**DÜZCE ÜNİVERSİTESİ**

**TEKNOLOJİ FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ GÖRSEL PROGRAMLAMA PROJE RAPORU**

**182114053 MURAT DEMİR / İÖ**

**KÜTÜPHANE OTOMASYONU**

İçindekiler

[GİRİŞ 3](#_Toc61598522)

[Veri Tabanını Oluşturmak 3](#_Toc61598523)

[Emanet Tablosu Neden Var? 3](#_Toc61598524)

[Projeyi Oluşturmak 3](#_Toc61598525)

[Katmanları Açıklamak 4](#_Toc61598526)

[Katmanlı Mimari Nedir? 4](#_Toc61598527)

[Data Access (Veri Erişimi) Katmanı 4](#_Toc61598528)

[1. DBAccessConnection.cs 5](#_Toc61598529)

[2. Kitap.cs 5](#_Toc61598530)

[3. Ogrenci.cs 8](#_Toc61598531)

[4. EmanetIslemleri.cs 8](#_Toc61598532)

[Business (İş) Katmanı 8](#_Toc61598533)

[1. KitapManager.cs 9](#_Toc61598534)

[2. OgrenciManager.cs 10](#_Toc61598535)

[3. EmanetIslemleriManager.cs 10](#_Toc61598536)

[Layout (Sunum) Katmanı 12](#_Toc61598537)

[1. Kitaplar.cs 13](#_Toc61598538)

[2. KitapEkle.cs 14](#_Toc61598539)

[3. KitapGuncelle.cs 14](#_Toc61598540)

[4. KitapDetay.cs 14](#_Toc61598541)

[5. Ogrenciler.cs 15](#_Toc61598542)

[6. OgrenciEkle.cs 15](#_Toc61598543)

[7. OgrenciGuncelle.cs 15](#_Toc61598544)

[8. OgrenciIslemleri.cs 15](#_Toc61598545)

[9. IslemlerListesi.cs 16](#_Toc61598546)

[10. Grafik.cs 17](#_Toc61598547)

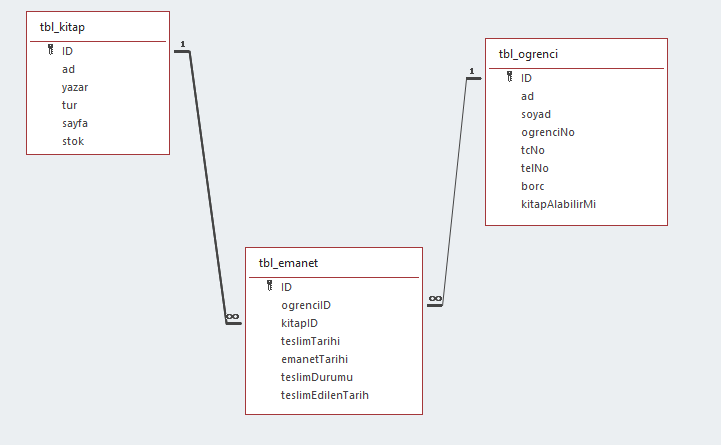
# GİRİŞ

Bu projede amacımız Visual Studio ortamında C# dilinden yararlanarak, verilerin veritabanında saklandığı, katmanlı mimariyi kullanan bir kütüphane otomasyonu oluşturmak. Projem için Visual Studio’da Windows Form uygulamasını kullandım. Veri tabanı yönetimi için Microsoft Access’ten yararlandım.

# Veri Tabanını Oluşturmak

Access’te 3 adet tablo oluşturdum;

* Kitap bilgileri için: **tbl\_kitap**
* Kütüphanede kitap işlemleri yapacak öğrenciler için: **tbl\_ogrenci**
* Kitap ile öğrenci arasındaki ilişkiyi kurmak için: **tbl\_emanet**

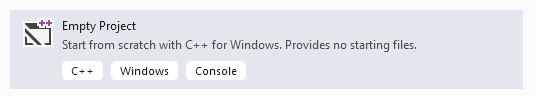


## Emanet Tablosu Neden Var?

Bir öğrenci kitap almak istediğinde emanet tablosunda öğrenci ile alınan kitabın ID’leri ve işlem bilgileri tutulur. Bir öğrenci birden fazla kitap alabileceği gibi, bir kitabı da birden fazla öğrenci alabilir, bu durumu simüle edebilmek için 3.bir tablo olan emanet işlemleri tablosuna ihtiyaç duydum.

# Projeyi Oluşturmak

Visual Studio’da katmanlı mimari kullanacağım için öncellikle boş bir proje oluşturdum.



Daha sonra projemin katman sayısına (3 Katman) karar verdim ve katman başlıklarını belirledim.



# Katmanları Açıklamak

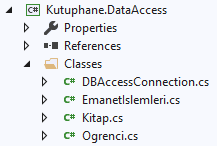
## Katmanlı Mimari Nedir?

Katmanlı Mimaride amaç işleri fonksiyonel gruplara bölerek, bu bölümlerin birbirlerine bu katmanlar aracılığı ile ulaşmasını sağlamaktır. Bu sayede proje büyüdükçe işler çok daha düzenli ve kolay kullanılabilir olacaktır.

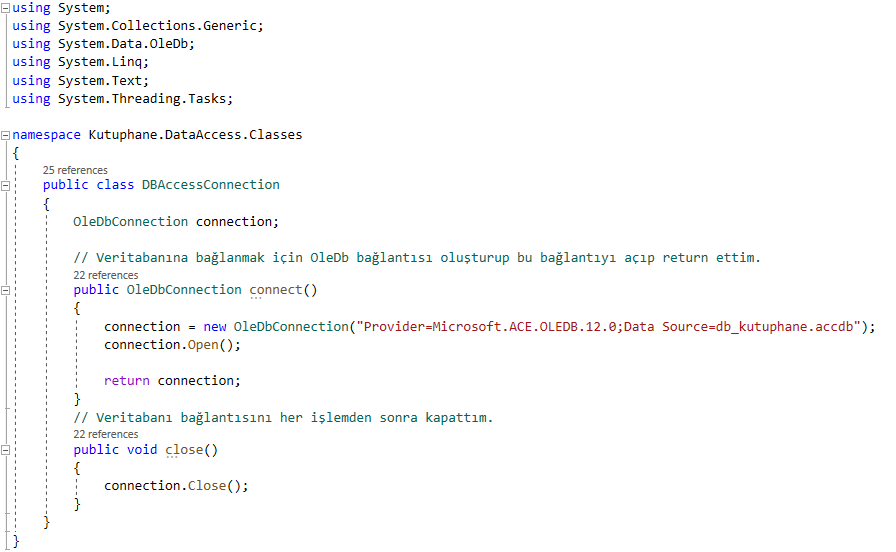
## Data Access (Veri Erişimi) Katmanı

Bu katmanın görevi projem ile veri tabanı arasındaki bağlantıyı kurmaktır. Projemin hiçbir aşamasında bu katmana uğramadan veri tabanımdaki verilere ne ulaşabilir ne de veri yükleyebilirim.

Bu katmanda 4 adet sınıf oluşturdum.



### 1. DBAccessConnection.cs

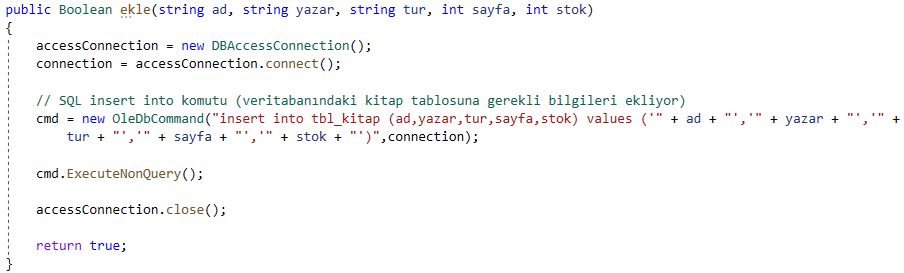


**OleDbConnection:** Sınıflarımın Ole Db desteği olan veri kaynaklarına erişebilmesi için OleDbConnection kullandım. Bu katmandaki diğer sınıflarım bu bağlantı sayesinde veri tabanım ile iletişim kuracak ve işi bittiğinde bu sınıftaki close() metodu ile bağlantıyı sonlandırabilecek.

### 2. Kitap.cs

Kitap sınıfımın içinde kitap ekleme, silme ve güncelleme işlemleri **sorgusuz** yapılmaktadır. Bu katmandaki Kitap sınıfımın tek görevi kendisine gelen verilerle istenen işlemi yapmaktır.

**ekle()** fonksiyonum kendisine gelen; kitap adı, yazar adı, kitap türü, sayfa sayısı ve stok adedi bilgilerini veri tabanımdaki kitap tabloma yeni veri ekliyor.

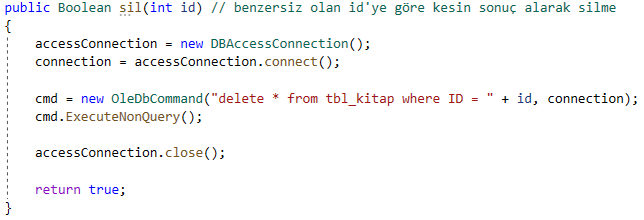


Veri tabanına bağlanmak için DBAccessConnection aracılığıyla bir connection (bağlantı) oluşturup bu bağlantıyı kullanarak veri tabanında SQL komutlarımı çalıştırdım.

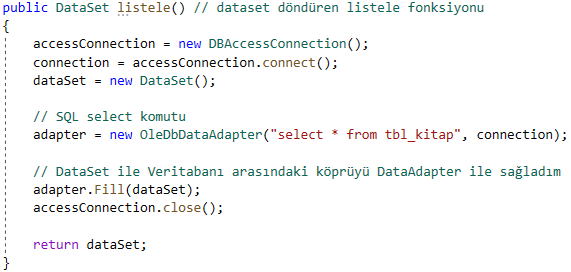
**OleDbCommand:** SQL komut cümlesi kullanarak veri kaynağındaki tablolarda sorgulama yapmamızı sağlar.

**ExecuteNonQuery:** SQL’deki INSERT, DELETE, UPDATE ve SET deyimleri gibi değer döndürmeyen komutları çalıştırır.

**sil()** fonksiyonum kendisine gelen kitap ID’si ile bir SQL sorgusu yaparak kitabı tablomdan siliyor.



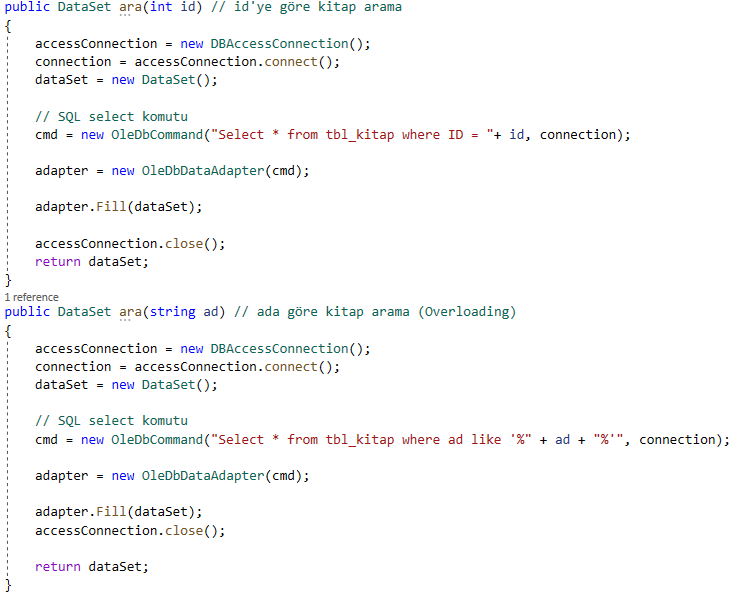
**listele()** fonksiyonum kitap tablomdaki bütün kitapların bilgilerini çekip data setime aktarıyor. Fonksiyonum çağırıldığında bu data setini daha sonra sunum katmanında kullanmak üzere geri döndürüyor.



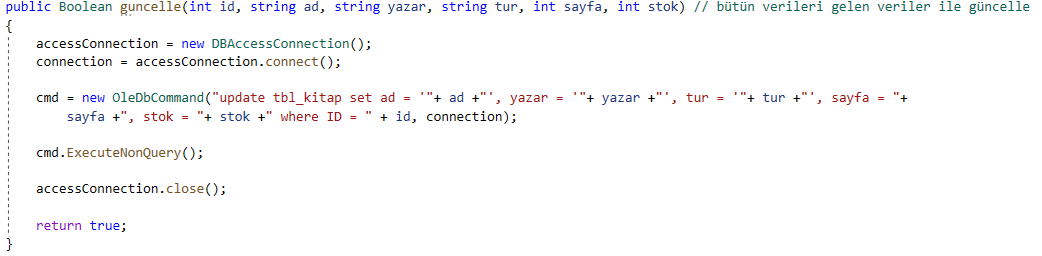
**OleDbDataAdapter**: Verileri almak ve kaydetmek için data set ve veri kaynağı arasında bir köprü görevi görür.

**DataSet:** Veri tabanındaki verileri tutmak için kullanılır. Daha sonra tablo oluşturma işlemlerinde çok fazla kolaylık sağlayacak.

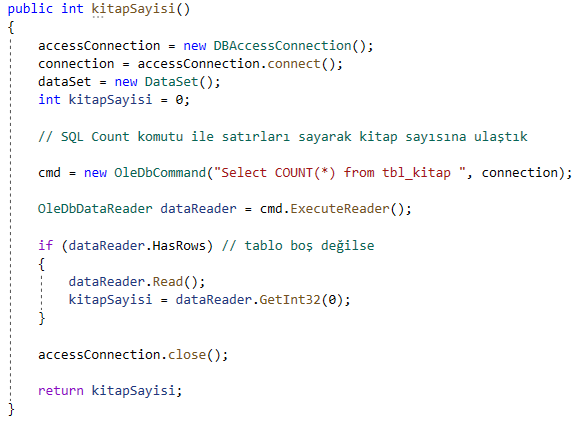
**ara()** fonksiyonum kitaplar tablosunda ID’si veya Adı girilen verinin varlığını kontrol eder. Eğer bu kitap varsa DataSet ile bilgilerini çağırıldığı uzaya geri gönderir.



**guncelle()** fonksiyonum kitap tablomda olan verinin bilgilerini kendisine gelen bilgilerle değiştirmek için kullanılıyor.



Son olarak daha sonra grafikte ihtiyacım olacağı için **SQL COUNT** sorgusu ile tablomdaki kitap sayısını sayan ve döndüren bir fonksiyon tasarladım.



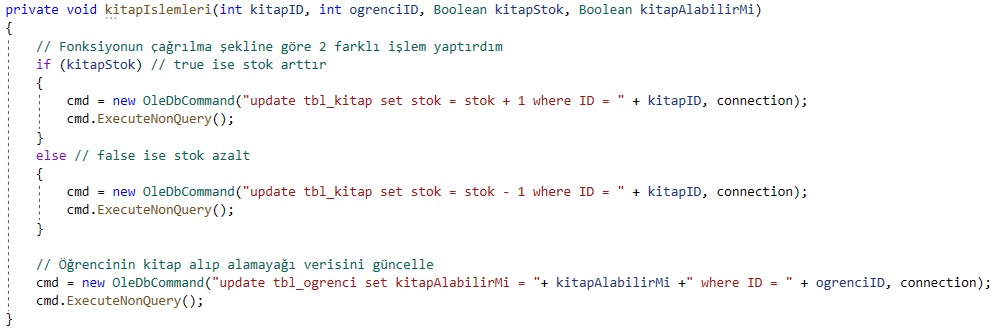
### 3. Ogrenci.cs

Bu sınıfım da Kitap sınıfının yaptığı her şeyi Öğrenci tablom için yapmaktadır.

### 4. EmanetIslemleri.cs

Bu sınıfım da benzer yapıları kullanarak listeleme, kitap alma ve kitap verme işlemlerini yapmaktadır.

Kitap alındığında veya iade edildiğinde kitapIslemleri fonksiyonum çağırılıyor. Bu fonksiyonun amacı yapılan işleme göre kitap stok durumunu ve öğrencinin kitap alabilme durumunu güncellemektir. Buradaki sorgu fazladan kod yazmamak için yapıldığından katmanlı mimariye ters düşmemektedir.

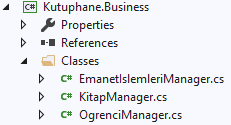


## 

## Business (İş) Katmanı

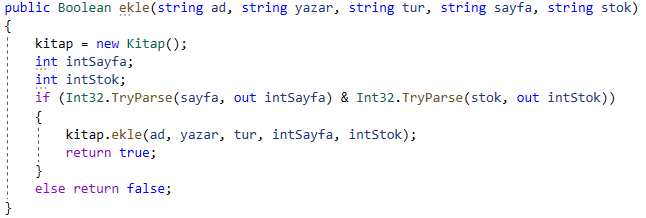
Bu katmanın görevi yapılacaklar işlemler için gerekli sorgulamaları yapmaktır. Sorgulama başarılı ise veri tabanı katmanı çağırılarak işlem tamamlanır. Sorgulama başarısız ise veri katmanı çağırılmaz ve işlem yapılmaz.

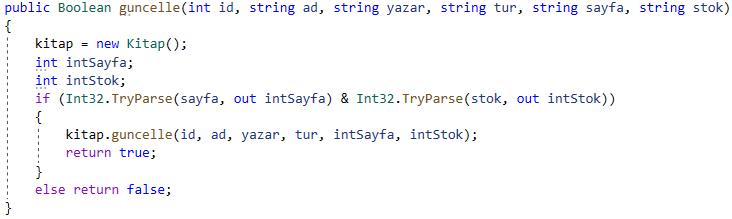
Bu katmanda 3 adet sınıf oluşturdum.



### 1. KitapManager.cs

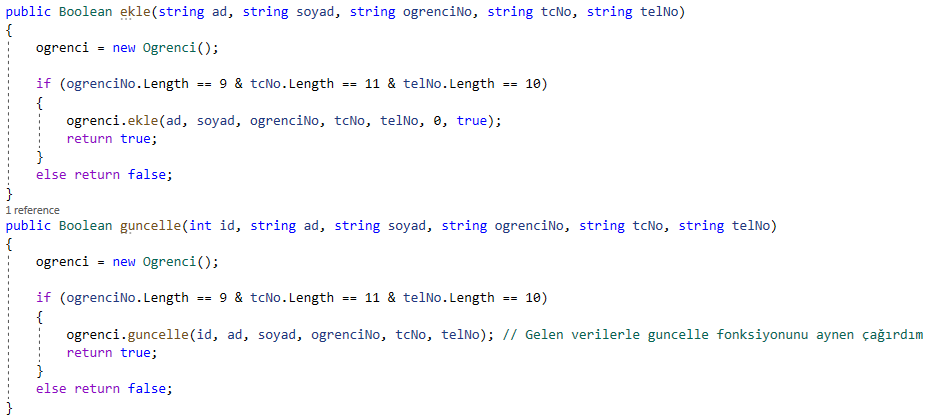
Bu sınıftaki ekle() ve guncelle() fonksiyonlarım gelen **String** türündeki sayfa ve stok değişkenlerini **Integera** çevirmeyi deniyor. Eğer bir sorun yoksa dönüşüm yaptığı değişkenleri kullanarak veri tabanı katmanındaki fonksiyonlarımı çağırıyor. Son olarak arayüz katmanındaki hata mesajlarım için, yapılan işlemin sonucuna göre **True** veya **False** değerleri döndürüyor. Bunun yanı sıra bu sınıfımda ekleme, listeleme, arama ve kitap sayısı döndüren fonksiyonlarım da bulunmakta.



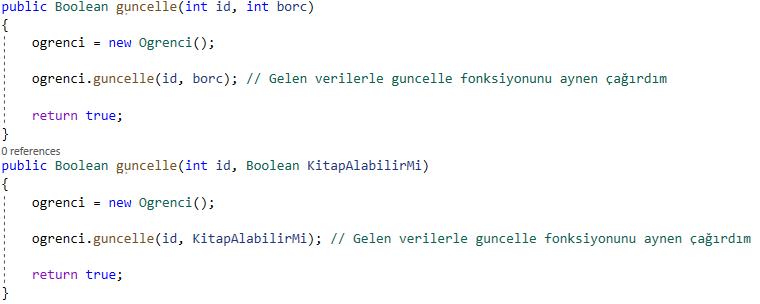


### 2. OgrenciManager.cs

Bu sınıftaki **ekle()** ve **guncelle()** fonksiyonlarım gelen numaraların istenen uzunlukta olup olmadığını kontrol ediyor. Yine bu sınıfta da listeleme, arama ve silme fonksiyonlarım bulunmakta.

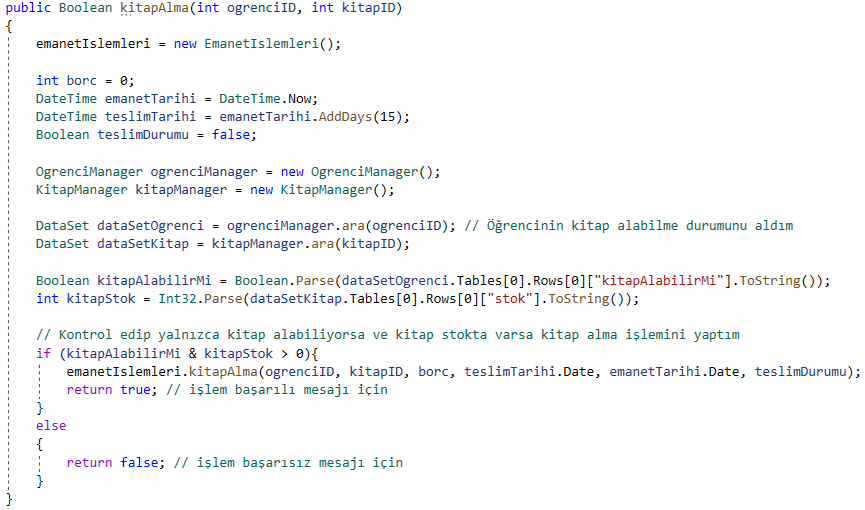


**guncelle()** fonksiyonumun sorgu yapılmayan, sadece bir veriyi değiştirmek için kullanılan aşırı yüklenmiş diğer fonksiyonları;

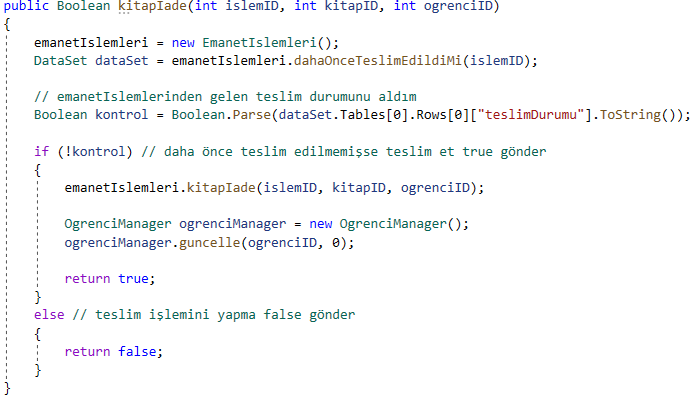


### 3. EmanetIslemleriManager.cs

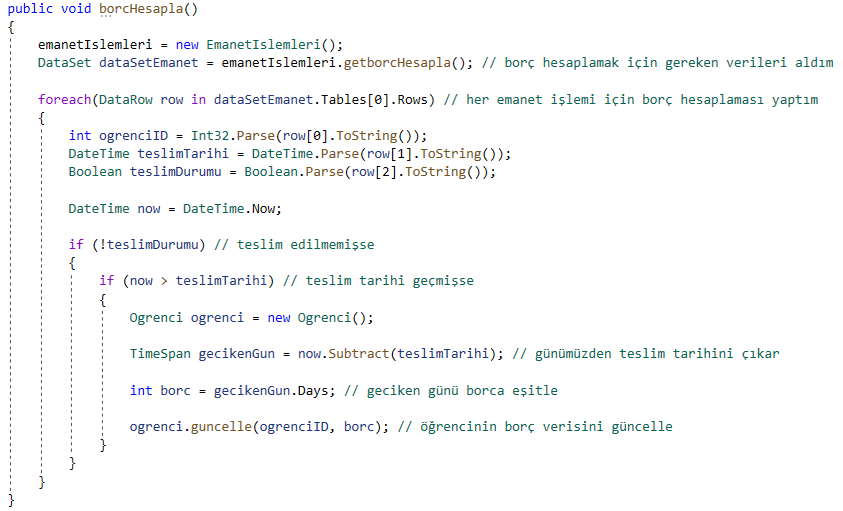
**kitapAlma()** fonksiyonumda OgrenciManager ve KitapManager sınıflarımdaki arama fonksiyonunu kullanarak öğrenci ve kitap hakkındaki bilgileri alıyorum. Öğrencinin elinde kitap yoksa ve stokta kitap varsa öğrenci işlemi başarılı şekilde gerçekleşiyor. Yeni veriler emanet tablosunda güncelleniyor.



**kitapIade()** fonksiyonum tablodan alınan teslim durumu bilgisine göre iade işlemini gerçekleştiriyor. Daha önceden teslim edilmediyse teslim işlemi yapılıyor ve öğrencinin borcu sıfırlanıyor.



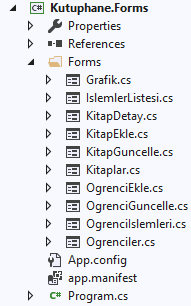
**borcHesapla()** fonksiyonu bütün emanet işlemleri için borç hesaplıyor. Borcu yalnızca teslim edilmemiş ve teslim tarihi geçmiş kitaplar için hesaplıyor. Borç hesabı her geciken gün için + **₺1** olacak şekilde hesaplanıyor.



## Layout (Sunum) Katmanı

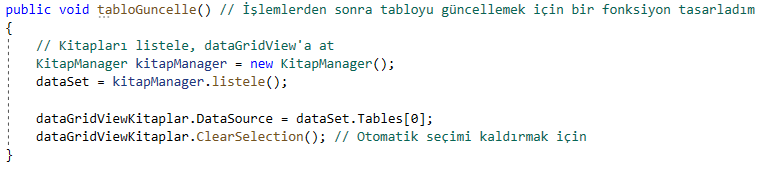
Bu katmanın görevi kullanıcıya programın sunumunu yapmak ve kullanıcı ile etkileşime girmektir. Başka bir deyişle **Backendi** hallettik artık **Frontend** kısmındayız.

Bu katmanda 10 adet form oluşturdum.

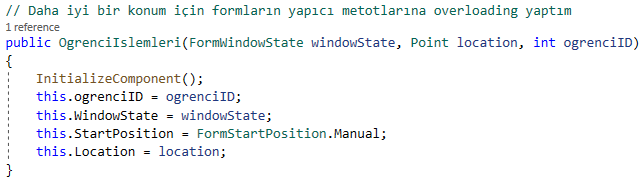


Tabloda veri listelediğim form sayfalarında tabloGuncelle() fonksiyonları yazdım.

Bu fonksiyon yapılan her işlemden sonra formumdaki DataGridView tablomun yenilenmesini sağlıyor.

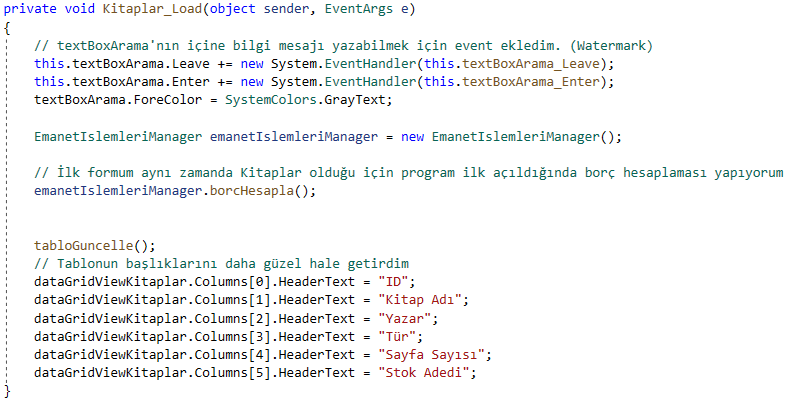


Yeni bir pencerede açılacak her formun yapıcı metodunu aşırı yükleyerek, kendisini çağıran formun pencere durumunu ve bölgesini kendi için de geçerli kılarak daha estetik form geçişleri sağladım.



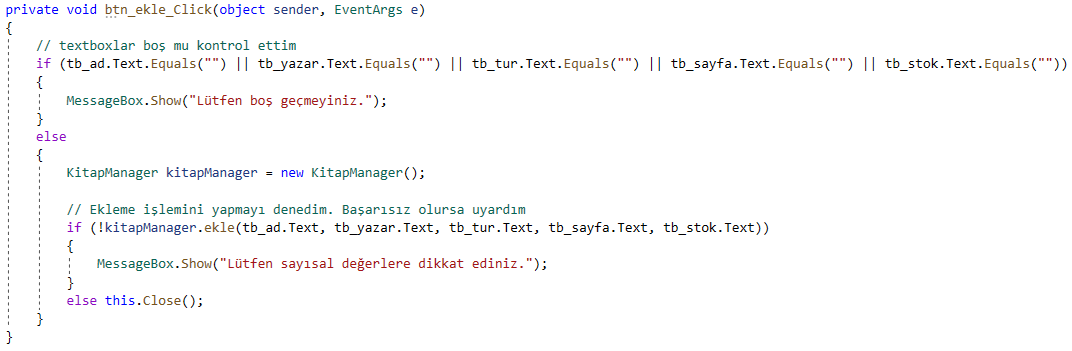
### 1. Kitaplar.cs

Bu form aynı zamanda programın ilk açılan formudur. Form yüklendiği zaman textBoxArama’ya 2 adet event ekledim. Bunu textbox’ın içine **Watermark** denilen bilgi mesajını yazmak için kullandım. Geçmiş emanet işlemleri için borç hesaplama işlemi de program ilk açıldığında yaptırdım. Ayrıca bu formda kitap ekleme, silme ve güncelleme işlemlerini yapabildiğimiz ve aradığımız kitabın detaylarına ulaşabildiğimiz butonlar bulunmakta.



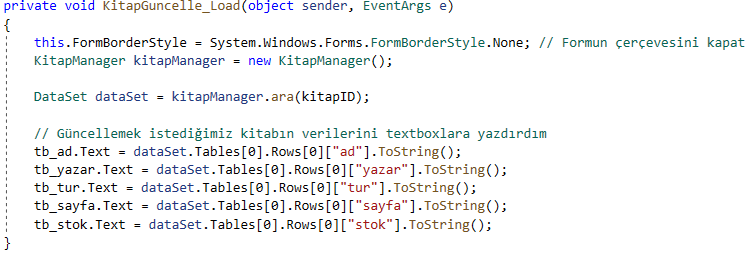
### 2. KitapEkle.cs

Bu form yeni kitap eklemek için kullanılıyor. İstenen verilerin boş bırakılmasına izin verilmiyor. Kitap eklenmeye çalıştığımızda sayısal değerlerde hata yaparsak Business katmanından gelen false değeri sayesinde ekrana uyarı mesajı yazdırıyor.



### 3. KitapGuncelle.cs

Bu form ekranına girebilmek için öncelikle tablodan bir satır seçmemiz gerekiyor. Aksi durumda Kitaplar formunda program bizi uyarıyor. KitapGuncelle formu yüklendiğinde KitapManagerdaki arama fonksiyonu sayesinde tıklanan kitabın verileri textboxlara yazdırılıyor. Daha sonra güncelle butonuna bastığımızda textboxtaki verilerle veri tabanındaki kitap verileri değiştiriliyor.



### 4. KitapDetay.cs

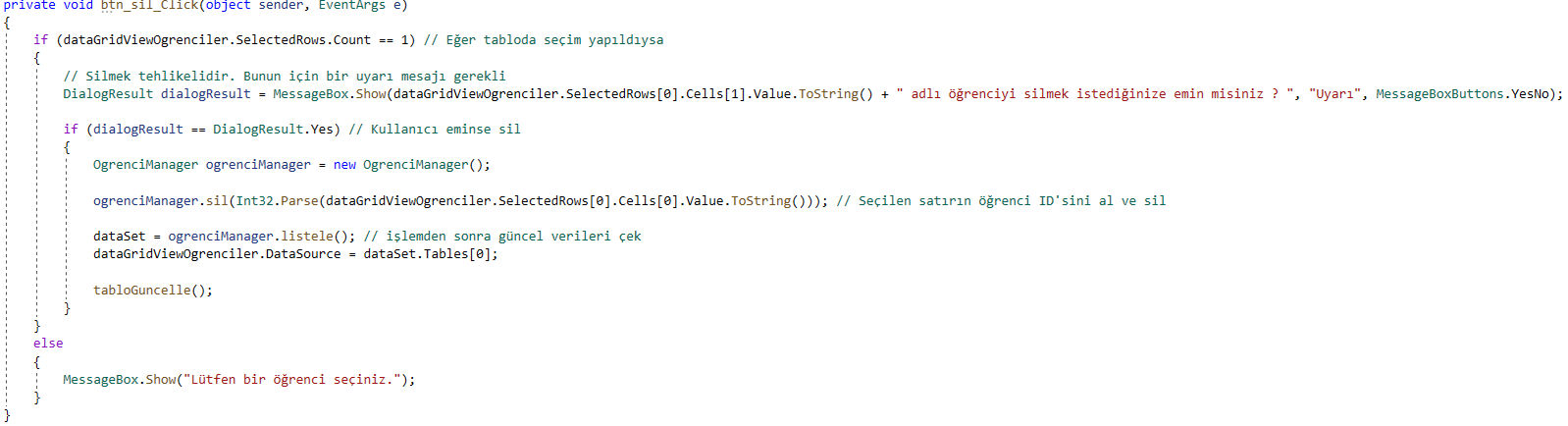
Kitaplar formunda aranan kitap varsa bu form sayfasında kitap hakkında detaylar listeleniyor. Aynı zamanda bu kitabı daha önceden alan, okuyan kişiler EmanetIslemleriManager ve OgrenciManager arama fonksiyonları sayesinde listeleniyor.



### 5. Ogrenciler.cs

Bu formda da Kitaplar formunda yapılan işlemlerin aynısı Öğrenciler için yapılmaktadır. Ekstra bir öğrenci seçilip işlem butonuna tıklandığında öğrenci için açılan yeni bir işlem formu bulunmaktadır.

DataGridView’in SelectedRows.Count sayesinde seçili satır sayısına ulaşabiliyoruz. Tabloda sadece 1 seçime izin verdiğim için eğer seçim sayısı 1 ise tabloda veri seçilmiştir kontrolünü sağladım. Bir öğrenciyi silmek istediğinizde DialogResult sayesinde seçim yapabildiğimiz bir MessageBox oluşturdum. Ve kullanıcı eminse silme işlemini gerçekleştirdim.



### 6. OgrenciEkle.cs

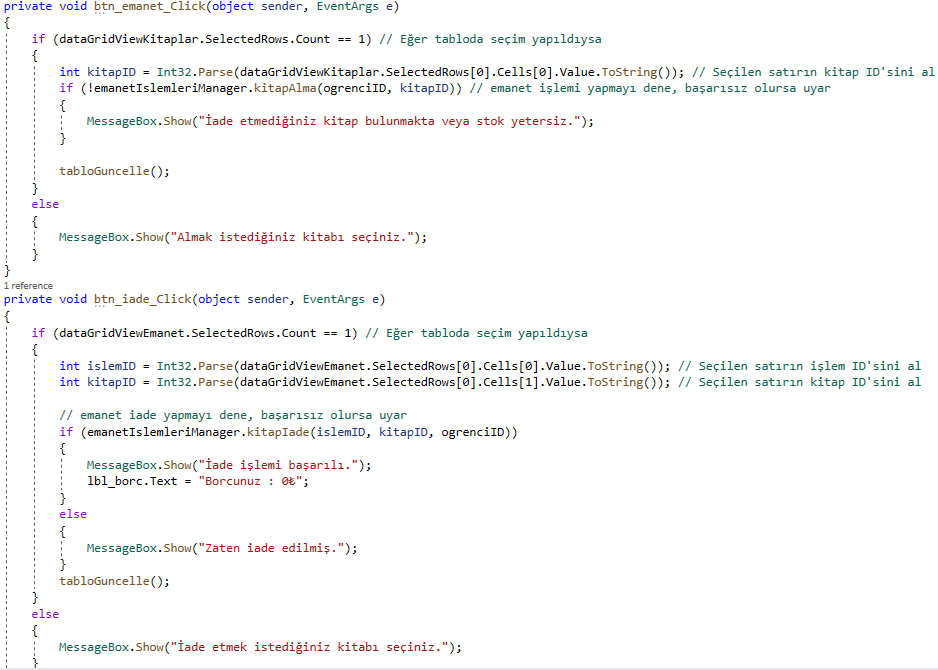
Bu formda KitapEkle formunda kitaplar için yapılan her şey öğrenci için yapılabilmektedir.

### 7. OgrenciGuncelle.cs

Bu formda KitapEkle formunda kitaplar için yapılan her şey öğrenci için yapılabilmektedir.

### 8. OgrenciIslemleri.cs

Bu form Ogrenciler formunda seçilen öğrenci için özel işlem yapılan bir formdur. Öğrenci bu formda kütüphaneden kitap alabilir, emanet alıp okuduğu kitapları iade edebilir ve geçmiş işlemlerini görebilir.

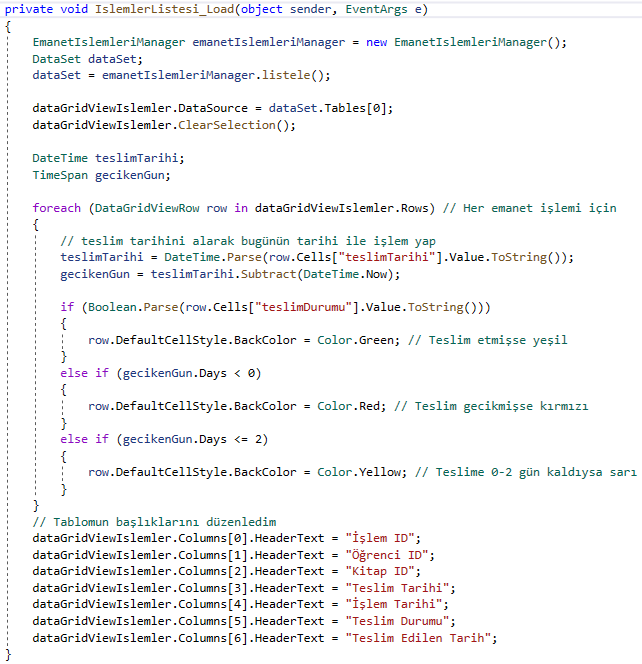


Öğrencinin yaptığı işlemler tablosunun her satırını kontrol etmek için foreach döngüsü kullandım. Teslim tarihinden bugünün tarihini çıkararak teslime kalan günü hesapladım. Öğrenci aldığı kitabı teslim ettiyse satır yeşil, teslim gününe 2 gün veya daha az kaldıysa sarı, teslim gününü geciktirdiyse kırmızı olacaktır.



### 9. IslemlerListesi.cs

Bu form açıldığında EmanetIslemleriManager kullanılarak geçmiş işlemler listelenip tabloya aktarılıyor. Aynı şekilde teslim durumuna ve gecikme duruma göre tablo renkleniyor. DataGridView’ın HeaderText özelliğini kullanarak her sütunun başlığını daha düzenli hale getirdim.



### 10. Grafik.cs

Bu form sayfası ise **ZedGraph** kullanarak kütüphanedeki kitap durumunu grafik haline getiriyor.

