Exception C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace HataYonetimi

{

public class Program

{

static void Main(string[] args)

{

try

{

//dene başarılı olan kodu yazarız. burada bir hata olursa burayı bırakıp

string[] students = new string[3] { "ömer", "yasemin", "büşra" };

students[3] = "Ezgi";

//hata olduğunda exception diye bir nesneye aktarılır. Exeption.mesa

}

catch (Exception e)

{

//kod buraya geçeçer.

Console.WriteLine(e.Message);//kullanıcıya biz bunu göstermeyiz. biz bu bilgileri hata olarak kullaılır.

//biz genellikle kullanıcıya göstermek yerine veritabanına veya başka yere loglarız. Sistemin hata verdiği yerlerde sistem niçin hata verdiini göre düzeltiriz. exception okumak önem arz eder.

} //burada catch artarsa işimiz vardır.

;

Console.ReadLine();

//oluşturulan exception hala risk oluşturur. oluşturulan hatalardan uygulama hakkında bilgi alabiliyoruz.

//çünkü oluşturulan exception sistem hakkında bilgi verir.

HandleException(() =>

{

Find();

}); //burada method göndermek istiyorum. parametre olarak. parametrenin türünün method olmasını kod blogun olmasını istiyorum. birçok dilde bu yoktur. belli başlı dilde mevcuttur. delege ile göndeririz.

}

//private static void Find()

//{

// throw new NotImplementedException();

//}

private static void HandleException(Action action)//parametresiz method bloğu herhangibibir retrun olmaz.

{

try

{

action.Invoke(); //ne yaptık methodu merkezi bir try catch gönderdik.

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

}

}

ADO.net giriş ve Veritabanı

Yaptığımzı yazılımların bir çoğu veriyi yönetmeyi barındırır. Veriler veritabanı denen ortamlarda tutulur. Veri saklanmak gerekir. Bir kullanıcı bir şey yaptığında diğer kullanıcı da bunu görmesi sınırlar içinde görmesi gerekir. C# ile bu veritabanına ulaşmak için.NET framework bulunan ADO.net kütüphanesidir. Adonet kütüphanesi öncesi veriyi yönetmek için bir ortama ihtiyaç vardır.

A screenshot of a computer

Description automatically generated with medium confidence gördüğümüz gibi SQL Server Object Explorer menüsü gelecektir.

VS Kurulan Express dediğimiz bir versiyonu kurulur. Geliştirilerin uygulama yazıp kodları test edildir. Işlemler bittiğinde gerçekveri tabanına taşınır ve orada sürdürülür. Geliştirici olduğumuz için kendi veritabanını kullanırı.z local isimli bir instance olan veritabanı görürüz.

Aşağıdaki ekrandaki gibi yeni bir veritabanı oluşturmak istesek; Database sağ tıklarız ve Add New Database deriz. A screenshot of a computer

Description automatically generated with medium confidence

Tablo oluşturduktan sonra (tarafım ETrade isimli database oluşturdu). Aşağıki gibi tablo eklemesi yaparız.

A screenshot of a computer

Description automatically generated with medium confidence

Veritabanarında her veritabanın eşsiz olarak verilen bir veri vardır. Bu diğerlerinden ayırt etmek için kullanırız. Kendi sığ kapalı alanda da kullanılabilinir genel anlamda geniş bir id olabilir. Id genelde integer türünde de olabiliinr. Bu id genellikle otomatik artısın diğe bir değer veririr. Bir bir artırarak şekilde ayarlayalım.

Özelliklere gelince; Identity specification altındaki ls identity True seçersek otomatik olarak artıracaktır.

Aşağıdaki resimde görülecektir.

A screenshot of a computer

Description automatically generated with medium confidence

CREATE TABLE [dbo].[Products]

(

[Id] INT NOT NULL PRIMARY KEY IDENTITY,

[Name] NCHAR(50) NOT NULL,

[UnitPrice] MONEY NOT NULL,

[StockAmount] INT NOT NULL

)

Yarattık; yani create ettik database

Şimdi biz ürünleri girmek için aşağıki view data seçeriz.

A screenshot of a computer

Description automatically generated with medium confidence

**Gibi ekran gelir.**

**Şimdi Windows Form üzerinde bu iş akışına devam edelim.**

A screenshot of a computer

Description automatically generated with medium confidence Solution üzerindeki Sağ tıklayıp Add-> New Project diyelim

Aşağıdaki ekran gelecektir. Windows Forms seçelim.

A screenshot of a computer

Description automatically generated with medium confidence Yukarıdaki gibi Windows Forms App(.net Framework) seçeriz.

Bir ürünü çekmek, güncellemek ve silmek…vs. oluşturmak istiyorum. Ürünleri listelemek istiyorum. Dolaysıyla bir grid ortmamında listeleyelbilirim.

View menüsünden toolbox seçeriz.

A screenshot of a computer

Description automatically generated

Seçeriz.

A screenshot of a computer

Description automatically generated with medium confidence

DataGridView seçeriz.

DataGrid yani DataViewGrid ile kullanıcıya göstermeye çalışırız ürünlerimizi. Ürünleri getirmeye çalışırız. AdoNetDemo içine class ekleyelim.

Class adı ProductDal diyelim. Data Access Interface anlamına gelir Dal. Tablonun kodsal karşılığına ihtiyaç vardır. İçinde bir sürü ürün vardır.

Şimdi de Product isimli sınfı oluşturalım

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

public class Product

{

public int id { get; set; }

public string name { get; set; }

public decimal unitPrace { get; set; }

public int StockAmount { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

public class ProductDal

{

public DataTable GetAll()

{

//iş yapacağız refactorin kod iyileştirmeye gireceğiz.

SqlConnection connection = new SqlConnection(@"server=(localdb)\ProjectModels;initial catalog=ETrade; integrated security=true");

//connection.Open(); //bağlantıyı açtık.

if (connection.State == ConnectionState.Closed)

{

connection.Open(); //eğer açık değilse aç.(kapalı ise aç)

}

//connection.Open();

//slq bir dili var ona göre yapıyoruz. bir tablodan veri çekmek

SqlCommand command = new SqlCommand("Select\* from Products", connection);

SqlDataReader reader = command.ExecuteReader();//referansı atamasını yapıyoruz.

DataTable dataTable = new DataTable();

dataTable.Load(reader);//IDataReadar türünde bir şey ver diyro. buraya direk reader verebiliriz.

reader.Close();//bağlantıyı kapat

connection.Close();

return dataTable;

}

}

}

Şimdi sql komutu oluşturmak çağırdığımz anlamı gelmez.

Datatable hem pahalı hem de eski bir teknolojidir. Bunun yerine kendimiz List<Product> yapısını kullanırız.

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

public class ProductDal

{

//public DataTable GetAll()

//{

// //iş yapacağız refactorin kod iyileştirmeye gireceğiz.

// SqlConnection connection = new SqlConnection(@"server=(localdb)\ProjectModels;initial catalog=ETrade; integrated security=true");

// //connection.Open(); //bağlantıyı açtık.

// if (connection.State == ConnectionState.Closed)

// {

// connection.Open(); //eğer açık değilse aç.(kapalı ise aç)

// }

// //connection.Open();

// //slq bir dili var ona göre yapıyoruz. bir tablodan veri çekmek

// SqlCommand command = new SqlCommand("Select\* from Products", connection);

// SqlDataReader reader = command.ExecuteReader();//referansı atamasını yapıyoruz.

// DataTable dataTable = new DataTable();

// dataTable.Load(reader);//IDataReadar türünde bir şey ver diyro. buraya direk reader verebiliriz.

// reader.Close();//bağlantıyı kapat

// connection.Close();

// return dataTable;

//}

public List<Product> GetAll2()

{

//iş yapacağız refactorin kod iyileştirmeye gireceğiz.

SqlConnection connection = new SqlConnection(@"server=(localdb)\ProjectModels;initial catalog=ETrade; integrated security=true");

//connection.Open(); //bağlantıyı açtık.

if (connection.State == ConnectionState.Closed)

{

connection.Open(); //eğer açık değilse aç.(kapalı ise aç)

}

//connection.Open();

//slq bir dili var ona göre yapıyoruz. bir tablodan veri çekmek

SqlCommand command = new SqlCommand("Select\* from Products", connection);

SqlDataReader reader = command.ExecuteReader();//referansı atamasını yapıyoruz.

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

while (reader.Read())//kayıtları tek tek oku ve okuduğun

{

Product product = new Product

{

id=Convert.ToInt32(reader["ID"]),

name = reader["Name"].ToString(),

StockAmount = Convert.ToInt32(reader["StockAmount"]),

unitPrace = Convert.ToDecimal(reader["UnitPrice"])

};

products.Add(product);//her product listeye ekliyorum.

}

reader.Close();//bağlantıyı kapat

connection.Close();

return products;

}

}

}

Bu kodla her okunan eleman product eklenecektir. O listeyi döneriz. GetAll2() listeli yönetmdir.

Yeni yazdığımız method dönecektir.

Ve Product yapısı değişmemiştir. Soyutlama yaptık ne de başka bir şey yaptık.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

public class Product

{

public int id { get; set; }

public string name { get; set; }

public decimal unitPrace { get; set; }

public int StockAmount { get; set; }

}

}

Bir şeyi okuyacak bir method yazabiliriz. Bu kodlama o kadar kısa halde kullanırız. Hazır Deppir gibi yapılar kullanırız. Olayımız süreci anlamaktır. Oerem nasıl çalıştığını anlarız.

From 1 geliriz ve bir data gireceğimz alan oluturmak istiyorum.

Ürün ismi, stok adedi,ürün fiyatı vardır.

Buttona çift tıklandığın bizi direk olarak yönlendireceği sayfaya ekleme yapacaktır.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

ProductDal \_productDal = new ProductDal();

private void Form1\_Load(object sender, EventArgs e)

{

dgwProducts.DataSource = \_productDal.GetAll2();

}

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

{

//buttona tıklama event çift tılanınca oluşacak eventter.

//BİZİM KODUMUZ productdal içinde idi.

\_productDal.Add(new Product

{

name=tbxName.Text,

unitPrice=Convert.ToDecimal(tbxUnitPrice.Text),

StockAmount=Convert.ToInt32(tbxStockAmount.Text)

});

MessageBox.Show("Product Add...!");

}

private void gbxadd\_Enter(object sender, EventArgs e)

{

}

}

}

Yukarıdaki kod ekleme yapabiliriz.

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

public class ProductDal

{

SqlConnection \_connection = new SqlConnection(@"server=(localdb)\ProjectModels;initial catalog=ETrade; integrated security=true");

void ConnectionControl()

{

if (\_connection.State == ConnectionState.Closed)

{

\_connection.Open(); //eğer açık değilse aç.(kapalı ise aç)

}

}

//public DataTable GetAll()

//{

// //iş yapacağız refactorin kod iyileştirmeye gireceğiz.

// SqlConnection connection = new SqlConnection(@"server=(localdb)\ProjectModels;initial catalog=ETrade; integrated security=true");

// //connection.Open(); //bağlantıyı açtık.

// if (connection.State == ConnectionState.Closed)

// {

// connection.Open(); //eğer açık değilse aç.(kapalı ise aç)

// }

// //connection.Open();

// //slq bir dili var ona göre yapıyoruz. bir tablodan veri çekmek

// SqlCommand command = new SqlCommand("Select\* from Products", connection);

// SqlDataReader reader = command.ExecuteReader();//referansı atamasını yapıyoruz.

// DataTable dataTable = new DataTable();

// dataTable.Load(reader);//IDataReadar türünde bir şey ver diyro. buraya direk reader verebiliriz.

// reader.Close();//bağlantıyı kapat

// connection.Close();

// return dataTable;

//}

public List<Product> GetAll2()

{

//iş yapacağız refactorin kod iyileştirmeye gireceğiz.

//connection.Open(); //bağlantıyı açtık.

ConnectionControl();

//connection.Open();

//slq bir dili var ona göre yapıyoruz. bir tablodan veri çekmek

SqlCommand command = new SqlCommand("Select\* from Products", \_connection);

SqlDataReader reader = command.ExecuteReader();//referansı atamasını yapıyoruz.

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

while (reader.Read())//kayıtları tek tek oku ve okuduğun

{

Product product = new Product

{

id=Convert.ToInt32(reader["ID"]),

name = reader["Name"].ToString(),

StockAmount = Convert.ToInt32(reader["StockAmount"]),

unitPrice = Convert.ToDecimal(reader["UnitPrice"])

};

products.Add(product);//her product listeye ekliyorum.

}

reader.Close();//bağlantıyı kapat

\_connection.Close();

return products;

}

public void Add(Product product) // bana bir product ver onu database ekleyim.

{

ConnectionControl();

SqlCommand command = new SqlCommand("Insert into Products values(@name,@unitPrace,@stockAmount)", \_connection);//vaules parametre olarak alacağım

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

command.Parameters.AddWithValue("@unitPrace", product.unitPrice);

command.Parameters.AddWithValue("@StockAmount", product.StockAmount);

command.ExecuteNonQuery();

\_connection.Close();

}

public void Update(Product product) // bana bir product ver onu database ekleyim.

{

ConnectionControl();

SqlCommand command = new SqlCommand("Update Products set Name=@name,unitPrice@unitPrace,StockAmount=@stockAmount where Id=@id)", \_connection);//vaules parametre olarak alacağım

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

command.Parameters.AddWithValue("@unitPrace", product.unitPrice);

command.Parameters.AddWithValue("@StockAmount", product.StockAmount);

command.Parameters.AddWithValue("@id", product.id);

command.ExecuteNonQuery();

\_connection.Close();

}

}

}

Yukarıdaki kodlama ile de update yapabiliriz.

# OBJECT RELATIONSHIP MAPPING

Bu yapıda yine Ado.net alt yapısını kullanarak dahanesnel bir yapıda class/lar ile Database arasında bağlantı kurarak hızlı programlama yapabiliriz. Daha nesnel ve ciddi anlamda daha rahat oluşturmakta hız kazandırmaktadır bizlere…

Yeni bir yapı ile biz şimdi bir Solition üzerine gelerek Add den yeni bir Windows Form yapısı oluşturalım.

Yeni isim EntityFrameWorkDemo olsun. Daha önceki yapıyı seçelim ve kopyalayalım.

Aşağıdaki şekli alsın veya siz de hazırlayabilirsiniz.

Graphical user interface

Description automatically generated

Sonra Product ismindeki cs uzantılı dosyayı kopyalayılım ve kullanalım.

A screenshot of a computer

Description automatically generated with medium confidenceManage Nuget Package isimli bir yapı gördüğünüzde yazılmış kütüphaneleri bulursunuz. Bu paketleri kullanmak için yapaılacak seçenek yukarıdaki gibidir.

Microsoft tarafından yazılmış bir pakettir. Browser kısmında aradığımız paketleri görebiliriz veya arayabiliriz de!

A screenshot of a computer

Description automatically generated

Yukaridaki işaretli olanı seçeriz.

# Generics

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Generics

{

public class Program

{

static void Main(string[] args)

{

}

}

}

/\*

\* Generics bizimiçin bir kurtarıcı özelliğe sahiptir. çok iyi anlamalıyız.

\*

\*

\* \*/

class Product

{

}

interface IProduct:IRepository<Product> //her biri için ayrı ayrı List<

{

//List<Product> GetAll();

//Product Get(int id);

//void Add(Product product);

//void Delete(Product product);

//void Update(Product product);

}

class Customer

{

}

interface ICustomer:IRepository<Customer>

{

//List<Customer> GetAll();

//Customer Get(int id);

//void Add(Customer customer);

//void Delete(Customer customer);

//void Update(Customer customer);

//interface sözleşmesi yaptık. customer olarak değiştirdik. temel olarka tüm nesnelerde kullanacağımz çoklu selecek get tek nesne getirme operasyonudur.

//ben bütün interface bunu yazmak zorunda kaldı. yapı çok benzer daha iyi iyi bir hale gelemez miyim? madem ki değişken customer ?

}

interface IRepository<T>

{

//generic bir yapıyı oluşturmak için interface, abstract class nesne bazlı bir yapıyı oluşturmak için hepsine sonuna <> içerisine bir şey yazmanınz yeterlidir.

//repository yapacağım ve çalışacağım tipi ver diyeceğim.

List<T> GetAll();

T Get(int id);

void Add(T entity);

void Delete(T entity);

void Update(T entity);

//T genel bir yapı ve bir de entity oluşturduk. repostoriy yaptım ve T veriyor olacağım.

//siz bir I

}

//repostory bir tasarım ismidir ve veritabanı operasyonarınd kulnr

class ProductDal : IProduct

{

public void Add(Product entity)

{

throw new NotImplementedException();

}

public void Delete(Product entity)

{

throw new NotImplementedException();

}

public Product Get(int id)

{

throw new NotImplementedException();

}

public List<Product> GetAll()

{

throw new NotImplementedException();

}

public void Update(Product entity)

{

throw new NotImplementedException();

}

}

class CustomerDal : ICustomer

{

public void Add(Customer entity)

{

throw new NotImplementedException();

}

public void Delete(Customer entity)

{

throw new NotImplementedException();

}

public Customer Get(int id)

{

throw new NotImplementedException();

}

public List<Customer> GetAll()

{

throw new NotImplementedException();

}

public void Update(Customer entity)

{

throw new NotImplementedException();

}

void Custom()

{

}

}

//generic bir repostory kullandık. Abstract classta olabilirdi.

Bir güzel kullanım genericler methodlar içinde kullanabiliriz. Yani class içinde değil de method için kullanabiliriz. Bir generic method yazdık sonuna t attık ve projelerimzde biz generic methodları yoğun kullanırız.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Generics

{

public class Program

{

static void Main(string[] args)

{

Utilities utilities = new Utilities();

List<string> result = utilities.BuildList<string>("Ankara", "İzmir", "Adana"); //methoda generic bir yapı kazandırdık.

foreach (var item in result)

{

Console.WriteLine(item);

}

List<Customer> result2 = utilities.BuildList<Customer>(new Customer { FirstName="ÖMER"}, new Customer { FirstName="Yasemin"});

Console.WriteLine();

}

}

public class Utilities

{

public List<T> BuildList<T>(params T[] items)

{

return new List<T>(items);

}//ne kadar olacağı belli değilse params yazılır. {

}

}

/\*

\* Generics bizimiçin bir kurtarıcı özelliğe sahiptir. çok iyi anlamalıyız.

\*

\*

\* \*/

class Product

{

}

interface IProduct:IRepository<Product> //her biri için ayrı ayrı List<

{

//List<Product> GetAll();

//Product Get(int id);

//void Add(Product product);

//void Delete(Product product);

//void Update(Product product);

}

class Customer

{

public string FirstName { get; set; }

}

interface ICustomer:IRepository<Customer>

{

//List<Customer> GetAll();

//Customer Get(int id);

//void Add(Customer customer);

//void Delete(Customer customer);

//void Update(Customer customer);

//interface sözleşmesi yaptık. customer olarak değiştirdik. temel olarka tüm nesnelerde kullanacağımz çoklu selecek get tek nesne getirme operasyonudur.

//ben bütün interface bunu yazmak zorunda kaldı. yapı çok benzer daha iyi iyi bir hale gelemez miyim? madem ki değişken customer ?

}

interface IRepository<T> where T:class,new() //T bir classtır şeklinde referans tip yazılabilir demektir

{

//generic bir yapıyı oluşturmak için interface, abstract class nesne bazlı bir yapıyı oluşturmak için hepsine sonuna <> içerisine bir şey yazmanınz yeterlidir.

//repository yapacağım ve çalışacağım tipi ver diyeceğim.

List<T> GetAll();

T Get(int id);

void Add(T entity);

void Delete(T entity);

void Update(T entity);

//T genel bir yapı ve bir de entity oluşturduk. repostoriy yaptım ve T veriyor olacağım.

//siz bir I

}

//repostory bir tasarım ismidir ve veritabanı operasyonarınd kulnr

class ProductDal : IProduct

{

public void Add(Product entity)

{

throw new NotImplementedException();

}

public void Delete(Product entity)

{

throw new NotImplementedException();

}

public Product Get(int id)

{

throw new NotImplementedException();

}

public List<Product> GetAll()

{

throw new NotImplementedException();

}

public void Update(Product entity)

{

throw new NotImplementedException();

}

}

class CustomerDal : ICustomer

{

public void Add(Customer entity)

{

throw new NotImplementedException();

}

public void Delete(Customer entity)

{

throw new NotImplementedException();

}

public Customer Get(int id)

{

throw new NotImplementedException();

}

public List<Customer> GetAll()

{

throw new NotImplementedException();

}

public void Update(Customer entity)

{

throw new NotImplementedException();

}

void Custom()

{

}

}

//generic bir repostory kullandık. Abstract classta olabilirdi.

//Generic kısıtları:

//Generics kısıtları ile T ile kısıtlarla başarabiliyoruz. Generic IReposotry uyguladığımizda onları generic constrain nedir?

// Repository hangi tipi verirsek onla ilgili bir constrain yapmaktadır. bir interface mizdaha olsun.

interface IStudent : IRepository<string> //string yazmasını engellemek istiyoruz. belirli tipleri yazmasını istiyorum. bir sınıfı yazabilsin istiyorum. bunun için IrEPOSİTORY GİDİP Where koruz. {

{

}

class Student : IStudent

{

public void Add(string entity)

{

throw new NotImplementedException();

}

public void Delete(string entity)

{

throw new NotImplementedException();

}

public string Get(int id)

{

throw new NotImplementedException();

}

public List<string> GetAll()

{

throw new NotImplementedException();

}

public void Update(string entity)

{

throw new NotImplementedException();

}

}

# ATTRIBUTE ile Çalışmak(Özellikler ile Çalışmak)

Öznitelikler: bir nesneye,c# yapılarına uygulanabilecek ayrıca yapılardır. Bu yapılar vasıtasıyla anlamlar katıyoruz.