ReCapProject

Geliştirme projesi ReCapProject 7.GÜN

Senaryo :

* **Entities, DataAccess, Business ve Console katmanlarını oluşturunuz.**
* **Bir araba nesnesi oluşturunuz. "Car"**
* **Özellik olarak : Id, BrandId, ColorId, ModelYear, DailyPrice, Description alanlarını ekleyiniz. (Brand = Marka)**
* **InMemory formatta GetById, GetAll, Add, Update, Delete operasyonlarını yazınız.**

Şekildeki gibi katmanlarımızı oluşturalım. Console katmanı hariç diğer katmanlara Abstract ve Concrete klasörlerimizi oluşturalım.

* DataAccess : Veri erişim,
* Business : İş katmanı,
* ConsoleUI: Konsol arayüzü,
* Entities : Varlık, yardımcı katman. Tüm katmanların kullanabileceği katman.
* Concrete : Somut yani gerçek işi yapan nesneler.
* Abstract : Soyut nesneleri, abstract classları, interfaceleri, base classları, referans tutucuları koyarız.

metin, skorbord, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Entities/Concrete/Car.cs : Car nesnesi oluşturduğumuz katman. Car varlıktır,nesne,somut. Senaryoda belirtilen özellikleri ekleyelim.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Entities/Abstract/IEntity.cs : Kural neydi hiçbir sınıf boş kalmayacak. IEntity implemente eden class bir veritabanı tablosudur. Car nesnesi veritabanında bir tabloya karşılık gelir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Veri erişim katmanına gelelim. Senaryoda operasyonlar belirtmiş. Bunları yapalım.

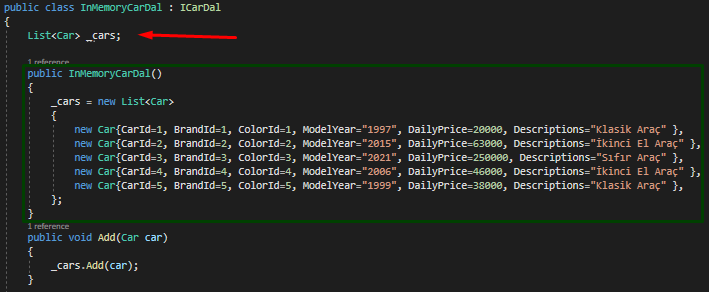
DataAccess/Abstract/ICarDal.cs : Operasyon sınıfım. I:İnterface, Car : Hangi tabloya karşılık geldiğini (Entities’e yani), Dal : Hangi katmana karşılık geldiğini(Data Access Layer) ifade eder. Senaryoda belirtilen operasyonlar bu soyut sınıfa yazılır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

DataAccess/Concrete/InMemory/InMemoryCarDal.cs : ICarDal’ da operasyonlarımızı yazdık. İnterface’i(ICarDal) iş yapan sınıf haline getirmeliyiz. Unutma interface yapıyorduk bir de somutunu yapıyorduk. İnterface’i burada implemente ediyoruz, somutlaştırıyoruz. Concrete somut nesne = InMemoryCarDal (Bellekte arabalar ile ilgili erişim kodları yazılacak yer). InMemoryCarDal bir ICarDal’dır yani onun bir implemantasyonu demek.

Şuan bellekte veri varmış ,biz onu yönetiyormuşuz gibi davranıyoruz. Dolayısıyla proje başladığında veri varmış gibi ortamı simüle edelim. Veri varmış gibi davranacağımız için araba listesi oluşturalım. Şöyle düşün ICarDal’ da arabaların hepsini getir diye operasyon var, bir araba listesi olması lazım ki getirsin.



Proje çalıştırıldığında bizim yerimize bir tane bellekte araba listesi oluşturdu. Sanki bu bize Oracle, Sql server, Postgres, MongoDb veritabanlarından geliyormuş gibi simüle ediyoruz.

**Kırmızı çizgi ;** Cars isimli bir değişken tanımladık. Değişken nasıl tanımlanır : string cars. Bizim bir araba listemiz varmış gibi davranıyoruz string yerine List< > veri tipini kullandık.

Bu nesneyi classın içinde ama metotların dışında tanımlandığı zaman bu tip değişkenlere global değişken deniyor yani o class için global. Genellikle alt çizgiyle gösterilir. (\_cars) Bu da nameing commention adını verdiğimiz bir isimle standartıdır.

Kırmızı çizgiyle gösterdiğimiz ifade bir referans tip tek başına bir anlam ifade etmiyor sadece değişkeni oluşturdu. Projeyi başlatınca bellekte bir tane araba listesi oluştur ifadesini yansıtır.

**Yeşil çizgi içindeki ;** Constructor yapısıdır. Bellekte referans aldığı zaman çalışacak olan bloktur. Void vs. döndürmüyor, direk classın ismiyle olunca constructor oluyor. Yani değişken oluşturduk ya o referans tip tek başına bir anlam ifade etmiyordu, constructor yapısıyla değişkene veri atadık. İçerisinde arabaları barındıran bir liste oluşturduk.

ICarDal classındaki operasyonları yazıyoruz. Ekleme operasyonunu yazdık. Bu arabayı bu List<Car> \_cars veri kaynağımıza ekleriz. Yani elimizdeki \_cars’a Add diyerek parametre olarak Business’tan gönderilen arabayı car veritabanına ekliyorum. O da List olduğu için bu şekilde listeye eklenir. Şimdi diğer operasyonlarımızı yazalım.

Delete operasyonunu bu şekilde yazdığımızda listeden araba silinmez neden ?

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Arayüzden bir tane araba newleyip gönderildiğinde bunun referansı 200 diyelim, sen arabalar listesinden 200 ‘ü sil diyorsun. Ama 200 yok çünkü oluşturduğum adreslerin referans numaraları farklı bu yüzden silmez bu ifade. Burada yapılması gereken (Car car) bunun id’sini kullanarak arabalar listesindeki adreslerden eşleşen id’ yi bulup o referansı yakalamak. Bu sisteme de LINQ denir. LINQ-Language Integrated Query (Dile Gömülü Sorgu)

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

carToDelete = silinecek ürün. Foreach ile arabaları tek tek dolaş. Her dolaştığındaki c o an dolaştığın elemana denk gelir. İf ile gönderilen car’ın CarId’si eşit eşitse o anki car’ın CarId’si kontrolü, doğruysa silinecek eleman o anki bu eleman diye şartlarız. Normalde burada az eleman olduğu için foreach ile elemanları tek tek dolaşırız ama bu işlem linq ile daha kolay bir şekilde yapılabilir. SingleOrDefault tek bir eleman bulmaya yarar.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

GetAll operasyonu veritabanındaki datayı(List<Car>) Business’a vermek lazım. Business arabalar listesini istediğinde, ona liste verilmeli. Tüm listeyi return ifadesi ile veritabanını olduğu gibi tümünü döndürüyoruz.

Nesneyi oluşturduk, veri erişimde için gereken operasyonları yazdık. Gelelim iş katmanımıza.

Business/Abstract/ICarService.cs : İş katmanının soyut interface’i Service.

metin içeren bir resim

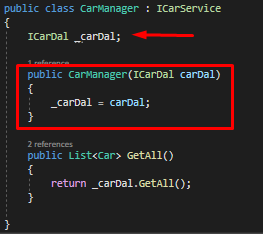
Açıklama otomatik olarak oluşturuldu

Business/Concrete/ICarManager.cs : İş katmanının somut sınıfı Manager.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bunu bu şekilde yazarsak iş kodlarının tamamı bellekle çalışır. Veritabanına geçeceğin zaman mesela EntityFramework kullanacağın zaman GetAll( ) gibi binlerce operasyon var, hepsini değiştirmen gerekir. Kural bir iş sınıfı başka sınıfları newlemez. Bunun yerine soyut nesne ile bağlantı kuracağız.



**Kırmızı ok ;** Değişken, referans tutucu, constructor injekte et.

**Kırmızı çizgi içindeki ;** Bu ne işe yarıyor ? CarManager newlendiğinde constructor diyor ki bana bir tane ICarDal referansı ver. (InMemory, EntityFramework, Depar vs.)

Sonrasında ICarService interface sınıfımızın GetAll() operasyonunu somutlaştırırız. Return ifadesinden önce varsa iş kodları if’ler bulunur. İf böyleyse,şöyleyse (Yetkisi var mı ?) gibi kontroller bulunur. Kuralları da yazdıktan sonra diyelim ki geçti DataAccess’e diyor ki bana arabaları verebilirsin ben kurallardan geçtim.

Console/Program.cs : Listedeki tüm arabaların tanımlarını(açıklamalarını) ekrana yazdırır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Geliştirme projesi ReCapProject 8.GÜN

Senaryo Devam : **Car nesnesine ek olarak;**

* **Brand ve Color nesneleri ekleyiniz(Entity)**

**Brand-->Id,Name**

**Color-->Id,Name**

* **Sql Server tarafında yeni bir veritabanı kurunuz. Cars,Brands,Colors tablolarını oluşturunuz. (Araştırma)**
* **Sisteme Generic IEntityRepository altyapısı yazınız.**
* **Car, Brand ve Color nesneleri için Entity Framework altyapısını yazınız.**
* **GetCarsByBrandId , GetCarsByColorId servislerini yazınız.**
* **Sisteme yeni araba eklendiğinde aşağıdaki kuralları çalıştırınız.**

**Araba ismi minimum 2 karakter olmalıdır**

**Araba günlük fiyatı 0'dan büyük olmalıdır.**

Entities/Concrete/Brand.cs : Entities/Concrete/Color.cs :

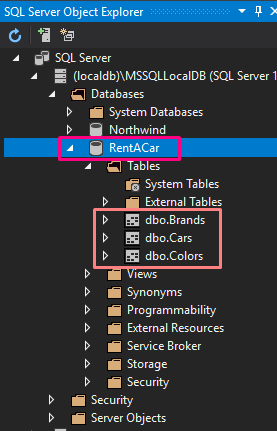
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Nesnelerimizi oluşturduktan sonra sql server tarafına veritabanımızı kuruyoruz.

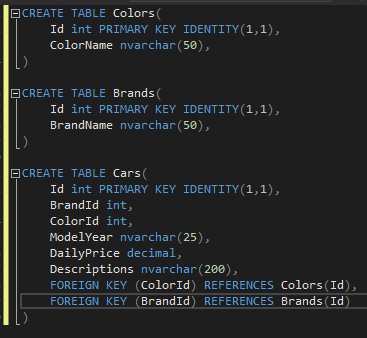
**SQL Server Object Explorer penceresinde ;**



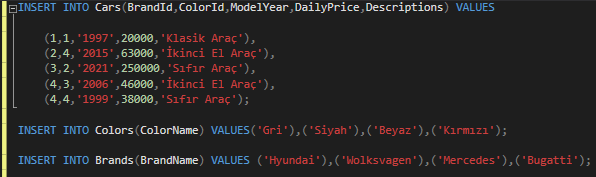
**Koyu Pembe :** Database Adı : RentACar

**Açık Pembe :** RentACar’ da yer alan tablolarımız : Cars, Brands, Colors

**Bu tabloları oluşturmak için RentACar -> Mouse Sağ Click -> New Query**

[](https://user-images.githubusercontent.com/59045890/107125055-9059af80-68b8-11eb-87bd-0a0eb47e71a6.png)

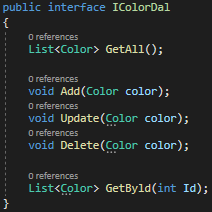
**Tabloların içine veriler eklemek için (Aynı Query sayfasına yazılabilir ya da RentACar -> Mouse Sağ Click -> New Query diyerek yeni sayfada da yazabilirsiniz.**



Veritabanında Car, Color, Brand tablolarını oluşturduk ve içerisine verileri ekledik. Şimdi gelelim DataAccess katmanına.

DataAccess/Abstract/ICarDal.cs : DataAccess/Abstract/IColorDal.cs :

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu 

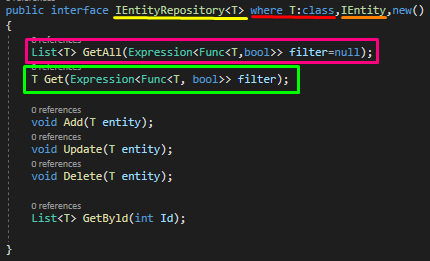
Car gibi Color nesnesini de yazdık. Operasyonlarını yazdık. Id’ye göre filtreleme işlemi yaptık. Aynı car nesnesindeki gibi. Değişen tek şeyin List’in veya operasyonların parantez içlerinin değiştiğini gördük yani car veya color. Bu şekilde bir sürü tablomuz, varlığımız olduğunu düşün her nesne için aynı işlemler yapmak yerine jeneric yapısını kullanabiliriz. Çünkü sadece tip değişiyor. Brand nesnesinin operasyonları da car ve color nesnelerinin operasyonuyla aynı , 3 farklı sınıfta aynı operasyonu, aynı içeriği tekrarlıyoruz.

Hatırla Jenerikler tasarladığımız interface, class, metod ya da parametrelerin(argümanların) belirli bir tip için değil bir şablon yapısına uyan her tip için çalışmasını sağlayan bir yapıdır. Tiplere genelde T denir. T ne istersen o olur, type için. < > içerisine tip yazılır.

Bunun yerine bir tane interface yapsak jenerik tipi olsa ICarDal daki car nesnesi veya IColorDal daki color yerine jenerik T yapsak daha kolay olur. Bu yapının adı da Generics Repository Design Pattern. (Jenerik Repository Tasarım Deseni)

Şöyle düşün operasyonlar ortak, içerik ortak değişen şey tip. jeneriklerde T yapısı kullanıyorduk burada bunu kullanabileceğimiz ortak bir class oluşturuyoruz.

İsmi IEntityRepository. Jenerik yapısı yapalım IEntityRepository<T> Buradaki T bana çalışacağım tipi söyle. Car dersem Car olacak, Color dersem Color olacak, Brand dersem Brand olacak.



Backend kodluyoruz belirli kurallar var karşılaşılan biz de bu IEntityRepository<T> T’yi sınırlandırmak istiyoruz. Herkes T’yi yazamasın T olarak gelmesi gereken veritabanı nesneleridir. Sonuçta biz neyi listeler, ekler, günceller ve sileriz veritabanı nesnelerimizi(entities).

Bunu sınırlandıralım buna GENERİC CONSTRAİNT (Jenerik Kısıt) deniyor.

Yani mesela ICarDal’da ICarDal:IEntityRepository<Car> car yerine int yazsak sistem bunu da kabul eder tip çünkü T yerine hepsi yazılabilir. Ama biz T için sadece nesnelerimizi yazalım diye kural koyuyoruz.

**Sarı çizgi ;** Jenerik yapısı.

**Kırmızı çizgi ;** Where koşulu ile T class olsun diyoruz. Tip olarak int,string filan olamasın.

* **where T : class** ; class : buradaki class referans tip olabilir demek.

Yani ICarDal : IEntityRepository<Car> Car yerine int yazsak kabul etmez bu sefer çünkü referans tip engeller. İnt,string değer tip engeller bunları.

**Turuncu çizgi ;** Where T:class olduğunda ise yine Car classı yerine bir class ismi yazılsa hata vermez o yüzden sadece IEntity classını şartlayacağım ki bana sadece nesnelerimi versin kısıtlasın. Nesne classlarının ortak özelliği hepsinin IEntity olması.

* **where T : class, IEntity**

Bu şu demek T bir referans tip olmalı (class) ve ya IEntity ya da IEntityden implemente olan bir   
şey olabilir.

* **IEntity**; IEntity olabilir veya IEntity implemente eden bir nesne olabilir.

Bu süreçlerden sonra ICarDal : IEntityRepository<Car> Car yerine color, brand, car ve IEntity yazılabilir sadece. Ama ben IEntity de yazmak istemiyorum soyut nesne işimi görmüyor, devre dışı bırakmak istiyorum.

* **where T:class,IEntity,new( ) ;** bu şekilde yazıldığında devre dışı bırakabilirim nasıl ?

new : new'lenebilir olmalı. IEntity interface newlenemez. Color, Brand, Car somut nesne newlenebilir.

Jenerik repository nesnelerinde genellikle liste, add, update, delete ve id’ye göre getirme olur.

**Pembe Kutucuk ;** Mesela car idsi 2 olan ürünleri getir. Bu filtreleme işlemini yapmak için manager sınıfında GetAll operasyonu içerisine return \_carDal.GetAll(p=>p.CarId == 2); yazarız.

GetAll operasyonunda datanın tamamını değil de belli bir kısmını getirmek istediğimizde böyle bir filtre vermemizi sağlayan yapının ismi EXPRESSİON’ dır. Bir şey verebilmemizi sağlayan yapı Expression’dır. Expression’ı verebilmenin syntax’ı budur. Bu da Linqle beraber geliyor. Filter == null filtre vermeyedebilirsin demek. Eğer filtre vermemişse tüm datayı istiyor, filtre vermişse filtreleyip verecek.

Yani kategoriye göre getir, ürünün fiyatına göre getir gibi ayrı ayrı metotlar yazmak gerekmiyor bu yapıyı kullandığımızda.

**Yeşil kutucuk ;** GetAll( ) gibi bir tane T döndüren Get( ) operasyonu yazalım.

Tek bir data getirmek için kullanırız bu genellikle bir şeyin detayına gitmek için mesela bir bankacılık uygulamasında hesaplar var GetAll( )’ la liste olarak geliyor, bir tane hesaba tıklayıp o hesabın detayına gidiyoruz o zamanda Get( ) kullanırız. Burada filtre vermek zorunda olduğumuz için filter == null kullanmayız.

Operasyonları barındıran tek bir class yaptığımıza göre DataAccess katmanımızın classlarını düzeltelim. Proje geliştirmede 3. Maddemiz olan IEntityRepository altyapısını sisteme kurduk.

DataAccess/Abstract/ICarDal.cs : ICarDal sen bir IEntityRepositorysin ve çalışma tipin Cardır. Yani IEntityRepository’i Car için yapılandırdın demektir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

DataAccess/Abstract/IColorDal.cs : IColorDal sen bir IEntityRepositorysin ve çalışma tipin Colordır. Yani IEntityRepository’i Color için yapılandırdın demektir.

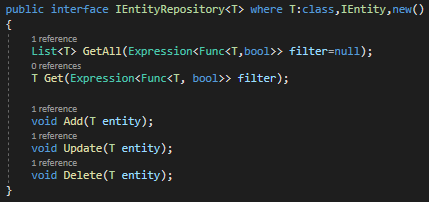


DataAccess/Abstract/IBrandDal.cs : IBrandDal sen bir IEntityRepositorysin ve çalışma tipin Branddır. Yani IEntityRepository’i Brand için yapılandırdın demektir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

DataAccess/Abstract/IEntityRepository.cs :



metin içeren bir resim

Açıklama otomatik olarak oluşturulduBunlar alternatif sistemler. Projeyi geliştirmeden önce InMemory bellekte çalışıyorduk. Şimdi EntityFramework ile çalışacağız. Projeye framework ekledik. İlerleyen zamanlarda farklı bir sisteme Deeper’a geçebiliriz gibi.

**Entity Framework**

Entity Framework Microsoft un bir ürünü. ORM (Object Relational Mapping) dediğimiz bir ürünü. Linq destekli çalışıyor. Amaç şu Orm demek veritabanındaki tabloyu sanki classmış gibi onunla ilişkilendirip bundan böyle bütün operasyonları yani sqlleri bizim linq ile yaptığımız bir ortam. Orm kodlar ile veritabanı nesneleri arasında bir ilişki, bağ kurup veritabanı işlerini yapma süreci.

Entity framework sisteme nasıl dahil edilir ?

Bugüne kadar C# ın kendisini kullandık. C# in .NET içerisinde implementasyonlarını kullandık ama biz ilerledikçe başkalarının yazdığı (bunlara paket diyoruz) kodları da kullanacağız. bu kodların ortak koyulduğu ve yönetildiği bir ortam var, bu ortamın adı NuGet.

.NetCore içerisinde default olarak Entity Framework bir paketle geliyor. Bu paketin adı Entity Framework.

Dataccess'e sağ tıkla manage nuget seçeneğinde browse da .netframeworkcore.sqlserver kur(3.1.0) dataccess properties target framework - 4.7.2

Şu ana kadar yapılan iş Entity Framework kurulumu. Veritabanı var artık.

İlk Aşama

Veritabanıyla kendi nesnelerimizi kısacası; Veritabanındaki cars tablosu ile nesnelerimizdeki cars tablosunu, veritabanındaki color tablosu ile color tablosunu, veritabanındaki brand tablosu ile brand tablosunu ilişkilendirmemiz gerekiyor. Bu işlemi yapabilmek için Context denilen yapıyı kurmalıyız.

* **Context :** Veritabanı ile kendi classlarımızı ilişkilendirdiğimiz class. Kısacası Db tabloları ile proje classlarını bağlamak.

DataAccess/Concrete/EntityFramework/RentACarContext : Entity Framework yapısı ile veritabanımızı bağlamaya, ilişkilendirmeye çalışıyoruz. RentACarContext adında class oluşturduk Entity Frameworkta. RentACarContext dememiz ismine, bunun Context olduğu anlamına gelmez. Entityframeworkla birlikte DbContext adında base bir class gelir. DbContext aslında bizim contextimizin ta kendisi.

Entity Framework ben nereye bağlanacağım diye sana soruyor veritabanını belirtmelisin.

OnConfiguring bu metod senin projen hangi veritabanıyla ilişkiliyi belirtilecek yer.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Kırmızı çizgi ;** Sql Server kullanacağız. Sql Server’a nasıl bağlanacağını belirtmelisin.

**Mor çizgi ;** optionsBuilder.UseSqlServer(@"Server = 175.45.2.12"); normal projede böyle IP görürüz.

Ama biz development ortamındayız onun yerine viewda sql server object explorerdaki yeri yazman gerekir. (localdb)\ mssqllocaldb

**Sarı çizgi ;** Bu serverda hangi veritabanı ? Onu da belirt. Database=RentACar

**Beyaz çizgi ;** Trusted\_Connection=true : kullanıcı girişimiz yok ya, bu veritabanına direk doğrudan giriş gibi. Kullanıcı adlı şifre gerektirmeden.

**Yeşil kutu ;** Context hangi veritabanına bağlanacağını buldu yukarıda. Burada ise benim hangi classım hangi tabloya karşılık geliyor ?

DataAccess/Concrete/EntityFramework/EfCarDal : EfCarDal sen bir ICarDal’sın. Soyutlama tekniği somutlaştırıyoruz. İnterfacete ki operasyonlarımızı implemente ediyoruz.

Bir classı newlediğimizde Garbage Collector(çöp kutusu) belli bir zamanda düzenli olarak gelir ve bellekten onu atar.

Using içerisine yazılan nesneler, using bitince garbage collectore gelip, beni bellekten at diyor.

Burada using içerisine yazılan RentACarContextle işi bitince bellekten atılacak.

Using CSharp’a özgü güçlü bir yapı. Buradaki Using IDisposable pattern implementation of c# ( CSharp’ın Idisposable pattern denilen bir implementasyonu demek )

Entity Framework’ta Using içerisine yazılan işlemlerle ekleme, silme, güncelleme operasyonları yazarız.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

EfCarDal operasyonlarımızı yazmaya devam ediyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

GetAll( ) operasyonu diyelim ki filtre göndermemişse veri kaynağındaki yani veritabanında ilgili tablodaki tüm datayı listelesin. Ama filtre vermişse filtreyi uygula ona göre datayı listele.

**Kırmızı çizgi** Cars’a yerleş. Veritabanındaki bütün tabloyu listeye çevir ve onu bana ver. **;** Eğer filtre null ise ilk kısım çalışır, **Sarı çizgi ;** değilse diğer kısım çalışır.

Operasyonlarımız aynı olduğu için EfCarDal’da gerçekleşen tüm işlemler EfColorDal ve EfBrandDal’da da gerçekleştirilir. Sadece tip değişiyor, Car yerine Color ve Brand yazalım.

DataAccess katmanımın yeni hali

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Senaryonun 4. Maddesi olan Car, Color, Brand nesneleri için Entity Framework altyapısını kurduk.

Senaryoda 5. Maddede istenilen GetCarsByBrandId, GetCarsByColorId servicelerini yazalım.

Business katmanına geldiğinde hiçbir şey değişmez. Çünkü standartlara uyduk. Daha önce InMemory ile çalışıyorduk. Şimdi Entity Frameworkla çalışıyoruz. İş katmanımda herhangi bir şeyi değiştimeme gerek kalmadı. Businessımız,iş kurallarımız değişmedi sadece veriye erişim yöntemimiz değişti.

Business/Abstract/ICarService : Businessta ICarService’te GetAll( ) tümünü getir diyoruz. Senaryoda istenilen serviceleri de yazalım. Araba listesine göre filtreleme istediği için bu servisleri buraya yazıyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Business/Abstract/IColorService : Business/Abstract/IBrandService :

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Business/Abstract/CarManager : Businessta interfacede listeleri yazdık Concrete Managerda işlet onu. Senaryonun son maddesi araba ekleme işlemi kurallarını gerçekleştiriyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

CarManager sen bir ICarService sin. İmplemente ederek kalan operasyonlarımızı yazıyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Business/Abstract/BrandManager : Business/Abstract/ColorManager :

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Console/Program.cs : Console’da GetAll, GetCarsByColorId çağırıp çalıştırdığında belirtilen filtreye göre gelir.

SOLID prensibinin O harfidir. Open Closed Principle--Yaptığın yazılıma yeni bir özellik ekliyorsan mevcuttaki hiçbir koduna dokunamazsın.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Geliştirme projesi ReCapProject 9.GÜN

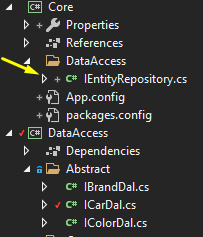
Senaryo Devam :

* **CarRental Projenizde Core katmanı oluşturunuz.**
* **IEntity, IDto, IEntityRepository, EfEntityRepositoryBase dosyalarınızı 9. gün dersindeki gibi oluşturup ekleyiniz.**
* **Car, Brand, Color sınıflarınız için tüm CRUD operasyonlarını hazır hale getiriniz.**
* **Console'da Tüm CRUD operasyonlarınızı Car, Brand, Color nesneleriniz için test ediniz. GetAll, GetById, Insert, Update, Delete.**
* **Arabaları şu bilgiler olacak şekilde listeleyiniz. CarName, BrandName, ColorName, DailyPrice. (İpucu : IDto oluşturup 3 tabloya join yazınız).**

Yazılım geliştirme projelerinde framework katmanı oluştururuz. CORE adında katman oluşturarak ortak kodları bu katmana yazarız bundan sonra bütün projelerde bu katmanı kullanabilir.

Core katmanına evrensel kodlarımızı yani tüm projelerde kullanılacak kodlarımızı koyabiliriz. O yüzden bu Core katmanında hangi katmanla ilgileneceksek o katmanla ilgili klasör oluşturuyoruz. Ben altyapımda veri erişimlerde kullanmak üzere bir klasör oluşturuyorum. Yani DataAccess katmanına hizmet edecek kodları bunun içine koyuyorum. Bizim için IEntityRepository tam olarak bu işe yarıyor.

**Core/DataAccess :** Veri erişimi ilgilendiren evrensel kodlar buraya konulur. IEntityRepository classını buraya taşıdık.



**Core/DataAccess/IEntityRepository.cs :** Burada IEntity’in altını çizdi. Burada Entities katmanını Core katmanına referans verilebilir olarak düşünebilirsiniz. Ama Core katmanına Entities katmanını referans verirsek bu CarRental projesindeki Entities’e bağımlı olur. Bu doğru değil. Peki ne yapacağız ? Entities katmanı IEntity sınıfını kullanıyor. IEntity classını Entities katmanı içerisinde bıraktık ama IEntity RentACar projesine özel bir şey değil ki. IEntity şu anlama geliyordu bir classa IEntity dediğimizde o class bir veritabanı tablosu görevi görüyordu. Yani bu IEntity IEntityRepository gibi Core katmanına özgü bir şey. Çünkü ben o IEntity’i her projede kullanabilirim.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Core katmanının mantığı; istediğim katmanı burada ayrı ayrı klasörleyip, implemente edeyim.

**Core/Entities/IEntity.cs :** Entities klasörü oluşturarak IEntity classını buraya taşıdık. Namespace’ni değiştir. Şuan yapılan işlem iki nesneyi Core’a taşıdık. Core benim evrensel katmanım. Yani sadece RentACar da değil, bütün .NET projelerinde Core katmanını kullanabilirim anlamına gelir. Core katmanı diğer katmanları referans almaz!!!!

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Pembe çizgi ;** Artık anlamsız onu silelim.

**Sarı çizgi ;** IEntity artık Core katmanından geliyor onu ekleyelim.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

IEntityRepository sınıfı düzeldi, önceden DataAccess katmanındaydı ve ICarDal, IColorDal, IBrandDal sınıfları ondan kalıtım ediyordu. Ve o sınıflarda şuan altı çizili. Şimdi bu sınıflar IEntityRepository kullanacaksa nerede bu sınıf ? Core katmanında o zaman bu sınıfları taşıyan DataAccess katmanı Core katmanına bağımlı. O zaman referans veririz.

**DataAccess katmanına sağ tıkla >> Project Reference >> Core katmanını ekle**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu işleme Code Refactoring (Kodun İyileştirilmesi) denir.

Aynı şekilde IEntity classını da Core katmanına taşıdık. Önceden Entities katmanındaki nesneler ondan kalıtım alıyordu. Bunları da iyileştirelim. Using core kütüphanemizi de ekleyelim.

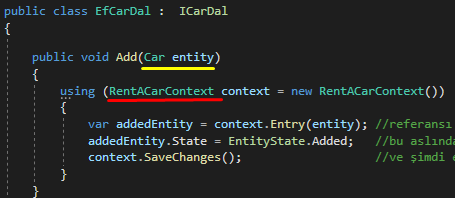
**Entities katmanına sağ tıkla >> Project Reference >> Core katmanını ekle**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Core katmanını oluşturduk, her şeyi düzelttik. Gelelim EfCarDal, EfColorDal, EfBrandDal classlarını jenerik yapıya çekmeye.

Data Access katmanımızın Abstract klasöründe IEntityRepository altyapısını kurmuştuk. IColorDal, IBrandDal, ICarDal sınıflarımızda aynı operasyonlar mevcuttu değişen tek şey tipti ve biz bunu bir jenerik sınıf(IEntityRepository) oluşturarak diğer sınıfların bu jenerik sınıftan kalıtım almasını sağlamıştık. Data Access katmanımızın Concrete klasörüne baktığımızda interfacelerin somutlaştırılmış hali, implemente ettiğimiz sınıflar EfCarDal, EfColorDal, EfBrandDal içerik aynı, operasyonlar aynı o zaman burada da jenerik yapısını kullanabiliriz.



Burada değişeceğini bildiğimiz kısımlar

1-)Çalıştığını bildiğimiz entity. Car,Color,Brand vs.

2-)Contextimiz değişken. Mesela CarRental veritabanına bağlandığımızda CarRentalContext olur orası.

Bunları base bir sınıfa temel bir sınıfa taşısak ve onu jenerik yapsak, bana çalışacağım tipi ver(entity) ve çalışacağım context'i ver desek jenerik olarak. Biz onları bütün varlıklar için kullansak.

**Core katmanına gel >> DataAccess sağ tıkla >> Add >> New Folder >> EntityFramework adını ver**

Bahsettiğim sınıflar DataAccess katmanının EntityFramework klasöründeydi , Core katmanında EntityFramework adında klasör oluşturdum. EntityFramework’ü kullanarak evrensel bir kod yazacağız. O yüzden klasörlüyoruz.

**Core/DataAccess/EntityFramework/EfEntityRepositoryBase.cs :** EntityFramework’ü kullanarak bir Repository Base’i oluştur demek. Bu sınıf EfCarDal,EfColorDal,EfBrandDal sınıflarında operasyonların ortak olması sadece tip ve contextin değişmesi sebebiyle oluşacak olan Jenerik sınıfım. Jenerik sınıflarda hatırla <T> yazıyorduk. Farklı isimlerde istediğin kadar T yazabilirsin. Sadece tip ve context değiştiği için TEntity, TContext diyoruz. Bu şu demek bana bir tane Entity tablo ver ve Context tipi ver ben ona göre çalışacağım demek bu yapı.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

EfCarDal da operasyonlarda hatırla Entity Framework kodu yazdık. Burada da o operasyonları yazacağız, entity framework kodu kullanmamız gerek, o zaman Core katmanına da DataAccess katmanına daha önceden eklediğimiz gibi Entity Framework sistemini eklemeliyiz.

**Solution sağ tıkla >> Manage Nuget >> Browse >> .MicrosoftEntityFrameworkCore.SqlServer >> Core işaretle**

metin, ekran görüntüsü, siyah, ekran içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kurulumdan sonra jenerik sınıfımızı yazmaya devam edelim. Hatırla jenerik kısıtlaması yapmıştık, programcı doğru bir şekilde o yapıyı kullanabilsin diye. Generic Constraint yapalım, çeşitli kurallar koyalım. T’yi sınırlandıralım.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Sarı çizgi ;** Jenerik yapısı.

**Kırmızı çizgi ;** Where koşulu ile TEntity class olsun diyoruz. Tip olarak int, string filan olamasın.

* **where TEntity : class** ; class : buradaki class referans tip olabilir demek.

**Turuncu çizgi ;** Where TEntity : class olduğunda ise yine Color classı yerine bir class ismi yazılsa hata vermez o yüzden sadece IEntity classını şartlayacağım ki bana sadece nesnelerimi versin kısıtlasın. Nesne classlarının ortak özelliği hepsinin IEntity olması.

* **where TEntity : class, IEntity**

Bu şu demek TEntity bir referans tip olmalı (class) ve ya IEntity ya da IEntityden implemente olan bir   
şey olabilir.

* **IEntity**; IEntity olabilir veya IEntity implemente eden bir nesne olabilir.

Bu süreçlerden sonra EfColorDal : EfEntityRepositoryBase<Color, Context> Color yerine brand, car, color ve IEntity yazılabilir sadece. Ama ben IEntity yazmak istemiyorum soyut nesne işimi görmüyor, devre dışı bırakmak istiyorum.

* **where T:class,IEntity,new( ) ;** bu şekilde yazıldığında devre dışı bırakabilirim nasıl ?

new : new'lenebilir olmalı. IEntity interface newlenemez. Brand, Car, Color somut nesne newlenebilir.

**Pembe çizgi ;** Bunun Context olabilmesi için EntityFramework DbContext’i inherit etmesi gerekiyor.

* **where TContext : DbContext** ; buraya her classı yazamazsın. DbContext’ten inherit etmesi lazım. Burada da newlenebilir yapıyı koyuyoruz ki sadece NorthwindContext gibi classlar gelsin.

Şimdi DataAccess katmanında Abstractta IColorDal bir IEntityRepository’di. Concrete klasöründeki EfColorDal ‘ da bir IColorDal’dı. O zaman EfColorDal gibi ortak operasyonlara sahip sınıfları taşıyan jenerik sınıfım EfEntityRepositoryBase bir IEntityRepository’dir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

EfEntityRepositoryBase sınıfını EfCarDal, EfColorDal, EfBrandDal sınıfları için oluşturmuştuk. EfCarDal da yazılan operasyonları al bu classa yapıştır. Car yazan yerlere TEntity, RentACarContext yazan yerlere de TContext yazarak jenerik hale getirelim.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**DataAccess/Concrete/EntityFramework/EfCarDal.cs :**

EfCarDal daki operasyonlarımızı aldıktan sonra bize kızıyor ama artık ben EfEntityRepositoryBase jenerik sınıfımı bu classlar(EfCarDal,EfColorDal,EfBrandDal) için oluşturduğumdan buraya yazmalıyım.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şuan EfCarDal’ da bütün veritabanları operasyonlarını yazmaya hazırız.

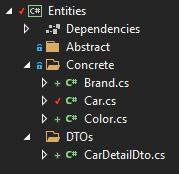
**DTO (DATA TRANSFORMATİON OBJECT)**

Bir e-ticaret sistemini düşünelim ürünün ismi ve kategorisinin ismi yanyana ekranda gözükür. Ürün ismi product tablosunda, kategori ismi kategori tablosundadır ama ekranda nasıl yanyana gelir ? Dto sayesinde

Örneğin ; veri tabanından çekilen veride isim, soyisim ayrı propertylerdedir. Sizin tek property olarak kullanmanız gerekir ya da iki farklı veri tablosundan maplediğiniz iki modelden gelen propertyleri harmanlayarak bir model oluşturmanız gerekebilir. İşte veri tabanından mapplenen modelleri uygulama içerisinde kullanacağınız modelle aktardığınızda bu modeller dto olarak adlandırılır.

CarDetailDto sen bir IDto’sun. Neden IEntity değil de IDto çünkü bir veritabanı tablosu değil, birden fazla tablonun joini olabilir. Context’e gidip eklemesin diye IDto dedik çünkü tablo değil, tabloya karşılık gelmiyor. Belki birkaç tablonun birkaç kolonuna karşılık geliyor olabilir ama tek başına bir tablo değil.

**Entities/DTOs/CarDetailDto.cs :** Entities katmanına DTOs adında bir klasör oluşturalım ve içine   
CarDetailDto isimli class oluşturalım. Bu dto classına senaryoda verilen özellikleri ekleyelim.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Core/Entities/IDto.cs :** IDTO interface classı evrensel kodları içereceğinden Core’un içindeki Entities katmanına ekliyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldumetin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**DataAccess/Abstract/ICarDal.cs :** Arabaların detaylarını getir listele diye metot yazalım.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

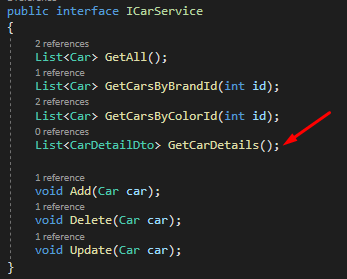
**DataAccess/Concrete/EntityFramework/EfCarDal.cs :** ICarDal da bir metot ekledik burası da onun somut sınıfı metodu implemente etmeliyiz o metodu somutlaştıyoruz. Tabloları join etmeliyiz ki Dto görevi yerine getirsin. Cars tablosu ile Colors tablosunu ve Brands tablosunu birleştiriyoruz. Tablolarda da Id’ler mevcut olduğundan bu özelliği eşitliyoruz(equals).

**Sarı Çizgi ;** CarDetailDto classındaki özellikleri; join ettiğimiz tablolardaki kolonlarına denkleştiriyoruz. Yani CarDetailDto classındaki CarName özelliğini, RentACarContextteki Cars tablosunun Descriptions özelliğine eşitliyoruz. CarDetailDto classındaki ColorName özelliğini, RentACarContextteki Colors tablosunun ColorName özelliğine eşitliyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Business/Abstract/ICarService.cs :** Metodu iş sınıfımızın Service interfacesinede ekleyelim.



**Business/Concrete/CarManager.cs :** İnterface sınıfımızı somutlaştıralım. Yeni metot ekledik interface sınıfımıza burada da implemente etmeliyiz. Return diyerek listenin tamamını döndürüyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Console/Program.cs :** Cars tablosu ile Brands tablosunu ve Cars tablosu ile Colors tablosunu join ederek CarDetailsDto classındaki özelliklere ilgili kolonları eşitledik. Ve burada artık CarName, BrandName, ColorName, DailyPrice yanyana yazabiliyoruz. Dto mantığı bu şekildedir.

Car, Color, Brand nesneleri için tüm CRUD operasyonlarını (GetAll,GetById,Insert,Update,Delete) Business katmanındaki Service iş sınıflarına ekleme işlemi yapalım. Eksik operasyonunuz varsa ekleyin. Program.cs de şimdi hangi Test alanını çağırırsam ekranda o çalışacak. Tüm nesneler için sırasıyla Add, GetById, GetAll operasyonlarını yazdırdık. CarDetailDtoTest alanında ise senaryoda istenilen CarName, BrandName, ColorName, DailyPrice özelliklerini yazdırdık. Ekranda bu özelliklerin yazmasını istediğim için CarDetailDtoTest alanını çağırıyorum, her alanı aynı anda çalıştırıp, yazdırabilirsiniz.

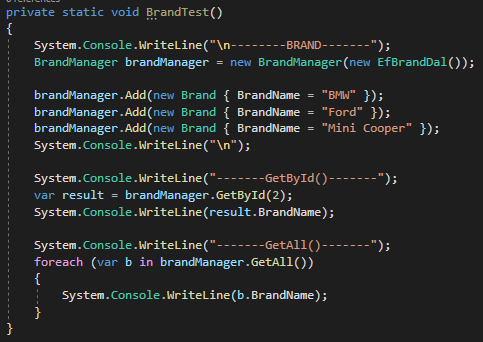
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldumetin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Geliştirme projesi ReCapProject 10.GÜN

Senaryo Devam :

* **Core katmanında Results yapılandırmasını yapınız.**
* **Daha önce geliştirdiğiniz tüm Business sınıflarını bu yapıya göre Refactor (kodu iyileştirme) ediniz.**

Result sonuç demek. Öncelikle biz Business katmanımızda metotları yazarken sadece Datayı döndürüyoruz ama bundan hariç mesela mesaj, işlem sonucu(true,false) da döndürmek isteyebiliriz bu durumda ne yapacağız ? Sonuç isimli sınıf oluşturarak, işleme başlayalım.

**Core/Utilities/Results/IResult.cs :**  IResult isimli interfacemiz olsun bu sınıfın içerisinde bir tane işlem sonucu, bir tane de kullanıcıyı bilgilendirmek adına mesaj olsun. Result sonuç demek, Success başarılı mı başarısız mı, Message yapmaya çalıştığın işlem başarılı true o zaman ürünü ekle gibi bilgilendirme.

Hatırla iki property vardı Getter, Setter. Getter okumak için, Setter yazmak için. Get diyoruz sadece. Çünkü programcı gidip bu alanlarda kafasına göre birşeyler yazamasın. Sınırlandırıyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Core/Utilities/Results/Result.cs :** IResult'un somut sınıfı. Result sen bir IResultsun. Getter return demekti o yüzden buraya ne yazarsak return eder.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Business/Abstract/ICarService.cs :** Sınıfları oluşturduk burada mesela void bir şey döndürmüyor ama biz Add operasyonu işlem sonucu true,false mu ya da mesaj döndürsün istiyoruz o zaman void yerine bu iki propertyi taşıyan IResult diyebiliriz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Business/Concrete/CarManager.cs :** CarService sınıfının somut hali orada değişiklik yaptık burada da düzeltmeliyiz. Void yerine IResult yazmalıyız ve döndürdüğümüz şey IResult return ile döndürmeliyiz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Dönen değer Result içerisine (true,”Ürün Eklendi”) yazdırmak istiyorum. Bunu yapabilmenin yöntemi Constructor’dır. Çünkü iki tane parametre yolluyoruz ama parametreler mevcut değil. Result hata verir çözüm olarak Result sınıfına constructor yapısını kur dediğimizde sorun çözülür.

**Result :**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Constructor’ın içerisini dolduralım. Aşağıdaki Message, parametredeki message’a eşitleyelim. Peki Message hani getti, set edilemezdi ? Getter read only’dir. Read onlyler Constructor da set edilebilirler! Zaten Constructor dışında Set etmeyeceğiz, o yüzden bu yapıyı kurduk getter dedik setter koymadık ki tamamen constructor yapısıyla kullanalım. Setter konulsaydı programcı kafasına göre kullanabilirdi (mesela result.success) ama biz onu sınırlandırarak,yazmamasını sağlıyoruz. Tip dönüşümlerini (return) constructorla yapacağız. Kodların okunurluğu standart olsun diye bu yapıyı standartize ediyoruz.

Resultta bu yapıyı kurduk Add metodunda sonuç başarılıysa message verecek. Ama ben mesaj yazdırmak istemiyorum sadece sonuç başarılı mı onu döndürmek istiyorum kullanıcıya o zaman bir metot daha yazarız.

metin içeren bir resim

Açıklama otomatik olarak oluşturulduBu şekilde yazdığımızda da kırmızı kutucuk çalıştırıldığında true,false döndürür ama üstteki metot çalıştırıldığında hem true,false hem de message döndürür. Üstteki metot alttakinin işlevini de görüyor o zaman üstteki metot içerisindeki success bilgisini kırmızı kutucuk içindeki metot set etsin diyoruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ama üstteki metot çalıştırıldığında ; alttaki metot da çalışsın o zaman this yapısını kullanıyoruz. C# da This demek classın kendisi demek. Result’u kastediyor. Her ikisi de duruma göre çalışır bu haldeyken.

**Kırmızı Çizgi ;** Result’un tek parametreli constructoruna success’i yolla.

**Result sınıfının son hali ;**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi bu sistemi biraz daha güzelleştirelim. Result yerine Success Result ya da Error Result döndürebilsin yapıyı geliştiriyoruz. Core katmanında SuccessResult ve ErrorResult classlarını oluştur.

**Core/Utilities/Results/SuccessResult.cs :** Sen bir Result’sun. İki metot olsun 1. SuccessResult sadece mesaj göndersin, Resultta ya success, ya da hem success hem mesaj göndersin. Base result demek. 2. İşlem sonucu döndürsün sadece mesaj vermesin.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

CarManager da Add operasyonu bu şekildeydi ben artık kullanıcıya burada true döndürmeyeyim, burası SuccessResult olsun ve sadece message yazsın, true ifadesini SuccessResult sınıfında döndüreyim demektir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kullanıcıya mesaj vermek istemiyoruz sadece true yollayacağız o zaman ne olacak? O zaman base ‘i tek parametreli olanı çalıştır. Kısacası CarManager da **return new SuccessResult( )** bu şekilde yazdığımızda SuccessResult sınıfına gider SuccessResult’u default olan base de true olan yani Result sınıfında. Result sınıfında onu bulup çalıştırıyor.

Madem SuccessResult sınıfımız var bir de ErrorResult sınıfımız olmasın mı ? Add operasyonunu şu şekilde ErrorResult dahil ederek yazmayalım mı ? ErrorResult sınıfı oluşturalım.

**Core/Utilities/Results/ErrorResult.cs :** HataSonucu sen bir Resultsun. True değil false döndürüyorsun. Dolasıyla Add operasyonunda if içerisinde false değer , değilse true değer işlevlenecek.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

CarManagerdaki Add operasyonumuz ; Araba isim uzunluğu 2 den büyük ve Günlük fiyatı 0 dan büyükse başarı mesajını döndür, arabayı ekle ; değilse hata mesajını döndür.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Mesajlar kötü bir yazılım şuan bunlardan kurtulalım. Dönen resultların içerisindeki stringler literatürde **Magic Strings** olarak adlandırılır. Bunun için çözüm geliştirelim.

O zaman Business katmanına bir klasör ekleyelim. Constant(Sabit) bu klasör proje sabitlerini içerir. Mesela metinler, mesajlar, IEnumerable içerir.

**Business katmanımın yeni hali ;**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Business/Constants/Messages.cs :** Bu static bir constant. Mesajları yazdık.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Artık CarManager da magic stringsleri temizleyerek daha iyi bir yazılım elde edebiliriz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

CarService sınıfına gelelim. Add operasyonunda void için IResult demiştik. Listler için burada da bir liste döndürdüğümüz için IDataResult diyelim. İçerisinde List<Product>(list of product) olan data sonucu. IDataResult hem işlem sonucunu, hem mesaj, hem de döndüreceği List<Product>(list of product) içeren bir yapı görevi görecek.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

O zaman Core/Utilities/Results altında bir IDataResult sınıfı oluşturalım.

**Core/Utilities/Results/IDataResult.cs :** Hangi tipi döndüreceğini bana söyle. IDataResult sen bir IResultsun. IDataResutta IResult gibi mesaj içersin, işlem sonucu içersin ama aynı zamanda data içersin. Mesajla, işlem sonucunu zaten IResult içeriyordu zaten o zaman IResult kalıtım veririz. Onun haricinde bir de T türünde data olacak.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yani kısacası IDataResult ama jenerik, hangi tiple çalışacağımı söylemem gerekiyor çünkü her metot başka bir şey döndürüyor(araba,marka listesi,dto gibi). Aynı zamanda IResulttaki yapılar var bir de ek olarak sen de Data var.

**Core/Utilities/Results/DataResult.cs:** Bir DataResult sınıfı oluştur Core da. Result sınıfından farkı Data içermesi. DataResult sen hem Resultsun hem de IDataResultsun. T tipinde Data döndür.

metin, iç mekan, ekran, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

CarService gel şimdi T sini ver. Önceden List<Car> döndürüyorduk artık diyorumki ben List<Car> ile birlikte mesajı da, işlem sonucu da döndürücem. GetById Car döndürüyordu T o zaman Car olur.   
GetAll List<Car > döndürüyordu T o zaman List<Car> döndürür. Bunlara dikkat et.

**Business/Abstract/ICarService.cs : sınıfımın son hali ;**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Business/Concrete/CarManager.cs :** ICarService te IDataResult yaptık yapıyı burada düzeltmeliyiz aynı şekilde. Artık DataResult döndürmek istiyorum.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ben DataResult döndürüyorum çalıştığım tip List<Car> **(Yeşil çizgi)**, döndürdüğüm data**(Kırmızı çizgi)**, işlem sonucum**(Sarı çizgi)**, mesajım**(Pembe çizgi)**

Nasılki Result’un SuccessResult,ErrorResult’unu gerçekleştirdik aynı şekilde süreci daha kolaylaştırmak için DataResultunda Success ve Error lerini gerçekleştirebiliriz.

**Core/Utilities/Results/SuccessDataResult.cs:** Sen bir DataResultsun.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Core/Utilities/Results/ErrorDataResult.cs:** Sen bir DataResultsun.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Business/Concrete/CarManager.cs :** Şimdi CarManagerdaki GetAll operasyonunu SuccessResult ve ErrorResult kullanarak yazalım daha düzgün kod olsun.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Kırmızı Çizgi;** Saat 22.00 olduğunda MaintenanceTime(Bakım) yapılsın.

Zaten SuccessResult true ya yazmayalım onu, arabalar eklendi yerine de Message yazalım.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Artık ProductManagerdaki tüm operasyonları düzenleyebiliriz. Tüm metotları GetAll gibi yapınız.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Messages sınıfının son hali;**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Console.Program.cs : CarDetailDto metodumuz IDataResult döndürdüğü için burada da iyileştirmeleri yapalım. Geriye kalan Color, Brand nesnelerimiz içinde aynı işlemi uygulayalım.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu