|  |  |
| --- | --- |
|  | Northwind ASP.NET MVC |
|  |  |
|  | Carrier, Christopher T  CISY 6503, Alfred State, Object Orientated Programming  5/4/14 |

**Table of Contents**

Screen Shots ……………………………………………………………………... 4

Project Details ……………………………………………………………………... 21

Abstract Company ……………………………………………………………………... 22

BrokenRule ……………………………………………………………………... 33

Broken Rules ……………………………………………………………………... 34

Category ……………………………………………………………………... 36

Product ……………………………………………………………………... 39

Shipper ……………………………………………………………………... 48

Customer ……………………………………………………………………... 50

Supplier ……………………………………………………………………... 52

CategoryIdentityMap ……………………………………………………………………... 54

ProductIdentityMap ……………………………………………………………………... 56

ShipperIdentityMap ……………………………………………………………………... 58

SupplierIdentityMap ……………………………………………………………………... 60

CategoryConnection ……………………………………………………………………... 62

ProductConnection ……………………………………………………………………... 67

ShipperConnection ……………………………………………………………………... 72

SupplierConnection ……………………………………………………………………... 75

NWAbstractFactory ……………………………………………………………………... 81

INWDataReader ……………………………………………………………………... 82

INWDataSet ……………………………………………………………………... 83

NWOdbcDataReader ……………………………………………………………………... 84

NWOdbcDataSet ……………………………………………………………………... 85

NWOdbcFactory ……………………………………………………………………... 86

NWOleDbDataReader ……………………………………………………………………... 88

NWOleDbDataSet ……………………………………………………………………... 89

NWOleDbFactory ……………………………………………………………………... 90

OleDbUtilityLoader ……………………………………………………………………... 92

FactoryCategoryConnection ……………………………………………………………………... 98

FactoryProductConnection …………………………………………………………………....... 100

**Table of Contents**

FactoryShipperConnection ……………………………………………………………………... 104

FactorySupplierConnection ……………………………………………………………………... 106

DBConnection ……………………………………………………………………... 109

HomeController ……………………………………………………………………... 123

FactoryController ……………………………………………………………………... 128

Index.cshtml ……………………………………………………………………... 131

CategoryIndex.cshtml ……………………………………………………………………... 132

CategoryFile.cshtml ……………………………………………………………………... 133

CategoryFilter.cshtml ……………………………………………………………………... 134

ByCategory.cshtml ……………………………………………………………………... 135

SupplierIndex.cshtml ……………………………………………………………………... 136

SupplierFile.cshtml ……………………………………………………………………... 137

SupplierFilter.cshtml ……………………………………………………………………... 138

BySupplier.cshtml ……………………………………………………………………... 139

ShipperIndex.cshtml ……………………………………………………………………... 140

ShipperFile.cshtml ……………………………………………………………………... 141

ShipperFilter.cshtml ……………………………………………………………………... 142

ByShipper.cshtml ……………………………………………………………………... 143

ProductIndex.cshtml ……………………………………………………………………... 144

ProductFile.cshtml ……………………………………………………………………... 145

ProductFilter.cshtml ……………………………………………………………………... 146

ByProduct.cshtml ……………………………………………………………………... 147

FactoryIndex.cshtml ……………………………………………………………………... 148

CategoryFactory.cshtml ……………………………………………………………………... 149

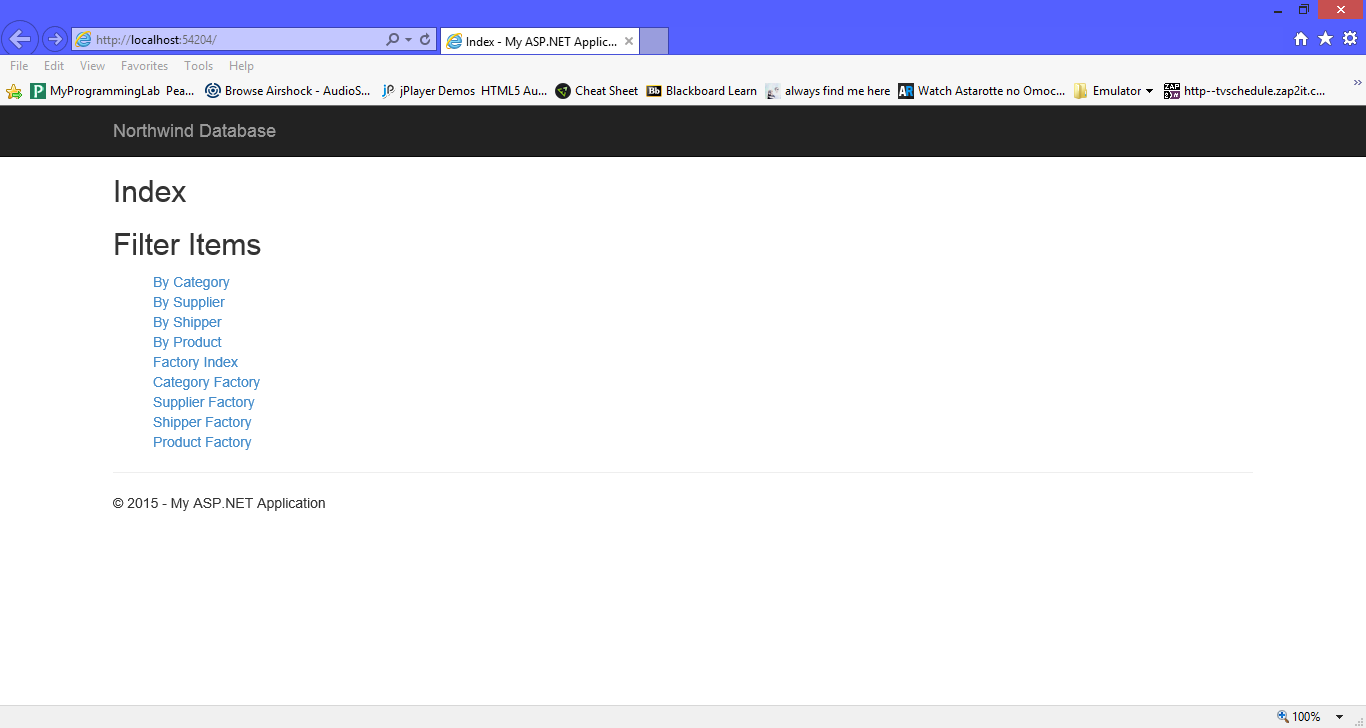
SupplierFactory.cshtml ……………………………………………………………………... 150

ShipperFactory.cshtml ……………………………………………………………………... 151

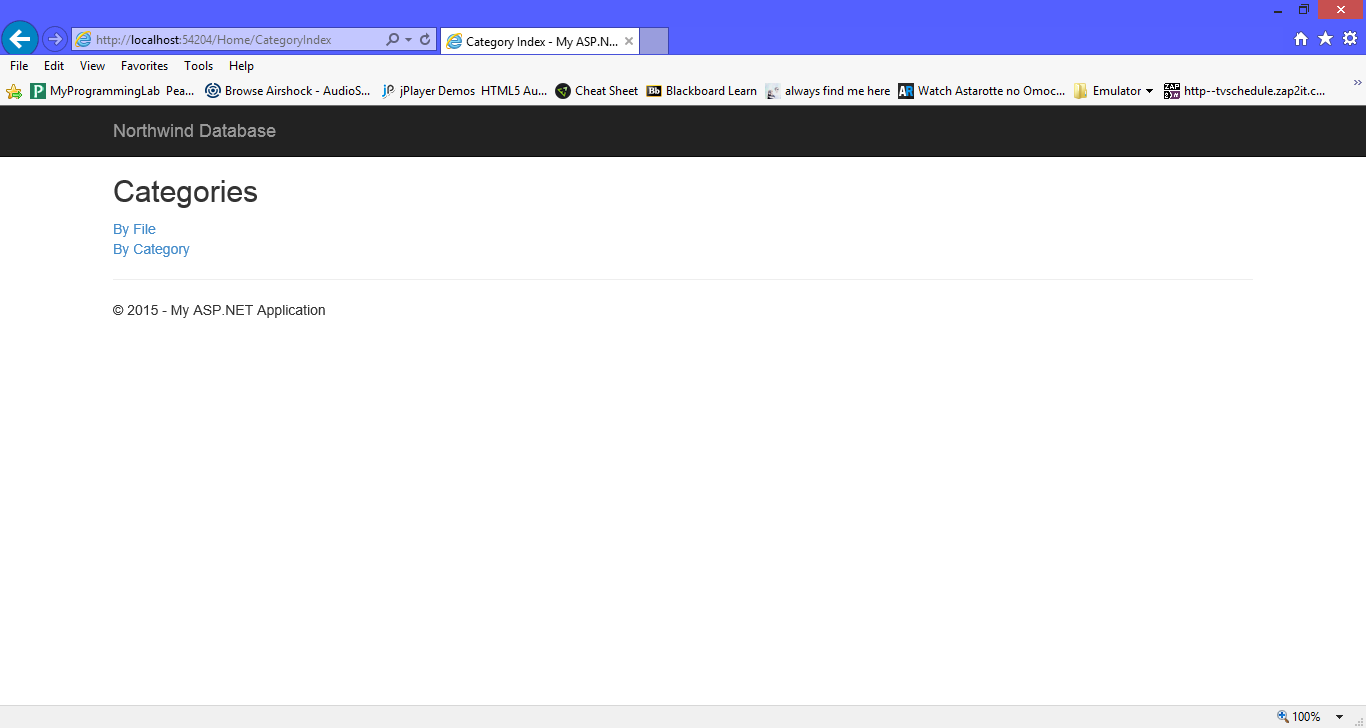
ProductFactory.cshmtl ……………………………………………………………………... 152

**Screen Shots**

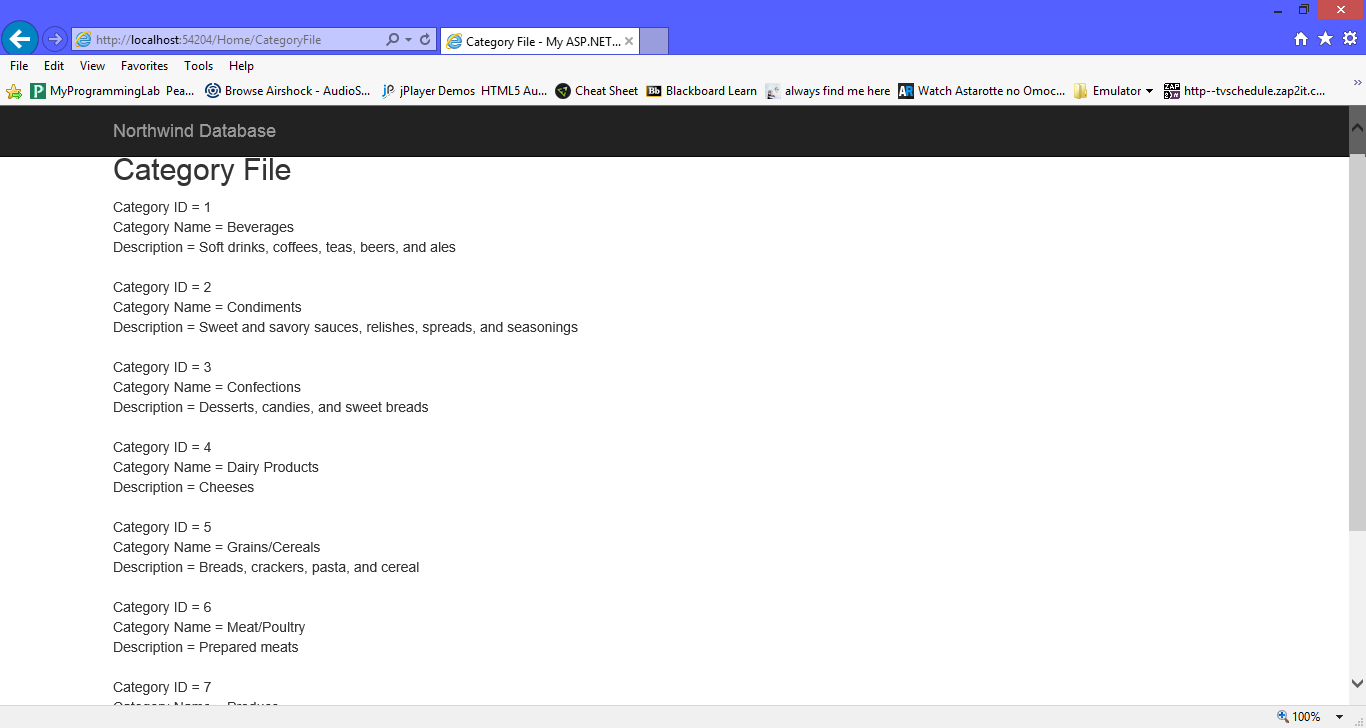
Index (Home Page)



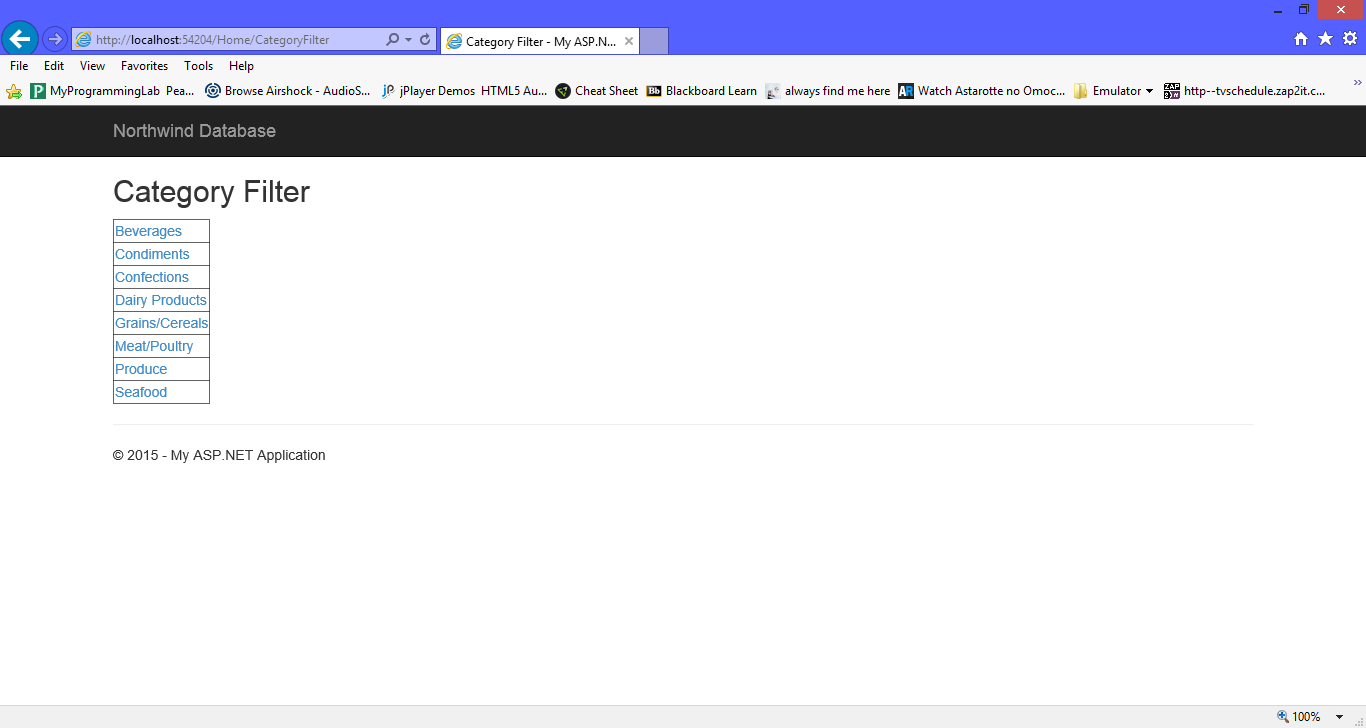
By Category Home



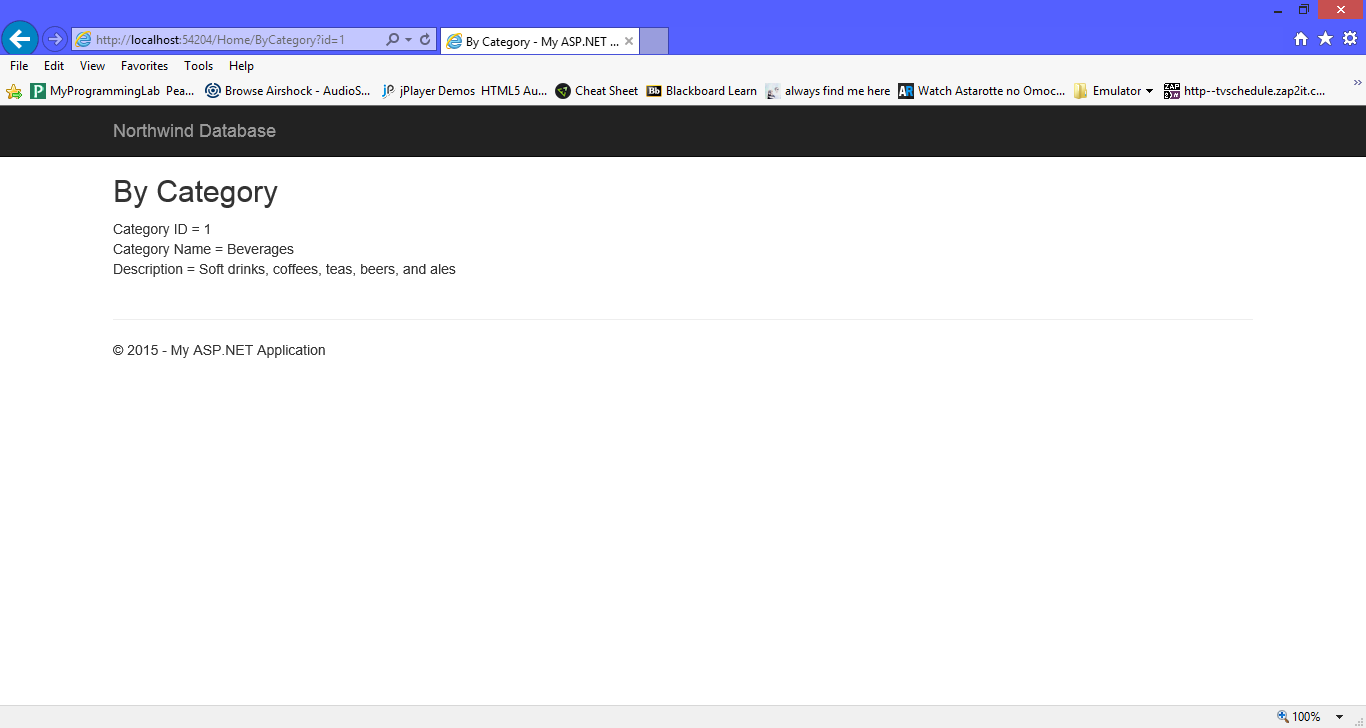
By Category File List



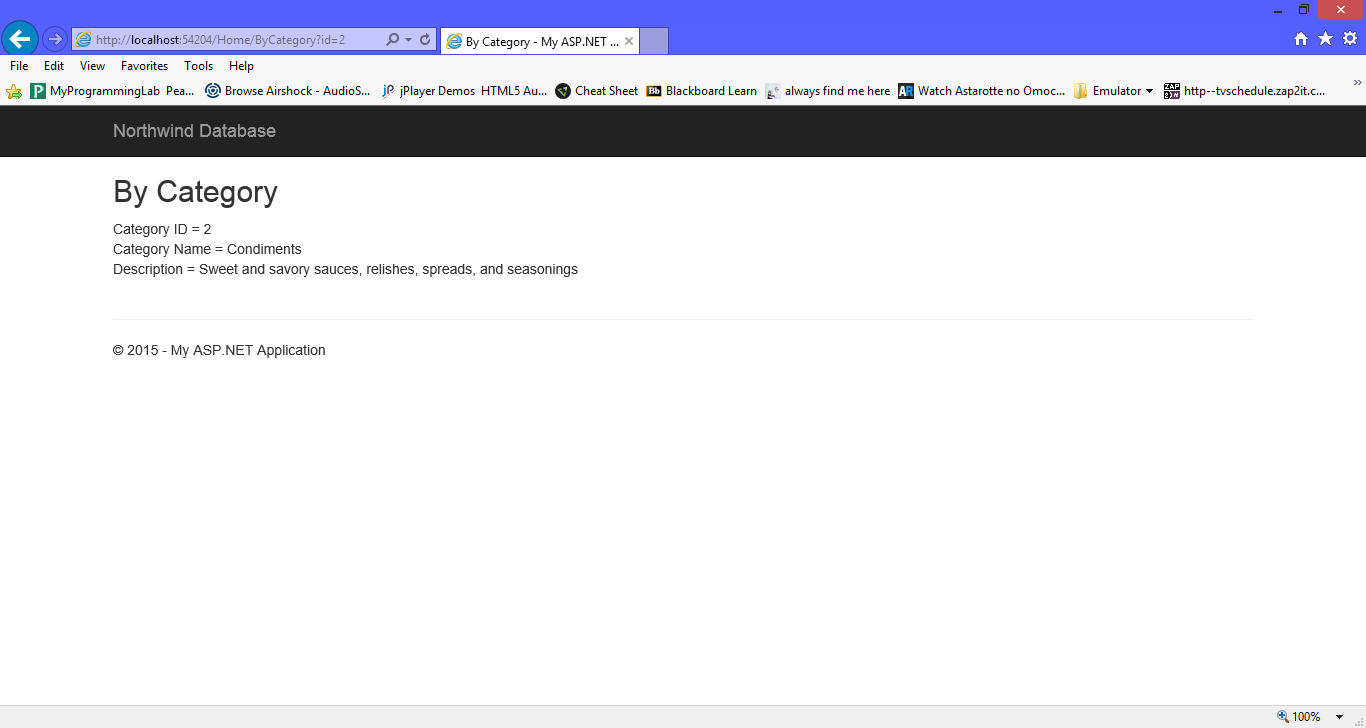
By Category Filter



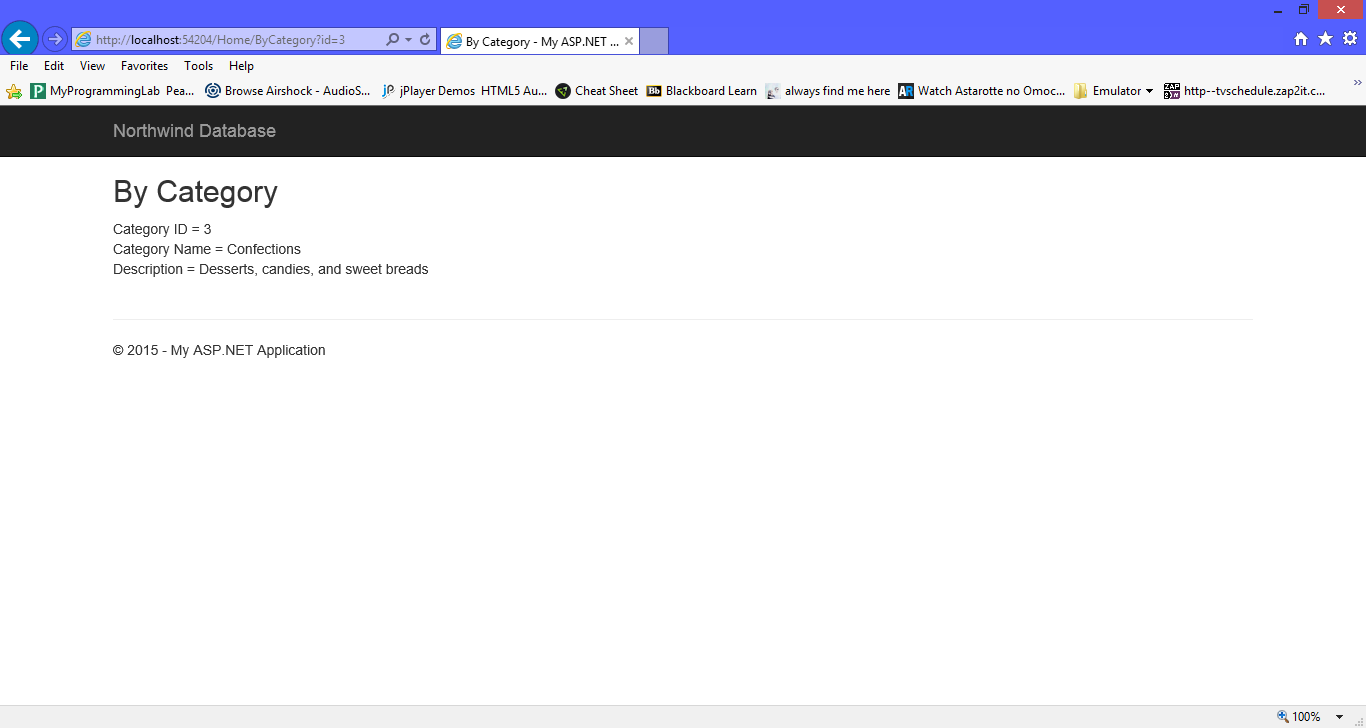
Beverage Filter



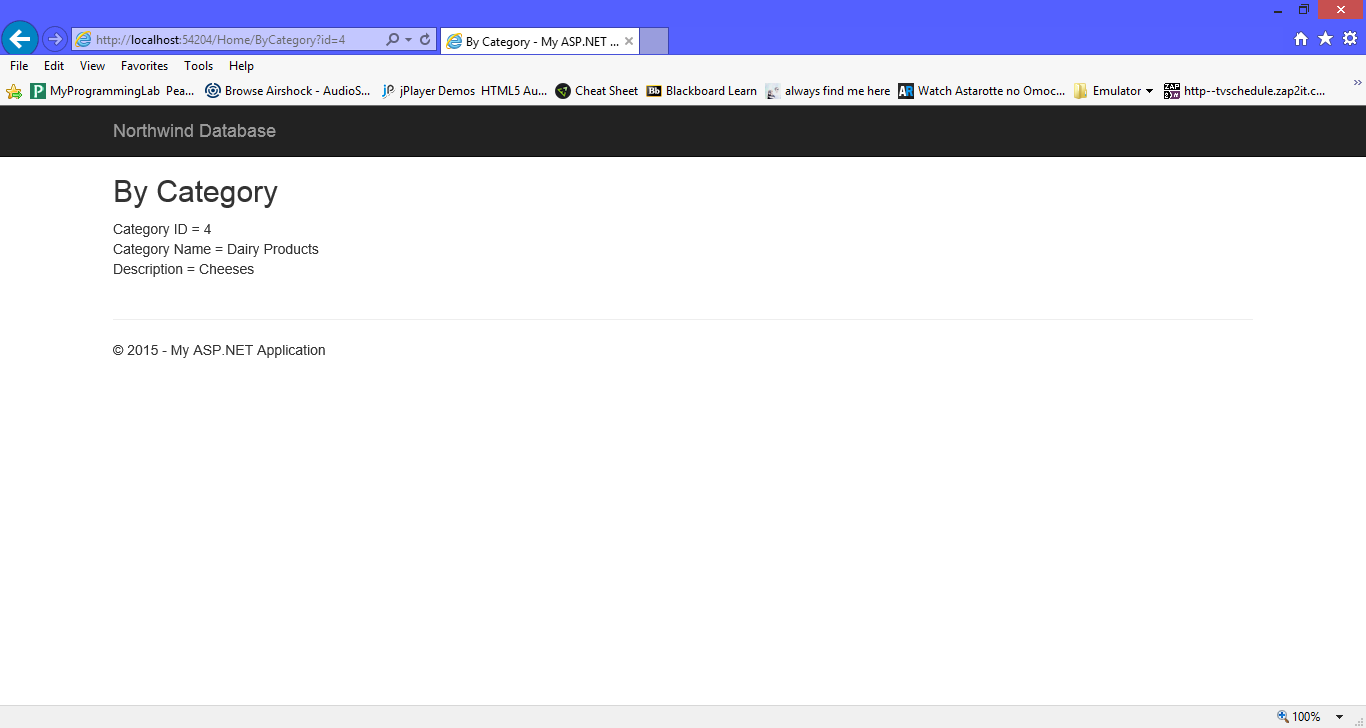
Condiments Filter



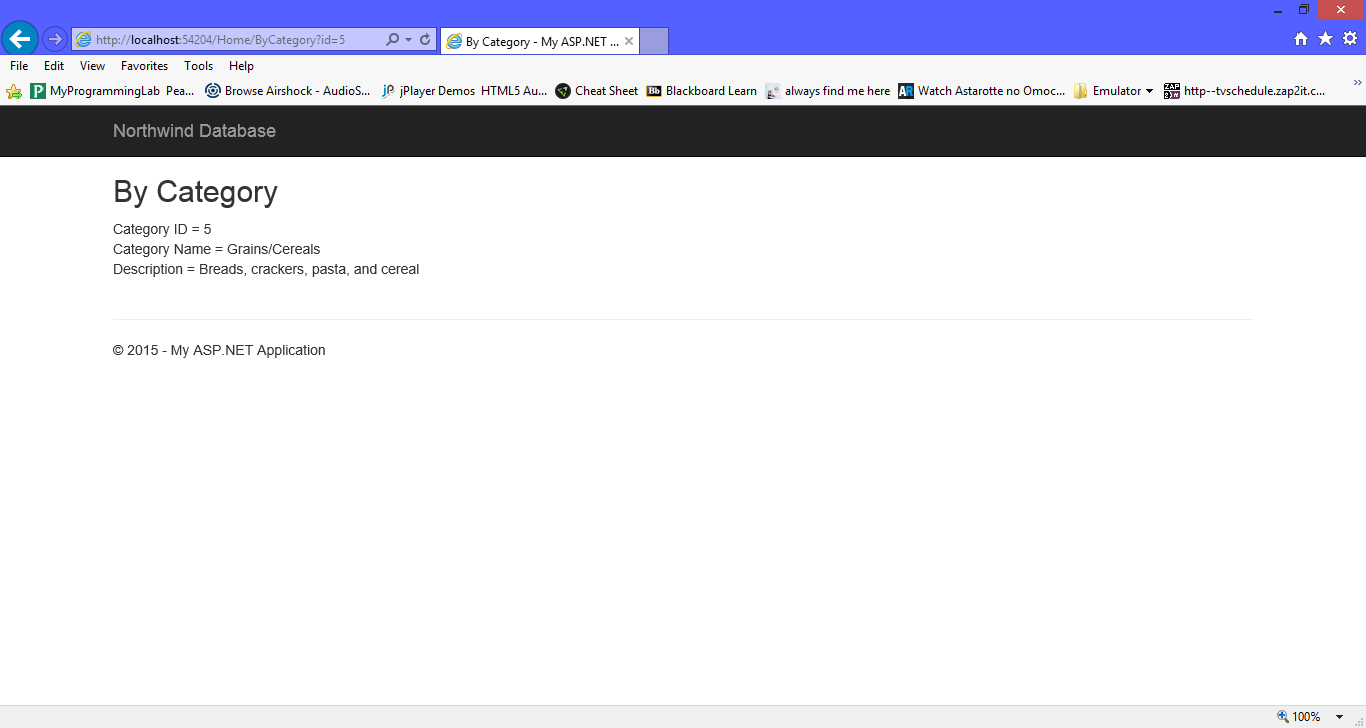
Confection Filter



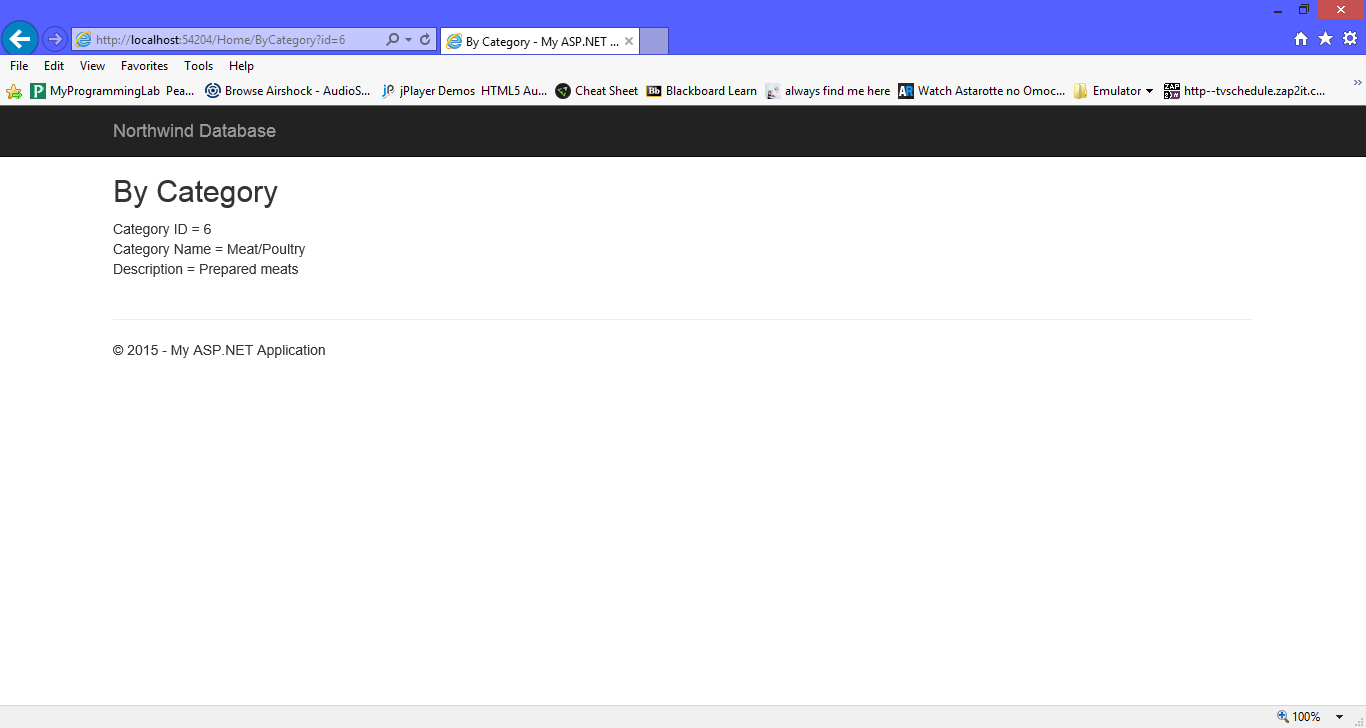
Dairy Product Filter



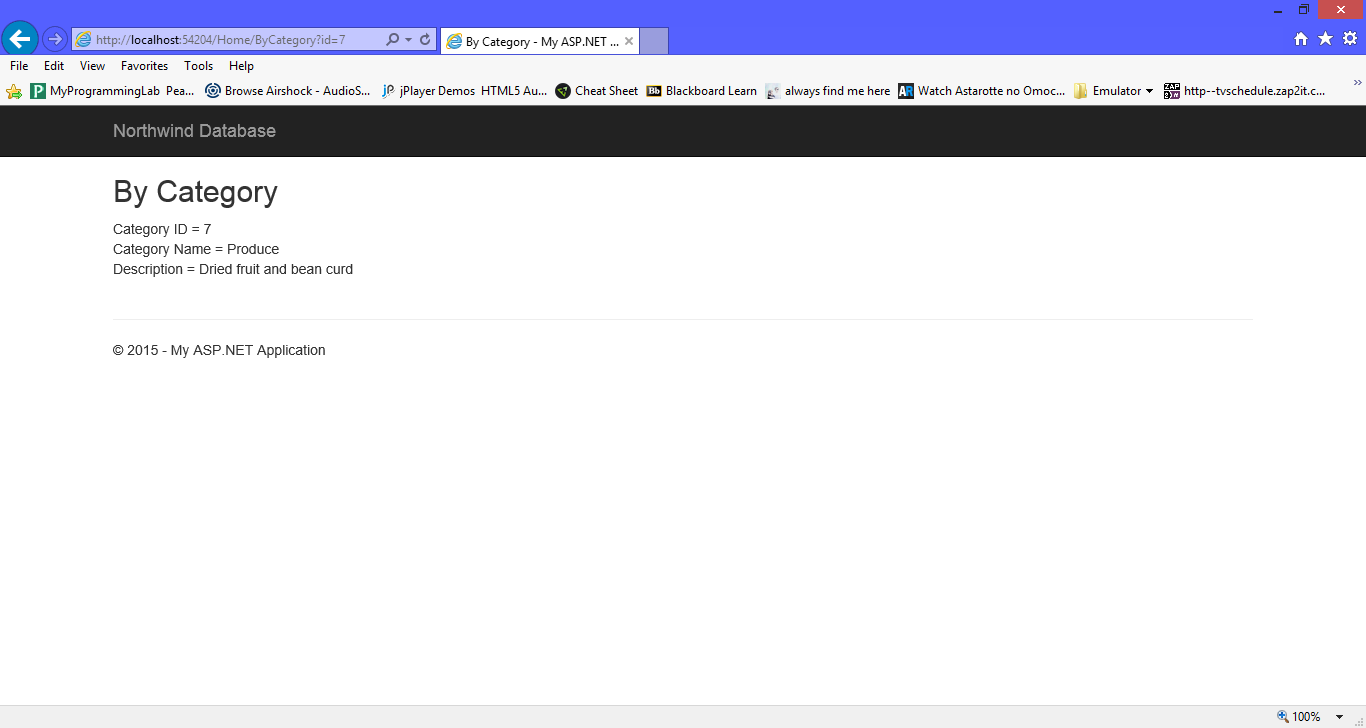
Grains/ Cereal Filter



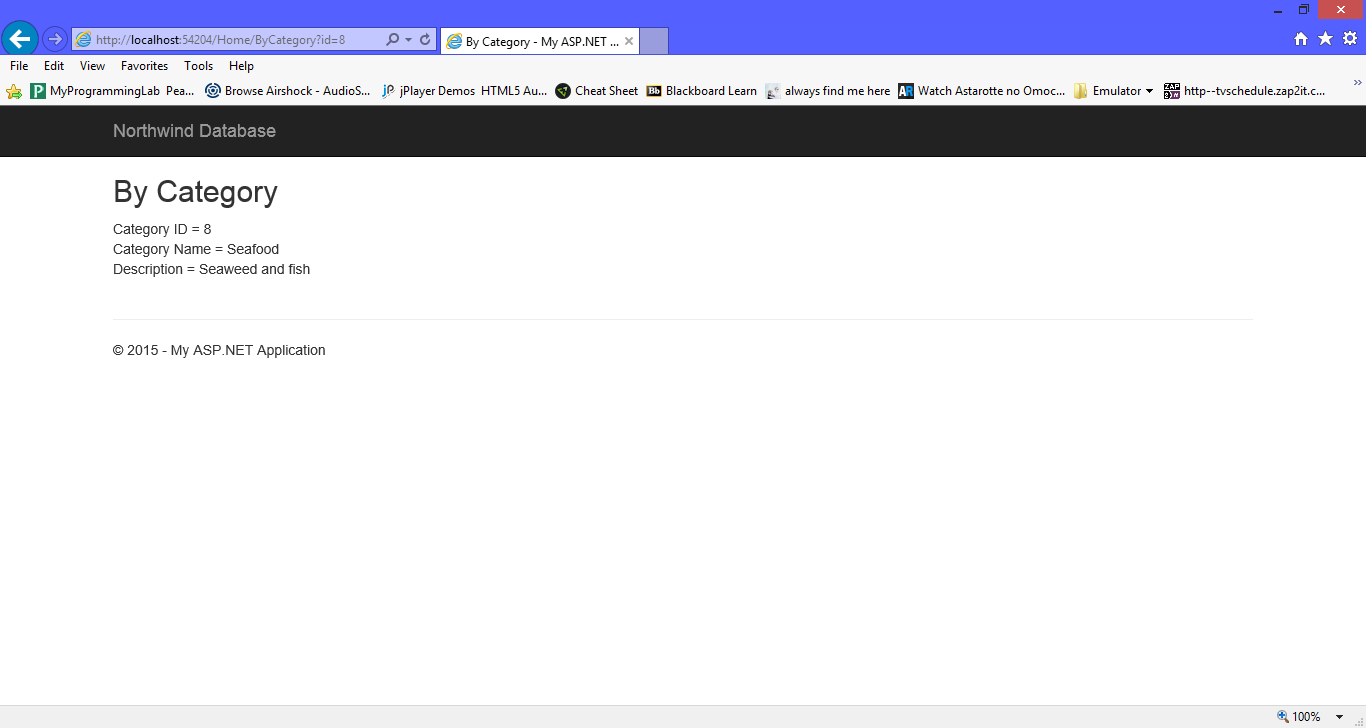
Meat / Poultry Filter



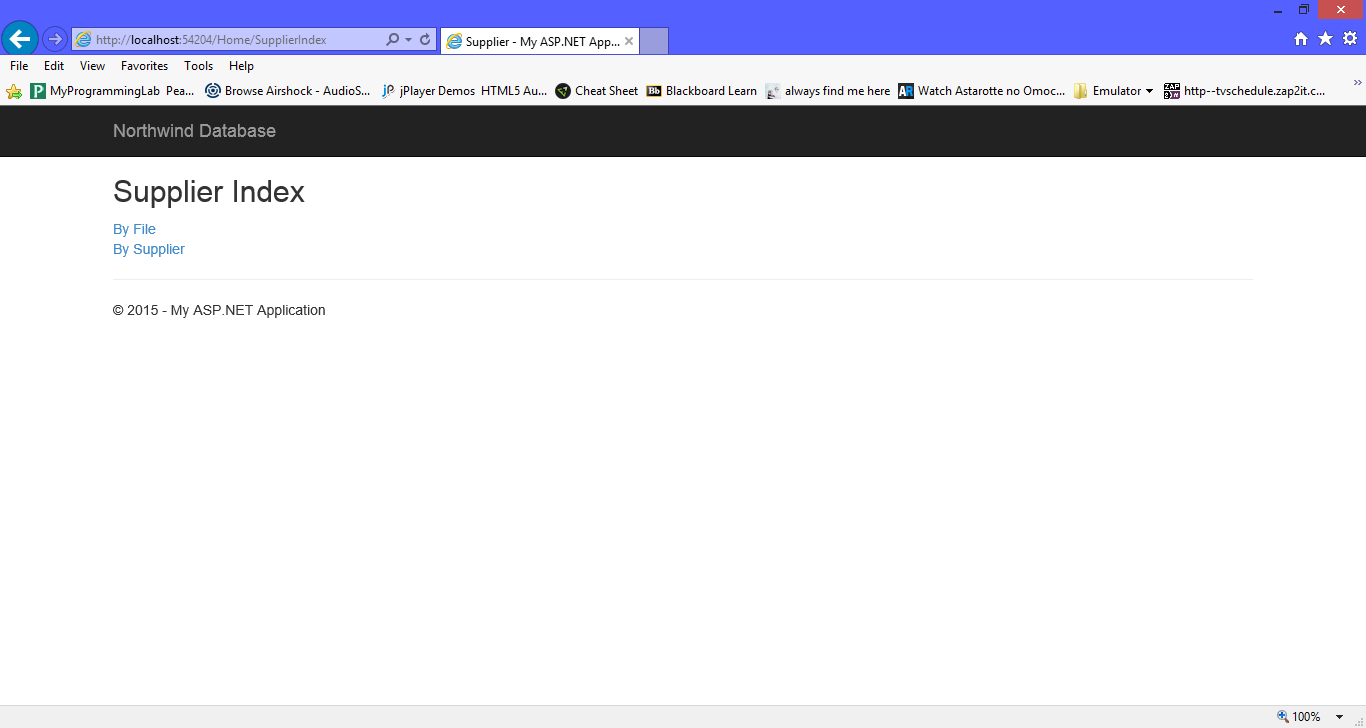
Product Filter



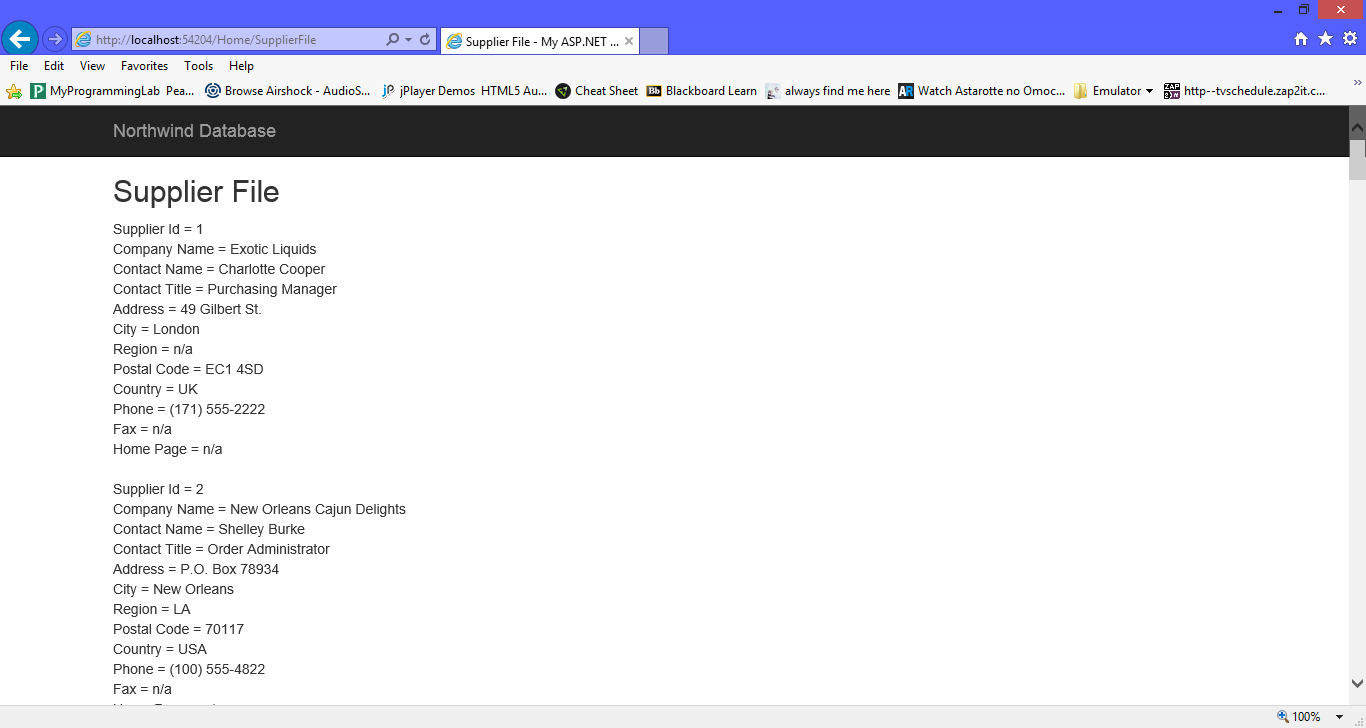
Seafood Filter



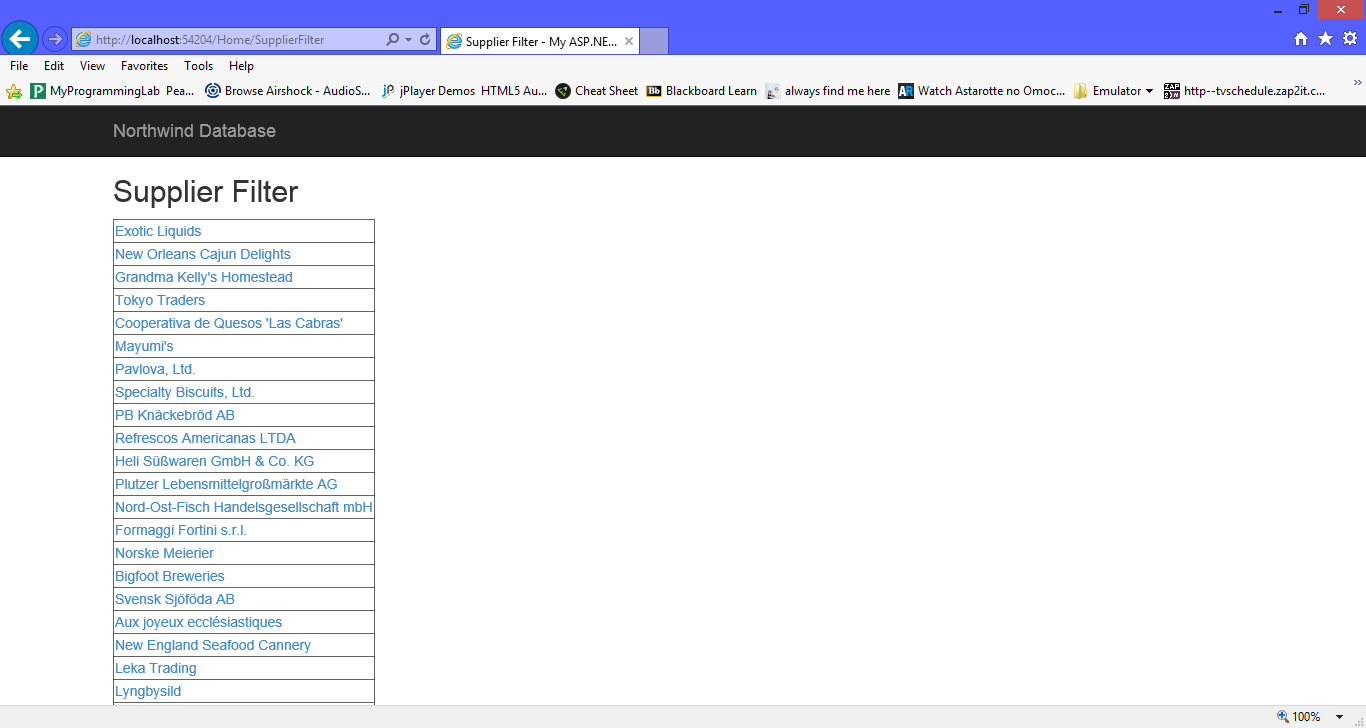
By Supplier Index



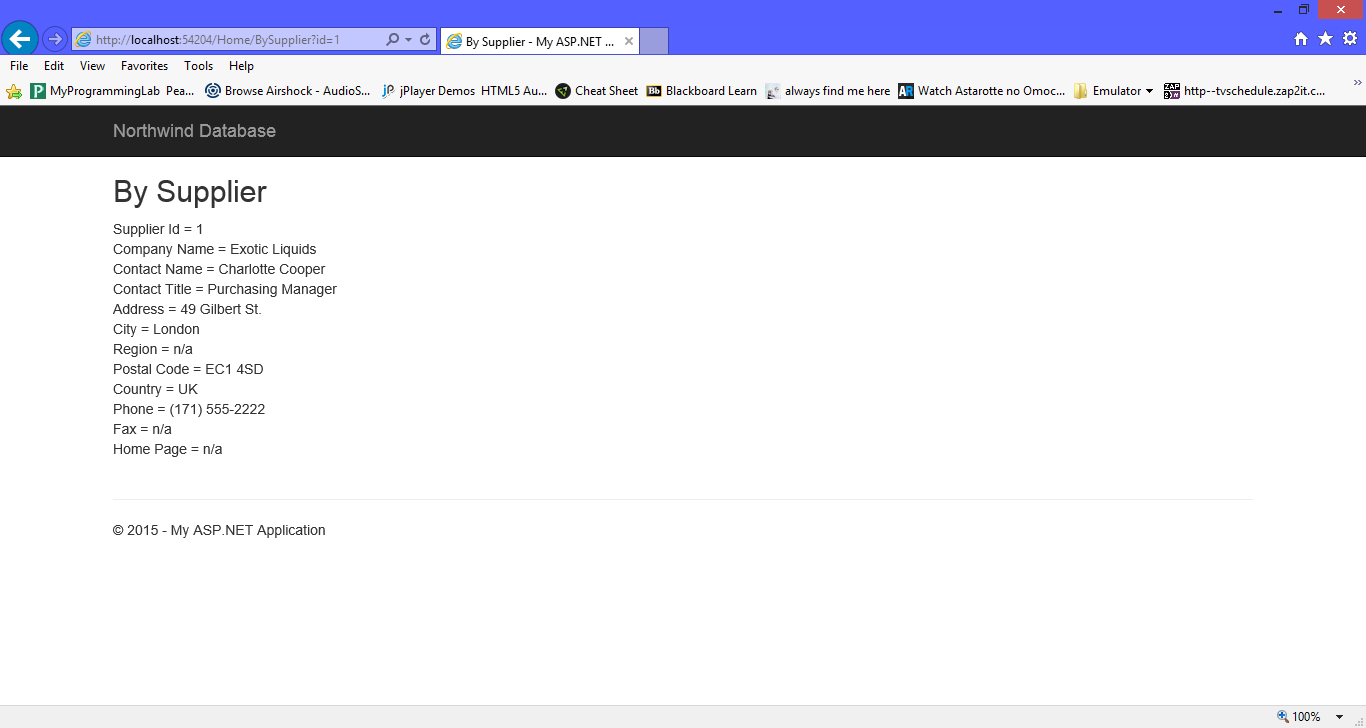
By Supplier File List



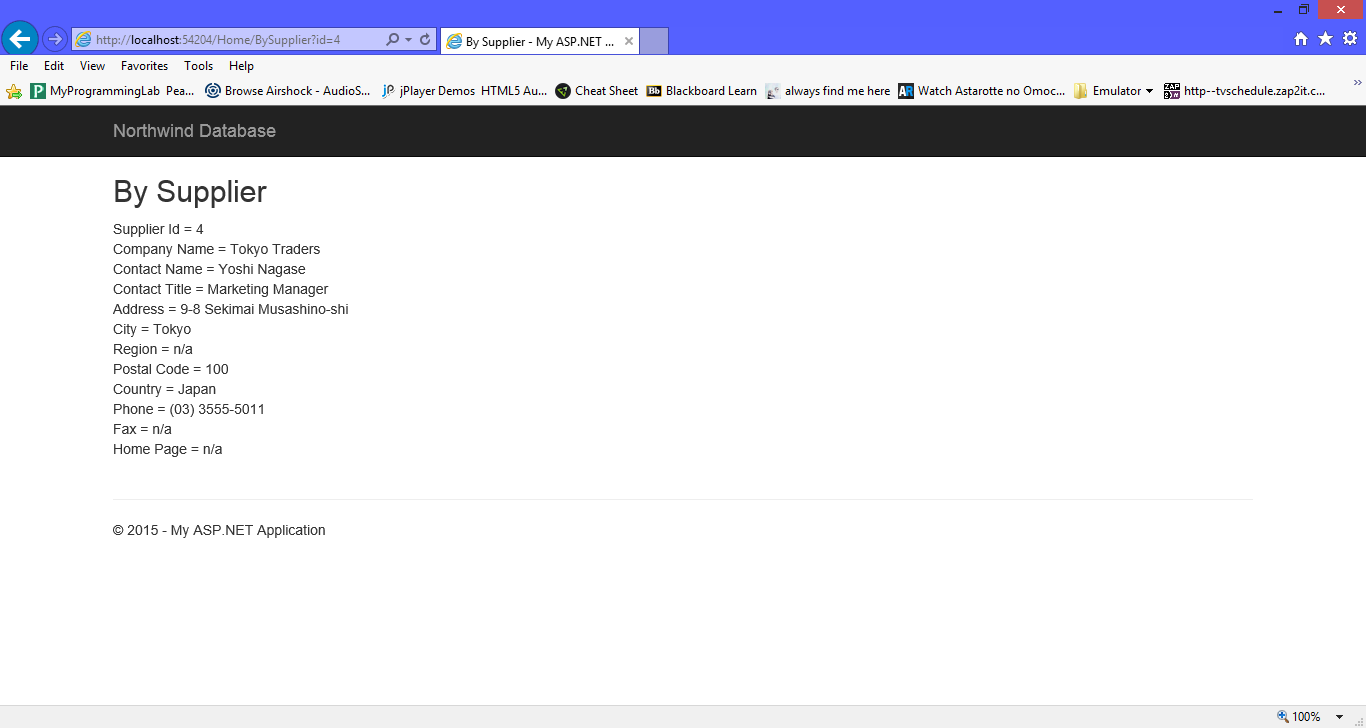
By Supplier Filter



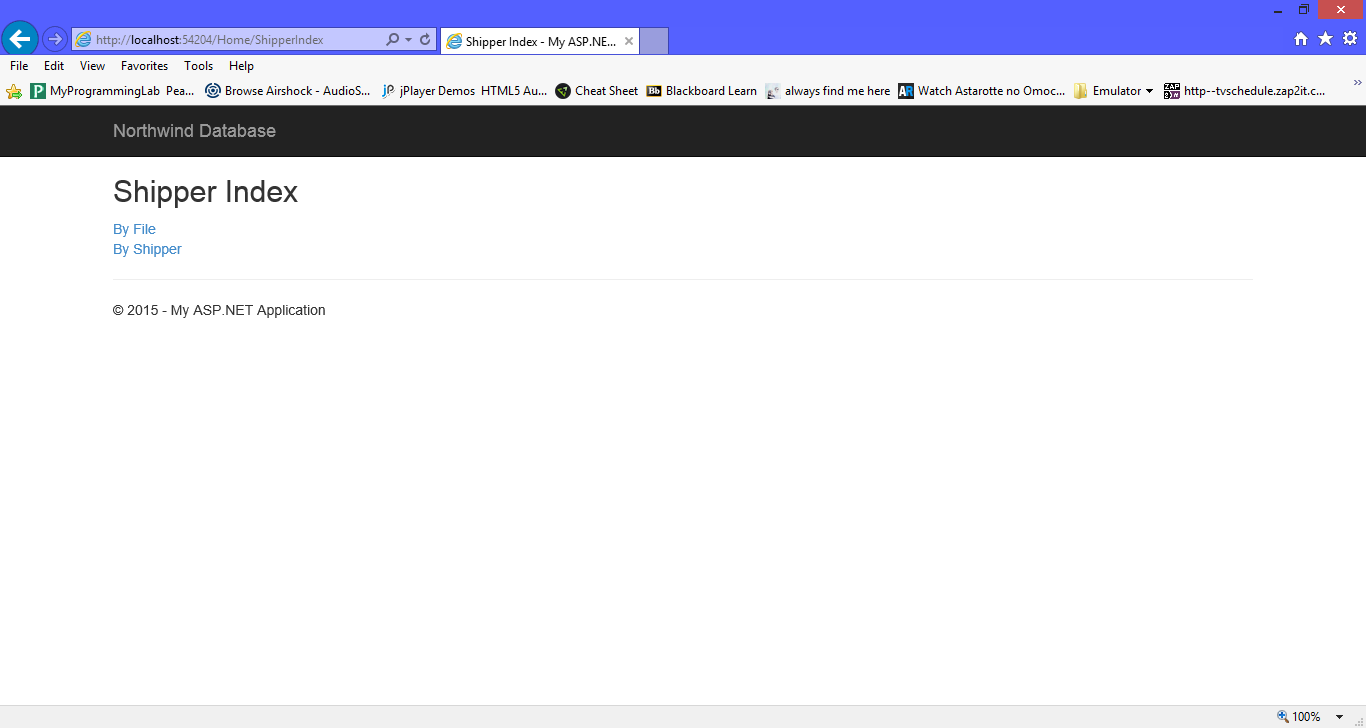
By Supplier Exotic Liquids Filter



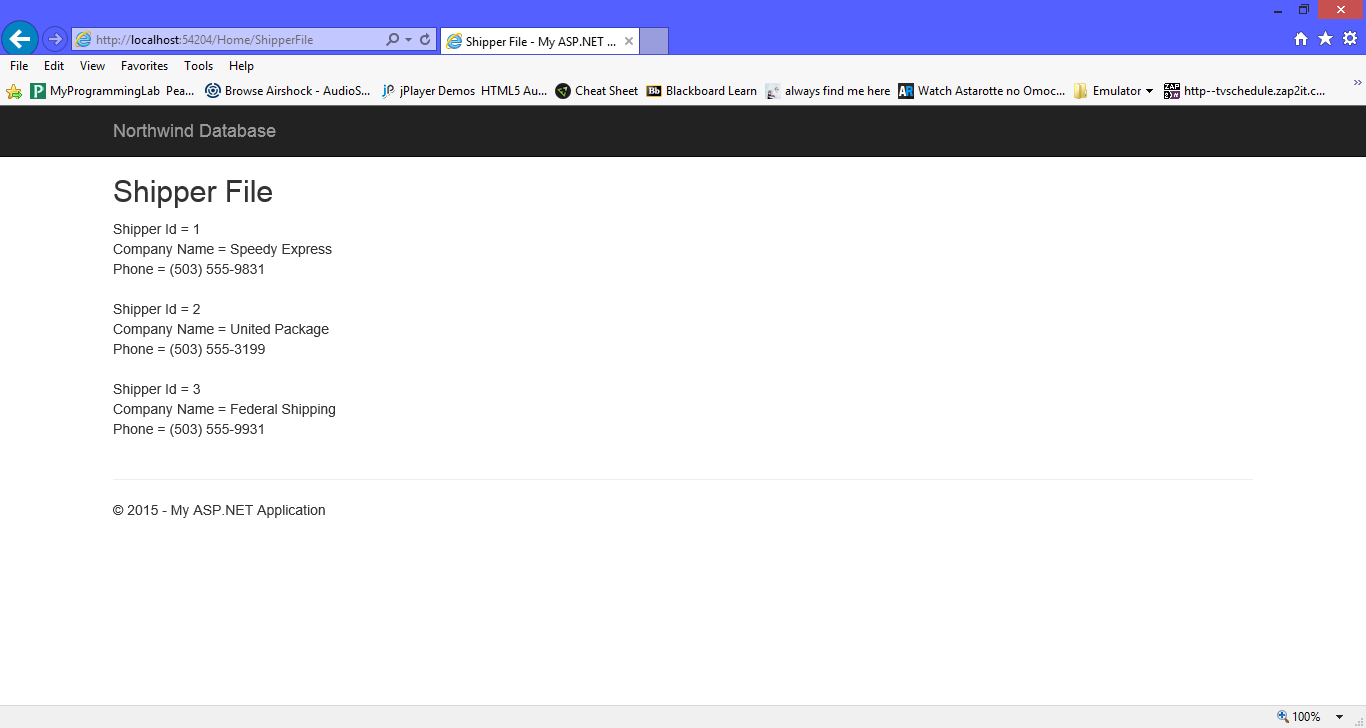
By Supplier Tokyo Trader Filter



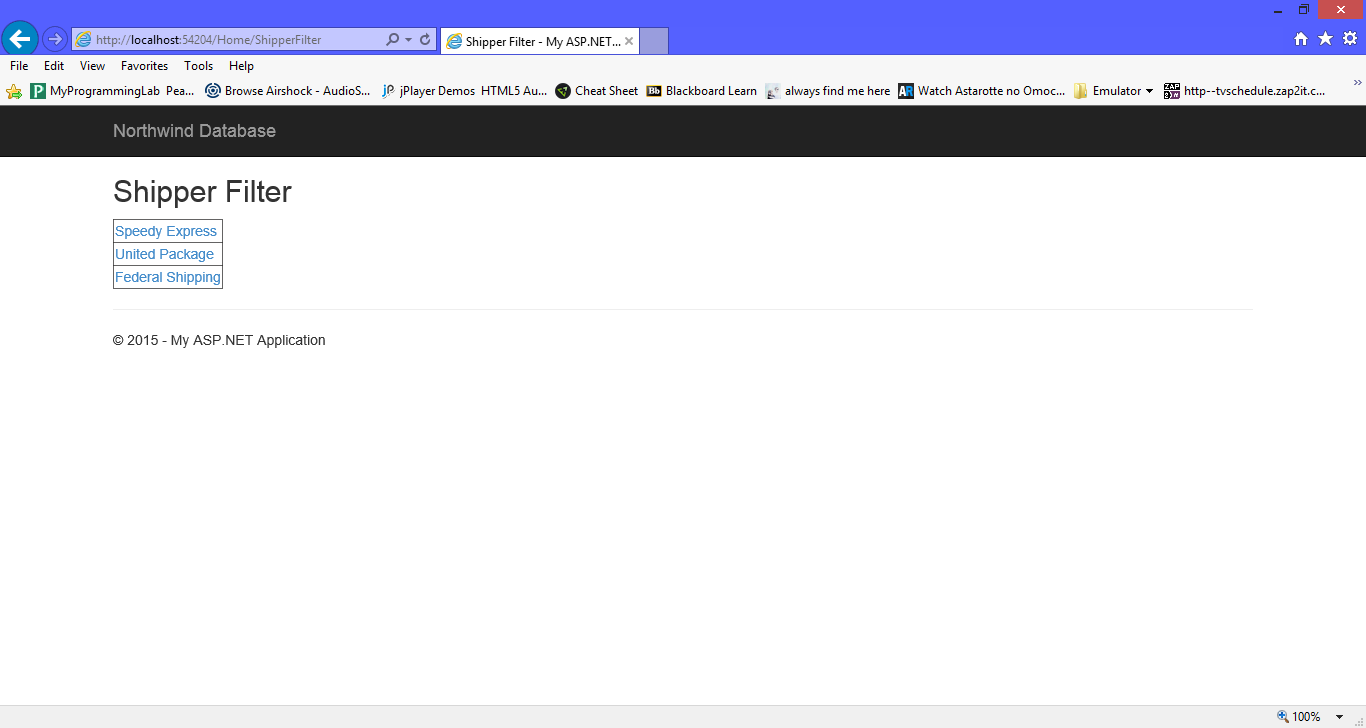
By Shipper Index



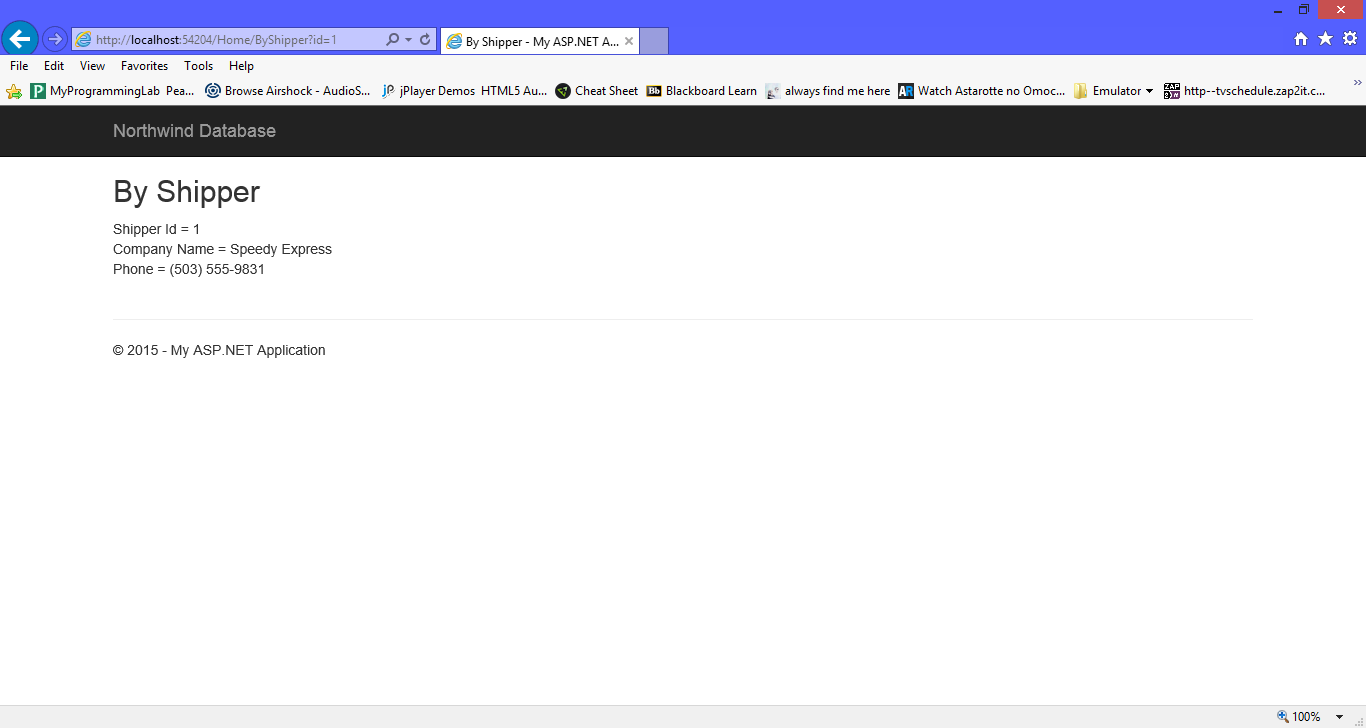
By Shipper File List



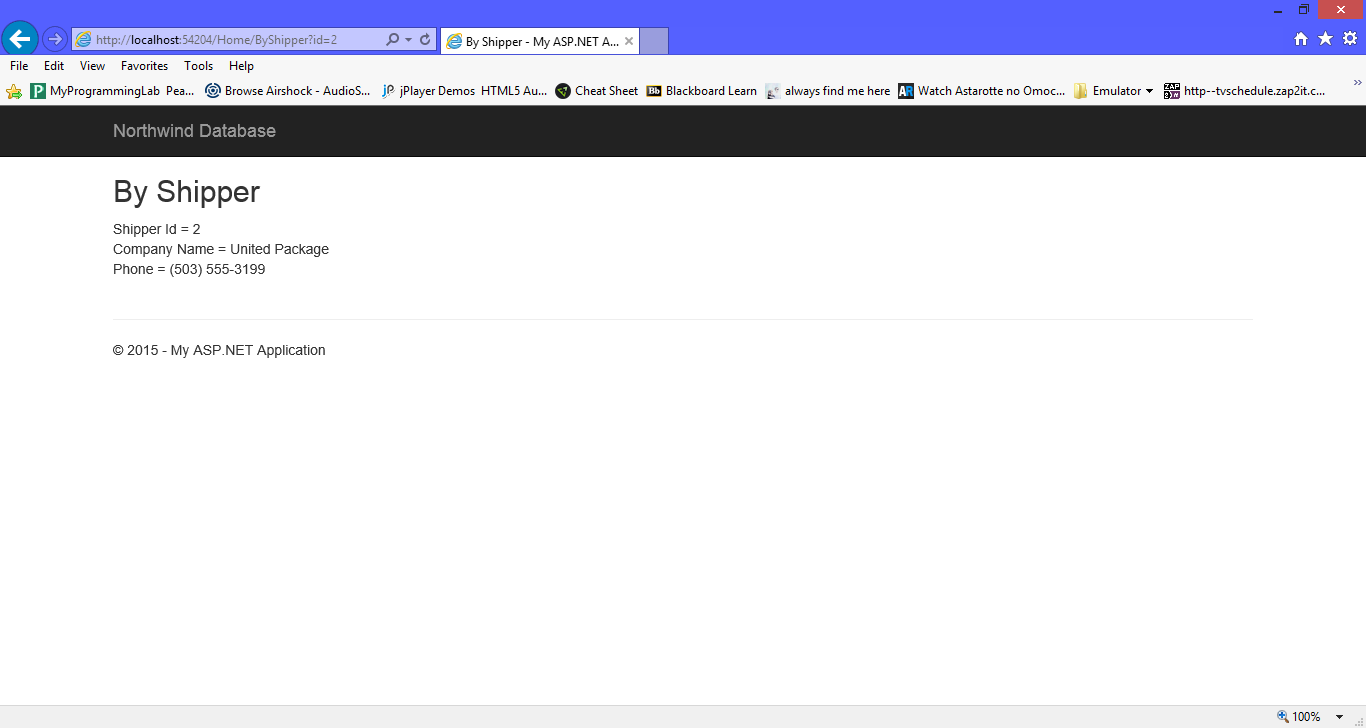
By Shipper Filter



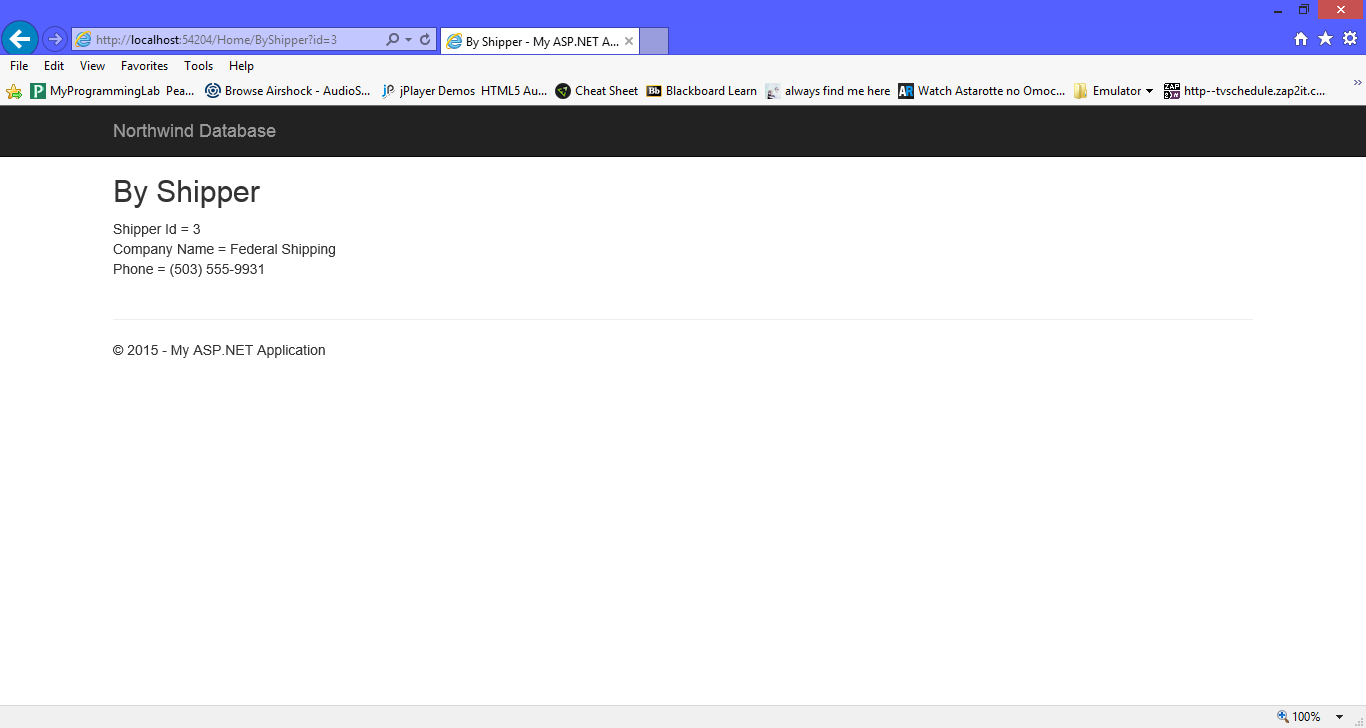
By Shipper Speedy Express Filter



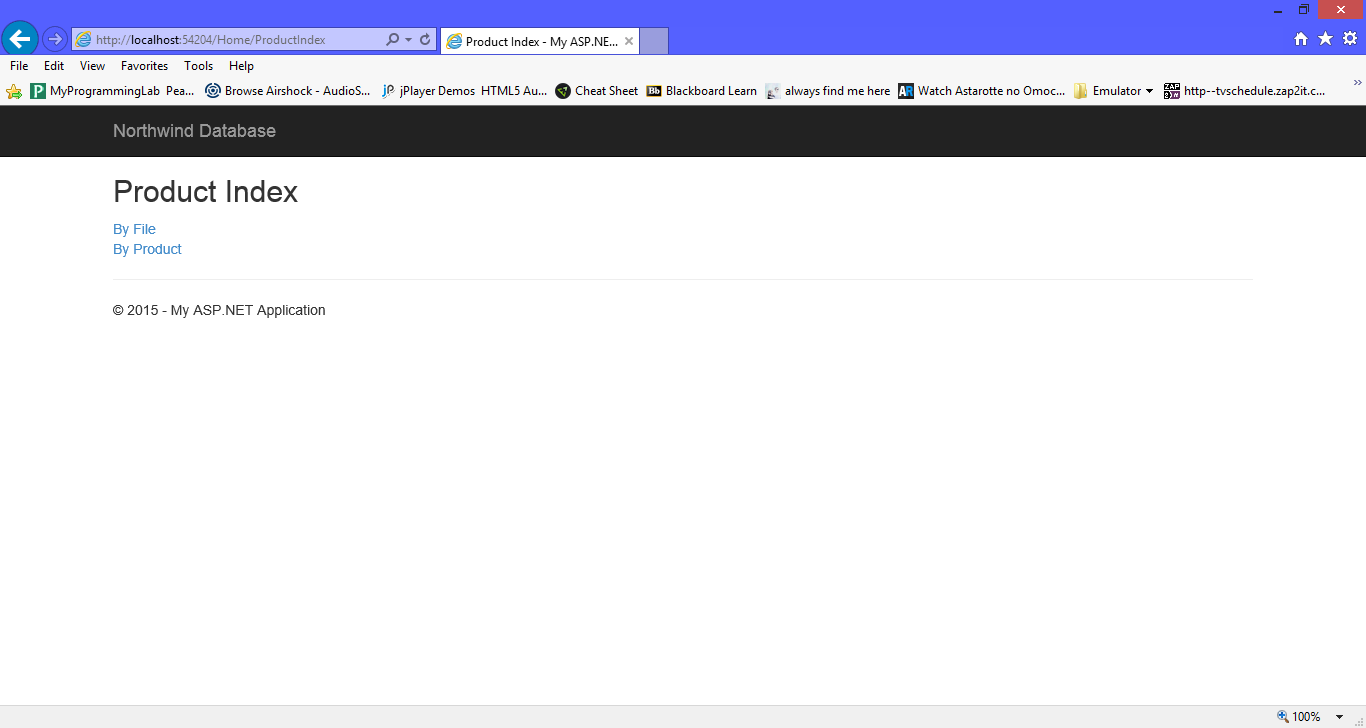
By Shipper United Package Filter



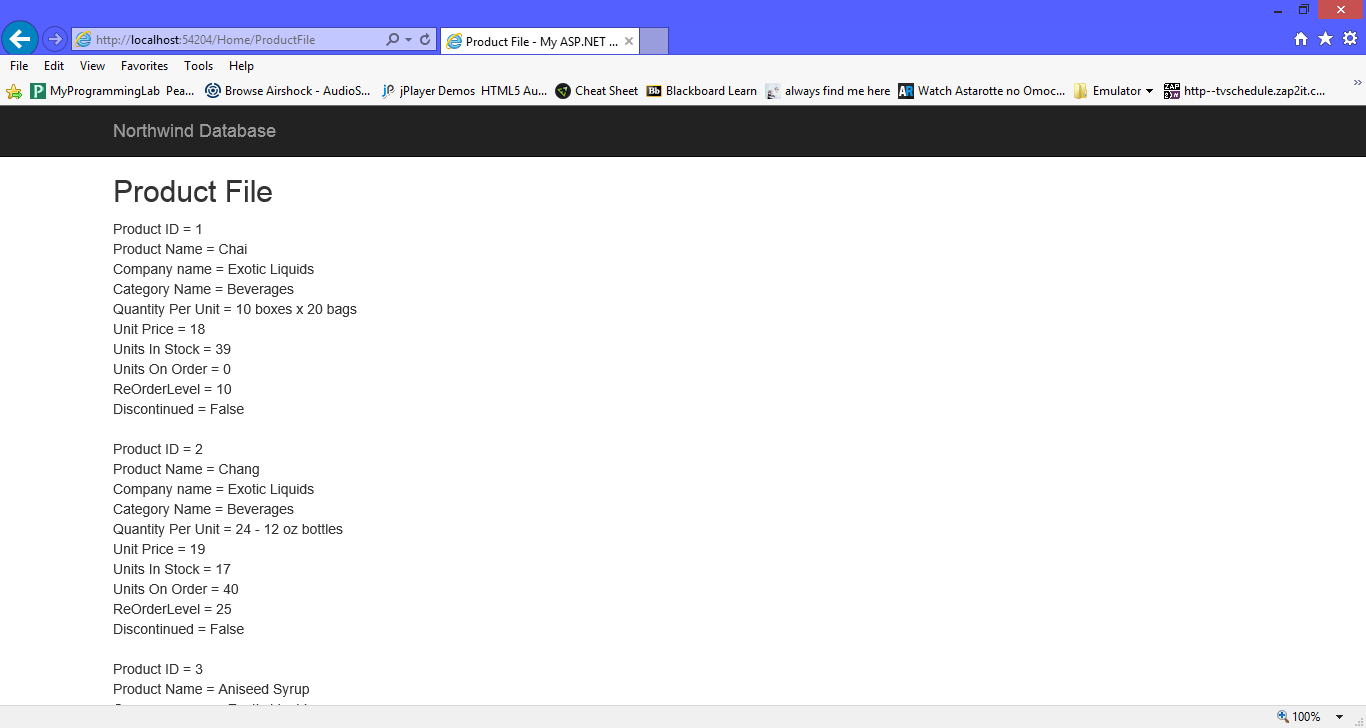
By Shipper Federal Shipping Filter



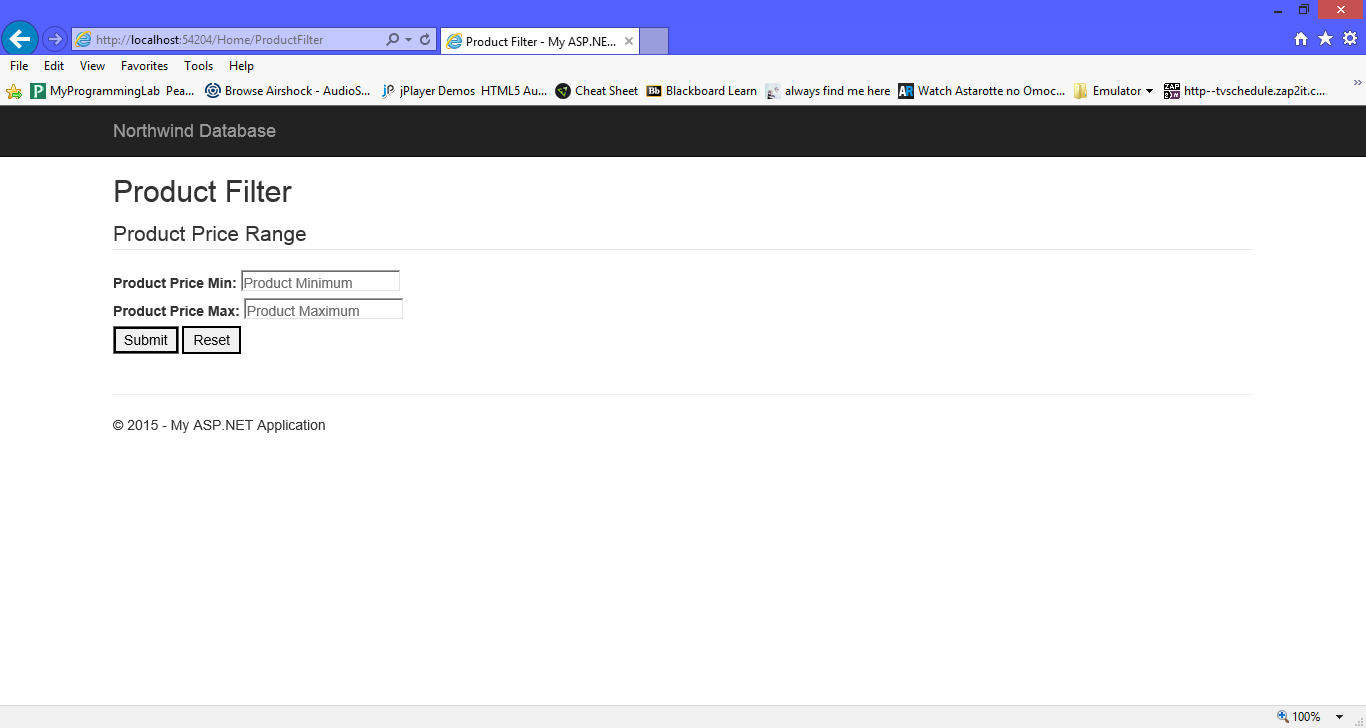
By Product Index



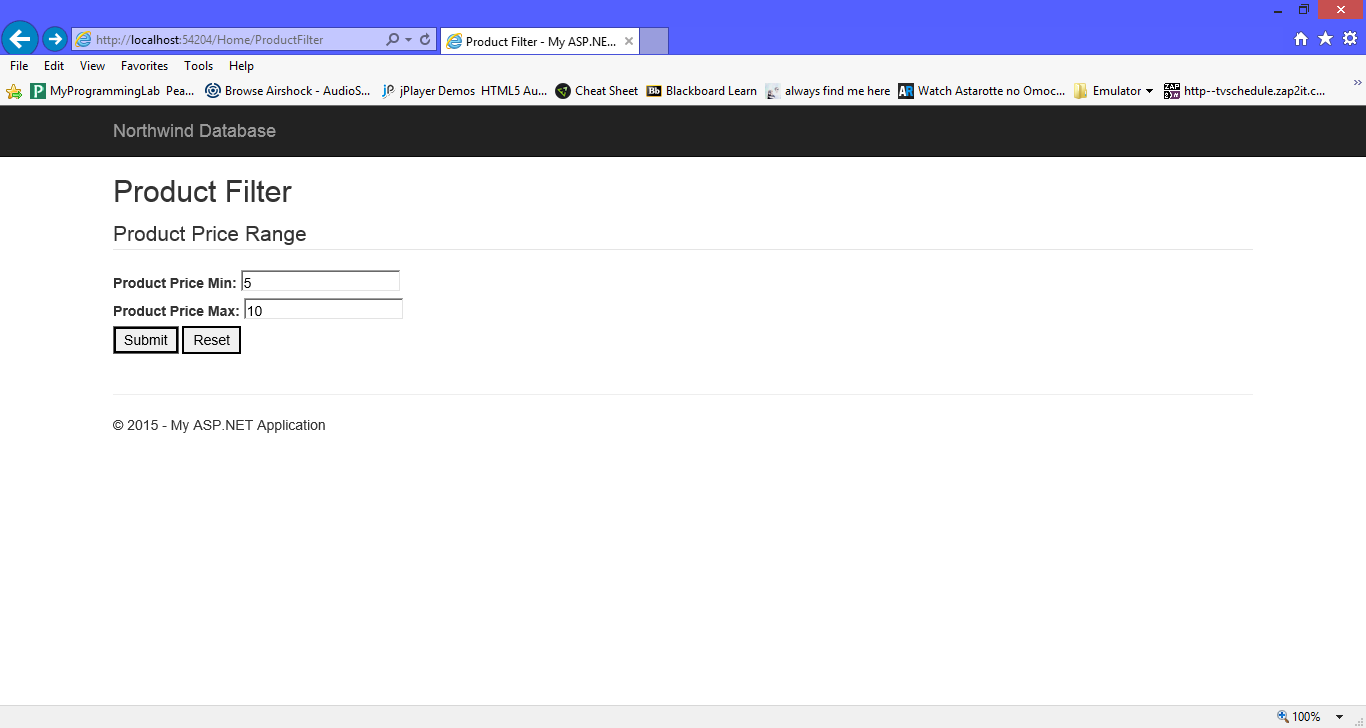
By Product File List



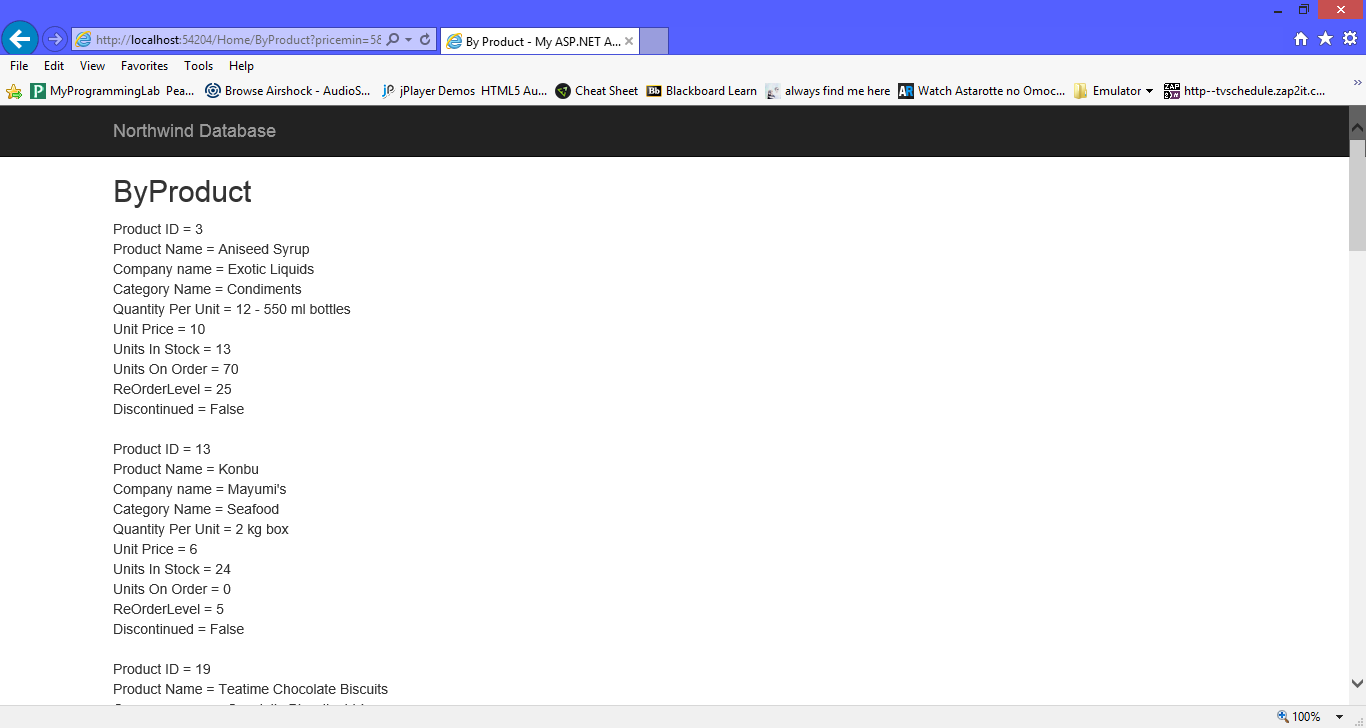
By Product Price Filter Unedited



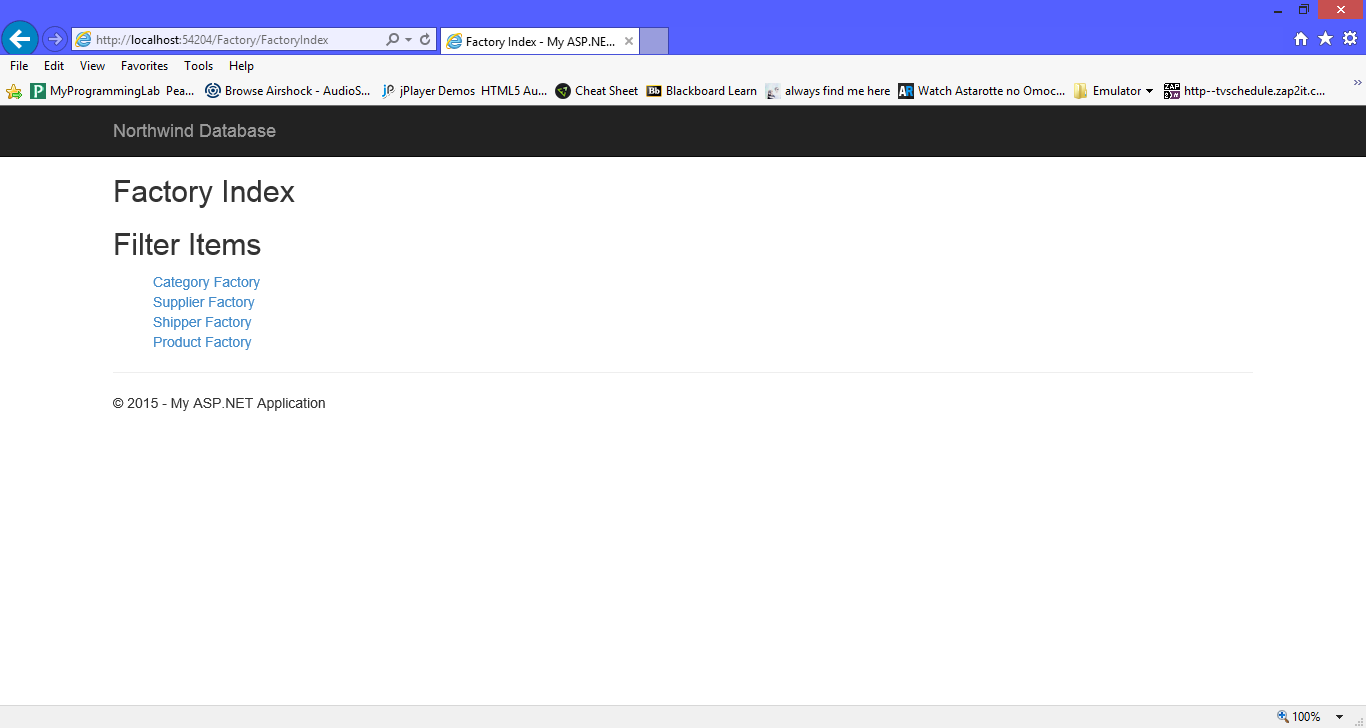
By Product Price Filter Edited



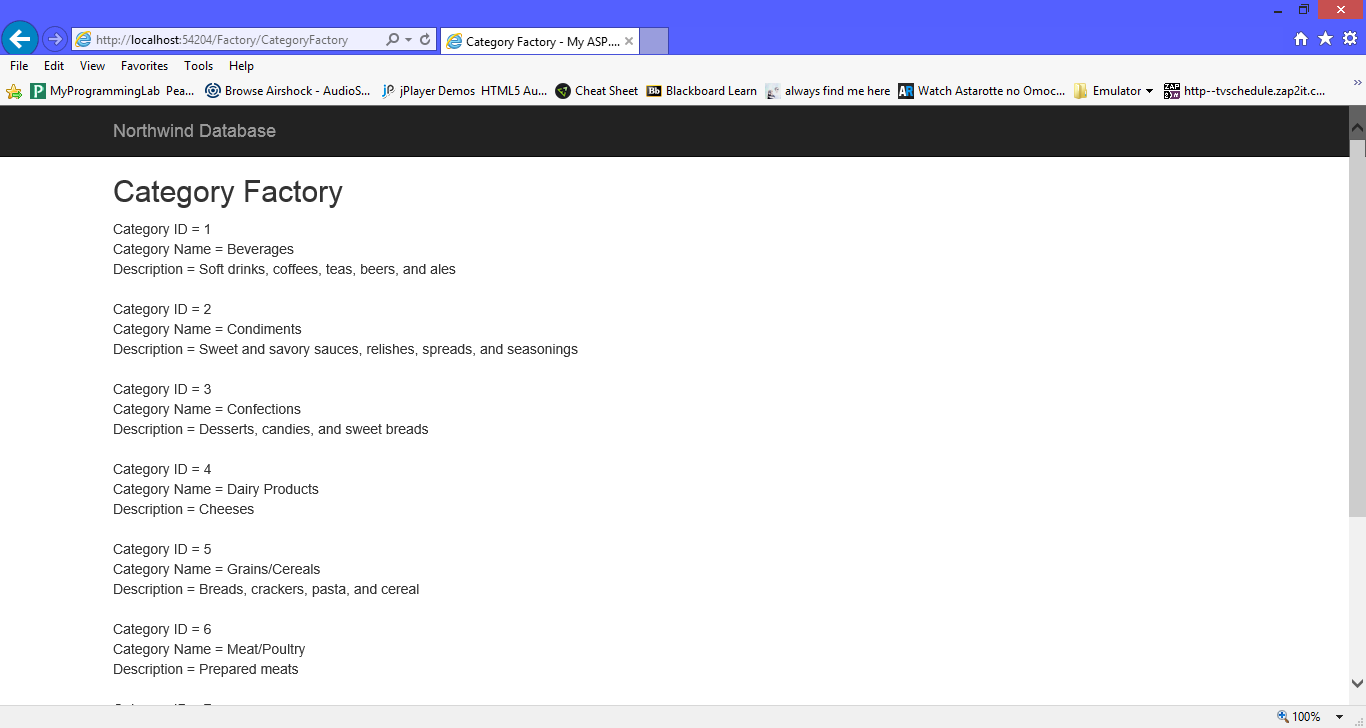
By Product Price Filter Results



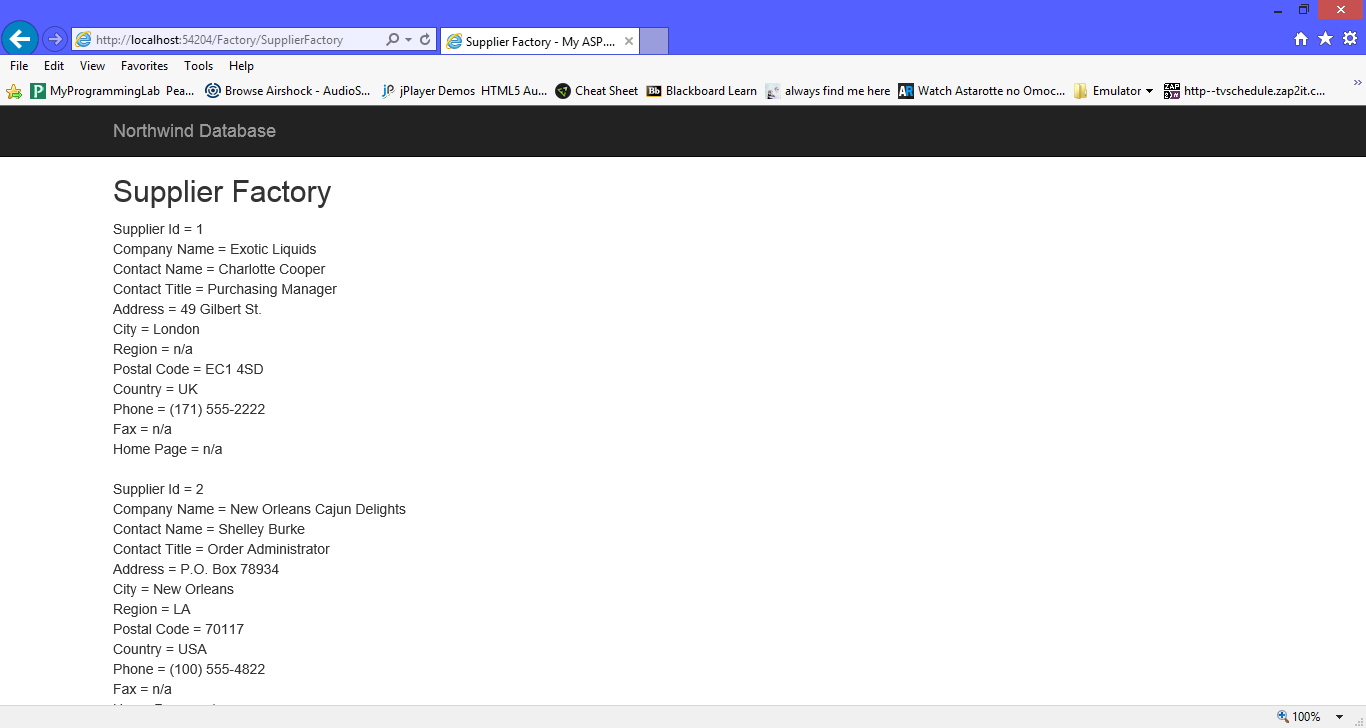
Factory Index Page



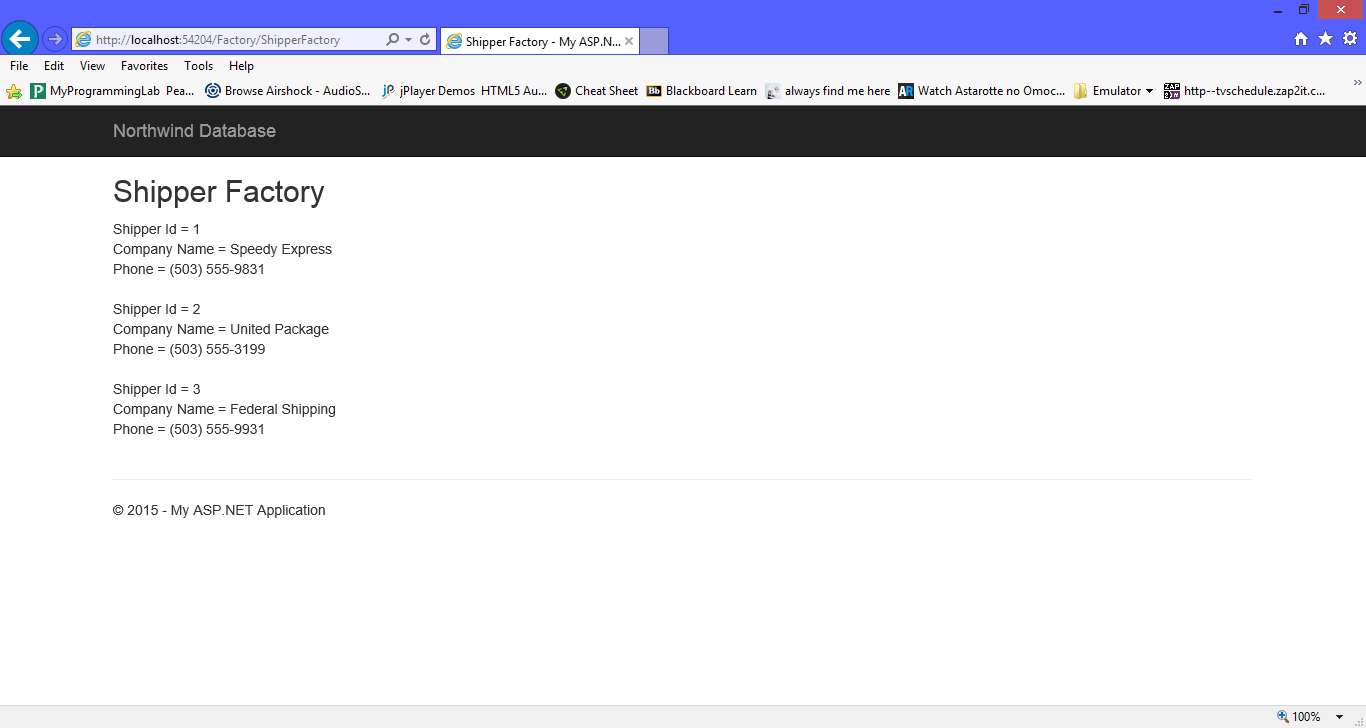
Category Factory List



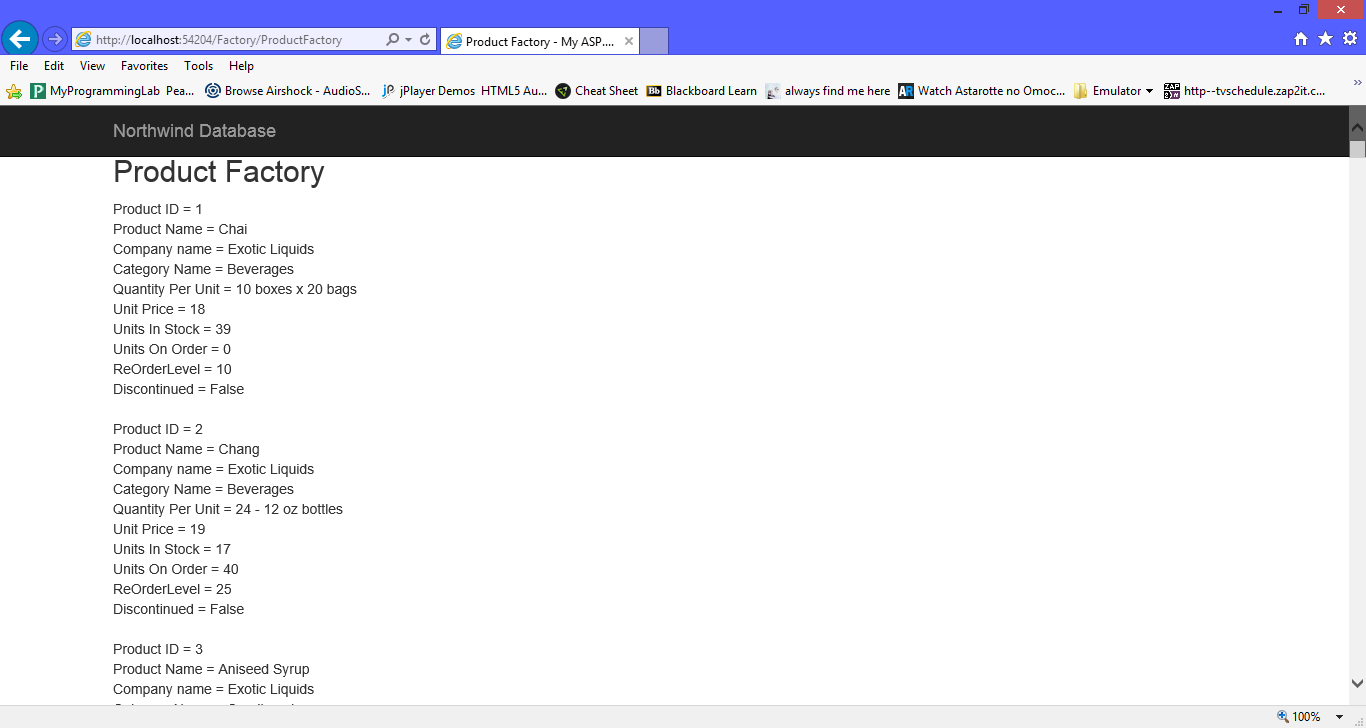
Supplier Factory List



Shipper Factory List



Product Factory List



This Program involves models, abstract, inherits, factory, views, controllers, gateway, broken rule, and identity map classes. . Northwind is a program that is based off of a database that keeps track of all its data through the company’s products, orders, employees, customer, shipper and suppliers. This program will involve working with C# and ASP.NET MVC5 framework. Also, this program will display design patterns containing the following patterns: inheritance of factory, abstract factory, table gateway, identity map and broken rules.

**Abstract Company Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class receives the Company data and then assigns the Company data to a the specific variable within

\* the class. Also, this class will test if the data is blank or not available by using constructors. It also sets up

\* the ToString() format for this file for when its called.

\*

\* \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public abstract class Company

{

// there is no primary key in Company()

private string companyName = "n/a";

private string contactName = "n/a";

private string contactTitle = "n/a";

private string address = "n/a";

private string city = "n/a";

private string region = "n/a";

private string postalCode = "n/a";

private string country = "n/a";

private string phone = "n/a";

private string fax = "n/a";

private string homePage = "n/a";

private bool isDirty = false;

private BrokenRules theBrokenRules = new BrokenRules();

public string CompanyName

{

get

{

return companyName;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.companyName = value;

}

else

{

this.companyName = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Company Name";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string ContactName

{

get

{

return contactName;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.contactName = value;

}

else

{

this.contactName = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Contact Name";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string ContactTitle

{

get

{

return contactTitle;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.contactTitle = value;

}

else

{

this.contactTitle = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Contact Title";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string Address

{

get

{

return address;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.address = value;

}

else

{

this.address = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Address";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string City

{

get

{

return city;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.city = value;

}

else

{

this.city = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry City";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string Region

{

get

{

return region;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.region = value;

}

else

{

this.region = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Region";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string PostalCode

{

get

{

return postalCode;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.postalCode = value;

}

else

{

this.postalCode = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Postal Code";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string Country

{

get

{

return country;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.country = value;

}

else

{

this.country = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Country";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string Phone

{

get

{

return phone;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.phone = value;

}

else

{

this.phone = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Phone";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string Fax

{

get

{

return fax;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.fax = value;

}

else

{

this.fax = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Fax";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string HomePage

{

get

{

return homePage;

}

set

{

isDirty = true;

if(value.Length > 0)

{

this.homePage = value;

}

else

{

this.homePage = "n/a";

}

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "NameRequired";

aRule.RuleDescription = "You must supply a non emptry Home Page";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public bool IsDirty

{

get

{

return isDirty;

}

set

{

isDirty = value;

}

}

public bool IsValid

{

get

{

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

{

return false;

}

else

{

return true;

}

}

}

public Company()

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

{

// We should be calling this from our children

// and I am not

}

public Company(string aCompanyName, string aContactName, string aContactTitle, string anAddress, string aCity, string aRegion,

string aPostalCode, string aCountry, string aPhone, string aFax, string aHomePage)

{

this.CompanyName = aCompanyName;

this.ContactName = aContactName;

this.ContactTitle = aContactTitle;

this.Address = anAddress;

this.City = aCity;

this.Region = aRegion;

this.PostalCode = aPostalCode;

this.Country = aCountry;

this.Phone = aPhone;

this.Fax = aFax;

this.HomePage = aHomePage;

}

public Company(string aCompanyName, string aContactName, string aContactTitle, string anAddress, string aCity, string aRegion,

string aPostalCode, string aCountry, string aPhone, string aFax)

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

{

}

public Company(string aCompanyName, string aContactName, string aContactTitle, string anAddress, string aCity, string aRegion,

string aPostalCode, string aCountry, string aPhone)

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

{

}

public Company(string aCompanyName, string aContactName, string aContactTitle, string anAddress, string aCity, string aRegion,

string aPostalCode, string aCountry)

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

{

}

public Company(string aCompanyName, string aContactName, string aContactTitle, string anAddress, string aCity, string aRegion,

string aPostalCode)

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

{

}

public Company(string aCompanyName, string aContactName, string aContactTitle, string anAddress, string aCity, string aRegion)

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

{

}

public Company(string aCompanyName, string aContactName, string aContactTitle, string anAddress, string aCity)

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

{

}

public Company(string aCompanyName, string aContactName, string aContactTitle, string anAddress)

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

{

}

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

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

{

}

public Company(string aCompanyName, string aContactName)

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

{

}

public Company(string aCompanyName)

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

{

}

// public string ToString() don't do this

public override string ToString()

{

string message = "";

message = message + "Company Name = " + this.CompanyName + "<br />";

message = message + "Contact Name = " + this.ContactName + "<br />";

message = message + "Contact Title = " + this.ContactTitle + "<br />";

message = message + "Address = " + this.Address + "<br />";

message = message + "City = " + this.City + "<br />";

message = message + "Region = " + this.Region + "<br />";

message = message + "Postal Code = " + this.PostalCode + "<br />";

message = message + "Country = " + this.Country + "<br />";

message = message + "Phone = " + this.Phone + "<br />";

message = message + "Fax = " + this.Fax + "<br />";

message = message + "Home Page = " + this.HomePage + "<br /><br />";

return message;

}

}

}

**BrokenRule Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class tests the ruleName and ruleDescription and then it decides whether the data from the classes

\* is usable or turn up blank / null.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class BrokenRule

{

private string ruleName = "n/a";

private string ruleDescription = "n/a";

public string RuleName

{

get

{

return ruleName;

}

set

{

ruleName = value;

}

}

public string RuleDescription

{

get

{

return ruleDescription;

}

set

{

ruleDescription = value;

}

}

public override string ToString()

{

string aMessage = this.RuleName;

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

return aMessage;

}

}

}

**BrokenRules Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to use the BrokenRules for the BrokenRule class. Which revolves around putting

\* categories into dictionaries where it will see if any of the data saved has been altered at any point in time.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class BrokenRules

{

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

public Dictionary<string, BrokenRule> GetDictionary()

{

return aDictionary;

}

public Boolean isInDictionary(string aKey)

{

bool answer = false;

answer = aDictionary.ContainsKey(aKey);

return answer;

}

public void AddRule(string aKey, BrokenRule aBrokenRule)

{

aDictionary.Add(aKey, aBrokenRule);

//alternatively

//aDictionary.Add(aCategory.CategoryId, aCategory);

// This is slightly less good because the key may not

// be the primary key

}

// use this method to retreive an item

// in the map

public BrokenRule GetRule(string aKey)

{

return aDictionary[aKey];

}

// you may at some point need to

// remove an item from the map

// this does not delete from

// the database

// all database functions are

// on the table gateway object

public void RemoveRule(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.AddRule(aRuleName, aRule);

}

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

{

this.RemoveRule(aRuleName);

}

}

public override string ToString()

{

string aMessage = "";

foreach(var b in aDictionary)

{

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

}

return aMessage;

}

}

}

**Category Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class receives the Category data and then assigns the Category data to a the specific variable within

\* the class. Also, this class will test if the data is blank or not available by using constructors. It also sets up

\* the ToString() format for this file for when its called.

\*

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class Category

{

private int categoryId = 0;

private string categoryName = "n/a";

private string description = "n/a";

private BrokenRules theBrokenRules = new BrokenRules();

private bool isDirty = false;

public int CategoryId

{

get

{

return categoryId;

}

}

public string CategoryName

{

get

{

return categoryName;

}

set

{

categoryName = value;

isDirty = true;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "Name Required";

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

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public string Description

{

get

{

return description;

}

set

{

description = value;

isDirty = true;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "DescriptionRequired";

aRule.RuleDescription = "You must supply a non emptry Category Description";

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

}

}

public bool IsDirty

{

get

{

return isDirty;

}

set

{

isDirty = value;

}

}

public bool IsValid

{

get

{

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

{

return false;

}

else

{

return true;

}

}

}

// Constructors

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, "n/a")

{

}

public Category(int aCategoryId)

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

{

}

public override string ToString()

{

string aString = "";

aString = aString + "Category ID = " + this.CategoryId + "<br />";

aString = aString + "Category Name = " + this.CategoryName + "<br />";

aString = aString + "Description = " + this.Description + "<br /><br />";

return aString;

}

}

}

**Product Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information about Product Class

This class receives the Product's data and then assigns the Product's data to a specific variable within the class. Also, this class will test if the data is blank or not available. It also sets up the .ToString() format for this file for when it’s called.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Northwind.Models

{

public class Product

{

// Declaration

private int productId = -1; // only one with a get

private string productName = "n/a";

private string supplierId = "n/a";

private string categoryId = "n/a";

private string quantityPerUnit = "n/a";

private double unitPrice = 1000000000;

private int unitsInStock = -1;

private int unitsOnOrder = -1;

private int reorderLevel = -1;

private bool discontinued = false;

private bool isDirty = false;

private BrokenRules theBrokenRules = new BrokenRules();

// Get And Sets

public int ProductId

{

get

{

return productId;

}

set

{

// ideally this is not needed because

// this is the primary key

// and there should not even be a set

// for this property

isDirty = true;

productId = value;

}

}

public string ProductName

{

get

{

return productName;

}

set

{

isDirty = true;

productName = value;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "ProductNameRequired";

aRule.RuleDescription = "You must supply a non emptry Product Name";

if(value.Length > 0)

{

this.productName = value;

}

else

{

this.productName = "n/a";

}

}

}

public string SupplierId

{

get

{

return supplierId;

}

set

{

isDirty = true;

supplierId = value;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "SupplierIDRequired";

aRule.RuleDescription = "You must supply a non emptry Supplier ID";

if(value.Length > 0)

{

this.supplierId = value;

}

else

{

this.supplierId = "n/a";

}

}

}

public string CategoryId

{

get

{

return categoryId;

}

set

{

isDirty = true;

categoryId = value;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "CategoryIDRequired";

aRule.RuleDescription = "You must supply a non emptry CategoryID";

if(value.Length > 0)

{

this.categoryId = value;

}

else

{

this.categoryId = "n/a";

}

}

}//\*/

public string QuantityPerUnit

{

get

{

return quantityPerUnit;

}

set

{

isDirty = true;

quantityPerUnit = value;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "QuantityPerUnitRequired";

aRule.RuleDescription = "You must supply a non emptry Quantity Per Unit";

if(value.Length > 0)

{

this.quantityPerUnit = value;

}

else

{

this.quantityPerUnit = "n/a";

}

}

}

public double UnitPrice

{

get

{

return unitPrice;

}

set

{

isDirty = true;

unitPrice = value;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "UnitPriceRequired";

aRule.RuleDescription = "You must supply a non emptry Unit Price";

if(value >= 0)

{

this.unitPrice = value;

}

else

{

this.unitPrice = -1;

}

}

}

public int UnitsInStock

{

get

{

return unitsInStock;

}

set

{

isDirty = true;

unitsInStock = value;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "UnitsInStockRequired";

aRule.RuleDescription = "You must supply a non emptry Units In Stock";

if(value >= 0)

{

this.unitsInStock = value;

}

else

{

this.unitsInStock = -1;

}

}

}

public int UnitsOnOrder

{

get

{

return unitsOnOrder;

}

set

{

isDirty = true;

unitsOnOrder = value;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "UnitsOnOrderRequired";

aRule.RuleDescription = "You must supply a non emptry Units On Order";

if(value >= 0)

{

this.unitsOnOrder = value;

}

else

{

this.unitsOnOrder = -1;

}

}

}

public int ReorderLevel

{

get

{

return reorderLevel;

}

set

{

isDirty = true;

reorderLevel = value;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "ReorderLevelRequired";

aRule.RuleDescription = "You must supply a non emptry Reorder Level";

if(value >= 0)

{

this.reorderLevel = value;

}

else

{

this.reorderLevel = -1;

}

}

}

public bool Discontinued

{

get

{

return discontinued;

}

set

{

isDirty = true;

discontinued = value;

// Somewhere rules may already exist

// that is, your organization may have standard rules

// we don't have that, thus we have to make our own broken rule

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BrokenRule aRule = new BrokenRule();

aRule.RuleName = "discontinuedRequired";

aRule.RuleDescription = "You must supply a non emptry Discontinued";

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

public Product()

{

}

public Product(int aProductId, string aproductName, string asupplierId, string acategoryId, string aquantityPerUnit, double aunitPrice,

int aunitsInStock, int aunitsOnOrder, int areorderLevel, bool adiscontinued)

: this()

{

this.ProductId = aProductId; // Do Not Capatilize the variable for anID

this.ProductName = aproductName;

this.SupplierId = asupplierId; // Capatilize the rest of the this.variable's

this.CategoryId = acategoryId;

this.QuantityPerUnit = aquantityPerUnit;

this.UnitPrice = aunitPrice;

this.UnitsInStock = aunitsInStock;

this.UnitsOnOrder = aunitsOnOrder;

this.ReorderLevel = areorderLevel;

this.Discontinued = adiscontinued;

}

public Product(int anId, string aproductName, string asupplierId, string acategoryId, string aquantityPerUnit, double aunitPrice,

int aunitsInStock, int aunitsOnOrder, int areorderLevel)

: this(anId, aproductName, asupplierId, acategoryId, aquantityPerUnit, aunitPrice, aunitsInStock,

aunitsOnOrder, areorderLevel, false)

{

}

public Product(int anId, string aproductName, string asupplierId, string acategoryId, string aquantityPerUnit, double aunitPrice,

int aunitsInStock, int aunitsOnOrder)

: this(anId, aproductName, asupplierId, acategoryId, aquantityPerUnit, aunitPrice, aunitsInStock, aunitsOnOrder, -1, false)

{

}

public Product(int anId, string aproductName, string asupplierId, string acategoryId, string aquantityPerUnit, double aunitPrice,

int aunitsInStock)

: this(anId, aproductName, asupplierId, acategoryId, aquantityPerUnit, aunitPrice, aunitsInStock,

-1, -1, false)

{

}

public Product(int anId, string aproductName, string asupplierId, string acategoryId, string aquantityPerUnit, double aunitPrice)

: this(anId, aproductName, asupplierId, acategoryId, aquantityPerUnit, aunitPrice, -1, -1, -1, false)

{

}

public Product(int anId, string aproductName, string asupplierId, string acategoryId, string aquantityPerUnit)

: this(anId, aproductName, asupplierId, acategoryId, aquantityPerUnit, 0, -1, -1, -1, false)

{

}

public Product(int anId, string aproductName, string asupplierId, string acategoryId)

: this(anId, aproductName, asupplierId, acategoryId, "n/a", 0, -1, -1, -1, false)

{

}

public Product(int anId, string aproductName, string asupplierId)

: this(anId, aproductName, asupplierId, "n/a", "n/a", 0, -1, -1, -1, false)

{

}

public Product(int anId, string aproductName)

: this(anId, aproductName, "n/a", "n/a", "n/a", 0, -1, -1, -1, false)

{

}

public Product(int anId)

: this(anId, "n/a", "n/a", "n/a", "n/a", -1, -1, -1, -1, false)

{

}

// Output

public override string ToString()

{

string aString = "";

aString = aString + "Product ID = " + ProductId + "<br />";

aString = aString + "Product Name = " + ProductName + "<br />";

aString = aString + "Company name = " + SupplierId + "<br />";

aString = aString + "Category Name = " + CategoryId + "<br />";

aString = aString + "Quantity Per Unit = " + QuantityPerUnit + "<br />";

aString = aString + "Unit Price = " + UnitPrice + "<br />";

aString = aString + "Units In Stock = " + UnitsInStock + "<br />";

aString = aString + "Units On Order = " + UnitsOnOrder + "<br />";

aString = aString + "ReOrderLevel = " + ReorderLevel + "<br />";

aString = aString + "Discontinued = " + Discontinued + "<br />";

return aString;

}

}

}

**Shipper Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class receives the Shipper data and then assigns the Shipper data to a the specific variable within

\* the class. It also inherits data from the Company class. Also, this class will test if the data is blank or not available

\* by using constructors. It also sets up the ToString() format for this file for when its called.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class Shipper : Company

{

private int shipperId = 0;

public string GetName()

{

return "";

}

public string GetPhone()

{

return "";

}

public int ShipperId

{

get

{

return shipperId;

}

// should be read only

set

{

shipperId = value;

}

}

public Shipper()

: this(0, "n/a", "n/a")

{

}

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

: base(aCompanyName, "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", aPhone, "n/a", "n/a")

{

ShipperId = aShipperId;

}

public override string ToString()

{

string message = "Shipper Id = " + this.ShipperId + "<br />";

message = message + "Company Name = " + base.CompanyName + " <br />";

message = message + "Phone = " + base.Phone + " <br /><br />";

return message;

}

}

}

**Customer Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class receives the Customer data and then assigns the Customer data to a the specific variable within

\* the class. This class also inherits data from the Company class. Also, this class will test if the data is blank or not

\* available by using constructors. It also sets up the ToString() format for this file for when its called.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class Customer : Company

{

// Incorectly inherits a Homepage property

private string customerId = "n/a";

public string CustomerId

{

get

{

return customerId;

}

// Should we be doing this????

// Should it be read only

set

{

customerId = value;

}

}

// By default, Constructors don't inherit

public Customer()

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

{

// \*\* ALWAYS ALWAYS ALWAYS Call the

// BASE() Constructor \*\*

}

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

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

{

CustomerId = aCustomerId;

}

public override string ToString()

{

string message = "Customer Id = " + this.CustomerId + "<br />";

message = message + base.ToString();

return message;

}

}

}

**Supplier Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class receives the Supplier data and then assigns the Supplier data to a the specific variable within

\* the class. It also inherits data from the Company class. Also, this class will test if the data is blank or not available

\* by using constructors. It also sets up the ToString() format for this file for when its called.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class Supplier: Company

{

private int supplierId = 0;

//private bool isDirty = false;

private BrokenRules theBrokenRules = new BrokenRules();

public int SupplierId

{

get

{

return supplierId;

}

}

/\*public bool isDirty

{

get

{

return IsDirty;

}

set

{

IsDirty = value;

}

}\*/

public Supplier()

:this(0, "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a")

{

}

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

string aCity, string aRegion, string aPostalCode, string aCountry, string aPhone, string aFax, string aHomePage)

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

{

this.supplierId = aSupplierId;

}

public override string ToString() {

string message = "Supplier Id = " + this.SupplierId + "<br />";

message = message + base.ToString();

return message;

}

}

}

**CategoryIdentityMap Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to use the CategoryIdentityMap for the Category class. Which revolves around putting categories into dictionaries where it will see if any of the data saved has been altered at any point in time

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class CategoryIdentityMap

{

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

public Dictionary<int, Category> GetDictionary()

{

return aDictionary;

}

public Boolean isInDictionary(int aKey)

{

bool answer = false;

answer = aDictionary.ContainsKey(aKey);

return answer;

}

public void AddCategory(int aKey, Category aCategory)

{

aDictionary.Add(aKey, aCategory);

//alternatively

//aDictionary.Add(aCategory.CategoryId, aCategory);

// This is slightly less good because the key may not

// be the primary key

}

// use this method to retreive an item

// in the map

public Category GetCategory(int aKey)

{

return aDictionary[aKey];

}

// you may at some point need to

// remove an item from the map

// this does not delete from

// the database

// all database functions are

// on the table gateway object

public void RemoveCategory(int aKey)

{

aDictionary.Remove(aKey);

}

}

}

**ProductIdentityMap Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to use the ProductIdentityMap for the Product class. Which revolves around putting

\* categories into dictionaries where it will see if any of the data saved has been altered at any point in time.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class ProductIdentityMap

{

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

public Dictionary<int, Product> GetDictionary()

{

return aDictionary;

}

public Boolean isInDictionary(int aKey)

{

bool answer = false;

answer = aDictionary.ContainsKey(aKey);

return answer;

}

public void AddProduct(int aKey, Product aProduct)

{

aDictionary.Add(aKey, aProduct);

//alternatively

//aDictionary.Add(aCategory.CategoryId, aCategory);

// This is slightly less good because the key may not

// be the primary key

}

// use this method to retreive an item

// in the map

public Product GetProduct(int aKey)

{

return aDictionary[aKey];

}

// you may at some point need to

// remove an item from the map

// this does not delete from

// the database

// all database functions are

// on the table gateway object

public void RemoveProduct(int aKey)

{

aDictionary.Remove(aKey);

}

}

}

**ShipperIdentityMap Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to use the ShipperIdentityMap for the Shipper class. Which revolves around putting

\* categories into dictionaries where it will see if any of the data saved has been altered at any point in time.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class ShipperIdentityMap

{

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

public Dictionary<int, Shipper> GetDictionary()

{

return aDictionary;

}

public Boolean isInDictionary(int aKey)

{

bool answer = false;

answer = aDictionary.ContainsKey(aKey);

return answer;

}

public void AddShipper(int aKey, Shipper aShipper)

{

aDictionary.Add(aKey, aShipper);

//alternatively

//aDictionary.Add(aCategory.CategoryId, aCategory);

// This is slightly less good because the key may not

// be the primary key

}

// use this method to retreive an item

// in the map

public Shipper GetShipper(int aKey)

{

return aDictionary[aKey];

}

// you may at some point need to

// remove an item from the map

// this does not delete from

// the database

// all database functions are

// on the table gateway object

public void RemoveShipper(int aKey)

{

aDictionary.Remove(aKey);

}

}

}

**SupplierIdentityMap**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to use the SupplierIdentityMap for the Suppler class. Which revolves around putting

\* categories into dictionaries where it will see if any of the data saved has been altered at any point in time.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public class SupplierIdentityMap

{

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

public Dictionary<int, Supplier> GetDictionary()

{

return aDictionary;

}

public Boolean isInDictionary(int aKey)

{

bool answer = false;

answer = aDictionary.ContainsKey(aKey);

return answer;

}

public void AddSupplier(int aKey, Supplier aSupplier)

{

aDictionary.Add(aKey, aSupplier);

//alternatively

//aDictionary.Add(aCategory.CategoryId, aCategory);

// This is slightly less good because the key may not

// be the primary key

}

// use this method to retreive an item

// in the map

public Supplier GetSupplier(int aKey)

{

return aDictionary[aKey];

}

// you may at some point need to

// remove an item from the map

// this does not delete from

// the database

// all database functions are

// on the table gateway object

public void RemoveSupplier(int aKey)

{

aDictionary.Remove(aKey);

}

}

}

**CategoryConnection Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to establish a connection to the Northwind database and to be used for the Category Table, as

\* well as the Category portion of this database program. It uses the OleDbConnection setup to establish the connection.

\*/

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 Northwind.Models

{

public class CategoryConnection

{

// Call Lists

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

// By Selection

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

//Call Variables

private Category aCategory = null;

// Call Identity Map

private CategoryIdentityMap aCategoryMap = new CategoryIdentityMap();

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static CategoryConnection anInstance = null;

public static CategoryConnection GetanInstance

{

get

{

if(CategoryConnection.anInstance == null)

{

CategoryConnection.anInstance = new CategoryConnection();

}

return CategoryConnection.anInstance;

}

}

public CategoryConnection()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

public List<Category> GetCategories()

{

int aCategoryId = -1;

string aCategoryName = "n/a";

string aDescription = "n/a";

// 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())

{

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

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

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

// If the item is not in the dictionary

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

if(aCategoryMap.isInDictionary(aCategoryId) == false)

{

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

aCategoryMap.AddCategory(aCategory.CategoryId, aCategory);

}

else

{

// the item already exists so get it

aCategoryMap.GetCategory(aCategoryId);

}

// this adds the category to this list

// either way

aListOfCategories.Add(aCategory);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aListOfCategories;

}

public void SaveAllCategories()

{

int aCategoryId = -1;

string aCategoryName = "n/a";

string aDescription = "n/a";

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

// 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)

{

aCategoryId = dictionary[c].CategoryId;

aCategoryName = dictionary[c].CategoryName;

aDescription = dictionary[c].Description;

// If the item is not in the dictionary

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

if(dictionary[c].IsDirty == true && dictionary[c].IsValid == 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();

}

public List<Category> GetByCategories(int Id)

{

int aCategoryId = -1;

string aCategoryName = "n/a";

string aDescription = "n/a";

// 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 = " + Id + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

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

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

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

// If the item is not in the dictionary

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

if(aCategoryMap.isInDictionary(aCategoryId) == false)

{

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

aCategoryMap.AddCategory(aCategory.CategoryId, aCategory);

}

else

{

// the item already exists so get it

aCategoryMap.GetCategory(aCategoryId);

}

// this adds the category to this list

// either way

aByCategories.Add(aCategory);

}

// close the connection

aConnection.Close();

// Return aByCategories

return aByCategories;

}

}

}

**ProductConnection Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to establish a connection to the Northwind database. Also, to be used for the Product Table, as

\* well as the Product portion of this database program. It uses the OleDbConnection setup to establish the connection.

\*/

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 Northwind.Models

{

public class ProductConnection

{

// Call Lists

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

// By Selection

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

// Call Variables

private Product aProduct = null;

// Call Identity Map

private ProductIdentityMap aProductMap = new ProductIdentityMap();

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static ProductConnection anInstance = null;

public static ProductConnection GetanInstance

{

get

{

if(ProductConnection.anInstance == null)

{

ProductConnection.anInstance = new ProductConnection();

}

return ProductConnection.anInstance;

}

}

public ProductConnection()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

// Get Price

public List<Product> GetProduct()

{

int aProductId = -1; // only one with a get

string aProductName = "n/a";

string aSupplierId = "n/a";

string aCategoryId = "n/a";

string aQuantityPerUnit = "n/a";

double aUnitPrice = 1000000000;

int aUnitsInStock = -1;

int aUnitsOnOrder = -1;

int aReorderLevel = -1;

bool aDiscontinued = false;

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT Products.ProductID, Products.ProductName, Suppliers.CompanyName, Categories.CategoryName, " +

"Products.QuantityPerUnit, Products.UnitPrice, Products.UnitsInStock, Products.UnitsOnOrder, Products.ReorderLevel, " +

"Products.Discontinued " +

"FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID) INNER JOIN Suppliers ON " +

"Products.SupplierID = Suppliers.SupplierID) " +

"ORDER BY Products.ProductID ASC;";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aProductId = Convert.ToInt32(aReader["ProductID"]);

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

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

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

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

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

aUnitsInStock = Convert.ToInt32(aReader["UnitsInStock"]);

aUnitsOnOrder = Convert.ToInt32(aReader["UnitsOnOrder"]);

aReorderLevel = Convert.ToInt32(aReader["ReorderLevel"]);

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

// If the item is not in the dictionary

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

if(aProductMap.isInDictionary(aProductId) == false)

{

aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice,

aUnitsInStock, aUnitsOnOrder, aReorderLevel, aDiscontinued);

aProductMap.AddProduct(aProduct.ProductId, aProduct);

}

else

{

// the item already exists so get it

aProductMap.GetProduct(aProductId);

}

// this adds the category to this list

// either way

aListOfProduct.Add(aProduct);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aListOfProduct;

}

public void SaveAllProducts()

{

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

// 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; // Product ID

string aProductName = dictionary[p].ProductName;

string aSupplierId = dictionary[p].SupplierId;

string 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 aDiscontinued = 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 && dictionary[p].IsValid == true)

{

aSQL = "Update Products";

aSQL = aSQL + "SET Product Name = " + aProductName + ", ";

aSQL = aSQL + "Company Name = " + aSupplierId + ", ";

aSQL = aSQL + "Category Name = " + aCategoryId + ", ";

aSQL = aSQL + "Quantity Per Unit = " + aQuantityPerUnit + ", ";

aSQL = aSQL + "WHERE UnitPrice = " + aUnitPrice + " ";

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

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

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

aSQL = aSQL + "Discontinued = " + aDiscontinued + " ";

aCommand.CommandText = aSQL;

aCommand.ExecuteNonQuery();

}

}

// close the connection

aConnection.Close();

}

public List<Product> GetByProducts(double pricemin, double pricemax)

{

int aProductId = -1; // only one with a get

string aProductName = "n/a";

string aSupplierId = "n/a";

string aCategoryId = "n/a";

string aQuantityPerUnit = "n/a";

double aUnitPrice = 1000000000;

int aUnitsInStock = -1;

int aUnitsOnOrder = -1;

int aReorderLevel = -1;

bool aDiscontinued = false;

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT Products.ProductID, Products.ProductName, Suppliers.CompanyName, Categories.CategoryName, " +

"Products.QuantityPerUnit, Products.UnitPrice, Products.UnitsInStock, Products.UnitsOnOrder, Products.ReorderLevel, " +

"Products.Discontinued " +

"FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID) INNER JOIN Suppliers ON " +

"Products.SupplierID = Suppliers.SupplierID) " +

"WHERE Products.UnitPrice >= " + pricemin + " AND Products.UnitPrice <= " + pricemax + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aProductId = Convert.ToInt32(aReader["ProductID"]);

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

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

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

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

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

aUnitsInStock = Convert.ToInt32(aReader["UnitsInStock"]);

aUnitsOnOrder = Convert.ToInt32(aReader["UnitsOnOrder"]);

aReorderLevel = Convert.ToInt32(aReader["ReorderLevel"]);

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

// If the item is not in the dictionary

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

if(aProductMap.isInDictionary(aProductId) == false)

{

aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice,

aUnitsInStock, aUnitsOnOrder, aReorderLevel, aDiscontinued);

aProductMap.AddProduct(aProduct.ProductId, aProduct);

}

else

{

// the item already exists so get it

aProductMap.GetProduct(aProductId);

}

// this adds the category to this list

// either way

aByProduct.Add(aProduct);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aByProduct;

}

}

}

**ShipperConnection Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to establish a connection to the Northwind database and to be used for the Shipper Table, as

\* well as the Shipper portion of this database program. It uses the OleDbConnection setup to establish the connection.

\*/

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 Northwind.Models

{

public class ShipperConnection

{

// Call List

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

// By Selection

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

// Call Variables

private Shipper aShipper = null;

// Call Identity Map

private ShipperIdentityMap aShipperMap = new ShipperIdentityMap();

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static ShipperConnection anInstance = null;

public static ShipperConnection GetanInstance

{

get

{

if(ShipperConnection.anInstance == null)

{

ShipperConnection.anInstance = new ShipperConnection();

}

return ShipperConnection.anInstance;

}

}

public ShipperConnection()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

// Get Shippers

public List<Shipper> GetShippers()

{

int aShipperId = -1; // Shipper Number

string aCompanyName = "n/a";

string aPhone = "n/a";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT ShipperID, CompanyName, Phone FROM Shippers" + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aShipperId = Convert.ToInt32(aReader["ShipperID"]);

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

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

// If the item is not in the dictionary

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

if(aShipperMap.isInDictionary(aShipperId) == false)

{

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

aShipperMap.AddShipper(aShipper.ShipperId, aShipper);

}

else

{

// the item already exists so get it

aShipperMap.GetShipper(aShipperId);

}

// this adds the category to this list

// either way

aListOfShipper.Add(aShipper);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aListOfShipper;

}

public void SaveAllShippers()

{

Dictionary<int, Shipper> dictionary = aShipperMap.GetDictionary();

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "";

foreach(var s in dictionary.Keys)

{

int aShipperId = dictionary[s].ShipperId; // Supplier ID

string aCompanyName = dictionary[s].CompanyName;

string aPhone = dictionary[s].Phone;

// If the item is not in the dictionary

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

if(dictionary[s].IsDirty == true && dictionary[s].IsValid == true)

{

aSQL = "Update Shippers";

aSQL = aSQL + "SET Company Name = " + aCompanyName + "', ";

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

aSQL = aSQL + "WHERE ShipperId = " + aShipperId;

aCommand.CommandText = aSQL;

aCommand.ExecuteNonQuery();

}

}

// close the connection

aConnection.Close();

}

public List<Shipper> GetByShippers(int id)

{

int aShipperId = -1; // Shipper Number

string aCompanyName = "n/a";

string aPhone = "n/a";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT ShipperID, CompanyName, Phone FROM Shippers WHERE ShipperID = " + id + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aShipperId = Convert.ToInt32(aReader["ShipperID"]);

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

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

// If the item is not in the dictionary

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

if(aShipperMap.isInDictionary(aShipperId) == false)

{

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

aShipperMap.AddShipper(aShipper.ShipperId, aShipper);

}

else

{

// the item already exists so get it

aShipperMap.GetShipper(aShipperId);

}

// this adds the category to this list

// either way

aByShipper.Add(aShipper);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aByShipper;

}

}

}

**SupplierConnection Class**

/\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to establish a connection to the Northwind database and to be used for the Supplier Table, as

\* well as the Supplier portion of this database program. It uses the OleDbConnection setup to establish the connection.

\*/

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 Northwind.Models

{

public class SupplierConnection

{

// Call Lists

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

// By Selection

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

// Call Variables

private Supplier aSupplier = null;

// Call Identity Map

private SupplierIdentityMap aSupplierMap = new SupplierIdentityMap();

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static SupplierConnection anInstance = null;

public static SupplierConnection GetanInstance

{

get

{

if(SupplierConnection.anInstance == null)

{

SupplierConnection.anInstance = new SupplierConnection();

}

return SupplierConnection.anInstance;

}

}

public SupplierConnection()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

public List<Supplier> GetSuppliers()

{

int aSupplierId = -1; // Supplier ID

string aCompanyName = "n/a";

string aContactName = "n/a";

string aContactTitle = "n/a";

string aAddress = "n/a";

string aCity = "n/a";

string aRegion = "n/a";

string aPostalCode = "n/a";

string aCountry = "n/a";

string aPhone = "n/a";

string aFax = "n/a";

string aHomePage = "n/a";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT SupplierID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, " +

"Country, Phone, Fax, HomePage FROM Suppliers" + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aSupplierId = Convert.ToInt32(aReader["SupplierID"]);

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

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

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

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

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

aRegion = Convert.ToString(aReader["Region"]) ?? "n/a";

aPostalCode = (string) aReader["PostalCode"];

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

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

aFax = Convert.ToString(aReader["Fax"]) ?? "n/a";

aHomePage = Convert.ToString(aReader["HomePage"]) ?? "n/a";

// If the item is not in the dictionary

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

if(aSupplierMap.isInDictionary(aSupplierId) == false)

{

aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity,

aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

aSupplierMap.AddSupplier(aSupplier.SupplierId, aSupplier);

}

else

{

// the item already exists so get it

aSupplierMap.GetSupplier(aSupplierId);

}

// this adds the category to this list

// either way

aListOfSuppliers.Add(aSupplier);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aListOfSuppliers;

}

public void SaveAllSuppliers()

{

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

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "";

foreach(var s in dictionary.Keys)

{

int aSupplierId = dictionary[s].SupplierId; // Supplier ID

string aCompanyName = dictionary[s].CompanyName;

string aContactName = dictionary[s].ContactName;

string aContactTitle = dictionary[s].ContactTitle;

string aAddress = dictionary[s].Address;

string aCity = dictionary[s].City;

string aRegion = dictionary[s].Region;

string aPostalCode = dictionary[s].PostalCode;

string aCountry = dictionary[s].Country;

string aPhone = dictionary[s].Phone;

string aFax = dictionary[s].Fax;

string aHomePage = dictionary[s].HomePage;

// If the item is not in the dictionary

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

if(dictionary[s].IsDirty == true && dictionary[s].IsValid == true)

{

aSQL = "Update Suppliers";

aSQL = aSQL + "SET Company Name = " + aCompanyName + ", ";

aSQL = aSQL + "Contact Title = " + aContactTitle + " ";

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

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

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

aSQL = aSQL + "Postal Code = " + aPostalCode + " ";

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

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

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

aSQL = aSQL + "Home Page = " + aHomePage + " ";

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

aCommand.CommandText = aSQL;

aCommand.ExecuteNonQuery();

}

}

// close the connection

aConnection.Close();

}

public List<Supplier> GetBySuppliers(int Id)

{

int aSupplierId = -1; // Supplier ID

string aCompanyName = "n/a";

string aContactName = "n/a";

string aContactTitle = "n/a";

string aAddress = "n/a";

string aCity = "n/a";

string aRegion = "n/a";

string aPostalCode = "n/a";

string aCountry = "n/a";

string aPhone = "n/a";

string aFax = "n/a";

string aHomePage = "n/a";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT SupplierID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, " +

"Country, Phone, Fax, HomePage FROM Suppliers WHERE SupplierID = " + Id + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aSupplierId = Convert.ToInt32(aReader["SupplierID"]);

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

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

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

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

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

aRegion = Convert.ToString(aReader["Region"]) ?? "n/a";

aPostalCode = (string) aReader["PostalCode"];

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

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

aFax = Convert.ToString(aReader["Fax"]) ?? "n/a";

aHomePage = Convert.ToString(aReader["HomePage"]) ?? "n/a";

// If the item is not in the dictionary

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

if(aSupplierMap.isInDictionary(aSupplierId) == false)

{

aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity,

aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

aSupplierMap.AddSupplier(aSupplier.SupplierId, aSupplier);

}

else

{

// the item already exists so get it

aSupplierMap.GetSupplier(aSupplierId);

}

// this adds the category to this list

// either way

aBySuppliers.Add(aSupplier);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aBySuppliers;

}

}

}

**NWAbstractFactory Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This code is one of the abstract factorys. This abstract class creates two of

\* the abstract mehtods that revolves around creating the DataSet and Reader for the Factory Patterns.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Northwind.Models

{

public abstract class NWAbstractFactory

{

//Two abstract methods

public abstract INWDataReader CreateReader(string aSQL);

public abstract INWDataSet CreateDataSet(string aSQL);

}

}

**INWDataReader Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to get and assign the data for the datareader for an interface.

\*/

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 Northwind.Models

{

public interface INWDataReader

{

IDataReader aDataReader

{

get;

set;

}

}

}

**INWDataSet Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This is an interface for the DataSet Factory. It contains information for the factory that will not change.

\* This class would only be updated if all other options where used and there was no other way to update the information.

\*/

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 Northwind.Models

{

public interface INWDataSet

{

DataSet aDataSet

{

get;

set;

}

}

}

**NWOdbcDataReader Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class implements the data from DataReader so it doesn't damage the whole application. Sort of like the middle man of the operation. This class is the odbc datareader version.

\*

\*/

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 Northwind.Models

{

public class NWOdbcDataReader:INWDataReader

{

private IDataReader aReader;

public IDataReader aDataReader

{

get

{

return this.aReader;

}

set

{

this.aReader = value;

}

}

}

}

**NWOdbcDataSet Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This is the odbc version for the dataset class. This class implements the data from the INWDataSet class.

\*/

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 Northwind.Models

{

public class NWOdbcDataSet : INWDataSet

{

DataSet aSet = new DataSet();

public DataSet aDataSet

{

get

{

return this.aSet;

}

set

{

this.aSet = value;

}

}

}

}

**NWOdbcFactory Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This factory will take the information from the NWDataSet and the NWReader which will implement the

\* interfaces. As well as overrides the methods in the abstract factory. The functioning code and the code that

\* is most often updated would go in this class.

\*/

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 Northwind.Models

{

public class NWOdbcFactory

{

private NWOdbcDataReader aNWReader = new NWOdbcDataReader();

private NWOdbcDataSet aNWDataSet = new NWOdbcDataSet();

public 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\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\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 INWDataSet CreatDataSet(string aSQL)

{

// create the connection object

OleDbConnection aConnection = new OleDbConnection();

// set the connection

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data;

Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\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;

}

}

}

**NWOleDbDataReader Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This is the OleDb data for DataReader portion of the Factory. It will implement the INWDataReader

\* interface.

\*/

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 Northwind.Models

{

public class NWOleDbDataReader:INWDataReader

{

private IDataReader aReader;

public IDataReader aDataReader

{

get

{

return this.aReader;

}

set

{

this.aReader = value;

}

}

}

}

**NWOleDbDataSet Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This is the OleDb data that runs the DataSet portion of the Factory. It also implements the INWDataSet

\* interface into the program.

\*/

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 Northwind.Models

{

public class NWOleDbDataSet:INWDataSet

{

DataSet aSet = new DataSet();

public DataSet aDataSet

{

get

{

return this.aSet;

}

set

{

this.aSet = value;

}

}

}

}

**NWOleDbFactory Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This factory takes the NWDataSet and the NWReader which both will implement the interaces and

\* overrides the methods in the abstract factory.

\*/

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 Northwind.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\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\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

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\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;

}

}

}

**OleDbUtilityLoader Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class pulls the data out of the database and puts it in lists so that it can be displayed in the output later. The lists are filtered through command strings. This class isn’t in use anymore.

\*/

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 Northwind.Models

{

public class OleDbUtilityLoader

{

// Call Lists

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

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

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

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

// By Selection

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

// Call Variables

private Category aCategory = null;

private Supplier aSupplier = null;

private Shipper aShipper = null;

private Product aProduct = null;

string aSQL = "n/a";

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static OleDbUtilityLoader anInstance = null;

public static OleDbUtilityLoader GetanInstance

{

get

{

if(OleDbUtilityLoader.anInstance == null)

{

OleDbUtilityLoader.anInstance = new OleDbUtilityLoader();

}

return OleDbUtilityLoader.anInstance;

}

}

public OleDbUtilityLoader()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

// Get Category

public List<Category> GetCategories()

{

int aCategoryId = -1;

string aCategoryName = "n/a";

string aDescription = "n/a";

// Set the SQL statement

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

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

//Loops through information from the database

while(aReader.Read())

{

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

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

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

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

aListOfCategories.Add(aCategory);

}

// Return aListOfCategories

return aListOfCategories;

}

// Get Supplier

public List<Supplier> GetSuppliers()

{

int aSupplierId = -1; // Supplier ID

string aCompanyName = "n/a";

string aContactName = "n/a";

string aContactTitle = "n/a";

string aAddress = "n/a";

string aCity = "n/a";

string aRegion = "n/a";

string aPostalCode = "n/a";

string aCountry = "n/a";

string aPhone = "n/a";

string aFax = "n/a";

string aHomePage = "n/a";

// Set the SQL statement

aSQL = "SELECT SupplierID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, " +

"Country, Phone, Fax, HomePage FROM Suppliers" + ";";

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

//Loop through database information

while(aReader.Read())

{

aSupplierId = Convert.ToInt32(aReader["SupplierID"]);

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

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

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

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

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

aRegion = Convert.ToString(aReader["Region"]) ?? "n/a";

aPostalCode = (string) aReader["PostalCode"];

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

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

aFax = Convert.ToString(aReader["Fax"]) ?? "n/a";

aHomePage = Convert.ToString(aReader["HomePage"]) ?? "n/a";

aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity,

aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

// this adds the category to this list

// either way

aListOfSuppliers.Add(aSupplier);

}

// Return aListOfCategories

return aListOfSuppliers;

}

// Get Shippers

public List<Shipper> GetShippers()

{

int aShipperId = -1; // Shipper Number

string aCompanyName = "n/a";

string aPhone = "n/a";

// Set the SQL statement

aSQL = "SELECT ShipperID, CompanyName, Phone FROM Shippers" + ";";

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

while(aReader.Read())

{

aShipperId = Convert.ToInt32(aReader["ShipperID"]);

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

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

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

// this adds the category to this list

// either way

aListOfShipper.Add(aShipper);

}

// Return aListOfCategories

return aListOfShipper;

}

// Get Price

public List<Product> GetProducts()

{

int aProductId = -1; // only one with a get

string aProductName = "n/a";

string aSupplierId = "n/a";

string aCategoryId = "n/a";

string aQuantityPerUnit = "n/a";

double aUnitPrice = 1000000000;

int aUnitsInStock = -1;

int aUnitsOnOrder = -1;

int aReorderLevel = -1;

bool aDiscontinued = false;

// Set the SQL statement

aSQL = "SELECT Products.ProductID, Products.ProductName, Suppliers.CompanyName, Categories.CategoryName, " +

"Products.QuantityPerUnit, Products.UnitPrice, Products.UnitsInStock, Products.UnitsOnOrder, Products.ReorderLevel, " +

"Products.Discontinued " +

"FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID) INNER JOIN Suppliers ON " +

"Products.SupplierID = Suppliers.SupplierID) " +

"ORDER BY Products.ProductID ASC;";

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

//Loop through database information

while(aReader.Read())

{

aProductId = Convert.ToInt32(aReader["ProductID"]);

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

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

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

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

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

aUnitsInStock = Convert.ToInt32(aReader["UnitsInStock"]);

aUnitsOnOrder = Convert.ToInt32(aReader["UnitsOnOrder"]);

aReorderLevel = Convert.ToInt32(aReader["ReorderLevel"]);

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

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice,

aUnitsInStock, aUnitsOnOrder, aReorderLevel, aDiscontinued);

aListOfProduct.Add(aProduct);

}

// Return aListOfCategories

return aListOfProduct;

}

// List Product

public List<Product> GetByProducts(double pricemin, double pricemax)

{

int aProductId = -1; // only one with a get

string aProductName = "n/a";

string aSupplierId = "n/a";

string aCategoryId = "n/a";

string aQuantityPerUnit = "n/a";

double aUnitPrice = 1000000000;

int aUnitsInStock = -1;

int aUnitsOnOrder = -1;

int aReorderLevel = -1;

bool aDiscontinued = false;

// Set the SQL statement

aSQL = "SELECT Products.ProductID, Products.ProductName, Suppliers.CompanyName, Categories.CategoryName, " +

"Products.QuantityPerUnit, Products.UnitPrice, Products.UnitsInStock, Products.UnitsOnOrder, Products.ReorderLevel, " +

"Products.Discontinued " +

"FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID) INNER JOIN Suppliers ON " +

"Products.SupplierID = Suppliers.SupplierID) " +

"WHERE Products.UnitPrice >= " + pricemin + " AND Products.UnitPrice <= " + pricemax + ";";

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

while(aReader.Read())

{

aProductId = Convert.ToInt32(aReader["ProductID"]);

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

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

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

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

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

aUnitsInStock = Convert.ToInt32(aReader["UnitsInStock"]);

aUnitsOnOrder = Convert.ToInt32(aReader["UnitsOnOrder"]);

aReorderLevel = Convert.ToInt32(aReader["ReorderLevel"]);

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

aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice,

aUnitsInStock, aUnitsOnOrder, aReorderLevel, aDiscontinued);

// this adds the category to this list

// either way

aByProduct.Add(aProduct);

}

// Return aListOfCategories

return aByProduct;

}

}

}

**FactoryCategoryConnection Class**

/\*

\* Author: Chris Carrier

\* Date: 5/2/15

\* Information: This class pulls the data out of the database for the Factory CategoryConnection and puts it in lists so that

\* it can be displayed in the output later. The lists are filtered through command strings.

\*/

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 Northwind.Models

{

public class FactoryCategoryConnection

{

// Call Variables

private Category aCategory = null;

// Call Lists

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

// By Selection

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

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static FactoryCategoryConnection anInstance = null;

public static FactoryCategoryConnection GetanInstance

{

get

{

if(FactoryCategoryConnection.anInstance == null)

{

FactoryCategoryConnection.anInstance = new FactoryCategoryConnection();

}

return FactoryCategoryConnection.anInstance;

}

}

public FactoryCategoryConnection()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

// Get Category

public List<Category> GetCategories()

{

int aCategoryId = -1;

string aCategoryName = "n/a";

string aDescription = "n/a";

// Set the SQL statement

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

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

//Loops through information from the database

while(aReader.Read())

{

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

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

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

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

aListOfCategories.Add(aCategory);

}

// Return aListOfCategories

return aListOfCategories;

}

}

}

**FactoryProductConnection Class**

/\*

\* Author: Chris Carrier

\* Date: 5/2/15

\* Information: This class pulls the data out of the database for the Factory ProductConnection and puts it in lists so that

\* it can be displayed in the output later. The lists are filtered through command strings.

\*/

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 Northwind.Models

{

public class FactoryProductConnection

{

// Call Lists

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

// By Selection

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

// Call Variables

private Product aProduct = null;

string aSQL = "n/a";

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static FactoryProductConnection anInstance = null;

public static FactoryProductConnection GetanInstance

{

get

{

if(FactoryProductConnection.anInstance == null)

{

FactoryProductConnection.anInstance = new FactoryProductConnection();

}

return FactoryProductConnection.anInstance;

}

}

public FactoryProductConnection()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

// Get Price

public List<Product> GetProducts()

{

int aProductId = -1; // only one with a get

string aProductName = "n/a";

string aSupplierId = "n/a";

string aCategoryId = "n/a";

string aQuantityPerUnit = "n/a";

double aUnitPrice = 1000000000;

int aUnitsInStock = -1;

int aUnitsOnOrder = -1;

int aReorderLevel = -1;

bool aDiscontinued = false;

// Set the SQL statement

aSQL = "SELECT Products.ProductID, Products.ProductName, Suppliers.CompanyName, Categories.CategoryName, " +

"Products.QuantityPerUnit, Products.UnitPrice, Products.UnitsInStock, Products.UnitsOnOrder, Products.ReorderLevel, " +

"Products.Discontinued " +

"FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID) INNER JOIN Suppliers ON " +

"Products.SupplierID = Suppliers.SupplierID) " +

"ORDER BY Products.ProductID ASC;";

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

//Loop through database information

while(aReader.Read())

{

aProductId = Convert.ToInt32(aReader["ProductID"]);

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

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

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

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

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

aUnitsInStock = Convert.ToInt32(aReader["UnitsInStock"]);

aUnitsOnOrder = Convert.ToInt32(aReader["UnitsOnOrder"]);

aReorderLevel = Convert.ToInt32(aReader["ReorderLevel"]);

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

Product aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice,

aUnitsInStock, aUnitsOnOrder, aReorderLevel, aDiscontinued);

aListOfProduct.Add(aProduct);

}

// Return aListOfCategories

return aListOfProduct;

}

// List Product

public List<Product> GetByProducts(double pricemin, double pricemax)

{

int aProductId = -1; // only one with a get

string aProductName = "n/a";

string aSupplierId = "n/a";

string aCategoryId = "n/a";

string aQuantityPerUnit = "n/a";

double aUnitPrice = 1000000000;

int aUnitsInStock = -1;

int aUnitsOnOrder = -1;

int aReorderLevel = -1;

bool aDiscontinued = false;

// Set the SQL statement

aSQL = "SELECT Products.ProductID, Products.ProductName, Suppliers.CompanyName, Categories.CategoryName, " +

"Products.QuantityPerUnit, Products.UnitPrice, Products.UnitsInStock, Products.UnitsOnOrder, Products.ReorderLevel, " +

"Products.Discontinued " +

"FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID) INNER JOIN Suppliers ON " +

"Products.SupplierID = Suppliers.SupplierID) " +

"WHERE Products.UnitPrice >= " + pricemin + " AND Products.UnitPrice <= " + pricemax + ";";

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

while(aReader.Read())

{

aProductId = Convert.ToInt32(aReader["ProductID"]);

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

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

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

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

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

aUnitsInStock = Convert.ToInt32(aReader["UnitsInStock"]);

aUnitsOnOrder = Convert.ToInt32(aReader["UnitsOnOrder"]);

aReorderLevel = Convert.ToInt32(aReader["ReorderLevel"]);

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

aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice,

aUnitsInStock, aUnitsOnOrder, aReorderLevel, aDiscontinued);

// this adds the category to this list

// either way

aByProduct.Add(aProduct);

}

// Return aListOfCategories

return aByProduct;

}

}

}

**FactoryShipperConnection Class**

/\*

\* Author: Chris Carrier

\* Date: 5/2/15

\* Information: This class pulls the data out of the database for the Factory ShipperConnection and puts it in lists so that

\* it can be displayed in the output later. The lists are filtered through command strings.

\*/

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 Northwind.Models

{

public class FactoryShipperConnection

{

// Call Lists

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

// Call Variables

private Shipper aShipper = null;

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static FactoryShipperConnection anInstance = null;

public static FactoryShipperConnection GetanInstance

{

get

{

if(FactoryShipperConnection.anInstance == null)

{

FactoryShipperConnection.anInstance = new FactoryShipperConnection();

}

return FactoryShipperConnection.anInstance;

}

}

public FactoryShipperConnection()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

// Get Shippers

public List<Shipper> GetShippers()

{

int aShipperId = -1; // Shipper Number

string aCompanyName = "n/a";

string aPhone = "n/a";

// Set the SQL statement

string aSQL = "SELECT ShipperID, CompanyName, Phone FROM Shippers" + ";";

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

while(aReader.Read())

{

aShipperId = Convert.ToInt32(aReader["ShipperID"]);

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

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

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

// this adds the category to this list

// either way

aListOfShipper.Add(aShipper);

}

// Return aListOfCategories

return aListOfShipper;

}

}

}

**FactorySupplierConnection Class**

/\*

\* Author: Chris Carrier

\* Date: 5/2/15

\* Information: This class pulls the data out of the database for the Factory SupplierConnection and puts it in lists so that

\* it can be displayed in the output later. The lists are filtered through command strings.

\*/

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 Northwind.Models

{

public class FactorySupplierConnection

{

// Declare Variables

private Supplier aSupplier = null;

// Call List

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

// Call By Selection

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

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static FactorySupplierConnection anInstance = null;

public static FactorySupplierConnection GetanInstance

{

get

{

if(FactorySupplierConnection.anInstance == null)

{

FactorySupplierConnection.anInstance = new FactorySupplierConnection();

}

return FactorySupplierConnection.anInstance;

}

}

public FactorySupplierConnection()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

// Get Supplier

public List<Supplier> GetSuppliers()

{

int aSupplierId = -1; // Supplier ID

string aCompanyName = "n/a";

string aContactName = "n/a";

string aContactTitle = "n/a";

string aAddress = "n/a";

string aCity = "n/a";

string aRegion = "n/a";

string aPostalCode = "n/a";

string aCountry = "n/a";

string aPhone = "n/a";

string aFax = "n/a";

string aHomePage = "n/a";

string aSQL = "SELECT SupplierID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, " +

"Country, Phone, Fax, HomePage FROM Suppliers" + ";";

//Implement the factories and create the reader

NWAbstractFactory aFactory = new NWOleDBFactory();

//NWAbstractFactory aFactory = new NWOleDbFactory();

INWDataReader aNWReader = aFactory.CreateReader(aSQL);

IDataReader aReader = aNWReader.aDataReader;

//Loop through database information

while(aReader.Read())

{

aSupplierId = Convert.ToInt32(aReader["SupplierID"]);

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

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

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

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

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

aRegion = Convert.ToString(aReader["Region"]) ?? "n/a";

aPostalCode = (string) aReader["PostalCode"];

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

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

aFax = Convert.ToString(aReader["Fax"]) ?? "n/a";

aHomePage = Convert.ToString(aReader["HomePage"]) ?? "n/a";

aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity,

aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

// this adds the category to this list

// either way

aListOfSuppliers.Add(aSupplier);

}

// Return aListOfCategories

return aListOfSuppliers;

}

}

}

**DBConnection Class**

/\*

\* Author: Chris Carrier

\* Date: 5/2/15

\* Information: This class isn't in use anymore. This class pulls the data out of the database for the Connection and puts

\* it in lists so that it can be displayed in the output later. The lists are filtered through command strings.

\*/

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 Northwind.Models

{

public class DBConnection

{

/\* create a List \*/

// Call Lists

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

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

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

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

// By Selection

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

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

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

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

// Call Variables

private Category aCategory = null;

private Supplier aSupplier = null;

private Shipper aShipper = null;

private Product aProduct = null;

// Call Identity Map

private CategoryIdentityMap aCategoryMap = new CategoryIdentityMap();

private SupplierIdentityMap aSupplierMap = new SupplierIdentityMap();

private ShipperIdentityMap aShipperMap = new ShipperIdentityMap();

private ProductIdentityMap aProductMap = new ProductIdentityMap();

// Call Connections

private static OleDbConnection aConnection = new OleDbConnection();

private static OleDbCommand aCommand = aConnection.CreateCommand();

private static OleDbDataAdapter anAdapter = new OleDbDataAdapter(aCommand);

private static DataSet aDataSet = new DataSet();

private static DBConnection anInstance = null;

public static DBConnection GetanInstance

{

get

{

if(DBConnection.anInstance == null)

{

DBConnection.anInstance = new DBConnection();

}

return DBConnection.anInstance;

}

}

public DBConnection()

{

// This is private which prevents any other code

// from creating one.

// No code anywhere -- EXCEPT from within

// this class -- can create an object

// from this class

// set the connection string

aConnection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\Users\Chris\Desktop\Visual Studio Programming\Visual Studio 2013\Projects\Object\_Oriented\_Programming\Northwind Part 3\Northwind\Data\Northwind.accdb";

}

// Get DataSet

public static DataSet GetDataSet

{

get

{

return aDataSet;

}

}

public List<Category> GetCategories()

{

int aCategoryId = -1;

string aCategoryName = "n/a";

string aDescription = "n/a";

// 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())

{

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

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

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

// If the item is not in the dictionary

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

if (aCategoryMap.isInDictionary(aCategoryId) == false)

{

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

aCategoryMap.AddCategory(aCategory.CategoryId, aCategory);

}

else

{

// the item already exists so get it

aCategoryMap.GetCategory(aCategoryId);

}

// this adds the category to this list

// either way

aByCategories.Add(aCategory);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aByCategories;

}

public void SaveAllCategories()

{

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

// 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 new item and add it to the map

if (dictionary[c].IsDirty == true && dictionary[c].IsValid == 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();

}

public List<Category> GetByCategories(int Id)

{

int aCategoryId = -1;

string aCategoryName = "n/a";

string aDescription = "n/a";

// 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 = " + Id + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

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

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

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

// If the item is not in the dictionary

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

if(aCategoryMap.isInDictionary(aCategoryId) == false)

{

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

aCategoryMap.AddCategory(aCategory.CategoryId, aCategory);

}

else

{

// the item already exists so get it

aCategoryMap.GetCategory(aCategoryId);

}

// this adds the category to this list

// either way

aByCategories.Add(aCategory);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aByCategories;

}

public List<Supplier> GetSuppliers()

{

int aSupplierId = -1; // Supplier ID

string aCompanyName = "n/a";

string aContactName = "n/a";

string aContactTitle = "n/a";

string aAddress = "n/a";

string aCity = "n/a";

string aRegion = "n/a";

string aPostalCode = "n/a";

string aCountry = "n/a";

string aPhone = "n/a";

string aFax = "n/a";

string aHomePage = "n/a";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT SupplierID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, " +

"Country, Phone, Fax, HomePage FROM Suppliers" + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aSupplierId = Convert.ToInt32(aReader["SupplierID"]);

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

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

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

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

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

aRegion = Convert.ToString(aReader["Region"]) ?? "n/a";

aPostalCode = (string)aReader["PostalCode"];

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

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

aFax = Convert.ToString(aReader["Fax"]) ?? "n/a";

aHomePage = Convert.ToString(aReader["HomePage"]) ?? "n/a";

// If the item is not in the dictionary

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

if(aSupplierMap.isInDictionary(aSupplierId) == false)

{

aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName,aContactTitle, aAddress, aCity,

aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

aSupplierMap.AddSupplier(aSupplier.SupplierId, aSupplier);

}

else

{

// the item already exists so get it

aSupplierMap.GetSupplier(aSupplierId);

}

// this adds the category to this list

// either way

aListOfSuppliers.Add(aSupplier);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aListOfSuppliers;

}

public void SaveAllSuppliers()

{

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

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "";

foreach(var s in dictionary.Keys)

{

int aSupplierId = dictionary[s].SupplierId; // Supplier ID

string aCompanyName = dictionary[s].CompanyName;

string aContactName = dictionary[s].ContactName;

string aContactTitle = dictionary[s].ContactTitle;

string aAddress = dictionary[s].Address;

string aCity = dictionary[s].City;

string aRegion = dictionary[s].Region;

string aPostalCode = dictionary[s].PostalCode;

string aCountry = dictionary[s].Country;

string aPhone = dictionary[s].Phone;

string aFax = dictionary[s].Fax;

string aHomePage =dictionary[s].HomePage;

// If the item is not in the dictionary

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

if(dictionary[s].IsDirty == true && dictionary[s].IsValid == true)

{

aSQL = "Update Suppliers";

aSQL = aSQL + "SET Company Name = " + aCompanyName + ", ";

aSQL = aSQL + "Contact Title = " + aContactTitle + " ";

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

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

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

aSQL = aSQL + "Postal Code = " + aPostalCode + " ";

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

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

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

aSQL = aSQL + "Home Page = " + aHomePage + " ";

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

aCommand.CommandText = aSQL;

aCommand.ExecuteNonQuery();

}

}

// close the connection

aConnection.Close();

}

public List<Supplier> GetBySuppliers(int Id)

{

int aSupplierId = -1; // Supplier ID

string aCompanyName = "n/a";

string aContactName = "n/a";

string aContactTitle = "n/a";

string aAddress = "n/a";

string aCity = "n/a";

string aRegion = "n/a";

string aPostalCode = "n/a";

string aCountry = "n/a";

string aPhone = "n/a";

string aFax = "n/a";

string aHomePage = "n/a";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT SupplierID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, " +

"Country, Phone, Fax, HomePage FROM Suppliers WHERE SupplierID = " + Id + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aSupplierId = Convert.ToInt32(aReader["SupplierID"]);

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

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

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

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

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

aRegion = Convert.ToString(aReader["Region"]) ?? "n/a";

aPostalCode = (string) aReader["PostalCode"];

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

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

aFax = Convert.ToString(aReader["Fax"]) ?? "n/a";

aHomePage = Convert.ToString(aReader["HomePage"]) ?? "n/a";

// If the item is not in the dictionary

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

if(aSupplierMap.isInDictionary(aSupplierId) == false)

{

aSupplier = new Supplier(aSupplierId, aCompanyName, aContactName, aContactTitle, aAddress, aCity,

aRegion, aPostalCode, aCountry, aPhone, aFax, aHomePage);

aSupplierMap.AddSupplier(aSupplier.SupplierId, aSupplier);

}

else

{

// the item already exists so get it

aSupplierMap.GetSupplier(aSupplierId);

}

// this adds the category to this list

// either way

aBySuppliers.Add(aSupplier);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aBySuppliers;

}

// Get Shippers

public List<Shipper> GetShippers()

{

int aShipperId = -1; // Shipper Number

string aCompanyName = "n/a";

string aPhone = "n/a";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT ShipperID, CompanyName, Phone FROM Shippers" + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aShipperId = Convert.ToInt32(aReader["ShipperID"]);

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

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

// If the item is not in the dictionary

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

if(aShipperMap.isInDictionary(aShipperId) == false)

{

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

aShipperMap.AddShipper(aShipper.ShipperId, aShipper);

}

else

{

// the item already exists so get it

aShipperMap.GetShipper(aShipperId);

}

// this adds the category to this list

// either way

aListOfShipper.Add(aShipper);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aListOfShipper;

}

public void SaveAllShippers()

{

Dictionary<int, Shipper> dictionary = aShipperMap.GetDictionary();

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "";

foreach(var s in dictionary.Keys)

{

int aShipperId = dictionary[s].ShipperId; // Supplier ID

string aCompanyName = dictionary[s].CompanyName;

string aPhone = dictionary[s].Phone;

// If the item is not in the dictionary

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

if(dictionary[s].IsDirty == true && dictionary[s].IsValid == true)

{

aSQL = "Update Shippers";

aSQL = aSQL + "SET Company Name = " + aCompanyName + ", ";

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

aSQL = aSQL + "WHERE ShipperId = " + aShipperId;

aCommand.CommandText = aSQL;

aCommand.ExecuteNonQuery();

}

}

// close the connection

aConnection.Close();

}

public List<Shipper> GetByShippers(int id)

{

int aShipperId = -1; // Shipper Number

string aCompanyName = "n/a";

string aPhone = "n/a";

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT ShipperID, CompanyName, Phone FROM Shippers WHERE ShipperID = " + id + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aShipperId = Convert.ToInt32(aReader["ShipperID"]);

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

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

// If the item is not in the dictionary

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

if(aShipperMap.isInDictionary(aShipperId) == false)

{

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

aShipperMap.AddShipper(aShipper.ShipperId, aShipper);

}

else

{

// the item already exists so get it

aShipperMap.GetShipper(aShipperId);

}

// this adds the category to this list

// either way

aByShipper.Add(aShipper);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aByShipper;

}

// Get Price

public List<Product> GetProduct()

{

int aProductId = -1; // only one with a get

string aProductName = "n/a";

string aSupplierId = "n/a";

string aCategoryId = "n/a";

string aQuantityPerUnit = "n/a";

double aUnitPrice = 1000000000;

int aUnitsInStock = -1;

int aUnitsOnOrder = -1;

int aReorderLevel = -1;

bool aDiscontinued = false;

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT Products.ProductID, Products.ProductName, Suppliers.CompanyName, Categories.CategoryName, " +

"Products.QuantityPerUnit, Products.UnitPrice, Products.UnitsInStock, Products.UnitsOnOrder, Products.ReorderLevel, " +

"Products.Discontinued " +

"FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID) INNER JOIN Suppliers ON " +

"Products.SupplierID = Suppliers.SupplierID) " +

"ORDER BY Products.ProductID ASC;";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aProductId = Convert.ToInt32(aReader["ProductID"]);

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

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

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

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

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

aUnitsInStock = Convert.ToInt32(aReader["UnitsInStock"]);

aUnitsOnOrder = Convert.ToInt32(aReader["UnitsOnOrder"]);

aReorderLevel = Convert.ToInt32(aReader["ReorderLevel"]);

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

// If the item is not in the dictionary

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

if(aProductMap.isInDictionary(aProductId) == false)

{

aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice,

aUnitsInStock, aUnitsOnOrder, aReorderLevel, aDiscontinued);

aProductMap.AddProduct(aProduct.ProductId, aProduct);

}

else

{

// the item already exists so get it

aProductMap.GetProduct(aProductId);

}

// this adds the category to this list

// either way

aListOfProduct.Add(aProduct);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aListOfProduct;

}

public void SaveAllProducts()

{

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

// 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; // Product ID

string aProductName = dictionary[p].ProductName;

string aSupplierId = dictionary[p].SupplierId;

string 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 aDiscontinued = 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 && dictionary[p].IsValid == true)

{

aSQL = "Update Products";

aSQL = aSQL + "SET Product Name = " + aProductName + ", ";

aSQL = aSQL + "Company Name = " + aSupplierId + ", ";

aSQL = aSQL + "Category Name = " + aCategoryId + ", ";

aSQL = aSQL + "Quantity Per Unit = " + aQuantityPerUnit + ", ";

aSQL = aSQL + "WHERE UnitPrice = " + aUnitPrice + " ";

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

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

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

aSQL = aSQL + "Discontinued = " + aDiscontinued + " ";

aCommand.CommandText = aSQL;

aCommand.ExecuteNonQuery();

}

}

// close the connection

aConnection.Close();

}

public List<Product> GetByProducts(double pricemin, double pricemax)

{

int aProductId = -1; // only one with a get

string aProductName = "n/a";

string aSupplierId = "n/a";

string aCategoryId = "n/a";

string aQuantityPerUnit = "n/a";

double aUnitPrice = 1000000000;

int aUnitsInStock = -1;

int aUnitsOnOrder = -1;

int aReorderLevel = -1;

bool aDiscontinued = false;

// open the connection

aConnection.Open();

// create a command object

OleDbCommand aCommand = aConnection.CreateCommand();

// Set the SQL statement

string aSQL = "SELECT Products.ProductID, Products.ProductName, Suppliers.CompanyName, Categories.CategoryName, " +

"Products.QuantityPerUnit, Products.UnitPrice, Products.UnitsInStock, Products.UnitsOnOrder, Products.ReorderLevel, " +

"Products.Discontinued " +

"FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID) INNER JOIN Suppliers ON " +

"Products.SupplierID = Suppliers.SupplierID) " +

"WHERE Products.UnitPrice >= " + pricemin + " AND Products.UnitPrice <= " + pricemax + ";";

aCommand.CommandText = aSQL;

// run the SQL statement

OleDbDataReader aReader = aCommand.ExecuteReader();

while(aReader.Read())

{

aProductId = Convert.ToInt32(aReader["ProductID"]);

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

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

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

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

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

aUnitsInStock = Convert.ToInt32(aReader["UnitsInStock"]);

aUnitsOnOrder = Convert.ToInt32(aReader["UnitsOnOrder"]);

aReorderLevel = Convert.ToInt32(aReader["ReorderLevel"]);

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

// If the item is not in the dictionary

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

if(aProductMap.isInDictionary(aProductId) == false)

{

aProduct = new Product(aProductId, aProductName, aSupplierId, aCategoryId, aQuantityPerUnit, aUnitPrice,

aUnitsInStock, aUnitsOnOrder, aReorderLevel, aDiscontinued);

aProductMap.AddProduct(aProduct.ProductId, aProduct);

}

else

{

// the item already exists so get it

aProductMap.GetProduct(aProductId);

}

// this adds the category to this list

// either way

aByProduct.Add(aProduct);

}

// close the connection

aConnection.Close();

// Return aListOfCategories

return aByProduct;

}

}

}

**HomeController Class**

/\*

\* Author: Chris Carrier

\* Date: 1/15/15

\* Information: This class is to run the ASP.NET MVC setups. Which I believe is the Views that are created. This controller

\* runs the regular views. This class links up to the Category, Product, Shipper, and Supplier Connections.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Northwind.Models;

using System.Data;

using System.Data.OleDb;

namespace Northwind.Controllers

{

public class HomeController : Controller

{

// Home

// GET: Home

public ActionResult Index()

{

//Company aCompany = new Company();

Customer aCustomer = new Customer();

Shipper aShipper = new Shipper();

Supplier aSupplier = new Supplier();

DBConnection aConnection = new DBConnection();

// Category Connection

CategoryConnection aCategoryConnection = new CategoryConnection();

// Shipper Connection

ShipperConnection aShipperConnection = new ShipperConnection();

// Supplier Connection

SupplierConnection aSupplierConnection = new SupplierConnection();

// Product Connection

ProductConnection aProductConnection = new ProductConnection();

// Category List

List<Category> aListOfCategories = aCategoryConnection.GetCategories();

// Supplier List

List<Supplier> aListOfSuppliers = aConnection.GetSuppliers();

// Shipper List

List<Shipper> aListOfShippers = aConnection.GetShippers();

//ViewBag.Company = aCompany;

ViewBag.Customer = aCustomer;

ViewBag.Shipper = aShipper;

ViewBag.Supplier = aSupplier;

ViewBag.Categories = aListOfCategories;

return View();

}

// Category

// Category Index

public ActionResult CategoryIndex()

{

// DBConnection aConnection = new DBConnection();

CategoryConnection aCategoryConnection = new CategoryConnection();

List<Category> aCategories = aCategoryConnection.GetCategories();

ViewBag.Categories = aCategories;

return View();

}

// Category Files

public ActionResult CategoryFile()

{

// DBConnection aConnection = new DBConnection();

CategoryConnection aCategoryConnection = new CategoryConnection();

List<Category> aFileCategories = aCategoryConnection.GetCategories(); // Change this later

ViewBag.Categories = aFileCategories;

return View();

}

// Category Filter

public ActionResult CategoryFilter()

{

// DBConnection aConnection = new DBConnection();

CategoryConnection aCategoryConnection = new CategoryConnection();

List<Category> aListOfCategories = aCategoryConnection.GetCategories();

ViewBag.Categories = aListOfCategories;

return View();

}

// By Category

public ActionResult ByCategory(int Id)

{

// DBConnection aConnection = new DBConnection();

CategoryConnection aCategoryConnection = new CategoryConnection();

List<Category> aByCategories = aCategoryConnection.GetByCategories(Id);

ViewBag.Categories = aByCategories;

return View();

}

// Supplier

// Supplier Index

public ActionResult SupplierIndex()

{

// DBConnection aConnection = new DBConnection();

SupplierConnection aSupplierConnection = new SupplierConnection();

List<Supplier> aSupplier = aSupplierConnection.GetSuppliers();

ViewBag.Supplier = aSupplier;

return View();

}

// Supplier Files

public ActionResult SupplierFile()

{

// DBConnection aConnection = new DBConnection();

SupplierConnection aSupplierConnection = new SupplierConnection();

List<Supplier> aSupplierFile = aSupplierConnection.GetSuppliers();

ViewBag.Supplier = aSupplierFile;

return View();

}

// Supplier Filter

public ActionResult SupplierFilter()

{

// DBConnection aSupplierConnection = new DBConnection();

SupplierConnection aSupplierConnection = new SupplierConnection();

List<Supplier> aSupplierFilter = aSupplierConnection.GetSuppliers();

ViewBag.Supplier = aSupplierFilter;

return View();

}

// Supplier Filter

public ActionResult BySupplier(int Id)

{

// DBConnection aSupplierConnection = new DBConnection();

SupplierConnection aSupplierConnection = new SupplierConnection();

List<Supplier> aBySupplier = aSupplierConnection.GetBySuppliers(Id);

ViewBag.Supplier = aBySupplier;

return View();

}

// Shipper

// Shipper Index

public ActionResult ShipperIndex()

{

// DBConnection aConnection = new DBConnection();

ShipperConnection aShipperConnection = new ShipperConnection();

List<Shipper> aShipperFile = aShipperConnection.GetShippers();

ViewBag.Shipper = aShipperFile;

return View();

}

// Shipper File

public ActionResult ShipperFile()

{

// DBConnection aConnection = new DBConnection();

ShipperConnection aShipperConnection = new ShipperConnection();

List<Shipper> aShipperFile = aShipperConnection.GetShippers();

ViewBag.Shipper = aShipperFile;

return View();

}

// Shipper Filter

public ActionResult ShipperFilter()

{

// DBConnection aConnection = new DBConnection();

ShipperConnection aShipperConnection = new ShipperConnection();

List<Shipper> aShipperFile = aShipperConnection.GetShippers();

ViewBag.Shipper = aShipperFile;

return View();

}

// Get By Shipper Filter Files

public ActionResult ByShipper(int id)

{

// DBConnection aConnection = new DBConnection();

ShipperConnection aShipperConnection = new ShipperConnection();

List<Shipper> aByShipper = aShipperConnection.GetByShippers(id);

ViewBag.Shipper = aByShipper;

return View();

}

// Product

// Price Index

public ActionResult ProductIndex()

{

// DBConnection aConnection = new DBConnection();

ProductConnection aProductConnection = new ProductConnection();

List<Product> aProductFile = aProductConnection.GetProduct();

ViewBag.Product = aProductFile;

return View();

}

// Product File

public ActionResult ProductFile()

{

// DBConnection aConnection = new DBConnection();

ProductConnection aProductConnection = new ProductConnection();

List<Product> aProductFile = aProductConnection.GetProduct();

ViewBag.Product = aProductFile;

return View();

}

// Product Filter

public ActionResult ProductFilter()

{

// DBConnection aConnection = new Connection();

ProductConnection aProductConnection = new ProductConnection();

List<Product> aProductFile = aProductConnection.GetProduct();

ViewBag.Product = aProductFile;

return View();

}

// Get Price ByProduct Filter Files

public ActionResult ByProduct(double pricemin, double pricemax)

{

// DBConnection aConnection = new DBConnection();

ProductConnection aProductConnection = new ProductConnection();

List<Product> aByProduct = aProductConnection.GetByProducts(pricemin, pricemax);

ViewBag.Product = aByProduct;

return View();

}

}

}

**FactoryController Class**

/\*

\* Author: Chris Carrier

\* Date: 5/2/15

\* Information: This class is to run the ASP.NET MVC setups. Which I believe is the Views that are created. This controller runs the factory views. This class links up to the Factory version of Category, Product, Shipper, and Supplier Connections.

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Northwind.Models;

using System.Data.OleDb;

using System.Data.Odbc;

using System.Data.Sql;

using System.Data;

namespace Northwind.Controllers

{

public class FactoryController : Controller

{

public ActionResult FactoryIndex()

{

// Links

// Customer Link

Customer aCustomer = new Customer();

// Shipper Link

Shipper aShipper = new Shipper();

// Supplier Link

Supplier aSupplier = new Supplier();

// Product Link

Product aProduct = new Product();

// Connections

// Category Connection

FactoryCategoryConnection aFactoryCategoryConnection = new FactoryCategoryConnection();

// Shipper Connection

FactoryShipperConnection aFactoryShipperConnection = new FactoryShipperConnection();

// Supplier Connection

FactorySupplierConnection aFactorySupplierConnection = new FactorySupplierConnection();

// Product Connection

FactoryProductConnection aFactoryProductConnection = new FactoryProductConnection();

// Lists

// Category List

List<Category> aListOfCategories = aFactoryCategoryConnection.GetCategories();

// Supplier List

List<Supplier> aListOfSuppliers = aFactorySupplierConnection.GetSuppliers();

// Shipper List

List<Shipper> aListOfShippers = aFactoryShipperConnection.GetShippers();

// Shipper List

List<Product> aListOfProducts = aFactoryProductConnection.GetProducts();

// ViewBags

// Customer ViewBag

ViewBag.Customer = aCustomer;

// Shipper ViewBag

ViewBag.Shipper = aShipper;

// Supplier ViewBag

ViewBag.Supplier = aSupplier;

// Product ViewBag

ViewBag.Product = aProduct;

// Categories ViewBag

ViewBag.Categories = aListOfCategories;

return View();

}

// Category Files

public ActionResult CategoryFactory()

{

// DBConnection aConnection = new DBConnection();

FactoryCategoryConnection aFactoryCategory = new FactoryCategoryConnection();

List<Category> aFactoryCategoriesFile = aFactoryCategory.GetCategories();

ViewBag.Categories = aFactoryCategoriesFile;

return View();

}

// Shipper File

public ActionResult ShipperFactory()

{

// DBConnection aConnection = new DBConnection();

FactoryShipperConnection aFactoryShipper = new FactoryShipperConnection();

List<Shipper> aFactoryShipperFile = aFactoryShipper.GetShippers();

ViewBag.Shipper = aFactoryShipperFile;

return View();

}

// Supplier Files

public ActionResult SupplierFactory()

{

// DBConnection aConnection = new DBConnection();

FactorySupplierConnection aFactorySupplier = new FactorySupplierConnection();

List<Supplier> aFactorySupplierFile = aFactorySupplier.GetSuppliers();

ViewBag.Supplier = aFactorySupplierFile;

return View();

}

// Product File

public ActionResult ProductFactory()

{

// DBConnection aConnection = new DBConnection();

FactoryProductConnection aFactoryProductConnection = new FactoryProductConnection();

List<Product> aFactoryProductFile = aFactoryProductConnection.GetProducts();

ViewBag.Product = aFactoryProductFile;

return View();

}

}

}

**Index.cshtml**

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Home Index Output

-->

@{

ViewBag.Title = "Index";

}

<head>

<link type="text/css" rel="stylesheet" href="~/Content/style.css" />

<!--<script src="~/Scripts/javascript-style.js"></script>-->

<title>Index</title>

</head>

<h2>Index</h2>

<h2>Filter Items</h2>

<ul class="index-list">

<li><a href="~/Home/CategoryIndex">By Category</a></li>

<li><a href="~/Home/SupplierIndex">By Supplier</a></li>

<li><a href="~/Home/ShipperIndex">By Shipper</a></li>

<li><a href="~/Home/ProductIndex">By Product</a></li>

<li><a href="~/Factory/FactoryIndex">Factory Index</a></li>

<li><a href="~/Factory/CategoryFactory">Category Factory</a></li>

<li><a href="~/Factory/SupplierFactory">Supplier Factory</a></li>

<li><a href="~/Factory/ShipperFactory">Shipper Factory</a></li>

<li><a href="~/Factory/ProductFactory">Product Factory</a></li>

</ul>

**CategoryIndex.cshtml**

@model Northwind.Models.Category

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Category Index Output

-->

@{

ViewBag.Title = "Category Index";

}

<head>

<title>Category Index</title>

</head>

<h2>Categories</h2>

<a href="~/Home/CategoryFile">By File</a><br />

<a href="~/Home/CategoryFilter">By Category</a><br />

**CategoryFile.cshtml**

@model Northwind.Models.Category

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Category List Output

-->

@{

ViewBag.Title = "Category File";

}

<head>

<title>Category File</title>

</head>

<!-- By Files All list -->

<h2>Category File</h2>

@foreach(var c in ViewBag.Categories)

{

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

}

**CategoryFilter.cshtml**

@model Northwind.Models.Category

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Category Filter List Output

-->

@{

ViewBag.Title = "Category File";

}

<head>

<title>Category File</title>

</head>

<!-- By Files All list -->

<h2>Category File</h2>

@foreach(var c in ViewBag.Categories)

{

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

}

**ByCategory.cshtml**

@model Northwind.Models.Category

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Category Filter Selection Output

-->

@{

ViewBag.Title = "By Category";

}

<h2>By Category</h2>

@foreach(var filecategory in ViewBag.Categories)

{

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

}

**SupplierIndex.cshtml**

@model Northwind.Models.Supplier

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Supplier Index Output

-->

@{

ViewBag.Title = "Supplier Index";

}

<h2>Supplier Index</h2>

<a href="~/Home/SupplierFile">By File</a><br />

<a href="~/Home/SupplierFilter">By Supplier</a><br />

**SupplierFile.cshtml**

@model Northwind.Models.Supplier

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Supplier List Output

-->

@{

ViewBag.Title = "Supplier File";

}

<head>

<title>Supplier File</title>

</head>

<h2>Supplier File</h2>

@foreach(var s in ViewBag.Supplier)

{

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

}

**SupplierFilter.cshtml**

@model Northwind.Models.Supplier

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Supplier Filter List Output

-->

@{

ViewBag.Title = "Supplier Filter";

}

<h2>Supplier Filter</h2>

<!-- By Category -->

<table border="1">

@foreach(var s in ViewBag.Supplier)

{

<tr class="something">

<td>

<a href="BySupplier?id=@s.SupplierId">

@s.CompanyName

</a>

</td>

</tr>

}

</table>

**BySupplier.cshtml**

@model Northwind.Models.Supplier

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Supplier Filter Selection Output

-->

@{

ViewBag.Title = "By Supplier";

}

<h2>By Supplier</h2>

@foreach(var filesupplier in ViewBag.Supplier)

{

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

}

**ShipperIndex.cshtml**

@model Northwind.Models.Shipper

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Shipper Index Output

-->

@{

ViewBag.Title = "Shipper Index";

}

<h2>Shipper Index</h2>

<a href="~/Home/ShipperFile">By File</a><br />

<a href="~/Home/ShipperFilter">By Shipper</a><br />

**ShipperFile.cshtml**

@model Northwind.Models.Shipper

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Shipper List Output

-->

@{

ViewBag.Title = "Shipper File";

}

<head>

<title>Shipper File</title>

</head>

<!-- By Files All list -->

<h2>Shipper File</h2>

@foreach(var sh in ViewBag.Shipper)

{

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

}

**ShipperFilter.cshtml**

@model Northwind.Models.Shipper

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Shipper Filter List Output

-->

@{

ViewBag.Title = "Shipper Filter";

}

<h2>Shipper Filter</h2>

<!-- By Shipper -->

<table border="1">

@foreach(var sh in ViewBag.Shipper)

{

<tr class="something">

<td>

<a href="ByShipper?id=@sh.ShipperId">

@sh.CompanyName

</a>

</td>

</tr>

}

</table>

**ByShipper.cshtml**

@model Northwind.Models.Shipper

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Shipper Filter Selection Output

-->

@{

ViewBag.Title = "By Shipper";

}

<h2>By Shipper</h2>

@foreach(var fileshipper in ViewBag.Shipper)

{

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

}

**ProductIndex.cshtml**

@model Northwind.Models.Product

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Product Index Output

-->

@{

ViewBag.Title = "Product Index";

}

<h2>Product Index</h2>

<a href="~/Home/ProductFile">By File</a><br />

<a href="~/Home/ProductFilter">By Product</a><br />

**ProductFile.cshtml**

@model Northwind.Models.Product

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Product List Output

-->

@{

ViewBag.Title = "Product File";

}

<head>

<title>Product File</title>

</head>

<!-- By Files All list -->

<h2>Product File</h2>

@foreach(var p in ViewBag.Product)

{

@Html.Raw(p.ToString());<br />

}

**ProductFilter.cshtml**

@model Northwind.Models.Product

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Product Filter Output

-->

@{

ViewBag.Title = "Product Filter";

<link type="text/css" rel="stylesheet" href="~/Content/style.css" />

}

<h2>Product Filter</h2>

<!-- By Product -->

<form action="~/Home/ByProduct">

<fieldset>

<legend>Product Price Range</legend>

<label>Product Price Min:</label>

<input type="text" name="pricemin" class="product-min" id="product\_min" placeholder="Product Minimum" min="0"

required="required" /><br />

<label>Product Price Max:</label>

<input type="text" name="pricemax" class="product-max" id="product\_max" placeholder="Product Maximum"

required="required"/><br />

<div class="button-price">

<input type="submit" value="Submit" class="submit-price" />

<input type="reset" value="Reset" class="reset-price" /><br />

</div>

</fieldset>

</form><br />

**ByProduct.cshtml**

@model Northwind.Models.Product

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Product Filter Selection Output

-->

@{

ViewBag.Title = "By Product";

}

<h2>ByProduct</h2>

@foreach(var fileproduct in ViewBag.Product)

{

@Html.Raw(fileproduct.ToString());<br />

}

**FactoryIndex.cshtml**

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Home Index Factory Output

-->

@{

ViewBag.Title = "Factory Index";

}

<head>

<title>Factory Index</title>

</head>

<h2>Factory Index</h2>

<h2>Filter Items</h2>

<ul class="index-list">

<li><a href="~/Factory/CategoryFactory">Category Factory</a></li>

<li><a href="~/Factory/SupplierFactory">Supplier Factory</a></li>

<li><a href="~/Factory/ShipperFactory">Shipper Factory</a></li>

<li><a href="~/Factory/ProductFactory">Product Factory</a></li>

</ul>

**CategoryFactory.cshtml**

@model Northwind.Models.Category

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Category Factory List Output

-->

@{

ViewBag.Title = "Category Factory";

}

<h2>Category Factory</h2>

<!-- By Files All list -->

@foreach(var fc in ViewBag.Categories)

{

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

}

**SupplierFactory.cshtml**

@model Northwind.Models.Supplier

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Supplier Factory List Output

-->

@{

ViewBag.Title = "Supplier Factory";

}

<head>

<title>Supplier Factory</title>

</head>

<h2>Supplier Factory</h2>

@foreach(var sf in ViewBag.Supplier)

{

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

}

**ShipperFactory.cshtml**

@model Northwind.Models.Shipper

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Shipper Factory List Output

-->

@{

ViewBag.Title = "Shipper Factory";

}

<head>

<title>Shipper Factory</title>

</head>

<!-- By Files All list -->

<h2>Shipper Factory</h2>

@foreach(var sh in ViewBag.Shipper)

{

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

}

**ProductFactory.cshtml**

@model Northwind.Models.Product

<!-- Author: Chris Carrier

Date: 1/15/15

Information: Product Factory List Output

-->

@{

ViewBag.Title = "Product Factory";

}

<head>

<title>Product Factory</title>

</head>

<!-- By Files All list -->

<h2>Product Factory</h2>

@foreach(var p in ViewBag.Product)

{

@Html.Raw(p.ToString());<br />

}