**Peer Evaluation for Lab 5 – Chapter 20**

|  |  |
| --- | --- |
| Your name: (Your lab is the one being evaluated) | **Steven Wilson** |
| Name(s) of peer evaluator(s) | **Patrick Behrens** |
| Date: | **5/20/2019** |

Instructions  
You should have already completed Lab 5. After you and a peer have evaluated your work, you will submit this evaluation along with screen shots and source code indicated in moodle. You may make corrections to your work as a result of the evaluation.

|  |  |
| --- | --- |
| ***In Class Exercises – 20-1*** | |
| Completed Exercise?   * Customer Maintenance application that allows you to search for customers, add a customer, modify a customer and delete a customer?   + MMABooksDB class has been created? Contains the method GetConnection?   + CustomerDB class has been created? Contains GetCustomer, UpdateCustomer, AddCustomer and DeleteCustomer methods?   + StateDB class has been created? Contains GetStates method? * Errors are handled appropriately? What kinds of errors are handled and how? * Screen shot of application running is included? * Source code includes 3 DB classes? | **Yes.**  **Yes.**  **Yes.**  **Yes.**  **Yes.**  **Yes.**  **Yes.**  **Yes.**  **SqlException errors are handled with a try/catch block.**  **Yes.**  **Yes.** |

|  |  |
| --- | --- |
| ***Product and ProductDB classes*** | |
| Completed Exercise?   * Product class has been created? * ProductDB class has been created? Contains GetProduct, UpdateProduct, AddProduct and DeleteProduct methods? * Each method is tested? * Errors are handled appropriately? What kinds of errors are handled and how? * Screen shot of application running is included? * Source code for the DB class is included? | **Yes.**  **Yes.**  **Yes.**  **Using a custom UI, yes.**  **Yes.**  **SqlException errors are handled with a try/catch block.**  **Yes.**  **Yes.** |

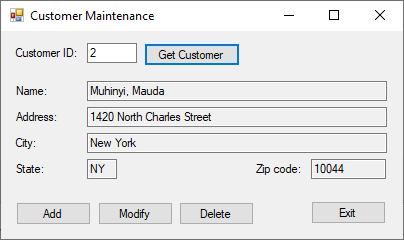
General comments and notes:

**NOTE: I took this into consideration but I really don’t know what I would comment that wouldn’t just be me repeating myself.**

**Everything seems to work. CRUD operations can be performed. Only thing could be some more comments.**

Screen Shots and Source Code

**CustomerMaintenance:**



**MMABooksDB:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.SqlClient;

namespace CustomerMaintenance

{

public class MMABooksDB

{

public static SqlConnection GetConnection()

{

string connectionString = "Data Source=DESKTOP-OCKQVIP;Initial Catalog=MMABooks;Integrated Security=True";

SqlConnection connection = new SqlConnection(connectionString);

return connection;

}

}

}

**CustomerDB:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data;

using System.Data.SqlClient;

namespace CustomerMaintenance

{

class CustomerDB

{

public static Customer GetCustomer(int customerID)

{

// Creates a connection object.

SqlConnection connection = MMABooksDB.GetConnection();

// Our SELECT statement, parameterized.

string statement = "SELECT \* FROM Customers WHERE CustomerID=@customerID;";

// The command object we're going to modify to access the database.

SqlCommand command = new SqlCommand(statement, connection);

// Adds the customerID parameter shown above.

command.Parameters.AddWithValue("@customerID", customerID);

try

{

// Opens the connection to the SQL server.

connection.Open();

// Creates the reader object so we can get the information from the SQL server.

// Also executes the statement above.

SqlDataReader reader = command.ExecuteReader(CommandBehavior.SingleRow);

if(reader.Read())

{

// Creates a new temporary customer object.

Customer customer = new Customer();

// Sets the variables of our customer object.

customer.CustomerID = (int)reader["CustomerID"];

customer.Name = (string)reader["Name"];

customer.Address = (string)reader["Address"];

customer.City = (string)reader["City"];

customer.State = (string)reader["State"];

customer.ZipCode = (string)reader["ZipCode"];

// Returns our customer object.

return customer;

}

else

{

// No object was found.

return null;

}

}

catch(SqlException ex)

{

throw ex;

}

finally

{

connection.Close();

}

}

public static bool UpdateCustomer(Customer oldCust, Customer newCust)

{

SqlConnection connection = MMABooksDB.GetConnection();

// This is a 'verbatim' string literal, which appears to be quite

// similar to a template string in JavaScript. I like it.

string statement = @"

UPDATE Customers SET

Name = @newName,

Address = @newAddress,

City = @newCity,

State = @newState,

ZipCode = @newZipCode

WHERE CustomerID = @oldCustomerID

AND Name = @oldName

AND Address = @oldAddress

AND City = @oldCity

AND State = @oldState

AND ZipCode = @oldZipCode;

";

SqlCommand command = new SqlCommand(statement, connection);

command.Parameters.AddWithValue("@newName", newCust.Name);

command.Parameters.AddWithValue("@newAddress", newCust.Address);

command.Parameters.AddWithValue("@newCity", newCust.City);

command.Parameters.AddWithValue("@newState", newCust.State);

command.Parameters.AddWithValue("@newZipCode", newCust.ZipCode);

command.Parameters.AddWithValue("@oldCustomerID", oldCust.CustomerID);

command.Parameters.AddWithValue("@oldName", oldCust.Name);

command.Parameters.AddWithValue("@oldAddress", oldCust.Address);

command.Parameters.AddWithValue("@oldCity", oldCust.City);

command.Parameters.AddWithValue("@oldState", oldCust.State);

command.Parameters.AddWithValue("@oldZipCode", oldCust.ZipCode);

try

{

connection.Open();

int rowCount = command.ExecuteNonQuery();

if (rowCount > 0) return true;

else return false;

}

catch(SqlException ex)

{

throw ex;

}

finally

{

connection.Close();

}

}

public static int AddCustomer(Customer newCust)

{

SqlConnection connection = MMABooksDB.GetConnection();

// SQL query that inserts a new customer with the given parameters. I think

// it's great that we're learning about parameterized queries first, though

// I do hope the book teaches WHY we're doing this. I see a lot of SQL injection

// attacks waiting to happen when browsing GitHub projects. It's a shame.

string statement = @"

INSERT INTO Customers (Name, Address, City, State, ZipCode)

VALUES (@newName, @newAddress, @newCity, @newState, @newZipCode);

";

SqlCommand command = new SqlCommand(statement, connection);

command.Parameters.AddWithValue("@newName", newCust.Name);

command.Parameters.AddWithValue("@newAddress", newCust.Address);

command.Parameters.AddWithValue("@newCity", newCust.City);

command.Parameters.AddWithValue("@newState", newCust.State);

command.Parameters.AddWithValue("@newZipCode", newCust.ZipCode);

try

{

connection.Open();

command.ExecuteNonQuery();

// Acquires the new customer's ID.

string getCustomerID = "SELECT IDENT\_CURRENT('Customers') FROM Customers;";

SqlCommand getCustomerIDCommand = new SqlCommand(getCustomerID, connection);

int CustomerID = Convert.ToInt32(getCustomerIDCommand.ExecuteScalar());

return CustomerID;

}

catch(SqlException ex)

{

throw ex;

}

finally

{

connection.Close();

}

}

public static bool DeleteCustomer(Customer toDelete)

{

SqlConnection connection = MMABooksDB.GetConnection();

// Should I be placing the semicolon at the end of each

// query? I always did it in my node web servers and I

// still don't know if it truly matters.

string statement = @"

DELETE FROM Customers

WHERE CustomerID = @customerID

AND Name = @name

AND Address = @address

AND City = @city

AND State = @state

AND ZipCode = @zipCode;

";

SqlCommand command = new SqlCommand(statement, connection);

command.Parameters.AddWithValue("@customerID", toDelete.CustomerID);

command.Parameters.AddWithValue("@name", toDelete.Name);

command.Parameters.AddWithValue("@address", toDelete.Address);

command.Parameters.AddWithValue("@city", toDelete.City);

command.Parameters.AddWithValue("@state", toDelete.State);

command.Parameters.AddWithValue("@zipCode", toDelete.ZipCode);

try

{

connection.Open();

int count = command.ExecuteNonQuery();

if (count > 0) return true;

else return false;

}

catch(SqlException ex)

{

throw ex;

}

finally

{

connection.Close();

}

}

}

}

**StateDB:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data;

using System.Data.SqlClient;

namespace CustomerMaintenance

{

class StateDB

{

public static List<State> GetStates()

{

// The list we're going to return.

List<State> states = new List<State>();

// Creates a new connection to the database.

SqlConnection connection = MMABooksDB.GetConnection();

// Selects the StateCode and StateName from the States table.

string statement = "SELECT \* FROM States ORDER BY StateName;";

SqlCommand command = new SqlCommand(statement, connection);

try

{

connection.Open();

SqlDataReader reader = command.ExecuteReader();

if (reader.Read())

{

while (reader.Read())

{

State state = new State();

state.StateCode = (string) reader["StateCode"];

state.StateName = (string) reader["StateName"];

states.Add(state);

}

reader.Close();

}

else

{

return null;

}

}

catch(SqlException ex)

{

throw ex;

}

finally

{

connection.Close();

}

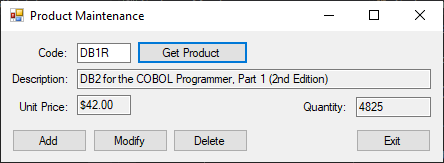
return states;

}

}

}

**ProductMaintenance:**



**Product:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ProductMaintenance

{

public class Product

{

private string productCode;

private string description;

private decimal unitPrice;

private int onHandQuantity;

public string ProductCode

{

get => productCode;

set => productCode = value;

}

public string Description

{

get => description;

set => description = value;

}

public decimal UnitPrice

{

get => unitPrice;

set => unitPrice = value;

}

public int OnHandQuantity

{

get => onHandQuantity;

set => onHandQuantity = value;

}

}

}

**ProductDB:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data;

using System.Data.SqlClient;

using CustomerMaintenance;

namespace ProductMaintenance

{

public class ProductDB

{

public static Product GetProduct(string productCode)

{

// I made the MMABooksDB class public for this. I'm

// not sure if that was a good idea or not.

SqlConnection connection = MMABooksDB.GetConnection();

string query = @"

SELECT \*

FROM Products

WHERE ProductCode = @productCode;

";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@productCode", productCode);

try

{

connection.Open();

SqlDataReader reader = command.ExecuteReader(CommandBehavior.SingleRow);

if (reader.Read())

{

Product product = new Product();

product.ProductCode = (string)reader["ProductCode"];

product.Description = (string)reader["Description"];

product.UnitPrice = (decimal)reader["UnitPrice"];

product.OnHandQuantity = (int)reader["OnHandQuantity"];

return product;

}

else

{

return null;

}

}

catch(SqlException ex)

{

throw ex;

}

finally

{

connection.Close();

}

}

public static bool UpdateProduct(Product oldProduct, Product newProduct)

{

SqlConnection connection = MMABooksDB.GetConnection();

string query = @"

UPDATE Products SET

Description = @newDescription,

UnitPrice = @newUnitPrice,

OnHandQuantity = @newQuantity

WHERE ProductCode = @productCode

AND Description = @oldDescription

AND UnitPrice = @oldUnitPrice

AND OnHandQuantity = @oldQuantity;

";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@newDescription", newProduct.Description);

command.Parameters.AddWithValue("@newUnitPrice", newProduct.UnitPrice);

command.Parameters.AddWithValue("@newQuantity", newProduct.OnHandQuantity);

command.Parameters.AddWithValue("@productCode", oldProduct.ProductCode);

command.Parameters.AddWithValue("@oldDescription", oldProduct.Description);

command.Parameters.AddWithValue("@oldUnitPrice", oldProduct.UnitPrice);

command.Parameters.AddWithValue("@oldQuantity", oldProduct.OnHandQuantity);

try

{

connection.Open();

int rowCount = command.ExecuteNonQuery();

if (rowCount > 0) return true;

else return false;

}

catch(SqlException ex)

{

throw ex;

}

finally

{

connection.Close();

}

}

public static bool AddProduct(Product newProduct)

{

SqlConnection connection = MMABooksDB.GetConnection();

string actionQuery = @"

INSERT INTO Products (ProductCode, Description, UnitPrice, OnHandQuantity)

VALUES (@newProductCode, @newDescription, @newUnitPrice, @newQuantity);

";

SqlCommand command = new SqlCommand(actionQuery, connection);

command.Parameters.AddWithValue("@newProductCode", newProduct.ProductCode);

command.Parameters.AddWithValue("@newDescription", newProduct.Description);

command.Parameters.AddWithValue("@newUnitPrice", newProduct.UnitPrice);

command.Parameters.AddWithValue("@newQuantity", newProduct.OnHandQuantity);

try

{

connection.Open();

command.ExecuteNonQuery();

// Returns whether or not the product was actually added.

string getProduct = "SELECT \* FROM Products WHERE ProductCode = @newProductCode";

SqlCommand getProductRows = new SqlCommand(getProduct, connection);

getProductRows.Parameters.AddWithValue("@newProductCode", newProduct.ProductCode);

int rows = getProductRows.ExecuteNonQuery();

if (rows > 0) return true;

else return false;

}

catch(SqlException ex)

{

throw ex;

}

finally

{

connection.Close();

}

}

public static bool DeleteProduct(Product toDelete)

{

SqlConnection connection = MMABooksDB.GetConnection();

string query = @"

DELETE FROM Products

WHERE ProductCode = @productCode

AND Description = @description

AND UnitPrice = @unitPrice

AND OnHandQuantity = @quantity;

";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@productCode", toDelete.ProductCode);

command.Parameters.AddWithValue("@description", toDelete.Description);

command.Parameters.AddWithValue("@unitPrice", toDelete.UnitPrice);

command.Parameters.AddWithValue("@quantity", toDelete.OnHandQuantity);

try

{

connection.Open();

int rows = command.ExecuteNonQuery();

if (rows > 0) return true;

else return false;

}

catch (SqlException ex)

{

throw ex;

}

finally

{

connection.Close();

}

}

}

}

**frmProductMaintenance:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using CustomerMaintenance;

namespace ProductMaintenance

{

public partial class frmProductMaintenance : Form

{

private Product product;

public frmProductMaintenance()

{

InitializeComponent();

}

// Unfortunately, I can't make the existing code any better,

// so everything here is basically just the code from

// frmCustomerMaintenance.

private void GetProductButton\_Click(object sender, EventArgs e)

{

if(Validator.IsPresent(ProductCodeTextBox))

{

string productCode = ProductCodeTextBox.Text;

GetProduct(productCode);

if (product == null)

{

MessageBox.Show(@"

No product was found with this product code.

Try again, please.

", "Product not found");

ClearControls();

}

else DisplayProduct();

}

}

private void GetProduct(string productCode)

{

try

{

product = ProductDB.GetProduct(productCode);

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void ClearControls()

{

ProductCodeTextBox.Text = "";

DescriptionTextBox.Text = "";

UnitPriceTextBox.Text = "";

OnHandQuantityTextBox.Text = "";

ModifyButton.Enabled = false;

DeleteButton.Enabled = false;

ProductCodeTextBox.Select();

}

private void DisplayProduct()

{

DescriptionTextBox.Text = product.Description;

UnitPriceTextBox.Text = product.UnitPrice.ToString("c");

OnHandQuantityTextBox.Text = product.OnHandQuantity.ToString();

ModifyButton.Enabled = true;

DeleteButton.Enabled = true;

}

private void AddButton\_Click(object sender, EventArgs e)

{

frmAddModifyProduct addProductForm = new frmAddModifyProduct();

addProductForm.addProduct = true;

DialogResult result = addProductForm.ShowDialog();

if(result == DialogResult.OK)

{

product = addProductForm.product;

ProductCodeTextBox.Text = product.ProductCode.ToString();

DisplayProduct();

}

}

private void ModifyButton\_Click(object sender, EventArgs e)

{

frmAddModifyProduct modifyProductForm = new frmAddModifyProduct();

modifyProductForm.addProduct = false;

modifyProductForm.product = product;

DialogResult result = modifyProductForm.ShowDialog();

if(result == DialogResult.OK)

{

product = modifyProductForm.product;

DisplayProduct();

}

else if(result == DialogResult.Retry)

{

GetProduct(product.ProductCode);

if (product != null)

DisplayProduct();

else

ClearControls();

}

}

private void DeleteButton\_Click(object sender, EventArgs e)

{

DialogResult result = MessageBox.Show("Delete Product " + product.ProductCode + "?", "Confirm Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if(result == DialogResult.Yes)

{

try

{

if (!ProductDB.DeleteProduct(product))

{

MessageBox.Show("Another user has updated or deleted that product.", "Database Error");

GetProduct(product.ProductCode);

if (product != null)

DisplayProduct();

else

ClearControls();

}

else ClearControls();

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

private void ExitButton\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

**frmAddModifyProduct:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using CustomerMaintenance;

namespace ProductMaintenance

{

public partial class frmAddModifyProduct : Form

{

public bool addProduct;

public Product product;

public frmAddModifyProduct()

{

InitializeComponent();

if (addProduct)

this.Text = "Add Product";

else

this.Text = "Modify Product";

}

private void AcceptButton\_Click(object sender, EventArgs e)

{

if (IsValidData())

{

if (addProduct)

{

product = new Product();

PutProductData(product);

if (ProductDB.AddProduct(product))

this.DialogResult = DialogResult.OK;

else

this.DialogResult = DialogResult.Abort;

}

else

{

Product newProduct = new Product();

newProduct.ProductCode = product.ProductCode;

this.PutProductData(newProduct);

try

{

if(!ProductDB.UpdateProduct(product, newProduct))

{

MessageBox.Show("Another user has updated or deleted that customer.", "Database Error");

this.DialogResult = DialogResult.Retry;

}

else

{

product = newProduct;

this.DialogResult = DialogResult.OK;

}

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

}

private bool IsValidData()

{

return

Validator.IsPresent(ProductCodeTextBox) &&

Validator.IsPresent(DescriptionTextBox) &&

Validator.IsPresent(UnitPriceTextBox) &&

Validator.IsPresent(OnHandQuantityTextBox);

}

private void PutProductData(Product newProduct)

{

product.ProductCode = ProductCodeTextBox.Text;

product.Description = DescriptionTextBox.Text;

try

{

product.UnitPrice = Decimal.Parse(UnitPriceTextBox.Text);

}

catch

{

MessageBox.Show("Invalid input for UnitPrice.", "Input Error");

}

try

{

product.OnHandQuantity = int.Parse(OnHandQuantityTextBox.Text);

}

catch

{

MessageBox.Show("Invalid input for on hand quantity.", "Input Error");

}

}

}

}