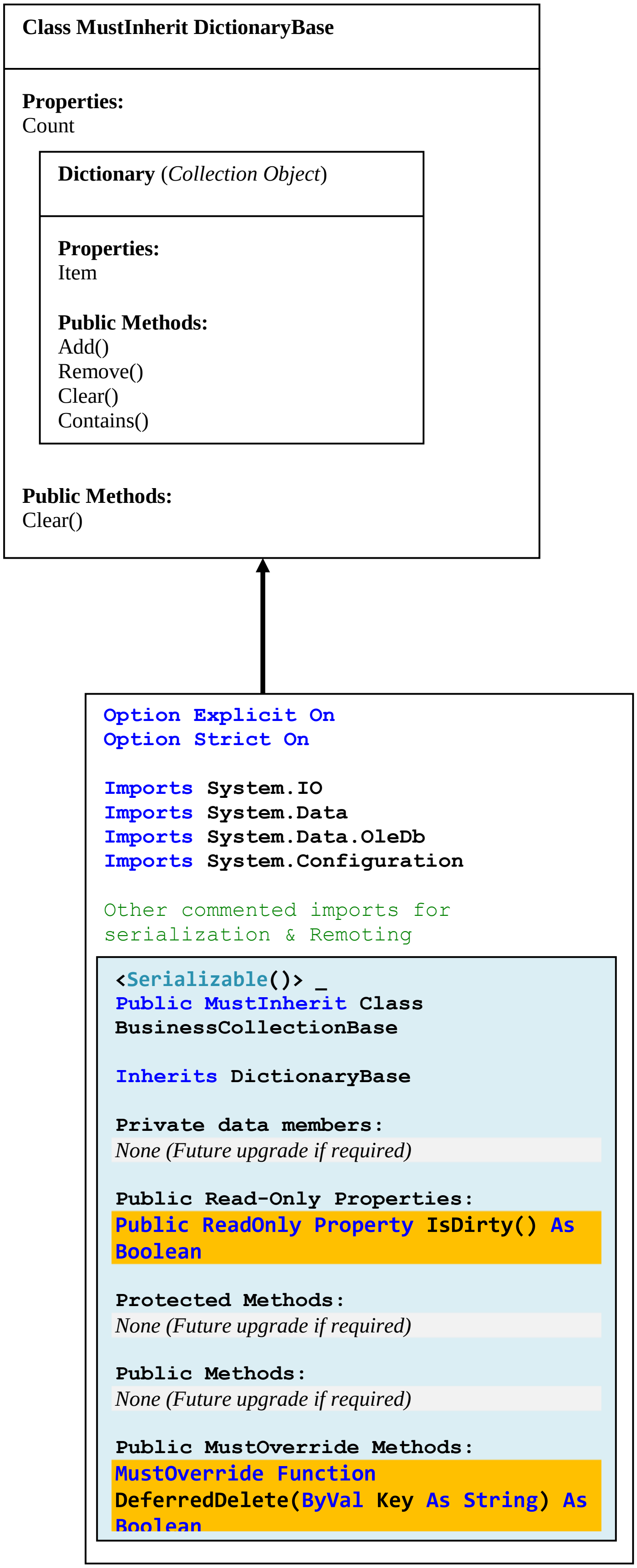


```
Option Explicit On
Option Strict On
Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

Enumerated Type Declaration:
Public MovieCategory (Action &
Adventure, Drama, Family &
Kids, Horror, SciFi & Fantasy,
Music, Sports, Romance, Comedy,
Western, None)
Public DVDFormat (DVD, HDDVD,
BLURAY DISC, None)

<Serializable()> _
Public Class DVD Inherit
Product

Private data members:
m_enumMovieCategory,
m_enumDVDFormat

Object Properties:
Public Property Procedures

Constructor Methods:
Default Constructor
Parameterized Constructor

Public Overrides Methods:
Overrides Sub Print()
Overrides Sub
Product_Rental()
Overrides Sub
Product_Return()
Overrides Sub Product_Sell()

Public Data Access Methods:
Shared Function Create()
Sub Load(Key)
Sub Save()
Sub ImmediateDelete(Key)

Protected Data Access
Methods:
Shared Function
DataPortal_Create()
Sub DataPortal_Fetch(Key)
Sub DataPortal_Update()
Sub DataPortal_Insert()
Sub DataPortal_Delete(Key)
```

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

Enumerated Type Declaration:
Public Rating (G, PG, PG13,
NC17, R, None)

<Serializable()> _
Public MustInherit Class Product
Inherits BusinessBase

Private data members:
m_IDNumber, m_Title,
m_Description, m_enumRating,
m_bolAvailable, m_SalePrice,
m_RentalRate, m_LateFee

Public Properties:
Public Property Procedures –
one for every regular data
member

Public Constructor Methods:
Default Constructor
Parameterized Constructor

Public MustOverride Methods:
MustOverride Sub Print()
MustOverride Sub
Product_Rental()
MustOverride Sub
Product_Return()
MustOverride Sub
Product_Sell()
```

```
Option Explicit On
Option Strict On
Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

Enumerated Type Declaration:
Public GameCategory (Action,
RolePlaying, Shooting,
Fighting, Racing, Sports,
Strategy, Horror, Flight
Simulators, Online, Rhythm,
None)
Public VideoGameFormat (XBox,
XBox360, PS3, PS2,GameCube,
DS, Wii, PC, None)

<Serializable()> _
Public Class VideoGame
Inherit Product
Private data members:
m_enumVideoGameCategory,
m_enumVideoGameFormat

Private Variable for DB
Connection String:
Private Const strConn As
String

Object Properties:
Public Property Procedures

Constructor Methods:
Default Constructor
Parameterized Constructor

Public Override Methods:
Overrides Sub Print()
Overrides Sub
Product_Rental()
Overrides Sub
Product_Return()
Overrides Sub Product_Sell()

Public Data Access Methods:
Shared Function Create()
Sub Load(Key)
Sub Save()
Sub ImmediateDelete(Key)

Protected Data Access
Methods:
Shared Function
DataPortal_Create()
Sub DataPortal_Fetch(Key)
Sub DataPortal_Update()
Sub DataPortal_Insert()
Sub DataPortal_Delete(Key)
```

4) **UPGRADE** The Object-Model with the following CUSTOM COLLECTION CLASS ***Inheritance*** structure between the **.NET *Library DictionaryBase*** class and the ***BusinessCollectionBase*** Class:



Class MustInherit DictionaryBase

Properties:
Count

Dictionary (*Collection Object*)

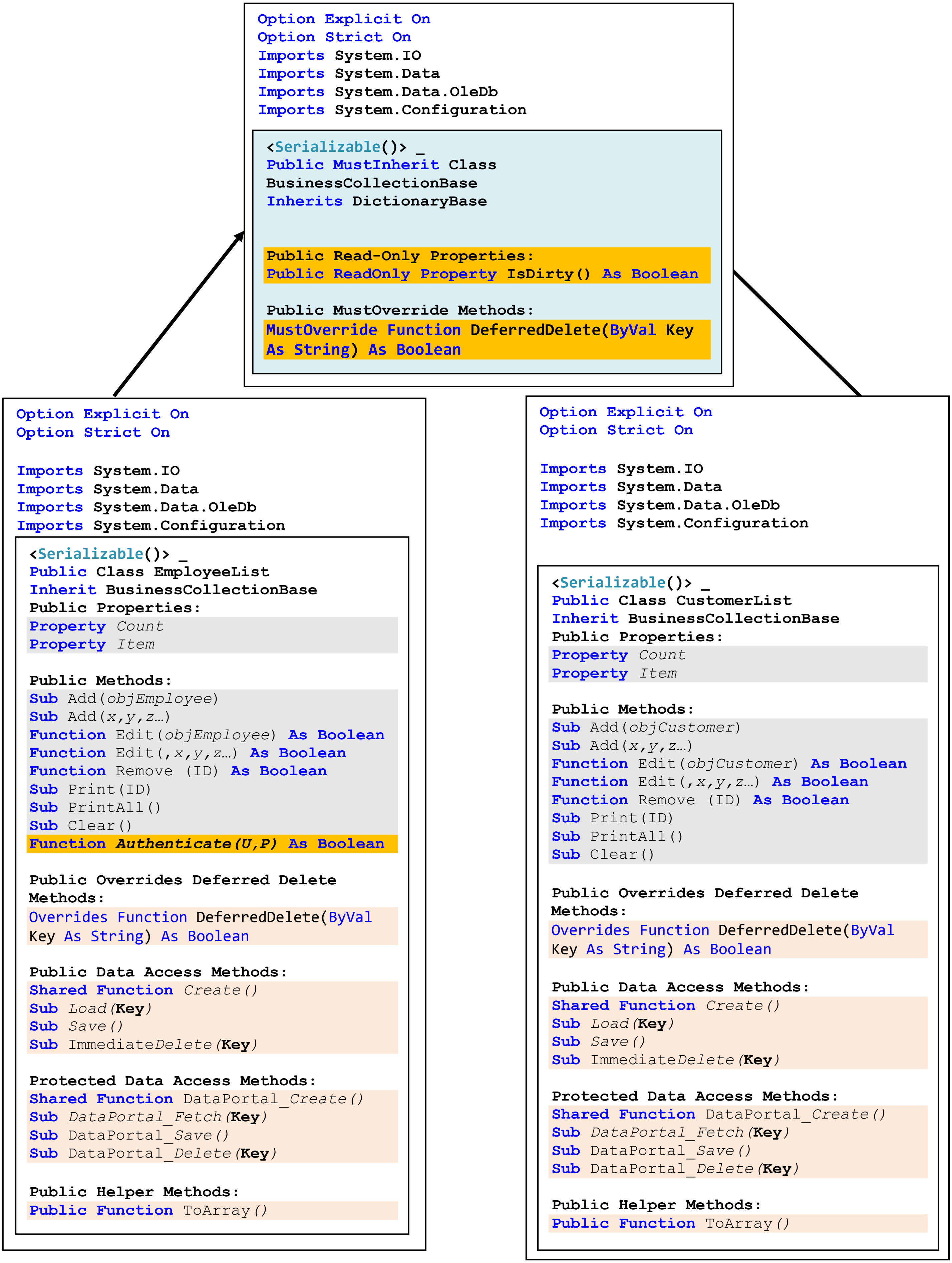
Properties:
Item

Public Methods:
Add()
Remove()
Clear()
Contains()

Public Methods:
Clear()

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

Other commented imports for
serialization & Remoting

<Serializable()> _
Public MustInherit Class
BusinessCollectionBase

Inherits DictionaryBase

Private data members:
None (Future upgrade if required)

Public Read-Only Properties:
Public ReadOnly Property IsDirty() As
Boolean

Protected Methods:
None (Future upgrade if required)

Public Methods:
None (Future upgrade if required)

Public MustOverride Methods:
MustOverride Function
DeferredDelete(ByVal Key As String) As
Boolean
```

5) **UPGRADE** The Object-Model with the following CUSTOM COLLECTION CLASS ***Inheritance*** structure between the ***BusinessCollectionBase***, ***EmployeeList*** & ***CustomerList*** classes:

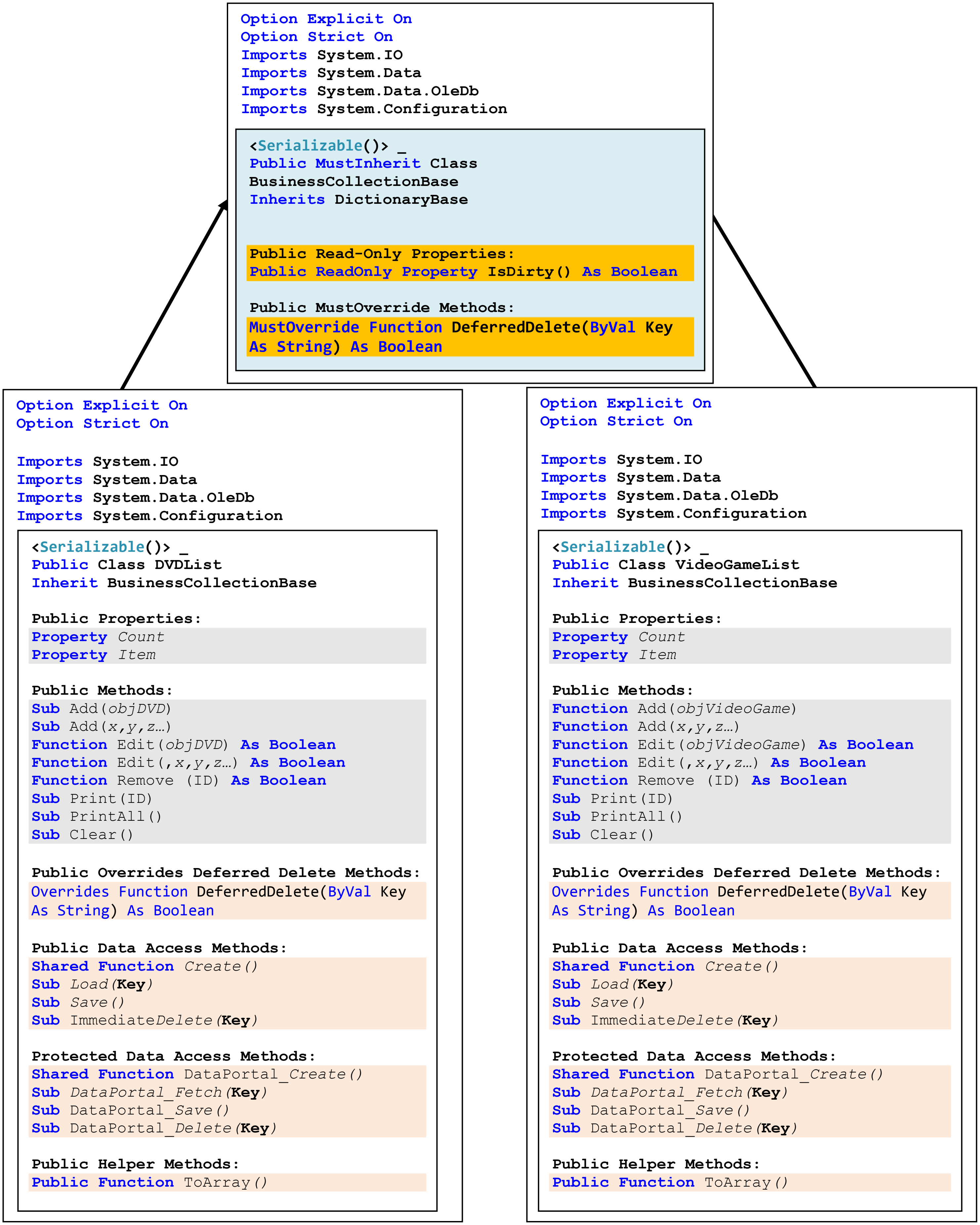


```
Option Explicit On
Option Strict On
Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
Public MustInherit Class
BusinessCollectionBase
Inherits DictionaryBase


Public Read-Only Properties:
Public ReadOnly Property IsDirty() As Boolean

Public MustOverride Methods:
MustOverride Function DeferredDelete(ByVal Key
As String) As Boolean
```

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
Public Class EmployeeList
Inherit BusinessCollectionBase
Public Properties:
Property Count
Property Item

Public Methods:
Sub Add(objEmployee)
Sub Add(x,y,z…)
Function Edit(objEmployee) As Boolean
Function Edit(,x,y,z…) As Boolean
Function Remove (ID) As Boolean
Sub Print(ID)
Sub PrintAll()
Sub Clear()
Function Authenticate(U,P) As Boolean

Public Overrides Deferred Delete
Methods:
Overrides Function DeferredDelete(ByVal
Key As String) As Boolean

Public Data Access Methods:
Shared Function Create()
Sub Load(Key)
Sub Save()
Sub ImmediateDelete(Key)

Protected Data Access Methods:
Shared Function DataPortal_Create()
Sub DataPortal_Fetch(Key)
Sub DataPortal_Save()
Sub DataPortal_Delete(Key)

Public Helper Methods:
Public Function ToArray()
```

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
Public Class CustomerList
Inherit BusinessCollectionBase
Public Properties:
Property Count
Property Item

Public Methods:
Sub Add(objCustomer)
Sub Add(x,y,z…)
Function Edit(objCustomer) As Boolean
Function Edit(,x,y,z…) As Boolean
Function Remove (ID) As Boolean
Sub Print(ID)
Sub PrintAll()
Sub Clear()

Public Overrides Deferred Delete
Methods:
Overrides Function DeferredDelete(ByVal
Key As String) As Boolean

Public Data Access Methods:
Shared Function Create()
Sub Load(Key)
Sub Save()
Sub ImmediateDelete(Key)

Protected Data Access Methods:
Shared Function DataPortal_Create()
Sub DataPortal_Fetch(Key)
Sub DataPortal_Save()
Sub DataPortal_Delete(Key)

Public Helper Methods:
Public Function ToArray()
```

6) **UPGRADE** The Object-Model with the following CUSTOM COLLECTION CLASS ***Inheritance*** structure between the ***BusinessCollectionBase***, ***DVDList*** & ***DVDList*** classes:



```
Option Explicit On
Option Strict On
Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
Public MustInherit Class
BusinessCollectionBase
Inherits DictionaryBase


Public Read-Only Properties:
Public ReadOnly Property IsDirty() As Boolean

Public MustOverride Methods:
MustOverride Function DeferredDelete(ByVal Key
As String) As Boolean
```

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
Public Class DVDList
Inherit BusinessCollectionBase

Public Properties:
Property Count
Property Item

Public Methods:
Sub Add(objDVD)
Sub Add(x,y,z…)
Function Edit(objDVD) As Boolean
Function Edit(,x,y,z…) As Boolean
Function Remove (ID) As Boolean
Sub Print(ID)
Sub PrintAll()
Sub Clear()

Public Overrides Deferred Delete Methods:
Overrides Function DeferredDelete(ByVal Key
As String) As Boolean

Public Data Access Methods:
Shared Function Create()
Sub Load(Key)
Sub Save()
Sub ImmediateDelete(Key)

Protected Data Access Methods:
Shared Function DataPortal_Create()
Sub DataPortal_Fetch(Key)
Sub DataPortal_Save()
Sub DataPortal_Delete(Key)

Public Helper Methods:
Public Function ToArray()
```

```
Option Explicit On
Option Strict On

Imports System.IO
Imports System.Data
Imports System.Data.OleDb
Imports System.Configuration

<Serializable()> _
Public Class VideoGameList
Inherit BusinessCollectionBase

Public Properties:
Property Count
Property Item

Public Methods:
Function Add(objVideoGame)
Function Add(x,y,z…)
Function Edit(objVideoGame) As Boolean
Function Edit(,x,y,z…) As Boolean
Function Remove (ID) As Boolean
Sub Print(ID)
Sub PrintAll()
Sub Clear()

Public Overrides Deferred Delete Methods:
Overrides Function DeferredDelete(ByVal Key
As String) As Boolean

Public Data Access Methods:
Shared Function Create()
Sub Load(Key)
Sub Save()
Sub ImmediateDelete(Key)

Protected Data Access Methods:
Shared Function DataPortal_Create()
Sub DataPortal_Fetch(Key)
Sub DataPortal_Save()
Sub DataPortal_Delete(Key)

Public Helper Methods:
Public Function ToArray()
```

Object Model Requirements Summary (CLASS DETAILS IN APPENDIX A)

- As shown in the Object-Model, Class requirements and details:
    - You are required to ADD the ***BusinessBase*** & ***BussinessCollection*** Classes to the DLL Project. These classes will be provided by the Professor. Details in APPENDIX A.

    - Inherit all Business & BussinessCollection Classes from ***BussinessBase*** & ***BusinessCollectionBase*** and **UPGRADE** the these **10** classes from version 2 as follows;

        - **Person**, **Employee**, **Customer**, **Product**, **DVD** and **VideoGame** all become ***Business Classes*** via INHERITANCE from ***BusinessBase*** Class. See **Architecture Diagram** above and details in **APPENDIX**
        - **CustomerList, EmployeeList, DVDList & VideoGameList** all become ***Business Collection Classes*** via INHERITANCE from ***BussinessCollectionBase*** Class. See **Architecture Diagram** above and details in **APPENDIX A**.

    - IMPORTANT! Follow this object model to the letter, the methods, properties and data should be **NAMED** as shown in diagram.
    - DO NOT CHANGE THE OBJECT MODEL UNLESS YOU HAVE A VALID REASON AND PERMISSION FROM THE PROJECT MANAGER (Prof Rodriguez

    - **OPTION STRICT & EXPLICIT = ON for all classes.**

    - **APPENDIX SECTION OF THIS DOCUMENT PROVIDES THE DETAILED DESCRIPTION AND DATA, PROPERTIES & METHODS OF EACH OF THE CLASSES. BASE YOU CLASSES ON THE INFORMATION PROVIDED IN THE APPENDIX A**.

BusinessBase & Business Classes Summary (CLASS DETAILS IN APPENDIX A)

- Detailed explanation of the ***BussinessBase*** & BusinessCollectionBase and all other classes is found **IN APPENDIX A**:

BusinessBase Class Explanation:

- The ***BusinessBase*** Class details are as follows:
    1. **`MustInherit`** class which will serve as the BASE CLASS to CONVERT ALL REGULAR CLASSES TO BUSINESS CLASSES
    2. ***Details in APPENDIX A***.

BusinessCollectionBase Class Explanation:

- The ***BusinessCollectionBase*** Class details are as follows:
    1. **`MustInherit`** class which will serve as the BASE CLASS to CONVERT ALL COLLECTION CLASSES TO BUSINESS COLLECTION CLASSES
    2. Implements the DIRTY, NEW & DEFERRED DELETED Mechanism.
    3. ***Details in APPENDIX A***.

Person Class Explanation:

- The ***Person*** Class details are as follows:
    1. **`MustInherit`** class which will serve as the BASE CLASS used as template for creating all employees and customers.
    2. Same functionality as PROJECT #1**, but MOFIDIED AS A BUSINESS CLASS.**
    3. ***Modified as specified in APPENDIX A***

Employee Class Explanation:

- The ***Employee*** Class details are as follows:
    1. ***Employee Class*** **`Inherits`** from ***Person*** & represents the **Employees** of the Business.
    2. Same functionality as PROJECT #1.
    3. ***Modified as specified in APPENDIX A***

Customer Class Explanation:

- The ***Customer*** class details are as follows:
  1. ***Customer Class*** **`Inherits`** from ***Person*** & represents the **Customers** of the Business
  2. Same functionality as PROJECT #1**, but MOFIDIED AS A BUSINESS CLASS.**
  3. ***Modified as specified in APPENDIX A***

Product Class Explanation:

- The ***Product*** Class requirements are as follows:
  1. **`MustInherit`** class which will serve as the BASE CLASS used as template for creating all ***DVD*** & ***VideoGame*** Classes.
  2. Same functionality as PROJECT #1.
  3. ***Modified as specified in APPENDIX A***

DVD Class Explanation:

- The ***DVD*** Class requirements are as follows:
  1. ***DVD Class*** **`Inherits`** from ***Product*** & represents the **DVD** rented and sold in the Business
  2. Same functionality as PROJECT #1**, but MOFIDIED AS A BUSINESS CLASS.**
  3. ***Modified as specified in APPENDIX A***

VideoGame Class Explanation:

- The ***VideoGame*** Class requirements are as follows:
  1. ***VideoGame Class*** **`Inherits`** from ***Product*** & represents the **Video Games** rented and sold in the Business
  2. Same functionality as PROJECT #1**, but MOFIDIED AS A BUSINESS CLASS.**
  3. ***Modified as specified in APPENDIX A***

BussinessCollectionBase & Custom Business Collection Classes Details and Summary (CLASS DETAILS IN APPENDIX A):

BusinessCollectionBase Class Explanation:

- The ***BusinessCollectionBase*** Class details are as follows:
  1. **`MustInherit`** class which will serve as the BASE CLASS to CONVERT ALL REGULAR COLLECTION CLASSES TO BUSINESS COLLECTION CLASSES.
  2. ***BusinessCollectionBase Class*** **`Inherits`** from ***DictionaryBase*** to provide ALL COLLECTION CLASSES WITH A DICTIONARY COLLECTION OBJECT.
  3. Implements the DIRTY & DEFERRED DELETED Mechanism for COLLECTION CLASSES
  4. ***Details in APPENDIX A***.

EmployeeList Class Details:

- The ***EmployeeList*** Class details are as follows:
  1. ***EmployeeList Class*** **`Inherits`** from ***BussinessBase***
  2. Same functionality as PROJECT #1**, but MOFIDIED AS A BUSINESS COLLECTION CLASS.**
  3. ***Modified as specified in APPENDIX A***

CustomerList Class Details:

- The ***CustomerList*** Class details are as follows:
  1. *CustomerList* ***Class*** **`Inherits`** from ***BussinessBase***
  2. Same functionality as PROJECT #1**, but MOFIDIED AS A BUSINESS COLLECTION CLASS.**
  3. ***Modified as specified in APPENDIX A***

DVDList Class Details:

- ❑ The ***DVDList*** Class details are as follows:

  1. *DVDList* ***Class*** **`Inherits`** from ***BussinessBase***
  2. Same functionality as PROJECT #1**, but MOFIDIED AS A BUSINESS COLLECTION CLASS.**
  3. ***Modified as specified in APPENDIX A***

VideoGameList Class Details:

- ❑ The ***VideoGameList*** Class details are as follows:

  1. *VideoGameList* ***Class*** **`Inherits`** from ***BussinessBase***
  2. Same functionality as PROJECT #1**, but MOFIDIED AS A BUSINESS COLLECTION CLASS.**
  3. ***Modified as specified in APPENDIX A***

- ❖ **(CLASS DETAILS IN APPENDIX A)**

Presentation/User-Interface Layer

- ❑ This section provides the overall requirements of Project2.
- ❑ **FORM DETAILS IN APPENDIX B**

Requirements #4 – KEEP ALL Requirements from CST3608 Project #2 As far as Form Design & Navigation

- ❑ The User-interface layer stays the same as CST3608 PROJECT 2. Keep all Forms and Form Flow

  a) ***Main Screen***

  b) ***POS Screen***

  c) ***Back-End Management Screen***

  d) ***Employee Management***:

  e) ***Customer Management***

  f) ***DVD Management***

  g) ***VideoGame Management***

Requirement #5 – In the Back-End Management Forms UPDATE THE FUNCTIONALITY of the DELETE/REMOVE BUTTON CLICK EVENT-HANDLER by calling the Collection.DeferredDelete(Key) method just before calling the Collection.Remove(Key) method

- ❑ For Every Management Form, **UPDATE** the **DELETE/REMOVE BUTTON CLICK EVENT HANDLER CODE** TO perform a DEFERRED DELETE by calling **objEmployeeList.DeferredDelete()** Method **IN ADDITION** TO calling **objEmployeeList.Remove()** Method.

  - ▪ UPGRADE this functionality in every MANAGEMENT FORM DELETE BUTTON by making a CALL to the ***DeferredDelete(KEY)*** Method of the LIST CLASSES IN ADDITION TO calling ***Remove(KEY)*** method.

  - ❖ **NOTE –** In next version of project the **Remove()** Method **WILL BE REMOVED. WE TEMPORARILY NEED Remove()** Method NOW to make the project work since we ARE NOT USING ALL THE CAPABILITIES OF BUSINESS OBJECT AT THIS TIME (NO data access code or ADO.NET code in the Business Classes & ALL DATA ACCESS BEING DONE FROM COLLECTION CLASSES) SO THE PROJECT WON'T WORK WITHOUT **Remove()** Method.

- ❑ **FORM DETAILS IN APPENDIX B**

Requirement #6 – MODULE Code Details (User-Interface)

- ❑ SAME AS Previous Version 2, no changes required.

  - ▪ Program Flow the same.
  - ▪ Authentication Requirements the same as Project 1.

APPENDIX A – CLASS DETAILS

Detailed Class Descriptions: (Updated Changes to Classes are highlighted)

Class BusinessBase (Class Provided. No need to Implement. Simply Add it to DLL Project)

- ❑ BASE Class that will contain the MECHANISM to SUPPORT Business Logic, DATA ACCESS & VALIDATION RULES FOR ALL BUSINESS CLASSES.
- ❑ **NOTE: YOU DO NOT HAVE TO IMPLEMENT THIS CLASS. PROFESSOR WILL PROVIDE YOU WITH THE CLASS. SIMPLY ADD TO YOUR PROJECT & MODIFY AS REQUIRED**.
- ❑ key properties/methods are described as follows:

General Information	Description
Imports System.IO Imports System.Data Imports System.Data.OleDb Imports System.Configuration 'Keep commented (for future use) 'System.Runtime.Serialization.Formatters.Binary 'Imports System.Runtime.Remoting 'Imports System.Runtime.Remoting.Channels System.Runtime.Remoting.Channels.Http	▪ Option Explicit and Option Strict should be On ▪ Imported .NET libraries to support: - File/IO - Database support for ADO.NET Data Access Technology ▪ Commented imported .NET libraries for future use to support: - Serialization - Remoting

General Class Information	Description
<Serializable()> _ Public MustInherit Class BusinessBase	▪ **MustInherit** Base Class – Designed for inheritance only. No objects of this class should or can be created. ▪ Provides the business rules, data access support and validation support for ALL BUSINESS CLASSES below (***Person***, ***Employee***, ***Customer***, ***Product***, ***DVD & VideoGame***). ▪ ADD the KEYWORD <Serializable()> _ to enable Serialization for this class.

Private Data	Description
Private m_flgIsDirty As Boolean = True	Purpose: Tracks whether the object has been modified. Data type: **Boolean**
Private m_flgIsNew As Boolean = True	Purpose: Tracks whether the object is a New Object. Data type: **Boolean**
Private m_flgIsDeleted As Boolean = False	Purpose: Implements Deferred Delete process. Tracks whether the object has been MARKED FOR DELETION. Data type: **Boolean**

Public Properties (GET/SET)	Description
Public Overridable ReadOnly Property IsDirty() As Boolean	Read-Only Property – GET: Returns **m_flgIsDirty** private data
Public ReadOnly Property IsNew() As Boolean	Read-Only– GET: Returns **m_flgIsNew** private data
Public Overridable ReadOnly Property IsDeleted() As Boolean	Read-Only– GET: Returns **m_flgIsDeleted** private data

Private & Protected Methods	Parameters	Return Type	Description
Protected Sub MarkDirty()	**None**	**None**	▪ **Implementation** of ***MarkDirty()*** method. ▪ Algorithm: 1. Sets the Dirty Flag **m_flgIsDirty** to **True.**
Private Sub MarkClean()	**None**	**None**	▪ **Implementation** of ***MarkClean()*** method. ▪ Algorithm: 1. Sets the Dirty Flag **m_flgIsDirty** to **False**.
Protected Sub MarkNew()	**None**	**None**	▪ **Implementation** of ***MarkNew()*** method. ▪ **Algorithm:** 1. Sets the New Flag **m_flgIsNew** to **True.** 2. Sets the Deleted Flag m_flgIsDeleted to **False**. 3. Calls ***MarkDirty()*** method.
Protected Sub MarkOld()	**None**	**None**	▪ **Implementation** of ***MarkOld()*** method. ▪ **Algorithm:** 1. Sets the New Flag **m_flgIsNew** to **False.** 2. Calls ***MarkClen()*** method
Protected Sub MarkDeleted()	**None**	**None**	▪ **Implementation** of ***MarkDeleted()*** method. ▪ **Algorithm:** 1. Sets the New Flag m_flgIsDeleted to **True.** 2. Calls ***MarkDirty()*** method

Public Methods	Parameters	Return Type	Description
Public Sub DeferredDelete()	**None**	**None**	▪ **Implementation** of ***DeferredDelete()*** method. ▪ Algorithm: 1. Call **MarkDeleted()** Method.

Class Person

- Represents a Person. Intended to be used as a base class
- The key properties/methods are described as follows:

General Information	Description

`Imports System.IO`
`Imports System.Data`
`Imports System.Data.OleDb`
`Imports System.Configuration`
 	
`'Keep commented (for future use)`
`'System.Runtime.Serialization.Formatters.Binary`
`'Imports System.Runtime.Remoting`
`'Imports System.Runtime.Remoting.Channels`
`System.Runtime.Remoting.Channels.Http`
	▪ Option Explicit and Option Strict should be On
	▪ Imported .NET libraries to support:
	- File/IO
	- Database support for ADO.NET Data Access Technology
	
	▪ Commented imported .NET libraries for future use to support:
	- Serialization
	- Remoting

General Class Information	Description

`<Serializable()> _`
`Public MustInherit Class Person`
`    Inherits BusinessBase`
	▪ **MustInherit** Base Class – Designed for inheritance only. No objects of this class should or can be created.
	▪ Represents the Persons to be either employees or customers to the business.
	▪ ADD the KEYWORD `<Serializable()> _` to enable Serialization for this class.
	▪ Class **Inherits** from ***BusinessBase*** Class.

Private Data	Description
Private m_SSNumber	Purpose: Represents person's social security number. Data type: **String**
Private m_FirstName	Purpose: Represents person's name. Data type: **String**
Private m_LastName	Purpose: Represents person's last name. Data type: **String**
Private m_Address	Purpose: Represents person's address. Data type: **String**
Private m_Phone	Purpose: Represents person's phone number. Data type: **String**
Private m_BirthDate	Purpose: Represents person's birth date. Data type: **Date**
Private m_Age	Purpose: Represents person's age. Data type: **Integer**
Private Shared m_Count = 0	Purpose: **Shared** variable use to keep count of customer objects created. Private data is initialized to 0 on creation. Data type: **Integer**

Public Properties (GET/SET)	Description
Public FirstName	▪ GET/SET **m_FirstName** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL `MyBase.MarkDirty()` method after setting the private data.
Public LastName	▪ GET/SET **m_LastName** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL `MyBase.MarkDirty()` method after setting the private data.
Public SSNumber	▪ GET/SET **m_SSNumber** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL `MyBase.MarkDirty()` method after setting the private data.
Public Address	▪ GET/SET **m_Address** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL `MyBase.MarkDirty()` method after setting the private data.
Public Phone	▪ GET/SET **m_Phone** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL `MyBase.MarkDirty()` method after setting the private data.
Public Overridable BirthDate	▪ Declared **Overridable** to allow derived classes to override ▪ SETS & GETS the **m_BirthDate** private data ▪ In SET portion also Calculates the Age based on Birthdate value and today's date as follows: o Calculates the age by subtracting Today's date from the Birthdate Value being SET. o Assigns the calculated age to the ***m_Age*** private data or property. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL `MyBase.MarkDirty()` method after setting both the Birthdate and Age private data. This should be the last statement in the SET property.
Public ReadOnly Age	▪ GET/SET **m_Age** private data.
Public Shared Count	▪ GET/SET Shared **m_Count** private data.

Constructors	Parameters	Return Type	Description
Public New()	None	N/A	▪ **Default Constructor**. Should initialize the PRIVATE DATA members with appropriate default values ▪ Sets to appropriate default values: - m_SSNumber, m_FirstName, m_LastName =”” - m_Birthdate = #1/1/1900# - m_Address , m_PhoneNumber =”” - m_Age = 0
Public New(x, y, z etc..)	▪ A parameter to Set each of the following **Properties** only (AGE & COUNT not included): - SSNumber - FirstName - LastName - Birthdate - Address - PhoneNumber ▪ Should have a total of 6 parameters: - **par1 to Par6** ▪ Name the parameters as you see fit. ▪ All parameters are Pass-by-Value	N/A	▪ Parameterized Constructor 1) Sets parameter list to all data members via **PUBLIC PROPERTIES** EXCEPT the Username, Password, Age & COUNT. 2) Match parameter list appropriately: - SSNumber = **par1** - FirstName = **par2** - LastName = **par3** - Birthdate = **par4** - Address = **par5** - PhoneNumber = **par6** 3) Age is handled by Birthdate Property, no need to address in the constructor

Public MustOverride Methods	Parameters	Return Type	Description
Public MustOverride Sub Print()	None	None	▪ MustOverride **Print()** method. ▪ Intended for derived classes to print their data ▪ **Declaration only**. Must be implemented in derived classes.
Public MustOverride Function Authenticate(user ,pass)	String user, String pass	True False	▪ MustOverride **Authenticate(u,p)** method. ▪ Intended for derived classes to authenticate themselves ▪ **Declaration only**. Must be implemented in derived classes.

Class Employee

- Represents the employees of the company. The key properties/methods are described as follows:

General Information	Description
Imports System.IO Imports System.Data Imports System.Data.OleDb Imports System.Configuration 'Keep commented (for future use) 'System.Runtime.Serialization.Formatters.Binary 'Imports System.Runtime.Remoting 'Imports System.Runtime.Remoting.Channels System.Runtime.Remoting.Channels.Http	▪ Option Explicit and Option Strict should be On ▪ Imported .NET libraries to support: - File/IO - Database support for ADO.NET Data Access Technology ▪ Commented imported .NET libraries for future use to support: - Serialization - Remoting

General Class Information	Description
<Serializable()> _ **Public Class Employee** **Inherits Person**	▪ Represents the **Employee** of the business. ▪ ADD the KEYWORD <Serializable()> _ to enable Serialization for this class. ▪ Class **Inherits** from ***Person*** Class.

Private Data	Description
Private m_JobTitle	Purpose: stores employee’s job title. Data type: **String**
Private m_UserName	Purpose: stores employee’s username. Data type: **String**
Private m_Password	Purpose: stores employee’s password. Data type: **String**

Public EVENT	Description
Event SecurityAlert(ByVal username, ByVal password)	▪ Event to handle any SECURITY ALERTS related with an Employee. ▪ Trigger or raise the event in the following method: - *Trigger* or ***raise*** this event ONLY inside the *Authenticate()* method **prior** to the verification of the username & password. - Pass into the RAISEEVENT call the parameters **username** & **password** passed into the method.

Public Properties (GET/SET)	Description
Public JobTitle	▪ GET/SET **m_JobTitle** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public UserName	▪ GET/SET **m_UserName** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public Password	▪ GET/SET **m_Password** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public Overrides Birthdate	Overrides the Birthdate property in order to SET/GET the BASE CLASS Birthdate PROPERTY and implement the following company policy: ▪ SETS & GETS **the m_ Birthdate** private data from Base class ▪ Implement the following company policy: ○ Under aged employees cannot work for this company. Employees under 16 years of age cannot work in this business. ○ **Validate that employee MUST BE 16 YEARS OF AGE or older based on Birthdate value and today's date otherwise THROW AND EXCEPTION**. ○ **Leverage the Person.Birthdate property in order to store the date & set the age**.

Constructors	Parameters	Return Type	Description
Public New()	None	N/A	▪ **Default Constructor**. ▪ Should initialize the private data members with appropriate default values for this Employee Class and the Base Class. **1.** Calls **Base Class Default Constructor** **2.** The data ***m_Count*** is incremented only **3.** Sets to appropriate default values: - m_JobTitle =”” - m_Username =”” - m_Password =”” **4.** **INCREMENTS the** Shared m_Count **data.**
Public New(**x, y, z etc..)**	▪ A parameter to Set each of the following Employee Class & Person Base Class **Properties**: - SSNumber - FirstName - LastName - Birthdate - Address - PhoneNumber - JobTitle ▪ Should have a total of **7** parameters: - **par1 to Par7** ▪ Name the parameters as you see fit. ▪ All parameters are Pass-by-Value	N/A	**1.** **Parameterized Constructor** **2.** Sets the **PROPERTIES** of the Person Base Class and this Employee Class matching parameter list. **3.** The data ***m_Count*** is incremented only 1. Calls **Base Class Parameterized Constructor** to handle the following parameters: - SSNumber = **par1** - FirstName = **par2** - LastName = **par3** - Birthdate = **par4** - Address = **par5** - PhoneNumber = **par6** 2. Sets the **PROPERTIES** of this class with the following parameters: - JobTitle = **par7** 3. The **Username** & **Password** are NOT part of the parameters and should be defaulted as follows: - Username = **“”** - Password = **“”** **4.** **INCREMENTS the Person Class Shared *Count.***

Public Overrides Methods	Parameters	Return Type	Description
Public Overrides Sub Print()	None	None	▪ **Implementation of Print()** method. ▪ The *Print()* Method WRITES ALL OBJECT'S DATA TO THE PRINTER FILE as follows: **1.** Opens ***Network_Printer.txt*** file for APPENDING. **2.** Write each object's property/data in the following FORMAT: Printing ***Employee*** .... ID Number = ***value*** Name = ***value*** Social Security = ***value*** Birthday = ***value*** Address = ***value*** Age = ***value*** Etc…. **3.** Close the file ▪ Add Error-Handling code using `try-catch-finally` to handle all required exceptions for any file access, array or general exceptions ▪ Follow best practice of trapping for unexpected general errors in addition to specific errors ▪ Use `throw` statement to re-throw all exceptions
Public Overrides Function Authenticate(user, pass)	String user, String user	True False	▪ Method to Authenticate Employee Username & Password as follows: 1) *Trigger* or ***raise*** this **SecurityAlert(U,P)** Event **prior** to the verification of the username & password 2) *Process:* Compares each of the TWO parameters (U, P) values to the private ***m_username*** & ***m_password*** variables 3) **Returns** a **TRUE** if both of these values match, otherwise it returns a **FALSE**. ▪ **The objective of this function is to AUTHENTICATE THE EMPLOYEE OBJECT ITSELF**.

Public Overridable Methods	Parameters	Return Type	Description
`Public Overridable Sub Product_Rental()`	None	`None`	- **Implementation of Product_Rental()** method. - `Overridable` so any future derived classes can override. - Handles future product rental transactions for an employee who wishes to rent. - NO IMPLEMENTATION, STUB METHOD FOR FUTURE UPGRADE. - CREATE THE HEADER WITH AN EMPTY BODY. - **This is a VIOLATION OF CURRENT COMPANY POLICY.** - **MUST DISABLED WITH Use** `Throw` **NotSupported Exception statement**
`Public Overridable Sub Product_Return()`	None	`None`	- **Implementation of Product_Return()** method. - `Overridable` so any future derived classes can override. - Handles future product returns transactions for an employee who wishes to rent. - NO IMPLEMENTATION, STUB METHOD FOR FUTURE UPGRADE. - CREATE THE HEADER WITH AN EMPTY BODY. - **This is a VIOLATION OF CURRENT COMPANY POLICY.** - **MUST DISABLED WITH Use** `Throw` **NotSupported Exception statement**

Public Data Access Methods	Parameters	Return Type	Description
`Public Shared Function Create()As Employee`	`None`	`Employee Reference or Pointer`	▪ **Implementation of `Create()`** method. ▪ Public interface to the data access method that CREATES A NEW OBJECT USING FACTORY METHOD. ▪ The `Create()` method does not perform the data access but calls upon the `DataPortal_Create()` method to do the work. ▪ Algorithm: **1.** CALL & `Return` the `DataPortal_Create()` method.
`Public Sub Load(key)`	`String key`	`None`	▪ **Implementation of `Load(key)`** method. ▪ Public interface to the data access method that retrieves the RECORD of the PRIMARY KEY ***KEY/SSNunber*** passed as a parameter. ▪ The `Load(key)` method does not perform the data access but calls upon the `DataPortal_Fetch(Key)` method to do the work. ▪ Algorithm: **1.** CALL the `DataPortal_Fetch(Key)` method.
`Public Sub Save()`	None	`None`	▪ **Implementation of Save()** method. ▪ The ***Save()*** Method is a very important method. ▪ **This Method uses the BusinessBase Class IsDirty, IsNew & IsDeleted TO DETERMINE WHICH DATA BASE OPERATION TO PERFORM ON THE OBJECT, EITHER INSERT, UPDATE OR DELETE**. ▪ The code for this method is provided in your BUSINESS CLASS TEMPLATE. Nevertheless, I will include the algorithm here. ▪ Algorithm: **1.** IF OBJECT is marked for deletion via its `Me.IsDeleted FLAG` or OBJECT is `Not NEW via its New FLAG Me.IsNew` then CALL `DataPortal_Delete(Me.`**`SSNumber`**`)` to DELETE RECORD FROM DATABASE. **2.** ELSE if OBJECT IS DIRTY via `Me.IsDirty` and IF OBJECT is NEW via `Me.IsNew` then CALL `DataPortal_Insert()` TO ADD RECORD TO DATABASE **3.** ELSE CALL `DataPortal_Update()` TO UPDATE THE RECORD IN THE DATABASE.:
`Public Sub ImmediateDelete(ByVal Key As String)`	`String key`	`None`	▪ **Implementation of `Delete(key)`** method. ▪ Public interface to the data access method that IMMEDIATELY DELETES the RECORD of the PRIMARY KEY ***KEY/SSNunber*** passed as a parameter FROM DATABASE. ▪ Algorithm: **1.** CALL the `DataPortal_ Delete(Key)` method.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Shared Function DataPortal_Create() As Employee	None	Employee Reference or Pointer	▪ **Implementation of Shared Function DataPortal_Create() method.** ▪ OBJECT FACTORY METHOD. ▪ Creates & RETURNS FULLY READY AND INITIALIZE EMPLOYEE OBJECTS. ▪ STUB FUNCTION METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY & RETURN KEYWORD.
Protected Sub DataPortal_Fetch(ByVal Key As String)	String key	None	▪ **Implementation of DataPortal_Fetch(Key)** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method `MyBase.MarkOld()`
Protected Sub DataPortal_Update()	None	None	▪ **Implementation of DataPortal_Update()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method `MyBase.MarkOld()`

Protected Data Access Methods	Parameters	Return Type	Description
Protected Sub DataPortal_Insert()	None	None	▪ **Implementation of DataPortal_Insert()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method `MyBase.MarkOld()`
Protected Sub DataPortal_Delete(ByVal Key As String)	None	None	▪ **Implementation of DataPortal_Delete()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as NEW by calling method `MyBase.New() since recod is no longer in the database and has been deleted.  At this point, the object is a new object.`

Class Customer

- ❑ Represents the customer.
- ❑ The Customer class has the following key properties/methods:

General Information	Description

`Imports System.IO`
`Imports System.Data`
`Imports System.Data.OleDb`
`Imports System.Configuration`
 	
`'Keep commented (for future use)`
`'System.Runtime.Serialization.Formatters.Binary`
`'Imports System.Runtime.Remoting`
`'Imports System.Runtime.Remoting.Channels`
`System.Runtime.Remoting.Channels.Http`
	▪ Option Explicit and Option Strict should be On
	▪ Imported .NET libraries to support:
	- File/IO
	- Database support for ADO.NET Data Access Technology
	
	▪ Commented imported .NET libraries for future use to support:
	- Serialization
	- Remoting

General Class Information	Description

`<Serializable()> _`
`Public Class Customer`
`    Inherits Person`
	▪ Represents the **Customer** of the business.
	▪ ADD the KEYWORD `<Serializable()> _` to enable Serialization for this class.
	▪ Class **`Inherits`** from ***Person*** Class.

Private Data	Description
`Private` m_IDNumber	Purpose: stores customer's ID number. Data type: **`String`**

Public Properties (GET/SET)	Description
`Public` IDNumber	▪ GET/SET **m_IDNumber** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **`MyBase.MarkDirty()`** method after setting the private data.

Constructors	Parameters	Return Type	Description
Public New()	None	**N/A**	▪ **Default Constructor**. ▪ Should initialize the private data members with appropriate default values for this ***Customer*** Class and the ***Person*** Base Class. **1.** Calls **Base Class Default Constructor** **2.** The data ***m_Count*** is incremented only **3.** Sets to appropriate default values: - m_IDNumber =”” **4.** **INCREMENTS the Person Class Shared *Count*.**
Public New(x, y, z etc..)	▪ A parameter to Set each of the following Employee Class & Person Base Class **Properties**: - IDNumber - FirstName - LastName - SSNumber - Birthdate - Address - PhoneNumber ▪ Should have a total of 7 parameters: - **par1 to Par7** ▪ Name the parameters as you see fit. ▪ All parameters are Pass-by-Value	**N/A**	• **Parameterized Constructor** • Sets the **PROPERTIES** of the ***Person*** Base Class and this ***Customer*** Class matching parameter list. • The data ***m_Count*** is incremented **1.** Calls **Base Class Parameterized Constructor** to handle the following parameters: - FirstName = **par2** - LastName = **par3** - SSNumber = **par4** - Birthdate = **par5** - Address = **par6** - PhoneNumber = **par7** **2.** Sets the **IDNumber PROPERTY** of this class with the following parameters : - IDNumber = **par1** **3.** **INCREMENTS the Person Class Shared *Count*.**

Public Overrides Methods	Parameters	Return Type	Description
`Public Overrides Sub Print()`	None	`None`	▪ **Implementation of Print()** method. ▪ The *Print()* Method WRITES ALL OBJECT'S DATA TO THE PRINTER FILE as follows: **1.** Opens ***Network_Printer.txt*** file for APPENDING. **2.** Write each object's property/data in the following FORMAT: Printing ***Customer*** .... ID Number = ***value*** Name = ***value*** Social Security = ***value*** Birthday = ***value*** Address = ***value*** Age = ***value*** Etc…. **3.** Close the file ▪ Add Error-Handling code using `try-catch-finally` to handle all required exceptions for any file access, array or general exceptions ▪ Follow best practice of trapping for unexpected general errors in addition to specific errors ▪ Use `throw` statement to re-throw all exceptions

`Public Overrides Function Authenticate(user, pass)`	`String user,`
	`String user`
		`True`
		`False`
			▪ **Implementation of Authenticate(u,p)** method.
			▪ **Company Policy does not dictate Customers need to authenticate. Create the method to satisfy the COMPILER, but do not implement this method.**
			▪ STUB METHOD FOR FUTURE UPGRADE.
			▪ CREATE THE HEADER WITH AN EMPTY BODY & RETURN KEYWORD.

Public Overridable Methods	Parameters	Return Type	Description
Public Overridable Sub Product_Rental()	None	None	▪ **Implementation of Product_Rental()** method. ▪ Overridable so any future derived classes can override. ▪ Handles future product rental transactions for an employee who wishes to rent. ▪ NO IMPLEMENTATION, STUB METHOD FOR FUTURE UPGRADE. ▪ **Create the method to satisfy the COMPILER, but do not implement this method** ▪ CREATE THE HEADER WITH AN EMPTY BODY.
Public Overridable Sub Product_Return()	None	None	▪ **Implementation of Product_Return()** method. ▪ Overridable so any future derived classes can override. ▪ Handles future product returns transactions for an employee who wishes to rent. ▪ NO IMPLEMETATION, STUB METHOD FOR FUTURE UPGRADE ▪ **Create the method to satisfy the COMPILER, but do not implement this method**. ▪ CREATE THE HEADER WITH AN EMPTY BODY.

Public Data Access Methods	Parameters	Return Type	Description
Public Shared Function Create()As Employee	None	Employee Reference or Pointer	▪ **Implementation of Create()** method. ▪ Public interface to the data access method that CREATES A NEW OBJECT USING FACTORY METHOD. ▪ The Create() method does not perform the data access but calls upon the DataPortal_Create() method to do the work. ▪ Algorithm: **1.** CALL & Return the DataPortal_Create() method.
Public Sub Load(key)	String key	None	▪ **Implementation of Load(key)** method. ▪ Public interface to the data access method that retrieves the RECORD of the PRIMARY KEY ***KEY/IDNunber*** passed as a parameter. ▪ The Load(key) method does not perform the data access but calls upon the DataPortal_Fetch(Key) method to do the work. ▪ Algorithm: **1.** CALL the DataPortal_Fetch(Key) method.
Public Sub Save()	None	None	▪ **Implementation of Save()** method. ▪ The ***Save()*** Method is a very important method. ▪ **This Method uses the BusinessBase Class IsDirty, IsNew & IsDeleted TO DETERMINE WHICH DATA BASE OPERATION TO PERFORM ON THE OBJECT, EITHER INSERT, UPDATE OR DELETE**. ▪ The code for this method is provided in your BUSINESS CLASS TEMPLATE. Nevertheless, I will include the algorithm here. ▪ Algorithm: **1.** IF OBJECT is marked for deletion via its Me.IsDeleted FLAG or OBJECT is Not NEW via its New FLAG Me.IsNew then CALL DataPortal_Delete(Me.**IDNumber**) to DELETE RECORD FROM DATABASE. **2.** ELSE if OBJECT IS DIRTY via Me.IsDirty and IF OBJECT is NEW via Me.IsNew then CALL **DataPortal_Insert()** TO ADD RECORD TO DATABASE **3.** ELSE CALL DataPortal_Update() TO UPDATE THE RECORD IN THE DATABASE.:
Public Sub ImmediateDelete(ByVal Key As String)	String key	None	▪ **Implementation of Delete(key)** method. ▪ Public interface to the data access method that IMMEDIATELY DELETES the RECORD of the PRIMARY KEY ***KEY/SSNunber*** passed as a parameter FROM DATABASE. ▪ Algorithm: **1.** CALL the DataPortal_ Delete(Key) method.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Shared Function DataPortal_Create() As Employee	**None**	**Employee Reference or Pointer**	▪ **Implementation of Shared Function DataPortal_Create() method.** ▪ OBJECT FACTORY METHOD. ▪ Creates & RETURNS FULLY READY AND INITIALIZE EMPLOYEE OBJECTS. ▪ STUB FUNCTION METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY & RETURN KEYWORD.
Protected Sub DataPortal_Fetch(ByVal Key As String)	**String key**	**None**	▪ **Implementation of DataPortal_Fetch(Key)** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method MyBase.MarkOld()
Protected Sub DataPortal_Update()	None	**None**	▪ **Implementation of DataPortal_Update()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method MyBase.MarkOld()
Protected Sub DataPortal_Insert()	None	**None**	▪ **Implementation of DataPortal_Insert()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method MyBase.MarkOld()
Protected Sub DataPortal_Delete(ByVal Key As String)	None	**None**	▪ **Implementation of DataPortal_Delete()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as NEW by calling method MyBase.New() since recod is no longer in the database and has been deleted. At this point, the object is a new object.

Class BusinessCollectionBase (Class Provided. No need to Implement)

- ❑ BASE Class that will contain the MECHANISM to SUPPORT Business Logic, DATA ACCESS & VALIDATION RULES FOR ALL **BUSINESS COLLECTION** CLASSES.
- ❑ **NOTE: YOU DO NOT HAVE TO IMPLEMENT THIS CLASS. PROFESSOR WILL PROVIDE YOU WITH THE CLASS. SIMPLY ADD TO YOUR PROJECT & MODIFY AS NECESSARY**.
- ❑ key properties/methods are described as follows:

General Information	Description

`Imports System.IO`
`Imports System.Data`
`Imports System.Data.OleDb`
`Imports System.Configuration`
 	
`'Keep commented (for future use)`
`'System.Runtime.Serialization.Formatters.Binary`
`'Imports System.Runtime.Remoting`
`'Imports System.Runtime.Remoting.Channels`
`System.Runtime.Remoting.Channels.Http`
	▪ Option Explicit and Option Strict should be On
	▪ Imported .NET libraries to support:
	- File/IO
	- Database support for ADO.NET Data Access Technology
	
	▪ Commented imported .NET libraries for future use to support:
	- Serialization
	- Remoting

General Class Information	Description

`<Serializable()> _`
`Public MustInherit Class BusinessCollectionBase`
`Inherits DictionaryBase`
	▪ `MustInherit` Base Class – Designed for inheritance only. No objects of this class should or can be created.
	▪ Provides the business rules, data access support and validation support for ALL BUSINESS COLLECTION CLASSES below (***EmployeeList***, ***CustomerList***, ***DVDList***, & ***VideoGameList***).
	▪ Class `Inherits` from ***DictionaryBase*** Class to provide the derived classes with a DICTIONARY COLLECTION OBJECT.
	▪ ADD the KEYWORD `<Serializable()> _` to enable Serialization for this class.

Private Data	Description
None	Not needed. For future upgrade if required

Public Properties (GET/SET)	Description
`Public ReadOnly Property IsDirty() As Boolean`	READ-ONLY GET: Searches the DICTIONARY COLLECTION for a DIRTY OBJECT. Returns `True` if DIRTY OBJECT FOUND, `True` otherwise. . ▪ GET Algorithm: **1.** LINEAR SEARCH of `MyBase.Dictionary` Collection **2.** Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection **3.** Use `CType()` Function to convert to native data type of `DictionaryEntry` POINTER to ***BusinessBase*** Class POINTER. **4.** Interrogates each object by testing its `m_flgIsDirty` flag via the `IsDirty()` property. **5.** Returns a `True` if DIRTY OBJECT FOUND, else, returns a `False` if no OBJECT IS DIRTY. **6.** Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public MustOverride Methods	Parameters	Return Type	Description
`Public Overrides Function DeferredDelete(ByVal strKey As String) As Boolean`	`String`	`None`	▪ `MustOverride DeferredDelete()` method. ▪ Intended for derived classes to IMPLEMENT DEFERRED DELETE ▪ **Declaration only**. Must be implemented in derived classes.

Class EmployeeList

- ❑ ***EmployeeList*** COLLECTION Class. Object of this class will store and manage **Employee Objects** in memory and load/save them from ***TEXT FILE***.
- ❑ This class encapsulates a ***DICTIONARY COLLECTION*** OBJECT provided via ***<u>INHERITANCE</u>*** from the **BUSINESSCOLLECTIONBASE** Class which ***<u>INHERITS</u>*** from ***DICTIONARYBASE CLASS***.
- ❑ The key properties/methods are described as follows:

General Information	Description
Option Explicit On Option Strict On 'Impoted Libraries Imports System.IO Imports System.Data Imports System.Data.OleDb Imports System.Configuration	▪ Option Explicit and Option Strict should be On

General Class Information	Description
<Serializable()> _ **Public Class clsEmployeeList** **Inherits** **BusinessCollectionBase**	▪ CUSTOM COLLECTION CLASS that encapsulates a DICTIONARY COLLECTION OBJECT that stores **Employee** objects. ▪ ADD the KEYWORD **<Serializable()> _** to enable Serialization for this class. ▪ Class **Inherits** from ***BusinessCollectionBase*** Class.

Public Properties (GET/SET)	Description
Public Shadows ReadOnly Property Count() As Integer	READ-ONLY GET: returns NUMBER OF ELEMENTS OR OBJECT IN THE COLLECTION DICTIONARY. Property should shadow the base class equivalent. ▪ GET Algorithm: **1.** Calls BASE CLASS **MyBase.Dictionary.Count** Property.
Public Property Item(ByVal key As String) As Employee	▪ **Wrapper PROPERTY** that **GET & SET** *Employee* **OBJECTS in the DICTIONARY COLLECTION** ▪ This Property GETS the POINTER to the OBJECT in the COLLECTION based on its KEY. ▪ This Property SETS the OBJECT WHO'S KEY is passed as parameter with the OBJECT being assigned. ▪ GET Algorithm: **1.** CALLS the BASE CLASS **MyBase.Dictionary.Item(key)** Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. **2.** Use **CType()** Function to convert to native data type of DICTIONARY collection to ***Employee*** Class type. ▪ SET Algorithm: **1.** CALLS the BASE CLASS **MyBase.Dictionary.Contains**(key) Method determine if KEY exists. **2.** If exists it calls **MyBase.Dictionary.Item(key)** to do the work of SETTING the VALUE. **3.** Else, THROWS Throw New System.ArgumentException("ID Not found") EXCEPTION indicating KEY WAS NOT FOUND.

Public Methods	Parameters	Return Type	Description
`Public Sub Add(ByVal key As String, ByVal objEmployee As Employee)`	`Employee objEmployee`	`None`	▪ **Implementation of Add(object)** method. ▪ ***Wrapper Method*** that ADDS the object & its associated KEY passed as argument into the collection. ▪ In this case, the KEY is the ***SSNumber*** PROPERTY of the Object. ▪ It is assumed the Object if CREATED & POPULATED in the User-Interface or calling program when passed as argument to the method call. Method simply adds the object to the COLLECTION. ▪ Algorithm: 1. Calls `MyBase.Dictionary.Add(key, objEmployee)` Method to add the object to Collection. 2. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Sub Add(ByVal x, ByVal y,ByVal z, etc.)`	Variable for each required Parameter to SET PROPERTY of the ***Employee*** Class Object. Normally the paramters of the Parameterized Constructor.	`None`	▪ **Implementation of Add(x,y,z etc.)** method. ▪ ***Wrapper Method*** that ADDS object into the collection. ▪ Same functionality as previous ADD, except NO populated OBJECT is passed as parameter, but the individual values that make up the OBJECT. ▪ The method itself creates the Object, populates it with the values from parameters and then ADDS the object to the COLLECTION with the KEY. ▪ In this case, the KEY is the ***SSNumber*** PARAMETER value of the method's parameter list. ▪ This version of ADD, requires less programming in the User-Interface. ▪ Algorithm: 1. Creates either DEFAULT Temporary ***Employee*** OBJECT & SETS the appropriate properties based on parameters VALUES passed to method. Or CREATES PARAMETERIZED Constructor OBJECT, passing to the OBJECT the VALUES of the parameters passed to method. 2. Calls `MyBase.Dictionary.Add(key, objEmployee)` Method to add the object to Collection. The KEY being the ***SSNumber*** PROPERTY of the object created. 3. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Function Edit(ByVal key As String, ByVal objEmployee As Employee) As Boolean`	`Employee objEmployee`	`True` `False`	▪ **Implementation of Edit(object)** method. ▪ ***Wrapper Method*** that EDITS the OBJECT in the COLLECTION whose KEY is passed as parameter to method. ▪ In this case, the KEY is the ***SSNumber*** PROPERTY of the Object ▪ The OBJECT in COLLECTION is edited by SETTING its PROPERTIES, with the values or the PROPERTIES of the OBJECT passed as parameter. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Item(key)` Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use `CType()` Function to convert to native data type of DICTIONARY collection to ***Employee*** Class type. 3. Returns a FALSE if POINTER returned by `MyBase.Dictionary.Item(key)` Property is a ***Nothing***. 4. Else, **GETS PROPERTIES** of Object passed as parameter & **SETS PROPERTIES** of OBJECT in the COLLECTION whose POINTER was returned by `Dictionary.Item(key)` Property & **Returns** a **True**. 5. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Function Edit(ByVal x, ByVal y,ByVal z, etc.) As Boolean`	Variable for each required Parameter to SET PROPERTY of the ***Employee*** Class Object. To be EDITED.	`True` `False`	▪ **Implementation of Edit(x,y,z)** method. ▪ ***Wrapper Method*** that EDITS the OBJECT in the COLLECTION who's associated KEY is passed as ONE of the parameter variables. The OBJECT in COLLECTION is edited by SETTING its PROPERTIES with the values passed as parameters to method, except the KEY parameter. ▪ In this case, the KEY is the ***SSNumber*** PARAMETER value of the method's parameter list. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Item(key)` Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use `CType()` Function to convert to native data type of DICTIONARY collection to ***Employee*** Class type. 3. Returns a FALSE if POINTER returned by `MyBase.Dictionary.Item(key)` Property is a ***Nothing***. 4. Else **SETS PROPERTIES of OBJECT** in the COLLECTION whose POINTER was returned by `MyBase.Dictionary.Item(key)` with values passed as parameters, except the value that represents the KEY & **Returns** a **True** 5. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Function Remove(ByVal key As String) As Boolean`	`String Key`	`True` `False`	▪ **Implementation of Remove(Key)** method. ▪ ***Wrapper Method*** that REMOVES the object from COLLECTION who's associated KEY is passed as parameter to method. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Contains(key)` Method determine if KEY exists. 2. If exists it calls `MyBase.Dictionary.Remove(key)` to do the work of REMOVING OBJECT from COLLECTION and returns a **TRUE**. 3. Else, if not exists, then it returns a **FALSE**. 4. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Function Print(ByVal key As String) As Boolean`	`String Key`	`True` `False`	▪ **Implementation of Print(Key)** method. ▪ ***Method*** that PRINTS the content of the OBJECT in the COLLECTION who's associated KEY is passed as parameter. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Item(key)` Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use `CType()` Function to convert to native data type of DICTIONARY collection to ***Employee*** Class type. 3. Returns a `True` if OBJECT FOUND AND PRINTED. 4. Returns a `False` if POINTER returned by `Dictionary.Item(key)` Property is a ***Nothing***. 5. Else CALLS the `object.Print()` Method of the OBJECT in the COLLECTION whose POINTER was returned by `MyBase.Dictionary.Item(key)`. 6. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Sub PrintAll()`	None	`None`	▪ **Implementation of PrintAll()** method. ▪ ***Method*** that PRINTS the content of ALL THE OBJECT in the COLLECTION. ▪ Algorithm: 1. Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection. 2. Use `CType()` Function to convert to native data type of `DictionaryEntry` object to ***Employee*** Class type. 3. CALLS the `object.Print()` Method of EACH OF THE OBJECT in the COLLECTION. 4. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Shadows Sub Clear()`	None	`None`	▪ **Implementation of Clear()** method. ▪ **Method** uses Shad keyword to Shadow the Base Class Method which already implements this functionality. We are simply repeating the process here. ▪ ***Method*** that CLEARS or DELETES ALL OBJECTS in the COLLECTION. ▪ Algorithm: 1. CALLS `MyBase.Dictionary.Clear()` to do the work. 2. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Overrides Function DeferredDelete(ByVal strKey As String) As Boolean`	`String key`	`Boolean`	▪ **Implementation of DeferredDelete(KEY)** method. ▪ ***Method*** that MARKS AND OBJECT FOR DELETION inside the COLLECTION. ▪ Algorithm: 1. Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection. 2. Use `CType()` Function to convert to native data type of `DictionaryEntry` POINTER to ***Employee*** Class POINTER. 3. Interrogates EACH OBJECT BY COMPARING SEARCH KEY with ID of OBJECT in COLLECTION. 4. If KEY is FOUND, VERIFIES OBJECT IS NOT NEW by getting its `IsNew` Property. 5. If OBJECT IS NOT NEW or OLD, then CALLS the `DeferredDelete()` method of the OBJECT IN COLLECTION to MARK IT FOR DEFERRED DELETION. 6. EXITS the LOOP using `Return True` since OBJECT FOUND & MARKED FOR DELETION & SEARCH IS OVER. 7. Else if Object is NEW then `Return False` since NEW object should NOT BE DELETED but INSERTED 8. If SEARCH AFTER END OF `For Each` is over and KEY WAS NOT FOUND `Return False`. 9. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Data Access Methods	Parameters	Return Type	Description
`Public Shared Function Create()As EmployeeList`	`None`	`EmployeeList Reference or Pointer`	▪ **Implementation of `Create()`** method. ▪ Public interface to the data access method that CREATES A NEW OBJECT USING FACTORY METHOD. ▪ The `Create()` method does not perform the data access but calls upon the `DataPortal_Create()` method to do the work. ▪ Algorithm: **1.** CALL & `Return` the `DataPortal_Create()` method.
`Public Sub Load()`	`String key`	`None`	▪ **Implementation of `Load(key)`** method. ▪ Public interface to the data access method that retrieves ALL THE ***Employee*** RECORDS FROM the Employee Table and ADDS them to the **DICTIONARY COLLECTION OBJECT**. ▪ The `Load()` method does not perform the data access but calls upon the `DataPortal_Fetch()` method to do the work. ▪ Algorithm: **1.** CALL the `DataPortal_Fetch()` method.
`Public Sub Save()`	None	`None`	▪ **Implementation of Save()** method. ▪ The ***Save()*** Method is a very important method. ▪ The `Save()` method does not perform the work but calls upon the `DataPortal_Save()` method to do the work. ▪ **IT HAS THE INTELLIGENCE TO DETERMINE IF THE COLLECTION HAS BEEN CHANGED OR NOT BY TESTING THE IsDirty PROPERTY.** ▪ **If OBJECT IS DIRTY IT CALLS THE FOLLOWING METHOD:** ▪ Algorithm: **1.** If Collection **IsDirty** CALL the `DataPortal_Save()` method. **2.** Otherwise nothing is done.
`Public Sub ImmediateDelete(ByVal Key As String)`	`String key`	`None`	▪ **Implementation of `DeleteObject(key)`** method. ▪ Public interface to the data access method that SEARCHES AN OBJECT IN THE **DICTIONARY COLLECTION OBJECT** & IMMEDIATELY DELETES the RECORD of the PRIMARY KEY ***KEY/SSNumber*** passed as a parameter FROM DATABASE. ▪ The `DeleteObject()` method does not perform the work but calls upon the `DataPortal_DeleteObject()` method to do the work. ▪ Algorithm: **1.** CALL the `DataPortal_DeleteObject(Key)` method.

Protected Data Access Methods	Parameters	Return Type	Description
`Protected Shared Function DataPortal_Create() As EmployeeList`	`None`	`EmployeeList Reference or Pointer`	▪ **Implementation of `Shared Function DataPortal_Create()` method**. ▪ OBJECT FACTORY METHOD. ▪ Creates & RETURNS FULLY READY AND INIALIZED **EmployeeList** OBJECTS. ▪ STUB FUNCTION METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY & RETURN KEYWORD.
`Protected Sub DataPortal_Fetch(ByVal Key As String)`	`String key`	`None`	▪ **Implementation of `DataPortal_Fetch(Key)`** method. ▪ **THE FOLLOWING CODE IS HERE TEMPORARILY TO SUPPORT LOADING FROM A FILE.** ▪ **NEXT PROJECT WE WILL LOAD FROM A REAL DATABASE AND THIS CODE WILL BE REMOVED.** ▪ LOAD the OBJECTS from the `CustomerData.txt` file and ADDS them to the COLLECTION. ▪ Algorithm: 1. USE `File.Exists("CustomerData.txt")` to verify if FILE EXISTS. 2. If FILE DOES NOT EXISTS IT CREATES IT using `File.Create("CustomerData.txt")` Method. 3. Open File for READING. 4. Read a LINE from file & PARSE each comma-delimited line. 5. Create new temporary OBJECT if the Customer Class. 6. SET OBJECT WITH values from LINE READ FROM FILE. 7. ADD OBJECT TO COLLECTION. 8. Repeat this process until EOF. 9. Add Error-Handling code using `Try-Catch-Finally` to handle a ***GENERAL*** Exception.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Sub **DataPortal_Save()**	None	**None**	▪ **Implementation of DataPortal_Save()** method. ▪ **YOU WILL IMPLEMENT THIS METHOD AS DICTATED BY THE BUSINESS COLLECTION CLASS TEMPLATE.** ▪ **NEVERTHELESS, the following CODE WILL NOT HELP US IN THIS PROJECT SINCE THE CURRENT BUSINESS CLASSES DON'T PERFORM REAL DATA ACCESS.** ▪ **THEREFORE THE FOLLOWING CODE IS FOR FUTURE USE, BUT IT MUST BE IMPLEMENTED AS SHOWN IN THE BUSINESS COLLECTION CLASS TEMPLATE:** ▪ WHAT ID DOES IS AS FOLLOWS: o LOOPS through the **DICTIONARY COLLECTION OBJECT** & CALLS EACH OBJECTS ***Save()*** to SAVE THE OBJECT. o The ides is the **EACH OBJECT IN COLLECTION SAVES ITSELF AND DECIDES WHETHER TO DO AN UPDATE OR INSERT BASED ON THE STATUS OF ITS ISDIRTY**, **ISNEW** AND **ISDELETED** FLAGS. ▪ **IMPORTANT! THIS CODE IS PROVIDED TO YOU IN THE *Business Collection Class* TEMPLATE. Simply COPY/PASTE modify for this EMPLOYEELIST CLASS.** ▪ **KEEP IN MIND THIS CODE HAS NO IMPACT TO THE PROGRAM, IT WILL EXECUTE AND CALL THE SAVE METHOD OF EACH OBJECT BUT THE DATA ACCESS METHODS HAVE NOT BEEN IMPLEMENTED YET. BUT IT MUST BE HERE FOR FUTURE USE.** ▪ *Algorithm*: **1.** Uses **For Each objDictionaryEntry In MyBase.Dictionary** LOOP to iterate through the COLLECTION. **2.** Use **CType()** Function to convert to native data type of DictionaryEntry object to ***Employee*** Class type. **3.** CALLS EACH OBJECT'S ***Save()*** Method to PERFORM EITHER AN UPDATE OR INSERT TO DATBASE. ▪ Add Error-Handling code using **Try-Catch-Finally** to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions ▪ Use **throw** statement to re-throw all exceptions

			▪ **THE FOLLOWING CODE NEEDS TO BE ADDED AT THE END OF THE CODE ABO VE IN ORDER TO SUPPORT SAVING TO FILE.** ▪ **IN THIS VERSION OF THE PROJECT WE ARE STILL USING FILES TO SAVE DATA SO WE NEED TO TEMPORARILY PLACE THIS CODE. IN THE FUTURE, WHEN WE ARE WORKING WITH A REAL DATABASE, THIS CODE WILL BE REMOVED AND ONLY THE ABOVE CODE WILL BE NEEDED.** ▪ **YOU SHOULD ALREADY HAVE THIS CODE FROM THE PREVIOUS VERSION OF THE PROJECT, JUST COPY/PASTE**. ▪ File Save algorithm: 1. SAVES the OBJECTS IN THE COLLECTION to the `EmployeeData.txt` file. 2. Algorithm: 3. Open File for WRITTING. 4. Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection. 5. Use `CType()` Function to convert to native data type of `DictionaryEntry` object to ***Employee*** Class type 6. GETS ALL THE PROPERTIES FOR EACH OBJECT IN ARRAY and CREATES A ***Comma-delimited string*** from ALL THE PROPERTIES OF THE OBJECT IN ARRAY. 7. CALLS THE STREAMREADER OBJECT `WriteLine()` Method TO WRITE THE ***Comma-delimited string*** LINE TO FILE. 8. Repeat this process until ALL OBJECTS IN ARRAY HAVE BEEN VISITED AND ITS PROPERTIES WRITTEN TO THE FILE AS A ***Comma-delimited string***. 9. Add Error-Handling code using `Try-Catch-Finally` to handle a ***GENERAL*** Exception.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Sub DataPortal_DeleteObject(By Val Key As String)	None	None	▪ **Implementation of DataPortal_DeleteObject()** method. ▪ A LOOP through the DICTIONARY COLLECTION OBJECT SEARCHING for the OBJECT WHOSE KEY/SSNuber is passed as parameter. WHEN FOUND, calls the OBJECTS ***Delete(Key)*** method to PERMANENTLY DELETE THE OBJECT FROM DATABASE. ▪ **IMPORTANT! THIS CODE IS PROVIDED TO YOU IN THE *Business Collection Class* TEMPLATE. Simply modify as desire**. ▪ Algorithm: **1.** Uses **For Each objDictionaryEntry In MyBase.Dictionary** LOOP to iterate through the COLLECTION. **2.** Use **CType()** Function to convert to native data type of DictionaryEntry object to ***Employee*** Class type. **3.** QUESTION each OBJECT if they are the ***KEY/SSNuber*** being SEARCHED. **4.** WHEN FOUND, CALLS the OBJECT'S ***Delete()*** Method to DELETE THE OBJECT FROM DATABASE. ▪ Add Error-Handling code using **Try-Catch-Finally** to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions ▪ Use **throw** statement to re-throw all exceptions

Public Helper Methods	Parameters	Return Type	Description
Public Function ToArray() As Employee()	**None**	**POINTER TO Employee ARRAY OBJECT Reference or Pointer to ARRAY**	▪ **Implementation of ToArray()** method. ▪ Support FOR DATA BINDING. ▪ METHOD THAT CONVERTS **DICTIONARY COLLECTION OBJECT** TO **ARRAY**. ▪ Returns the ARRAY POINTER ***Employee()***. ▪ Algorithm: **1.** Creates TEMP **ARRAY** of ***Employee*** **2.** USES **DICTIONARY CLASS** CopyTo() Method TO COVERT **DICTIONARY COLLECTION OBJECT** TO TEMP **ARRAY**. **3.** **Return** The POPULATED TEMP **ARRAY**.

Class CustomerList

- ***CustomerList*** COLLECTION Class. Object of this class will store and manage **Customer Objects** in memory and load/save them from ***TEXT FILE***.
- This class encapsulates a ***DICTIONARY COLLECTION*** OBJECT provided via ***INHERITANCE*** from the **BUSINESSCOLLECTIONBASE** Class which ***INHERITS*** from ***DICTIONARYBASE CLASS***.
- The key properties/methods are described as follows:

General Information	Description

`Option Explicit On`
`Option Strict On`
 	
`'Impoted Libraries`
`Imports System.IO`
`Imports System.Data`
`Imports System.Data.OleDb`
`Imports System.Configuration`
	▪ Option Explicit and Option Strict should be On

General Class Information	Description

`<Serializable()> _`
`Public Class CustomerList`
`Inherits`
`BusinessCollectionBase`
	▪ CUSTOM COLLECTION CLASS that encapsulates a DICTIONARY COOLECTION OBJECT that stores **Customer** objects.
	▪ ADD the KEYWORD **`<Serializable()> _`** to enable Serialization for this class.
	▪ Class **`Inherits`** from ***BusinessCollectionBase*** Class.

Public Properties (GET/SET)	Description
`Public Shadows ReadOnly Property Count() As Integer`	READ-ONLY GET: returns NUMBER OF ELEMENTS OR OBJECT IN THE COLLECTION DICTIONARY. Property should shadow the base class equivalent. ▪ GET Algorithm: **1.** Calls BASE CLASS **`MyBase.Dictionary.Count`** Property.
`Public Property Item(ByVal key As String) As Customer`	▪ **Wrapper PROPERTY** that **GET & SET** *Customer* **OBJECTS in the DICTIONARY COLLECTION** ▪ This Property GETS the POINTER to the OBJECT in the COLLECTION based on its KEY. ▪ This Property SETS the OBJECT WHO'S KEY is passed as parameter with the OBJECT being assigned. ▪ GET Algorithm: **1.** CALLS the BASE CLASS **`MyBase.Dictionary.Item(key)`** Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. **2.** Use **`CType()`** Function to convert to native data type of DICTIONARY collection to ***Customer*** Class type. ▪ SET Algorithm: **1.** CALLS the BASE CLASS **`MyBase.Dictionary.Contains`**`(key)` Method determine if KEY exists. **2.** If exists it calls **`MyBase.Dictionary.Item(key)`** to do the work of SETTING the VALUE. **3.** Else, THROWS `Throw New System.ArgumentException("ID Not found")` EXCEPTION indicating KEY WAS NOT FOUND.

Public Methods	Parameters	Return Type	Description
`Public Sub Add(ByVal key As String, ByVal objCustomer As Customer)`	`Customer objCustomer`	`None`	▪ **Implementation of Add(object)** method. ▪ ***Wrapper Method*** that ADDS the object & its associated KEY passed as argument into the collection. ▪ In this case, the KEY is the ***IDNumber*** PROPERTY of the Object. ▪ It is assumed the Object if CREATED & POPULATED in the User-Interface or calling program when passed as argument to the method call. Method simply adds the object to the COLLECTION. ▪ Algorithm: 1. Calls `MyBase.Dictionary.Add(key, objCustomer)` Method to add the object to Collection. 2. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Sub Add(ByVal x, ByVal y,ByVal z, etc.)`	Variable for each required Parameter to SET PROPERTY of the ***Customer*** Class Object. Normally the paramters of the Parameterized Constructor.	`None`	▪ **Implementation of Add(x,y,z etc.)** method. ▪ ***Wrapper Method*** that ADDS object into the collection. ▪ Same functionality as previous ADD, except NO populated OBJECT is passed as parameter, but the individual values that make up the OBJECT. ▪ The method itself creates the Object, populates it with the values from parameters and then ADDS the object to the COLLECTION with the KEY. ▪ In this case, the KEY is the ***IDNumber*** PARAMETER value of the method's parameter list. ▪ This version of ADD, requires less programming in the User-Interface. ▪ Algorithm: 1. Creates either DEFAULT Temporary ***Customer*** OBJECT & SETS the appropriate properties based on parameters VALUES passed to method. Or CREATES PARAMETERIZED Constructor OBJECT, passing to the OBJECT the VALUES of the parameters passed to method. 2. Calls `MyBase.Dictionary.Add(key, objCustomer)` Method to add the object to Collection. The KEY being the ***IDNumber*** PROPERTY of the object created. 3. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Function Edit(ByVal key As String, ByVal objCustomer As Customer) As Boolean`	`Customer objCustomer`	`True` `False`	▪ **Implementation of Edit(object)** method. ▪ ***Wrapper Method*** that EDITS the OBJECT in the COLLECTION whose KEY is passed as parameter to method. ▪ In this case, the KEY is the ***IDNumber*** PROPERTY of the Object ▪ The OBJECT in COLLECTION is edited by SETTING its PROPERTIES, with the values or the PROPERTIES of the OBJECT passed as parameter. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Item(key)` Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use `CType()` Function to convert to native data type of DICTIONARY collection to ***Customer*** Class type. 3. Returns a FALSE if POINTER returned by `MyBase.Dictionary.Item(key)` Property is a ***Nothing***. 4. Else, **GETS PROPERTIES** of Object passed as parameter & **SETS PROPERTIES** of OBJECT in the COLLECTION whose POINTER was returned by `Dictionary.Item(key)` Property & **Returns** a **True**. 5. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Function Edit(ByVal x, ByVal y,ByVal z, etc.) As Boolean`	Variable for each required Parameter to SET PROPERTY of the ***Customer*** Class Object. To be EDITED.	`True` `False`	▪ **Implementation of Edit(x,y,z)** method. ▪ ***Wrapper Method*** that EDITS the OBJECT in the COLLECTION who's associated KEY is passed as ONE of the parameter variables. The OBJECT in COLLECTION is edited by SETTING its PROPERTIES with the values passed as parameters to method, except the KEY parameter. ▪ In this case, the KEY is the ***IDNumber*** PARAMETER value of the method's parameter list. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Item(key)` Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use `CType()` Function to convert to native data type of DICTIONARY collection to ***Customer*** Class type. 3. Returns a FALSE if POINTER returned by `MyBase.Dictionary.Item(key)` Property is a ***Nothing***. 4. Else **SETS PROPERTIES** of OBJECT in the COLLECTION whose POINTER was returned by `MyBase.Dictionary.Item(key)` with values passed as parameters, except the value that represents the KEY & **Returns** a **True**. 5. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Function Remove(ByVal key As String) As Boolean`	`String Key`	`True` `False`	▪ **Implementation of Remove(Key)** method. ▪ ***Wrapper Method*** that REMOVES the object from COLLECTION who's associated KEY is passed as parameter to method. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Contains(key)` Method determine if KEY exists. 2. If exists it calls `MyBase.Dictionary.Remove(key)` to do the work of REMOVING OBJECT from COLLECTION and returns a **TRUE**. 3. Else, if not exists, then it returns a **FALSE**. 4. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Function Print(ByVal key As String) As Boolean`	`String Key`	`True` `False`	▪ **Implementation of Print(Key)** method. ▪ ***Method*** that PRINTS the content of the OBJECT in the COLLECTION who's associated KEY is passed as parameter. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Item(key)` Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use `CType()` Function to convert to native data type of DICTIONARY collection to ***Customer*** Class type. 3. Returns a FALSE if POINTER returned by `Dictionary.Item(key)` Property is a ***Nothing***. 4. Else CALLS the `object.Print()` Method of the OBJECT in the COLLECTION whose POINTER was returned by `MyBase.Dictionary.Item(key)`. 5. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Sub PrintAll()`	None	None	▪ **Implementation of PrintAll()** method. ▪ ***Method*** that PRINTS the content of ALL THE OBJECT in the COLLECTION. ▪ Algorithm: 1. Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection. 2. Use `CType()` Function to convert to native data type of `DictionaryEntry` object to ***Customer*** Class type. 3. CALLS the `object.Print()` Method of EACH OF THE OBJECT in the COLLECTION. 4. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Shadows Sub Clear()`	None	None	▪ **Implementation of Clear()** method. ▪ **Method** uses Shad keyword to Shadow the Base Class Method which already implements this functionality. We are simply repeating the process here. ▪ ***Method*** that CLEARS or DELETES ALL OBJECTS in the COLLECTION. ▪ Algorithm: 1. CALLS `MyBase.Dictionary.Clear()` to do the work. 2. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Overrides Function DeferredDelete(ByVal strKey As String) As Boolean`	`String key`	`Boolean`	▪ **Implementation of DeferredDelete(KEY)** method. ▪ ***Method*** that MARKS AND OBJECT FOR DELETION inside the COLLECTION. ▪ Algorithm: 1. Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection. 2. Use `CType()` Function to convert to native data type of `DictionaryEntry` POINTER to ***Customer*** Class POINTER. 3. Interrogates EACH OBJECT BY COMPARING SEARCH KEY with ID of OBJECT in COLLECTION. 4. If KEY is FOUND, VERIFIES OBJECT IS NOT NEW by getting its `IsNew` Property. 5. If OBJECT IS NOT NEW or OLD, then CALLS the `DeferredDelete()` method of the OBJECT IN COLLECTION to MARK IT FOR DEFERRED DELETION. 6. EXITS the LOOP using `Return True` since OBJECT FOUND & MARKED FOR DELETION & SEARCH IS OVER. 7. Else if Object is NEW then `Return False` since NEW object should NOT BE DELETED but INSERTED 8. If SEARCH AFTER END OF `For Each` is over and KEY WAS NOT FOUND `Return False`. 9. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Data Access Methods	Parameters	Return Type	Description
Public Shared Function Create()As CustomerList	None	CustomerList Reference or Pointer	▪ **Implementation of Create()** method. ▪ Public interface to the data access method that CREATES A NEW OBJECT USING FACTORY METHOD. ▪ The Create() method does not perform the data access but calls upon the DataPortal_Create() method to do the work. ▪ Algorithm: **1.** CALL & Return the DataPortal_Create() method.
Public Sub Load()	String key	None	▪ **Implementation of Load(key)** method. ▪ Public interface to the data access method that retrieves ALL THE ***Customer*** RECORDS FROM the Employee Table and ADDS them to the **DICTIONARY COLLECTION OBJECT**. ▪ The Load() method does not perform the data access but calls upon the DataPortal_Fetch() method to do the work. ▪ Algorithm: **1.** CALL the DataPortal_Fetch() method.
Public Sub Save()	None	None	▪ **Implementation of Save()** method. ▪ The ***Save()*** Method is a very important method. ▪ The Save() method does not perform the work but calls upon the DataPortal_Save() method to do the work. ▪ **IT HAS THE INTELLIGENCE TO DETERMINE IF THE COLLECTION HAS BEEN CHANGED OR NOT BY TESTING THE IsDirty PROPERTY.** ▪ **If OBJECT IS DIRTY IT CALLS THE FOLLOWING METHOD:** ▪ Algorithm: **1.** If Collection **IsDirty** CALL the DataPortal_Save() method. **2.** Otherwise nothing is done.
Public Sub ImmediateDelete(ByVal Key As String)	String key	None	▪ **Implementation of DeleteObject(key)** method. ▪ Public interface to the data access method that SEARCHES AN OBJECT IN THE **DICTIONARY COLLECTION OBJECT** & IMMEDIATELY DELETES the RECORD of the PRIMARY KEY ***KEY/IDNumber*** passed as a parameter FROM DATABASE. ▪ The DeleteObject() method does not perform the work but calls upon the DataPortal_DeleteObject() method to do the work. ▪ Algorithm: **1.** CALL the DataPortal_DeleteObject(Key) method.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Shared Function DataPortal_Create() As CustomerList	None	CustomerList Reference or Pointer	▪ **Implementation of Shared Function DataPortal_Create() method**. ▪ OBJECT FACTORY METHOD. ▪ Creates & RETURNS FULLY READY AND INITIALIZED ***CustomerList*** OBJECTS. ▪ STUB FUNCTION METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY & RETURN KEYWORD.
Protected Sub DataPortal_Fetch(ByVal Key As String)	String key	None	▪ **Implementation of DataPortal_Fetch(Key)** method. ▪ **THE FOLLOWING CODE IS HERE TEMPORARILY TO SUPPORT LOADING FROM A FILE.** ▪ **NEXT PROJECT WE WILL LOAD FROM A REAL DATABASE AND THIS CODE WILL BE REMOVED.** ▪ LOAD the OBJECTS from the CustomerData.txt file and ADDS them to the COLLECTION. ▪ Algorithm: 1. USE File.Exists("CustomerData.txt") to verify if FILE EXISTS. 2. If FILE DOES NOT EXISTS IT CREATES IT using File.Create("CustomerData.txt") Method. 3. Open File for READING. 4. Read a LINE from file & PARSE each comma-delimited line. 5. Create new temporary OBJECT if the Customer Class. 6. SET OBJECT WTH values from LINE READ FROM FILE. 7. ADD OBJECT TO COLLECTION. 8. Repeat this process until EOF. 9. Add Error-Handling code using Try-Catch-Finally to handle a ***GENERAL*** Exception.

Protected Data Access Methods	Parameters	Return Type	Description
`Protected Sub DataPortal_Save()`	None	**`None`**	▪ **Implementation of `DataPortal_Save()`** method. ▪ **YOU WILL IMPLEMENT THIS METHOD AS DICTATED BY THE BUSINESS COLLECTION CLASS TEMPLATE.** ▪ **NEVERTHELESS, the following CODE WILL NOT HELP US IN THIS PROJECT SINCE THE CURRENT BUSINESS CLASSES DON'T PERFORM REAL DATA ACCESS.** ▪ **THEREFORE THE FOLLOWING CODE IS FOR FUTURE USE, BUT IT MUST BE IMPLEMENTED AS SHOWN IN THE BUSINESS COLLECTION CLASS TEMPLATE:** ▪ WHAT ID DOES IS AS FOLLOWS: o LOOPS through the **DICTIONARY COLLECTION OBJECT** & CALLS EACH OBJECTS ***`Save()`*** to SAVE THE OBJECT. o The ides is the **EACH OBJECT IN COLLECTION SAVES ITSELF AND DECIDES WHETHER TO DO AN UPDATE OR INSERT BASED ON THE STATUS OF ITS ISDIRTY**, **ISNEW** AND **ISDELETED** FLAGS. ▪ **IMPORTANT! THIS CODE IS PROVIDED TO YOU IN THE *Business Collection Class* TEMPLATE. Simply COPY/PASTE modify for this CUSTOMERLIST CLASS.** ▪ **KEEP IN MIND THIS CODE HAS NO IMPACT TO THE PROGRAM, IT WILL EXECUTE AND CALL THE SAVE METHOD OF EACH OBJECT BUT THE DATA ACCESS METHODS HAVE NOT BEEN IMPLEMENTED YET. BUT IT MUST BE HERE FOR FUTURE USE.** ▪ *Algorithm*: **1.** Uses **`For Each objDictionaryEntry In MyBase.Dictionary`** LOOP to iterate through the COLLECTION. **2.** Use **`CType()`** Function to convert to native data type of `DictionaryEntry` object to ***Customer*** Class type. **3.** CALLS EACH OBJECT'S ***`Save()`*** Method to PERFORM EITHER AN UPDATE OR INSERT TO DATABASE. ▪ Add Error-Handling code using **`Try-Catch-Finally`** to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions ▪ Use **`throw`** statement to re-throw all exceptions

			▪ **THE FOLLOWING CODE NEEDS TO BE ADDED AT THE END OF THE CODE ABO VE IN ORDER TO SUPPORT SAVING TO FILE.** ▪ **IN THIS VERSION OF THE PROJECT WE ARE STILL USING FILES TO SAVE DATA SO WE NEED TO TEMPORARILY PLACE THIS CODE. IN THE FUTURE, WHEN WE ARE WORKING WITH A REAL DATABASE, THIS CODE WILL BE REMOVED AND ONLY THE ABOVE CODE WILL BE NEEDED.** ▪ **YOU SHOULD ALREADY HAVE THIS CODE FROM THE PREVIOUS VERSION OF THE PROJECT, JUST COPY/PASTE**. ▪ File Save algorithm: 1. SAVES the OBJECTS IN THE COLLECTION to the `CustomerData.txt` file. 2. Algorithm: 3. Open File for WRITING. 4. Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection. 5. Use `CType()` Function to convert to native data type of `DictionaryEntry` object to ***Customer*** Class type 6. GETS ALL THE PROPERTIES FOR EACH OBJECT IN ARRAY and CREATES A ***Comma-delimited string*** from ALL THE PROPERTIES OF THE OBJECT IN ARRAY. 7. CALLS THE STREAMREADER OBJECT `WriteLine()` Method TO WRITE THE ***Comma-delimited string*** LINE TO FILE. 8. Repeat this process until ALL OBJECTS IN ARRAY HAVE BEEN VISITED AND ITS PROPERTIES WRITTEN TO THE FILE AS A ***Comma-delimited string***. 9. Add Error-Handling code using `Try-Catch-Finally` to handle a ***GENERAL*** Exception.

Protected Data Access Methods	Parameters	Return Type	Description
`Protected Sub DataPortal_DeleteObject(ByVal Key As String)`	None	`None`	▪ **Implementation of `DataPortal_DeleteObject()`** method. ▪ LOOPS through the **DICTIONARY COLLECTION OBJECT** SEARCHING for the OBJECT WHOSE ***KEY/IDNumber*** is passed as parameter. WHEN FOUND, calls the OBJECTS ***`Delete(Key)`*** method to PERMANENTLY DELETE THE OBJECT FROM DATABASE. ▪ **IMPORTANT! THIS CODE IS PROVIDED TO YOU IN THE *Business Collection Class* TEMPLATE. Simply modify as desire**. ▪ Algorithm: **1.** Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the COLLECTION. **2.** Use `CType()` Function to convert to native data type of `DictionaryEntry` object to ***Customer*** Class type. **3.** QUESTION each OBJECT if they are the ***KEY/IDNumber*** being SEARCHED. **4.** WHEN FOUND, CALLS the OBJECT'S ***`Delete()`*** Method to DELETE THE OBJECT FROM DATABASE. ▪ Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions ▪ Use `throw` statement to re-throw all exceptions

Public Helper Methods	Parameters	Return Type	Description
`Public Function ToArray() As Customer()`	`None`	**POINTER TO Employee ARRAY OBJECT Reference or Pointer to ARRAY**	▪ **Implementation of `ToArray()`** method. ▪ Support FOR DATA BINDING. ▪ METHOD THAT CONVERTS **DICTIONARY COLLECTION OBJECT** TO **ARRAY**. ▪ Returns the ARRAY POINTER ***`Customer()`***. ▪ Algorithm: **1.** Creates TEMP **ARRAY** of ***`Customer`*** **2.** USES **DICTIONARY CLASS** `CopyTo()` Method TO COVERT **DICTIONARY COLLECTION OBJECT** TO TEMP **ARRAY**. **3.** `Return` The POPULATED TEMP **ARRAY**.

Class Product

- ❑ Represents ***Products*** to be sold, serviced or rented by a company. Intended to be used as a base class
- ❑ The key properties/methods are described as follows:

General Information	Description

`Imports System.IO`
`Imports System.Data`
`Imports System.Data.OleDb`
`Imports System.Configuration`
 	
`'Keep commented (for future use)`
`'System.Runtime.Serialization.Formatters.Binary`
`'Imports System.Runtime.Remoting`
`'Imports System.Runtime.Remoting.Channels`
`System.Runtime.Remoting.Channels.Http`
	▪ Option Explicit and Option Strict should be On
	▪ Imported .NET libraries to support:
	- File/IO
	- Database support for ADO.NET Data Access Technology
	
	▪ Commented imported .NET libraries for future use to support:
	- Serialization
	- Remoting

PUBLIC Enumerated Data Type Declarations	Description
`Public Enum Rating`	▪ Purpose: This is an enumerated Data type declaration that represents product's rating, example: PG, PG13, and R. ▪ Data type: **`Enumerated`** ▪ Enumerated Values: *G, PG, PG-13, NC-17, R* , None

General Class Information	Description

`<Serializable()> _`
`Public MustInherit Class Product`
`    Inherits BusinessBase`
	▪ **`MustInherit`** Base Class – Designed for inheritance only. No objects of this class should or can be created.
	▪ Represents the products to be sold, rented, etc., in the business.
	▪ ADD the KEYWORD `<Serializable()> _` to enable Serialization for this class.
	▪ Class **`Inherits`** from ***BusinessBase*** Class.

Private Data	Description
`Private` m_IDNumber	Purpose: Represents product's ID number. Data type: **`String`**
`Private` m_Title	Purpose: Represents product's name. Data type: **`String`**
`Private` m_Description	Purpose: Represents product's description. Data type: **`String`**
`Private` **m_enumRating**	▪ Purpose: This is variable of the **`Enum`** **`Rating`** Data which will store only **`Rating`** variables. ▪ Data type: **`Rating`** ▪ Enumerated Values: **`Rating`**.*G*, **`Rating`**.*PG*, **`Rating`**.*PG13*, **`Rating`**.*NC17*, **`Rating`**.*R*, **`Rating`**.*None*.
`Private` m_Available	Purpose: Represents product's availability, in stock (true or false). Data type: **`Boolean`**
`Private` m_SalePrice	Purpose: Represents product's sales price. Data type: **`Decimal`**
`Private` m_RentalRate	Purpose: Represents product's daily rental rate. Data type: **`Decimal`**
`Private` m_LateFee	Purpose: Represents product's daily late fees for rentals. Data type: **`Decimal`**

Public Properties (GET/SET)	Description
Public IDNumber	▪ GET/SET **m_IDNumber** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public Title	▪ GET/SET **m_Title** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public Description	▪ GET/SET **m_Description** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public Rating	▪ GET/SET **m_Rating** private data. ▪ Data type: Rating ▪ Enumerated Values: Rating.***G***, Rating.***PG***, Rating.***PG13***, Rating.***NC17***, Rating.***R***, Rating.***None***. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public Available	▪ GET/SET **m_Available** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public SalePrice	▪ GET/SET **m_SalePrice** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public RentalRate	▪ GET/SET **m_RentalRate** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public LateFee	▪ GET/SET **m_LateFee** private data. ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.

Constructors	Parameters	Return Type	Description
Public New()	None	N/A	▪ **Default Constructor**. Should initialize the PRIVATE DATA members with appropriate default values ▪ Sets to appropriate default values: - m_IDNumber, m_Title, m_Description =”” - m_enumRating = Rating.*None* - m_Available = **True** - m_SalePrice, m_RentalRate, m_LateFee = 0.0
Public New(x, y, z etc..)	▪ A parameter to Set each of the following **Properties** only (AVAILABLE not included): - IDNumber - Title - Description - Rating - SalePrice - RentalRate - LateFee ▪ Should have a total of 7 parameters: - **par1 to Par7** ▪ Name the parameters as you see fit. ▪ All parameters are Pass-by-Value	N/A	▪ **Parameterized Constructor** ▪ Sets the following **PROPERTIES** to matching parameter list, EXCEPT the AVAILABLE which should be set by its’ Private Data directly: - IDNumber = **par1** - Title = **par2** - Description = **par3** - Rating = **par4** - SalePrice = **par5** - RentalRate = **par6** - LateFee = **par7** **5.** The Available **PROPERTY** is not part of the parameter so it needs to be defaulted: - **m_Available** = **True**

Public MustOverride Methods	Parameters	Return Type	Description
Public MustOverride Sub Print()	None	None	▪ MustOverride **Print()** method. ▪ Intended for derived classes to print their data ▪ **Declaration only**. Must be implemented in derived classes.
Public MustOverride Sub Product_Rental()	None	None	▪ MustOverride **Product_Rental()** method. ▪ Intended for derived classes to be able to perform Product Rentals. ▪ **Declaration only**. Must be implemented in derived classes.
Public MustOverride Sub Product_Return()	None	None	▪ MustOverride **Product_Return()** method. ▪ Intended for derived classes to be able to perform Product Returns as part of rental process. ▪ **Declaration only**. Must be implemented in derived classes.
Public MustOverride Sub Product_Sell()	None	None	▪ MustOverride **Product_Sell()** method. ▪ Intended for derived classes to be able to SELL the Products of the business. ▪ **Declaration only**. Must be implemented in derived classes.

Class DVD

- ❑ Represents the DVD to be sold and rented.
- ❑ The key properties/methods are described as follows:

General Information	Description
Imports System.IO Imports System.Data Imports System.Data.OleDb Imports System.Configuration 'Keep commented (for future use) 'System.Runtime.Serialization.Formatters.Binary 'Imports System.Runtime.Remoting 'Imports System.Runtime.Remoting.Channels System.Runtime.Remoting.Channels.Http	▪ Option Explicit and Option Strict should be On ▪ Imported .NET libraries to support: - File/IO - Database support for ADO.NET Data Access Technology ▪ Commented imported .NET libraries for future use to support: - Serialization - Remoting

PUBLIC Enumerated Data Type Declarations	Description
Public Enum MovieCategory	Purpose: Enumerated Data type that represents the movie category such as: **Action, Drama, Comedy etc.** Data type: **enum** Enumerated Values: ***Action_Adventure, Drama, Famil_Kids, Horror, Sci-Fi_Fantasy, Music, Sports, Romance, Comedy, Western, None***
Public Enum DVDFormat	Purpose: Enumerated Data type that represents the DVD movie format: **DVD, HDVD, Blue-ray etc.** Data type: **enum** Enumerated Values: ***DVD, HD-DVD, BLU-RAY DISC, None***

General Class Information	Description
<Serializable()> _ **Public Class DVD** **Inherits Product**	▪ Defines blueprint for **DVD** OBJECTS to be rented and sold in the business. ▪ ADD the KEYWORD <Serializable()> _ to enable Serialization for this class. ▪ Class **Inherits** from ***Product*** Class.

Private Data	Description
Private m_enumMovieCategory	▪ Purpose: Enumerated Data type that represents the movie category such as: **Action, Drama, Comedy etc.** ▪ Data type: MovieCategory ▪ Enumerated Values: MovieCategory.***Action_Adventure,*** MovieCategory.***Drama,*** MovieCategory.***Family_Kids,*** MovieCategory.***Horror,*** MovieCategory.***SciFi_Fantasy,*** MovieCategory.***Music,*** MovieCategory.***Sports,*** MovieCategory.***Romance,*** MovieCategory.***Western,*** MovieCategory.***Comedy,*** MovieCategory.***None***
Private m_enumDVDFormat	▪ Purpose: Enumerated Data type that represents the DVD movie format: **DVD, HDVD, Blue-ray etc.** ▪ Data type: DVDFormat ▪ Enumerated Values: DVDFormat.***DVD,*** DVDFormat.***HDDVD,*** DVDFormat.***BLURAY DISC,*** DVDFormat.***None***

Public Properties (GET/SET)	Description
Public **Category**	▪ **GET/SET enumMovieCategory private data** ▪ Enumerated values that can be assigned to this property: MovieCategory.***Action_Adventure***, MovieCategory.***Drama***, MovieCategory.***Family_Kids***, MovieCategory.***Horror***, MovieCategory.***SciFi_Fantasy***, MovieCategory.***Music***, MovieCategory.***Sports***, MovieCategory.***Romance***, MovieCategory.***Western***, MovieCategory.***Comedy***, MovieCategory.***None*** ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public **Format**	▪ **GET/SET enumDVDFormat private data** ▪ Enumerated values that can be assigned to this property: DVDFormat.***DVD***, DVDFormat.***HDDVD***, DVDFormat.***BLURAY DISC***, DVDFormat.***None*** ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.

Constructors	Parameters	Return Type	Description
Public New()	None	**N/A**	▪ **Default Constructor**. ▪ Should initialize the private data members with appropriate default values for this Class and the Base Class. ▪ Calls **Base Class Default Constructor** ▪ Sets to appropriate default values: - m_enumMovieCategory = MovieCategory.***None*** - m_enumDVDFormat = DVDFormat.***None***
Public New(x, y, z etc..)	▪ A parameter to Set each of the following Base Class & THIS class **Properties** (AVAILABLE not included): - IDNumber - Title - Description - Rating - SalePrice - RentalRate - LateFee - Category - Format ▪ Should have a total of 9 parameters: - **par1 to Par9** ▪ Name the parameters as you see fit. ▪ All parameters are Pass-by-Value	**N/A**	▪ **Parameterized Constructor** ▪ Sets the **PROPERTIES** of the Person Base Class and this ***DVD*** Class matching parameter list. ▪ Calls **Base Class Parameterized Constructor** to handle the following parameters: - IDNumber = **par1** - Title = **par2** - Description = **par3** - Rating = **par4** - SalePrice = **par5** - RentalRate = **par6** - LateFee = **par7** ▪ Sets the **PROPERTIES** with the following parameters: - m_enumMovieCategory = **par8** - m_enumDVDFormat = **par9**

Public Overrides Methods	Parameters	Return Type	Description
`Public Overrides Sub Print()`	None	`None`	▪ **Implementation of Print()** method. ▪ The *Print()* Method WRITES ALL OBJECT'S DATA TO THE PRINTER FILE as follows: **1.** Opens ***Network_Printer.txt*** file for APPENDING. **2.** Write each object's property/data in the following FORMAT: Printing ***DVD*** .... ID Number = ***value*** Title = ***value*** Description = ***value*** Rating = ***value*** SalePrice = ***value*** RentalRate = ***value*** Etc…. **3.** Close the file ▪ Add Error-Handling code using `try-catch-finally` to handle all required exceptions for any file access, array or general exceptions ▪ Follow best practice of trapping for unexpected general errors in addition to specific errors ▪ Use `throw` statement to re-throw all exceptions
`Public Overrides Sub Product_Rental()`	None	`None`	▪ **Implementation of Product_Rental()** method. ▪ `Overrides` to satisfy `MustOverride` Base Class requirements. ▪ Handles future product rental transactions. ▪ NO IMPLEMENTATION, STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY.
`Public Overrides Sub Product_Return()`	None	`None`	▪ **Implementation of Product_Return()** method. ▪ `Overrides` to satisfy `MustOverride` Base Class requirements. ▪ Handles future product returns transactions as part of rental process. ▪ NO IMPLEMENTATION, STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY.
`Public Overrides Sub Product_Sell()`	None	`None`	▪ **Implementation of Product_Sell()** method. ▪ `Overrides` to satisfy `MustOverride` Base Class requirements. ▪ Handles future product selling transactions. ▪ NO IMPLEMENTATION, STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY.

Public Data Access Methods	Parameters	Return Type	Description
Public Shared Function Create()As DVD	None	DVD Reference or Pointer	▪ **Implementation of Create()** method. ▪ Public interface to the data access method that CREATES A NEW OBJECT USING FACTORY METHOD. ▪ The Create() method does not perform the data access but calls upon the DataPortal_Create() method to do the work. ▪ Algorithm: **1.** CALL & Return the DataPortal_Create() method.
Public Sub Load(key)	String key	None	▪ **Implementation of Load(key)** method. ▪ Public interface to the data access method that retrieves the RECORD of the PRIMARY KEY ***KEY/IDNumber*** passed as a parameter. ▪ The Load(key) method does not perform the data access but calls upon the DataPortal_Fetch(Key) method to do the work. ▪ Algorithm: **1.** CALL the DataPortal_Fetch(Key) method.
Public Sub Save()	None	None	▪ **Implementation of Save()** method. ▪ The ***Save()*** Method is a very important method. ▪ **This Method uses the BusinessBase Class IsDirty, IsNew & IsDeleted TO DETERMINE WHICH DATA BASE OPERATION TO PERFORM ON THE OBJECT, EITHER INSERT, UPDATE OR DELETE**. ▪ The code for this method is provided in your BUSINESS CLASS TEMPLATE. Nevertheless, I will include the algorithm here. ▪ Algorithm: **1.** IF OBJECT is marked for deletion via its Me.IsDeleted FLAG or OBJECT is Not NEW via its New FLAG Me.IsNew then CALL DataPortal_Delete(Me.IDNumber) to DELETE RECORD FROM DATABASE. **2.** ELSE if OBJECT IS DIRTY via Me.IsDirty and IF OBJECT is NEW via Me.IsNew then CALL DataPortal_Insert() TO ADD RECORD TO DATABASE **3.** ELSE CALL DataPortal_Update() TO UPDATE THE RECORD IN THE DATABASE.
Public Sub ImmediateDelete(ByVal Key As String)	String key	None	▪ **Implementation of Delete(key)** method. ▪ Public interface to the data access method that IMMEDIATELY DELETES the RECORD of the PRIMARY KEY ***KEY/SSNunber*** passed as a parameter FROM DATABASE. ▪ Algorithm: **1.** CALL the DataPortal_ Delete(Key) method.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Shared Function DataPortal_Create() As DVD	None	DVD Reference or Pointer	▪ **Implementation of Shared Function DataPortal_Create() method**. ▪ OBJECT FACTORY METHOD. ▪ Creates & RETURNS FULLY READY AND INITIALIZED ***DVD*** OBJECTS. ▪ STUB FUNCTION METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY & RETURN KEYWORD.
Protected Sub DataPortal_Fetch(ByVal Key As String)	String key	None	▪ **Implementation of DataPortal_Fetch(Key)** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method MyBase.MarkOld()
Protected Sub DataPortal_Update()	None	None	▪ **Implementation of DataPortal_Update()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method MyBase.MarkOld()
Protected Sub DataPortal_Insert()	None	None	▪ **Implementation of DataPortal_Insert()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method MyBase.MarkOld()
Protected Sub DataPortal_Delete(ByVal Key As String)	None	None	▪ **Implementation of DataPortal_Delete()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as NEW by calling method MyBase.New() since recod is no longer in the database and has been deleted. At this point, the object is a new object.

Class VideoGame

- Represents the Video Game to be sold and rented.
- The key properties/methods are described as follows:

General Information	Description
Imports System.IO Imports System.Data Imports System.Data.OleDb Imports System.Configuration 'Keep commented (for future use) 'System.Runtime.Serialization.Formatters.Binary 'Imports System.Runtime.Remoting 'Imports System.Runtime.Remoting.Channels System.Runtime.Remoting.Channels.Http	▪ Option Explicit and Option Strict should be On ▪ Imported .NET libraries to support: - File/IO - Database support for ADO.NET Data Access Technology ▪ Commented imported .NET libraries for future use to support: - Serialization - Remoting

PUBLIC Enumerated Data Type Declarations	Description
Public Enum VideoGameCategory	▪ Purpose: Enumerated Data type that represents the video game category such as: **Action, shooting, Racing etc.** ▪ Data type: **Enum** ▪ Enumerated Values: ***Action, Roleplaying, Shooting, Fighting, Racing, Sports, Strategy, Horror, Flight Simulators, Online, Rhythm, None***
Public Enum VideoGameFormat	▪ Purpose: Enumerated Data type that represents the video game format or type of Game Stations the game is intended for: **XBox, Play Station etc.** ▪ Data type: **enum** ▪ Enumerated Values: ***XBox, XBox 360, PS3, PS2,GameCube, DS, Wii, PC, None***

General Class Information	Description
<Serializable()> _ **Public Class VideoGame** **Inherits Product**	▪ Defines blueprint for **VideoGame** OBJECTS to be rented and sold in the business. ▪ ADD the KEYWORD <Serializable()> _ to enable Serialization for this class. ▪ Class **Inherits** from ***Product*** Class.

Private Data	Description
Private **m_enumVideoGameCategory**	▪ Purpose: Variable of the **enum VideoGameCategory** Enumerated Data type that represents the video game category such as: **Action, shooting, Racing etc.** ▪ Data type: **VideoGameCategory** ▪ Values assigned: **VideoGameCategory*.Action*, VideoGameCategory*.Role-playing*, VideoGameCategory*.Shooting*, VideoGameCategory*.Fighting*, VideoGameCategory*.Racing*, VideoGameCategory*.Sports*, VideoGameCategory*.Strategy*, VideoGameCategory*.Horror*, VideoGameCategory*.Flight Simulators*, VideoGameCategory*.Online*, VideoGameCategory*.Rhythm*, VideoGameCategory*.None***
Private **m_enumVideoGameFormat**	▪ Purpose: Enumerated Data type that represents the video game format or type of Game Stations the game is intended for: **XBox, Play Station etc.** ▪ Data type: **VideoGameFormat** ▪ Enumerated Values: **VideoGameFormat*.XBox*, VideoGameFormat*.XBox 360*, VideoGameFormat*.PS3*, VideoGameFormat*.PS2*, VideoGameFormat*.GameCube*, VideoGameFormat*.DS*, VideoGameFormat*.Wii*, VideoGameFormat*.PC*, VideoGameFormat*.None***

Public Properties (GET/SET)	Description
Public **Category**	▪ **GET/SET enumVideoGameCategory private data** ▪ Enumerated values that can be assigned to this property: VideoGameCategory.***Action***, VideoGameCategory.***RolePlaying***, VideoGameCategory.***Shooting***, VideoGameCategory.***Fighting***, VideoGameCategory.***Racing***, VideoGameCategory.***Sports***, VideoGameCategory.***Strategy***, VideoGameCategory.***Horror***, VideoGameCategory.***Flight Simulators***, VideoGameCategory.***Online***, VideoGameCategory.***Rhythm***, VideoGameCategory.***None*** ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.
Public **Format**	▪ **GET/SET enumVideoGameFormat private data** ▪ Enumerated values that can be assigned to this property: VideoGameFormat.***XBox***, VideoGameFormat.***-Box 360***, VideoGameFormat.***PS3***, VideoGameFormat.***PS2***, VideoGameFormat.***GameCube***, VideoGameFormat.***DS***, VideoGameFormat.***Wii***, VideoGameFormat.***PC***, VideoGameFormat.***None*** ▪ In SET portion: To support the BUSINESS RULES, DATA ACCESS & VALIDATION MECHANISM, you MUST CALL **MyBase.MarkDirty()** method after setting the private data.

Constructors	Parameters	Return Type	Description
Public New()	None	**N/A**	▪ **Default Constructor**. ▪ Should initialize the private data members with appropriate default values for this Class and the Base Class. ▪ Calls **Base Class Default Constructor** ▪ Sets to appropriate default values: - m_enumVideoGameCategory = VideoGameCategory.***None*** - m_enumVideoGameFormat = VideoGameFormat.***None***
Public New(x, y, z etc..)	▪ A parameter to Set each of the following Base Class & THIS class **Properties** (AVAILABLE not included): - IDNumber - Title - Description - Rating - SalePrice - RentalRate - LateFee - Category - Format ▪ Should have a total of 9 parameters: - **par1 to Par9** ▪ Name the parameters as you see fit. ▪ All parameters are Pass-by-Value	**N/A**	▪ **Parameterized Constructor** ▪ Sets the **PROPERTIES** of the Person Base Class and this ***VideoGame*** Class matching parameter list. **1.** Calls **Base Class Parameterized Constructor** to handle the following parameters: - IDNumber = **par1** - Title = **par2** - Description = **par3** - Rating = **par4** - SalePrice = **par5** - RentalRate = **par6** - LateFee = **par7** **2.** Sets the **PROPERTIES** of this class with the following parameters: - **Category** = **par8** - **Format** = **par9**

Public Overrides Methods	Parameters	Return Type	Description
`Public Overrides Sub Print()`	None	`None`	▪ **Implementation of Print()** method. ▪ The *Print()* Method WRITES ALL OBJECT'S DATA TO THE PRINTER FILE as follows: **1.** Opens ***Network_Printer.txt*** file for APPENDING. **2.** Write each object's property/data in the following FORMAT: Printing ***VideoGame*** .... ID Number = ***value*** Title = ***value*** Description = ***value*** Rating = ***value*** SalePrice = ***value*** RentalRate = ***value*** Etc…. **3.** Close the file ▪ Add Error-Handling code using `try-catch-finally` to handle all required exceptions for any file access, array or general exceptions ▪ Follow best practice of trapping for unexpected general errors in addition to specific errors ▪ Use `throw` statement to re-throw all exceptions
`Public Overrides Sub Product_Rental()`	None	`None`	▪ **Implementation of Product_Rental()** method. ▪ `Overrides` to satisfy `MustOverride` Base Class requirements. ▪ Handles future product rental transactions. ▪ NO IMPLEMETATION, STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY.
`Public Overrides Sub Product_Return()`	None	`None`	▪ **Implementation of Product_Return()** method. ▪ `Overrides` to satisfy `MustOverride` Base Class requirements. ▪ Handles future product returns transactions as part of rental process. ▪ NO IMPLEMETATION, STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY.
`Public Overrides Sub Product_Sell()`	None	`None`	▪ **Implementation of Product_Sell()** method. ▪ `Overrides` to satisfy `MustOverride` Base Class requirements. ▪ Handles future product selling transactions. ▪ NO IMPLEMETATION, STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY.

Public Data Access Methods	Parameters	Return Type	Description
Public Shared Function Create()As VideoGame	None	VideoGame Reference or Pointer	▪ **Implementation of Create()** method. ▪ Public interface to the data access method that CREATES A NEW OBJECT USING FACTORY METHOD. ▪ The Create() method does not perform the data access but calls upon the DataPortal_Create() method to do the work. ▪ Algorithm: **1.** CALL & Return the DataPortal_Create() method.
Public Sub Load(key)	String key	None	▪ **Implementation of Load(key)** method. ▪ Public interface to the data access method that retrieves the RECORD of the PRIMARY KEY ***KEY/IDNumber*** passed as a parameter. ▪ The Load(key) method does not perform the data access but calls upon the DataPortal_Fetch(Key) method to do the work. ▪ Algorithm: **1.** CALL the DataPortal_Fetch(Key) method.
Public Sub Save()	None	None	▪ **Implementation of Save()** method. ▪ The ***Save()*** Method is a very important method. ▪ **This Method uses the BusinessBase Class IsDirty, IsNew & IsDeleted TO DETERMINE WHICH DATA BASE OPERATION TO PERFORM ON THE OBJECT, EITHER INSERT, UPDATE OR DELETE**. ▪ The code for this method is provided in your BUSINESS CLASS TEMPLATE. Nevertheless, I will include the algorithm here. ▪ Algorithm: **1.** IF OBJECT is marked for deletion via its Me.IsDeleted FLAG or OBJECT is Not NEW via its New FLAG Me.IsNew then CALL DataPortal_Delete(Me.IDNumber) to DELETE RECORD FROM DATABASE. **2.** ELSE if OBJECT IS DIRTY via Me.IsDirty and IF OBJECT is NEW via Me.IsNew then CALL DataPortal_Insert() TO ADD RECORD TO DATABASE **3.** ELSE CALL DataPortal_Update() TO UPDATE THE RECORD IN THE DATABASE.:
Public Sub ImmediateDelete(ByVal Key As String)	String key	None	▪ **Implementation of Delete(key)** method. ▪ Public interface to the data access method that IMMEDIATELY DELETES the RECORD of the PRIMARY KEY ***KEY/SSNunber*** passed as a parameter FROM DATABASE. ▪ Algorithm: **2.** CALL the DataPortal_ Delete(Key) method.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Shared Function DataPortal_Create() As VideoGame	None	VideoGame Reference or Pointer	▪ **Implementation of** Shared Function **DataPortal_Create() method**. ▪ OBJECT FACTORY METHOD. ▪ Creates & RETURNS FULLY READY AND INITIALIZED *VideoGame* OBJECTS. ▪ STUB FUNCTION METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY & RETURN KEYWORD.
Protected Sub DataPortal_Fetch(ByVal Key As String)	String key	None	▪ **Implementation of DataPortal_Fetch(Key)** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method `MyBase.MarkOld()`
Protected Sub DataPortal_Update()	None	None	▪ **Implementation of DataPortal_Update()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method `MyBase.MarkOld()`
Protected Sub DataPortal_Insert()	None	None	▪ **Implementation of DataPortal_Insert()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as OLD by calling method `MyBase.MarkOld()`
Protected Sub DataPortal_Delete(ByVal Key As String)	None	None	▪ **Implementation of DataPortal_Delete()** method. ▪ STUB METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY with exception of the ONLY CODE THAT SHOULD BE AT THE VERY END OF THIS METHOD: **1.** Marks the object as NEW by calling method `MyBase.New() since recod is no longer in the database and has been deleted.  At this point, the object is a new object.`

Class DVDList

- ❑ ***DVDList*** COLLECTION Class. Object of this class will store and manage **DVD Objects** in memory and load/save them from ***TEXT FILE***.
- ❑ This class encapsulates a ***DICTIONARY COLLECTION*** OBJECT provided via ***INHERITANCE*** from the **BUSINESSCOLLECTIONBASE** Class which ***INHERITS*** from ***DICTIONARYBASE CLASS***.
- ❑ The key properties/methods are described as follows:

General Information	Description
Option Explicit On Option Strict On 'Impoted Libraries Imports System.IO Imports System.Data Imports System.Data.OleDb Imports System.Configuration	▪ Option Explicit and Option Strict should be On

General Class Information	Description
<Serializable() > _ **Public Class DVDList** **Inherits** **BusinessCollectionBase**	▪ CUSTOM COLLECTION CLASS that encapsulates a DICTIONARY COLLECTION OBJECT that stores **DVD** objects. ▪ ADD the KEYWORD **<Serializable() > _** to enable Serialization for this class. ▪ Class **Inherits** from ***BusinessCollectionBase*** Class.

Public Properties (GET/SET)	Description
Public Shadows ReadOnly Property Count() As Integer	READ-ONLY GET: returns NUMBER OF ELEMENTS OR OBJECT IN THE COLLECTION DICTIONARY. Property should shadow the base class equivalent. ▪ GET Algorithm: **1.** Calls BASE CLASS **MyBase.Dictionary.Count** Property.
Public Property Item(ByVal key As String) As DVD	▪ **Wrapper PROPERTY** that **GET & SET** *DVD* **OBJECTS in the DICTIONARY COLLECTION** ▪ This Property GETS the POINTER to the OBJECT in the COLLECTION based on its KEY. ▪ This Property SETS the OBJECT WHO'S KEY is passed as parameter with the OBJECT being assigned. ▪ GET Algorithm: **1.** CALLS the BASE CLASS **MyBase.Dictionary.Item(key)** Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. **2.** Use **CType()** Function to convert to native data type of DICTIONARY collection to ***DVD*** Class type. ▪ SET Algorithm: **1.** CALLS the BASE CLASS **MyBase.Dictionary.Contains**(key) Method determine if KEY exists. **2.** If exists it calls **MyBase.Dictionary.Item(key)** to do the work of SETTING the VALUE. **3.** Else, THROWS Throw New System.ArgumentException("ID Not found") EXCEPTION indicating KEY WAS NOT FOUND.

Public Methods	Parameters	Return Type	Description
`Public Sub Add(ByVal key As String, ByVal objDVD As DVD)`	`DVD objDVD`	`None`	▪ **Implementation of Add(object)** method. ▪ ***Wrapper Method*** that ADDS the object & its associated KEY passed as argument into the collection. ▪ In this case, the KEY is the ***IDNumber*** PROPERTY of the Object. ▪ It is assumed the Object if CREATED & POPULATED in the User-Interface or calling program when passed as argument to the method call. Method simply adds the object to the COLLECTION. ▪ Algorithm: 1. Calls `MyBase.Dictionary.Add(key, objDVD)` Method to add the object to Collection. 2. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Sub Add(ByVal x, ByVal y,ByVal z, etc.)`	`Variable for each required Parameter to SET PROPERTY of the DVD Class Object. Normally the paramters of the Parameterized Constructor.`	`None`	▪ **Implementation of Add(x,y,z etc.)** method. ▪ ***Wrapper Method*** that ADDS object into the collection. ▪ Same functionality as previous ADD, except NO populated OBJECT is passed as parameter, but the individual values that make up the OBJECT. ▪ The method itself creates the Object, populates it with the values from parameters and then ADDS the object to the COLLECTION with the KEY. ▪ In this case, the KEY is the ***IDNumber*** PARAMETER value of the method's parameter list. ▪ This version of ADD, requires less programming in the User-Interface. ▪ Algorithm: 1. Creates either DEFAULT Temporary ***DVD*** OBJECT & SETS the appropriate properties based on parameters VALUES passed to method. Or CREATES PARAMETERIZED Constructor OBJECT, passing to the OBJECT the VALUES of the parameters passed to method. 2. Calls `MyBase.Dictionary.Add(key, objDVD)` Method to add the object to Collection. The KEY being the ***IDNumber*** PROPERTY of the object created. 3. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Function Edit(ByVal key As String, ByVal objDVD As DVD) As Boolean`	`DVD objDVD`	**`True`** **`False`**	▪ **Implementation of Edit(object)** method. ▪ ***Wrapper Method*** that EDITS the OBJECT in the COLLECTION whose KEY is passed as parameter to method. ▪ In this case, the KEY is the ***IDNumber*** PROPERTY of the Object ▪ The OBJECT in COLLECTION is edited by SETTING its PROPERTIES, with the values or the PROPERTIES of the OBJECT passed as parameter. ▪ Algorithm: 1. CALLS the BASE CLASS **`MyBase.Dictionary.Item(key)`** Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use **`CType()`** Function to convert to native data type of DICTIONARY collection to ***DVD*** Class type. 3. Returns a FALSE if POINTER returned by **`MyBase.Dictionary.Item(key)`** Property is a ***Nothing***. 4. Else, **GETS PROPERTIES** of Object passed as parameter & **SETS PROPERTIES** of OBJECT in the COLLECTION whose POINTER was returned by `Dictionary.Item(key)` Property & **Returns** a **True**. 5. Add Error-Handling code using **`Try-Catch-Finally`** to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Function Edit(ByVal x, ByVal y,ByVal z, etc.) As Boolean`	`Variable for each required Parameter to SET PROPERTY of the` ***`DVD`*** `Class Object. To be EDITED.`	**`True`** **`False`**	▪ **Implementation of Edit(x,y,z)** method. ▪ ***Wrapper Method*** that EDITS the OBJECT in the COLLECTION who's associated KEY is passed as ONE of the parameter variables. The OBJECT in COLLECTION is edited by SETTING its PROPERTIES with the values passed as parameters to method, except the KEY parameter. ▪ In this case, the KEY is the ***IDNumber*** PARAMETER value of the method's parameter list. ▪ Algorithm: 1. CALLS the BASE CLASS **`MyBase.Dictionary.Item(key)`** Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use **`CType()`** Function to convert to native data type of DICTIONARY collection to ***DVD*** Class type. 3. Returns a FALSE if POINTER returned by **`MyBase.Dictionary.Item(key)`** Property is a ***Nothing***. 4. Else **SETS PROPERTIES** of OBJECT in the COLLECTION whose POINTER was returned by **`MyBase.Dictionary.Item(key)`** with values passed as parameters, except the value that represents the KEY & **Returns** a **True**. 5. Add Error-Handling code using **`Try-Catch-Finally`** to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
Public Function Remove(ByVal key As String) As Boolean	String Key	True False	▪ **Implementation of Remove(Key)** method. ▪ ***Wrapper Method*** that REMOVES the object from COLLECTION who's associated KEY is passed as parameter to method. ▪ Algorithm: 1. CALLS the BASE CLASS **MyBase.Dictionary.Contains**(key) Method determine if KEY exists. 2. If exists it calls **MyBase.Dictionary.Remove(key)** to do the work of REMOVING OBJECT from COLLECTION and returns a **TRUE**. 3. Else, if not exists, then it returns a **FALSE**. 4. Add Error-Handling code using **Try-Catch-Finally** to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
Public Function Print(ByVal key As String) As Boolean	String Key	True False	▪ **Implementation of Print(Key)** method. ▪ ***Method*** that PRINTS the content of the OBJECT in the COLLECTION who's associated KEY is passed as parameter. ▪ Algorithm: 1. CALLS the BASE CLASS **MyBase.Dictionary.Item(key)** Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use **CType()** Function to convert to native data type of DICTIONARY collection to ***DVD*** Class type. 3. Returns a FALSE if POINTER returned by Dictionary.Item(key) Property is a ***Nothing***. 4. Else CALLS the **object.Print()** Method of the OBJECT in the COLLECTION whose POINTER was returned by **MyBase.Dictionary.Item(key)**. 5. Add Error-Handling code using **Try-Catch-Finally** to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Sub PrintAll()`	None	None	▪ **Implementation of PrintAll()** method. ▪ ***Method*** that PRINTS the content of ALL THE OBJECT in the COLLECTION. ▪ Algorithm: 1. Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection. 2. Use `CType()` Function to convert to native data type of `DictionaryEntry` object to ***DVD*** Class type. 3. CALLS the `object.Print()` Method of EACH OF THE OBJECT in the COLLECTION. 4. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Shadows Sub Clear()`	None	None	▪ **Implementation of Clear()** method. ▪ **Method** uses Shad keyword to Shadow the Base Class Method which already implements this functionality. We are simply repeating the process here. ▪ ***Method*** that CLEARS or DELETES ALL OBJECTS in the COLLECTION. ▪ Algorithm: 1. CALLS `MyBase.Dictionary.Clear()` to do the work. 2. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
Public Overrides Function DeferredDelete(ByVal strKey As String) As Boolean	String key	Boolean	▪ **Implementation of DeferredDelete(KEY)** method. ▪ ***Method*** that MARKS AND OBJECT FOR DELETION inside the COLLECTION. ▪ Algorithm: 1. Uses For Each objDictionaryEntry In MyBase.Dictionary LOOP to iterate through the collection. 2. Use CType() Function to convert to native data type of DictionaryEntry POINTER to ***DVD*** Class POINTER. 3. Interrogates EACH OBJECT BY COMPARING SEARCH KEY with ID of OBJECT in COLLECTION. 4. If KEY is FOUND, VERIFIES OBJECT IS NOT NEW by getting its IsNew Property. 5. If OBJECT IS NOT NEW or OLD, then CALLS the DeferredDelete() method of the OBJECT IN COLLECTION to MARK IT FOR DEFERRED DELETION. 6. EXITS the LOOP using Return True since OBJECT FOUND & MARKED FOR DELETION & SEARCH IS OVER. 7. Else if Object is NEW then Return False since NEW object should NOT BE DELETED but INSERTED 8. If SEARCH AFTER END OF For Each is over and KEY WAS NOT FOUND Return False. 9. Add Error-Handling code using Try-Catch-Finally to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Data Access Methods	Parameters	Return Type	Description
`Public Shared Function Create()As DVDList`	`None`	`DVDList Reference or Pointer`	▪ **Implementation of `Create()`** method. ▪ Public interface to the data access method that CREATES A NEW OBJECT USING FACTORY METHOD. ▪ The `Create()` method does not perform the data access but calls upon the `DataPortal_Create()` method to do the work. ▪ Algorithm: **1.** CALL & `Return` the `DataPortal_Create()` method.
`Public Sub Load()`	`String key`	`None`	▪ **Implementation of `Load(key)`** method. ▪ Public interface to the data access method that retrieves ALL THE ***DVD*** RECORDS FROM the Employee Table and ADDS them to the **DICTIONARY COLLECTION OBJECT**. ▪ The `Load()` method does not perform the data access but calls upon the `DataPortal_Fetch()` method to do the work. ▪ Algorithm: **1.** CALL the `DataPortal_Fetch()` method.
`Public Sub Save()`	None	`None`	▪ **Implementation of Save()** method. ▪ The ***Save()*** Method is a very important method. ▪ The `Save()` method does not perform the work but calls upon the `DataPortal_Save()` method to do the work. ▪ **IT HAS THE INTELLIGENCE TO DETERMINE IF THE COLLECTION HAS BEEN CHANGED OR NOT BY TESTING THE IsDirty PROPERTY.** ▪ **If OBJECT IS DIRTY IT CALLS THE FOLLOWING METHOD:** ▪ Algorithm: **1.** If Collection **IsDirty** CALL the `DataPortal_Save()` method. **2.** Otherwise nothing is done.
`Public Sub ImmediateDelete(ByVal Key As String)`	`String key`	`None`	▪ **Implementation of `DeleteObject(key)`** method. ▪ Public interface to the data access method that SEARCHES AN OBJECT IN THE **DICTIONARY COLLECTION OBJECT** & IMMEDIATELY DELETES the RECORD of the PRIMARY KEY ***KEY/IDNumber*** passed as a parameter FROM DATABASE. ▪ The `DeleteObject()` method does not perform the work but calls upon the `DataPortal_DeleteObject()` method to do the work. ▪ Algorithm: **2.** CALL the `DataPortal_DeleteObject(Key)` method.

Protected Data Access Methods	Parameters	Return Type	Description
`Protected Shared Function DataPortal_Create() As DVDList`	`None`	`DVDList Reference or Pointer`	▪ **Implementation of** `Shared Function DataPortal_Create()` **method**. ▪ OBJECT FACTORY METHOD. ▪ Creates & RETURNS FULLY READY AND INITIALIZED ***DVDList*** OBJECTS. ▪ STUB FUNCTION METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY & RETURN KEYWORD.
`Protected Sub DataPortal_Fetch(ByVal Key As String)`	`String key`	`None`	▪ **Implementation of** `DataPortal_Fetch(Key)` method. ▪ **THE FOLLOWING CODE IS HERE TEMPORARILY TO SUPPORT LOADING FROM A FILE.** ▪ **NEXT PROJECT WE WILL LOAD FROM A REAL DATABASE AND THIS CODE WILL BE REMOVED.** ▪ LOAD the OBJECTS from the `DVDData.txt` file and ADDS them to the COLLECTION. ▪ Algorithm: 1. USE `File.Exists("DVDData.txt")` to verify if FILE EXISTS. 2. If FILE DOES NOT EXISTS IT CREATES IT using `File.Create("DVDData.txt")` Method. 3. Open File for READING. 4. Read a LINE from file & PARSE each comma-delimited line. 5. Create new temporary OBJECT if the DVD Class. 6. SET OBJECT WTH values from LINE READ FROM FILE. 7. ADD OBJECT TO COLLECTION. 8. Repeat this process until EOF. 9. Add Error-Handling code using `Try-Catch-Finally` to handle a ***GENERAL*** Exception.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Sub DataPortal_Save()	None	None	▪ **Implementation of** DataPortal_Save() method. ▪ **YOU WILL IMPLEMENT THIS METHOD AS DICTATED BY THE BUSINESS COLLECTION CLASS TEMPLATE.** ▪ **NEVERTHELESS, the following CODE WILL NOT HELP US IN THIS PROJECT SINCE THE CURRENT BUSINESS CLASSES DON'T PERFORM REAL DATA ACCESS.** ▪ **THEREFORE THE FOLLOWING CODE IS FOR FUTURE USE, BUT IT MUST BE IMPLEMENTED AS SHOWN IN THE BUSINESS COLLECTION CLASS TEMPLATE:** ▪ WHAT ID DOES IS AS FOLLOWS: o LOOPS through the **DICTIONARY COLLECTION OBJECT** & CALLS EACH OBJECTS ***Save()*** to SAVE THE OBJECT. o The ides is the **EACH OBJECT IN COLLECTION SAVES ITSELF AND DECIDES WHETHER TO DO AN UPDATE OR INSERT BASED ON THE STATUS OF ITS ISDIRTY**, **ISNEW** AND **ISDELETED** FLAGS. ▪ **IMPORTANT! THIS CODE IS PROVIDED TO YOU IN THE *Business Collection Class* TEMPLATE. Simply COPY/PASTE modify for this DVDLIST CLASS.** ▪ **KEEP IN MIND THIS CODE HAS NO IMPACT TO THE PROGRAM, IT WILL EXECUTE AND CALL THE SAVE METHOD OF EACH OBJECT BUT THE DATA ACCESS METHODS HAVE NOT BEEN IMPLEMENTED YET. BUT IT MUST BE HERE FOR FUTURE USE.** ▪ *Algorithm*: **1.** Uses **For Each objDictionaryEntry In MyBase.Dictionary** LOOP to iterate through the COLLECTION. **2.** Use **CType()** Function to convert to native data type of DictionaryEntry object to ***DVD*** Class type. **3.** CALLS EACH OBJECT'S ***Save()*** Method to PERFORM EITHER AN UPDATE OR INSERT TO DATABASE. ▪ Add Error-Handling code using **Try-Catch-Finally** to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions ▪ Use **throw** statement to re-throw all exceptions

			▪ **THE FOLLOWING CODE NEEDS TO BE ADDED AT THE END OF THE CODE IN ORDER TO SUPPORT SAVING TO FILE.** ▪ **IN THIS VERSION OF THE PROJECT WE ARE STILL USING FILES TO SAVE DATA SO WE NEED TO TEMPORARILY PLACE THIS CODE. IN THE FUTURE, WHEN WE ARE WORKING WITH A REAL DATABASE, THIS CODE WILL BE REMOVED AND ONLY THE ABOVE CODE WILL BE NEEDED.** ▪ **YOU SHOULD ALREADY HAVE THIS CODE FROM THE PREVIOUS VERSION OF THE PROJECT, JUST COPY/PASTE**. ▪ File Save algorithm: 1. SAVES the OBJECTS IN THE COLLECTION to the **`DVDData.txt`** file. 2. Algorithm: 3. Open File for WRITTING. 4. Uses **`For Each objDictionaryEntry In MyBase.Dictionary`** LOOP to iterate through the collection. 5. Use **`CType()`** Function to convert to native data type of **`DictionaryEntry`** object to ***DVD*** Class type 6. GETS ALL THE PROPERTIES FOR EACH OBJECT IN ARRAY and CREATES A ***Comma-delimited string*** from ALL THE PROPERTIES OF THE OBJECT IN ARRAY. 7. CALLS THE STREAMREADER OBJECT `WriteLine()` Method TO WRITE THE ***Comma-delimited string*** LINE TO FILE. 8. Repeat this process until ALL OBJECTS IN ARRAY HAVE BEEN VISITED AND ITS PROPERTIES WRITTEN TO THE FILE AS A ***Comma-delimited string***. 9. Add Error-Handling code using **`Try-Catch-Finally`** to handle a ***GENERAL*** Exception.

Protected Data Access Methods	Parameters	Return Type	Description
`Protected Sub DataPortal_DeleteObject(By Val Key As String)`	None	`None`	▪ **Implementation of `DataPortal_DeleteObject()`** method. ▪ LOOPS through the **DICTIONARY COLLECTION OBJECT** SEARCHING for the OBJECT WHOSE ***KEY/IDNumber*** is passed as parameter. WHEN FOUND, calls the OBJECTS ***`Delete(Key)`*** method to PERMANENTLY DELETE THE OBJECT FROM DATABASE. ▪ **IMPORTANT! THIS CODE IS PROVIDED TO YOU IN THE *Business Collection Class* TEMPLATE. Simply modify as desire**. ▪ Algorithm: **1.** Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the COLLECTION. **2.** Use `CType()` Function to convert to native data type of `DictionaryEntry` object to ***DVD*** Class type. **3.** QUESTION each OBJECT if they are the ***KEY/IDNumber*** being SEARCHED. **4.** WHEN FOUND, CALLS the OBJECT'S ***`Delete()`*** Method to DELETE THE OBJECT FROM DATABASE. ▪ Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions ▪ Use `throw` statement to re-throw all exceptions

Public Helper Methods	Parameters	Return Type	Description
`Public Function ToArray() As DVD()`	`None`	**`POINTER TO Employee ARRAY OBJECT Reference or Pointer to ARRAY`**	▪ **Implementation of `ToArray()`** method. ▪ Support FOR DATA BINDING. ▪ METHOD THAT CONVERTS **DICTIONARY COLLECTION OBJECT** TO **ARRAY**. ▪ Returns the ARRAY POINTER ***DVD ()***. ▪ Algorithm: **1.** Creates TEMP **ARRAY** of ***DVD*** **2.** USES **DICTIONARY CLASS** `CopyTo()` Method TO COVERT **DICTIONARY COLLECTION OBJECT** TO TEMP **ARRAY**. **3.** `Return` The POPULATED TEMP **ARRAY**.

Class VideoGameList

- ***VideoGameList*** COLLECTION Class. Object of this class will store and manage **VideoGame Objects** in memory and load/save them from ***TEXT FILE***.
- This class encapsulates a ***DICTIONARY COLLECTION*** OBJECT provided via ***INHERITANCE*** from the **BUSINESSCOLLECTIONBASE** Class which ***INHERITS*** from ***DICTIONARYBASE CLASS***.
- The key properties/methods are described as follows:

General Information	Description

`Option Explicit On`
`Option Strict On`
 	
`'Impoted Libraries`
`Imports System.IO`
`Imports System.Data`
`Imports System.Data.OleDb`
`Imports System.Configuration`
	▪ Option Explicit and Option Strict should be On

General Class Information	Description

`<Serializable()> _`
`Public Class VideoGameList`
`Inherits BusinessCollectionBase`
	▪ CUSTOM COLLECTION CLASS that encapsulates a DICTIONARY COOLECTION OBJECT that stores **VideoGame** objects.
	▪ ADD the KEYWORD **`<Serializable()> _`** to enable Serialization for this class.
	▪ Class **`Inherits`** from ***BusinessCollectionBase*** Class.

Public Properties (GET/SET)	Description
`Public Shadows ReadOnly Property Count() As Integer`	READ-ONLY GET: returns NUMBER OF ELEMENTS OR OBJECT IN THE COLLECTION DICTIONARY. Property should shadow the base class equivalent. ▪ GET Algorithm: **1.** Calls BASE CLASS **`MyBase.Dictionary.Count`** Property.
`Public Property Item(ByVal key As String) As VideoGame`	▪ **Wrapper PROPERTY** that **GET & SET *VideoGame* OBJECTS in the DICTIONARY COLLECTION** ▪ This Property GETS the POINTER to the OBJECT in the COLLECTION based on its KEY. ▪ This Property SETS the OBJECT WHO'S KEY is passed as parameter with the OBJECT being assigned. ▪ GET Algorithm: **1.** CALLS the BASE CLASS **`MyBase.Dictionary.Item(key)`** Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. **2.** Use **`CType()`** Function to convert to native data type of DICTIONARY collection to ***VideoGame*** Class type. ▪ SET Algorithm: **1.** CALLS the BASE CLASS **`MyBase.Dictionary.Contains`**`(key)` Method determine if KEY exists. **2.** If exists it calls **`MyBase.Dictionary.Item(key)`** to do the work of SETTING the VALUE. **3.** Else, THROWS `Throw New System.ArgumentException("ID Not found")` EXCEPTION indicating KEY WAS NOT FOUND.

Public Methods	Parameters	Return Type	Description
`Public Sub Add(ByVal key As String, ByVal objVideoGame As VideoGame)`	`VideoGame objVideoGame`	`None`	▪ **Implementation of Add(object)** method. ▪ ***Wrapper Method*** that ADDS the object & its associated KEY passed as argument into the collection. ▪ In this case, the KEY is the ***IDNumber*** PROPERTY of the Object. ▪ It is assumed the Object if CREATED & POPULATED in the User-Interface or calling program when passed as argument to the method call. Method simply adds the object to the COLLECTION. ▪ Algorithm: 1. Calls `MyBase.Dictionary.Add(key, objVideoGame)` Method to add the object to Collection. 2. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Sub Add(ByVal x, ByVal y,ByVal z, etc.)`	Variable for each required Parameter to SET PROPERTY of the ***VideoGame*** Class Object. Normally the paramters of the Parameterized Constructor.	`None`	▪ **Implementation of Add(x,y,z etc.)** method. ▪ ***Wrapper Method*** that ADDS object into the collection. ▪ Same functionality as previous ADD, except NO populated OBJECT is passed as parameter, but the individual values that make up the OBJECT. ▪ The method itself creates the Object, populates it with the values from parameters and then ADDS the object to the COLLECTION with the KEY. ▪ In this case, the KEY is the ***IDNumber*** PARAMETER value of the method's parameter list. ▪ This version of ADD, requires less programming in the User-Interface. ▪ Algorithm: 1. Creates either DEFAULT Temporary ***VideoGame*** OBJECT & SETS the appropriate properties based on parameters VALUES passed to method. Or CREATES PARAMETERIZED Constructor OBJECT, passing to the OBJECT the VALUES of the parameters passed to method. 2. Calls `MyBase.Dictionary.Add(key, objVideoGame)` Method to add the object to Collection. The KEY being the ***IDNumber*** PROPERTY of the object created. 3. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Function Edit(ByVal key As String, ByVal objVideoGame As VideoGame) As Boolean`	`VideoGame objVideoGame`	`True` `False`	▪ **Implementation of Edit(object)** method. ▪ ***Wrapper Method*** that EDITS the OBJECT in the COLLECTION whose KEY is passed as parameter to method. ▪ In this case, the KEY is the ***IDNumber*** PROPERTY of the Object ▪ The OBJECT in COLLECTION is edited by SETTING its PROPERTIES, with the values or the PROPERTIES of the OBJECT passed as parameter. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Item(key)` Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use `CType()` Function to convert to native data type of DICTIONARY collection to ***VideoGame*** Class type. 3. Returns a FALSE if POINTER returned by `MyBase.Dictionary.Item(key)` Property is a ***Nothing***. 4. Else, **GETS PROPERTIES** of Object passed as parameter & **SETS PROPERTIES** of OBJECT in the COLLECTION whose POINTER was returned by `Dictionary.Item(key)` Property & **Returns** a **True**. 5. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Function Edit(ByVal x, ByVal y,ByVal z, etc.) As Boolean`	Variable for each required Parameter to SET PROPERTY of the ***VideoGame*** Class Object. To be EDITED.	`True` `False`	▪ **Implementation of Edit(x,y,z)** method. ▪ ***Wrapper Method*** that EDITS the OBJECT in the COLLECTION who's associated KEY is passed as ONE of the parameter variables. The OBJECT in COLLECTION is edited by SETTING its PROPERTIES with the values passed as parameters to method, except the KEY parameter. ▪ In this case, the KEY is the ***IDNumber*** PARAMETER value of the method's parameter list. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Item(key)` Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use `CType()` Function to convert to native data type of DICTIONARY collection to ***VideoGame*** Class type. 3. Returns a FALSE if POINTER returned by `MyBase.Dictionary.Item(key)` Property is a ***Nothing***. 4. Else **SETS PROPERTIES** of OBJECT in the COLLECTION whose POINTER was returned by `MyBase.Dictionary.Item(key)` with values passed as parameters, except the value that represents the KEY & **Returns** a **True**. 5. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Function Remove(ByVal key As String) As Boolean`	`String Key`	`True` `False`	▪ **Implementation of Remove(Key)** method. ▪ ***Wrapper Method*** that REMOVES the object from COLLECTION who's associated KEY is passed as parameter to method. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Contains(key)` Method determine if KEY exists. 2. If exists it calls `MyBase.Dictionary.Remove(key)` to do the work of REMOVING OBJECT from COLLECTION and returns a **TRUE**. 3. Else, if not exists, then it returns a **FALSE**. 4. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Function Print(ByVal key As String) As Boolean`	`String Key`	`True` `False`	▪ **Implementation of Print(Key)** method. ▪ ***Method*** that PRINTS the content of the OBJECT in the COLLECTION who's associated KEY is passed as parameter. ▪ Algorithm: 1. CALLS the BASE CLASS `MyBase.Dictionary.Item(key)` Property to return the POINTER to the OBJECT in Collection who's KEY is passed as argument. 2. Use `CType()` Function to convert to native data type of DICTIONARY collection to ***VideoGame*** Class type. 3. Returns a FALSE if POINTER returned by `Dictionary.Item(key)` Property is a ***Nothing***. 4. Else CALLS the `object.Print()` Method of the OBJECT in the COLLECTION whose POINTER was returned by `MyBase.Dictionary.Item(key)`. 5. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Sub PrintAll()`	None	`None`	▪ **Implementation of PrintAll()** method. ▪ ***Method*** that PRINTS the content of ALL THE OBJECT in the COLLECTION. ▪ Algorithm: 1. Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection. 2. Use `CType()` Function to convert to native data type of `DictionaryEntry` object to ***VideoGame*** Class type. 3. CALLS the `object.Print()` Method of EACH OF THE OBJECT in the COLLECTION. 4. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions
`Public Shadows Sub Clear()`	None	`None`	▪ **Implementation of Clear()** method. ▪ **Method** uses Shad keyword to Shadow the Base Class Method which already implements this functionality. We are simply repeating the process here. ▪ ***Method*** that CLEARS or DELETES ALL OBJECTS in the COLLECTION. ▪ Algorithm: 1. CALLS `MyBase.Dictionary.Clear()` to do the work. 2. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Methods	Parameters	Return Type	Description
`Public Overrides Function DeferredDelete(ByVal strKey As String) As Boolean`	`String key`	`Boolean`	▪ **Implementation of DeferredDelete(KEY)** method. ▪ ***Method*** that MARKS AND OBJECT FOR DELETION inside the COLLECTION. ▪ Algorithm: 1. Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the collection. 2. Use `CType()` Function to convert to native data type of `DictionaryEntry` POINTER to ***VideoGame*** Class POINTER. 3. Interrogates EACH OBJECT BY COMPARING SEARCH KEY with ID of OBJECT in COLLECTION. 4. If KEY is FOUND, VERIFIES OBJECT IS NOT NEW by getting its `IsNew` Property. 5. If OBJECT IS NOT NEW or OLD, then CALLS the `DeferredDelete()` method of the OBJECT IN COLLECTION to MARK IT FOR DEFERRED DELETION. 6. EXITS the LOOP using `Return True` since OBJECT FOUND & MARKED FOR DELETION & SEARCH IS OVER. 7. Else if Object is NEW then `Return False` since NEW object should NOT BE DELETED but INSERTED 8. If SEARCH AFTER END OF `For Each` is over and KEY WAS NOT FOUND `Return False`. 9. Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions

Public Data Access Methods	Parameters	Return Type	Description
Public Shared Function Create()As VideoGameList	None	VideoGameList Reference or Pointer	▪ **Implementation of Create()** method. ▪ Public interface to the data access method that CREATES A NEW OBJECT USING FACTORY METHOD. ▪ The Create() method does not perform the data access but calls upon the DataPortal_Create() method to do the work. ▪ Algorithm: **1.** CALL & Return the DataPortal_Create() method.
Public Sub Load()	String key	None	▪ **Implementation of Load(key)** method. ▪ Public interface to the data access method that retrieves ALL THE ***VideoGame*** RECORDS FROM the Employee Table and ADDS them to the **DICTIONARY COLLECTION OBJECT**. ▪ The Load() method does not perform the data access but calls upon the DataPortal_Fetch() method to do the work. ▪ Algorithm: **1.** CALL the DataPortal_Fetch() method.
Public Sub Save()	None	None	▪ **Implementation of Save()** method. ▪ The ***Save()*** Method is a very important method. ▪ The Save() method does not perform the work but calls upon the DataPortal_Save() method to do the work. ▪ **IT HAS THE INTELLIGENCE TO DETERMINE IF THE COLLECTION HAS BEEN CHANGED OR NOT BY TESTING THE IsDirty PROPERTY.** ▪ **If OBJECT IS DIRTY IT CALLS THE FOLLOWING METHOD:** ▪ Algorithm: **1.** If Collection **IsDirty** CALL the DataPortal_Save() method. **2.** Otherwise nothing is done.
Public Sub ImmediateDelete(ByVal Key As String)	String key	None	▪ **Implementation of DeleteObject(key)** method. ▪ Public interface to the data access method that SEARCHES AN OBJECT IN THE **DICTIONARY COLLECTION OBJECT** & IMMEDIATELY DELETES the RECORD of the PRIMARY KEY ***KEY/IDNumber*** passed as a parameter FROM DATABASE. ▪ The DeleteObject() method does not perform the work but calls upon the DataPortal_DeleteObject() method to do the work. ▪ Algorithm: **1.** CALL the DataPortal_DeleteObject(Key) method.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Shared Function DataPortal_Create() As VideoGameList	None	VideoGameList Reference or Pointer	▪ **Implementation of Shared Function DataPortal_Create() method**. ▪ OBJECT FACTORY METHOD. ▪ Creates & RETURNS FULLY READY AND INIALIZED ***VideoGameList*** OBJECTS. ▪ STUB FUNCTION METHOD FOR FUTURE UPGRADE. ▪ CREATE THE HEADER WITH AN EMPTY BODY & RETURN KEYWORD.
Protected Sub DataPortal_Fetch(ByVal Key As String)	String key	None	▪ **Implementation of** DataPortal_Fetch(Key) method. ▪ **THE FOLLOWING CODE IS HERE TEMPORARILY TO SUPPORT LOADING FROM A FILE.** ▪ **NEXT PROJECT WE WILL LOAD FROM A REAL DATABASE AND THIS CODE WILL BE REMOVED.** ▪ LOAD the OBJECTS from the VideoGameData.txt file and ADDS them to the COLLECTION. ▪ Algorithm: 1. USE File.Exists("VideoGameData.txt") to verify if FILE EXISTS. 2. If FILE DOES NOT EXISTS IT CREATES IT using File.Create("VideoGameData.txt") Method. 3. Open File for READING. 4. Read a LINE from file & PARSE each comma-delimited line. 5. Create new temporary OBJECT if the ***VideoGame*** Class. 6. SET OBJECT WTH values from LINE READ FROM FILE. 7. ADD OBJECT TO COLLECTION. 8. Repeat this process until EOF. 9. Add Error-Handling code using Try-Catch-Finally to handle a ***GENERAL*** Exception.

Protected Data Access Methods	Parameters	Return Type	Description
`Protected Sub DataPortal_Save()`	None	`None`	▪ **Implementation of `DataPortal_Save()`** method. ▪ **YOU WILL IMPLEMENT THIS METHOD AS DICTATED BY THE BUSINESS COLLECTION CLASS TEMPLATE.** ▪ **NEVERTHELESS, the following CODE WILL NOT HELP US IN THIS PROJECT SINCE THE CURRENT BUSINESS CLASSES DON'T PERFORM REAL DATA ACCESS.** ▪ **THEREFORE THE FOLLOWING CODE IS FOR FUTURE USE, BUT IT MUST BE IMPLEMENTED AS SHOWN IN THE BUSINESS COLLECTION CLASS TEMPLATE:** ▪ WHAT ID DOES IS AS FOLLOWS: o LOOPS through the **DICTIONARY COLLECTION OBJECT** & CALLS EACH OBJECTS ***`Save()`*** to SAVE THE OBJECT. o The ides is the **EACH OBJECT IN COLLECTION SAVES ITSELF AND DECIDES WHETHER TO DO AN UPDATE OR INSERT BASED ON THE STATUS OF ITS ISDIRTY**, **ISNEW** AND **ISDELETED** FLAGS. ▪ **IMPORTANT! THIS CODE IS PROVIDED TO YOU IN THE *Business Collection Class*** TEMPLATE. **Simply COPY/PASTE modify for this VideoGameLIST CLASS.** ▪ **KEEP IN MIND THIS CODE HAS NO IMPACT TO THE PROGRAM, IT WILL EXECUTE AND CALL THE SAVE METHOD OF EACH OBJECT BUT THE DATA ACCESS METHODS HAVE NOT BEEN IMPLEMENTED YET. BUT IT MUST BE HERE FOR FUTURE USE.** ▪ *Algorithm*: **1.** Uses `For Each objDictionaryEntry In MyBase.Dictionary` LOOP to iterate through the COLLECTION. **2.** Use `CType()` Function to convert to native data type of `DictionaryEntry` object to ***VideoGame*** Class type. **3.** CALLS EACH OBJECT'S ***`Save()`*** Method to PERFORM EITHER AN UPDATE OR INSERT TO DATABASE. ▪ Add Error-Handling code using `Try-Catch-Finally` to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions ▪ Use `throw` statement to re-throw all exceptions

			▪ **THE FOLLOWING CODE NEEDS TO BE ADDED AT THE END OF THE CODE IN ORDER TO SUPPORT SAVING TO FILE.** ▪ **IN THIS VERSION OF THE PROJECT WE ARE STILL USING FILES TO SAVE DATA SO WE NEED TO TEMPORARILY PLACE THIS CODE. IN THE FUTURE, WHEN WE ARE WORKING WITH A REAL DATABASE, THIS CODE WILL BE REMOVED AND ONLY THE ABOVE CODE WILL BE NEEDED.** ▪ **YOU SHOULD ALREADY HAVE THIS CODE FROM THE PREVIOUS VERSION OF THE PROJECT, JUST COPY/PASTE**. ▪ File Save algorithm: 1. SAVES the OBJECTS IN THE COLLECTION to the **`VideoGameData.txt`** file. 2. Algorithm: 3. Open File for WRITTING. 4. Uses **`For Each objDictionaryEntry In MyBase.Dictionary`** LOOP to iterate through the collection. 5. Use **`CType()`** Function to convert to native data type of **`DictionaryEntry`** object to ***VideoGame*** Class type 6. GETS ALL THE PROPERTIES FOR EACH OBJECT IN ARRAY and CREATES A ***Comma-delimited string*** from ALL THE PROPERTIES OF THE OBJECT IN ARRAY. 7. CALLS THE STREAMREADER OBJECT `WriteLine()` Method TO WRITE THE ***Comma-delimited string*** LINE TO FILE. 8. Repeat this process until ALL OBJECTS IN ARRAY HAVE BEEN VISITED AND ITS PROPERTIES WRITTEN TO THE FILE AS A ***Comma-delimited string***. 9. Add Error-Handling code using **`Try-Catch-Finally`** to handle a ***GENERAL*** Exception.

Protected Data Access Methods	Parameters	Return Type	Description
Protected Sub DataPortal_DeleteObject(By Val Key As String)	None	None	▪ **Implementation of DataPortal_DeleteObject()** method. ▪ LOOPS through the **DICTIONARY COLLECTION OBJECT** SEARCHING for the OBJECT WHOSE ***KEY/IDNumber*** is passed as parameter. WHEN FOUND, calls the OBJECTS ***Delete(Key)*** method to PERMANENTLY DELETE THE OBJECT FROM DATABASE. ▪ **IMPORTANT! THIS CODE IS PROVIDED TO YOU IN THE *Business Collection Class* TEMPLATE. Simply modify as desire**. ▪ Algorithm: **1.** Uses For Each objDictionaryEntry In MyBase.Dictionary LOOP to iterate through the COLLECTION. **2.** Use CType() Function to convert to native data type of DictionaryEntry object to ***VideoGame*** Class type. **3.** QUESTION each OBJECT if they are the ***KEY/IDNumber*** being SEARCHED. **4.** WHEN FOUND, CALLS the OBJECT'S ***Delete()*** Method to DELETE THE OBJECT FROM DATABASE. ▪ Add Error-Handling code using Try-Catch-Finally to handle all required ***COLLECTION*** & ***GENERAL*** Exceptions ▪ Use throw statement to re-throw all exceptions

Public Helper Methods	Parameters	Return Type	Description
Public Function ToArray() As VideoGame()	None	POINTER TO Employee ARRAY OBJECT Reference or Pointer to ARRAY	▪ **Implementation of ToArray()** method. ▪ Support FOR DATA BINDING. ▪ METHOD THAT CONVERTS **DICTIONARY COLLECTION OBJECT** TO **ARRAY**. ▪ Returns the ARRAY POINTER ***VideoGame ()***. ▪ Algorithm: **1.** Creates TEMP **ARRAY** of ***VideoGame*** **2.** USES **DICTIONARY CLASS** CopyTo() Method TO COVERT **DICTIONARY COLLECTION OBJECT** TO TEMP **ARRAY**. **3.** Return The POPULATED TEMP **ARRAY**.

APPENDIX B– User-Interface & FORM DETAILS

User-Interface:

General User-Interface Requirements

- ❑ SAME AS PREVIOUS VERSION 2, BUT UPDATE TO THE REMOVE/DELETE CLICK EVENT-HANDLER FOR ALL MANAGEMENT FORMS (See below)
- ❑ It is made up of the following:
  - ▪ MODULES
  - ▪ FORMS

Module Requirements

- ❑ Module requirements same as Version 2 of the project.

Form Design & Navigation:

PROJECT VERSION 3.0 - User-Interface Requirements

Forms Requirements

- ❑ KEEP THE SAME FORM REQUIREMENTS AS IN VERSION 2.0:
  - ▪ Login Form:
    - *__Login Screen__*
  - ▪ Front-End Main & Application Selection Forms:
    - *__Main Welcome Form__*
    - *__Video POS System Form__*
    - *__Back-End Management Form__*
  - ▪ Back-End Management Forms:
    - *__Employee Management Form__*
    - *__Customer Management Form__*
    - *__DVD Management Form__*
    - *__Video Game Management Form__*
- ❑ You can ADD ANY ADDITIONAL FORM YOU MAY WANT OR NEED IN ORDER TO IMPLEMENT YOUR DESIGN.

Employee Management Screen UPDATE:

❑ **KEEP ALL FUNCTIONALITY OF VERSION 2.0 BUT MAKE THE FOLLOWING MODIFICATIONS:**

- In the ***Employee Management screens*** UPDATE, the DELETE or REMOVE BUTTON CLICK EVENT HANDLER CODE TO perform a DEFERRED DELETE by calling **objEmployeeList.DeferredDelete()** Method IN ADDITION TO calling **objEmployeeList.Remove()** Method..

❖ **NOTE –** In next version of project the **objEmployeeList.Remove()** Method **WILL BE REMOVED. WE TEMPORARILY NEED** **objEmployeeList.Remove()** Method NOW to make the project work since we ARE NOT USING ALL THE CAPABILITIES OF BUSINESS OBJECT AT THIS TIME (NO data access code or ADO.NET code in the Business Classes & ALL DATA ACCESS BEING DONE FROM COLLECTION CLASSES) SO THE PROJECT WON'T WORK WITHOUT **objEmployeeList.Remove()** Method.

Functionality	Action Taken	Graphical Control/Event-Handler	Description /Comments
REMOVE: ▪ **MARKS OBJECCT FOR DELETION** in the COLLECTION, who's KEY or ***SSNumber*** is entered on the Form Textbox control. ▪ The **DEFERRED DELETE** is done by the ***Business Object Layer*** via ***objEmployeeList*** Collection Class Object	1. **Grabs the *SSNumber* from the Form's** *TextBox.* 2. Calls **objEmployeeList .DeferredDelete(Key)** method to do the work. 3. Calls **objEmployeeList .Remove()** method to do the work. 4. If value returned from call is **False** or NOT FOUND, displays a message indicating "***Employee*** not found"	▪ **Button Control** – to respond to user Click & ***Event-Handler*** to execute code. ▪ **TextBox Control** – to accept ***SSNumber*** of OBJECT to be deleted. ▪ **Labels**. – To label and describe all controls. ▪ **Others** – You feel are necessary for your design.	▪

Customer Management Screen UPDATE:

- **KEEP ALL FUNCTIONALITY OF VERSION 2.0 BUT MAKE THE FOLLOWING MODIFICATIONS:**

  - In the ***Customer Management Screen*** UPDATE, the DELETE or REMOVE BUTTON CLICK EVENT HANDLER CODE TO perform a DEFERRED DELETE by calling **objCustomerList.DeferredDelete()** Method IN ADDITION TO calling **objCustomerList.Remove()** Method..

- **NOTE –** In next version of project the **objCustomerList.Remove()** Method **WILL BE REMOVED. WE TEMPORARILY NEED** **objCustomerList.Remove()** Method NOW to make the project work since we ARE NOT USING ALL THE CAPABILITIES OF BUSINESS OBJECT AT THIS TIME (NO data access code or ADO.NET code in the Business Classes & ALL DATA ACCESS BEING DONE FROM COLLECTION CLASSES) SO THE PROJECT WON'T WORK WITHOUT **objCustomerList.Remove()** Method.

Functionality	Action Taken	Graphical Control/Event-Handler	Description /Comments
REMOVE: ▪ **MARKS OBJECCT FOR DELETION** the Object in the COLLECTION, who's KEY or ***IDNumber*** is entered on the Form Textbox control. ▪ The **DEFERRED DELETE** is done by the ***Business Object Layer*** via ***objCustomerList*** Collection Class Object	1. **Grabs the *IDNumber* from the Form's** *TextBox.* 2. Calls **objCustomerList .DeferredDelete()** method to do the work. 3. Calls **objCustomerList .Remove()** method to do the work. 4. If value returned from call is **False** or NOT FOUND, displays a message indicating "***Customer not found***"	▪ **Button Control** – to respond to user Click & ***Event-Handler*** to execute code. ▪ **TextBox Control** – to accept ***IDNumber*** of OBJECT to be deleted. ▪ **Labels**. – To label and describe all controls. ▪ **Others** – You feel are necessary for your design.	▪

DVD Management Screen UPDATE:

- **KEEP ALL FUNCTIONALITY OF VERSION 2.0 BUT MAKE THE FOLLOWING MODIFICATIONS:**

  - In the ***DVD Management screen*** UPDATE, the DELETE or REMOVE BUTTON CLICK EVENT HANDLER CODE TO perform a DEFERRED DELETE by calling **objDVDList.DeferredDelete()** Method IN ADDITION TO calling **objDVDList.Remove()** Method..

  - **NOTE –** In next version of project the **objDVDList.Remove()** Method **WILL BE REMOVED. WE TEMPORARILY NEED objDVDList.Remove()** Method NOW to make the project work since we ARE NOT USING ALL THE CAPABILITIES OF BUSINESS OBJECT AT THIS TIME (NO data access code or ADO.NET code in the Business Classes & ALL DATA ACCESS BEING DONE FROM COLLECTION CLASSES) SO THE PROJECT WON'T WORK WITHOUT **objDVDList.Remove()** Method.

  -

Functionality	Action Taken	Graphical Control/Event-Handler	Description /Comments
REMOVE: ▪ **MARKS OBJECCT FOR DELETION** the Object in the COLLECTION, who's KEY or ***IDNumber*** is entered on the Form Textbox control. ▪ The **DEFERRED DELETE** is done by the ***Business Object Layer*** via ***objDVDList*** Collection Class Object	1. **Grabs the *IDNumber* from the Form's** *TextBox*. 2. Calls **objDVDList .DeferredDelete()** method to do the work. 3. Calls **objDVDList .Remove()** method to do the work 4. If value returned from call is **False** or NOT FOUND, displays a message indicating "**DVD not found**"	▪ **Button Control** – to respond to user Click & ***Event-Handler*** to execute code. ▪ **TextBox Control** – to accept ***IDNumber*** of OBJECT to be deleted. ▪ **Labels**. – To label and describe all controls. ▪ **Others** – You feel are necessary for your design.	▪

VideoGame Management Screen UPDATE:

- ❑ **KEEP ALL FUNCTIONALITY OF VERSION 2.0 BUT MAKE THE FOLLOWING MODIFICATIONS:**

  - In the ***Video Game Management screen*** UPDATE, the DELETE or REMOVE BUTTON CLICK EVENT HANDLER CODE TO perform a DEFERRED DELETE by calling **objVideoGameList.DeferredDelete()** Method IN ADDITION TO calling **objVideoGameList.Remove()** Method..

  - ❖ **NOTE –** In next version of project the **objVideoGameList.Remove()** Method **WILL BE REMOVED. WE TEMPORARILY NEED objVideoGameList.Remove()** Method NOW to make the project work since we ARE NOT USING ALL THE CAPABILITIES OF BUSINESS OBJECT AT THIS TIME (NO data access code or ADO.NET code in the Business Classes & ALL DATA ACCESS BEING DONE FROM COLLECTION CLASSES) SO THE PROJECT WON'T WORK WITHOUT **objVideoGameList.Remove()** Method.

Functionality	Action Taken	Graphical Control/Event-Handler	Description /Comments
REMOVE: ▪ **MARKS OBJECCT FOR DELETION** the Object in the COLLECTION, who's KEY or ***IDNumber*** is entered on the Form Textbox control. ▪ The **DEFERRED DELETE** is done by the ***Business Object Layer*** via ***objVideoGameList*** Collection Class Object	1. **Grabs the *IDNumber* from the Form's** *TextBox*. 2. Calls **objVideoGameList .DeferredDelete()** method to do the work. 3. Calls **objVideoGameList .Remove()** method to do the work. 4. If value returned from call is **False** or NOT FOUND, displays a message indicating "**VideoGame not found**"	▪ **Button Control** – to respond to user Click & ***Event-Handler*** to execute code. ▪ **TextBox Control** – to accept ***IDNumber*** of OBJECT to be deleted. ▪ **Labels**. – To label and describe all controls. ▪ **Others** – You feel are necessary for your design.	▪