CS708 Lecture Notes

Visual Basic.NET Object-Oriented Programming

Implementing Business Objects
Data Access Code

Part (III of III)
(Lecture Notes 3C)

Professor: A. Rodriguez
# Chapter 7  Business Objects Data Access Code

## 7.1 Business Objects Review & Status

### 7.1.1 Business Objects Requirements Status

- Ok, let’s review and see where we are as far as our Business Object implementation is concerned:

<table>
<thead>
<tr>
<th>Business Object Requirements</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Business Object Represents Real-World Business Entities</strong> – Business Objects contain the necessary attributes &amp; methods to behave like their real-world counterparts.</td>
<td>▪ Done</td>
</tr>
<tr>
<td><strong>User Interface Support</strong> – The Business Objects should contain the following logic to support the User Interface (UI):</td>
<td>▪ DONE</td>
</tr>
<tr>
<td>- Business objects RAISE <em>NotSupportedException</em> when business rules are violated.</td>
<td>▪</td>
</tr>
<tr>
<td>- FORMS/UI can trap &amp; handle</td>
<td>▪ Done</td>
</tr>
<tr>
<td><strong>Scalable &amp; Reusable</strong> – Business Objects should evolve &amp; gain new data, properties &amp; methods to support more functionality</td>
<td>▪ DONE.</td>
</tr>
<tr>
<td>- Created Class Library or DLL COMPONENT to encapsulate our classes.</td>
<td>▪</td>
</tr>
<tr>
<td><strong>Business Rules, Validation or Enforcement &amp; Status Tracking</strong> – Business Objects should contain the following logic:</td>
<td>▪ PARTIALLY DONE.</td>
</tr>
<tr>
<td>- Business Objects should VALIDATE that the data being set by the user is valid, correct data type, length etc</td>
<td>▪</td>
</tr>
<tr>
<td>- Business Objects should keep track of its status</td>
<td>▪</td>
</tr>
<tr>
<td>- The Business Objects should keep track of the business rules that are broken.</td>
<td>▪</td>
</tr>
<tr>
<td>- Business Objects should protect itself from unauthorized or unwanted, harmful access</td>
<td>▪</td>
</tr>
<tr>
<td>- We implemented the NEW &amp; DIRTY object mechanism.</td>
<td>▪</td>
</tr>
<tr>
<td>- Implemented Field or Property-level validation mechanism for NO-BLANKS, MAX-LENGTH, EXACT-LENGTH, etc.</td>
<td>▪</td>
</tr>
<tr>
<td><strong>BO Manage their own data &amp; database access</strong> – Business Objects should contain logic to handle data access. Operations such as searching, inserting, updating, deleting the database should be done by the business objects</td>
<td>▪ OPEN REQUIREMENT</td>
</tr>
<tr>
<td><strong>Distributed Business Objects</strong> – Business Objects should be design base with the following network distribution scheme in mind:</td>
<td>▪ OPEN REQUIREMENT (WILL NOT BE DONE IN THIS COURSE)</td>
</tr>
<tr>
<td>- Business Objects should contain the technology to allow them to be distributed across processes, network and applications.</td>
<td>▪</td>
</tr>
<tr>
<td>- Implemented Serialization</td>
<td>▪</td>
</tr>
<tr>
<td>- Implemented Remoting</td>
<td>▪</td>
</tr>
</tbody>
</table>

- WE HAVE TWO REQUIREMENTS LEFT. WE WILL ONLY IMPLEMENT ONE IN THIS COURSE, THAT IS THE DATA ACCESS CODE
7.2 Business Objects – Data Access Requirements

7.2.1 Objectives

- The next requirement we must address is Data Access.
- Since Business Objects need to handle their own Data Access, we will now cover the methods required to do so.

Data Access Objectives:

- Since it is our BUSINESS OBJECTS THAT PERFORM THEIR OWN DATA ACCESS
- Our objectives is to implement the Data Access Code for the 3-tiered and 4-tiered application architectures:

   - IN THIS LECTURE, WE WILL IMPLEMENT THE FIRST DIAGRAM OR DATA ACCESS FOR THE 3-TIERED APPLICATION ARCHITECTURE

6.2.2 Implementation Overview

Data Access Methods Details:

- In previous lecture, we divided the data access methods into two sections, PUBLIC DATA ACCESS METHODS and PROTECTED OR PRIVATE DATA ACCESS METHODS:

  - **Public Data Access Methods** – These methods are Public and assessable to the User-Interface or clients. These methods DO NOT CONTAIN ADO.NET CODE. They call the PROTECTED DATA ACCESS METHODS TO DO THE WORK.

  - **Protected, Private Data Access methods** – These methods can only be accessed internally within the class and its inherited children. These methods will actually perform the data access and contain the ADO.NET CODE with SQL queries or Stored Procedure calls. These methods are called by the Public Data Access Methods.

- Diagram of our Data Access implementation goals:
7.2.3 Preparing Our Classes for DATA ACCESS (Summary)

- Our BUSINESS CLASSES need to have the proper mechanism to support the Data Access Code.
- We have already done this as follows:
  - Created Public & Protected Data Access Methods
  - We created a BASE CLASS (BusinessBase) that will force the Data Access code to our Business Classes
  - We created our BUSINESS CLASSES, with DIRTY & NEW logic mechanism to work hand-in-hand with the Data Access Methods
  - The DIRTY & NEW mechanism in our classes work with the Data Access Methods as follows:

  - CREATE:
    - MARKS OBJECT AS NEW, WHEN CREATING A NEW OBJECT WITH DEFAULT DATA FROM DB

  - SELECT:
    - MARKS OBJECT AS OLD, AFTER RETRIEVING RECORDS FROM DB

  - INSERT:
    - ONLY PERFORMED WHEN OBJECT IS DIRTY & NEW.
    - MARKS OBJECT AS OLD AFTER INSERT

  - UPDATE:
    - ONLY PERFORMED WHEN OBJECT IS DIRTY & OLD.
    - MARKS OBJECT AS OLD AFTER UPDATE

  - DELETE:
    - MARKS OBJECT AS NEW, AFTER DELETE SINCE OBJECT DOES NOT EXIST IN DB.

- Our Business Classes such as clsCustomer, clsEmployee, clsProduct etc., inherit from BUSINESSBASE class the proper support for DATA ACCESS methods
- Same is true for our Business Collection Classes, such as clsCustomerList, clsEmployeeList, clsProductList etc., which inherit from BUSINESSCOLLECTIONBASE class the support for DATA ACCESS methods.
- The BASE classes have the following format:

<table>
<thead>
<tr>
<th>Imports</th>
<th>Class MustInherit clsBusinessBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Serializable()&gt;</td>
<td>_</td>
</tr>
<tr>
<td>Private Business Rules data:</td>
<td>mflgIsDirty, mflgIsNew</td>
</tr>
<tr>
<td>Public MustOverride Data Access Methods:</td>
<td>Create(), Load(Key), DeleteObject(Key), Save()</td>
</tr>
<tr>
<td>Protected MustOverride Data Access Methods:</td>
<td>DataPortal_Create(), DataPortal_Fetch(Key), DataPortal_Update(), DataPortal_Insert(), DataPortal_DeleteObject(Key)</td>
</tr>
<tr>
<td>Public Helper Data Access Methods:</td>
<td>DBConnectionString(DBName)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Imports</th>
<th>Class MustInherit clsBusinessCollectionBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherits DictionaryBase</td>
<td></td>
</tr>
<tr>
<td>Public Business Rule Properties:</td>
<td>IsDirty</td>
</tr>
<tr>
<td>Public MustOverride Data Access Methods:</td>
<td>Create(), Load(), DeleteObject(Key), Save()</td>
</tr>
<tr>
<td>Protected MustOverride Methods:</td>
<td>DataPortal_Create(), DataPortal_Fetch(), DataPortal_Save(), DataPortal_DeleteObject(Key)</td>
</tr>
<tr>
<td>Public Helper Data Access Methods:</td>
<td>DBConnectionString(DBName)</td>
</tr>
</tbody>
</table>
The FORMAT of the Business Classes we create should be based on the following BUSINESS TEMPLATES:

```
Imports <Serializable>() _
Class clsBusinessClass
Inherits clsBusinessBase

Private data:
Public Event Declarations:
Public Properties:
Public Constructors:
Public Methods:
Public Shared Data Access Methods:
  Create()
  Load(Key)
  DeleteObject(Key)
  Save()
Protected Override Data Access Methods:
  DataPortal_Create()
  DataPortal_Fetch(Key)
  DataPortal_Update()
  DataPortal_Insert()
  DataPortal_DeleteObject(Key)

Public Helper Methods:
```

```
Imports <Serializable>() _
Class clsBusinessCollectionClass
Inherits clsBusinessCollectionBase

Public Properties:
Public Wrapper Methods:
Public Regular Methods:
Public Shared Data Access Methods:
  Create()
  Load()
  DeleteObject(Key)
  Save()
Protected Override Data Access Methods:
  DataPortal_Create()
  DataPortal_Fetch()
  DataPortal_Save()
  DataPortal_DeleteObject(Key)

Public Helper Methods:
```
7.3 Sample Program #1 – Customer Retail Management Program Data Access, Using Microsoft Access Database

7.3.1 Overview

- No we upgrade the Customer Management Application from previous lecture (Lecture 3B, Sample Program #5), by implementing the DATA ACCESS METHODS using ADO.NET.
- In the previous lecture 3B, we upgraded this application by applying the Business Rules & Logic, Validation, and data access as follows:
  1. Inherited our Business Class clsCustomer from BusinessBase
  2. Inherited our Business Collection Class clsCustomerList from BusinessCollectionBase
  3. Modify the Business Class clsCustomer by following the format provided by the template class BusinessClass
  4. Modify the Business Collection Class clsCustomerList by following the format provided by the template class BusinessCollectionClass

- Using ADO.NET, we will now implement the DATA ACCESS CODE for the four PROTECTED Data Access Methods:
  - Protected Overrides DataPortal_Create()
  - Protected Overrides DataPortal_Fetch()
  - Protected Overrides DataPortal_Update()
  - Protected Overrides DataPortal_Insert()
  - Protected Overrides DataPortal_DeleteObject()

7.3.2 Object Model Requirements

- We will maintain all the Business Objects requirements from Lecture 3B Sample Program #5, these were as follows.

  1. Inherited Business classes from BusinessBase & modified the Business Classes by applying the format of the BusinessClassTemplate. Object model looks as follows:
2. Inherited the Business Collection Classes from BusinessCollectionBase, and modified the Collection Classes based on the template provided by BusinessCollectionClass templates. Object model looks as follows:

![Object Model Diagram]

### 7.3.3 Business Rules, Logic & Validation Requirements

- We will maintain all the BUSINESS LOGIC AND VALIDATION RULES from Lecture 3B Sample Program #5, with the EXCEPTION OF THE FILE ACCESS CODE, this we will remove
- The requirements are as follows:

1. Maintain the **Dirty Objects** to ALL OUR PROPERTY SET:
   - Customer Name: Call MARK-DIRTY()
   - Social Security & Customer ID Number – Call MARK-DIRTY()
   - Address, & Phone – Call MARK-DIRTY().

2. Maintain the same enforced **Field-Level Validation** to our Properties:
   - Customer Name – NO-BLANK & MAX-LENGTH.
   - Social Security & Customer ID Number – WRITE-ONCE, EXACT LENGTH & NO-BLANK/EMPTY
   - Address, & Phone – NO-BLANK/EMPTY.

3. **REMOVE THE FILE ACCESS CODE** from the COLLECTION CLASSES (clsCustomerList), which are located in the methods: DataPortal_Fetch() & DataPortal_Save(), we will be implementing these using ADO.NET.

4. Place Database Connection String as a private data member to each Business Object. More on this below.

### 7.3.4 Database Requirements for ACCESS DATABASE

- Before we can begin to add the ADO.NET code and test our application, we need to create our database tables with their relationships.
- We will begin this application by first creating the DATA ACCESS LAYER or Database portion first

#### Database & Client Architecture

- In this case we will use Microsoft Access Database; therefore our application will continue to be a **Single-Tier Client/Server Application**. Name the database smallbusinessapp.

#### Database Schema, Queries & Database Location

- The database needs to reside in the BIN\DEBUG FOLDER of the Client program, which is the same location as the CLIENT EXECUTABLE.
- This is done so the ADO.NET Connection String will not require the FULL PATH; it will automatically search for the database file in the location of CLIENT EXECUTABLE.
- Create the database, tables and their relationships, as well as populate the table with data and testing our queries in MS ACCESS.
- Once all our queries are tested we will use them in our ADO.NET code in the application.
Database Connection String Location

In this example, we will NOT USE THE BEST PRACTICE MECHANISM PROVIDED BY THE BUSINESS BASE CLASSES (BusinessBase & BusinessCollectionBase). The base classes provide us with a Helper Data Access Method named DBConnectionString(Key) which allows us to retrieve the connection string from file. In this example we will CREATE THE CONNECTION STRING IN-LINE WITH OUR CODE INSIDE THE OBJECTS, as shown in diagram below:

7.3.4 Form Requirements

The FORM requirements are similar to the previous version of this application. Below is a listing of the three main Forms, Main Form, Customer Management & Retail Management Form:
New Requirement for Customer Management Form

- The current *Customer Management* Form requires only ONE MODIFICATION.
- The only issue we have now that we are implementing actual database code, is the following:
  - **THE DELETE BUTTON** simply REMOVES AN ITEM OR CUSTOMER FROM THE COLLECTION ONLY! NOT FROM DATABASE
  - **WE NEED TO TELL THE ITEM OR CUSTOMER TO DELETE ITSELF FROM DATABASE BEFORE REMOVING IT FROM THE COLLECTION**
  - So, we will need to modify the exiting code in the Delete_Click Event-handler to perform the following steps:

`'Step 1-Calls DATA ACCESS METHOD TO DELETE OBJECT FROM DB
objCustomerList.DeleteObject(txtIDNumber.Text.Trim)`

`'Step 2-Calls Remove() method to REMOVE OBJECT FROM COLLECTION
bolResults = objCustomerList.Remove(txtIDNumber.Text.Trim)`

- **Note** that we are first DELETING the object from the DATABASE, then DELETING THE OBJECT FROM COLLECTION

- View of Customer Retail Screen. No changes are made in the graphical layout, but inside the DELETE CLICK EVENT, WE MODIFY THE CODE AS SHOWN ABOVE:
New Requirement for Retail Management Form

- The current Retail Management Form has the following characteristics:
  - Populates text controls with default object data in the Form_Load() event
  - LOADS the CustomerList Collection with ALL Business Objects from FILE
  - Search the CustomerList Collection for customer that is going to shop
  - Allows for Shopping via “By Now” button
  - SAVES the Collection to FILE when closing via the FORM_CLOSE event

- Comments on current Retail Management Form:
  - Not efficient:
    - Bad performance for Retail or Point-Of-Sales application that handles sales transactions with customers because it loads all customer objects (records) to memory in order to work on ONE CUSTOMER ONLY
    - Loading memory with objects you are not using!
  - Retail User-Interface should really deal with the ONE CUSTOMER OBJECT.
  - THE CUSTOMER OBJECT SHOULD LOAD & SAVE ITSELF TO DATABASE.
  - FOR NEXT CUSTOMER, A NEW OBJECT IS CRATED AND PROCESS REPEATED.

- NEW FORM has the following characteristics:
  - Retail Form ONLY OPERATES ON ONE CUSTOMER OBJECT AT A TIME.
  - The GET or LOAD PROCESS WILL LOAD THE ONE CUSTOMER WITH DATA FROM DATABASE
  - THE ONE CUSTOMER OBJECT SAVE ITSELF TO DATABASE.
  - FOR NEXT CUSTOMER, A NEW OBJECT IS CRATED AND PROCESS REPEATED.
  - FORM HAS USER-INTERFACE LOGIC TO SET the Customer Information UI CONTROLS TEXT BOXES to READ-ONLY. ONLY the CUSTOMER ID TEXT BOX will allow input from the user. The other controls are READ-ONLY.
  - The Purchase ITEM Text Box will only activate when the CUSTOMER RECORD IS LOADED, all other times it is DISABLED.

- View of new Retail Screen:
7.3.5 Problem Statement

**Single-Tier Client/Server**

- The requirements for Sample program #1 are as follows:

**Example #1 – Small Business Customer Retail Management Application Using MS ACCESS**

**Problem statement:**
- Upgrade the Customer Management application from Lecture 3B Sample Program #5 by adding DATA ACCESS CODE using ADO.NET
- The requirements are as follows:

**Application Architecture – Programming Methodology**
- Continue to implement the 3-tiered layer Application Architecture:

**Client/Server Architecture – One-Tier Client/Server**
- Maintain the 1-tiered client/server architecture:

---

**Business Object Layer – Business Class & DLL Requirements**
- Keep the DLL COMPONENT and Object Model and Business Rules
- Create the DATABASE CONNECTION STRING IN-LINE within THE CLASSES OR OBJECTS
- DELETE the FILE ACCESS CODE from the DataPortal_Fetch() & DataPortal_Save() methods in clsCustomerList

**Presentation/UI Layer – Client Process requirements:**
- Maintain the same Form/User Interface requirements
- No changes required

**Data Service Layer – Database Requirements**
- Create an access database to support the application
- Name the database smallbusinessapp.mdb
- Place the smallbusinessapp.mdb file in the User-Interface Client Application BIN\DEBUG FOLDER

---
HOW IT’S DONE:

Data Service Layer (Microsoft Access Database 2003)

Part I – Create The MS Access Database:

Step 1: Open Microsoft Access 2003 & Create the Database

- Open MS Access 2003 and create a Database named SmallBusinessApp.
- This was done as follows:

Step 1: Create Blank Database in MS Access 2003:

a. Open MS Access 2003
b. Select File|New… to invoke the New File Window on the right-hand pane of MS Access & Select Blank Database…:

c. In the File New Database screen, name the database
d. Browse to the desire path & click Create:
e. The SmallBusinessApp.mdb Database is ready:

f. File Structure:
**Step 2: Create the Database Table(s) and Enter Data**

- In the Object Windows, Create the table(s) and enter test data. This was done as follows:

**Step 1: Create the Customer Table:**

- **a.** Click the “Table” Object and double-click the “Create table in Design View” icon:

  ![Image of Object Windows with Create Table in Design View icon](image)

- **b.** In the Table Design View, enter the following columns & properties:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Properties</th>
<th>Other Settings/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer_ID</td>
<td>Text</td>
<td>Text size = Default = 50</td>
<td>Set as PRIMARY KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Steps:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. Enter the name &amp; data type for the column</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. Right-Click on row and in the drop-down</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>menu select Primary Key.</td>
</tr>
<tr>
<td>Customer_Name</td>
<td>Text</td>
<td>Text size = Default = 50</td>
<td></td>
</tr>
<tr>
<td>Customer_SSNum</td>
<td>Text</td>
<td>Text size = Default = 50</td>
<td></td>
</tr>
<tr>
<td>Customer_BDate</td>
<td>Date/Time</td>
<td>Short Date</td>
<td></td>
</tr>
<tr>
<td>Customer_Address</td>
<td>Text</td>
<td>Text size = Default = 50</td>
<td></td>
</tr>
<tr>
<td>Customer_Phone</td>
<td>Text</td>
<td>Text size = Default = 50</td>
<td></td>
</tr>
<tr>
<td>Customer_TotalItems</td>
<td>Integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
c. The Table Design window should now look as follows:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer_ID</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Customer_Name</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Customer_Address</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Customer_Date</td>
<td>Date/Time</td>
<td></td>
</tr>
<tr>
<td>Customer_Phone</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Customer_Total</td>
<td>Number</td>
<td></td>
</tr>
</tbody>
</table>

Field Properties:
- Format: Short Date
- Input Mask: 
- Default Value: 
- Validation Rule: 
- Validation Text: 
- Required: No
- Indexed: No
- DDE Mode: No Control
- String Mode: Name

d. Save and Name the table: “Customer”. Table now appears in the main object window:
Step 3: Add Test Data to the table:

a. Double-Click on the Customer icon in the object window to invoke the table’s "Data View":

b. Insert the cursor in the first row and begin typing in the data from your previous application CustomerData.txt file.

c. After all data entry has been completed, the “Data View” should look as follows:

d. Save & close the table
Step 3: Create Queries & SQL Statements to test the database

- Now we create queries to test our database.
- NOTE! In my examples and screen shot, I will use the SQL VIEW of MS Access to simply write the SQL Statements instead of using the GRAPHICAL DESIGNER.
- We will create the queries and SQL Statements required for the application. Once these queries are tested, we can COPY/PASTE and modify them into our DO.NET code:

**Step 1: Create & Test Query to Select a Customer by ID:**

- Create and test the Query to return the record for a Customer by ID.

  a. The query should have the following syntax and format:

    - **Syntax:**
      ```sql
      SELECT *
      FROM Table
      WHERE Column1 = <value>;
      ```
    - **Example:**
      - String values in Access must be enclosed in single-quotes and end in semicolon “;”
      - Example:
        ```sql
        SELECT * FROM Customer WHERE Customer_ID = '111';
        ```

  b. In the “Object Window” select the “Queries” Button:

  c. Double-Click on the “Create Query in Design View” icon to invoke the Design View Window

  d. The “Show Table” screen provides a listing of the available tables for you to select and add to the designer:
e. Select the "Customer" table from the "Show Table" screen, and click the "Add" button.
f. Repeat this process until all tables you need for the query are entered. Click "Close" button to close the "Show Table" screen.
g. The designer should now looks as follows:

![Image of query designer with tables selected]

h. Now switch to SQL VIEW. Select the "Designer" Windows, then in the menu bar, select "View|SQL View", this will invoke the query screen:

![Image of SQL query screen]

i. Now enter the query into the query window:

```
SELECT * 
FROM Customer; 
WHERE Customer_ID = '111';
```

![Image of query window with SQL query]

j. Execute, in the menu bar, select "Query|Run" or simply click on the run icon:

![Image of query results with customer information]

Record: 1 of 2
k. Save and name the query “Select Customer by ID”:

![Query Naming](image)

l. In the “Object Window” should now have an icon for the query:

![Object Window](image)

**Step 2: Create & Test the UPDATE SQL Statement:**

- Create and test the Query update or modify a Customer’s record on the database.
- Repeat the same steps used for the previous query “Select Customer by ID”

a. The query should have the following syntax and format:

   - **Syntax:**
     ```sql
     UPDATE Table
     SET Column2=<value2>,
     Column3=<value3>,
     Column4=<value4>,
     Column5=<value5>
     Column6=<value6>,
     Column7=<value7>
     WHERE Column1=<value1>;
     ```

   - **Example:**
     ```sql
     UPDATE Customer
     SET Customer_Name = 'Joe Smith',
     Customer_BDate = #5/23/1971#,
     Customer_Address = '111 Smith Street',
     Customer_Phone = '718 260-5000',
     Customer_TotalItems = 250
     WHERE Customer_ID='111';
     ```

   - **THE UPDATE QUERY REQUIRED FOR OUR APPLICATION DOES NOT MODIFY THE SOCIAL SECURITY NUMBER COLUMN BECAUSE SSNUM PROPERTY IN THE CUSTOMER HAS A WRITE-ONCE BUSINESS RULE. THIS RULE STATES THAT THE SOCIAL SECURITY CAN ONLY BE WRITTEN ONCE, UPON THE CREATION OF THE NEW CUSTOMER. THEREFORE THE UPDATE STATEMENT CANNOT HAVE AN ENTRY TO MODIFY THE CUSTOMER_SSNUM COLUMN.**

- Example:
b. In the “Object Window” select the “Queries” Button:
c. Double-Click on the “Create Query in Design View” icon to invoke the Design View Window
d. The “Show Table” screen provides a listing of the available tables for you to select and add to the designer:
e. Select the “Customer” table from the “Show Table” screen, and click the “Add” button.
f. Repeat this process until all tables you need for the query are entered. Click “Close” button to close the “Show Table” screen.
g. The designer should now looks as follows:

![Design View Window]

h. Now switch to SQL VIEW. Select the “Designer” Windows, then in the menu bar, select “View|SQL View”
i. Now enter the query into the query window:

```
UPDATE Customer
SET Customer_Name = 'Joe Smith',
    Customer_EDate = '5/23/1971',
    Customer_Address = '111 Smith Street',
    Customer_Phone = '718-260-5000',
    Customer_TotalItems = 250
WHERE Customer_ID = 111;
```

j. Execute query. You are prompted to modify a record, click yes:

![Prompt to Update]

k. Save and name the query “Update Customer”. Exit the “SQL Design” view

l. Since Action Queries DO NOT RETURN RECORDS, open the Customer Table in “Data View” to verify the UPDATE was successfully done:
m. In the “Object Window” should now have an icon for the UPDATE query:

![Object Window](image)

### Step 3: Create & Test the INSERT SQL Statement:

- Create and test the Query to INSERT a NEW Customer record to the database.
- Repeat the same steps used for the previous query “UPDATE Customer” query

**a.** The query should have the following syntax and format:

- **Syntax:**

  ```sql
  INSERT INTO Table(Column1, Column2, Column3, Column4, Column5, Column6, Column7)
  VALUES(<value1>,<value2>,<value3>,<value4>,<value5>,<value6>,<value7>);
  ```

- **Example:**

  ```sql
  INSERT INTO Customer(Customer_ID, Customer_Name, Customer_SSNum, 
  Customer_BDate, Customer_Address, Customer_Phone, 
  Customer_TotalItems)
  VALUES('777', 'Amanda Rodriguez', '777-77-7777', #12/07/87#, '777 Madison Ave', 
  '212-777-7777', 50);
  ```

**b.** Follow all the steps from previous query

**c.** Now switch to SQL VIEW. Select the “Designer” Windows, then in the menu bar, select “View|SQL View”

**d.** In “View|SQL View” enter the query into the query window:

```
INSERT INTO Customer(Customer_ID, Customer_Name, Customer_SSNum, Customer_BDate, Customer_Address, Customer_Phone, Customer_TotalItems)
VALUES('777', 'Amanda Rodriguez', '777-77-7777', #12/07/87#, '777 Madison Ave', '212-777-7777', 50);
```

**e.** Execute query. You are prompted to modify a record, click yes:
f. Save and name the query "INSERT Customer". Exit the "SQL Design" view.

g. Since Action Queries DO NOT RETURN RECORDS, open the Customer Table in "Data View" to verify the INSERT was successfully done. Note that a new record has been inserted ‘777’:

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>Customer_Name</th>
<th>Customer_SSN</th>
<th>Customer_BDate</th>
<th>Customer_Address</th>
<th>Customer_Phone</th>
<th>Customer_TotalItems</th>
</tr>
</thead>
<tbody>
<tr>
<td>222</td>
<td>Angel Rod</td>
<td>222-22-2222</td>
<td>3/12/1967</td>
<td>222 Jay Street</td>
<td>718 260-6000</td>
<td>99</td>
</tr>
<tr>
<td>333</td>
<td>Sam Franks</td>
<td>333-22-3333</td>
<td>3/12/1967</td>
<td>333 Jay Street</td>
<td>718 250-5333</td>
<td>0</td>
</tr>
<tr>
<td>444</td>
<td>Mary Jones</td>
<td>444-44-4444</td>
<td>1/23/1974</td>
<td>444 Jay Street</td>
<td>718 260-4444</td>
<td>0</td>
</tr>
<tr>
<td>555</td>
<td>Nancy Ramirez</td>
<td>555-56-5555</td>
<td>1/23/1975</td>
<td>555 JAY STREET</td>
<td>718 250-5655</td>
<td>6</td>
</tr>
<tr>
<td>777</td>
<td>Amanda Rodriguez</td>
<td>777-77-7777</td>
<td>12/7/1997</td>
<td>777 Madison Ave</td>
<td>212-777-7777</td>
<td>50</td>
</tr>
</tbody>
</table>

h. In the "Object Window" should now have an icon for the DELETE query.
Step 4: Create & Test the DELETE SQL Statement:

- Create and test the Query to DELETE a Customer record from the database.
- Repeat the same steps used for the previous query query

a. The query should have the following syntax and format:

   - Syntax:
     
     ```sql
     DELETE
     FROM Table
     WHERE Column1 = <value1>;
     ```

   - Example:
     
     ```sql
     DELETE
     FROM Customer
     WHERE Customer_ID = '777';
     ```

b. Follow all the steps from previous query

- Now switch to SQL VIEW. Select the “Designer” Windows, then in the menu bar, select “View|SQL View”

- In “View|SQL View” enter the query into the query window:

- Execute query. You are prompted to modify a record, click yes:

- Save and name the query “DELETE Customer”. Exit the “SQL Design” view

- Since Action Queries DO NOT RETURN RECORDS, open the Customer Table in “Data View” to verify the INSERT was successfully done. Note the record whose ID is ‘777’ has been deleted:

h. In the “Object Window” should now have an icon for the INSERT query:
Step 5: Create & Test the SELECT ALL SQL Statement:

- Create and test the Query to SELECT ALL Customer records from the database.
- Repeat the same steps used for the previous query.

a.  The query should have the following syntax and format:

   - Syntax:
     
     \[
     \text{SELECT} \ast \\
     \text{FROM} \ Table
     \]

   - Example:
     
     \[
     \text{SELECT} \ast \\
     \text{FROM} \ Customer
     \]

b.  Follow all the steps from previous query

c.  Now switch to SQL VIEW. Select the “Designer” Windows, then in the menu bar, select “View|SQL View”

d.  In “View|SQL View” enter the query into the query window:

\[
\text{SELECT} \ast \\
\text{FROM} \ Customer
\]

```
<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>Customer_Name</th>
<th>Customer_SSN</th>
<th>Customer_BDate</th>
<th>Customer_Address</th>
<th>Customer_Phone</th>
<th>Customer_TotalItems</th>
</tr>
</thead>
<tbody>
<tr>
<td>222</td>
<td>Angel Ruiz</td>
<td>222-22-2222</td>
<td>3/12/1987</td>
<td>222 Jay Street</td>
<td>718.360-9000</td>
<td>99</td>
</tr>
<tr>
<td>333</td>
<td>Sam Franks</td>
<td>333-22-3333</td>
<td>3/12/1987</td>
<td>333 Jay Street</td>
<td>718.360-5333</td>
<td>0</td>
</tr>
<tr>
<td>444</td>
<td>Mary Jones</td>
<td>444-44-4444</td>
<td>1/2/1974</td>
<td>444 Jay Street</td>
<td>718.360-4444</td>
<td>0</td>
</tr>
<tr>
<td>555</td>
<td>Nancy Ramirez</td>
<td>555-55-5555</td>
<td>12/3/1975</td>
<td>555 JAY STREET</td>
<td>718.360-5555</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
Record: 1 of 5
```

e.  Execute query:

```
Customer_ID | Customer_Name  | Customer_SSN | Customer_BDate | Customer_Address     | Customer_Phone | Customer_TotalItems |
-------------|----------------|--------------|----------------|---------------------|----------------|---------------------|
222          | Angel Ruiz     | 222-22-2222  | 3/12/1987      | 222 Jay Street      | 718.360-9000   | 99                  |
333          | Sam Franks     | 333-22-3333  | 3/12/1987      | 333 Jay Street      | 718.360-5333   | 0                   |
444          | Mary Jones     | 444-44-4444  | 1/2/1974       | 444 Jay Street      | 718.360-4444   | 0                   |
555          | Nancy Ramirez  | 555-55-5555  | 12/3/1975      | 555 JAY STREET      | 718.360-5555   | 0                   |
```

```
Record: 1 of 5
```

f.  Save and name the query “SELECT ALL Customers”. Exit the “SQL Design” view

g.  In the “Object Window” should now have an icon for the SELECT ALL query:

- AT THIS POINT WE HAVE CREATED OUR DATA SERVICE LAYER USING MS ACCESS
- AND SUCCESSFULLY TESTED THE POSSIBLE QUERIES THAT WE MAY NEED.
- AT THIS POINT WE ARE READY TO PROCEED TO CREATE OUR APPLICATION
ADD ADO.NET Code to Data Access Methods in Business Objects

- We are now ready to add the ADO.NET CODE TO OUR DATA ACCESS METHODS OF THE BUSINESS OBJECTS.
- At this point, the Customer Management Application has been upgraded to contain all the Business Logic, Validation & Data Access methods based on the rules imposed by the BusinessBase & BusinessCollectionBase classes.
- Now we need to do the following:
  - Remove the File Access Code from the Collection Class. Going forward we will use the Access Database we just created
  - Add the ADO.NET code inside the Data Access Methods
  - Test the application

Step 1: Open the Customer Management Application with Business Rules & Logic Applied

- Open the last version of the Customer Management Retail Application.
- This version contains all the Business Rules, Validation, Data Access methods, and File Access Code to continue to save and load from a file.
- When you open the application, the application will contain the DLL & Windows Client Projects as follows:
Step 2: Add Data Access Code to clsCustomer class

- Now we focus on the Data Access code to the clsCustomer class. We are referring to the DataPortal_XXX methods that are intended to carry out the Data Access.
- In the previous example or version, we left these methods empty for the customer class.
- Now we will add ADO.NET code to carry out the Data Access.
- Once again, the diagram below illustrates the current structure of the clsCustomer class. If we designed our Business Class correctly, we should only need to work with the section labeled “Protected Data Access Code”.
- Nevertheless, we will add a NEW Private Region and we will declare the Connection String global within the class as a private member therefore the new structure will look as follow with one additional region.
Step 1: Add Connection String in the General Class declaration and Private data Section:

- A requirement to the application is that the database Connection String need be embedded to each Business Object.
- Also the decision was made to OPEN & CLOSE the database connection during each DATA ACCESS PROCESS. This means that each DATA ACCESS METHOD WILL OPEN THE CONNECTION, PERFORM THE DATA ACCESS AND THEN CLOSE THE CONNECTION.
- We will create a Class-Level Connection String variable declaration so ALL DATA ACCESS METHOD CAN USE THIS CONNECTION STRING. MORE IMPORTANT WE CAN CHANGE IT ONLY IN ONE LOCATION INSIDE THE BUSINESS OBJECT.
- IMPORTANT! NOTE THAT IF THE PATH OF THE DATABASE IS NOT PROVIDED IN Connection String, THE IT MUST RESIDE IN THE DEBUG\BIN FOLDER:

```vbnet
Option Explicit On
Option Strict On

Imports System.IO 'File/IO
Imports System.Data 'Data Access (DataSet)
Imports System.Data.OleDb 'OLEDB Provider
Imports System.Configuration 'Configuration File for DB Connection
'Keep commented. will be configure later
Imports System.Runtime.Remoting 'Remoting
Imports System.Runtime.Remoting.Channels 'Remoting

<Serializable()>
Public Class clsCustomer
    Inherits clsPerson

#Region "Connection String Declaration"
    'Data Access Connection string. If FULL PATH is provide, database must
    'Be located in the Bin\Debug folder of application
    Private Const strConn As String = "Provider=Microsoft.Jet.OleDb.4.0;" & _
        "Data Source=SmallBusinessApp.mdb"
#End Region

#Region "Protected Data Access Methods"

'****************************************************************************
'************** Data Access Code for Creating a New Business Object with
'************** DEFAULT DATA from database
'****************************************************************************
 Protected Overrides Sub DataPortal_Create()
    'Create object and assign default values from database etc.

    'ADD DATA ACCESS CODE HERE USING ADO.NET

    'At the end, set New flag to True a new object is created
    MyBase.MarkNew()
End Sub
```
Step 3: Open the DataPortal_Fetch(Key) method and add ADO.NET Data Access Code:

- We will use the ADO.NET Connection, Command, Parameters and DataReader objects to carry out the Data Access.

```vbnet
Protected Overrides Sub DataPortal_Fetch(ByVal Key As Object)
' Step 1 - Create Connection, assign Connection to string
Dim objConn As New OleDbConnection(strConn)

' Step 2 - Open connection
Try
  objConn.Open()
  ' Step 3 - Create SQL string
  Dim strSQL As String = "SELECT * FROM Customer WHERE Customer_ID = ?"
  ' Step 4 & 5 - Create Command object, pass query/connection & Add parameters
  Dim objCmd As New OleDbCommand(strSQL, objConn)
  objCmd.Parameters.Add("@Customer_ID", OleDbType.VarChar).Value = Key

  ' Step 6 - Create DATAREADER object & Execute Query
  Dim objDR As OleDbDataReader = objCmd.ExecuteReader

  ' Step 7 - Test to make sure there is data in the DataReader Object
  If objDR.HasRows Then
    ' Step 8a - Call Read() Method to point and read the first record
    objDR.Read()
    ' Step 8b - Extract data from a row & Populate Yourself.
    Me.CustomerID = CStr(objDR.Item(0))
    Me.Name = CStr(objDR.Item(1))
    Me.SocialSecurity = CStr(objDR.Item(2))
    Me.BirthDate = CDate(objDR.Item(3))
    Me.Address = CStr(objDR.Item(4))
    Me.Phone = CStr(objDR.Item(5))
    Me.TotalItemsPurchased = CInt(objDR.Item(6))
  Else
    ' Step 9 - No data returned, Record not found!
    Throw New System.ApplicationException("Load Error! Record Not Found")
  End If

  ' Step 10 - Terminate ADO Objects
  objDR.Close()
  objDR = Nothing
  objCmd.Dispose()
  objCmd = Nothing

  ' Step B - Trap for BO, App & General Exceptions
  Catch objBOEx As NotSupportedException
    Throw New System.NotSupportedException(objBOEx.Message)
  Catch objA As ApplicationException
    Throw New System.ApplicationException(objA.Message)
  Catch objEx As Exception
    Throw New System.Exception("Load Error: " & objEx.Message)
  Finally
    ' Step 11 - Terminate connection
    objConn.Close()
    objConn.Dispose()
    objConn = Nothing
  End Try

  ' At the end, set New flag to False. NOT Dirty since found in database
  MyBase.MarkOld()
End Sub
```
Step 4: Open DataPortal_Update Method and add data access code:

- We will use the ADO.NET Connection, Command, Parameters and DataReader objects to carry out the Data Access.

```vbnet
Protected Overrides Sub DataPortal_Update()
    'Step 1 - Create Connection, assign Connection to string
    Dim objConn As New OleDbConnection(strConn)

    'Step A - Start Error Trapping
    Try
        'Step 2 - Open connection
        objConn.Open()

        'Step 3 - Create Query, Command Object & initialize
        Dim strSQL As String
        strSQL = 
            "UPDATE(Customer)"
            & 
            "SET Customer_Name = ?,"
            & 
            "Customer_BDate = ?,"
            & 
            "Customer_Address = ?,"
            & 
            "Customer_Phone = ?,"
            & 
            "Customer_TotalItems = ?"
            & 
            "WHERE Customer_ID = ?"

        'Step 4 - Create Command object, pass string and connection object as arguments
        Dim objCmd As New OleDbCommand(strSQL, objConn)

        'Step 5 - Add Parameter to Collection & Set Value
        objCmd.Parameters.Add("@Customer_Name", OleDbType.VarChar).Value = Me.Name
        objCmd.Parameters.Add("@Customer_BDate", OleDbType.Date).Value = Me.BirthDate
        objCmd.Parameters.Add("@Customer_Address", OleDbType.VarChar).Value = Me.Address
        objCmd.Parameters.Add("@Customer_Phone", OleDbType.VarChar).Value = Me.Phone
        objCmd.Parameters.Add("@Customer_TotalItems", OleDbType.Integer).Value = Me.TotalItemsPurchased

        'Step 6 - Execute Non-Row Query Test result and throw exception if failed
        Dim intRecordsAffected As Integer = objCmd.ExecuteNonQuery()
        If intRecordsAffected <> 1 Then
            Throw New System.ApplicationException("UPDATE Query Failed")
        End If

        'Step 7 - Terminate Command Object
        objCmd.Dispose()
        objCmd = Nothing

        'Step 8 - Trap for B0, App & General Exceptions
        Catch objBOEx As NotSupportedException
            Throw New System.NotSupportedException(objBOEx.Message)
        Catch objA As ApplicationException
            Throw New System.ApplicationException(objA.Message)
        Catch objEx As Exception
        Finally
            'Step 8 - Terminate connection
            objConn.Close()
            objConn.Dispose()
            objConn = Nothing
    End Try

    'Set New flag to False since exist in database/and is Not dirty any longer
    MyBase.MarkOld()
End Sub
```
Step 5: Open Database DataPortal_Insert() and Add Data Access code to Insert a Record into the Database

```
'*****************************************************************************
'Data Access Code to insert a new object to database
Protected Overrides Sub DataPortal_Insert()

'Step 1 - Create Connection, assign Connection to string
Dim objConn As New OleDbConnection(strConn)

'Step A - Start Error Trapping
Try

'Step 2 - Open connection
objConn.Open()

'Step 3 - Create Command, Query, assing query, and assign connection
Dim strSQL As String
strSQL = "INSERT INTO Customer ( Customer_ID,Customer_Name," & 
& "Customer_SSNum,Customer_BDate,Customer_Address," & 
& "Customer_Phone,Customer_TotalItems )" & 
& "VALUES (?,?, ?, ?, ?, ?, ?, ?)"

'Step 4 - Create Command object, pass connection info as arguments
Dim objCmd As New OleDbCommand(strSQL, objConn)

'Step 5 - Add Parameter to Parameters Collection
objCmd.Parameters.Add("@Name", OleDbDbType.VarChar).Value = Me.Name
objCmd.Parameters.Add("@SSNum", OleDbDbType.VarChar).Value = Me.SocialSecurity
objCmd.Parameters.Add("@BirthDate", OleDbDbType.Date).Value = Me.BirthDate
objCmd.Parameters.Add("@Address", OleDbDbType.VarChar).Value = Me.Address
objCmd.Parameters.Add("@PhoneNumber", OleDbDbType.VarChar).Value = Me.Phone
objCmd.Parameters.Add("@Customer_TotalItems", OleDbDbType.Int).Value = Me.TotalItemsPurchased

'Step 6 - Execute Non-Row Query Test result and throw exception if failed
Dim intRecordsAffected As Integer = objCmd.ExecuteNonQuery()
If intRecordsAffected <> 1 Then
    Throw New System.ApplicationException("INSERT Query Failed")
End If

'Step 7 - Terminate Command Object
objCmd.Dispose()
objCmd = Nothing

'Step B - Trap for BO, App & General Exceptions
Catch objBO As NotSupportedException
    Throw New System.NotSupportedException(objBO.Message)
Catch objA As ApplicationException
    Throw New System.ApplicationException(objA.Message)
Catch objEx As Exception

Finally
'Step 8 - Terminate connection
objConn.Close()
objConn.Dispose()
objConn = Nothing
End Try

'Set New flag to False since exist in database/and is Not dirty any longer
MyBase.MarkOld()
End Sub
```
Step 6: Open Database DataPortal_DeleteObject() and Add Data Access code to delete a Record from the Database

```
'*********************************************************************
'Data Access Code to immediately delete an object from database.
Protected Overrides Sub DataPortal_DeleteObject(ByVal Key As Object)
  'ADO.NET Queries for deleting (Delete/From/Where) or Stored Procedures

  'Step 1-Create Connection, assign Connection to string
  Dim objConn As New OleDbConnection(strConn)

  'Step A-Start Error Trapping
  Try

    'Step 2-Open connection
    objConn.Open()

    'Step 3-Create Command, Query, assing query, and assign connection
    Dim strSQL As String = "DELETE FROM Customer WHERE Customer_ID = ?"

    'Step 4-Create Command object, pass string and connection object as arguments
    Dim objCmd As New OleDbCommand(strSQL, objConn)

    'Step 5-Add Parameter to Collection & Set Value
    objCmd.Parameters.Add("@Customer_ID", OleDbType.VarChar).Value = Key

    'Step 6-Execute Non-Row Query Test result and throw exception if failed
    Dim intRecordsAffected As Integer = objCmd.ExecuteNonQuery()
    If intRecordsAffected <> 1 Then
      Throw New System.ApplicationException("DELETE Query Failed")
    End If

    'Step 7-Terminate Command Object
    objCmd.Dispose()
    objCmd = Nothing

  'Step A-Trap for BO, App & General Exceptions
  Catch objBO As NotSupportedException
    Throw New System.NotSupportedException(objBO.Message)
  Catch objA As ApplicationException
    Throw New System.ApplicationException(objA.Message)
  Catch objEx As Exception
  Finally

    'Step 8-Terminate connection
    objConn.Close()
    objConn.Dispose()
    objConn = Nothing
  End Try

  'Object no longer in database, therefore reset our status to be a new object
  MyBase.MarkNew()
End Sub

#End Region
```
Step 3: Add Data Access Code to the clsCustomerList Collection Class

Data Access Requirements

- Now we focus on the Data Access code to the clsCustomerList Collection Business Class. We are referring to the DataPortal_Fetch(), DataPortal_Save() and DataPortal_DeleteObject() methods that are intended to carry out the Data Access.
- In the previous example or version, we used FILE ACCESS CODE to simulate the database, now we will REMOVE THE FILE ACCESS CODE AND ADD ADO.NET code to carry out the true Data Access.
- Once again, the diagram below illustrates the current structure of the clsCustomerList class. If we designed our Business Class correctly, we should only need to work with the section labeled “Protected Data Access Code”.
- Nevertheless, we will add a Private Region and we will declare the Connection String global within the class as a private member therefore the new structure will look as follow with one additional region:

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

'Keep commented, will be configure later
Imports System.Runtime
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels

<Serializable>
Public Class clsCustomerList
    Inherits BusinessCollectionBase

    #Region "Connection String Declaration"
    'Data Access Connection string. If no path is provide, database wont
    'Be located in the Bin\Debug folder of application
    Private Const strConn As String = "Provider=Microsoft.Jet.OleDB.4.0;" \
    "Data Source=SmallBusinessApp.mdb"
    #End Region

    Public Properties Declarations
    Public Wrapper Methods Declarations
    Public Regular Methods Declarations
    Public Data Access Methods
    Protected Data Access Methods
    Helper Methods
```
Step 1: General Class declaration:

- We continue to derive the class from `BusinessCollectionBase` and adding the Data Access Library declarations.
- In addition we add a region and declaration for the connection string.
- **IMPORTANT! NOTE THAT IF THE PATH OF THE DATABASE IS NOT PROVIDED IN Connection String, THE IT MUST RESIDE IN THE DEBUG BIN FOLDER**

```vbnet
Option Explicit On
Option Strict On

Imports System.IO          'File/I0
Imports System.Data        'Data Access (DataSet)
Imports System.Data.OleDb  'OLEDB Provider
Imports System.Configuration 'Configuration File for DB
Connection

'Keep commented. will be configure later
Imports System.Runtime.Remoting                          'Remoting
Imports System.Runtime.Remoting.Channels               'Remoting
<Serializable()>
Public Class clsCustomerList
    Inherits BusinessCollectionBase

#Region "Connection String Declaration"
    Private Const strConn As String = "Provider=Microsoft.Jet.OleDB.4.0;" & _
        "Data Source=SmallBusinessApp.mdb"
#End Region
```

Step 2: ADD ADO.NET code to `DataPortal_Create`: (OPTIONAL NOT IMPLEMENTED IN THIS PROGRAM)

- If our Business Objects required DEFAULT DATA from the database we can place that code here.

```vbnet
#Region "Protected Data Access Methods"
'******************************************************************************************
'//'<summary>
'//' Data Access or other Code for Creating a New Business COLLECTION Object
'//' Used when object requires data from db upon creation
'//' </summary>
'//'<remarks></remarks>
Protected Overrides Sub DataPortal_Create()
    'Create COLLECTION object and assign default values from database etc.

End Sub
```
Explanation of FETCH() Implementation

- We CANNOT IMPLEMENT the FETCH method in the Collection Class using the same ALGORITHM used in the FILE ACCESS CODE.
- In the FILE ACCESS CODE, the algorithm was as follows.

1. Load or READ a LINE from FILE
2. CREATE A NEW OBJECT
3. PARSED LINE (using SPLIT() method)
4. POPULATE THE OBJECT WITH THE DATA FROM THE PARSED LINE SETTING ITS PROPERTIES!!!:
5. ADD THE OBJECT TO THE COLLECTION
6. REPEAT

- We CANNOT TAKE THIS APPROACH WITH OUR DATA ACCESS CODE USING THE BUSINESS OBJECTS DUE TO THE FOLLOWING REASONS:
  - CREATING a NEW OBJECT SETS the NEW FLAG to TRUE
  - IN THE FILE ACCESS CODE, NOTE THAT WE ARE SETTING THE OBJECT VIA THE PROPERTIES which has the following affect to the BUSINESS OBJECT:
    - IN THE BUSINESS OBJECT THIS WILL MAKE THE OBJECT DIRTY
    - TRIGGER ANY VALIDATION CODE. This is NO PROBLEM, since we simply TRY/CATCH A NotSupportedException
  - So what we have is a NEW & DIRTY OBJECT:
    - LOADING A BUSINESS OBJECT MEANS IT EXISTS IN THE DATABASE AND IS OLD (NOT NEW).
    - SO AN OBJECT AFTER BEING LOADED CANNOT BE DIRTY & NEW, IT SHOULD BE DIRTY & OLD!!!!!
    - THE ONLY METHOD OF MARKING AN OBJECT AS OLD (NEW = FALSE) IS VIA THE DATA ACCESS METHODS, of the BUSINESS OBJECT ITSELF.
  - In summary, WE CANNOT SIMPLY GET DATA FROM THE DATABASE AND POPULATE THE OBJECT VIA THE PROPERTIES AS WE DID WITH THE FILE ACCESS CODE!
  - WE NEED TO LET THE OBJECT LOAD ITSELF BY CALLING:

        objCustomer.Load(KEY)

  - BUT THE PROBLEM IS THAT WE NEED TO KNOW THE KEY OF EACH OBJECT BEFORE HAND! SO WE NEED TO QUERY THE DATABASE FOR ALL THE KEYS FIRST, THEN CALL LOAD() FOR EACH OF THE KEYS.

- FETCH ALGORITHM:

1. USING ADO.NET, Query the TABLE for ALL KEYS
2. LOOP
3. CREATE a NEW CUSTOMER OBJECT
4. EXTRACT THE NEXT KEY FROM DATAREADER
5. Call OBJECT.Load(KEY)
6. ADD THE OBJECT TO THE COLLECTION
7. REPEAT
Step 3: Add Data Access code to the Method DataPortal_Fetch()

- We now delete the FILE ACCESS Code and replace with ADO.NET Data Access code.

```vbnet
Protected Overrides Sub DataPortal_Fetch()

    'Step 1-Create Connection, assign Connection to string
    Dim objConn As New OleDbConnection(strConn)

    'Step A-Start Error Trapping
    Try

    'Step 2-Open connection
    objConn.Open()

    'Step 3-Create SQL string to get all Primary Keys of Customers
    Dim strSQL As String = "SELECT Customer_ID FROM Customer"

    'Step 4-Create Command object
    Dim objCmd As New OleDbCommand(strSQL, objConn)

    'Step 5-Create DATAREADER object & Execute Query
    Dim objDR As OleDbDataReader = objCmd.ExecuteReader

    'Step 6-Test to make sure there is data in the DataReader Object
    If objDR.HasRows Then
        'Step 7-Iterate through DataReader one record at a time.
        Do While objDR.Read
            'Step 8-Customer Object
            Dim objItem As New clsCustomer
            'Step 9-Get Key from DataReader
            Dim strKey As String = objDR.GetString(0)

            'Step 10-ITEM will load itself based on key.
            objItem.Load(strKey)

            'Step 11-Add object to collection
            Me.Add(objItem.CustomerID, objItem)

            'Step 12-Terminate new object
            objItem = Nothing
        Loop
        Else
            'Step 13-No data returned, Record not found.
            Throw New System.ApplicationException("Load Error! Record Not Found")
        End If
    End If

    'Step 14-Terminate Command Object
    objCmd.Dispose()
    objCmd = Nothing
    objDR.Close()
    objDR = Nothing

    'Step 15-Trap for Business Object, OleDB, Record not found & general Exceptions
    Catch objBOEx AsNotSupportedException
        Throw New System.NotSupportedException(objBOEx.Message)
    Catch objA As ApplicationException
        Throw New System.ApplicationException(objA.Message)
    Catch objEx As Exception
        Throw New System.Exception("Load Error: " & objEx.Message)
    Finally
        'Step 16-Terminate connection
        objConn.Close()
        objConn.Dispose()
        objConn = Nothing
    End Try
End Sub
```
Step 4: Data Access code for Method DataPortal_Save()

- We now DELETE the FILE ACCESS Code.
- The MAIN logic here is that there is NO DATA ACCESS CODE REQUIRED!
- We simply iterate through the Collection and ask each Object to Save themselves. Thus the ROOT or PARENT collection is simply telling its children to handle their own DATA ACCESS.

```
'******************************************************************
''""<summary>
''""SAVES all objects from database by Iterating through Collection, and
''""calling Each ITEM object SAVE() method so each Item saves itself
''""</summary>
''""<remarks></remarks>
Protected Overrides Sub DataPortal_Save()

'Iterates through Collection, Calling Each CHILD object.Save() method
'CHILD Objects save themselves
'Step A- Begin Error trapping
Try
  'Step 1-Step 1-Create Temporary Person and Dictionary object POINTERS
  Dim objDictionaryEntry As DictionaryEntry
  Dim objChild As clsCustomer

  'Step 2-Use For..Each loop to iterate through Collection
  For Each objDictionaryEntry In MyBase.Dictionary
    'Step 3-Convert DictionaryEntry pointer returned to Type Person
    objChild = CType(objDictionaryEntry.Value, clsCustomer)

    'Step 4-Call Child to Save itself
    objChild.Save()
  Next

'Step B-Traps for general exceptions.
Catch objE As Exception
  'Step C-Throw an general exceptions
  Throw New System.Exception("Save Error! " & objE.Message)
End Try

End Sub
```
Step 5: Add Data Access Code to DataPortal_DeleteObject

- This method implements the immediate DELETE of an Object from the Collection and Database.
- The MAIN logic here is that there is NO DATA ACCESS CODE REQUIRED!
- We simply iterate through the Collection, searching for the Object whose key is passed into this method.
- Once the object is found, the method calls its DeleteObject(KEY) method to carry out the task. The object deletes itself.

```vbnet
'**********************************************************
''' <summary>
''' DELETES AN OBJECT BY ID from database by Iterating through Collection
''' and calling Each ITEM object DELETE(ID) method so each Item delete itself
''' </summary>
''' <param name="Key"></param>
''' <remarks></remarks>
Protected Overrides Sub DataPortal_DeleteObject(ByVal Key As Object)
    'Iterates through Collection, Calling Each CHILD object.Delete() method
    'CHILD Objects Delete themselves
    Try
        'Step 1- Begin Error trapping
        Dim objDictionaryEntry As DictionaryEntry
        Dim objItem As clsCustomer
        For Each objDictionaryEntry In MyBase.Dictionary
            'Step 3- Convert DictionaryEntry pointer returned to Type Person
            objItem = CType(objDictionaryEntry.Value, clsCustomer)
            'Step 4- Find target object based on key
            If objItem.CustomerID = CStr(Key) Then
                'Step 5- Object deletes itself
                objItem.DeleteObject(Key)
            End If
        Next
    Catch objE As Exception
        'Step C- Throw an general exceptions
        Throw New System.Exception("Delete Error! " & objE.Message)
    End Try
End Sub
#End Region
```
Step 3: Modify the Code in the Module

- NO CHANGES REQUIRED IN THE MODULE.

Step 4: Modify the User-Interface Customer Management Form!

Customer Management Form

- The Customer Management Form looks as follows:

- The requirements stated that ONLY ONE MODIFICATION IS REQUIRED, and that is MODIFY THE DELETE PROCESSESS TO DELETE THE OBJECT IN THE COLLECTION FROM DATABASE AND THEN THE COLLECTION:
'***********************<summary>'''' Name: Event-Handler for btnDelete button'''' Purpose: To delete an object from the collection base on ID or Key'''' </summary>'''' <param name="sender"></param>'''' <param name="e"></param>'''' <remarks></remarks>Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnDelete.Click 'Step A- Begin Error trapping Try Dim bolResults As Boolean 'Step 1-Calls DATA ACCESS METHOD TO DELETE OBJECT FROM DB objCustomerList.DeleteObject(txtIDNumber.Text.Trim) 'Step 2- Calls Remove() method to REMOVE OBJECT FROM COLLECTION bolResults = objCustomerList.Remove(txtIDNumber.Text.Trim) 'Step 3- Clear all controls txtIDNumber.Text = "" txtSSNum.Text = "" txtName.Text = "" txtBirthDate.Text = "" txtAddress.Text = "" txtPhone.Text = "" 'Step 4- If not found display Message & clear all controls If bolResults <> True Then MessageBox.Show("Customer Not Found") End If 'Step B- Traps for Business Rule violations Catch objNSE As NotSupportedException MessageBox.Show("Business Rule violation! " & objNSE.Message) 'Step C- Traps for ArgumentNullException when key is Nothing or null. Catch objX As ArgumentNullException 'Step D- Inform User MessageBox.Show(objX.Message) 'Step E- Traps for general exceptions. Catch objE As Exception 'Step F- Inform User MessageBox.Show(objE.Message) End Try End Sub
Step 5: Modify the User-Interface Retail Management Form!

Retail Management Form

As per the requirements we need to do the following:

1. DO NOT LOAD THE COLLECTION ANYMORE. ONLY LOAD THE ONE CUSTOMER WHICH IS SHOPPING
2. DO NOT SAVE THE COLLECTION ANYMORE. ONLY SAVE THE ONE CUSTOMER SHOPPING
3. WITH THE EXCEPTION OF THE CUSTOMER ID TEXTBOX, SET ALL OTHER TEXTBOXES TO READ-ONLY
4. ENABLE THE Items TEXTBOX ONLY WHEN THE CUSTOMER IS LOADED, OTHER TIMES DISABLE IT

The Retail Management Form looks as follows:

![Retail Management Form](image)

Let’s begin the modifications.

Step 1: NO CHANGES IN THE Form Level Object declaration

NO CHANGES REQUIRED HERE:

```
Private WithEvents objCustomer As BusinessObjectsDLL.clsCustomer
```
Step 2: MODIFY THE FORM_LOAD() event-handler

- The modifications required here are:
  1. WITH THE EXCEPTION OF CUSTOMER ID, DISABLE ALL OTHER TEXT BOXES

```vbnet
'******************************************************************************
''' <summary>
''' Form_Load event. Create object and populate Form controls
''' With object's default values. Also Sets text box to Read-only
''' in MODULE
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Private Sub frmRetailManagement_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'Step A-Begins Exception handling.
    Try

    'Step 1-Set Text boxes to Read-only
    txtName ReadOnly = True
    txtBirthDate.ReadOnly = True
    txtAddress.ReadOnly = True
    txtPhone.ReadOnly = True

    'Step 2-Clear Items textbox
    txtTotalPurchases.ReadOnly = True
    txtItems.Enabled = False

    'Step B-Traps for Business Rule violations
    Catch objNSE As NotSupportedException
        MessageBox.Show("Business Rule violation! " & objNSE.Message)
    'Step C-Traps for general exceptions.
    Catch objE As Exception
        'Step D-Inform User
        MessageBox.Show(objE.Message)
    End Try

End Sub
```
Step 3: Get_Click() event-handler – Load Customer Record from DB & Populate Form

- Now we NO LONGER LOAD THE COLLECTION, ONLY THE ONE CUSTOMER WHICH IS SHOPPING.
- NOTE! A NEW CUSTOMER OBJECT NEEDS TO BE CREATED EVERY TIME WE LOAD A CUSTOMER

```
'******************************************************************************
''' Calls CUSTOMER.LOAD method RETRIEVE CUSTOMER RECORD FROM database
''' whose ID is passed as argument. EXTRACT THE OBJECT'S DATA AND
''' found, else returns a Nothing.
'''</summary>
'''<summary>
Private Sub btnGet_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnGet.Click
    'Step A-Begins Exception handling.
    Try
        'Step 1-RE-CREATE THE Form-Level Object
        'OBJECT MUST BE NEW EVERY TIME BEFORE WE CAN LOAD()
        objCustomer = New BusinessObjectsDLL.clsCustomer

        'Step 2-Call OBJECT.LOAD() To load object with data from DB
        objCustomer.Load(txtIDNumber.Text.Trim)

        'Step 3-populate text boxes with customer data
        With objCustomer
            txtName.Text = .Name
            txtIDNumber.Text = .CustomerID
            txtBirthDate.Text = CStr(.BirthDate)
            txtAddress.Text = .Address
            txtPhone.Text = .Phone
            'Set total purchases
            txtTotalPurchases.Text = CStr(.TotalItemsPurchased)
        End With

        'Step 4-Enable the Items text box
        txtItems.Enabled = True

        'Step B-Traps for ApplicationException generated
        'by Customer.Load() method when record not found
        Catch objAppEx As ApplicationException
            MessageBox.Show("Customer Not Found! " & objAppEx.Message)

        'Step 5-Clear all controls
        txtName.Text = ""
        txtIDNumber.Text = ""
        txtBirthDate.Text = ""
        txtAddress.Text = ""
        txtPhone.Text = ""

        'Step C-Traps for Business Rule violations
        Catch objNSE As NotSupportedException
            MessageBox.Show("Business Rule violation! " & objNSE.Message)

        'Step D-Traps for general exceptions.
        Catch objE As Exception
            'Step E-Inform User
            MessageBox.Show(objE.Message)
        End Try
    End Sub
```
Step 4: SHOP_Click() event-handler – SHOP & SAVE

- Tells Customer to SHOP the number of ITEMS.
- TELLS CUSTOMER OBJECT TO SAVE ITSELF TO DATABASE

```
'******************************************************************************
'''<summary>
''' Calls customer object Shop() method to purchase items and clear text box. 
''' Also displays total purchases of customer and SAVES CUSTOMER TO DATABASE
'''</summary>
'''<param name="sender"></param>
'''<param name="e"></param>
'''<remarks></remarks>
Private Sub btnShop_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnShop.Click
  'Step A-Begins Exception handling.
  Try
    'Step 1-Call the Shop Method of the Object to shop and trigger event
    objCustomer.Shop(CInt(txtItems.Text.Trim))
    'Step 2-Clear Items textbox
    txtItems.Text = ""
    'Step 3-Set total purchases
    txtTotalPurchases.Text = CStr(objCustomer.TotalItemsPurchased)
    'Step 4-SAVE OBJECT
    objCustomer.Save()
    'Step B-Traps for Business Rule violations
    Catch objNSE As NotSupportedException
      MessageBox.Show("Business Rule violation! " & objNSE.Message)
    'Step C-Traps for general exceptions.
    Catch objE As Exception
      'Step D-Inform User
      MessageBox.Show(objE.Message)
  End Try
End Sub
```
**Step 5: The FORM_CLOSE() event-handler**

- IN THIS METHOD SIMPLY DESTROYS THE FORM-LEVEL CUSTOMER OBJECT.
- NO OTHER CODE REQUIRED, WE NO LONGER NEED TO SAVE THE COLLECTION BECAUSE WE DON’T NEED A COLLECTION TO GET OUR OBJECTS. THE OBJECTS LOAD AND SAVE THEMSELVES.

```
'***********************************************************************************
'''<summary>
'''Name:  Event-Handler Form_Close()  
'''Purpose:Destroys Form-level object pointer when form closes 
'''Saves Collection objects to file and clears the collection
'''</summary>
'''<param name="sender"></param>
'''<param name="e"></param>
'''<remarks></remarks>
Private Sub frmRetailManagement_FormClosed(ByVal sender As Object, ByVal e As System.Windows.Forms.FormClosedEventArgs) Handles Me.FormClosed
    'Step A-Begins Exception handling.
    Try
    'Step 1-Destroy Form-Level Objects
    objCustomer = Nothing
    'Step B-Traps for Business Rule violations
    Catch objNSE As NotSupportedException
        MessageBox.Show("Business Rule violation! " & objNSE.Message)
    'Step C-Traps for general exceptions.
    Catch objE As Exception
        'Step D-Inform User
        MessageBox.Show(objE.Message)
    End Try
End Sub
```

**Step 6: EXIT, PRINT & ONSHOPPING event-handler REQUIRE NO MODIFICATIONS**

- NO MODIFICATION REQUIRED FOR THE REMAINING EVENT HANDLERS.
Step 1: Compile and Build the project.

Step 2: Execute the application.

Step 3: RETAIL FORM

- Display Retail Form, SHOP for 50 ITEMS & AUTOMATICALLY SAVE CUSTOMER TO DATABASE:
The Customer table should reflect the Insertions & updates. Note the MS ACCESS database file smallbusinessapp.mdb is located in the BIN\DEBUG folder of the solution. I deleted the CustomerData.txt file since we don't need it any more.