**METHODS**

using System;

using System.Linq;

namespace Methods

{

    class Program

    {

        static void Main(string[] args)

        {

            Ekle();

            Ekle();

            Ekle();

            // var kendine gelen verinin türünü alır.

            var result = Topla(3, 8);

            Console.WriteLine(result);

            Console.WriteLine(Topla(27, 8));

            Console.WriteLine(Topla(4));

            /\*Eğer bir değişken fonksiyona referansla(ref keyword) gönderilirse bellekteki değeri değişir ve değişkene fonksiyon içerisindeki değeri atanır \*/

            /\* OUT KEYWORD kullanılımı ref keyword ile aynıdır. FAKAT number1'a daha önce değer atamaya gerek yoktur ama metot içerisinde atama yapmak zorunludur \*/

            Console.WriteLine("\nREF AND OUT KEYWORDS");

            int number1 = 20;

            int number2 = 100;

            Console.WriteLine("Number 1'in ilk değeri:" + number1);

            Console.WriteLine(Topla2(ref number1, number2));

            Console.WriteLine("Number 1 referansla fonksiyona gönderildiğinde:" + number1);

            Console.WriteLine("\nMETHOD OVERLOADING");

            Console.WriteLine(Carp(2, 4));

            Console.WriteLine(Carp(2, 4, 5));

            Console.WriteLine("\nPARAMS KEYWORD");

            Console.WriteLine(Topla3(4, 5, 6, 5));

        }

        /\* Method en basit tanımıyla budur. Kendini tekrarlamayı önler. Daha rahat çalışma alanı sunar.

         return komutunun kullanıldığı metotlarda void kullanılamaz. void sadece içerisine yazılanı yapar.

         Geriye veri döndürmez.\*/

        static void Ekle()

        {

            Console.WriteLine("Eklendi!!");

        }

        /\* Tüm parametrelere default değer atanabilir. Default değer atamaya sondan başlanır. Kullanıcı değer vermezse default değer kullanılır. \*/

        static int Topla(int number1, int number2=30)

        {

            return number1 + number2;

        }

        static int Topla2(ref int number1, int number2 = 30)

        {

            number1 = 100;

            return number1 + number2;

        }

        /\* METHOD OVERLOADING metotlara aynı ismi verip, metotlarda daha fazla değişken kullanmaya denir.\*/

        static int Carp(int number1, int number2)

        {

            return number1 \* number2;

        }

        static int Carp(int number1, int number2, int number3)

        {

            return number1 \* number2 \*number3;

        }

        /\* PARAMS KEYWORD ile sınırsız parametre alınabilir. En sonda olmalıdır. Başına başka parametreler eklenebilir\*/

        static int Topla3(params int[] numbers)

        {

            return numbers.Sum();

        }

    }

}

**STRINGS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace StringCalismasi

{

    class Program

    {

        static void Main(string[] args)

        {

            // Stringler char listeleridir. Toplanabilirler.

            string city1 = "Ankara";

            Console.WriteLine(city1[0]+ "\n");

            string city2 = "İstanbul";

            string result = city1 + city2;

            Console.WriteLine(result);

            // Değişkene string formatıyla yerleştirilebilirler.

            string result2 = String.Format("\n{0} ve {1} Türkiye'nin şehirleridir.", city1, city2);

            Console.WriteLine(result2);

            // Writeline zaten string formata sahip olduğu için yazılmasına gerek yoktur.

            Console.WriteLine("\n{0} ve {1} Türkiye'nin şehirleridir.\n", city1, city2);

            // foreach ile tek tek harfleri yazdırılabilir.

            foreach(var harf in city1)

            {

                Console.WriteLine(harf+ "\n");

            }

            string sentence = "ilk cümle";

            // Klon oluşturulurken tamamen kopyalama işlemi yapılır. Klon ve asıl farklı yerlerde saklanır. Ayrı şeylerdir.

            var result3 = sentence.Length;

            var result4 = sentence.Clone();

            sentence = "ikinci cümle";

            Console.WriteLine(result4+"\n");

            bool result5 = sentence.EndsWith("e");

            bool result6 = sentence.StartsWith("a");

            // Bulamazsa -1 döner. Bulduğu ilk şeyin indeksini verir.

            var result7 = sentence.IndexOf("cümle");

            Console.WriteLine(result7 + "\n");

            // Aramaya sondan başlar.

            var result8= sentence.LastIndexOf(" "+"\n");

            Console.WriteLine(result8);

            // Stringe ekleme yapmaya yarar.

            var result9 = sentence.Insert(0, "bu ");

            Console.WriteLine(result9+"\n");

            Console.WriteLine(sentence + "\n");

            // Stringi bölerek almaya yarar.

            var result10 = sentence.Substring(5);

            Console.WriteLine(result10 + "\n");

            var result11 = sentence.Substring(5,3);

            Console.WriteLine(result11 + "\n");

            //ToLower ve ToUpper

            //sentence.Replace(" ", "-"); şeklinde karakterler değiştirilebilir.

            //sentence.Remove(2); 2. indexten itibaren siler.

            //sentence.Remove(2,4); 2. indexten itibaren 4 karakter siler.

        }

    }

}

**CLASSES**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Classlar

{

    class Program

    {

        static void Main(string[] args)

        {

            //Gruplandırmaya Yarar. Kolaylık Sağlar.

            CustomerManager customerManager = new CustomerManager();

            customerManager.Add();

            customerManager.Update();

            // Projeye sağ tıklayıp, add-class ile class dosyası ekleyerek classları oraya taşıyabiliriz.

            // ProductManager class'ı burada yer almamasına rağmen çalışıyor.

            ProductManager productManager = new ProductManager();

            productManager.Add();

            productManager.Update();

            // Bilgiler farklı şekillerde set edilebilir.

            Customer customer = new Customer();

            customer.Id = 1;

            customer.City = "Kastamonu";

            customer.LastName = "Keskin";

            customer.FirstName = "Furkan";

            // ctrl+space'e basınca neler eklenebilir gösteriyor.

            Customer customer2 = new Customer {Id=2,FirstName="Kagawa",LastName="Çarık",City="Sakarya"};

            Console.WriteLine(customer.City);

        }

        class CustomerManager

        {

            public void Add()

            {

                Console.WriteLine("Customer Added!");

            }

            public void Update()

            {

                Console.WriteLine("Customer Updated!");

            }

        }

        class Customer

        {

            //prop yazıp, iki kere tab tuşuna basarsan otomatik olarak iskeleti getirir. Buna property denir.

            public int Id { get; set; }

            public string FirstName { get; set; }

            public string LastName { get; set; }

            public string City { get; set; }

        }

    }

}

**INTERFACES**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Interfaceler

{

    class Program

    {

        static void Main(string[] args)

        {

            // Farklı kullanımlar barındırır.

            PersonManager personManager = new PersonManager();

            personManager.Add(new Customer { Id = 1, FirstName = "Furkan", LastName = "Keskin", Address = "Bla Bla Street" });

            Student student = new Student { Id = 324, FirstName = "Öğrenci", LastName = "Soy adı", Department = "YBS" };

            personManager.Add(student);

            // Classlara tek tek işlem yapabiliriz.

            CustomerManager customerManager = new CustomerManager();

            customerManager.Add(new SqlServerCustomerDal());

            //Tüm classlara aynı işlemi yapmamıız sağlar.

            ICustomerDal[] customerDals = new ICustomerDal[2]

            {

                new SqlServerCustomerDal(),

                new OracleServerCustomerDal(),

            };

            foreach(var customerDal in customerDals)

            {

                customerDal.Add();

            }

        }

        interface IPerson

        {

            int Id { get; set; }

            string FirstName { get; set; }

            string LastName { get; set; }

        }

        class Customer : IPerson

        {

          public int Id { get; set; }

          public string FirstName { get; set; }

          public string LastName { get; set; }

          public string Address { get; set; }

        }

        class Student : IPerson

        {

          public int Id { get; set; }

          public string FirstName { get; set; }

          public string LastName { get; set; }

          public string Department { get; set; }

        }

        class PersonManager

        {

            //IPerson olarak kullanırak IPersonlarda ortak olanlarla işlem yapabiliriz. Mesela Customer customer şeklinde tanımlansaydı adres bilgisi girebilirdik.

            public void Add(IPerson person)

            {

                Console.WriteLine(person.FirstName);

            }

        }

    }

}

**IcustomerDal.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Interfaceler

{

    interface ICustomerDal

    {

        void Add();

        void Update();

        void Delete();

    }

    class SqlServerCustomerDal: ICustomerDal

    {

        public void Add()

        {

            Console.WriteLine("Sql Added");

        }

        public void Delete()

        {

            Console.WriteLine("Sql Deleted");

        }

        public void Update()

        {

            Console.WriteLine("Sql Updated");

        }

    }

    class OracleServerCustomerDal : ICustomerDal

    {

        public void Add()

        {

            Console.WriteLine("Oracle Added");

        }

        public void Delete()

        {

            Console.WriteLine("Oracle Deleted");

        }

        public void Update()

        {

            Console.WriteLine("Oracle Updated");

        }

    }

    class CustomerManager

    {

        public void Add(ICustomerDal customerDal)

        {

            customerDal.Add();

        }

    }

}

**INTERFACES EXERCİSE**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Interfaceler2

{

    class Program

    {

        static void Main(string[] args)

        {

            // Doğru interface planlaması bu şekildedir.

            IWorker[] workers = new IWorker[3]

            {

                new Manager(),

                new Worker(),

                new Robot(),

            };

            foreach(var worker in workers)

            {

                worker.Work();

            }

            IEat[] eats = new IEat[2]

            {

                new Manager(),

                new Worker(),

            };

            foreach(var human in eats)

            {

                human.Eat();

            }

        }

    }

    // İçlerine birden fazla işlem de eklenebilir.

    interface IWorker

    {

        void Work();

    }

interface IEat

    {

        void Eat();

    }

    interface IGetSalary

    {

        void GetSalary();

    }

    // Bir classın birden fazla interface'i olabilir.

    class Manager : IWorker, IEat, IGetSalary

    {

        public void Eat()

        {

            throw new NotImplementedException();

        }

        public void GetSalary()

        {

            throw new NotImplementedException();

        }

        public void Work()

        {

            throw new NotImplementedException();

        }

    }

    class Worker : IWorker, IEat, IGetSalary

    {

        public void Eat()

        {

            throw new NotImplementedException();

        }

        public void GetSalary()

        {

            throw new NotImplementedException();

        }

        public void Work()

        {

            throw new NotImplementedException();

        }

    }

class Robot : IWorker

    {

        public void Work()

        {

            throw new NotImplementedException();

        }

    }

}

INHERITANCE

//Inheritance'lar interface gibi kullanılır ama her class'ın tek bir atası olabilir. class olarak tanımlanırlar.

Virtual Methods

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace VirtualMethodlar

{

    class Program

    {

        static void Main(string[] args)

        {

            Sql sql = new Sql();

            sql.Add();

        }

        class Database

        {

            // virtual olarak tanımlarsak bu fonksiyonu inherit eden diğer classlarda override yaparak ekleme veya tamamen değiştirme yapabiliriz.

            public virtual void Add()

            {

                Console.WriteLine("ADDED");

            }

        }

        class Sql : Database

        {

 //override ederek inherit ettiğimiz bir fonksiyonu değiştirdik.

            public override void Add()

            {

                Console.Write("SQL ");

                //base olanı kullanmayı baştan yazsak da olur.

                base.Add();

            }

        }

    }

}

**ABSTRACT CLASSES**

using System;

namespace ABSTRACTCLASSLAR

{

    class Program

    {

        static void Main(string[] args)

        {

            //Farklı şekillerde tanımlanabilir.

            Database[] database = new Database[1]

            {

                new SqlServer(),

            };

            database[0].Add();

            database[0].Delete();

            Database database2 = new OracleServer();

            database2.Add();

            database2.Delete();

        }

        //İnterface ve inheritancelerin karışımıdır. İnterfacete olduğu gibi kendi kendine newlenemez.

        abstract class Database

        {

            public void Add()

            {

                Console.WriteLine("ADDED");

            }

            public abstract void Delete ();

        }

        class SqlServer : Database

        {

            public override void Delete()

            {

                Console.WriteLine("DELETED BY SQL");

            }

        }

        class OracleServer : Database

        {

            public override void Delete()

            {

                Console.WriteLine("DELETED BY ORACLE");

            }

        }

    }

}

**Access Modifiers**

//private sadece bulunduğu blokta çalışır.

//protected private'ın özelliklerini taşır. tek farkı inheritance durumunda çalışmasıdır.

//internal class'ın defaultudur. Bağlı bulunduğu projede referans gerekmeden kullanılabilir.

//public referans olarak eklendikten ve using DiğerProjeninAdı; satırını yazdıktan sonra başka projelerde de kullanılabilir.

**CONSTRUCTORS**

using System;

namespace Constructorlar

{

    class Program

    {

        static void Main(string[] args)

        {

            Product product = new Product { Id = 1, Name = "Phone" };

            Product product3 = new Product();

            // Pratik tanım yapmamızı sağlar.

            Product product2 = new Product(2,"Laptop");

            Console.WriteLine(product2.Name);

            Console.WriteLine(product2.Id);

        }

        class Product

        {

            public int Id { get; set; }

            public string Name { get; set; }

            // ctor yazıp 2 kere tab tuşuna basarsan iskeleti verir.

            public Product()

            {

                Console.WriteLine("Normal");

            }

            private int \_id;

            private string \_name;

            // Parantez içerisinde giriş yapılırsa alt satırdaki işlemleri de gerçekleştirir.

            public Product(int id, string name)

            {

                // Başka işlemler yapacaksak bu şekilde kullanabiliriz.

                \_id = id;

                \_name = name;

                // get ve set terimlerinin içerisine yerleştirebiliriz. Bu da bize pratiklik sağlayabilir.

                Name = name;

                Id = id;

                Console.WriteLine("Pratik");

            }

        }

    }

}

**Exercise 2**

using System;

namespace Constructorlar2

{

    class Program

    {

        static void Main(string[] args)

        {

            // Hangi veritabanını kullanacağımızı unutmamak için kullanılabilir. Boş kaldığında uyarı verir.

            CustomerManager customerManager = new CustomerManager(new Sql());

            customerManager.Add();

        }

        interface ILogger

        {

            void Log();

        }

        class Sql : ILogger

        {

            public void Log()

            {

                Console.WriteLine("Sql Logged");

            }

        }

        class Oracle : ILogger

        {

            public void Log()

            {

                Console.WriteLine("Oracle Logged");

            }

        }

        class CustomerManager

        {

            private ILogger \_logger;

            public CustomerManager(ILogger logger)

            {

                \_logger = logger;

            }

            public void Add()

            {

                \_logger.Log();

                Console.WriteLine("Added");

            }

        }

    }

}

**Exercise 3**

using System;

namespace Constructorlar3

{

    class Program

    {

        static void Main(string[] args)

        {

            // İsmi girmezsen hata alırsın.

            Person person = new Person("Furkan");

            person.Kayit();

        }

        class BaseClass

        {

            private string \_name;

            public BaseClass(string name)

            {

                \_name = name;

            }

            public void Ad()

            {

                Console.WriteLine("Adınız {0}", \_name);

            }

        }

        // İnherit eden classı oluştururken base classın dışarıdan aldığı değerleri onun için de almalısınız.

        class Person : BaseClass

        {

            public Person(string name) : base(name)

            {

            }

            public void Kayit()

            {

                Console.Write("Sizin ");

                Ad();

            }

        }

    }

}

**Exercise 4**

using System;

namespace Constructorlar4

{

    class Program

    {

        static void Main(string[] args)

        {

            // Statik değişkenler newlenmeden çağırılır.

            Utilities.Validation();

            // Statik fonksiyonu direk çağırabilirken, statik olmayan için classı newlemeniz gerekir.

            Manager.DoSomething();

            Manager manager = new Manager();

            manager.DoSomething2();

        }

    }

    // Static bir classın altındaki tüm değişkenler statik olmak zorundadır.

    // Statik değişkenler üzerinde gerçekleşen işlemler her yerde geçerlidir.

    // Örneğin statik bir değişkenin değerini değiştirirseniz her yerde değişir.

    static class Utilities

    {

        public static void Validation()

        {

            Console.WriteLine("Validation is Done.");

        }

    }

    // Statik olmayan bir classta statik fonksiyonlar oluşturulabilir.

    class Manager

    {

        public static void DoSomething()

        {

            Console.WriteLine("Done");

        }

        public void DoSomething2()

        {

            Console.WriteLine("Done2");

        }

    }

}

**Collections**

using System;

//ArrayListi kullanmak için eklemeliyiz.

using System.Collections;

//Typesafe collection kullanmak için eklemeliyiz.

//Genelde Typesafe collectionlar kullanılır.

using System.Collections.Generic;

namespace Collectionlar

{

    class Program

    {

        static void Main(string[] args)

        {

            // Collectionlar arraylerde yaşanan boyut sıkıntılarına pratiklik getirerek çözer.

            ArrayList cities = new ArrayList();

            cities.Add("İstanbul");

            cities.Add(2);

            cities.Add("Ankara");

            cities.Add("Kastamonu");

            cities.Add(5);

            cities.Add(1905);

            // Citiesin İstanbul içerip, içermediğini kontrol etmek.

            Console.WriteLine(cities.Contains("İstanbul"));

            foreach (var eleman in cities)

            {

                Console.WriteLine(eleman);

            }

            // Typesafe collectionlar belirli bir class veya değişken türüne özel olarak tanımlanır. Farklı olanları kabul etmez.

            // Farklı şekillerde ekleme yapılabilir.

            List<Customer> customers = new List<Customer>

            {

                new Customer {Id=1, FirstName="Furkan"},

            };

            customers.Add(new Customer { Id = 2, FirstName = "Kezzkin" });

            var customer11 = new Customer { Id = 3, FirstName = "Aydi" };

            customers.Add(customer11);

            //Toplu eleman ekleme yöntemi. Paranteze dikkat edilmeli.

            customers.AddRange(new Customer[2]{

                new Customer { Id=4,FirstName="Ayda"},

                new Customer { Id=5, FirstName="Melo"},

            });

            foreach (var customer in customers)

            {

                Console.WriteLine(customer.FirstName);

            }

            // Eleman sayısını verir.

            var count = customers.Count;

            Console.WriteLine("Eleman sayısı: {0}",count);

            // Tüm elemanları siler.

            //customers.Clear();

            count = customers.Count;

            Console.WriteLine("Clear komutundan sonraki sayi:{0} ",count);

            // Indexof komutu elemanın kaçıncı indexte olduğunu verir.

            var index = customers.IndexOf(customer11);

            Console.WriteLine("Index : {0}", index);

            // İstediğin sıraya değer eklemeye yarar. Diğerlerini kaydırır.

            customers.Insert(4, customer11);

            // Sondan arayarak indexi bulur. İndex numarası değişmez.

            // Fakat birden fazla customer11 ekleseydik ilk bulduğunun indexini verecekti.(ekledik)

            var lastindex = customers.LastIndexOf(customer11);

            Console.WriteLine("Index2 : {0}", lastindex);

            // Bulduğu ilk değeri siler. Bulamazsa silmez.

            customers.Remove(customer11);

            // İsmi Ayda olanların hepsini siler.

            customers.RemoveAll(c => c.FirstName == "Ayda");

        }

        class Customer

        {

            public int Id { get; set; }

            public string FirstName { get; set; }

        }

    }

}

**Dictionaries**

using System;

using System.Collections.Generic;

using System.Collections;

namespace Dictionaries

{

    class Program

    {

        static void Main(string[] args)

        {

            // Tanımlanması aşağıdaki gibidir.

            // Collection metotları kullanılabilir.

            Dictionary<string, string> sözlük = new Dictionary<string,string>();

            sözlük.Add("kitap", "book");

            sözlük.Add("tablo", "chart");

            //Anahtar kelimenin karşılığını yazdırma.

            Console.WriteLine(sözlük["tablo"]);

            // chart tablo anahtar kelimesinin karşılığıdır. Anahtar kelime olmadığı için hata verir.

            // Console.WriteLine(sözlük["chart"]);

            foreach(var item in sözlük)

            {

                // Tüm dictionary'i verir.

                Console.WriteLine(item);

                // Sözlükteki anahtar kelimeleri verir.

                Console.WriteLine("Anahtar Kelime: {0}",item.Key);

                // Sözlükteki anahtar kelimelerin karşılığını verir.

                Console.WriteLine("Anahtar Değeri: {0}",item.Value);

            }

            // Anahtar kelimeleri ve değerlerini içerip, içermediğini görebiliriz.

            Console.WriteLine(sözlük.ContainsKey("glass"));

            Console.WriteLine(sözlük.ContainsValue("book"));

        }

    }

}

**EXCEPTION HANDLING**

using System;

namespace ExceptionHandling

{

    class Program

    {

        static void Main(string[] args)

        {

            // try yazıp tabtablayınca iskelet verilir.

            // hata oluştuğunda gelen mesajı kontrol etmeye yarar.

            try

            {

                string[] students = new string[3] { "Furkan", "Keskin", "Burda" };

                students[3] = "Ali";

            }

            //exception değişkenini kullanmayacaksak catch(IndexOutOfRangeException) şeklinde yazılabilir.

            catch (IndexOutOfRangeException exception)

            {

                Console.WriteLine("Dizi boyutunu aştınız");

                Console.WriteLine(exception.Message);

            }

            catch(Exception exception)

            {

                // Developera gelen hata mesajını yazar. Programla ilgili bilgi verebileceği için tehlikeli olabilir.

                Console.WriteLine(exception.Message);

                // Geniş çaplı hatalarda ayrıntılı bilgi verir.

                Console.WriteLine(exception.InnerException);

            }

            /\* Tüm hatalar için genel bir kod

            catch (Exception)

            {

                Console.WriteLine("Hata oluştu");

            }

            \*/

        }

    }

}

**KENDİ HATA SINIFINI OLUŞTURMA, ACTION VE FUNC**

using System;

using System.Collections.Generic;

namespace ExceptionHandling2

{

    class Program

    {

        // Hatamızı class olarak tanımlıyoruz.

        // Exception class'ını inherit ediyoruz.

        // Hata sınıfımızı başka bir class dosyasında da oluşturabiliriz.

        public class RecordNotFoundException : Exception

        {

            // Exception classındaki message'a girdi gönderiyoruz.

            // Göndermezsek exception.Message komutu kullanılamaz.

            // Bu girdiyi de hatayı fırlatacakken yazıyoruz.

            public RecordNotFoundException(string message) : base(message)

            {

            }

        }

        static void Main(string[] args)

        {

            try

            {

                Find();

            }

            catch (RecordNotFoundException exception)

            {

                Console.WriteLine(exception.Message);

            }

            // Başka bir fonksiyona fonksiyonu parametre olarak gönderme.

            HandleException(() =>

            {

                Find();

            });

            // Func

            // Fonksiyonun aldığı parametre türleri ve en sonda da return ettiği parametre türlerini <> arasına yazdık.

            // Topla içerisine burda veri gönderirsek hata verir.

            Func<int, int, int> add = Topla;

            Console.WriteLine(add(3, 2));

            // Parametresiz methodlar için

            // Methodu yanında da tanımlayabiliriz.

            Func<int> getRandomNumber = delegate ()

            {

                Random random = new Random();

                return random.Next(1,100);

            };

            Console.WriteLine(getRandomNumber());

            // Diğer bir method tanımlama şekli.

            Func<int> getRandomNumber2 = () => new Random().Next(1, 100);

            Console.WriteLine(getRandomNumber2());

        }

        static int Topla(int sayi1, int sayi2)

        {

            return sayi1+sayi2;

        }

        static int Topla2(int sayi1, int sayi2)

        {

            return sayi1 + sayi2;

        }

        public static void Find()

        {

            List<string> students = new List<string> { "Furkan", "Engin", "Derin" };

            if (!students.Contains("Ahmet"))

            {

                throw new RecordNotFoundException("String Bulunamadı");

            }

            else

            {

                Console.WriteLine("Record Found");

            }

        }

        // Profesyonel çalışmak için fonksiyonu başka bir dosyaya yazıp, hata kodlarını orda düzenleyebiliriz.

        // Action fonksiyon içerisine fonksiyon göndermeye yarar.

        private static void HandleException(Action action)

        {

            try

            {

                // Gönderilen methodu çalıştırır.

                action.Invoke();

            }

            // Birden fazla hata kodu eklenebilir.

            catch (Exception exception)

            {

                Console.WriteLine(exception.Message);

            }

        }

    }

}

**ADO.NET GİRİŞ VE VERİ TABANI OLUŞTURMA**

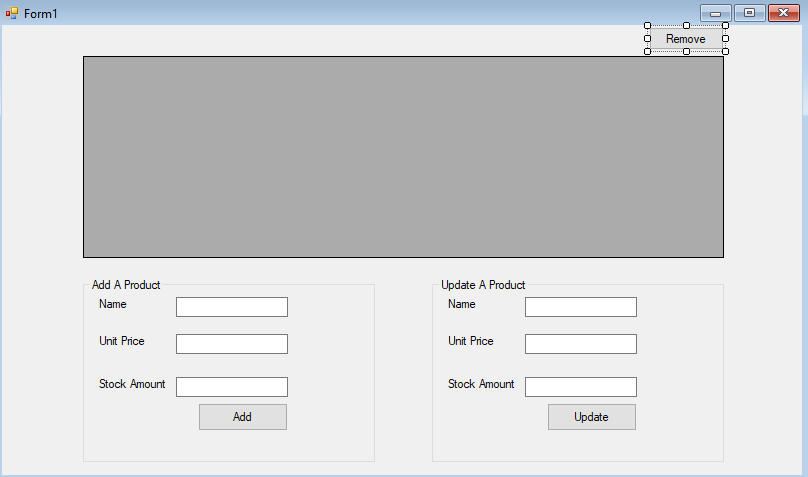
// View --> Sql Server Object Explorer --> Database'e sağ tıklayarak yeni sql database'imizi oluşturabiliriz.

// Databasede tablo oluşturduktan sonra Id'nin otomatik artması için propertiesden Identity Specification-->Is Identity seçeneğini true yapmalıyız.

// Alttaki konsoldan tabloya isim verebiliriz.

// Tabloyla işimiz bittikten sonra sol üstten update butonuna tıklayarak kaydedebiliriz.

**FORM1.CS(DESIGN)**



**FORM1.CS**

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 AdoNetDemo

{

    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }

        // ProductDal classındaki fonksiyonları kullanabilmek için productdal türünde bir nesne oluşturduk.

        ProductDal \_productDal = new ProductDal();

        // Formdaki gride getall fonksiyonunu yansıtacak bir fonksiyon yazdık.

        public void LoadProducts()

        {

          dgwProducts.DataSource = \_productDal.GetAll();

        }

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

        {

            LoadProducts();

        }

        private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

        {

        }

        private void groupBox1\_Enter(object sender, EventArgs e)

        {

        }

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

        {

            Product product = new Product

            {

                Name = tbxName.Text,

                UnitPrice=Convert.ToDecimal(tbxUnitPrice.Text),

                StockAmount=Convert.ToInt32(tbxStockAmount.Text)

            };

            \_productDal.Add(product);

            LoadProducts();

            MessageBox.Show("Product Added!");

        }

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

        {

        }

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

        {

        }

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

        {

        }

        // Gridin herhangi bir yerine tıkladığımızda veriler update grubuna yazılsın diye kod yazdık.

        private void dgwProducts\_CellClick(object sender, DataGridViewCellEventArgs e)

        {

            tbxNameUpdate.Text = dgwProducts.CurrentRow.Cells[1].Value.ToString();

            tbxUnitPriceUpdate.Text = dgwProducts.CurrentRow.Cells[2].Value.ToString();

            tbxStockAmountUpdate.Text = dgwProducts.CurrentRow.Cells[3].Value.ToString();

        }

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

        {

            Product product = new Product {

            // Productımızın idsini formdaki seçili satırdaki id değerine  eşitledik.

            // Değiştirmek istediğimiz productın idsini yeni productımıza  atadık.

            Id = Convert.ToInt32(dgwProducts.CurrentRow.Cells[0].Value),

            // Değişen verileri de textboxtan alarak yeni productımıza atadık.

            Name = tbxNameUpdate.Text,

            UnitPrice=Convert.ToDecimal(tbxUnitPriceUpdate.Text),

            StockAmount=Convert.ToInt32(tbxStockAmountUpdate.Text)

        };

            // Id'ye göre değerleri databasede updateleyecek fonksiyonumuzu  çağırdık.

            \_productDal.Update(product);

            // Gridi tekrar yükledik ve mesajı gösterdik.

            LoadProducts();

            MessageBox.Show("Product Updated!");

        }

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

        {

            // Seçili olan satırın idsini id değişkenine atadık.

            int id = Convert.ToInt32(dgwProducts.CurrentRow.Cells[0].Value);

            // ProductDal classının delete komutunu çağırdık. Parametre olarak seçili verinin idsini gönderdik.

            \_productDal.Delete(id);

            // Gridi tekrar yükledik ve mesajı gösterdik.

            LoadProducts();

            MessageBox.Show("Product Deleted!");

        }

    }

}

**PRODUCTS.CS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AdoNetDemo

{

    public class Product

    {

        // Veri tabanındaki verileri listelerken kullanacağımız class. Fonksiyonda kullanacağız.

        public int Id { get; set; }

        public string Name { get; set; }

        public decimal UnitPrice { get; set; }

        public int StockAmount { get; set; }

    }

}

**PRODUCSTDAL.CS**

using System;

using System.Collections.Generic;

// ConnectionState kullanabilmek için gerekli.

using System.Data;

// Sql Kodlarını kullanmak için ekliyoruz.

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AdoNetDemo

{

    public class ProductDal

    {

        // @ işareti tırnaklar arasındaki işaretleri string olarak algılamaya yarar.

        // initial catalog bağlanılacak database'i seçmeye yarar.

        // integrated securit windows ile bağlanmayı sağlar. eğer id pw ile girilmesi isteniyorsa false olmalı.

        // uid=furkan;password="1234" şeklinde id pw eklenebilir.

        // Bağlantı nesnesini oluşturduk.

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

        private void ConnectionControl()

        {

            // Bağlantı açıksa tekrar açmaya çalışmamak için kontrol ediyoruz.

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

            {

                // Bağlantıyı kurar.

                \_connection.Open();

            }

        }

        // Fonksiyonun döndüreceği türle aynı olması gerek. Burada bir product listesi gönderiyoruz.

        public List<Product> GetAll()

        {

            ConnectionControl();

            // Sql'e komut gönderme.

            // Products tablosundan her şeyi seç getir diyoruz.

            // Yanına hangi bağlantıya gönderileceğini ekliyoruz.

            // Komutu oluşturduk. Çalıştırmadık.

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

            // Okunan verileri bir değişkene atadık.

            SqlDataReader reader = command.ExecuteReader();

            //Liste tanımladık.

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

            while (reader.Read())

            {

                // Readerın okuduğu sütunlardaki veri türlerini dönüştürerek products listesine yazdık.

                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);

            }

            reader.Close();

            \_connection.Close();

            return products;

        }

        // DataTable kullanışsızdır. Fazla rame ihtiyaç duyar.

        public DataTable GetAll2()

        {

            ConnectionControl();

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

            SqlDataReader reader = command.ExecuteReader();

            // DataTable türünde bir değişken oluşturduk.

            DataTable dataTable = new DataTable();

            // Load komutu reader türünde nesneleri alır. Sql'den çektiğimiz verileri içine attık.

            dataTable.Load(reader);

            reader.Close();

            \_connection.Close();

            return dataTable;

        }

        // Ekleyeceğimiz yeni ürün için product türünde bir değişkene ihtiyaç var.

        // (Product product) kısmında bunu istiyor.

        public void Add(Product product)

        {

            ConnectionControl();

            // Values içerisine @ ile sütun isimlerini yazdık.

            // Sql büyük/küçük duyarlı değil.

            SqlCommand command = new SqlCommand("Insert into Products values(@name,@unitPrice,@stockAmount)",\_connection);

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

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

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

            // Komutu execute ediyoruz(çalıştırıyoruz).

            command.ExecuteNonQuery();

            \_connection.Close();

        }

        public void Update(Product product)

        {

            ConnectionControl();

            SqlCommand command = new SqlCommand("Update Products set Name=@name, UnitPrice=@unitPrice, StockAmount=@stockAmount where Id=@id ", \_connection);

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

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

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

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

            command.ExecuteNonQuery();

            \_connection.Close();

        }

        // Silme işleminde id üzerinden gideceğimiz için id parametresi istedik.

        public void Delete(int id)

        {

            ConnectionControl();

            SqlCommand command = new SqlCommand("Delete from Products where Id=@id ", \_connection);

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

            command.ExecuteNonQuery();

            \_connection.Close();

        }

    }

}

**WORKING WITH ENTITY FRAMEWORK AND LINQ**

**APP.CONFIG**

<!--Database'e bağlanmak için connectionStrings altındaki bağlantıları kullanacak-->

    <connectionStrings>

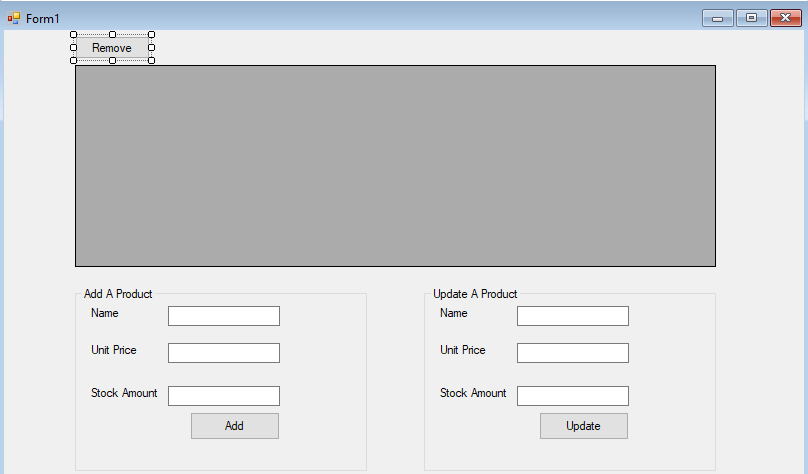
        <add name="ETradeContext"

             connectionString="server=(localdb)\mssqllocaldb;initial catalog=ETrade;integrated security=true"

             providerName="System.Data.SqlClient"/>

    </connectionStrings>

**FORM1.CS[DESIGN]**



**FORM1.CS**

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 EntityFrameworkDemo

{

    public partial class Form1 : Form

    {

        ProductDal \_productDal = new ProductDal();

        private void LoadProducts()

        {

            dgwProducts.DataSource = \_productDal.GetAll();

        }

        public Form1()

        {

            InitializeComponent();

        }

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

        {

            LoadProducts();

        }

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

        {

            \_productDal.Add(new Product

            {

                Name = tbxName.Text,

                UnitPrice = Convert.ToDecimal(tbxUnitPrice.Text),

                StockAmount = Convert.ToInt32(tbxStockAmount.Text)

            });

            LoadProducts();

            MessageBox.Show("Product Added!");

        }

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

        {

            Product product = new Product

            {

                Id = Convert.ToInt32(dgwProducts.CurrentRow.Cells[0].Value),

                Name = tbxNameUpdate.Text,

                UnitPrice = Convert.ToDecimal(tbxUnitPriceUpdate.Text),

                StockAmount = Convert.ToInt32(tbxStockAmountUpdate.Text)

            };

            \_productDal.Update(product);

            LoadProducts();

            MessageBox.Show("Product Updated!");

        }

        // Gridin herhangi bir yerine tıkladığımızda veriler update grubuna yazılsın diye kod yazdık.

        private void dgwProducts\_CellClick(object sender, DataGridViewCellEventArgs e)

        {

            tbxNameUpdate.Text = dgwProducts.CurrentRow.Cells[1].Value.ToString();

            tbxUnitPriceUpdate.Text = dgwProducts.CurrentRow.Cells[2].Value.ToString();

            tbxStockAmountUpdate.Text = dgwProducts.CurrentRow.Cells[3].Value.ToString();

        }

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

        {

            // Entity Frameworkde silme işlemi primary keylere göre yapıldığından id girsek yeterli.

            \_productDal.Delete(new Product

            {

                Id = Convert.ToInt32(dgwProducts.CurrentRow.Cells[0].Value)

            });

            LoadProducts();

            MessageBox.Show("Product Deleted!");

        }

        private void tbxSearch\_TextChanged(object sender, EventArgs e)

        {

            // Textbox içerisindeki text her değiştiğinde burası çalışır.

            SearchProducts(tbxSearch.Text);

        }

        private void SearchProducts(string key)

        {

            // p alt satırda bizim verdiğimiz bir harf.

            // Dataya tüm veriyi alıp onu filtreleyerek yolluyoruz.

            // Filtrede p için p'nin ismi gönderilen harfi içeriyorsa bunu listeye dönüştür yazıyor.

            // Sorguyu bize gelen listeye göre yaptık. Veritabanına göre yapılan türü de var. Yaptık.

            // Büyük küçük duyarlıdır. Name.ToLower.Contains(key.ToLower) şeklinde yazılırsa bu duyar kalkar.

            // var result= \_productDal.GetAll().Where(p=>p.Name.Contains(key)).ToList();

            // Veri tabanına göre yazdığımız fonksiyon.

            // Büyük küçük duyarsızdır ve daha iyi performans verir.

            var result = \_productDal.GetByName(tbxSearch.Text);

            dgwProducts.DataSource = result;

        }

    }

}

**PRODUCT.CS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace EntityFrameworkDemo

{

    public class Product

    {

        // Veri tabanındaki verileri listelerken kullanacağımız class. Fonksiyonda kullanacağız.

        public int Id { get; set; }

        public string Name { get; set; }

        public decimal UnitPrice { get; set; }

        public int StockAmount { get; set; }

    }

}

**–**

**ETRADECONTEXT.CS**

using System;

using System.Collections.Generic;

// Entity Framework için gereken kütüphane.

using System.Data.Entity;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace EntityFrameworkDemo

{

    public class ETradeContext:DbContext

    {

        // Bağlanacağı adresi ve diğer bilgileri app.configte configuration altından alır.

        // Tablolarda productsı arar ve bulursa bağlanır.

        // Products tablosundaki verileri product classındaki nesneler olarak çekmeye yarayan kod.

        public DbSet<Product> Products { get; set; }

    }

}

**PRODUCTDAL.CS**

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace EntityFrameworkDemo

{

    public class ProductDal

    {

        public List<Product> GetAll()

        {

            // Using ile tanımlarsak garbage collector beklenmeden nesne işi bittiğinde dispode edilir.

            using (ETradeContext context = new ETradeContext())

            {

                // Contextin bağlı olduğu tablodaki verileri listeye dönüştürerek gride göndermeye yarar.

                return context.Products.ToList();

            };

        }

        public List<Product> GetByName(string key)

        {

            using (ETradeContext context = new ETradeContext())

            {

                // Filtrede p için p'nin ismi gönderilen harfi içeriyorsa bunu listeye dönüştür yazıyor.

                // Sorgulama direk veritabanında yapılıyor.

                // Daha iyi performans verir

                return context.Products.Where(p => p.Name.Contains(key)).ToList();

            };

        }

            public void Add(Product product)

        {

            using (ETradeContext context = new ETradeContext())

            {

                // Parametre olarak aldığımız product nesnesini Products tablosuna ekliyoruz.

                //context.Products.Add(product);

                //veya

                // Productı entry değişkenine atar.

                var entity = context.Entry(product);

                // Bu kod satırı gönderdiğimiz productı tabloya yazar.

                entity.State = EntityState.Added;

                // Değişiklikleri kaydediyoruz.

                context.SaveChanges();

            };

        }

       public void Update(Product product)

        {

            using (ETradeContext context = new ETradeContext())

            {

                // Productı entry değişkenine atar.

                var entity = context.Entry(product);

                // Gönderdiğimiz idye sahip productı bulur ve gönderdiğimiz productı üstüne yazar.

                entity.State = EntityState.Modified;

                context.SaveChanges();

            };

        }

        public void Delete(Product product)

        {

            using (ETradeContext context = new ETradeContext())

            {

                // Productı entry değişkenine atar.

                var entity = context.Entry(product);

                // Gönderdiğimiz idye sahip productı bulur ve siler.

                entity.State = EntityState.Deleted;

                context.SaveChanges();

            };

        }

        // Tek sonuç alacağımız zaman filtreleme

        // Liste olmadığı için Liste türünde oluşturmadık.

        // Butonlar eklenmemiştir ve çalıştırılmamıştır.

        public Product GetById(int id)

        {

            using (ETradeContext context = new ETradeContext())

            {

                // FirstOrDefault ilk sonucu verir bulamazsa null verir.

                // SingleOrDefault versiyonu birden fazlaysa hata veren versiyonudur.

                var result = context.Products.FirstOrDefault(p => p.Id == id);

                return result;

            }

        }

    }

}

**GENERICS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Genericler

{

    class Program

    {

        // Interface, abstract class, class olabilir. Repositoryler sayesinde içerilerine istediğimiz nesneyi gönderebiliriz.

        // Where ile sınıflandırma yapılabilir. class referans tipler için, struct değer tipler için kullanılır.

        // new() ise yenilenebilir nesneler demektir, her zaman en sonda olmalıdır. Filtreye kendi oluşturduğumuz türü de verebiliriz.

        interface IRepository<T> where T:class,IEntity,new()

        {

            List<T> GetAll();

            T Get(int id);

            void Add(T entity);

            void Delete(T entity);

            void Update(T entity);

        }

        // İşlem yapmayacağımız için nesnelerin içi boş.

        // Filtreleme yaparken repositorynin sadece IEntity türüne sahip nesneleri alması için classları ientityden implemente ettik.

        interface IEntity

        {

        }

        class Product:IEntity

        {

        }

        class Customer:IEntity

        {

            public string Name { get; set; }

        }

        // Repositoryimizi başka bir interface ile inherit edersek nesneye özel method ekleyebiliriz.

        // Direkt ProductDal sınıfına inherit etseydik özelleştirme yapamayacaktık.

        interface IProductDal:IRepository<Product>

        {

            void Custom();

        }

        class ProductDal : IProductDal

        {

            public void Add(Product entity)

            {

                throw new NotImplementedException();

            }

            // Özel methodumuzu ekledik.

            public void Custom()

            {

                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();

            }

        }

        // Repositoryi inherit ettiğimiz için özelleştirme yapamayacağız.

        // Onun yerine başka bir interface oluşturup, onu inherit etseydik o interface içerisinde özelleştirme yapabilirdik.

        class CustomerDal : IRepository<Customer>

        {

            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();

            }

        }

        // Generic Methodlar

        static void Main(string[] args)

        {

            // Utilities türünde bir nesne oluşturduk.

            Utilities utilities = new Utilities();

            // Utilities classı içerisindeki BuildList Repository fonksiyonu ile istediğimiz türde bir liste oluşturduk.

            List<Customer> result = utilities.BuildList<Customer>(new Customer{Name="Furkan" },new Customer{Name="Szchwenger"});

            // Liste içerisindeki Name değişkenlerini yazdırdık.

            foreach (var customer in result)

            {

                Console.WriteLine(customer.Name);

            }

            // Konsolu açık tuttuk.

            Console.ReadLine();

        }

        class Utilities

        {

            // Gönderilen nesne türünde liste oluşturup, geri döndürme.

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

            {

                return new List<T>(items);

            }

        }

    }

}

**ATTRIBUTES**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Attiributelar

{

    class Program

    {

        // Çoğu kod sadece iskelet. İçleri boş.

        static void Main(string[] args)

        {

            Customer customer = new Customer { Id = 1, LastName = "Keskin", Age = 20 };

            CustomerDal customerDal = new CustomerDal();

            // Obsolete attribute'u çalışıyor. Altı çizili.

            customerDal.Add(customer);

            Console.ReadLine();

        }

        class Customer

        {

            public int Id { get; set; }

            [RequiredProperty]

            public string FirstName { get; set; }

            [RequiredProperty]

            public string LastName { get; set; }

            [RequiredProperty]

            public int Age { get; set; }

        }

        // Customers tablosuyla ilişkilendirmek için kullanabiliriz.

        [ToTable("Customers")]

        class CustomerDal

        {

            // Hazır attributeları da kullanabiliriz.

            [Obsolete("Burada Kullanıcıya Uyarı Verilebilir.")]

            public void Add(Customer customer)

            {

                Console.WriteLine("{0}, {1}, {2}, {3} added!", customer.Id, customer.FirstName, customer.LastName, customer.Age);

            }

        }

        // Kendimiz bir attribute oluştururken attribute classını ona inherit ederiz.

        // AttributeUsage attribute'un kullanım şeklini belirlemek içindir.

        // AttributeTarget kullanılabileceği alanları belirler.

        // AttributeTargets.Property | AttributeTargets.Class gibi kullanımlar mümkündür.

        // AllowMultiple ise attribute'un birden fazla kullanıp, kullanılamayacağını belirler.

        [AttributeUsage(AttributeTargets.Property,AllowMultiple=false)]

        class RequiredPropertyAttribute : Attribute

        {

        }

        /\* AllowMultiple burda olumludur. Çünkü ToTable attribute'u class'ı

        birden fazla tabloyla ilişkilendirmesi veya farklı isimlerle

        tabloyu arayıp bulduğunda işlemi gerçekleştirmesi için kullanabiliriz.\*/

        [AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]

        class ToTableAttribute : Attribute

        {

            private string \_tableName;

            public ToTableAttribute(string tableName)

            {

                \_tableName = tableName;

            }

        }

    }

}

**REFLECTIONS**

using System;

using System.Collections.Generic;

using System.Linq;

//Reflectionlar için gerekebilecek library.

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace ReflectionİleCalismak

{

    class Program

    {

        static void Main(string[] args)

        {

            /\*

            // Instance oluştururken parametre istediği için girmeliyiz.

            DortIslem dortIslem = new DortIslem(2,3);

            // Değeri çağırırken aldı.

            Console.WriteLine(dortIslem.Topla(5,7));

            // Değeri instance oluştururken aldı.

            Console.WriteLine(dortIslem.Topla2());

            \*/

            // Reflectionlar program çalışırken çalışır. Az performans kaybı yaşatır.

            // DortIslem'in tipini değişkene atadık.

            // Normalde typeof'un içerisine temsili bir şey koyarız ki tipini alacağımız şeyi oraya atayalım.

            var tip = typeof(DortIslem);

            // Program çalışırken instance işlemi yaptık.

            // Classımız kullanıcıdan değer istediği için tip'in yanına o değerleri eklemezsek hata verir.

            DortIslem dortIslem =(DortIslem) Activator.CreateInstance(tip,3,4);

            Console.WriteLine(dortIslem.Topla(2,3));

            Console.WriteLine(dortIslem.Topla2());

            // Instance oluşturduk.

            var instance = Activator.CreateInstance(tip, 9,5);

 // Method bilgisine ulaştık. Çalıştıktan sonra instance bağı kayboldu.

            MethodInfo methodInfo = instance.GetType().GetMethod("Topla2");

            // Method içerisine instance'ı gönderdik.

            Console.WriteLine(methodInfo.Invoke(instance,null));

            var metodlar = tip.GetMethods();

            foreach(var info in metodlar)

            {

                // Method isimlerini aldık.

                Console.WriteLine("Metod Adı: {0}", info.Name);

                // Methodların istedikleri parametreleri aldık.

                foreach(var parameterInfo in info.GetParameters())

                {

                    Console.WriteLine("Parametre: {0}", parameterInfo.Name);

                }

                // Attributeleri aldık.

                foreach(var attribute in info.GetCustomAttributes())

                {

                    Console.WriteLine("Attribute: {0}", attribute.GetType().Name);

                }

            }

            Console.ReadLine();

        }

        class DortIslem

        {

            int \_sayi1, \_sayi2;

            public DortIslem(int sayi1,int sayi2)

            {

                \_sayi1 = sayi1;

                \_sayi2 = sayi2;

            }

            // Fonksiyonu çağırırken değer alan fonksiyon.

            public int Topla(int sayi1, int sayi2)

            {

                return sayi1 + sayi2;

            }

            // Instance oluştururken alınan değerle işlem yapan fonksiyon.

            public int Topla2()

            {

                return \_sayi1 + \_sayi2;

            }

        }

    }

}

**DELEGATES**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Delegateler

{

    // Delegateler elçi gibi. Mesaj iletiyorlar.

    public delegate void MyDelegate();

    public delegate void MyDelegate2(string text);

    class Program

    {

        static void Main(string[] args)

        {

            CustomerManager customerManager = new CustomerManager();

            // Delegate'in içerisine fonksiyonu yolluyoruz.

            MyDelegate myDelegate = customerManager.SendMessage;

            // Toplama, çıkarma gibi işlemleri kullanabiliriz.

            myDelegate += customerManager.ShowAlert;

            myDelegate();

            // Fonksiyon mesajı dışarıdan alıyorsa tek delegate ile ayrı ayrı mesaj yazılmıyor.

            // İkisi de aynı mesajı gösteriyor.

            MyDelegate2 myDelegate2 = customerManager.SendMessage2;

            myDelegate2 += customerManager.ShowAlert2;

            // Yukarıdaki olayı topla, çarp gibi fonksiyonlara yapsakdık sonuç en son yapılan işlemin sonucu olurdu.

            // public delegate int MyDelegate3(int number1, int number2); ile denenebilir.

            myDelegate2("Delegate 2");

            Console.ReadLine();

        }

    }

    public class CustomerManager

    {

        public void SendMessage()

        {

            Console.WriteLine("Hello");

        }

        public void SendMessage2(string text)

        {

            Console.WriteLine(text);

        }

        public void ShowAlert()

        {

            Console.WriteLine("Be careful");

        }

        public void ShowAlert2(string text)

        {

            Console.WriteLine(text);

        }

    }

}

**EVENTS**

**PRODUCT.CS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Eventler

{

    public delegate void StockControl();

    public class Product

    {

        private int \_stock;

        // Class'a dışarıdan stock aldık ve class içerisindeki \_stock'a attık.

        public Product(int stock)

        {

            \_stock = stock;

        }

        // Event tanımladık.

        public event StockControl StockControlEvent;

        public string ProductName { get; set; }

        public int Stock {

            // Set üzerinde ekleme yapacağımız için bu şekilde yazdık.

            get

            {

                // Normal get'den farkı yok. Stocku okumaya yarar.

                return \_stock;

            }

            set

            {

                // Normal set'den farkı yok. Stocku set eder.

                // \_stocku programın gönderdiği değere eşitliyor.

                \_stock= value ;

                // Eğer stock <=15 ise ve stockcontrolevente abone olunmuşsa eventi fırlatıyor.

                if(value<=15 && StockControlEvent != null)

                {

                    StockControlEvent();

                }

            } }

        public void Sell(int amount)

        {

            Stock -= amount;

            Console.WriteLine("{1} Stock Amount: {0}", Stock, ProductName);

        }

    }

}

**PROGRAM.CS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Eventler

{

    class Program

    {

        static void Main(string[] args)

        {

            // Gönderdiğimiz stock değeri Product.Stock'a set ediliyor.

            Product harddisk = new Product(50);

            harddisk.ProductName = "Hard Disk";

            Product gsm = new Product(50);

            gsm.ProductName = "GSM";

            // Product'ı evente abone ediyoruz. +='den sonra 2 kere tab'a basarsak isim ve kodları oluşur.

            gsm.StockControlEvent += Gsm\_StockControlEvent;

            for (int i = 0; i < 10; i++)

            {

                harddisk.Sell(10);

                gsm.Sell(10);

                Console.ReadLine();

            }

            Console.ReadLine();

        }

        // İçerisine eventte gerçekleşecek şeyleri yazıyoruz.

        private static void Gsm\_StockControlEvent()

        {

            Console.WriteLine("Gsm stocks is low!");

        }

    }

}

**PROJE 1**

**APP.CONFIG**

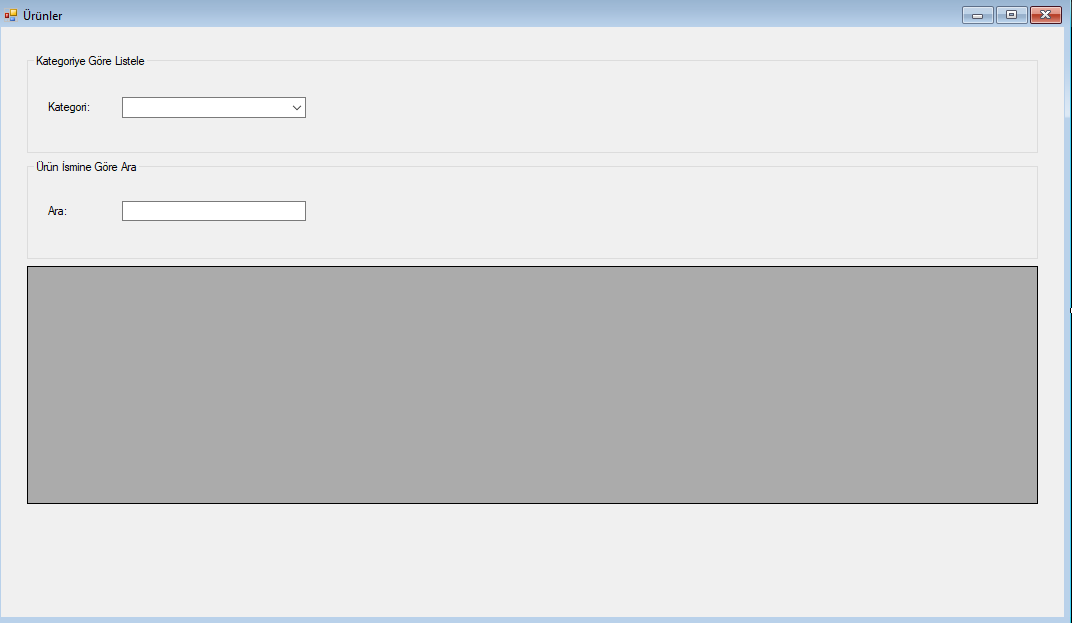
<connectionStrings>

        <add name="NorthwindContext" connectionString="Data source=(localdb)\mssqllocaldb;initial catalog=Northwind ;integrated security=true"

             providerName='System.Data.SqlClient'/>

    </connectionStrings>

**FORM1.CS[DESIGN]**



**FORM1.CS**

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 RecapProject1

{

    // Kategori ve isim araması birbirinden bağımsız aramalar olarak yazıldı.

    // İhtiyaca göre kodlarla ilişkilendirilebilir.

    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }

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

        {

            LoadProducts();

            LoadCategories();

        }

        public void LoadProducts()

        {

            using (NorthwindContext context = new NorthwindContext())

            {

                dgwProduct.DataSource = context.Products.ToList();

            }

        }

        public void LoadCategories()

        {

            using (NorthwindContext context = new NorthwindContext())

            {

                cbxCategory.DataSource=context.Categories.ToList();

                // Gösterilen isim ile id'yi ilişkilendirdik.

                cbxCategory.DisplayMember = "CategoryName";

                cbxCategory.ValueMember = "CategoryId";

            }

        }

        private void cbxCategory\_SelectedIndexChanged(object sender, EventArgs e)

        {

            /\* Eğer sadece belirli bir ürün türünü göstereceksek index üzerinden gidilebilir.

            Ama bu tüm ürünleri bir arada göstermeyi imkansız kılar.

            int value = (cbxCategory.SelectedIndex) + 1;

            ListProductsByCategory(value);

            \*/

            // İlk başta tüm categoryleri gösterebilmek için seçili değere göre filtreleme yaptık.

            // Fakat ilk başta seçilmiş değer olmadığı için hata verecektir. Hatayı bypass ettik.

            try

            {

              ListProductsByCategory(Convert.ToInt32(cbxCategory.SelectedValue));

            }

            catch

            {

            }

        }

        public void ListProductsByCategory(int categoryId)

        {

            using (NorthwindContext context = new NorthwindContext())

            {

                //Products listesindeki p'lerin CategoryId'si gönderilen categoryId ile aynıysa ürünleri listele.

                dgwProduct.DataSource = context.Products.Where(p => p.CategoryId == categoryId).ToList();

            }

        }

        private void tbxSearch\_TextChanged(object sender, EventArgs e)

        {

            string key = tbxSearch.Text;

            // Textboxın içi boş ise grid'e ürünleri tekrar yükleyecek.

            if (string.IsNullOrEmpty(key))

            {

                LoadProducts();

            }

            // Textbox boş değilse filtreleme yapacak.

            else

            {

                ListProductsByText(tbxSearch.Text);

            }

        }

        public void ListProductsByText(string text)

        {

            using (NorthwindContext context = new NorthwindContext())

            {

                // Products listesindeki p'lerin Name'i gönderilen texti içeriyorsa listele.

                // Sql key sensitive olmadığı için tolower veya toupper komutlarını kullanmaya gerek yok.

                // Oracle'da veya c# listelerinde lazım olabilir.

                // Bu komutları ekledikten sonra ise aramaya bir şey yazıp silince liste yüklenmiyor.

                // Bunu textchanged kısmında düzelteceğiz.

                dgwProduct.DataSource = context.Products.Where(p => p.ProductName.ToLower().Contains(text.ToLower())).ToList();

            }

        }

    }

}

**PROGRAM.CS**

// Program.cs dosyasına hiç dokunulmadı. Sadece Form1'i açıyor.

**PRODUCT.CS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RecapProject1

{

    class Product

    {

        public int ProductId { get; set; }

        public int CategoryId { get; set; }

        public string ProductName { get; set; }

        public decimal UnitPrice { get; set; }

        public Int16 UnitsInStock { get; set; }

        public string QuantityPerUnit { get; set; }

    }

}

**CATEGORY.CS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RecapProject1

{

    class Category

    {

        public int CategoryId { get; set; }

        public string CategoryName { get; set; }

    }

}

**NORTHWINDCONTEXT.CS**

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RecapProject1

{

    class NorthwindContext:DbContext

    {

        public DbSet<Product> Products { get; set; }

        public DbSet<Category> Categories { get; set; }

    }

}