Amanda Akins – Lab 6

So, again, I don’t know why the peer evaluations are downloading as Excel spreadsheets… Here is what I’ve got so far. I know it’s not all correct but I will work over the summer as discussed and stuff 😊. Thank you so much for everything! I will come bug you next term.

**Product.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ToolsCSharp;

using EventPropsClasses;

using ProductDB = EventDBClasses.CustomerSQLDB;

using System.Data;

using EventPropsClassses;

namespace EventClasses

{

public class Product : BaseBusiness

{

#region SetUpStuff

/// <summary>

///

/// </summary>

protected override void SetDefaultProperties()

{

}

/// <summary>

/// Sets required fields for a record.

/// </summary>

protected override void SetRequiredRules()

{

mRules.RuleBroken("ProductID", true);

mRules.RuleBroken("ProductCode", true);

mRules.RuleBroken("Description", true);

}

/// <summary>

/// Instantiates mProps and mOldProps as new Props objects.

/// Instantiates mbdReadable and mdbWriteable as new DB objects.

/// </summary>

protected override void SetUp()

{

mProps = new EventProps();

mOldProps = new EventProps();

if (this.mConnectionString == "")

{

mdbReadable = new ProductDB();

mdbWriteable = new ProductDB();

}

else

{

mdbReadable = new ProductDB(this.mConnectionString);

mdbWriteable = new ProductDB(this.mConnectionString);

}

}

#endregion

#region constructors

/// <summary>

/// Default constructor - does nothing.

/// </summary>

public Product() : base()

{

}

/// <summary>

/// One arg constructor.

/// Calls methods SetUp(), SetRequiredRules(),

/// SetDefaultProperties() and BaseBusiness one arg constructor.

/// </summary>

/// <param name="cnString">DB connection string.

/// This value is passed to the one arg BaseBusiness constructor,

/// which assigns the it to the protected member mConnectionString.</param>

public Product(string cnString)

: base(cnString)

{

}

/// <summary>

/// Two arg constructor.

/// Calls methods SetUp() and Load().

/// </summary>

/// <param name="key">ID number of a record in the database.

/// Sent as an arg to Load() to set values of record to properties of an

/// object.</param>

/// <param name="cnString">DB connection string.

/// This value is passed to the one arg BaseBusiness constructor,

/// which assigns the it to the protected member mConnectionString.</param>

public Product(int key, string cnString)

: base(key, cnString)

{

}

public Product(int key)

: base(key)

{

}

// \*\*\* I added these 2 so that I could create a

// business object from a properties object

// I added the new constructors to the base class

public Product(ProductProps props)

: base(props)

{

}

public Product(ProductProps props, string cnString)

: base(props, cnString)

{

}

#endregion

#region properties

/// <summary>

/// Read-only ID property.

/// </summary>

public int ID

{

get

{

return ((ProductProps)mProps).ID;

}

}

/// <summary>

/// Read/Write property.

/// </summary>

public int ProductID

{

get

{

return ((ProductProps)mProps).ProductID;

}

set

{

if (!(value == ((ProductProps)mProps).ProductID))

{

if (value > 0)

{

mRules.RuleBroken("ProductID", false);

((EventProps)mProps).ProductID = value;

mIsDirty = true;

}

else

{

throw new ArgumentOutOfRangeException("ProductID must be a positive number.");

}

}

}

}

/// <summary>

/// Read/Write property.

/// </summary>

/// <exception cref="ArgumentException">

///

/// </exception>

public int ProductCode

{

get

{

return ((ProductProps)mProps).code;

}

set

{

if (!(value == ((ProductProps)mProps).code))

{

if (value > 0)

{

mRules.RuleBroken("ProductCode", false);

((ProductProps)mProps).ProductCode = value;

mIsDirty = true;

}

else

{

throw new ArgumentOutOfRangeException("Product Code must be a positive number.");

}

}

}

}

/// <summary>

/// Read/Write property.

/// </summary>

/// <exception cref="ArgumentException">

///

/// </exception>

public string Description

{

get

{

return ((ProductProps)mProps).description;

}

set

{

if (!(value == ((ProductProps)mProps).description))

{

if (value.Length >= 1 && value.Length <= 2000)

{

mRules.RuleBroken("Description", false);

((ProductProps)mProps).description = value;

mIsDirty = true;

}

else

{

throw new ArgumentException("Description must be between 1 and 2000 characters");

}

}

}

}

#endregion

#region others

/// <summary>

/// Retrieves a list of Events.

/// </summary>

///

// \*\*\* I had to change this

public static List<Product> GetList(string cnString)

{

ProductDB db = new ProductDB(cnString);

List<Product> products = new List<Product>();

List<ProductProps> props = new List<ProductProps>();

// \*\*\* methods in the textdb and sqldb classes don't match

// Ideally, I should go back and fix the IReadDB interface!

props = (List<ProductProps>)db.RetrieveAll(props.GetType());

foreach (ProductProps prop in props)

{

// \*\*\* creates the business object from the props objet

Product p = new Product(prop, cnString);

Product.Add(p);

}

return products;

}

// \*\*\* this o

public override object GetList()

{

List<Product> products = new List<Product>();

List<ProductProps> props = new List<ProductProps>();

props = (List<ProductProps>)mdbReadable.RetrieveAll(props.GetType());

foreach (ProductProps prop in props)

{

Product p = new Product(prop, this.mConnectionString);

products.Add(p);

}

return products;

}

// \*\*\* this is new

public static DataTable GetTable(string cnString)

{

ProductDB db = new ProductDB(cnString);

return db.RetrieveTable();

}

public static DataTable GetTable()

{

ProductDB db = new ProductDB();

return db.RetrieveTable();

}

/// <summary>

/// Deletes the customer identified by the id.

/// </summary>

public static void Delete(int id)

{

ProductDB db = new ProductDB();

db.Delete(id);

}

public static void Delete(int id, string cnString)

{

ProductDB db = new ProductDB(cnString);

db.Delete(id);

}

#endregion

}

}

**Customer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ToolsCSharp;

using EventPropsClasses;

using CustomerDB = EventDBClasses.CustomerSQLDB;

using System.Data;

using EventPropsClassses;

namespace EventClasses

{

public class Customer : BaseBusiness

{

#region SetUpStuff

/// <summary>

///

/// </summary>

protected override void SetDefaultProperties()

{

}

/// <summary>

/// Sets required fields for a record.

/// </summary>

protected override void SetRequiredRules()

{

mRules.RuleBroken("CustomerID", true);

mRules.RuleBroken("Name", true);

mRules.RuleBroken("Address", true);

}

/// <summary>

/// Instantiates mProps and mOldProps as new Props objects.

/// Instantiates mbdReadable and mdbWriteable as new DB objects.

/// </summary>

protected override void SetUp()

{

mProps = new CustomerProps();

mOldProps = new CustomerProps();

if (this.mConnectionString == "")

{

mdbReadable = new CustomerDB();

mdbWriteable = new CustomerDB();

}

else

{

mdbReadable = new CustomerDB(this.mConnectionString);

mdbWriteable = new CustomerDB(this.mConnectionString);

}

}

#endregion

#region constructors

/// <summary>

/// Default constructor - does nothing.

/// </summary>

public Customer()

: base()

{

}

/// <summary>

/// One arg constructor.

/// Calls methods SetUp(), SetRequiredRules(),

/// SetDefaultProperties() and BaseBusiness one arg constructor.

/// </summary>

/// <param name="cnString">DB connection string.

/// This value is passed to the one arg BaseBusiness constructor,

/// which assigns the it to the protected member mConnectionString.</param>

public Customer(string cnString)

: base(cnString)

{

}

/// <summary>

/// Two arg constructor.

/// Calls methods SetUp() and Load().

/// </summary>

/// <param name="key">ID number of a record in the database.

/// Sent as an arg to Load() to set values of record to properties of an

/// object.</param>

/// <param name="cnString">DB connection string.

/// This value is passed to the one arg BaseBusiness constructor,

/// which assigns the it to the protected member mConnectionString.</param>

public Customer(int key, string cnString)

: base(key, cnString)

{

}

public Customer(int key)

: base(key)

{

}

// \*\*\* I added these 2 so that I could create a

// business object from a properties object

// I added the new constructors to the base class

public Customer(CustomerProps props)

: base(props)

{

}

public Customer(CustomerProps props, string cnString)

: base(props, cnString)

{

}

#endregion

#region properties

/// <summary>

/// Read-only ID property.

/// </summary>

public int ID

{

get

{

return ((CustomerProps)mProps).ID;

}

}

/// <summary>

/// Read/Write property.

/// </summary>

public int CustomerID

{

get

{

return ((CustomerProps)mProps).CustomerID;

}

set

{

if (!(value == ((CustomerProps)mProps).CustomerID))

{

if (value > 0)

{

mRules.RuleBroken("CustomerID", false);

((CustomerProps)mProps).CustomerID = value;

mIsDirty = true;

}

else

{

throw new ArgumentOutOfRangeException("CustomerID must be a positive number.");

}

}

}

}

/// <summary>

/// Read/Write property.

/// </summary>

/// <exception cref="ArgumentException">

///

/// </exception>

public string Name

{

get

{

return ((CustomerProps)mProps).name;

}

set

{

if (!(value == ((CustomerProps)mProps).name))

{

if (value.Length >= 1 && value.Length <= 50)

{

mRules.RuleBroken("Name", false);

((CustomerProps)mProps).name = value;

mIsDirty = true;

}

else

{

throw new ArgumentException("Name must be between 1 and 50 characters");

}

}

}

}

/// <summary>

/// Read/Write property.

/// </summary>

/// <exception cref="ArgumentException">

///

/// </exception>

public string Address

{

get

{

return ((CustomerProps)mProps).address;

}

set

{

if (!(value == ((CustomerProps)mProps).address))

{

if (value.Length >= 1 && value.Length <= 2000)

{

mRules.RuleBroken("Address", false);

((CustomerProps)mProps).address = value;

mIsDirty = true;

}

else

{

throw new ArgumentException("Address must be between 1 and 2000 characters");

}

}

}

}

#endregion

#region others

/// <summary>

/// Retrieves a list of Events.

/// </summary>

///

// \*\*\* I had to change this

public static List<Customer> GetList(string cnString)

{

CustomerDB db = new CustomerDB(cnString);

List<Customer> customers = new List<Customer>();

List<CustomerProps> props = new List<CustomerProps>();

// \*\*\* methods in the textdb and sqldb classes don't match

// Ideally, I should go back and fix the IReadDB interface!

props = (List<CustomerProps>)db.RetrieveAll(props.GetType());

foreach (CustomerProps prop in props)

{

// \*\*\* creates the business object from the props objet

Customer c = new Customer(prop, cnString);

customers.Add(c);

}

return customers;

}

// \*\*\* this o

public override object GetList()

{

List<Customer> customers = new List<Customer>();

List<CustomerProps> props = new List<CustomerProps>();

props = (List<CustomerProps>)mdbReadable.RetrieveAll(props.GetType());

foreach (CustomerProps prop in props)

{

Customer c = new Customer(prop, this.mConnectionString);

customers.Add(c);

}

return customers;

}

// \*\*\* this is new

public static DataTable GetTable(string cnString)

{

CustomerDB db = new CustomerDB(cnString);

return db.RetrieveTable();

}

public static DataTable GetTable()

{

CustomerDB db = new CustomerDB();

return db.RetrieveTable();

}

/// <summary>

/// Deletes the customer identified by the id.

/// </summary>

public static void Delete(int id)

{

CustomerDB db = new CustomerDB();

db.Delete(id);

}

public static void Delete(int id, string cnString)

{

CustomerDB db = new CustomerDB(cnString);

db.Delete(id);

}

#endregion

}

}

**CustomerSQLDB.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using EventPropsClasses;

using ToolsCSharp;

using System.Data;

using System.Data.SqlClient;

// \*\*\* I use an "alias" for the ado.net classes throughout my code

// When I switch to an oracle database, I ONLY have to change the actual classes here

using DBBase = ToolsCSharp.BaseSQLDB;

using DBConnection = System.Data.SqlClient.SqlConnection;

using DBCommand = System.Data.SqlClient.SqlCommand;

using DBParameter = System.Data.SqlClient.SqlParameter;

using DBDataReader = System.Data.SqlClient.SqlDataReader;

using DBDataAdapter = System.Data.SqlClient.SqlDataAdapter;

using EventPropsClassses;

namespace EventDBClasses

{

//DBBase is the abstract base class with a bunch of methods that can help

public class CustomerSQLDB : DBBase, IReadDB, IWriteDB

{

#region Constructors

public CustomerSQLDB() : base() { }

public CustomerSQLDB(string cnString) : base(cnString) { }

public CustomerSQLDB(DBConnection cn) : base(cn) { }

#endregion

public IBaseProps Create(IBaseProps c)

{

int rowsAffected = 0;

CustomerProps props = (CustomerProps)c;

DBCommand command = new DBCommand();

command.CommandText = "usp\_CustomerCreate";

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@Name", SqlDbType.Int);

command.Parameters.Add("@Address", SqlDbType.Char);

command.Parameters.Add("@City", SqlDbType.VarChar);

command.Parameters.Add("@State", SqlDbType.Money);

command.Parameters.Add("@ZipCode", SqlDbType.Int);

command.Parameters[0].Direction = ParameterDirection.Output;

command.Parameters["@Name"].Value = props.Name;

command.Parameters["@Address"].Value = props.Address;

command.Parameters["@City"].Value = props.City;

command.Parameters["@State"].Value = props.State;

command.Parameters["@ZipCode"].Value = props.ZipCode;

try

{

rowsAffected = RunNonQueryProcedure(command); //how many records were affected when I did this?

if (rowsAffected == 1)

{

props.ID = (int)command.Parameters[0].Value;

props.ConcurrencyID = 1;

return props;

}

else

throw new Exception("Unable to insert record. " + props.ToString());

}

catch (Exception e)

{

// log this error

throw;

}

finally

{

if (mConnection.State == ConnectionState.Open)

mConnection.Close();

}

}

public bool Update(IBaseProps c)

{

int rowsAffected = 0;

CustomerProps props = (CustomerProps)c;

DBCommand command = new DBCommand();

command.CommandText = "usp\_CustomerUpdate";

command.CommandType = CommandType.StoredProcedure;

//command.Parameters.Add("@CustomerID", SqlDbType.Int);

command.Parameters.Add("@Name", SqlDbType.VarChar);

command.Parameters.Add("@Address", SqlDbType.VarChar);

command.Parameters.Add("@City", SqlDbType.VarChar);

command.Parameters.Add("@State", SqlDbType.Char);

command.Parameters.Add("@ZipCode", SqlDbType.Char);

//command.Parameters.Add("@ConcurrencyID", SqlDbType.Int);

command.Parameters["@Name"].Value = props.Name;

command.Parameters["@Address"].Value = props.Address;

command.Parameters["@City"].Value = props.City;

command.Parameters["@State"].Value = props.State;

command.Parameters["@ZipCode"].Value = props.ZipCode;

//command.Parameters["@ConcurrencyID"].Value = props.ConcurrencyID;

try

{

rowsAffected = RunNonQueryProcedure(command);

if (rowsAffected == 1)

{

props.ConcurrencyID++;

return true;

}

else

{

string message = "Record cannot be updated. It has been edited by another user.";

throw new Exception(message);

}

}

catch (Exception e)

{

// log this exception

throw;

}

finally

{

if (mConnection.State == ConnectionState.Open)

mConnection.Close();

}

} // end of Update()

public bool Delete(IBaseProps c)

{

CustomerProps props = (CustomerProps)c;

int rowsAffected = 0;

DBCommand command = new DBCommand();

command.CommandText = "usp\_CustomerDelete";

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@CustomerID", SqlDbType.Int);

command.Parameters.Add("@ConcurrencyID", SqlDbType.Int);

command.Parameters["@Customer ID"].Value = props.ID;

command.Parameters["@ConcurrencyID"].Value = props.ConcurrencyID;

try

{

rowsAffected = RunNonQueryProcedure(command);

if (rowsAffected == 1)

{

return true;

}

else

{

string message = "Record cannot be deleted. It has been edited by another user.";

throw new Exception(message);

}

}

catch (Exception e)

{

// log this exception

throw;

}

finally

{

if (mConnection.State == ConnectionState.Open)

mConnection.Close();

}

} // end of Delete()

public IBaseProps Retrieve(object key)

{

DBDataReader data = null;

ProductProps props = new ProductProps();

DBCommand command = new DBCommand();

command.CommandText = "usp\_CustomerSelect";

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@CustomerID", SqlDbType.Int);

command.Parameters["@CustomerID"].Value = (Int32)key;

try

{

data = RunProcedure(command);

if (!data.IsClosed)

{

if (data.Read())

{

props.SetState(data);

}

else

throw new Exception("Record does not exist in the database.");

}

return props;

}

catch (Exception e)

{

// log this exception

throw;

}

finally

{

if (data != null)

{

if (!data.IsClosed)

data.Close();

}

}

} //end of Retrieve() }

public object RetrieveAll(Type type)

{

List<CustomerProps> list = new List<CustomerProps>();

DBDataReader reader = null;

CustomerProps props;

try

{

reader = RunProcedure("usp\_CustomerSelectAll");

if (!reader.IsClosed)

{

while (reader.Read())

{

props = new CustomerProps();

props.SetState(reader);

list.Add(props);

}

}

return list;

}

catch (Exception e)

{

// log this exception

throw;

}

finally

{

if (!reader.IsClosed)

{

reader.Close();

}

}

}

}

}

**ProductSQLDB.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using EventPropsClasses;

using ToolsCSharp;

using System.Data;

using System.Data.SqlClient;

// \*\*\* I use an "alias" for the ado.net classes throughout my code

// When I switch to an oracle database, I ONLY have to change the actual classes here

using DBBase = ToolsCSharp.BaseSQLDB;

using DBConnection = System.Data.SqlClient.SqlConnection;

using DBCommand = System.Data.SqlClient.SqlCommand;

using DBParameter = System.Data.SqlClient.SqlParameter;

using DBDataReader = System.Data.SqlClient.SqlDataReader;

using DBDataAdapter = System.Data.SqlClient.SqlDataAdapter;

using EventPropsClassses;

namespace EventDBClasses

{

//DBBase is the abstract base class with a bunch of methods that can help

public class CustomerSQLDB : DBBase, IReadDB, IWriteDB

{

#region Constructors

public CustomerSQLDB() : base() { }

public CustomerSQLDB(string cnString) : base(cnString) { }

public CustomerSQLDB(DBConnection cn) : base(cn) { }

#endregion

public IBaseProps Create(IBaseProps p)

{

int rowsAffected = 0;

ProductProps props = (ProductProps)p;

DBCommand command = new DBCommand();

command.CommandText = "usp\_ProductCreate";

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@ProductID", SqlDbType.Int);

command.Parameters.Add("@ProductCode", SqlDbType.Char);

command.Parameters.Add("@Description", SqlDbType.VarChar);

command.Parameters.Add("@UnitPrice", SqlDbType.Money);

command.Parameters.Add("@OnHandQuantity", SqlDbType.Int);

command.Parameters[0].Direction = ParameterDirection.Output;

command.Parameters["@ProductCode"].Value = props.Code;

command.Parameters["@Description"].Value = props.Description;

command.Parameters["@UnitPrice"].Value = props.UnitPrice;

command.Parameters["@OnHandQuantity"].Value = props.Quantity;

try

{

rowsAffected = RunNonQueryProcedure(command); //how many records were affected when I did this?

if (rowsAffected == 1)

{

props.ID = (int)command.Parameters[0].Value;

props.ConcurrencyID = 1;

return props;

}

else

throw new Exception("Unable to insert record. " + props.ToString());

}

catch (Exception e)

{

// log this error

throw;

}

finally

{

if (mConnection.State == ConnectionState.Open)

mConnection.Close();

}

}

public bool Update(IBaseProps p)

{

int rowsAffected = 0;

ProductProps props = (ProductProps)p;

DBCommand command = new DBCommand();

command.CommandText = "usp\_ProductUpdate";

command.CommandType = CommandType.StoredProcedure;

//command.Parameters.Add("@ProductID", SqlDbType.Int);

command.Parameters.Add("@ProductCode", SqlDbType.Char);

command.Parameters.Add("@Description", SqlDbType.VarChar);

command.Parameters.Add("@UnitPrice", SqlDbType.Money);

command.Parameters.Add("@OnHandQuantity", SqlDbType.Int);

//command.Parameters.Add("@ConcurrencyID", SqlDbType.Int);

command.Parameters["@ProductCode"].Value = props.Code;

command.Parameters["@Description"].Value = props.Description;

command.Parameters["@UnitPrice"].Value = props.UnitPrice;

command.Parameters["@OnHandQuantity"].Value = props.Quantity;

//command.Parameters["@ConcurrencyID"].Value = props.ConcurrencyID;

try

{

rowsAffected = RunNonQueryProcedure(command);

if (rowsAffected == 1)

{

props.ConcurrencyID++;

return true;

}

else

{

string message = "Record cannot be updated. It has been edited by another user.";

throw new Exception(message);

}

}

catch (Exception e)

{

// log this exception

throw;

}

finally

{

if (mConnection.State == ConnectionState.Open)

mConnection.Close();

}

} // end of Update()

public bool Delete(IBaseProps p)

{

ProductProps props = (ProductProps)p;

int rowsAffected = 0;

DBCommand command = new DBCommand();

command.CommandText = "usp\_ProductDelete";

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@ProductID", SqlDbType.Int);

command.Parameters.Add("@ConcurrencyID", SqlDbType.Int);

command.Parameters["@Product ID"].Value = props.ID;

command.Parameters["@ConcurrencyID"].Value = props.ConcurrencyID;

try

{

rowsAffected = RunNonQueryProcedure(command);

if (rowsAffected == 1)

{

return true;

}

else

{

string message = "Record cannot be deleted. It has been edited by another user.";

throw new Exception(message);

}

}

catch (Exception e)

{

// log this exception

throw;

}

finally

{

if (mConnection.State == ConnectionState.Open)

mConnection.Close();

}

} // end of Delete()

public IBaseProps Retrieve(object key)

{

DBDataReader data = null;

ProductProps props = new ProductProps();

DBCommand command = new DBCommand();

command.CommandText = "usp\_ProductSelect";

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@ProductID", SqlDbType.Int);

command.Parameters["@ProductID"].Value = (Int32)key;

try

{

data = RunProcedure(command);

if (!data.IsClosed)

{

if (data.Read())

{

props.SetState(data);

}

else

throw new Exception("Record does not exist in the database.");

}

return props;

}

catch (Exception e)

{

// log this exception

throw;

}

finally

{

if (data != null)

{

if (!data.IsClosed)

data.Close();

}

}

} //end of Retrieve() }

public object RetrieveAll(Type type)

{

List<ProductProps> list = new List<ProductProps>();

DBDataReader reader = null;

ProductProps props;

try

{

reader = RunProcedure("usp\_ProductSelectAll");

if (!reader.IsClosed)

{

while (reader.Read())

{

props = new ProductProps();

props.SetState(reader);

list.Add(props);

}

}

return list;

}

catch (Exception e)

{

// log this exception

throw;

}

finally

{

if (!reader.IsClosed)

{

reader.Close();

}

}

}

}

}

**ProductProps.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

using System.IO;

using ToolsCSharp;

using DBDataReader = System.Data.SqlClient.SqlDataReader; //Gave data reader an alias to use when we want to swap databases

namespace EventPropsClassses

{

[Serializable]

public class ProductProps : IBaseProps

{

#region instance variables

/// <summary>

///

/// </summary>

public int ID = Int32.MinValue;

public string Code = "";

/// <summary>

///

/// </summary>

public string Description = "";

/// <summary>

///

/// </summary>

public decimal UnitPrice = 0m;

/// <summary>

///

/// </summary>

public int Quantity = 0;

/// <summary>

/// ConcurrencyID. See main docs, don't manipulate directly

/// Are two people trying to do stuff with the same class at the same time

/// </summary>

public int ConcurrencyID = 0;

#endregion

#region constructor

/// <summary>

/// Constructor. This object should only be instantiated by Customer, not used directly.

/// </summary>

public ProductProps()

{

}

#endregion

#region BaseProps Members

/// <summary>

/// Serializes this props object to XML, and writes the key-value pairs to a string.

/// </summary>

/// <returns>String containing key-value pairs</returns>

public string GetState()

{

XmlSerializer serializer = new XmlSerializer(this.GetType()); //Takes object and turns it into a string

StringWriter writer = new StringWriter();

serializer.Serialize(writer, this);

return writer.GetStringBuilder().ToString();

}

// I don't always want to generate xml in the db class so the

// props class can read in from xml

public void SetState(DBDataReader dr) //Takes fields out of the database and puts it into object

{

this.ID = (Int32)dr["ProductID"]; //properties - fields in database

this.Code = ((string)dr["ProductCode"]).Trim();//casting the properties as needed data type

this.Description = (string)dr["Description"];//taking it out of the data reader (dr)

this.UnitPrice = (decimal)dr["UnitPrice"];

this.Quantity = (int)dr["OnHandQuantity"];

this.ConcurrencyID = (Int32)dr["ConcurrencyID"];

}

/// <summary>

///

/// </summary>

public void SetState(string xml)

{

XmlSerializer serializer = new XmlSerializer(this.GetType()); //De-seralizes the stuff we just seralized

StringReader reader = new StringReader(xml);

ProductProps p = (ProductProps)serializer.Deserialize(reader);

this.ID = p.ID;

this.Code = p.Code;

this.Description = p.Description;

this.UnitPrice = p.UnitPrice;

this.Quantity = p.Quantity;

this.ConcurrencyID = p.ConcurrencyID;

}

#endregion

#region ICloneable Members

/// <summary>

/// Clones this object.

/// </summary>

/// <returns>A clone of this object.</returns>

public Object Clone()

{

ProductProps p = new ProductProps(); //Puts properties out of objects that we have and puts into the properties we just created

p.ID = this.ID;

p.Code = this.Code;

p.Description = this.Description;

p.UnitPrice = this.UnitPrice;

p.Quantity = this.Quantity;

p.ConcurrencyID = this.ConcurrencyID;

return p;

}

#endregion

}

}

**CustomerProps.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

using System.IO;

using ToolsCSharp;

using DBDataReader = System.Data.SqlClient.SqlDataReader;

namespace EventPropsClasses

{

[Serializable]

public class CustomerProps : IBaseProps

{

//ID, Name, Address, City, State, ZipCode, ConcurrencyID

#region instance variables

/// <summary>

///

/// </summary>

public int ID = Int32.MinValue;

/// <summary>

///

/// </summary>

public string Name = "";

/// <summary>

///

/// </summary>

public string Address = "";

/// <summary>

///

/// </summary>

public string City = "";

/// <summary>

///

/// </summary>

public string State = "";

/// <summary>

///

/// </summary>

public int ZipCode = 0;

/// <summary>

///

/// </summary>

public int ConcurrencyID = 0;

#endregion

#region constructor

/// <summary>

///

/// </summary>

public CustomerProps()

{

}

#endregion

#region BaseProps Members

/// <summary>

/// Serializes this props object to XML, and writes the key-value pairs to a string.

/// </summary>

/// <returns>String containing key-value pairs</returns>

public string GetState()

{

XmlSerializer serializer = new XmlSerializer(this.GetType()); //Takes object and turns it into a string

StringWriter writer = new StringWriter();

serializer.Serialize(writer, this);

return writer.GetStringBuilder().ToString();

}

// I don't always want to generate xml in the db class so the

// props class can read in from xml

public void SetState(DBDataReader dr) //Takes fields out of the database and puts it into object

{

this.ID = (Int32)dr["CustomerID"]; //properties - fields in database

this.Name = (string)dr["Name"];//casting the properties as needed data type

this.Address = (string)dr["Address"];//taking it out of the data reader (dr)

this.City = (string)dr["City"];

this.State = (string)dr["State"];

this.ZipCode = (int)dr["ZipCode"];

this.ConcurrencyID = (Int32)dr["ConcurrencyID"];

}

/// <summary>

///

/// </summary>

public void SetState(string xml)

{

XmlSerializer serializer = new XmlSerializer(this.GetType()); //De-seralizes the stuff we just seralized

StringReader reader = new StringReader(xml);

CustomerProps c = (CustomerProps)serializer.Deserialize(reader);

this.ID = c.ID;

this.Name = c.Name;

this.Address = c.Address;

this.City = c.City;

this.State = c.State;

this.ZipCode = c.ZipCode;

this.ConcurrencyID = c.ConcurrencyID;

}

#endregion

#region ICloneable Members

/// <summary>

/// Clones this object.

/// </summary>

/// <returns>A clone of this object.</returns>

public Object Clone()

{

CustomerProps c = new CustomerProps(); //Puts properties out of objects that we have and puts into the properties we just created

c.ID = this.ID;

c.Name = this.Name;

c.Address = this.Address;

c.City = this.City;

c.State = this.State;

c.ZipCode = this.ZipCode;

c.ConcurrencyID = this.ConcurrencyID;

return c;

}

#endregion

}

}

**EventTests.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NUnit.Framework;

using EventClasses;

using EventPropsClasses;

using EventDBClasses;

using ToolsCSharp;

using System.Xml;

using System.Xml.Serialization;

using System.IO;

using System.Data;

using System.Data.SqlClient;

using DBCommand = System.Data.SqlClient.SqlCommand;

namespace EventTestClasses

{

[TestFixture]

public class EventTests

{

//private string folder = "C:\\Courses\\CS234CSharp\\Demos\\FrameworkExampleEvent\\Files\\";

// \*\*\* changed the name AND folder to db connection string

private string dataSource="Data Source=19158-S17809\\mssqlserver2014;Initial Catalog=EventCalendar;Integrated Security=True";

[Test]

public void TestNewEventConstructor()

{

// not in Data Store - no id

Customer e = new Customer(dataSource);

Console.WriteLine(e.ToString());

Assert.Greater(e.ToString().Length, 1);

}

[Test]

public void TestRetrieveFromDataStoreContructor()

{

// retrieves from Data Store

Customer e = new Customer(1, dataSource);

Assert.AreEqual(e.ID, 1);

Assert.AreEqual(e.Name, "First Event");

Console.WriteLine(e.ToString());

}

[Test]

public void TestSaveToDataStore()

{

Customer e = new Customer(dataSource);

e.CustomerID = 1;

e.Name = "Third Event";

e.Address = "This is the third event in my event list.";

e.Save();

Assert.AreEqual(3, e.ID);

}

[Test]

public void TestUpdate()

{

Customer e = new Customer(1, dataSource);

e.CustomerID = 3;

e.Name = "Edited Event";

e.Save();

e = new Customer(1, dataSource);

Assert.AreEqual(e.ID, 1);

Assert.AreEqual(e.CustomerID, 3);

Assert.AreEqual(e.Name, "Edited Event");

}

[Test]

public void TestDelete()

{

Customer e = new Customer(2, dataSource);

e.Delete();

e.Save();

Assert.Throws<Exception>(() => new Customer(2, dataSource));

}

[Test]

public void TestStaticDelete()

{

Customer.Delete(2, dataSource);

Assert.Throws<Exception>(() => new Customer(2, dataSource));

}

[Test]

public void TestStaticGetList()

{

List<Customer> events = Customer.GetList(dataSource);

Assert.AreEqual(2, events.Count);

Assert.AreEqual(1, events[0].ID);

Assert.AreEqual("First Event", events[0].Name);

}

// \*\*\* I added this

[Test]

public void TestGetList()

{

Customer e = new Customer(dataSource);

List<Customer> events = (List<Customer>)e.GetList();

Assert.AreEqual(2, events.Count);

Assert.AreEqual(1, events[0].ID);

Assert.AreEqual("First Event", events[0].Name);

}

// \*\*\* I added this

[Test]

public void TestGetTable()

{

DataTable events = Customer.GetTable(dataSource);

Assert.AreEqual(events.Rows.Count, 2);

}

[Test]

public void TestNoRequiredPropertiesNotSet()

{

// not in Data Store - userid, title and description must be provided

Customer e = new Customer(dataSource);

Assert.Throws<Exception>(() => e.Save());

}

[Test]

public void TestSomeRequiredPropertiesNotSet()

{

// not in Data Store - userid, title and description must be provided

Customer e = new Customer(dataSource);

Assert.Throws<Exception>(() => e.Save());

e.CustomerID = 1;

Assert.Throws<Exception>(() => e.Save());

e.Name = "this is a test";

Assert.Throws<Exception>(() => e.Save());

}

[Test]

public void TestInvalidPropertyUserIDSet()

{

Customer e = new Customer(dataSource);

Assert.Throws<ArgumentOutOfRangeException>(() => e.CustomerID = -1);

}

// \*\*\* I added this

[Test]

public void TestConcurrencyIssue()

{

Customer e1 = new Customer(1, dataSource);

Customer e2 = new Customer(1, dataSource);

e1.Name = "Updated this first";

e1.Save();

e2.Name = "Updated this second";

Assert.Throws<Exception>(() => e2.Save());

}

#region OtherStuff

/\*

[SetUp]

public void WriteListOfProps()

{

List<EventProps> events = new List<EventProps>();

EventProps props = new EventProps();

props.ID = 1;

props.userID = 1;

props.date = DateTime.Now;

props.title = "First Event";

props.description = "This is the description of the first event";

events.Add(props);

props = new EventProps();

props.ID = 2;

props.userID = 1;

props.date = DateTime.Now;

props.title = "Second Event";

props.description = "This is the description of the second event";

events.Add(props);

XmlSerializer serializer = new XmlSerializer(events.GetType());

Stream writer = new FileStream(folder + "EventTextDB.xml", FileMode.Create);

serializer.Serialize(writer, events);

writer.Close();

}

\*/

// \*\*\* I added changed this. It calls the stored procedure to reset the db

[SetUp]

public void TestResetDatabase()

{

EventSQLDB db = new EventSQLDB(dataSource);

DBCommand command = new DBCommand();

command.CommandText = "usp\_testingResetData";

command.CommandType = CommandType.StoredProcedure;

db.RunNonQueryProcedure(command);

}

[Test]

public void TestPropsRetrieve()

{

EventSQLDB db = new EventSQLDB(dataSource);

EventProps props = (EventProps)db.Retrieve(2);

Assert.AreEqual(props.ID, 2);

Console.WriteLine(props.GetState());

}

#endregion

}

}

**ProductDBTests.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NUnit.Framework;

using EventPropsClasses;

using EventPropsClassses;

using EventDBClasses;

using System.Data;

using System.Data.SqlClient;

using DBCommand = System.Data.SqlClient.SqlCommand;

namespace EventTestClasses

{

[TestFixture]

public class ProductDBTests

{

private string dataSource = "Data Source=MANDA\\SQLEXPRESS;Initial Catalog=MMABooksUpdated;Integrated Security=True";

private CustomerSQLDB db;

[SetUp]

public void TestResetDatabase()

{

db = new CustomerSQLDB(dataSource);

DBCommand command = new DBCommand();

command.CommandText = "usp\_testingResetData";

command.CommandType = CommandType.StoredProcedure;

db.RunNonQueryProcedure(command);

}

[Test]

public void TestProductDBRetrieve()

{

ProductProps props = (ProductProps)db.Retrieve(1);

Assert.AreEqual("A4CS", props.Code);

}

[Test]

public void TestProductDBRetrieveAll()

{

List<ProductProps> list = new List<ProductProps>();

list = (List<ProductProps>)db.RetrieveAll(list.GetType()); //This is different than anything else we've done before -- type class

Assert.AreEqual(16, list.Count);

}

[Test]

public void TestProductDBCreate()

{

ProductProps p = new ProductProps();

p.Code = "p100";

p.Description = "Test Product";

p.Quantity = 10;

p.UnitPrice = 100m;

ProductProps newP = (ProductProps)db.Create(p);

Assert.AreEqual(17, newP.ID);

Assert.AreEqual(newP.ConcurrencyID, 1);

}

[Test]

public void TestProductDBUpdate()

{

//ProductProps p = new ProductProps();

//p.Code = "p100";

//p.Description = "Test Product";

//p.Quantity = 10;

//p.UnitPrice = 100m;

}

[Test]

public void TestProductDBDelete()

{

ProductProps p = new ProductProps();

}

}

}

**ProductPropsTests.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NUnit.Framework;

using EventPropsClasses;

using EventPropsClassses;

namespace EventTestClasses

{

[TestFixture]

public class ProductPropsTests

{

private ProductProps p;

[SetUp]

public void SetUpTests()

{

p = new ProductProps();

p.ID = 10;

p.Code = "p000";

p.Description = "This is a test";

p.Quantity = 10;

p.ConcurrencyID = 1;

}

[Test]

public void TestClone()

{

//ProductProps p = new ProductProps();

//p.ID = 10;

//p.Code = "p000";

//p.Description = "This is a test";

//p.Quantity = 10;

//p.ConcurrencyID = 1;

ProductProps newP = (ProductProps)p.Clone();

Assert.AreEqual(p.ID, newP.ID);

Assert.AreEqual(p.Code, newP.Code);

Assert.AreEqual(p.Description, newP.Description);

Assert.AreEqual(p.Quantity, newP.Quantity);

Assert.AreEqual(p.ConcurrencyID, newP.ConcurrencyID);

}

[Test]

public void TestGetState()

{

string output = p.GetState();

Assert.True(output.Contains("This is a test"));

Console.WriteLine(output);

}

[Test]

public void TestSetState()

{

string output = p.GetState();

ProductProps newP = new ProductProps();

newP.SetState(output);

Assert.AreEqual(p.ID, newP.ID);

Assert.AreEqual(p.Code, newP.Code);

Assert.AreEqual(p.Description, newP.Description);

Assert.AreEqual(p.Quantity, newP.Quantity);

Assert.AreEqual(p.ConcurrencyID, newP.ConcurrencyID);

}

}

}