

Northwind ASP.NET MVC

Sarah Jastrzab



December 7, 2014

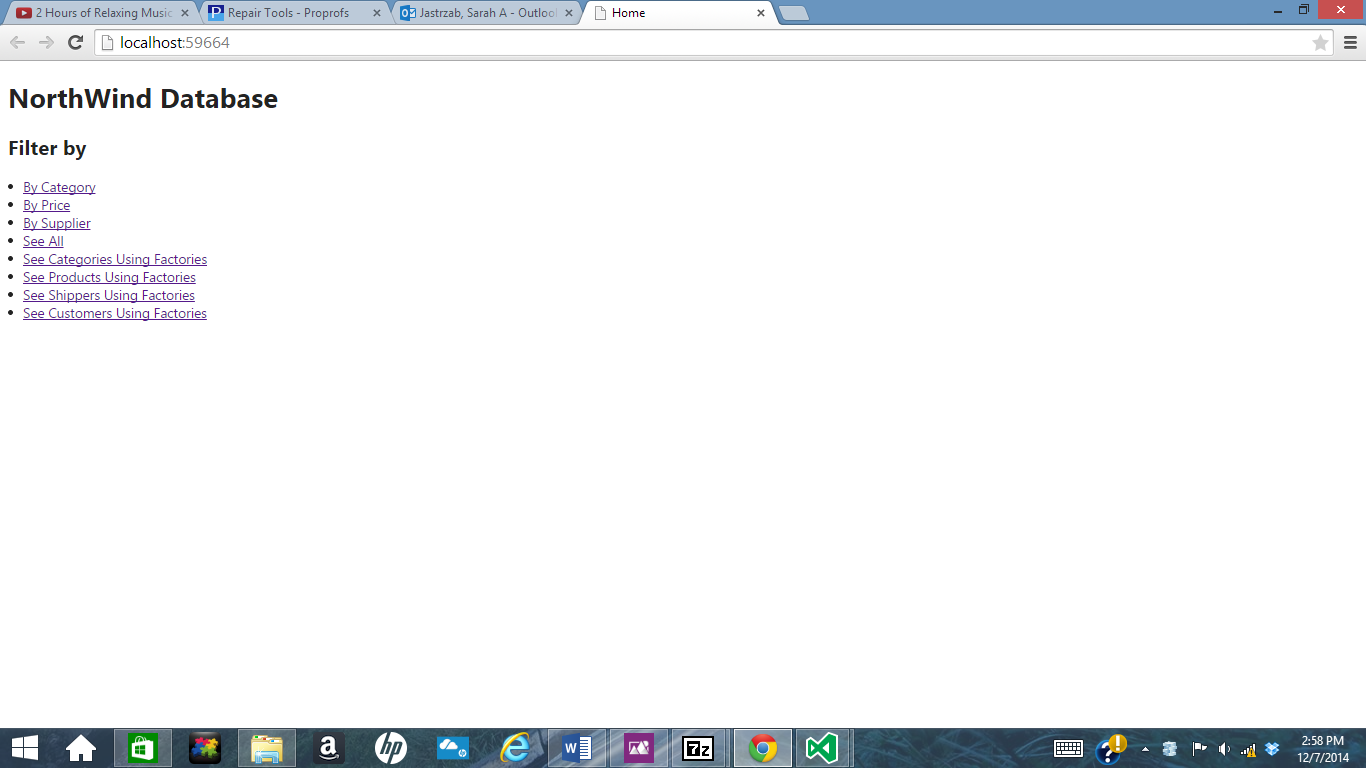
Alfred STate College

Object Oriented Programming

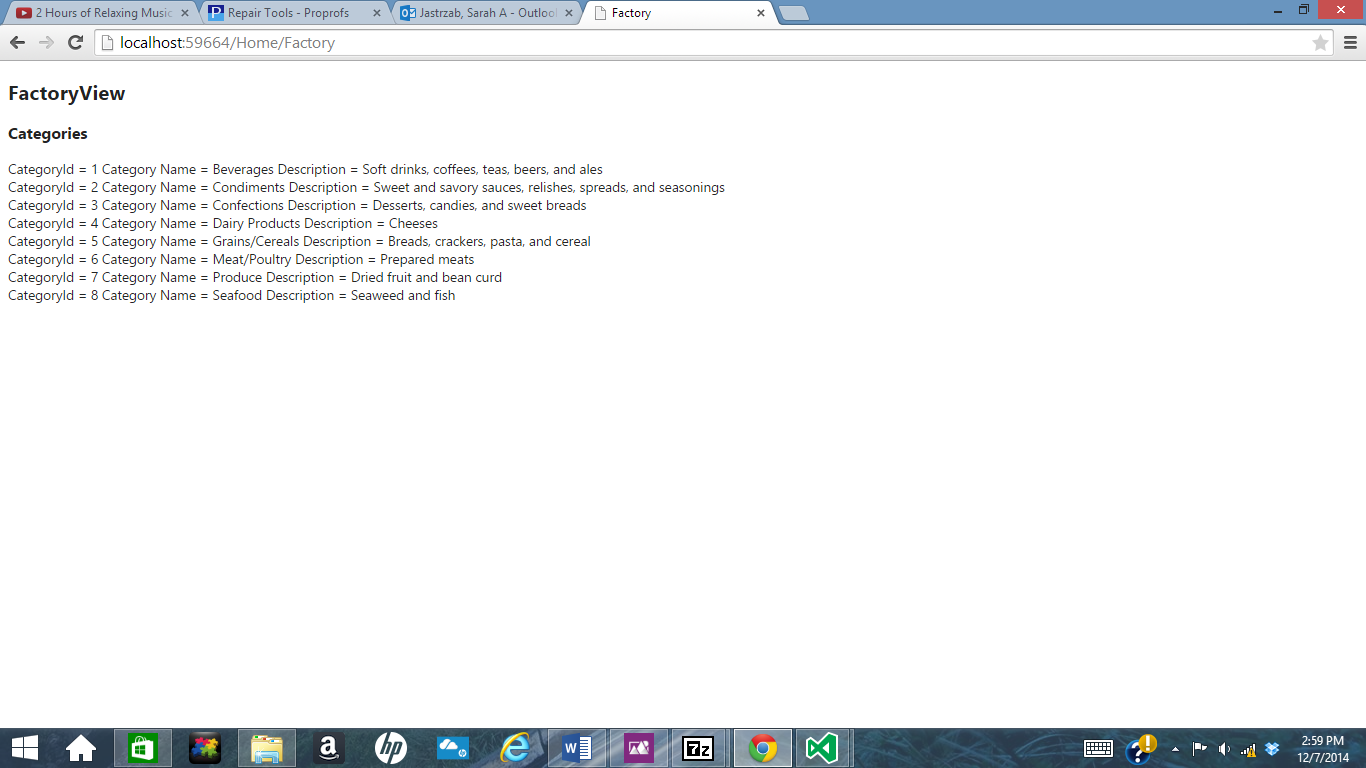
Northwind is a database tracks information about a company’s products, orders, employees, customer, shipper and suppliers. This program uses data from some of the tables in this database to show inheritance the factory, abstract factory, table gateway, identity map and broken rules design patterns. The program is written in C# using the ASP.Net MVC 5 framework.

Screen Shots

Index View(Home Page)



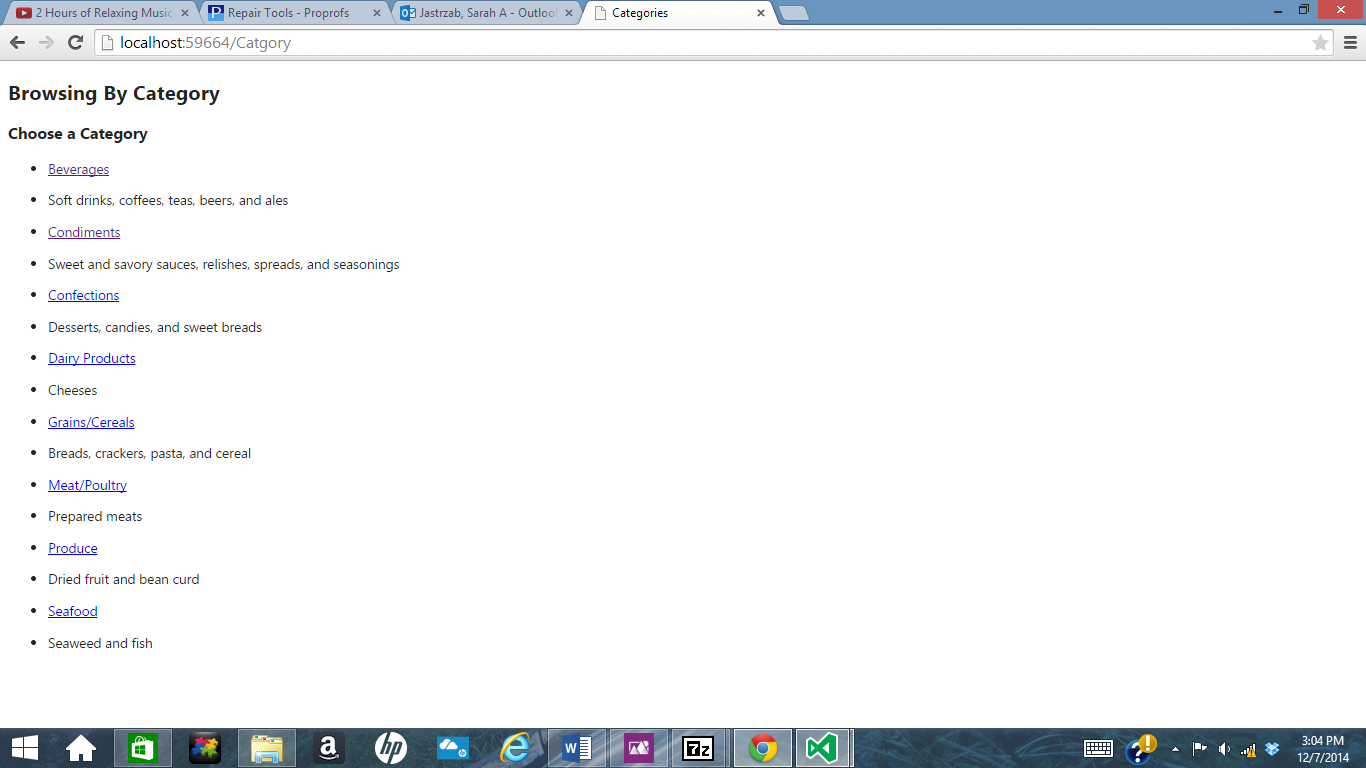
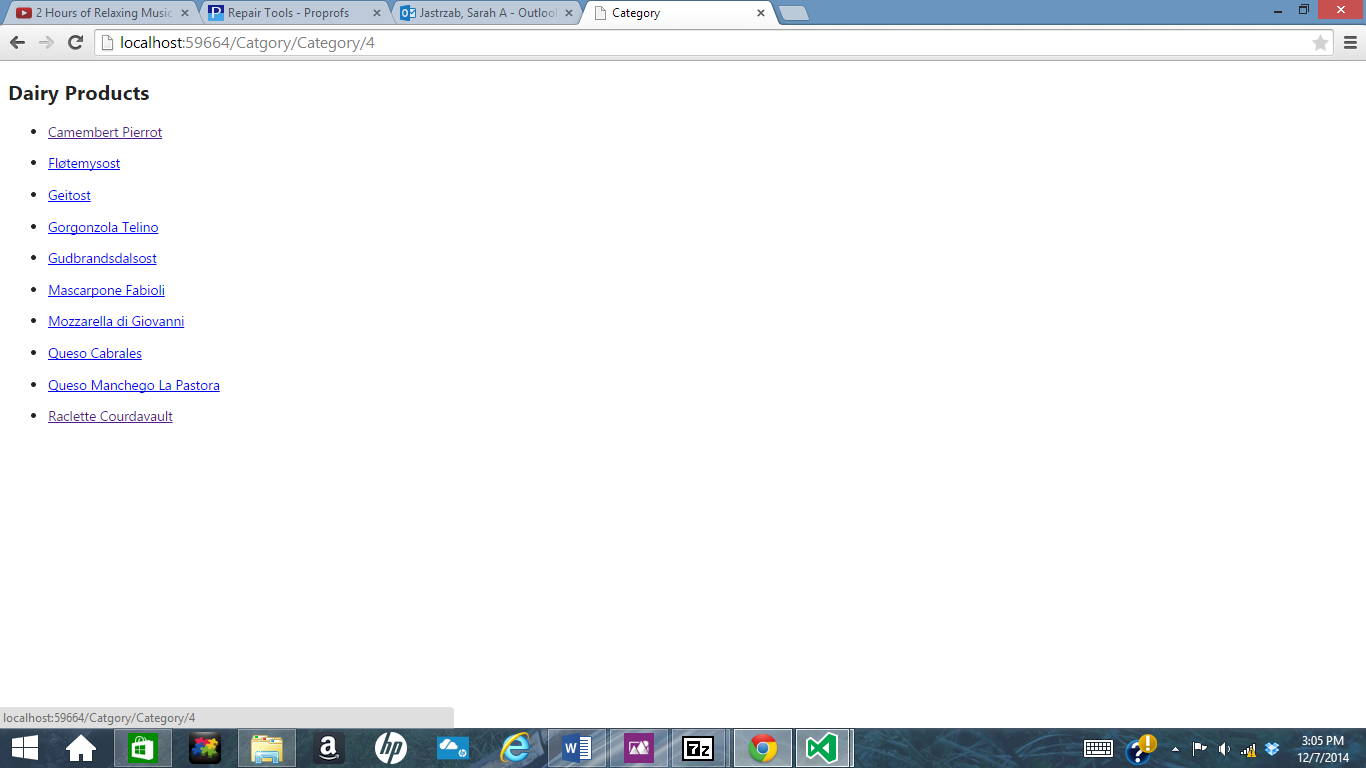
Categories Data Reader Printed using Factory



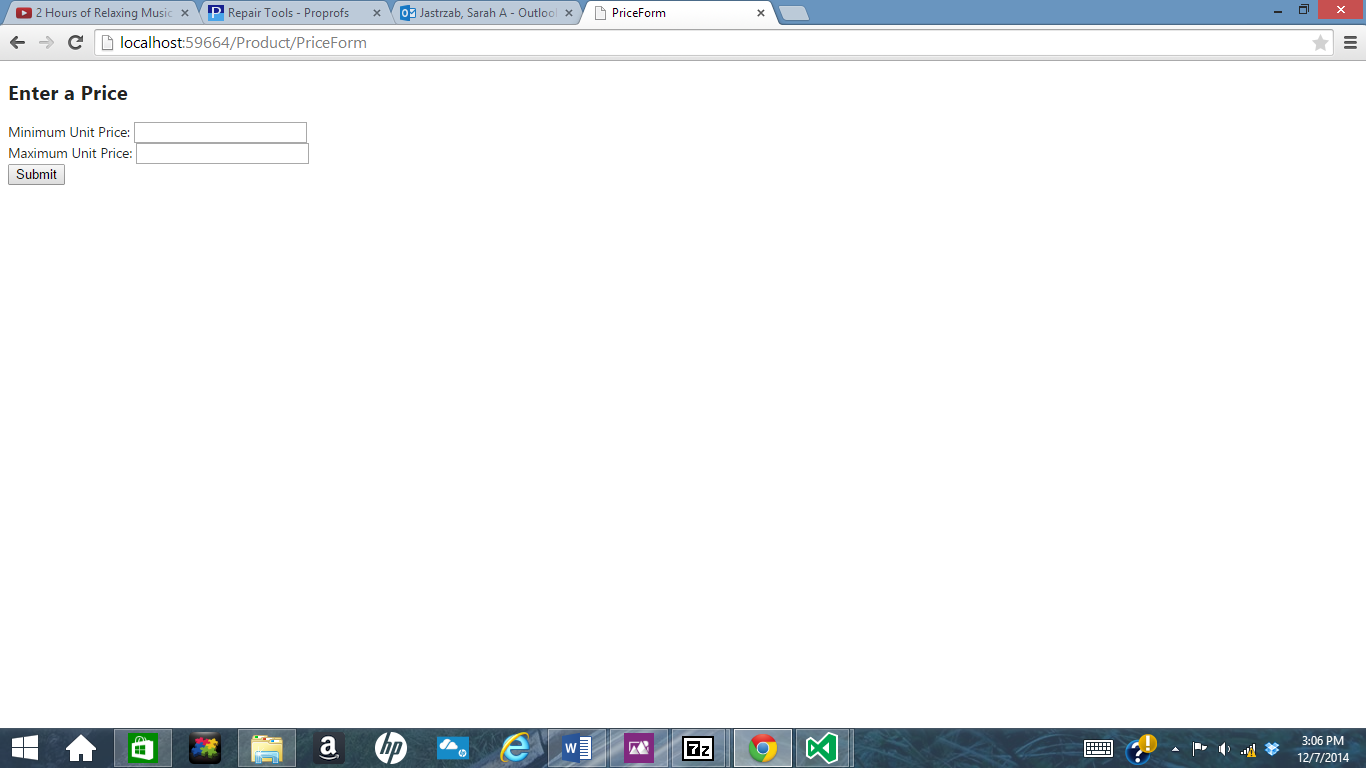
Customers Data Reader Printed using Factory



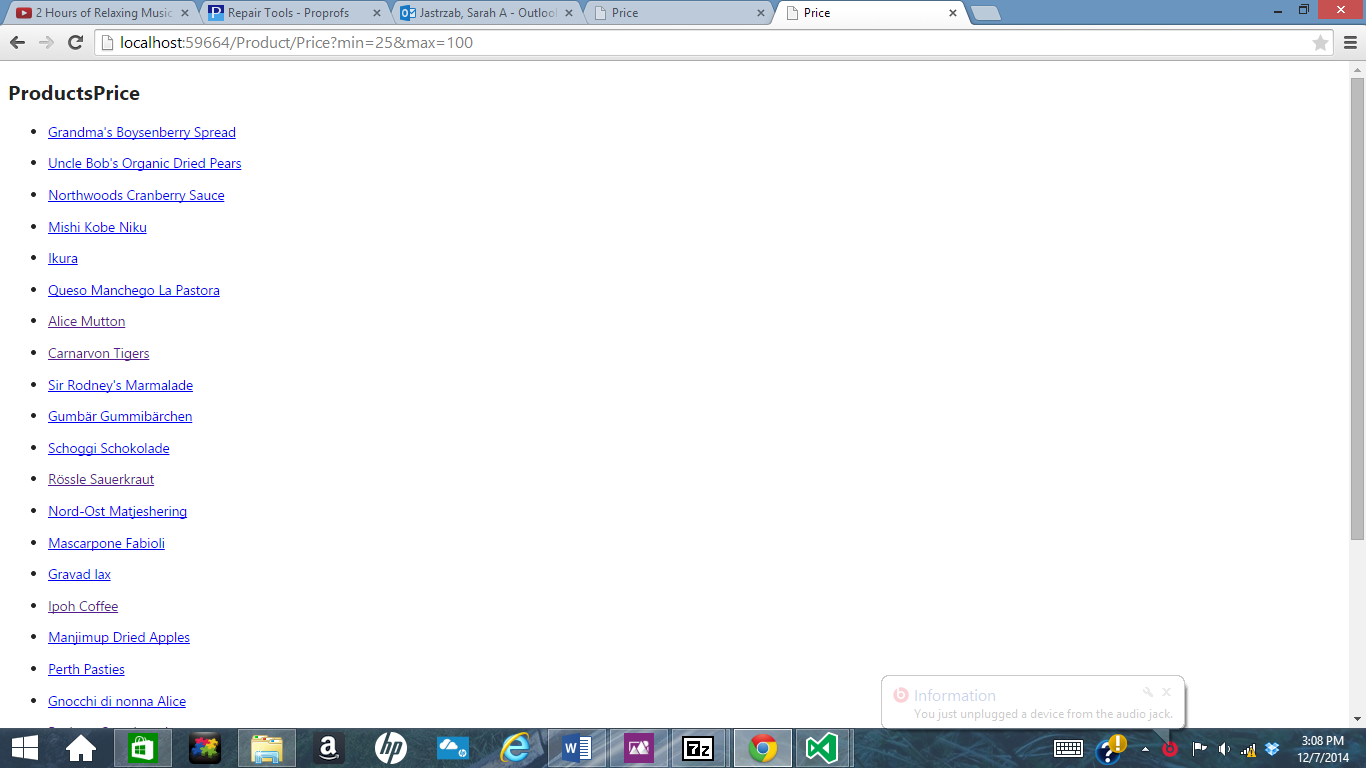
Filtering by Category All the Dairy Products



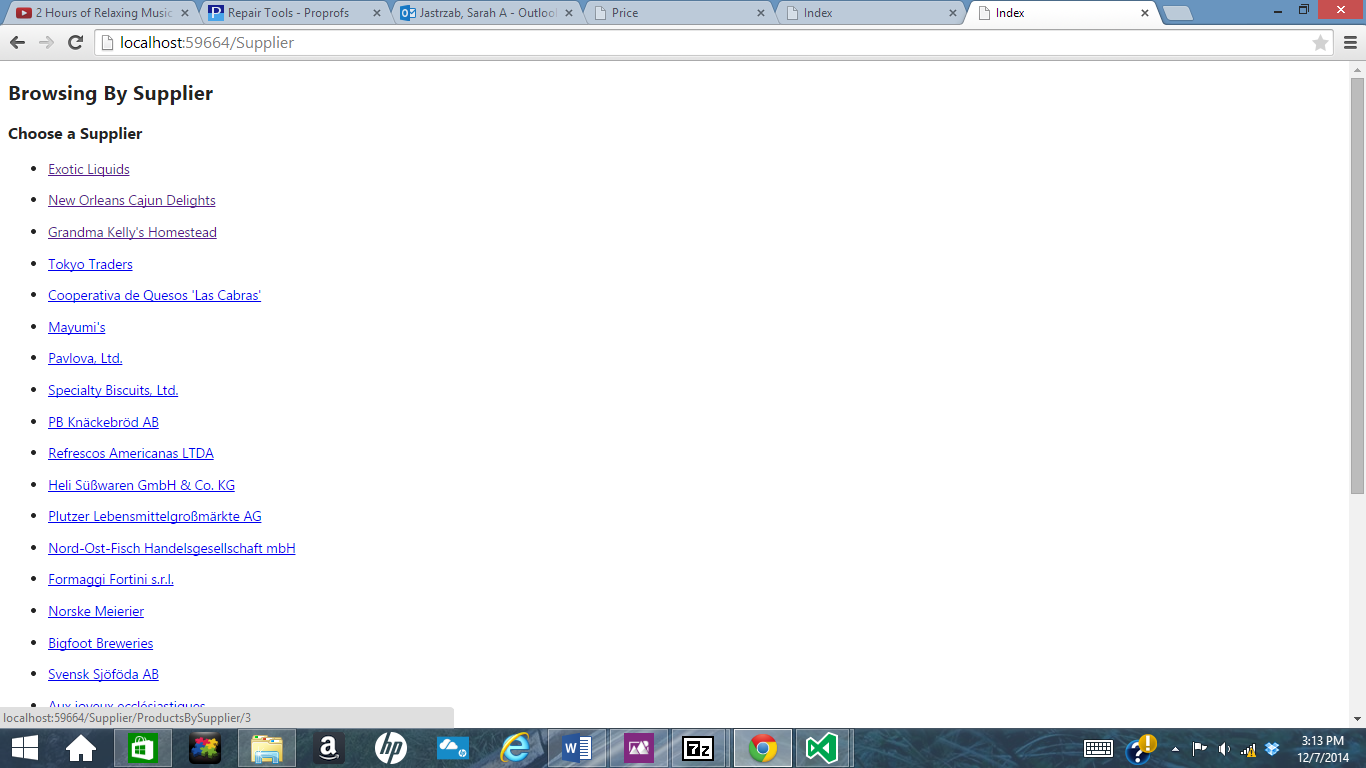
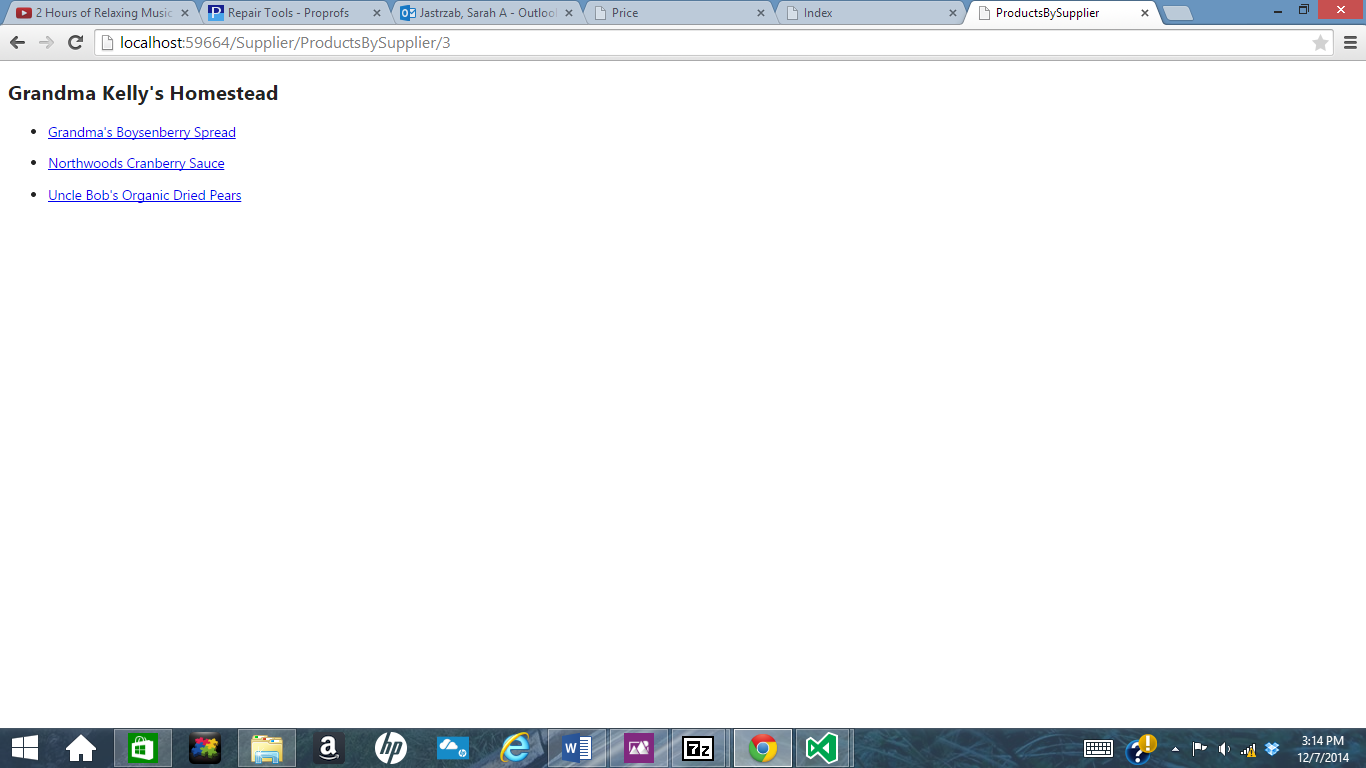
Filtering By Price



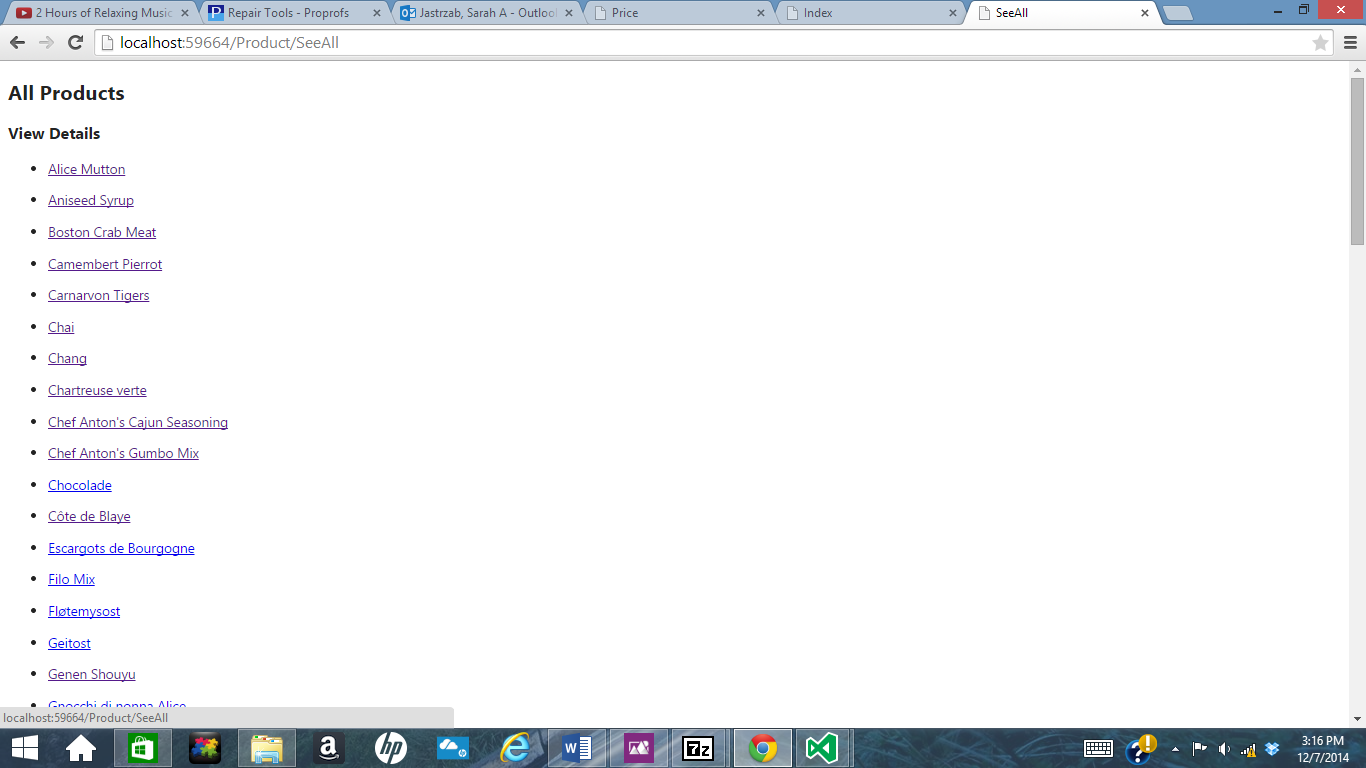
Filtered List of Products by Price



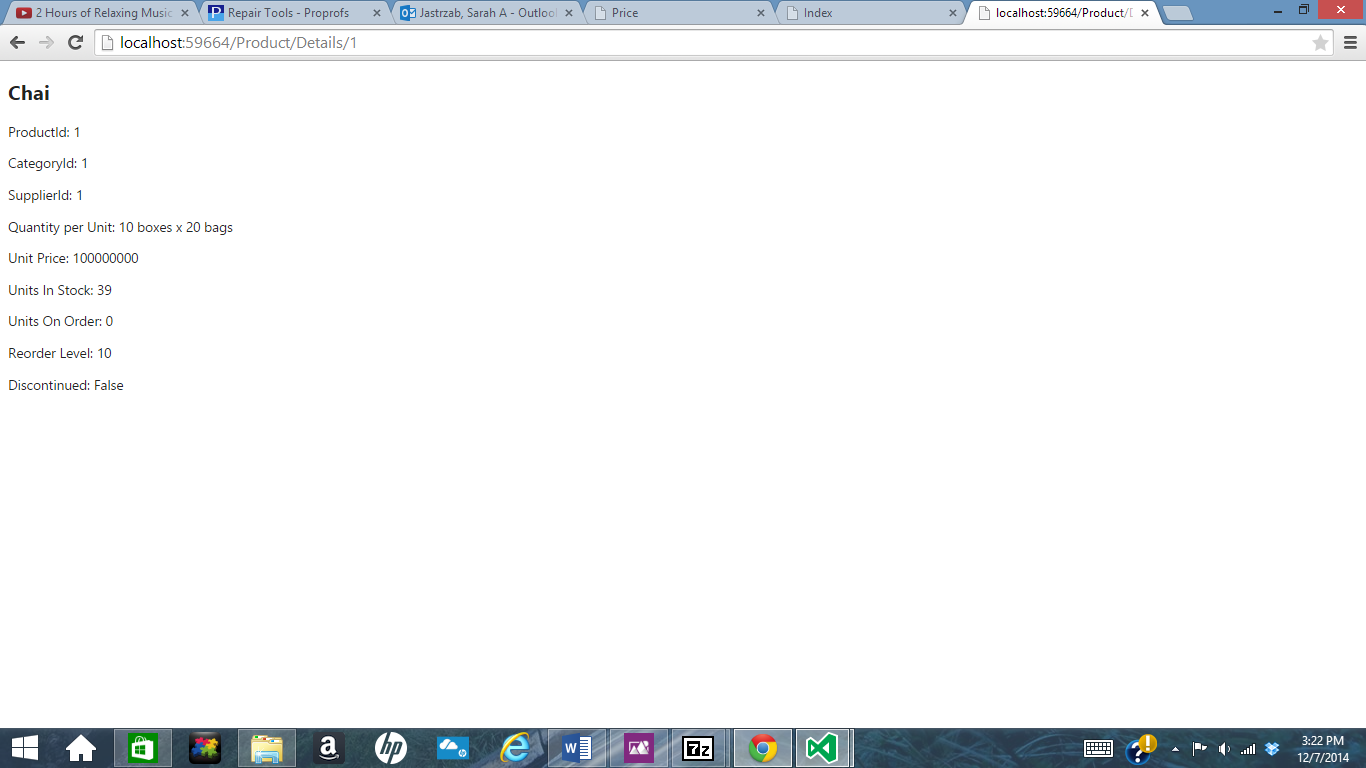
Browse by Supplier Supplier’s Product List



See All Products



Product Details Printout



The following classes are models. There is a model for the product, shipper, supplier, customer, and category based on the Northwind database. There are also two abstract class, Company and CompanyContact, that can’t be implemented and are used to demonstrate inheritance. Company is the top-most class in the hierarchy. Shipper and CompanyContact inherit from it. CompanyContact has properties that Shipper doesn’t have but Customer and Supplier do. These classes all contain private variables. Accessor methods are used to change this variables for the objects that are built based on these classes. This protects private data and keeps the code functioning.

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Product Class. It contained the variables, the accessor methods,

\* the constructor and the toString method for the object. It also implement

\* broken rules and can be defined as dirty if changed due to the Identity Maps.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

public class Product

{

//private class members are created here

private int productId = -1;

private string productName = "na";

private int supplierId = -1;

private int categoryId = -1;

private string quantityPerUnit = "na";

private double unitPrice = 10000000.00;

private int unitsInStock = -1;

private int unitsOnOrder = -1;

private int reorderLevel = -1;

private bool discontinued = false;

private static int numberOfProducts = 0;

private bool isDirty = false;

private BrokenRules theBrokenRules = new BrokenRules();

//Accessor methods begin here. ProductId is read only. If statements are used to //set values if a field is left blank by a user. Broken rules and dirty are implements in set methods

public int ProductId

{

get

{

return this.supplierId;

}

}

public string ProductName

{

get

{

return this.productName;

}

set

{

this.productName = value;

isDirty = true;

if (productName == "na")

{

this.productName = "Error";

}

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "No Name";

aRule.RuleDescription = "There is no name";

theBrokenRules.CheckRule(aRule.RuleName, aRule, !(value == null));

}

}

public int SupplierId

{

get

{

return this.supplierId;

}

set

{

this.supplierId = value;

isDirty = true;

if (supplierId == -1)

{

this.supplierId = -1;

}

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "No shipper";

aRule.RuleDescription = "There is no shipper";

theBrokenRules.CheckRule(aRule.RuleName, aRule, !(value == null));

}

}

public int CategoryId

{

get

{

return this.categoryId;

}

set

{

this.categoryId = value;

isDirty = true;

if (categoryId == -1)

{

this.categoryId = -1;

}

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "No Category";

aRule.RuleDescription = "There is no category";

theBrokenRules.CheckRule(aRule.RuleName, aRule, !(value == null));

}

}

public string QuantityPerUnit

{

get

{

return this.quantityPerUnit;

}

set

{

this.quantityPerUnit = value;

isDirty = true;

if (quantityPerUnit == "na")

{

this.quantityPerUnit = "Error";

}

}

}

public double UnitPrice

{

get

{

return this.unitPrice;

}

set

{

isDirty = true;

this.unitPrice = 100000000.00;

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "Negative Price";

aRule.RuleDescription = "The Price is not negative";

theBrokenRules.CheckRule(aRule.RuleName, aRule, !(value < 0));

}

}

public int UnitsInStock

{

get

{

return this.unitsInStock;

}

set

{

isDirty = true;

this.unitsInStock = value;

}

}

public int UnitsOnOrder

{

get

{

return this.unitsOnOrder;

}

set

{

isDirty = true;

this.unitsOnOrder = value;

}

}

public int ReorderLevel

{

get

{

return this.reorderLevel;

}

set

{

isDirty = true;

this.reorderLevel = value;

}

}

public bool Discontinued

{

get

{

return this.discontinued;

}

set

{

isDirty = true;

this.discontinued = value;

}

}

public bool IsDirty

{

get

{

return isDirty;

}

set

{

isDirty = value;

}

}

public bool isValid

{

get

{

if (theBrokenRules.GetDictionary().Count > 0)

{

return false;

}

else

{

return true;

}

}

}

//Constructors begin here, starting with the empty constructor.

public Product()

{

Product.numberOfProducts = Product.numberOfProducts + 1;

}

public Product(int aProductId, string aProductName, int aSupplierId, int aCategoryId, string aQuantityPerUnit, double aUnitPrice, int aUnitsInStock, int aUnitsOnOrder, int aReorderLevel, bool aDiscontinued)

: this()

{

this.productId = aProductId;

this.ProductName = aProductName;

this.SupplierId = aSupplierId;

this.CategoryId = aCategoryId;

this.QuantityPerUnit = aQuantityPerUnit;

this.UnitPrice = aUnitPrice;

this.UnitsInStock = aUnitsInStock;

this.UnitsOnOrder = aUnitsOnOrder;

this.ReorderLevel = aReorderLevel;

this.Discontinued = aDiscontinued;

}

public Product(int aProductId, string aProductName, int aSupplierId, int aCategoryId, string aQuantityPerUnit, double aUnitPrice, int aUnitsInStock, int aUnitsOnOrder, int aReorderLevel)

: this(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, false)

{

}

public Product(int aProductId, string aProductName, int aSupplierId, int aCategoryId, string aQuantityPerUnit, double aUnitPrice, int aUnitsInStock, int aUnitsOnOrder)

: this(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, -1, false)

{

}

public Product(int aProductId, string aProductName, int aSupplierId, int aCategoryId, string aQuantityPerUnit, double aUnitPrice, int aUnitsInStock)

: this(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, -1, -1, false)

{

}

public Product(int aProductId, string aProductName, int aSupplierId, int aCategoryId, string aQuantityPerUnit, double aUnitPrice)

: this(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, -1, -1, -1, false)

{

}

public Product(int aProductId, string aProductName, int aSupplierId, int aCategoryId, string aQuantityPerUnit)

: this(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, 1000000000, -1, -1, -1, false)

{

}

public Product(int aProductId, string aProductName, int aSupplierId, int aCategoryId)

: this(aProductId, aProductName, aSupplierId, aCategoryId, "na", 1000000000, -1, -1, -1, false)

{

}

public Product(int aProductId, string aProductName, int aSupplierId)

: this(aProductId, aProductName, aSupplierId, -1, "na", 1000000000, -1, -1, -1, false)

{

}

public Product(int aProductId, string aProductName)

: this(aProductId, aProductName, -1, -1, "na", 1000000000, -1, -1, -1, false)

{

}

public Product(int aProductId)

: this(aProductId, "na", -1, -1, "na", 1000000000, -1, -1, -1, false)

{

}

//The toString method begins here

public override string ToString()

{

string aString = "";

aString = aString + "Product Id = " + ProductId + "\n";

aString = aString + "Product Name = " + ProductName + "\n";

aString = aString + "Supplier Id = " + SupplierId + "\n";

aString = aString + "Category Id = " + CategoryId + "\n";

aString = aString + "Quantity per unit = " + QuantityPerUnit + "\n";

aString = aString + "UnitPrice = " + UnitPrice + "\n";

aString = aString + "UnitInStock = " + UnitsInStock + "\n";

aString = aString + "UnitsOnOrder = " + UnitsOnOrder + "\n";

aString = aString + "ReorderLevel = " + ReorderLevel + "\n";

aString = aString + "Discontinued = " + Discontinued + "\n";

aString = aString + "NumberOfProducts = " + Product.numberOfProducts + "\n";

return aString;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Category Class. It contained the variables, the accessor methods,

\* the constructor and the toString method for the object. This class also implements

\* broken rules and an Identity Map.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

public class Category

{

//Create private class members

private int categoryId = 0000;

private string categoryName = "n/a";

private string description = "n/a";

private BrokenRules theBrokenRules = new BrokenRules();

private bool isDirty = false;

//Accessor methods begin here, create class properties

//aCategoryId is read only. If statements are used to set values for fields

//left blank by users.

public int CategoryId

{

get

{

return this.categoryId;

}

}

public bool isValid

{

get

{

if (theBrokenRules.GetDictionary().Count > 0)

{

return false;

}

else

{

return true;

}

}

}

public string CategoryName

{

get

{

return this.categoryName;

}

set

{

this.categoryName = value;

isDirty = true;

/\*if (categoryName == "na")

{

this.categoryName = "Error";

}\*/

//Broken Rules are implemented

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non empty Category Name";

theBrokenRules.CheckRule(aRule.RuleName, aRule, !(value.Length > 0));

}

}

public string Description

{

get

{

return this.description;

}

set

{

this.description = value;

isDirty = true;

/\*if (description == "na")

{

this.description = "Error";

}\*/

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non empty Category Name";

theBrokenRules.CheckRule(aRule.RuleName, aRule, !(value.Length > 0));

}

}

public bool IsDirty

{

get

{

return isDirty;

}

set

{

isDirty = value;

}

}

//Constructors begin here, starting with the empty constructor.

public Category()

{

}

public Category(int aCategoryId, string aCategoryName, string aDescription)

: this()

{

this.categoryId = aCategoryId;

this.CategoryName = aCategoryName;

this.Description = aDescription;

}

public Category(int aCategoryId, string aCategoryName)

: this(aCategoryId, aCategoryName, "na")

{

}

public Category(int aCategoryId)

: this(aCategoryId, "na", "na")

{

}

//toString method begins here

public new string ToString()

{

string aString = "";

aString = aString + "CategoryId = " + CategoryId + "\n";

aString = aString + "Category Name = " + CategoryName + "\n";

aString = aString + "Description = " + Description + "\n";

return aString;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Company Class. It contained the variables, the accessor methods,

\* the constructor and the toString method for the Company base object.

\* This object cannot be created!

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

//The base class

public abstract class Company

{

//Encapsulated Variables

private string companyName = "n/a";

private string phone = "n/a";

private bool isDirty = false;

//Get and set methods

public bool IsDirty

{

get

{

return isDirty;

}

set

{

isDirty = value;

}

}

public string CompanyName

{

get

{

return this.companyName;

}

set

{

this.companyName = value;

isDirty = true;

if (companyName == "na")

{

this.companyName = "na";

}

}

}

public string Phone

{

get

{

return this.phone;

}

set

{

this.phone = value;

isDirty = true;

if (phone == "na")

{

this.phone = "na";

}

}

}

//Constructors

public Company()

{

}

public Company(string aCompanyName, string aPhone)

: this()

{

this.CompanyName = aCompanyName;

this.Phone = aPhone;

}

public Company(string aCompanyName)

: this(aCompanyName, "na")

{

}

//ToString method

public override string ToString()

{

string aString = "";

aString = aString + "Company Name = " + CompanyName + "\n";

aString = aString + "Phone Number = " + Phone + "\n";

return aString;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the CompanyContact Class. It contained the variables, the accessor methods,

\* the constructor and the toString method for the object. This class acts as a base

\* class and the object cannot be created.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

//A class that inherits from Company

public abstract class CompanyContact:Company

{

//encapsulated variables

private string contactName = "na";

private string contactTitle = "na";

private string address = "na";

private string city = "na";

private string region = "na";

private string postalCode = "na";

private string country = "na";

private string phone = "na";

private string fax = "na";

private bool isDirty = false;

//Accessor methods

public bool IsDirty

{

get

{

return isDirty;

}

set

{

isDirty = value;

}

}

public string ContactName

{

get

{

return this.contactName;

}

set

{

this.contactName = value;

isDirty = true;

if (contactName == "na")

{

this.contactName = "na";

}

}

}

public string ContactTitle

{

get

{

return this.contactTitle;

}

set

{

this.contactTitle = value;

isDirty = true;

if (contactTitle == "na")

{

this.contactTitle = "na";

}

}

}

public string Address

{

get

{

return this.address;

}

set

{

this.address = value;

isDirty = true;

if (address == "na")

{

this.address = "na";

}

}

}

public string City

{

get

{

return this.city;

}

set

{

this.city = value;

isDirty = true;

if (city == "na")

{

this.city = "na";

}

}

}

public string Region

{

get

{

return this.region;

}

set

{

this.region = value;

isDirty = true;

if (region == "na")

{

this.region = "na";

}

}

}

public string PostalCode

{

get

{

return this.postalCode;

}

set

{

this.postalCode = value;

isDirty = true;

if (postalCode == "na")

{

this.postalCode = "na";

}

}

}

public string Country

{

get

{

return this.country;

}

set

{

this.country = value;

isDirty = true;

if (country == "na")

{

this.country = "na";

}

}

}

public string Phone

{

get

{

return this.phone;

}

set

{

this.phone = value;

isDirty = true;

if (phone == "na")

{

this.phone = "na";

}

}

}

public string Fax

{

get

{

return this.fax;

}

set

{

this.fax = value;

isDirty = true;

if (fax == "na")

{

this.fax = "na";

}

}

}

//Constructors that call the base constructor from the parent class

public CompanyContact()

: base()

{

}

public CompanyContact(string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry, string aPhone, string aFax)

: base(aCompanyName, aPhone)

{

this.ContactName = aContactName;

this.ContactTitle = aContactTitle;

this.Address = aAddress;

this.City = aCity;

this.Region = aRegion;

this.PostalCode = aPostalCode;

this.Country = aCountry;

this.Fax = aFax;

}

public CompanyContact(string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry, string aPhone)

: this(aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, "n/a")

{

}

public CompanyContact(string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry)

: this(aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, "n/a", "n/a")

{

}

public CompanyContact(string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode)

: this(aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, "n/a", "n/a", "n/a")

{

}

public CompanyContact(string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion)

: this(aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, "n/a", "n/a", "n/a", "n/a")

{

}

public CompanyContact(string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity)

: this(aCompanyName, aContactName, aContactTitle, aAddress, aCity, "n/a", "n/a", "n/a", "n/a", "n/a")

{

}

public CompanyContact(string aCompanyName, string aContactName, string aContactTitle, string aAddress)

: this(aCompanyName, aContactName, aContactTitle, aAddress, "n/a", "n/a", "n/a", "n/a", "n/a", "n/a")

{

}

public CompanyContact(string aCompanyName, string aContactName, string aContactTitle)

: this(aCompanyName, aContactName, aContactTitle, "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a")

{

}

public CompanyContact(string aCompanyName, string aContactName)

: this(aCompanyName, aContactName, "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a")

{

}

public CompanyContact(string aCompanyName)

: this(aCompanyName, "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a")

{

}

//ToString method

public override string ToString()

{

string aString = base.ToString();

aString = aString + "ContactName = " + ContactName + "\n";

aString = aString + "ContactTitle = " + ContactTitle + "\n";

aString = aString + "Address = " + Address + "\n";

aString = aString + "City = " + City + "\n";

aString = aString + "Region = " + Region + "\n";

aString = aString + "Postal Code = " + PostalCode + "\n";

aString = aString + "Country = " + Country + "\n";

aString = aString + "Phone = " + Phone + "\n";

aString = aString + "Fax = " + Fax + "\n";

return aString;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Shipper Class. It contained the variables, the accessor methods,

\* the constructor and the toString method for the object. It inherits from Company

\* Class.\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

//Inheriting from Company

public class Shipper : Company

{

//private class members are created here

private int shipperId = -1;

//Accessor methods begin here. ShipperId is read only. If statements are used to set values if a field is left blank by a user

public int ShipperId

{

get

{

return this.shipperId;

}

}

public String GetPhone()

{

return "";

}

//Constructors begin here, starting with the empty constructor.

public Shipper()

: base()

{

}

public Shipper(int aShipperId, string aCompanyName, string aPhone)

: base(aCompanyName, aPhone)

{

this.shipperId = aShipperId;

}

//The toString method begins here

public override string ToString()

{

string aString = base.ToString();

aString = aString + "Shipper Id = " + ShipperId + "\n";

return aString;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Category Class. It contained the variables, the accessor methods,

\* the constructor and the toString method for the object.It inherits from Company

\* Contact.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

public class Supplier:CompanyContact

{

//The private class members are created here

private int supplierId = -1;

private string homePage = "na";

private bool isDirty = false;

private BrokenRules theBrokenRules = new BrokenRules();

//Accessor methods are created, SupplierId is read only. If statements are used

//to set values when fields are left blank by users

public int SupplierId

{

get

{

return this.supplierId;

}

}

public string HomePage

{

get

{

return this.homePage;

}

set

{

this.homePage = value;

isDirty = true;

if (homePage == "na")

{

this.homePage = "Error";

}

}

}

public bool IsDirty

{

get

{

return isDirty;

}

set

{

isDirty = value;

}

}

public bool isValid

{

get

{

if (theBrokenRules.GetDictionary().Count > 0)

{

return false;

}

else

{

return true;

}

}

}

//Constructors begin here, starting with the empty constructor.

public Supplier()

: base()

{

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry, string aPhone, string aFax, string aHomePage)

: base(aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, aFax)

{

this.supplierId = aSupplierId;

this.HomePage = aHomePage;

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry, string aPhone, string aFax)

: this(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, aFax, "na")

{

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry, string aPhone)

: this(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, "na", "na")

{

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry)

: this(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, "na", "na", "na")

{

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode)

: this(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, "na", "na", "na", "na")

{

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion)

: this(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, "na", "na", "na", "na", "na")

{

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity)

: this(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, "na", "na", "na", "na", "na", "na")

{

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName, string aContactTitle, string aAddress)

: this(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, "na", "na", "na", "na", "na", "na", "na")

{

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName, string aContactTitle)

: this(aSupplierId, aCompanyName, aContactName, aContactTitle, "na", "na", "na", "na", "na", "na", "na", "na")

{

}

public Supplier(int aSupplierId, string aCompanyName, string aContactName)

: this(aSupplierId, aCompanyName, aContactName, "na", "na", "na", "na", "na", "na", "na", "na", "na")

{

}

public Supplier(int aSupplierId, string aCompanyName)

: this(aSupplierId, aCompanyName, "na", "na", "na", "na", "na", "na", "na", "na", "na", "na")

{

}

public Supplier(int aSupplierId)

: this(aSupplierId, "na", "na", "na", "na", "na", "na", "na", "na", "na", "na", "na")

{

}

//toString method begins here.

public override string ToString()

{

string aString = base.ToString();

aString = "Supplier Id = " + SupplierId + aString + "\n";

aString = aString + "HomePage = " + HomePage + "\n";

return aString;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Customer Class. It contained the variables, the accessor methods,

\* the constructor and the toString method for the object.It inherits from Company

\* Contact.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

public class Customer : CompanyContact

{

//private class members are created here

private string customerId = "-1";

//Accessor methods begin here. CustomerId is read only. If statements are used to //set values if a field is left blank by a user

public string CustomerId

{

get

{

return this.customerId;

}

}

public Customer()

: base()

{

//ALWAYS CALL THE BASE CONSTRUCTOR

}

public Customer(string aCustomerId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry, string aPhone, string aFax)

: base(aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, aFax)

{

this.customerId = aCustomerId;

}

public Customer(string aCustomerId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry, string aPhone)

: this(aCustomerId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, "na")

{

}

public Customer(string aCustomerId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode, string aCountry)

: this(aCustomerId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, "na", "na")

{

}

public Customer(string aCustomerId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion, string aPostalCode)

: this(aCustomerId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, "na", "na", "na")

{

}

public Customer(string aCustomerId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity, string aRegion)

: this(aCustomerId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, "na", "na", "na", "na")

{

}

public Customer(string aCustomerId, string aCompanyName, string aContactName, string aContactTitle, string aAddress, string aCity)

: this(aCustomerId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, "na", "na", "na", "na", "na")

{

}

public Customer(string aCustomerId, string aCompanyName, string aContactName, string aContactTitle, string aAddress)

: this(aCustomerId, aCompanyName, aContactName, aContactTitle, aAddress, "na", "na", "na", "na", "na", "na")

{

}

public Customer(string aCustomerId, string aCompanyName, string aContactName, string aContactTitle)

: this(aCustomerId, aCompanyName, aContactName, aContactTitle, "na", "na", "na", "na", "na", "na", "na")

{

}

public Customer(string aCustomerId, string aCompanyName, string aContactName)

: this(aCustomerId, aCompanyName, aContactName, "na", "na", "na", "na", "na", "na", "na", "na")

{

}

public Customer(string aCustomerId, string aCompanyName)

: this(aCustomerId, aCompanyName, "na", "na", "na", "na", "na", "na", "na", "na", "na")

{

}

public Customer(string aCustomerId)

: this(aCustomerId, "na", "na", "na", "na", "na", "na", "na", "na", "na", "na")

{

}

//The toString method begins here

public override string ToString()

{

string aString = base.ToString();

aString = "CustomerId = " + CustomerId + aString + "\n";

return aString;

}

}

}

The two classes that follow are used to create the Broken Rules design pattern. Business Rules allow only data that meets certain requirements to be added to the database. This keeps the database free of errors and protects a program. The Broken Rule class serves as a model. All rules that become broken use the properties and methods defined in this class. The Broken Rules Class contains a dictionary or collection of all the broken rules and has methods for adding, removing and checking the rules. Both of these classes are needed. They are then used in the set methods of the models as seen in the Product and Category classes.

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This is the Broken Rule Class. It defines the properties and methods for business

\* rules. Business Rules that are broken are entered into a log or database and the

\* information that is trying to enter the database will not be saved.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

public class BrokenRule

{

//Properties

private string ruleName = "n/a";

private string ruleDescription = "n/a";

//Methods

public string RuleName

{

get

{

return ruleName;

}

set

{

ruleName = value;

}

}

public string RuleDescription

{

get

{

return ruleDescription;

}

set

{

ruleDescription = value;

}

}

//ToString method

public override string ToString()

{

string aMessage = "";

aMessage = this.RuleName + " ";

aMessage = aMessage + this.RuleDescription + "<br />";

return aMessage;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This is the BrokenRules Class. Much like an Identity Map it uses dictionaries.

\* When business rules are broken, this class keeps a record of them and provides methods

\* to alter the list.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

public class BrokenRules

{

//create a dictionary

private Dictionary<string, BrokenRule> aDictionary = new Dictionary<string, BrokenRule>();

public Dictionary<string, BrokenRule> GetDictionary()

{

return aDictionary;

}

//Checks if something is in the dictionary

public Boolean isInDictionary(string aKey)

{

bool answer = false;

answer = aDictionary.ContainsKey(aKey);

return answer;

}

//Add, remove or check a rule

public void AddBrokenRule(string aKey, BrokenRule aBrokenRuleId)

{

aDictionary.Add(aKey, aBrokenRuleId);

}

public void RemoveBrokenRule(string aKey)

{

aDictionary.Remove(aKey);

}

public void CheckRule(string aRuleName, BrokenRule aRule, bool isBroken)

{

bool alreadyExists = this.isInDictionary(aRuleName);

if (isBroken == true && alreadyExists == false)

{

this.AddBrokenRule(aRuleName, aRule);

}

else if (isBroken == false && alreadyExists == true)

{

this.RemoveBrokenRule(aRuleName);

}

}

//ToString()

public override string ToString()

{

string aMessage = "";

foreach (var b in aDictionary){

aMessage = aMessage + b.Value.ToString();

}

return aMessage;

}

}

}

Identity Maps are a database access pattern used to prevent duplication of retrieving data. If the data has already been retrieved the Identity Map returns the instance that has already been loaded. In this Northwind Project there are three identity maps, one for Categories, Suppliers and one for Products.

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Category Identity Map Class. It puts categories into dictionaries and

\* check if the data trying to be saved has been altered.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

public class CategoryIdentityMap

{

private Dictionary<int, Category> aDictionary = new Dictionary<int, Category>();

//gets the dictionary

public Dictionary<int, Category> GetDictionary()

{

return aDictionary;

}

//test if the entry is in the dictionary

public Boolean isInDictionary(int aKey)

{

bool answer = false;

answer = aDictionary.ContainsKey(aKey);

return answer;

}

//adds an entry

public void AddCategory(int aKey, Category aCategory)

{

aDictionary.Add(aKey, aCategory);

}

//gets an entry

public Category GetCategory(int aKey)

{

return aDictionary[aKey];

}

//removes an entry

public void RemoveCategory(int aKey)

{

aDictionary.Remove(aKey);

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Category Identity Map Class. It puts categories into dictionaries and check if the data trying to be saved has been altered.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

public class SupplierIdentityMap

{

private Dictionary<int, Supplier> aDictionary = new Dictionary<int, Supplier>();

//gets the dictionary

public Dictionary<int, Supplier> GetDictionary()

{

return aDictionary;

}

//test if the entry is in the dictionary

public Boolean isInDictionary(int aKey)

{

bool answer = false;

answer = aDictionary.ContainsKey(aKey);

return answer;

}

//adds an entry

public void AddSupplier(int aKey, Supplier aSupplier)

{

aDictionary.Add(aKey, aSupplier);

}

//gets an entry

public Supplier GetSupplier(int aKey)

{

return aDictionary[aKey];

}

//removes an entry

public void RemoveSupplier(int aKey)

{

aDictionary.Remove(aKey);

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Product Identity Map Class. It puts categories into dictionaries and check

\* if the data trying to be saved has been altered.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

public class ProductIdentityMap

{

private Dictionary<int, Product> aDictionary = new Dictionary<int, Product>();

//gets the dictionary

public Dictionary<int, Product> GetDictionary()

{

return aDictionary;

}

//test if the entry is in the dictionary

public Boolean isInDictionary(int aKey)

{

bool answer = false;

answer = aDictionary.ContainsKey(aKey);

return answer;

}

//adds an entry

public void AddProduct(int aKey, Product aProduct)

{

aDictionary.Add(aKey, aProduct);

}

//gets an entry

public Product GetProduct(int aKey)

{

return aDictionary[aKey];

}

//removes an entry

public void RemoveProduct(int aKey)

{

aDictionary.Remove(aKey);

}

}

}

The following three classes are Table Gateways. These classes hold all the SQL for accessing a single table or view. Other code calls the methods within it for all interactions within the database. These keeps database code more organized and secure. There is a gateway for Categories, Products and Suppliers.

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the ProductGateway Class. It pulls product data out of an Access database and puts the data in lists.

\* These lists are filtered using CommandStrings and parameters passed into their creation methods.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.OleDb;

using System.Data.Odbc;

namespace Inheritance1.Models

{

Public class ProductGateway

{

private ProductIdentityMap aMap = new ProductIdentityMap();

private Product aProduct = null;

public static string connectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

public static OleDbConnection aConnection = new OleDbConnection(connectionString);

public OleDbCommand aCommand = aConnection.CreateCommand();

//Gets a List of Products from a database

public List<Product> GetProducts()

{

// open the connection

aConnection.Open();

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

aCommand.CommandText = "SELECT [ProductId],[ProductName],[SupplierId], CategoryId, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM Products";

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aProductId = (int)aReader["ProductId"];

string aProductName = (string)aReader["ProductName"];

int aSupplierId = (int)aReader["SupplierId"];

int aCategoryId = (int)aReader["CategoryId"];

string aQuantityPerUnit = (string)aReader["QuantityPerUnit"];

double aUnitPrice = (double)(decimal)aReader["UnitPrice"];

int aUnitsInStock = aReader["UnitsInStock"] as Int16? ?? 0;

int aUnitsOnOrder = (short)aReader["UnitsOnOrder"];

int aReorderLevel = (short)aReader["ReorderLevel"];

bool isDiscontinued = (bool)aReader["Discontinued"];

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aProductList.Add(aProduct);

}

aConnection.Close();

return aProductList;

}

public List<Product> GetProductsByCategory(String theId)

{

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

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

aCommand.CommandText = "SELECT ProductId,ProductName,SupplierId, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM Products WHERE CategoryID = " + theId;

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aProductId = (int)aReader["ProductId"];

string aProductName = (string)aReader["ProductName"];

int aSupplierId = (int)aReader["SupplierId"];

int aCategoryId = (int)aReader["CategoryId"];

string aQuantityPerUnit = (string)aReader["QuantityPerUnit"];

double aUnitPrice = (double)(decimal)aReader["UnitPrice"];

int aUnitsInStock = (short)aReader["UnitsInStock"];

int aUnitsOnOrder = (short)aReader["UnitsOnOrder"];

int aReorderLevel = (short)aReader["ReorderLevel"];

bool isDiscontinued = (bool)aReader["Discontinued"];

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aProductList.Add(aProduct);

}

aConnection.Close();

return aProductList;

}

public List<Product> GetProductsBySupplier(String theId)

{

// open the connection

aConnection.Open();

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

aCommand.CommandText = "SELECT ProductId, ProductName, SupplierId, CategoryId, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM Products WHERE SupplierId = " + theId;

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aProductId = (int)aReader["ProductId"];

string aProductName = (string)aReader["ProductName"];

int aSupplierId = (int)aReader["SupplierId"];

int aCategoryId = (int)aReader["CategoryId"];

string aQuantityPerUnit = (string)aReader["QuantityPerUnit"];

double aUnitPrice = (double)(decimal)aReader["UnitPrice"];

int aUnitsInStock = (short)aReader["UnitsInStock"];

int aUnitsOnOrder = (short)aReader["UnitsOnOrder"];

int aReorderLevel = (short)aReader["ReorderLevel"];

bool isDiscontinued = (bool)aReader["Discontinued"];

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aProductList.Add(aProduct);

}

aConnection.Close();

return aProductList;

}

//Gets a list of products filtered by the Unit Price

public List<Product> GetProductsByUnitPrice(double min, double max)

{

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

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

OleDbCommand aCommand = aConnection.CreateCommand();

BrokenRules theBrokenRules = new BrokenRules();

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "Negative Price";

aRule.RuleDescription = "The Price is not negative";

theBrokenRules.CheckRule(aRule.RuleName, aRule, !(min < 0));

theBrokenRules.CheckRule(aRule.RuleName, aRule, !(max < 0));

aCommand.CommandText = "SELECT [ProductId],[ProductName],[SupplierId], CategoryId, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM Products WHERE UnitPrice BETWEEN " + min + " AND " + max;

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aProductId = (int)aReader["ProductId"];

string aProductName = (string)aReader["ProductName"];

int aSupplierId = (int)aReader["SupplierId"];

int aCategoryId = (int)aReader["CategoryId"];

string aQuantityPerUnit = (string)aReader["QuantityPerUnit"];

double aUnitPrice = Convert.ToDouble(aReader["UnitPrice"] as decimal? ?? 9999);

int aUnitsInStock = (short)aReader["UnitsInStock"];

int aUnitsOnOrder = (short)aReader["UnitsOnOrder"];

int aReorderLevel = (short)aReader["ReorderLevel"];

bool isDiscontinued = (bool)aReader["Discontinued"];

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aProductList.Add(aProduct);

}

aConnection.Close();

return aProductList;

}

public List<Product> GetProductByID(string anID)

{

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

aConnection.Open();

aCommand.CommandText = "SELECT ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM Products WHERE ProductID = " + anID;

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aProductID = aReader["ProductID"] as int? ?? -1;

string aProductName = aReader["ProductName"] as string ?? String.Empty;

int aSupplierID = aReader["SupplierID"] as int? ?? -1;

int aCategoryID = aReader["CategoryID"] as int? ?? -1;

string aQuantityPerUnit = aReader["QuantityPerUnit"] as string ?? String.Empty;

double aUnitPrice = Convert.ToDouble(aReader["UnitPrice"] as decimal? ?? 999999);

int aUnitsInStock = aReader["UnitsInStock"] as Int16? ?? 0;

int aUnitsOnOrder = aReader["UnitsOnOrder"] as Int16? ?? 0;

int aReorderLevel = aReader["ReorderLevel"] as Int16? ?? 0;

bool isDiscontinued = aReader["Discontinued"] as bool? ?? true;

Product aProduct = new Product(aProductID, aProductName, aSupplierID, aCategoryID, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aList.Add(aProduct);

}

aConnection.Close();

return aList;

}

public void SaveAllProducts()

{

Dictionary<int, Product> dictionary = aMap.GetDictionary();

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "";

foreach (var p in dictionary.Keys)

{

int aProductId = dictionary[p].ProductId;

string aProductName = dictionary[p].ProductName;

int aSupplierId = dictionary[p].SupplierId;

int aCategoryId = dictionary[p].CategoryId;

string aQuantityPerUnit = dictionary[p].QuantityPerUnit;

double aUnitPrice = dictionary[p].UnitPrice;

int aUnitsInStock = dictionary[p].UnitsInStock;

int aUnitsOnOrder = dictionary[p].UnitsOnOrder;

int aReorderLevel = dictionary[p].ReorderLevel;

bool isDiscontinued = dictionary[p].Discontinued;

// If the item is not in the dictionary

// make a new item and add it to the map

if (dictionary[p].IsDirty == true)

{

aSQL = "Update Products";

aSQL = aSQL + "SET ProductName = " + "'" + aProductName + "', ";

aSQL = aSQL + "SupplierId = " + "'" + aSupplierId + "' ";

aSQL = aSQL + "CategoryId = " + "'" + aCategoryId + "' ";

aSQL = aSQL + "QuantityPerUnit = " + "'" + aQuantityPerUnit + "' ";

aSQL = aSQL + "UnitPrice = " + "'" + aUnitPrice + "' ";

aSQL = aSQL + "UnitsInStock = " + "'" + aUnitsInStock + "' ";

aSQL = aSQL + "UnitsOnOrder = " + "'" + aUnitsOnOrder + "' ";

aSQL = aSQL + "ReorderLevel = " + "'" + aReorderLevel + "' ";

aSQL = aSQL + "Discontinued = " + "'" + isDiscontinued + "' ";

aSQL = aSQL + "WHERE ProductId = " + aProductId;

aCommand.CommandText = aSQL;

aCommand.ExecuteNonQuery();

}

}

// close the connection

aConnection.Close();

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the SupplierGateway Class. It pulls supplier data out of an Access database

\* and puts the data in lists. These lists are filtered using CommandStrings and

\* parameters passed into their creation methods.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.OleDb;

using System.Data.Odbc;

namespace Inheritance1.Models

{

public class SupplierGateway

{

private SupplierIdentityMap aMap = new SupplierIdentityMap();

private Supplier aSupplier = null;

public static string connectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

public static OleDbConnection aConnection = new OleDbConnection(connectionString);

public OleDbCommand aCommand = aConnection.CreateCommand();

//Gets a list of all suppliers

public List<Supplier> GetSuppliers()

{

// open the connection

aConnection.Open();

List<Supplier> aSupplierList = new List<Supplier>();

aCommand.CommandText = "SELECT [SupplierId],[CompanyName],[ContactName], ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax, HomePage FROM Suppliers";

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aSupplierId = (int)aReader["SupplierId"];

string aCompanyName = (string)aReader["CompanyName"];

string aContactName = (string)aReader["ContactName"];

string aContactTitle = (string)aReader["ContactTitle"];

string aAddress = (string)aReader["Address"];

string aCity = (string)aReader["City"];

string aRegion = aReader["Region"] as string ?? String.Empty;

string aPostalCode = aReader["PostalCode"] as string ?? String.Empty;

string aCountry = (string)aReader["Country"];

string aPhone = (string)aReader["Phone"];

string aFax = aReader["Fax"] as string ?? String.Empty;

string aHomePage = aReader["HomePage"] as string ?? String.Empty;

Supplier aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

aSupplierList.Add(aSupplier);

}

aConnection.Close();

return aSupplierList;

}

//Gets a list of suppliers filtered by the supplier Id

public List<Supplier> GetSuppliersById(String theSupplier)

{

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

List<Supplier> aSupplierList = new List<Supplier>();

OleDbCommand aCommand = aConnection.CreateCommand();

aCommand.CommandText = "SELECT [SupplierId],[CompanyName],[ContactName], ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax, HomePage FROM Suppliers WHERE SupplierID = " + theSupplier;

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aSupplierId = (int)aReader["SupplierId"];

string aCompanyName = (string)aReader["CompanyName"];

string aContactName = (string)aReader["ContactName"];

string aContactTitle = (string)aReader["ContactTitle"];

string aAddress = (string)aReader["Address"];

string aCity = (string)aReader["City"];

string aRegion = aReader["Region"] as string ?? String.Empty;

string aPostalCode = aReader["PostalCode"] as string ?? String.Empty;

string aCountry = (string)aReader["Country"];

string aPhone = (string)aReader["Phone"];

string aFax = aReader["Fax"] as string ?? String.Empty;

string aHomePage = aReader["HomePage"] as string ?? String.Empty;

Supplier aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

aSupplierList.Add(aSupplier);

}

aConnection.Close();

return aSupplierList;

}

public void SaveAllSuppliers()

{

Dictionary<int, Supplier> dictionary = aMap.GetDictionary();

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "";

foreach (var p in dictionary.Keys)

{

int aSupplierId = dictionary[p].SupplierId;

string aCompanyName = dictionary[p].CompanyName;

string aContactName = dictionary[p].ContactName;

string aContactTitle = dictionary[p].ContactTitle;

string aAddress = dictionary[p].Address;

string aCity = dictionary[p].City;

string aRegion = dictionary[p].Region;

string aPostalCode = dictionary[p].PostalCode;

string aCountry = dictionary[p].Country;

string aPhone = dictionary[p].Phone;

string aFax = dictionary[p].Fax;

string aHomePage = dictionary[p].HomePage;

// If the item is not in the dictionary

// make a new item and add it to the map

if (dictionary[p].IsDirty == true)

{

aSQL = "Update Suppliers";

aSQL = aSQL + "SET CompanyName = " + "'" + aCompanyName + "', ";

aSQL = aSQL + "ContactName = " + "'" + aContactName + "' ";

aSQL = aSQL + "ContactTitle = " + "'" + aContactTitle + "' ";

aSQL = aSQL + "Address = " + "'" + aAddress + "' ";

aSQL = aSQL + "City = " + "'" + aCity + "' ";

aSQL = aSQL + "Region = " + "'" + aRegion + "' ";

aSQL = aSQL + "PostalCode = " + "'" + aPostalCode + "' ";

aSQL = aSQL + "Country = " + "'" + aCountry + "' ";

aSQL = aSQL + "Phone = " + "'" + aPhone + "' ";

aSQL = aSQL + "Fax = " + "'" + aFax + "' ";

aSQL = aSQL + "HomePage = " + "'" + aHomePage + "' ";

aSQL = aSQL + "WHERE SupplierId = " + aSupplierId;

aCommand.CommandText = aSQL;

aCommand.ExecuteNonQuery();

}

}

// close the connection

aConnection.Close();

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Category Gateway Class. It pulls category data out of an Access database and puts the data in lists.

\* These lists are filtered using CommandStrings and parameters passed into their creation methods.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.OleDb;

using System.Data.Odbc;

namespace Inheritance1.Models

{

public class CategoryGateWay

{

private CategoryIdentityMap aMap = new CategoryIdentityMap();

private Category aCategory = null;

public List<Category> GetCategories()

{

// create a List

List<Category> aListOfCategories = new List<Category>();

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT CategoryId, CategoryName, Description FROM Categories";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aCategoryId = (int)aReader["CategoryId"];

string aCategoryName = (string)aReader["CategoryName"];

string aDescription = (string)aReader["Description"];

Category aCategory = new Category(aCategoryId, aCategoryName, aDescription);

aListOfCategories.Add(aCategory);

}

// close the connection

aConnection.Close();

return aListOfCategories;

}

//Gets a List of Categories ssorted by ID

public List<Category> GetCategoriesById(String theId)

{

// create a List

List<Category> aListOfCategories = new List<Category>();

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT CategoryId, CategoryName, Description FROM Categories WHERE CategoryId = " + theId;

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aCategoryId = (int)aReader["CategoryId"];

string aCategoryName = (string)aReader["CategoryName"];

string aDescription = (string)aReader["Description"];

Category aCategory = new Category(aCategoryId, aCategoryName, aDescription);

aListOfCategories.Add(aCategory);

}

// close the connection

aConnection.Close();

return aListOfCategories;

}

public void SaveAllCategories()

{

Dictionary<int, Category> dictionary = aMap.GetDictionary();

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "";

foreach (var c in dictionary.Keys)

{

int aCategoryId = dictionary[c].CategoryId;

string aCategoryName = dictionary[c].CategoryName;

string aDescription = dictionary[c].Description;

// If the item is not in the dictionary

// make a ew item and add it to the map

if (dictionary[c].IsDirty == true)

{

aSQL = "Update Categories";

aSQL = aSQL + "SET CategoryName = " + "'" + aCategoryName + "', ";

aSQL = aSQL + "Description = " + "'" + aDescription + "' ";

aSQL = aSQL + "WHERE CategoryId = " + aCategoryId;

aCommand.CommandText = aSQL;

aCommand.ExecuteNonQuery();

}

}

// close the connection

aConnection.Close();

}

}

}

This class is the OleDbUtilityLoader. This class can load all of the tables from Northwind. It is no longer used in the program because it makes the code used to load databases unorganized. Table Gateways replaced this code.

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Database Loader Class. It pulls data out of an Access database and puts the data in lists.

\* These lists are filtered using CommandStrings and parameters passed into their creation methods.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data;

using System.Data.OleDb;

using System.Data.Odbc;

namespace Inheritance1.Models

{

public class OleDbUtilityLoader

{

//Variables used to create a connection to a access database

public static string connectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

public static OleDbConnection aConnection = new OleDbConnection(connectionString);

public OleDbCommand aCommand = aConnection.CreateCommand();

//Gets a List of Products from a database

public List<Product> GetProducts(){

// open the connection

aConnection.Open();

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

aCommand.CommandText = "SELECT [ProductId], [ProductName], [SupplierId], CategoryId, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM Products";

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aProductId = (int)aReader["ProductId"];

string aProductName = (string)aReader["ProductName"];

int aSupplierId = (int)aReader["SupplierId"];

int aCategoryId = (int)aReader["CategoryId"];

string aQuantityPerUnit = (string)aReader["QuantityPerUnit"];

double aUnitPrice = (double)(decimal)aReader["UnitPrice"];

int aUnitsInStock = aReader["UnitsInStock"] as Int16? ?? 0;

int aUnitsOnOrder = (short)aReader["UnitsOnOrder"];

int aReorderLevel = (short)aReader["ReorderLevel"];

bool isDiscontinued = (bool)aReader["Discontinued"];

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aProductList.Add(aProduct);

}

aConnection.Close();

return aProductList;

}

//Gets a list of Categories from a database

public List<Category> GetCategories()

{

// create a List

List<Category> aListOfCategories = new List<Category>();

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT CategoryId, CategoryName, Description FROM Categories";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aCategoryId = (int)aReader["CategoryId"];

string aCategoryName = (string)aReader["CategoryName"];

string aDescription = (string)aReader["Description"];

Category aCategory = new Category(aCategoryId, aCategoryName, aDescription);

aListOfCategories.Add(aCategory);

}

// close the connection

aConnection.Close();

return aListOfCategories;

}

//Gets a List of Categories ssorted by ID

public List<Category> GetCategoriesById(String theId)

{

// create a List

List<Category> aListOfCategories = new List<Category>();

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT CategoryId, CategoryName, Description FROM Categories WHERE CategoryId = " + theId;

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aCategoryId = (int)aReader["CategoryId"];

string aCategoryName = (string)aReader["CategoryName"];

string aDescription = (string)aReader["Description"];

Category aCategory = new Category(aCategoryId, aCategoryName, aDescription);

aListOfCategories.Add(aCategory);

}

// close the connection

aConnection.Close();

return aListOfCategories;

}

//Gets a List of Products organized by CategoryId

public List<Product> GetProductsByCategory(String theId)

{

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

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

aCommand.CommandText = "SELECT ProductId,ProductName,SupplierId, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM Products WHERE CategoryID = " + theId;

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read()){

int aProductId = (int) aReader["ProductId"];

string aProductName = (string) aReader["ProductName"];

int aSupplierId = (int) aReader["SupplierId"];

int aCategoryId = (int)aReader["CategoryId"];

string aQuantityPerUnit = (string) aReader["QuantityPerUnit"];

double aUnitPrice = (double)(decimal) aReader["UnitPrice"];

int aUnitsInStock = (short)aReader["UnitsInStock"];

int aUnitsOnOrder = (short)aReader["UnitsOnOrder"];

int aReorderLevel = (short)aReader["ReorderLevel"];

bool isDiscontinued = (bool)aReader["Discontinued"];

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aProductList.Add(aProduct);

}

aConnection.Close();

return aProductList;

}

//Gets a list of suppliers filtered by the supplier Id

public List<Supplier> GetSuppliersById(String theSupplier)

{

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

List<Supplier> aSupplierList = new List<Supplier>();

OleDbCommand aCommand = aConnection.CreateCommand();

aCommand.CommandText = "SELECT [SupplierId],[CompanyName],[ContactName], ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax, HomePage FROM Suppliers WHERE SupplierID = " + theSupplier;

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read()){

int aSupplierId = (int) aReader["SupplierId"];

string aCompanyName = (string) aReader["CompanyName"];

string aContactName = (string) aReader["ContactName"];

string aContactTitle = (string)aReader["ContactTitle"];

string aAddress = (string) aReader["Address"];

string aCity = (string) aReader["City"];

string aRegion = aReader["Region"] as string ?? String.Empty;

string aPostalCode = aReader["PostalCode"] as string ?? String.Empty;

string aCountry = (string)aReader["Country"];

string aPhone = (string)aReader["Phone"];

string aFax = aReader["Fax"] as string ?? String.Empty;

string aHomePage = aReader["HomePage"] as string ?? String.Empty;

Supplier aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

aSupplierList.Add(aSupplier);

}

aConnection.Close();

return aSupplierList;

}

//Gets a list of all suppliers

public List<Supplier> GetSuppliers()

{

// open the connection

aConnection.Open();

List<Supplier> aSupplierList = new List<Supplier>();

if (aConnection.State == ConnectionState.Open)

{

aCommand.CommandText = "SELECT [SupplierId],[CompanyName],[ContactName], ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax, HomePage FROM Suppliers";

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aSupplierId = (int)aReader["SupplierId"];

string aCompanyName = (string)aReader["CompanyName"];

string aContactName = (string)aReader["ContactName"];

string aContactTitle = (string)aReader["ContactTitle"];

string aAddress = (string)aReader["Address"];

string aCity = (string)aReader["City"];

string aRegion = aReader["Region"] as string ?? String.Empty;

string aPostalCode = aReader["PostalCode"] as string ?? String.Empty;

string aCountry = (string)aReader["Country"];

string aPhone = (string)aReader["Phone"];

string aFax = aReader["Fax"] as string ?? String.Empty;

string aHomePage = aReader["HomePage"] as string ?? String.Empty;

Supplier aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

aSupplierList.Add(aSupplier);

}

}

aConnection.Close();

return aSupplierList;

}

//Gets a list of products filtered by supplier Id

public List<Product> GetProductsBySupplier(String theId)

{

// open the connection

aConnection.Open();

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

if (aConnection.State == ConnectionState.Open)

{

aCommand.CommandText = "SELECT ProductId, ProductName, SupplierId, CategoryId, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM Products WHERE SupplierId = " + theId;

OleDbDataReader aReader = aCommand.ExecuteReader();

while (aReader.Read())

{

int aProductId = (int)aReader["ProductId"];

string aProductName = (string)aReader["ProductName"];

int aSupplierId = (int)aReader["SupplierId"];

int aCategoryId = (int)aReader["CategoryId"];

string aQuantityPerUnit = (string)aReader["QuantityPerUnit"];

double aUnitPrice = (double)(decimal)aReader["UnitPrice"];

int aUnitsInStock = (short)aReader["UnitsInStock"];

int aUnitsOnOrder = (short)aReader["UnitsOnOrder"];

int aReorderLevel = (short)aReader["ReorderLevel"];

bool isDiscontinued = (bool)aReader["Discontinued"];

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aProductList.Add(aProduct);

}

}

aConnection.Close();

return aProductList;

}

//Gets a list of products filtered by the Unit Price

public List<Product> GetProductsByUnitPrice(double min, double max)

{

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

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

OleDbCommand aCommand = aConnection.CreateCommand();

aCommand.CommandText = "SELECT [ProductId],[ProductName],[SupplierId], CategoryId, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued FROM Products WHERE UnitPrice BETWEEN " + min + " AND " + max; ;

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read()){

int aProductId = (int) aReader["ProductId"];

string aProductName = (string) aReader["ProductName"];

int aSupplierId = (int) aReader["SupplierId"];

int aCategoryId = (int)aReader["CategoryId"];

string aQuantityPerUnit = (string) aReader["QuantityPerUnit"];

double aUnitPrice = Convert.ToDouble(aReader["UnitPrice"] as decimal? ?? 9999);

int aUnitsInStock = (short)aReader["UnitsInStock"];

int aUnitsOnOrder = (short)aReader["UnitsOnOrder"];

int aReorderLevel = (short)aReader["ReorderLevel"];

bool isDiscontinued = (bool)aReader["Discontinued"];

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aProductList.Add(aProduct);

}

aConnection.Close();

return aProductList;

}

}

}

The next set of classes are used to build an abstract factory that would allow a used to load data either using an Odbc DataSet, Odbc DataReader, OleDb DataReader and OleDb DataSet. An abstract factory produces implementations of each interface and then each interface creates a different type of object. The abstract factory in this case uses regular factories to implement the interfaces. These factories to separate data that changes frequently from base classes to help keep code functional.

/\*Written By: Sarah Jastrzab

\* 12/7/2014

\* This code is an abstract factory. This abstract class creates two of

\* the abstract mehtods that will later be implemented using a

\* Factory Pattern\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Inheritance1.Models

{

//Ckkass can't be implemented

public abstract class NWAbstractFactory

{

//Two abstract methods

public abstract INWDataReader CreateReader(String aSQL);

public abstract INWDataSet CreateDataSet(String aSQL);

}

}

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This is the interface for a DataReader Factory. It contains the information that will

\* not change for the factory. This class would only be updated if there was no other way

\* to upgrade the software.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data;

using System.Data.OleDb;

using System.Data.Sql;

using System.Data.Odbc;

namespace Inheritance1.Models

{

//Interface that will later be implemented to create objects

public interface INWDataReader

{

IDataReader aDataReader

{

//Methods

get;

set;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This is the interface for a DataSet Factory. It contains the information that will not

\* for the factory. This class would only be updated if there was no other way to upgrade

\* the software.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data;

using System.Data.OleDb;

using System.Data.Sql;

using System.Data.Odbc;

namespace Inheritance1.Models

{

//Interface that will create an object

public interface INWDataSet

{

DataSet aDataSet

{

get;

set;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This is the Odbc data for a DataReader Factory. It implements the INWDataReader

\* interface. Using these methods, this concrete class is able to have its specialized

\* data changed without risking destroying the whole application

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data;

using System.Data.OleDb;

using System.Data.Sql;

using System.Data.Odbc;

namespace Inheritance1.Models

{

public class NWOdbcDataReader : INWDataReader

{

//implementing the INWDataReader

private IDataReader aReader;

public IDataReader aDataReader

{

get { return this.aReader; }

set { this.aReader = value; }

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This is the Odbc data for a DataReader Factory. It implements the INWDataReader

\* interface. Using these methods, this concrete class is able to have its specialized

\* data changed without risking destroying the whole application

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data;

using System.Data.OleDb;

using System.Data.Sql;

using System.Data.Odbc;

namespace Inheritance1.Models

{

public class OdbcDataSet : INWDataSet

{

//implementing the INWDataSet

DataSet aSet = new DataSet();

public DataSet aDataSet

{

get { return this.aSet; }

set { this.aSet = value; }

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This is the OleDb data for a DataReader Factory. It implements the INWDataReader

\* interface. Using these methods, this concrete class is able to have its specialized

\* data changed without risking destroying the whole application.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data;

using System.Data.OleDb;

using System.Data.Sql;

using System.Data.Odbc;

namespace Inheritance1.Models

{

public class NWOleDbDataReader : INWDataReader

{

//implementing the INWDataReader

private IDataReader aReader;

public IDataReader aDataReader

{

get { return this.aReader; }

set { this.aReader = value; }

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This is the OleDb data for a DataSet Factory. It implements the INWDataSet

\* interface. Using these methods, this concrete class is able to have its specialized

\* data changed without risking destroying the whole application.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data;

using System.Data.OleDb;

using System.Data.Sql;

using System.Data.Odbc;

namespace Inheritance1.Models

{

public class NWOleDbDataSet : INWDataSet

{

DataSet aSet = new DataSet();

//implementing the INWDataSet

public DataSet aDataSet

{

get { return this.aSet; }

set { this.aSet = value; }

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This factory takes the NWDataSet and the NWReader which implement interaces and

\* overrides the methods in the abstract factory. The functioning code and the code that

\* is most often updated would go in this class. From here the user will have the ability

\* to use an oledb dataset or an oledb datareader for getting data from a database.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data;

using System.Data.OleDb;

using System.Data.Sql;

using System.Data.Odbc;

namespace Inheritance1.Models

{

public class NWOleDBFactory : NWAbstractFactory

{

private NWOleDbDataReader aNWReader = new NWOleDbDataReader();

private NWOleDbDataSet aNWDataSet = new NWOleDbDataSet();

public override INWDataReader CreateReader(string aSQL)

{

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// run the SQL statement

aCommand.CommandText = aSQL;

OleDbDataReader aReader = aCommand.ExecuteReader();

aNWReader.aDataReader = aReader;

return aNWReader;

}

public override INWDataSet CreateDataSet(string aSQL)

{

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

anAdapter.Fill(aNWDataSet.aDataSet);

return aNWDataSet;

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 12/4/2014

\* This factory takes the NWDataSet and the NWReader which implement interaces and

\* overrides the methods in the abstract factory. The functioning code and the code that

\* is most often updated would go in this class. From here the user will have the ability

\* to use an odbc dataset or an odbc datareader for getting data from a database.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data;

using System.Data.OleDb;

using System.Data.Sql;

using System.Data.Odbc;

namespace Inheritance1.Models

{

public class NWOdbcFactory : NWAbstractFactory

{

private NWOleDbDataReader aNWReader = new NWOleDbDataReader();

private NWOleDbDataSet aNWDataSet = new NWOleDbDataSet();

public override INWDataReader CreateReader(string aSQL)

{

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// run the SQL statement

aCommand.CommandText = aSQL;

OleDbDataReader aReader = aCommand.ExecuteReader();

aNWReader.aDataReader = aReader;

return aNWReader;

}

public override INWDataSet CreateDataSet(string aSQL)

{

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Sarah\Documents\Northwind.accdb";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

anAdapter.Fill(aNWDataSet.aDataSet);

return aNWDataSet;

}

}

}

Controllers are used to create objects. The controllers in the ASP.NET MVC framework take the information stored in the models and then send it to the view which displays it on the web. There are multiple controllers in this project to implement different parts of the application. The controllers all contain methods that correspond to a view which the information is displayed.

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Home Controller.

\* It loads the home page and allows the user to navigate the site. The home page also

\* has all the methods that use the factories to display information. Each method

\* corresponds to a view.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Inheritance1.Models;

using System.Data.OleDb;

using System.Data.Odbc;

using System.Data.Sql;

using System.Data.Odbc;

using System.Data;

namespace Inheritance1.Controllers

{

public class HomeController : Controller

{

//

// GET: /Home/

//Calls the home page

public ActionResult Index()

{

return View();

}

//Calls the factory page whichs prints category information

public ActionResult Factory()

{

//Create a list

List<Category> aListOfCategories = new List<Category>();

//Implement the factories and create the reader

**NWAbstractFactory** aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader("Select \* from categories");

IDataReader aReader = aNWReader.aDataReader;

//Loops through information from the database

while (aReader.Read())

{

int aCategoryId = (int)aReader["CategoryId"];

string aCategoryName = (string)aReader["CategoryName"];

string aDescription = (string)aReader["Description"];

Category aCategory = new Category(aCategoryId, aCategoryName, aDescription);

aListOfCategories.Add(aCategory);

}

ViewBag.aListOfcategories = aListOfCategories;

return View();

}

//Calls the factory page whichs prints product information

public ActionResult ProductFactory()

{

//Create a List

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

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader("Select \* from products");

IDataReader aReader = aNWReader.aDataReader;

//Loop through database information

while (aReader.Read())

{

int aProductId = (int)aReader["ProductId"];

string aProductName = (string)aReader["ProductName"];

int aSupplierId = (int)aReader["SupplierId"];

int aCategoryId = (int)aReader["CategoryId"];

string aQuantityPerUnit = (string)aReader["QuantityPerUnit"];

double aUnitPrice = (double)(decimal)aReader["UnitPrice"];

int aUnitsInStock = aReader["UnitsInStock"] as Int16? ?? 0;

int aUnitsOnOrder = (short)aReader["UnitsOnOrder"];

int aReorderLevel = (short)aReader["ReorderLevel"];

bool isDiscontinued = (bool)aReader["Discontinued"];

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice, aUnitsInStock, aUnitsOnOrder, aReorderLevel, isDiscontinued);

aListOfProducts.Add(aProduct);

aListOfProducts.Add(aProduct);

}

ViewBag.aListOfproducts = aListOfProducts;

return View();

}

//Calls the factory page whichs prints shipper information

public ActionResult ShipperFactory()

{

//Create a List

List<Shipper> aListOfShippers = new List<Shipper>();

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader("Select \* from shippers");

IDataReader aReader = aNWReader.aDataReader;

//Loop through database information

while (aReader.Read())

{

int aShipperId = (int?)aReader["ShipperID"] ?? -1;

string aCompanyName = (string)aReader["CompanyName"] ?? "n/a";

string aPhone = (string)aReader["Phone"] ?? "n/a";

Shipper aShipper = new Shipper(aShipperId, aCompanyName, aPhone);

aListOfShippers.Add(aShipper);

}

ViewBag.aListOfshippers = aListOfShippers;

return View();

}

//Calls the factory page whichs prints customer information

public ActionResult CustomerFactory()

{

//Create a List

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

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader("Select \* from customers");

IDataReader aReader = aNWReader.aDataReader;

//Loop through database information

while (aReader.Read())

{

string aCustomerId = (string)aReader["CustomerID"] ?? "n/a";

string aCompanyName = (string)aReader["CompanyName"] ?? "n/a";

string aContactName = (string)aReader["ContactName"] ?? "n/a";

string aContactTitle = (string)aReader["ContactTitle"] ?? "n/a";

string aAddress = (string)aReader["Address"] ?? "n/a";

string aCity = (string)aReader["City"] ?? "n/a";

string aRegion = aReader["Region"] as string ?? "n/a";

string aPostalCode = aReader["PostalCode"] as string ?? "n/a";

string aCountry = (string)aReader["Country"] ?? "n/a";

string aPhone = (string)aReader["Phone"] ?? "n/a";

string aFax = aReader["Fax"] as string ?? "n/a";

Customer aCustomer = new Customer(aCustomerId, aCompanyName, aContactName, aContactTitle, aAddress, aCity, aRegion, aPostalCode, aCountry, aPhone, aFax);

aListOfCustomers.Add(aCustomer);

}

ViewBag.aListOfcustomers = aListOfCustomers;

return View();

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Category Controller.

\* It pulls data fromListed created in the Database loader and sends them to the view.

\* By using parameters we are able to filter the data.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Inheritance1.Models;

namespace Inheritance1.Controllers

{

public class CatgoryController : Controller

{

//Sends a List of Categories to the view

public ActionResult Index()

{

CategoryGateWay aCatGate= new CategoryGateWay();

List<Category> aCategoryList = aCatGate.GetCategories();

ViewBag.Categories = aCategoryList;

return View();

}

//Sends a list of Categories filtered by ID and a list of Products filtered by category

public ActionResult Category(string ID)

{

CategoryGateWay aCatGate = new CategoryGateWay();

ProductGateway aProGate = new ProductGateway();

List<Product> aListOfProductsByCategory = aProGate.GetProductsByCategory(ID);

List<Category> aListOfCategories = aCatGate.GetCategoriesById(ID);

var products = from p in aListOfProductsByCategory

orderby p.ProductName ascending

select p;

ViewBag.Products = products;

ViewBag.Categories = aListOfCategories[0];

return View();

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Product Controller.

\* It pulls data from lists created in the Database loader and sends them to the view.

\* By using parameters we are able to filter the data.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Inheritance1.Models;

namespace Inheritance1.Controllers

{

public class ProductController : Controller

{

//

// GET: /Product/

//Prints out all Product Names

public ActionResult SeeAll()

{

ProductGateway aProGate = new ProductGateway();

// OleDbUtilityLoader aConnection = new OleDbUtilityLoader();

List<Product> aProductList = aProGate.GetProducts();

var products = from p in aProductList

orderby p.ProductName ascending

select p;

ViewBag.Products = products;

return View();

}

//Prints all fields for a Product Name

public ActionResult Details(String ID)

{

ProductGateway aProGate = new ProductGateway();

List<Product> aProductList = aProGate.GetProductByID(ID);

ViewBag.Products = aProductList[0];

return View();

}

//Displays an HTML form that allows user to input price

public ActionResult PriceForm()

{

return View();

}

//Displays the filtered through products by prices

public ActionResult Price(int min, int max)

{

ProductGateway aProGate = new ProductGateway();

List<Product> aListOfProductsByPrice = aProGate.GetProductsByUnitPrice(min, max);

ViewBag.Products = aListOfProductsByPrice;

return View();

}

}

}

/\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This is the Supplier Controller.

\* It pulls data from lists created in the Database loader and sends them to the view.

\* By using parameters we are able to filter the data.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Inheritance1.Models;

namespace Inheritance1.Controllers

{

public class SupplierController : Controller

{

//Gets a lists of Suppliers

public ActionResult Index()

{

SupplierGateway aSupGate = new SupplierGateway();

List<Supplier> aSupplierList = aSupGate.GetSuppliers();

ViewBag.Suppliers = aSupplierList;

return View();

}

//Gets a list of products filtered by supplier

public ActionResult ProductsBySupplier(string ID)

{

SupplierGateway aSupGate = new SupplierGateway();

ProductGateway aProGate = new ProductGateway();

List<Product> aListOfProductsBySupplier = aProGate.GetProductsBySupplier(ID);

List<Supplier> aListOfSuppliers = aSupGate.GetSuppliersById(ID);

var products = from p in aListOfProductsBySupplier

orderby p.ProductName ascending

select p;

ViewBag.Products = products;

ViewBag.Suppliers = aListOfSuppliers[0];

return View();

}

//Calls the details of the Category

public ActionResult Details(string ID)

{

ProductGateway aProGate = new ProductGateway();

ViewBag.Categories = aProGate.GetProductsByCategory(ID)[0];

return View();

}

}

}

The following classes are views. In ASP.NET MVC the views correspond to a method in the controller. They are a mix of C# and Html. Each view displays a different page of the website and does something a litle different. This set of bbiews uses the Table Gateways to create a site that would allow a user to browse Northwind’s product offerings.

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page allows is the home page and allows the user to go to different pages to filter products. It also goes to the faactory pages.

\*/-->

@{

ViewBag.Title = "Home";

}

<h1>NorthWind Database</h1>

<!--Links take user to one of multiple pages for filtering-->

<h2>

Filter by

</h2>

<li><a href="/Catgory">By Category</a></li>

<li><a href="/Product/PriceForm">By Price</a></li>

<li><a href="/Supplier">By Supplier</a></li>

<li><a href="/Product/SeeAll">See All</a></li>

<li><a href="/Home/Factory">See Categories Using Factories</a></li>

<li><a href="/Home/ProductFactory">See Products Using Factories</a></li>

<li><a href="/Home/ShipperFactory">See Shippers Using Factories</a></li>

<li><a href="/Home/CustomerFactory">See Customers Using Factories</a></li>

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page allows the used to select a category.

\* The user is then taken to the Category page to see a list of products in that Category.

\*/-->

@model Inheritance1.Models.Category

@{

ViewBag.Title = "Categories";

}

<h2>Browsing By Category</h2>

<h3>Choose a Category</h3>

<!--Allows user to choose a category to view the products in that category-->

@foreach (var c in ViewBag.Categories)

{

<ul>

<li><a href="~/Catgory/Category/@c.CategoryId">@c.CategoryName</a></li>

<li><p>@Html.Raw(c.Description + "<br />")</p></li>

</ul>

}

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page allows the used to select a product that is in a certain Category.

\* The user is then taken to the Details page.

\*/-->

@{

ViewBag.Title = "Category";

}

<h2>@ViewBag.Categories.CategoryName</h2>

<!--Goes through the product list and finds a product with the requested ID -->

@foreach (var c in ViewBag.Products)

{

<ul class="nav">

<li><a href="~/Product/Details/@c.ProductId">@c.ProductName</a></li>

</ul>

}

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page prints the information about the selected product

\*/-->

@{

}

<!--All the details of the desired product-->

<h2>@ViewBag.Products.ProductName</h2>

<p>ProductId: @ViewBag.Products.ProductId</p>

<p>CategoryId: @ViewBag.Products.CategoryId</p>

<p>SupplierId: @ViewBag.Products.SupplierId</p>

<p>Quantity per Unit: @ViewBag.Products.QuantityPerUnit</p>

<p>Unit Price: @ViewBag.Products.UnitPrice</p>

<p>Units In Stock: @ViewBag.Products.UnitsInStock</p>

<p>Units On Order: @ViewBag.Products.UnitsOnOrder</p>

<p>Reorder Level: @ViewBag.Products.ReorderLevel</p>

<p>Discontinued: @ViewBag.Products.Discontinued</p>

<a href="~/Product/Details/@ViewBag.Products.ProductId" />

<a href="~/Product/Details/@ViewBag.Products.ProductId" />

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page allows the use to select a product that is in a range of Unit Prices

\* The user is then taken to the Details page.

\*/-->

@model Inheritance1.Models.Product

@{

ViewBag.Title = "Price";

}

<!--Prints out the products fallig in the desired Unit Price-->

<h2>ProductsPrice</h2>

@foreach (var c in ViewBag.Products)

{

<ul class="nav">

<li><a href="~/Product/Details/@c.ProductId">@c.ProductName</a></li>

</ul>

}

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page allows the user enter the range of prices in an html form

\* The user is then taken to the Price page.

\*/-->

@{

ViewBag.Title = "PriceForm";

}

<!--Form for UnitPrice input-->

<h2>Enter a Price</h2>

<form id="min/max" action="~/Product/Price" method="get">

Minimum Unit Price: <input type="text" id="min" name="min" /> <br />

Maximum Unit Price: <input type="text" id="max" name="max" /> <br />

<button type="submit" value="Submit">Submit</button>

</form>

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page allows the used to select a product for a master list.

\* The user is then taken to the Details page.

\*/-->

@model Inheritance1.Models.Product

@{

ViewBag.Title = "SeeAll";

}

<!--All prodcts are printed-->

<h2>All Products</h2>

<h3>View Details</h3>

@foreach (var p in ViewBag.Products)

{

<ul>

<li><a href="~/Product/Details/@p.ProductId">@p.ProductName</a></li>

</ul>

}

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page allows the user to select a Shipper they what to see products for.

\* The user is then taken to the ProductsByShipper page.

\*/-->

@model Inheritance1.Models.Supplier

@{

ViewBag.Title = "Index";

}

<!--Company names are printed out and customer can make a decision for filtering products based on company-->

<h2>Browsing By Supplier</h2>

<h3>Choose a Supplier</h3>

@foreach (var s in ViewBag.Suppliers)

{

<ul>

<li><a href="~/Supplier/ProductsBySupplier/@s.SupplierId">@s.CompanyName</a></li>

<!--<li><p>//Html.Raw(s.ToString() + "<br />")</p></li> -->

</ul>

}

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page allows the used to select a product that is in a certain Supplier.

\* The user is then taken to the Details page.

\*/-->

@{

ViewBag.Title = "ProductsBySupplier";

}

<h2>@ViewBag.Suppliers.CompanyName</h2>

<!--Desired Product is selected-->

@foreach (var c in ViewBag.Products)

{

<ul class="nav">

<li><a href="~/Product/Details/@c.ProductId">@c.ProductName</a></li>

</ul>

}

The following classes are views. In ASP.NET MVC the views correspond to a method in the controller. They are a mix of C# and Html. Each view displays a different page of the website and does something a litle different. This set of views uses the factories to display some information from database tables on the web. It uses the same home controller.

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page prints out information from customers table in the northwind database. The data comes from the controller and it shows the functionaltiy of the abstract factory.

\*/-->

@{

ViewBag.Title = "CustomerFactory";

}

<!--Loop through Customers-->

@foreach (var c in ViewBag.aListOfcustomers)

{

@Html.Raw(c.ToString());

<br />

}

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page prints out information from categories table in the northwind database. The data comes from the controller and it shows the functionaltiy of the abstract factory.

\*/-->

@{

ViewBag.Title = "Factory";

}

<h2>FactoryView</h2>

<h3>Categories</h3>

<!--Loop through Categories-->

@foreach (var c in ViewBag.aListOfcategories)

{

@Html.Raw(c.ToString());

<br />

}

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page prints out information from productss table in the northwind database. The data comes from the controller and it shows the functionaltiy of the abstract factory.

\*/-->

@{

ViewBag.Title = "ProductFactory";

}

<h2>Products</h2>

<!--Loop through Products-->

@foreach (var c in ViewBag.aListOfproducts)

{

@Html.Raw(c.ToString());

<br />

}

<!--

\*Written By:Sarah Jastrzab

\* Date: 10/15/2014

\* This page prints out information from shippers table in the northwind database. The data comes from the controller and it shows the functionaltiy of the abstract factory.

\*/-->

@{

ViewBag.Title = "ShipperFactory";

}

<h2>ShipperFactory</h2>

<!--Loop through Shippers-->

@foreach (var c in ViewBag.aListOfshippers)

{

@Html.Raw(c.ToString());

<br />

}