Resource Software Solution

# .NET Programming Essentials

## Training Assignments

Contents

# Assignment: Building Database Application with ADO.NET

## Objectives:

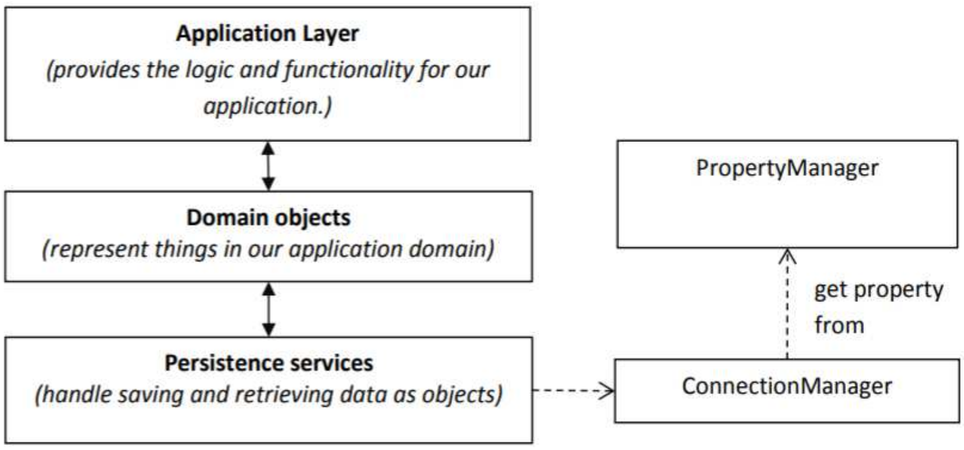
» Understand how to connect to database server and all components of ADO.NET

» How to work with SqlConnection, SqlCommand, SqlParameter, SqlDataReader, Stored procedures using INPUT/OUTPUT paramentters

» How to call and execute stored procedures with C# program.

## Product Architecture:

This application will be developed using following architecture:


Application Layer
(provides the logic and functionality for our application.)
Domain objects
(represent things in our application domain)
Persistence services
(handle saving and retrieving data as objects)
PropertyManager
get property from
ConnectionManager

» The domain layer contains objects and logic related to our application.

» The persistence layer contains data access objects (DAO) that provide services for accessing persistent data. DAO provide 4 basic services referred to as CRUD:

- **Creaate**: save new object data to the database.
- **Retrieve**: find object data in the database and recreate objects. There may be several methods for this service to enable different forms of object lookup.
- **Update**: update data for an object already saved to the database.
- **Delete**: delete object data from the database.

## <u>Specifications:</u>

Write a program that simulates the functions of the sales system.

Create a database named **SMS** for the sales system that has the following data tables:

> **Customer** (*<u>customer_id</u> int auto*, customer_name nvarchar(100))
>
> **Employee** (*<u>employee_id</u> int auto*, employee_name nvarchar(100), salary double, *supervisor_id int*)
>
> **Product** (*<u>product_id</u> int auto*, product_name nvarchar(100), product_price double)
>
> **Orders** (*<u>order_id</u> int auto*, order_date datetime, *customer_id int*, *employee_id int*, total double)
>
> **LineItem** (<u>*order_id, product_id*</u>, quantity int, price double)

*Notice that, all of the tables belong the same "dbo" schema.*

## <u>Technical Requirements:</u>

Create a project named **NPL.SMS** will have the following namespaces:

Inside namespace ***R2S.Training.Entities*** contains the following classes:

The **LineItem** class:

- Four private instance variables: *orderId* (int), *productId* (int), *quantity* (int), *price* (double)
- Default constructor and the constructor has 4 parameters to initialize value of attributes.
- Getter and setter methods.

The **Customer** class:

- Two private instance variables: *customerId* (int), *customerName* (String)
- Default constructor and the constructor has 2 parameters to initialize value of attributes.
- Getter and setter methods.

The **Employee** class:

- Four private instance variables: *employeeId* (int), *employeeName* (String), *salary* (double), *spvrId* (int)
- Default constructor and the constructor has 4 parameters to initialize value of attributes.
- Getter and setter methods.

The **Product** class:

- Three private instance variables: *productId* (int), *productName* (String), *productPrice* (double)
- Default constructor and the constructor has 3 parameters to initialize value of attributes.
- Getter and setter methods.

The **Order** class:

- Five private instance variables: *orderId* (int), *orderDate* (Date), *customerId* (int), *employeeId* (int), *total* (double)
- Default constructor and the constructor has 5 parameters to initialize value of attributes.
- Getter and setter methods.

Each entities will have its own interfaces and classe which are allocated inside ***R2S.Training.Dao****,* follow the following pattern: for interfaces (i.e **OrderDAO, ProductDAO**), for impl class (I**OrderDAO, IProductDAO**).

## Functional Requirements

1) List all customers consist of *customer id*, *customer name* in the database, returns a list with all customers in the order table (`List<Customer> GetAllCustomer()` method)
2) List all orders consist of order id, order date, customer id, employee id, total for a customer, returns a list with all the orders for a given customer id (`List<Order> GetAllOrdersByCustomerId(int customerId)` method)
3) List all lineitems for an order, returns a list with all line items for a given order id (`List<LineItem> GetAllItemsByOrderId(int orderId)` method)
4) Compute order total, returns a list with a row containing the computed order total from the line items (named as **total_price**) for a given order id. You must use an UDF (`Double ComputeOrderTotal(int orderId)` method)
5) Add a customer into the database, you must use a Stored Procedure (`bool AddCustomer(Customer customer)` method)
6) Delete a customer from the database, make sure to also delete Orders and LineItem for the deleted customer. You must use a Stored Procedure (`bool DeleteCustomer(int customerId)` method).
7) Update a customer in the database, you must use a Stored Procedure (`bool UpdateCustomer(Customer customer)` method).
8) Create an order into the database (`bool AddOrder(Order order)` method).
9) Create a lineitem into the database (`bool AddLineItem(LineItem item)` method).
10) Update an order total into the database (`bool UpdateOrderTotal(int orderId)` method).

## User Interface Requirements

Create a **SaleManagement** class Inside namespace ***R2S.Training.Main*** that contains a Main() method to test the above functional methods.

**-- THE END --**