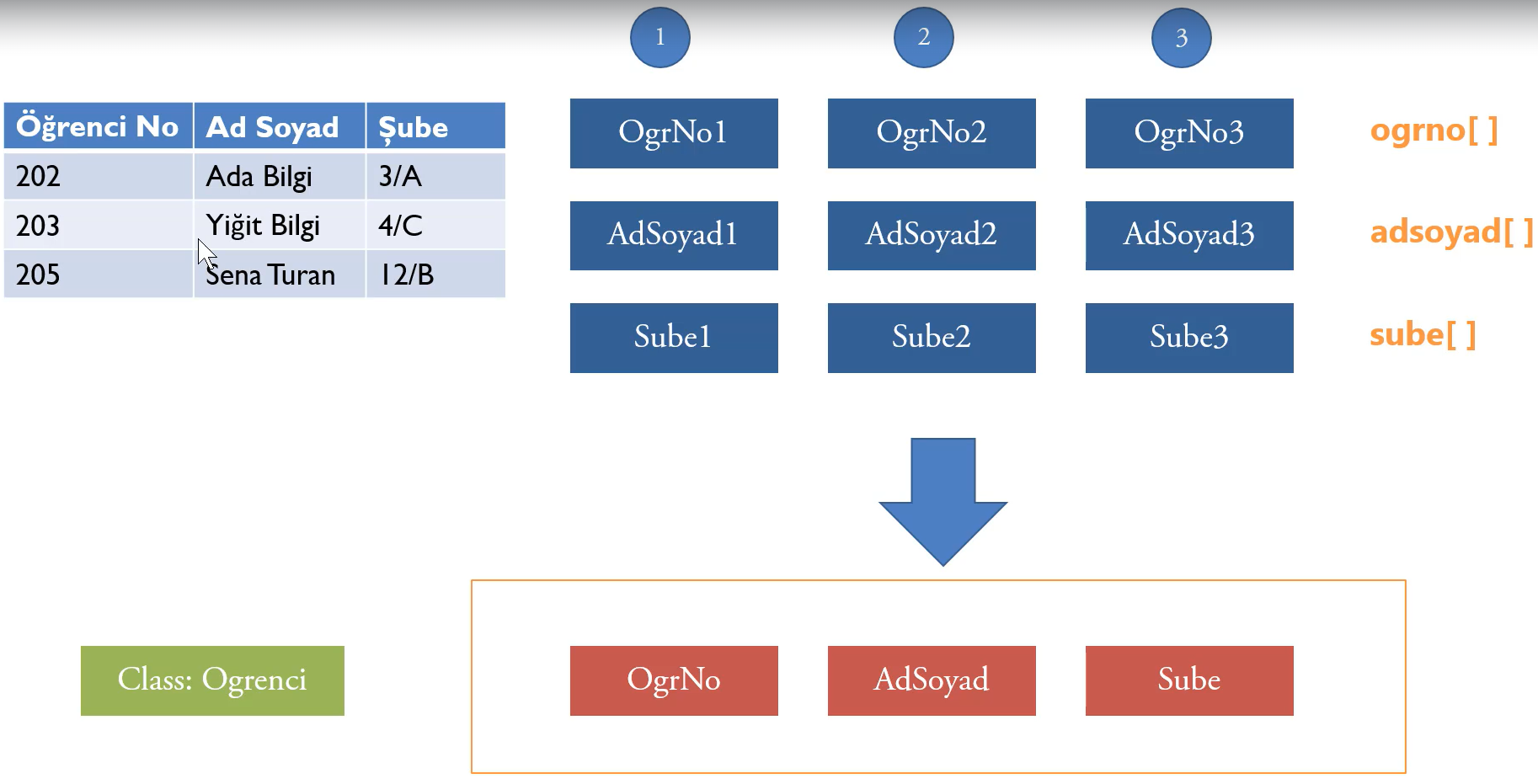
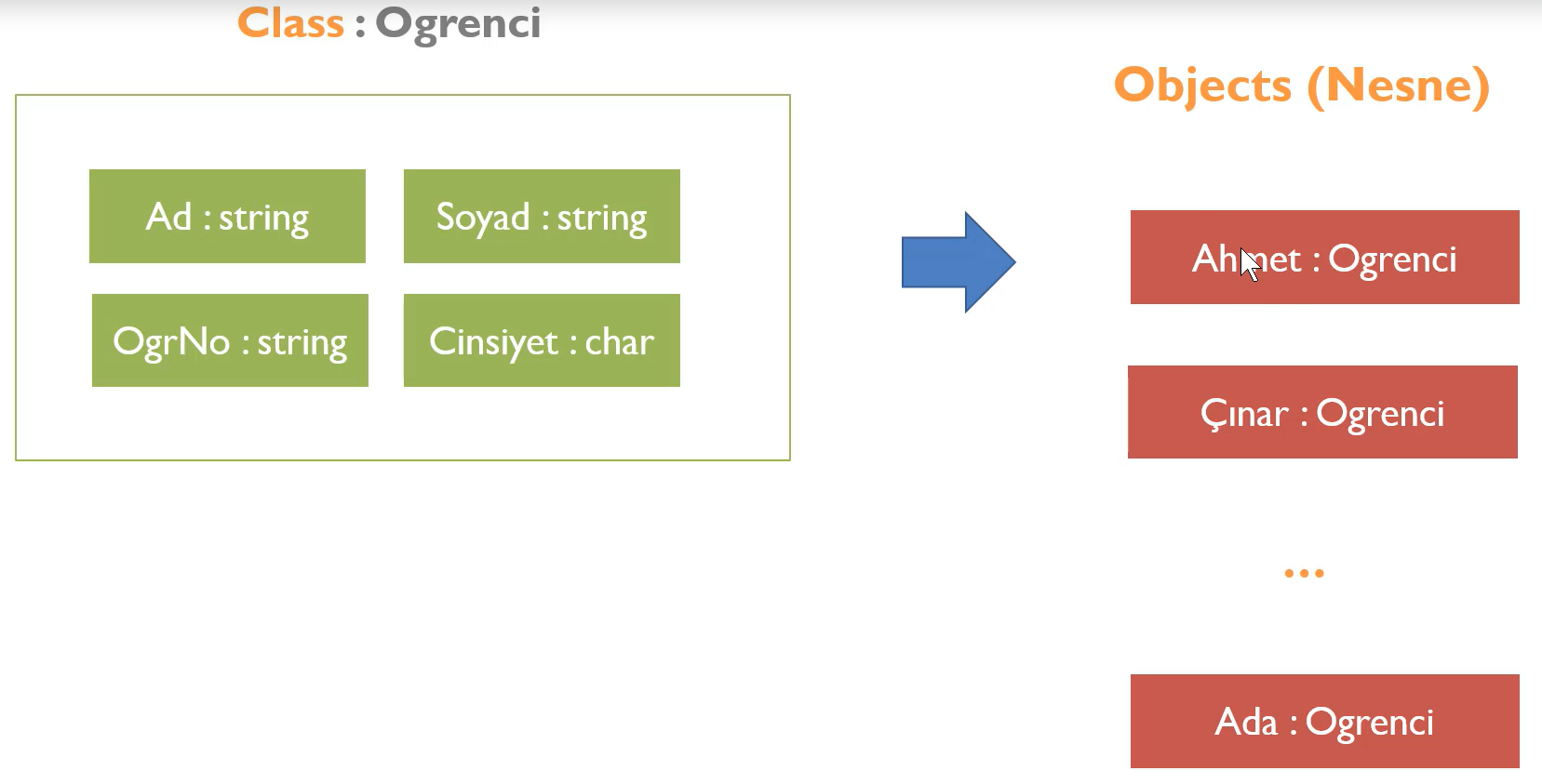
**C# Nesne Yönelimli Programlama**

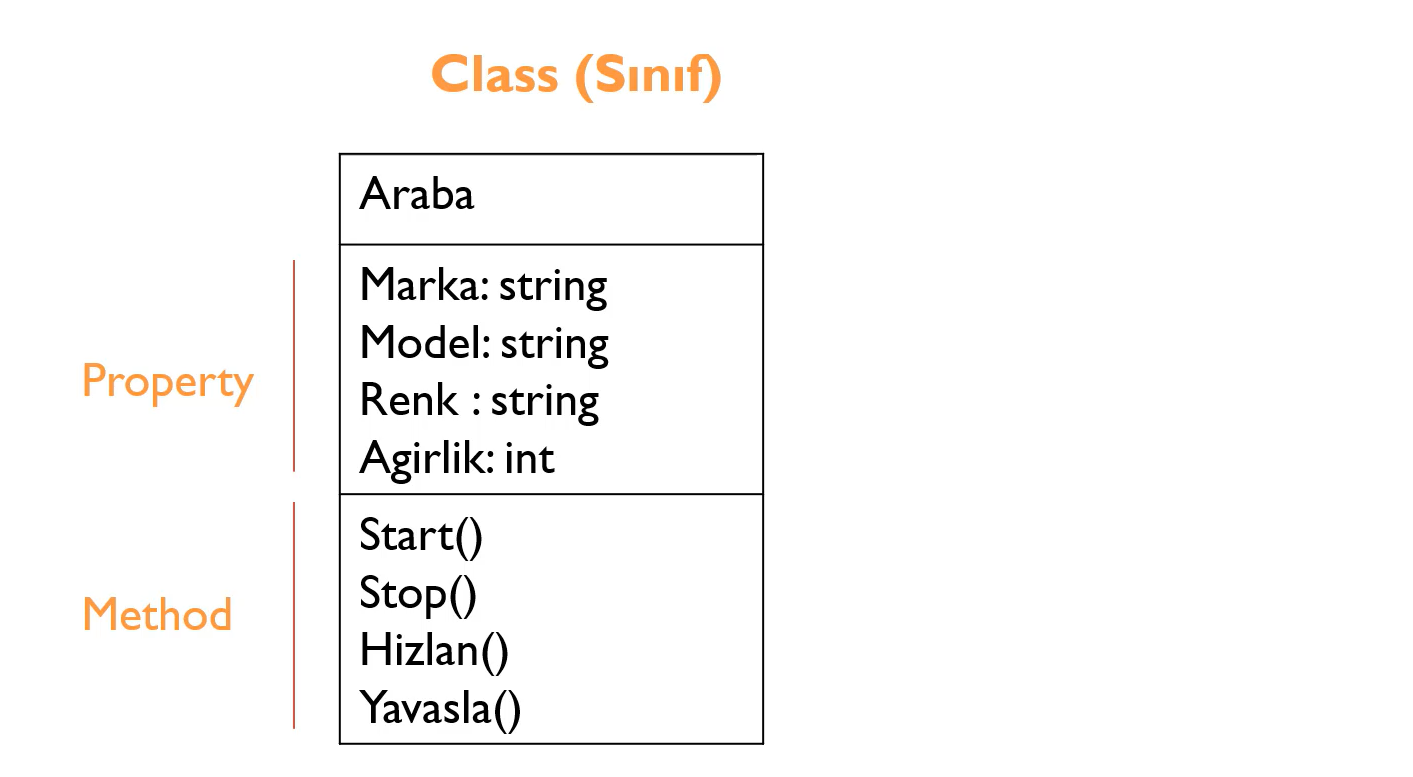
**Class**



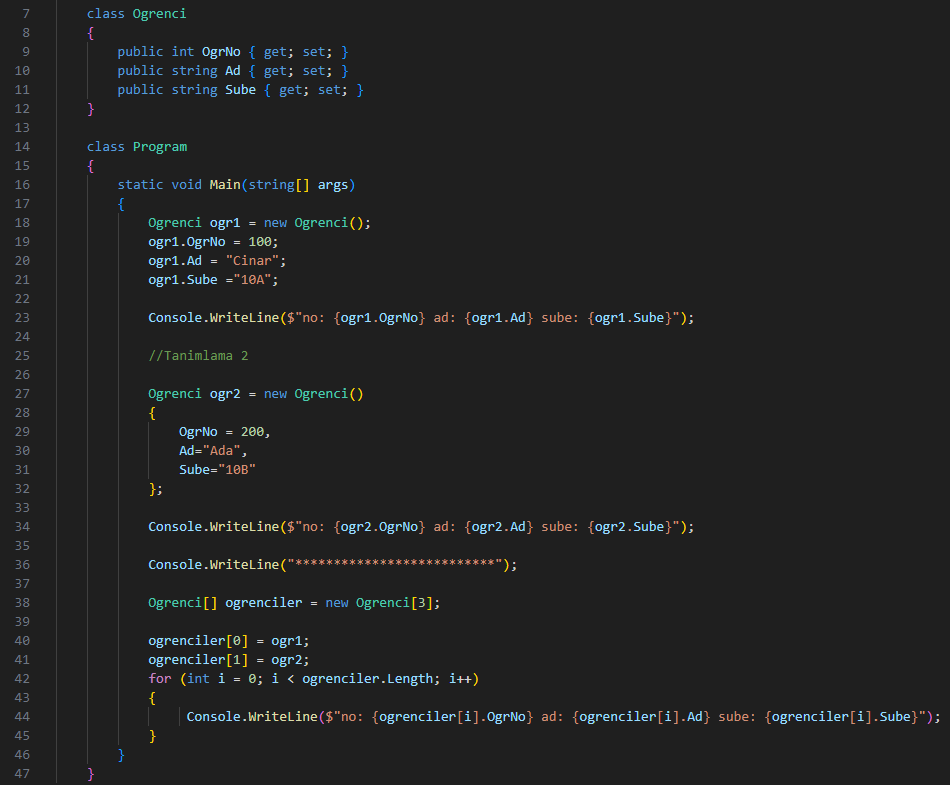
* Biz normal olarak bir öğrenci listesi yaparsak ve isim, okul no ve sube bilgileri olunca farklı listelerde tutabilir. Lakin işler büyüyünce biraz karışıklık olacaktır.
* Aşağı kısımdaki gibi bir class oluşturarak daha kolay kontrol edilebilir bir yapı tasarlayabiliriz.



* Bir class yapısından Objects(nesne) oluşturulmaktadır.

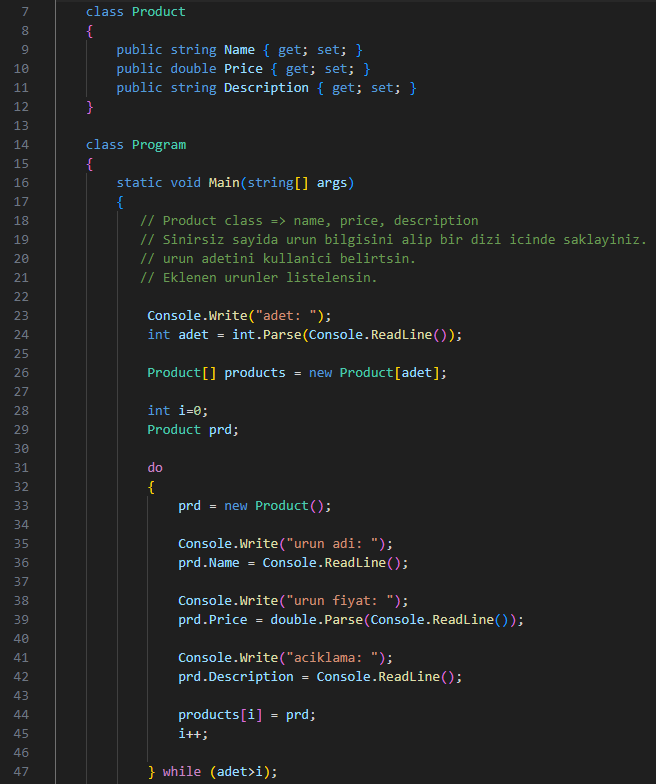


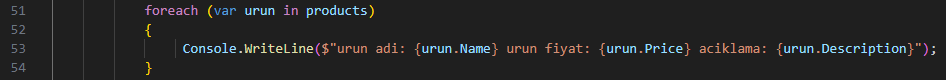
* Class’da bir tek nesne tanımlamanın yanında bu nesnelere methodlar tanımlıyoruz ve bu metotlar sayesinde belli işlemleri bu metotlar içerisinde yapılabilirmektedir.



* İlk olarak 7. Satırda class yazıp class ismini tanımlıyoruz.
* Daha sonra public yazıp property(değişken olarak da düşünülebilir) tanımlıyoruz. Süslü içerisindeki get ve set olmalı.
* 18. Satırda ogr1 isimli Objects(Nesne) tanımlıyoruz.
* 19-21. Satırlarda bu öğrencinin özelliklerini giriyoruz.
* 23. Satırda oluşturduğumuz ogr1 in adı vs yazdırıyoruz.
* 27. Satırda ogr1 kısmına göre daha kolay Objects (nesne) tanımlaması yapıyoruz.
* 38. Satırda Ogrenci dizisi oluşturdum ve 40-41. Satırlarda elemanlarını girdim.
* 42. Satırda for döngüsü ile elemanları yazdırdım.

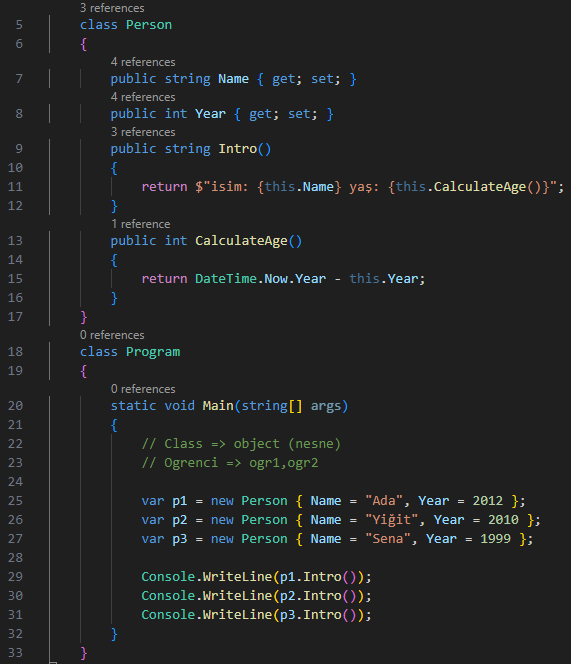
**Class Uygulama**





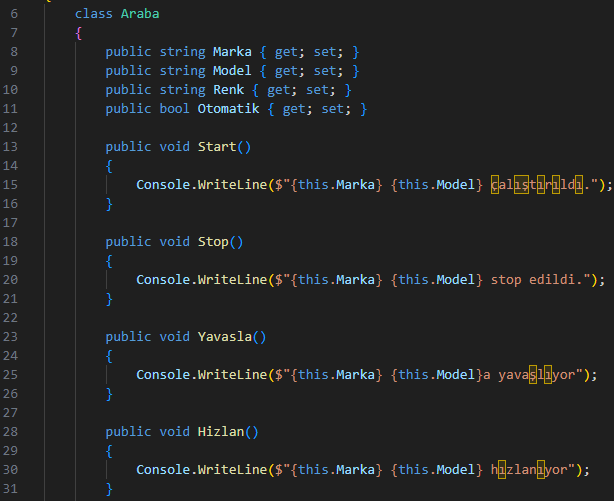
* 9. Satırda class tanımlaması yapıldı.
* Genel anlamda diğer kullanımları önceki notlarda anlattık.
* 51. Satırda foreach döngüsü örneği bulunmakta. Bu döngü ile products listesi içerisindeki elemanları tek tek urun içerisine atama işlemi yapar. Örnek [“bilal”, “mehmet”, “Ahmet”] var mesela ilk olarak bilal, sonra Mehmet sonra Ahmet değerini teker teker urun’e atar ve listenin bütün elemanlarını dolaşır.
* Böylelikle 53. Satırda product[i] yerine urun diyebilirdik. Çünkü product[i] elemanı ürüne eşit oldu.

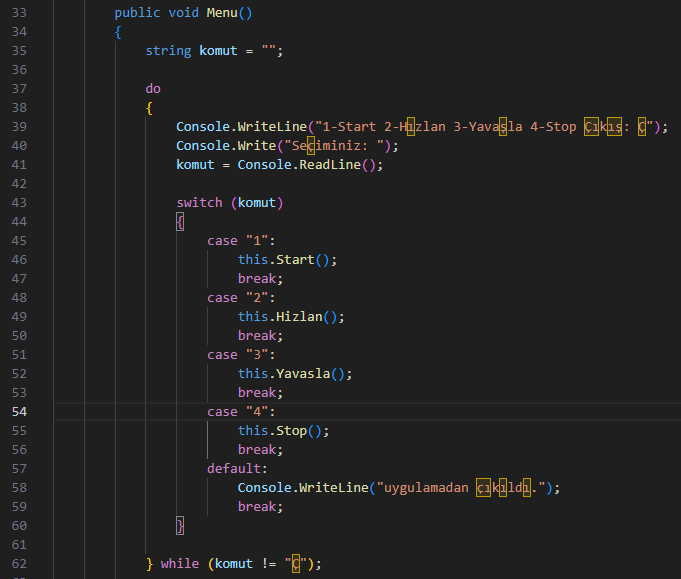
**Metotlar**



* 9. Ve 13. Satırlarda metotlar tanımladık ve bu metotlar bir işlevi yerine getirmektedir.
* Intro metodu isim ve yaş hesaplamasını yapmaktadır, yaş hesaplamasını yaparken alttaki metodu çağırmaktadır. Matot çağrılırken metotismi() şeklinde çağrılır.
* Normalde nesne oluştururken p1, ogr1 gibi isim verip daha sonra ogr1.name çağırıyorduk. Class içerisinde daha nesne tanımlanmadığı ve oluşturulacak tüm nesneler için aynı özelliği tanımlaması için ogr veya p1 yerine this kullanılmaktadır.
* Return ile metoda bir değer dönderilmektedir. Örnek 13. Satırdaki metotda yaş hesaplaması yapıldı ve 20 çıktı. Bu metoda geri gönderilir ve 11. Satırın son kısmında oraya 20 girer.
* 25-27 Satırlarda nesne üretilmektedir.
* 29-31. Satırlarda bu öğrencilerin bilgileri yazılmaktadır. P1.Intro() dediğimiz zaman 9. Satırdaki metoda gider ve this yerlerine p1 koyup işlemi geri gönderir.
* Nesne oluştururken var da kullanılabilir, Person nesnesi oluşturacaktır zaten.

**Metotlar Uygulama**

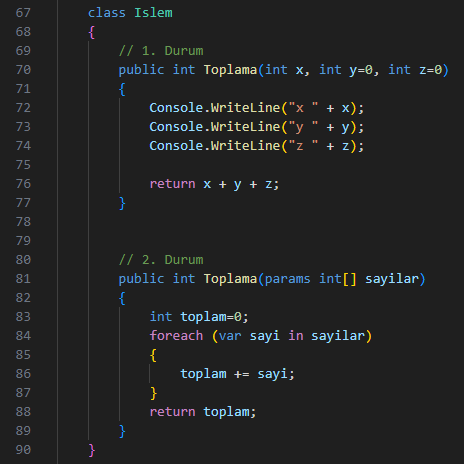




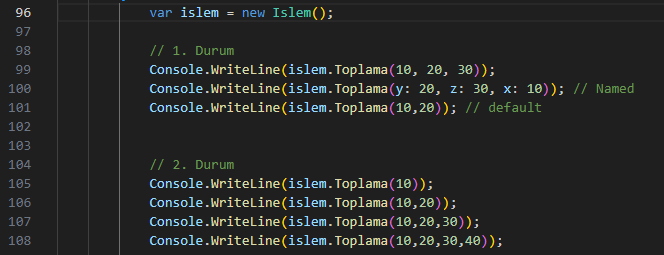


* 8-11. Satırlarda propertyler tanımlandı. (Araba ortak özellikleri)
* 13-62 arasında metotlar tanımlandı, 33. Satırda menü hazırlandı.
* 71-90 arasında opel ve mazda nesneleri oluşturulup metotlar çağrıldı.

**Metot Parametreleri**

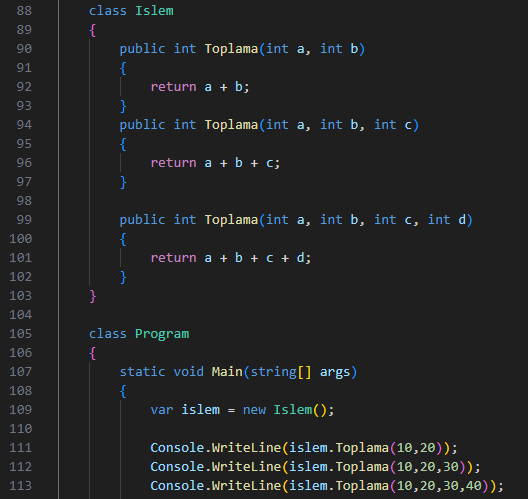


* İşlem isminde bir class var ve 70. Satırdaki gibi bir metot var. Bu metoda () içerisinde parametre gönderebiliriz. Yani bu metotları sonradan dışarıdan tanımlayabiliriz.
* 1. Durumda 3 sayı toplanırken, 2. Durumda istenilen sayida sayi toplama işlemi yapılabilmektedir.
* 70. Satırda int y=0 olarak tanımlama durumu y ye varsayılan değeri 0 verildi. Eğer kullanıcı y’ye değer vermezse 0 kabul edilir, verirse girmiş olduğu değer kabul edilir.



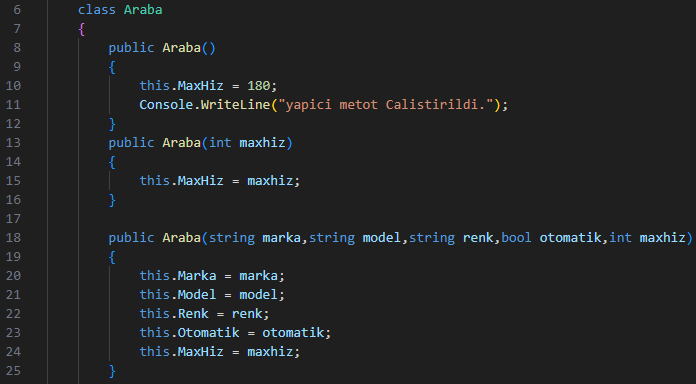
* Şimdi 1. Durumda normal olarak metot çağrıldığında örnek Toplama(10,20,30) sırası ile yukarıda tanımladığımız sırayla x,y,z değerlerini alır (99.satır). Eğer bu sıralamayı kendi isteğimizle yapmak istersek 100. Satırdaki gibi tanımlama yapmamız gerekir.
* 105-108 arasında tanımlanan işlemler ise classda 2. Durum kısmını kapsamaktadır. Yani istenilen sayının toplamı yapılabilir.

**Aşırı Yüklenmiş Metotlar**

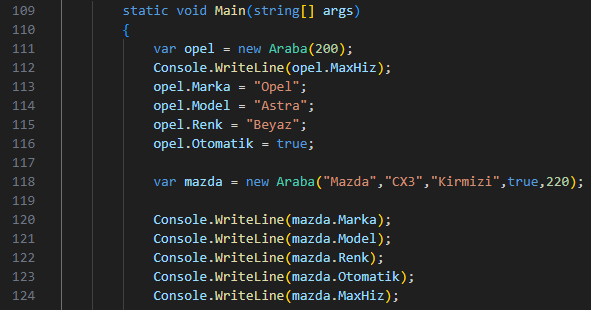


* Aynı isimde birden fazla metot normalde olamaz. Eğer bu kullanmak istersek aşırı yüklenmiş metot taktiği uygulanması gerekiyor.
* Bu taktik de ise metot isimleri aynı olabilir lakin parametreler farklı olması gerekmektedir. Buna aşırı yüklenmiş metotlar denir.

**Yapıcı Metotlar**

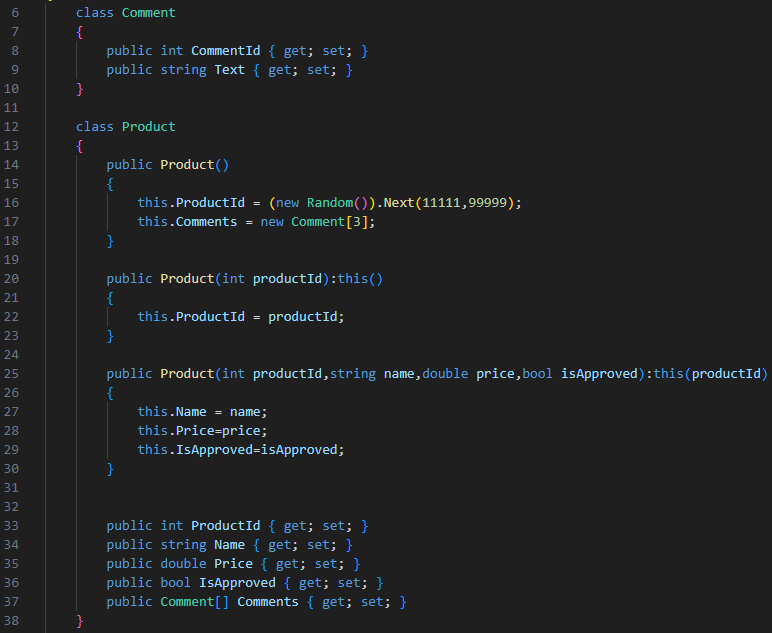


* Bu şekilde yapıcı metotlar oluşturulabilir. Yapıcı metotlar class ismi ile aynı isimde olur. Bu yapıcı metotlar nesne oluşturulurken direk çağrılır. Normal metotlar ise manuel çağrılması gerekir. Python daki \_\_init\_\_ yapısı olarak düşünülebilir.
* Tabi bu metotlar da aşırı yüklenmiş özelliği ile oluşturulabilir.



* 111. Satırdaki nesne tanımlaması 13. Satırdaki metot ile tanımlanmıştır. Max hız girilmiştir daha sonra diğer özellikler doldurulmuştur.
* 118. Satırdaki nesne ise 18. Satırdaki matot ile doldurulmuştur. Diğerine kıyasla bütün özellikler parametre olarak yazılmıştır. Lakin 20-24. Satırdaki gibi tanımlama yapılıyor olması gerekmektedir.

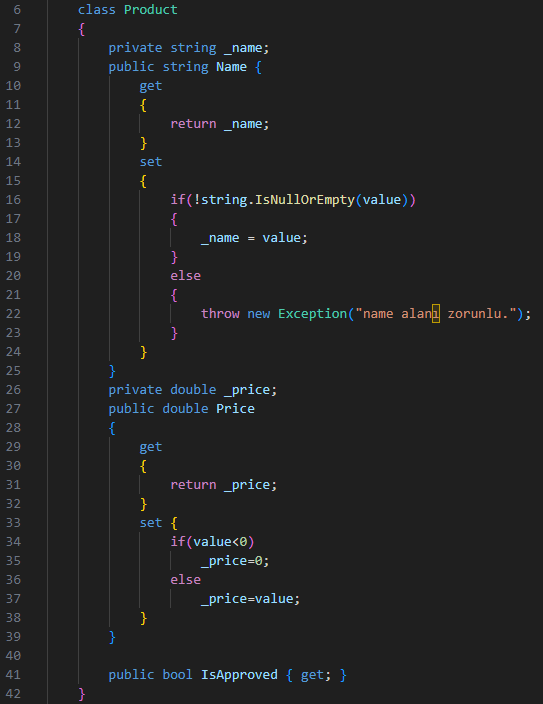
**Yapıcı Metotlar Uygulama**

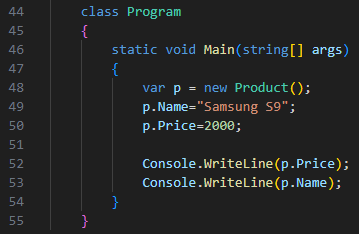




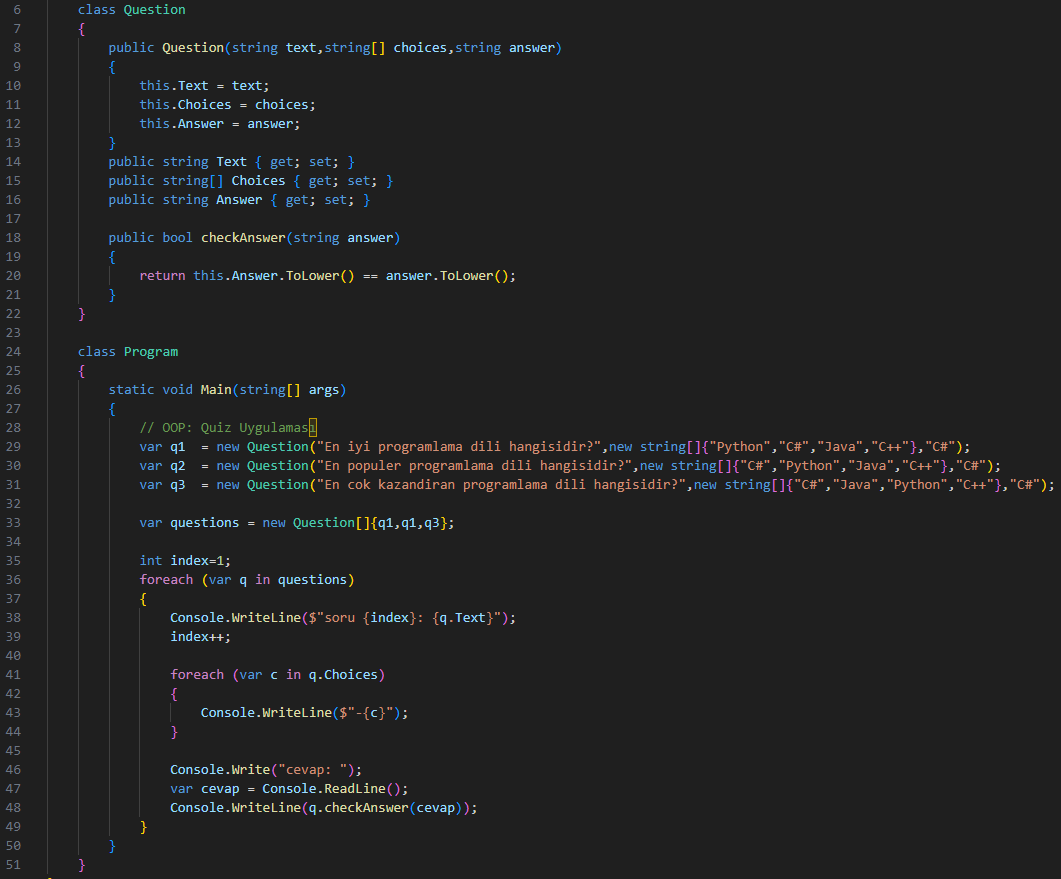
* Comment adındaki class yorumları, product adındaki class ürün bilgilerini tutmaktadır.
* Commnet de yorum id ve yorum yazısı bulunmaktadır.
* Product içerisinde 3 tane yapıcı metot yazdık, direk Product() şeklinde veya parametre göndererek nesne oluşturulabilir.
* 14. Satırdaki yapıcı metotda commnets listesi tanımlaması yapıldı yani yorumlar dizisi tanımlandı ve varsayılan 3 tane yorum olsun dedik. (Yorum sonra tanımlanacak.)
* 25 ve 20. Satırlardaki metot sonlarında .this yapısı bulunmaktadır. Bu metotlarda yorum için dizi tanımlaması yapılmadı (17. Satır). 20. Satırdaki .this diyerek 14. Metot çağrılmış oldu aslında. 25. Satırda ise direk 1.’yi çağıramadığımız için (productid random atıyor) 20. Satırdaki metot çağrılıp sonra 14. Satırdaki metot çağrılır.
* 33-37 arasında değişkenleri tanımlamasını yaptık.
* Program classı içerisinde main kodlarımız bulunmaktadır.
* 44. Satırda bir tane yorum tanımlandı. (max 3 tane tanımlayabilirim). Eğer kaç tane tanımlayacağınızı bilmediğiniz zaman ileride diziler yerine listeler kullanılacak.
* 47,59,76. Satırda yorum dizisine tanımladığımız yorumu atadık.

**Properties**



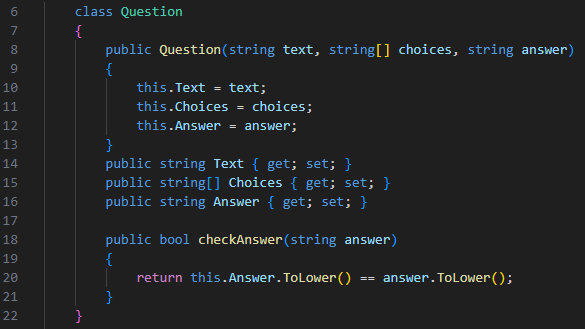


* 8. Satırda private tipinde bir değişken tanımlanmıştır. Bu değişkenler yanlızca ilgili class/metot içerisinde tanımlıdır. (\_değişken) şeklinde tanımlanan değişkenler dışarıdan erişilmesi engellenmektedir. Örnek üniversite classında akademisyen maaşlarını dışarıdan erişilmesi bu şekilde engellenir.
* 9. Satır ile name değişkenine dışarıdan erişilmesi için kod yazılmıştır. Get ile bu değişkeni getirebilirken, set ile bu değişken güncellenmektedir.
* 16. Satırda IsNullOrEmpty ile boş değer olması sorgulanmaktadır.
* 22. Satırda hata fırlatılmaktadır. İlerleyen derslerde tekrar konusu işlenecek. Şimdilik Consola hata yazılmış gibi düşünülebilir.
* 26-27. Satırlarda name değişkenine yapılan aynı işlemler yapıldı.
* 41. Satırda eğer bir değişken okunsun ama değiştirilmesin istiyorsanız get bırakılıp set silinmektedir.

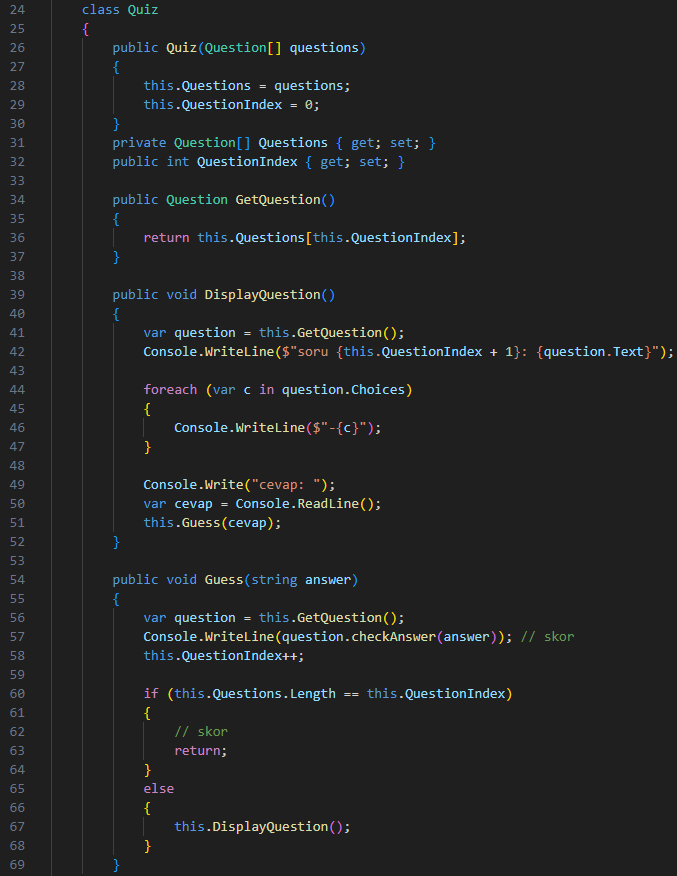
**Uygulama 1 (Soru (Question) Sınıfı)**

* Yukarıda bir quiz uygulaması yapılmıştır. Text: soru, Choices şıklar, Answer verilen cevaptır.
* 18. Satırda bir soruya verilen cevap kontrolü yapılmaktadır. Metoda gelen answer’in yazısını küçük harfe çevirip, cevabıda küçük harfe çevirip kontrol yapar.
* 29-31 sorular,şıklar,cevap tanımlanmıştır.
* 33. Satırda quiz dizisi içerisine sorular tanımlamıştır.
* 36. Satır ile consol uygulaması yapıldı. Soru geliyor, şıklar yazıyor daha sonra cevap veriliyor, cevaba göre True False dönmektedir.
* 33. Satırda q1,q2,q3 olmalıdır.

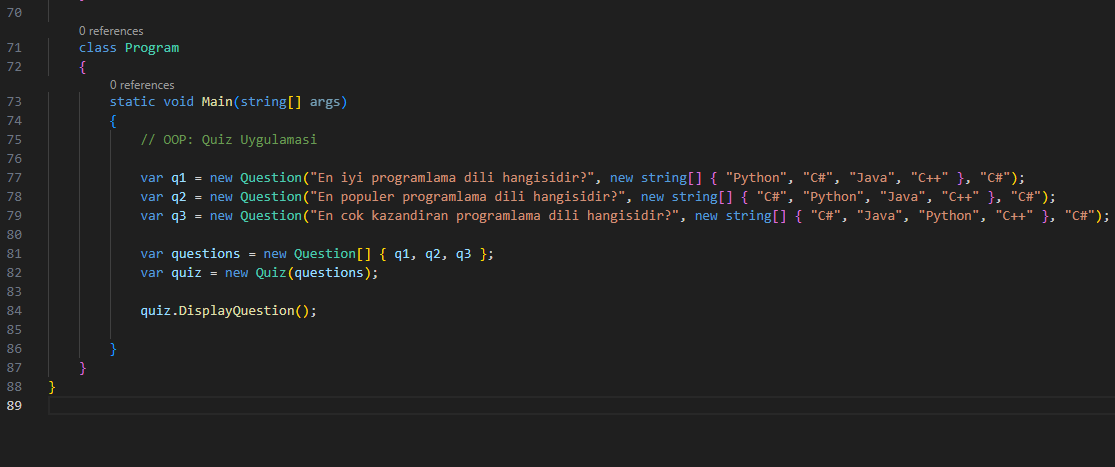
**Uygulama 2 (Quiz Sınıfı)**



* Soru classında bir değişiklik yapılmadı.
* Şimdi bir quiz sınıfı tanımlayalım ve soru değiştirme, cevap gibi işlevleri o class içerisinden tanımlayıp sadece class çağırma işlemi yapalim.

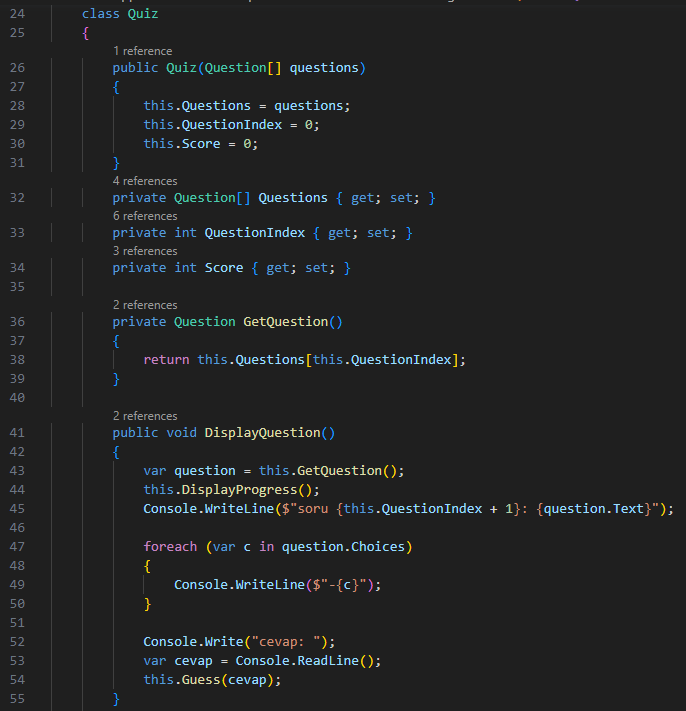


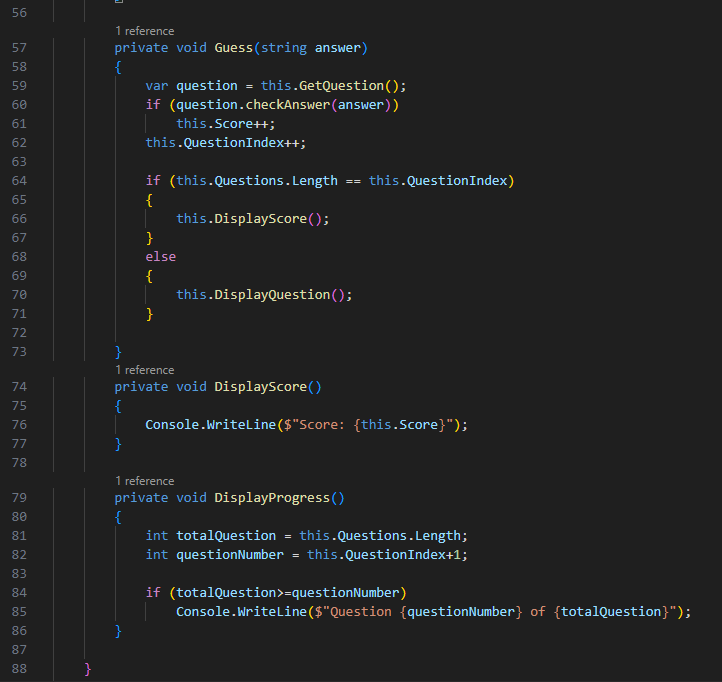
* Questions dizisi ile soruları, QuestionIndex ile soruların index numarası (1.2.3. soru gibi) tanımlandı. İndex numarasına göre soruların gösterim işlemini yapacağız. Rastgele soruların gösterilmesi istenirse .Random kullanabilir.
* 31. Satırda sorular private şeklinde tanımlandı.
* 34. Satırdaki metot ile bu soruları getirme işlemi yapılmaktadır. [] içerisinde index’e göre geldiğine dikkat edelim. Örnek [1] içerisinden gelecek soru aslında 2. Soru. [0] 1. Soruya takabül eder.
* 39. Satırdaki metot, sorular gösterme işlevi yapmaktadır. 41. Satır ile soru getirilmektedir. 42. Satırda soru yazdırılıp, 44. Satırda şıklar gelmektedir. 49-50. Satırda kullanıcıdan cevap alır ve Guess isminde metot’a(53.satır) cevabı göndermektedir.
* 53. Satırdaki Guess metodu, gelen cevabı kontrol edip ekrana yazdırır, sonraki soruya geçer. 55. Satır ile quiz’i getirdik, böylelikle 56. Satırda sorunun kontrolü yapılıp ekrana yazdırıldı (bir sonraki dersde skor hesaplanacak buraya değişken vercez.) 57. Satır ile sorunun index’ini arttırdık, 59. Satırdaki if ile eğerki quiz içerisinde soru yoksa skor yazdıracağız, eğer soru bitmemişse 66. Satır ile soru getirmeye devam eder.
* 66. Satırda ekranda gösterim fonksiyonu çağrılmaktadır. Aslında burada **Recursion (Özyineleme)** li fonksiyon kullanılmıştır. Anlaşılmadıysa internette araştırabilirsiniz.



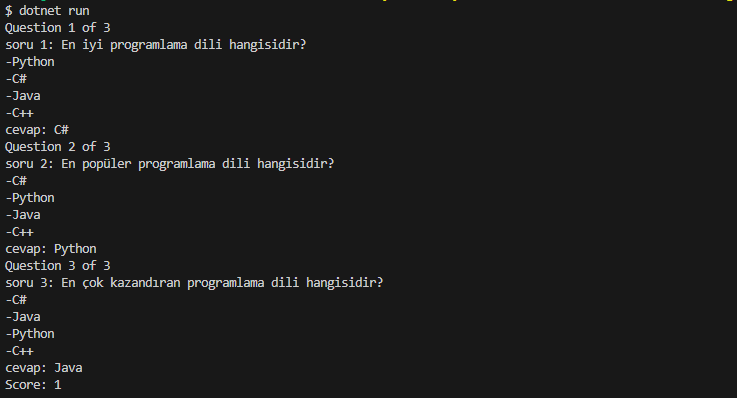
* 77-79. Satırlar sorular tanımlandı.
* 81. Satırda sorular dizisi oluşturuldu.
* 82. Satırda sorular dizisi Quiz Classına verildi.
* 84. Satırda soruları gösterecek metot çağrıldı. İşlem bu kadardı.

**Uygulama 3 (Skor Bilgilerinin Hazırlanması)**

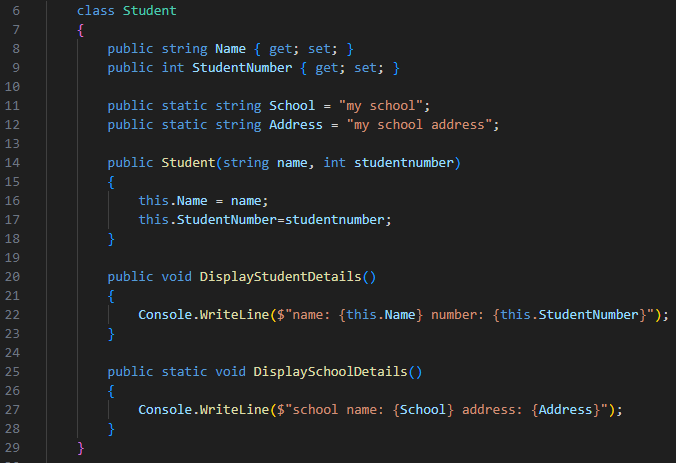




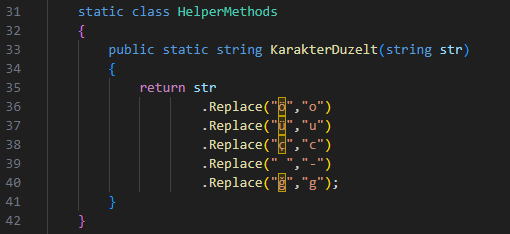
* 44. Satır eklendi, 79. Satırdaki metodu çağırmaktadır. Bu metot ile kullanıcı kaçıncı soruda olduğunu görebilmektedir.
* 30. Satırda ve 34. Satırda score değişkeni tanımlandı.
* 60. Satırda eğer cevap doğru verildiyse 61. Satırda skor 1 artmaktadır. (Değişmeden önce console.write ile doğru veya yanlış yazmaktaydı.)
* 64. Satır ile eğer quiz içerisindeki sorular bitmişse skoru yazdıracak metot çağrılmaktadır (66.satır).
* 74. Satırdaki metot ile scor bilgileri yazılmaktadır.
* 79. Satırdaki metot ile kullanıcı kaçıncı soruda olduğunu görebilmektedir.



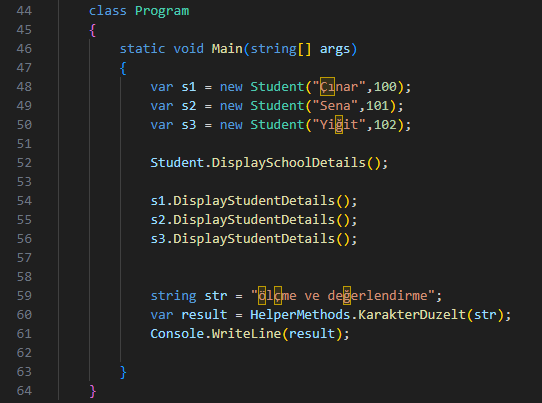
**Statik Members**



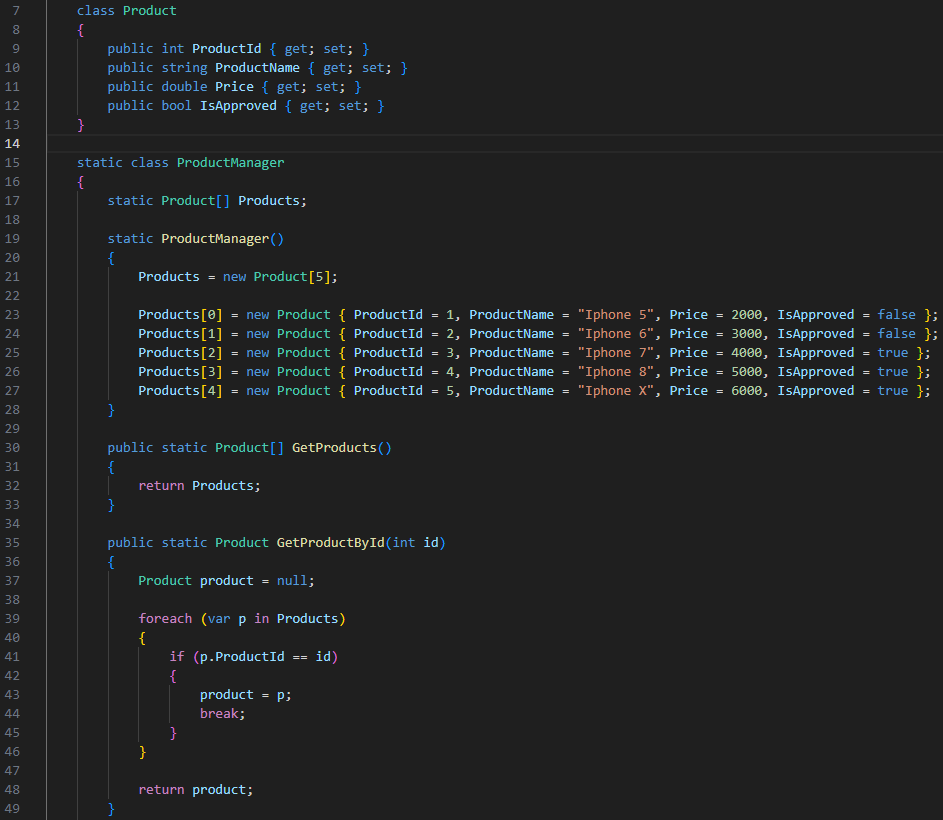
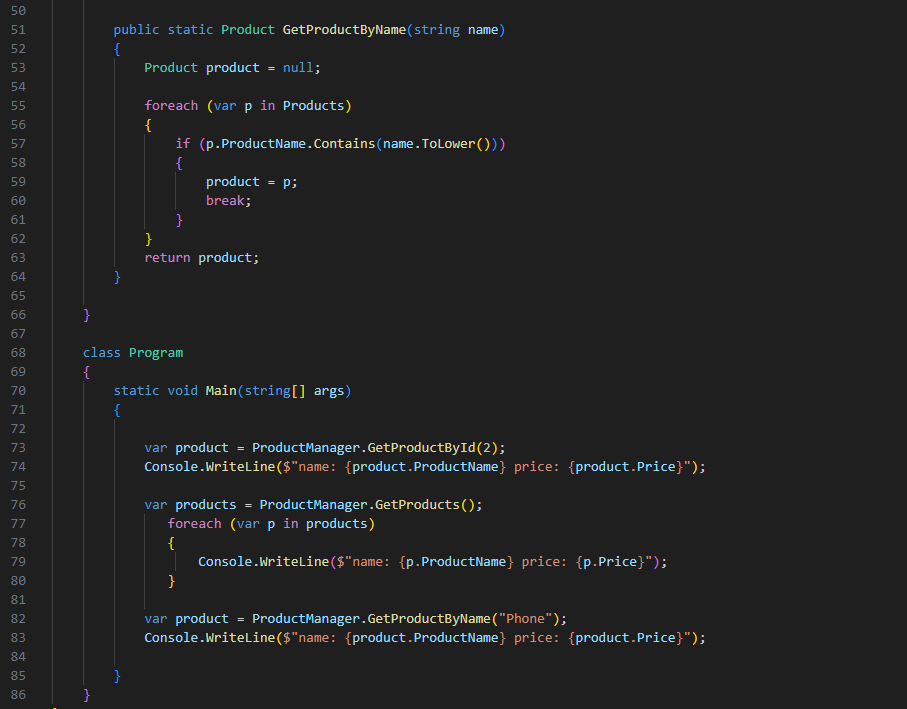
* 11. Ve 12. Satırda statik değişken tanımlandı. Bu değişkenler genel olarak nesne üretme noktasında kullanılmamaktadır. Örnek öğrencilerin isimleri, numaraları değişebilir lakin okul ismi adresi aynı kalmaktadır. İşte bu noktada kullanılmaktadır.
* 25. Satırdaki metotda statik bir metot tanımlanmıştır. 27. Satırda statik ile tanımladığımız değişkenler ile nesne tanımlanmadığı için this kullanılmamaktadır.
* Şimdi statik class oluşturalım.



* Yukarıda statik class tanımlanmıştır. Statik classlar içerisinde bir işlem yapılması için kullanılır, nesne üretilemez.

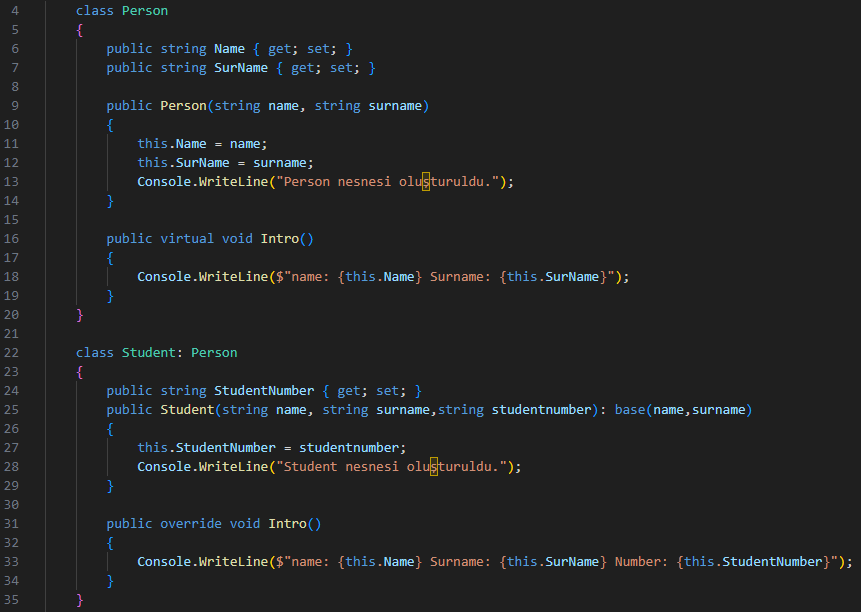


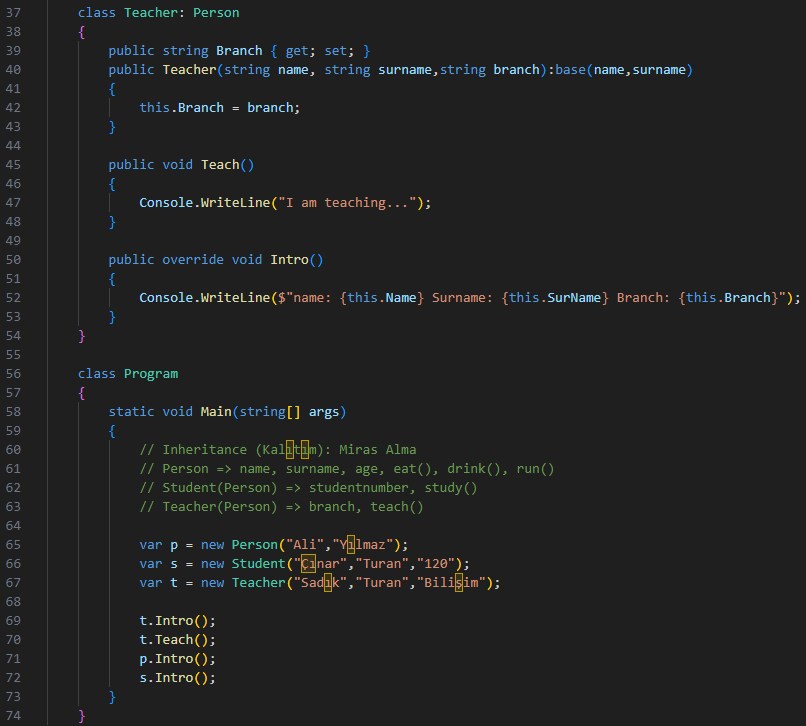
* 52. Satır Statik metot çağrılırken nesne üzerinden çağrılmaz. Örnek s1.DisplayStudentDetails demek yerine Student.DisplaySchoolDetails diyerek class ismi ile çağırdık.
* 59-61. Satırlardaki örnek statik class kullanımına örnektir.

**Statik Members Uygulama**

* Ürün ile uygulama yapılacaktır. ProductManager bir veri tabanı olarak düşünülebilir.
* 15. Satırda statik bir class tanımladık.
* 30. Satırdaki metot products listesini getirmektedir.
* 35. Satırdaki metot İd’ye göre ürün getirmektedir.
* 51. Satırdaki metot Name göre ürün getirmektedir. Lakin aynı isimde iki ürün varsa ilkini getirir.
* Statik class olduğu için veriler program içerisinde değil, class içerisinde tanımlandı. (Veri tabanı gibi)
* 73-74. Satırlarda id’ye göre arama yapıp yazdırdık.
* 76-80. Satırlarda bütün ürünleri ekranda yazdırdık.
* 82-83. Satırla isme göre arama yapıldı.

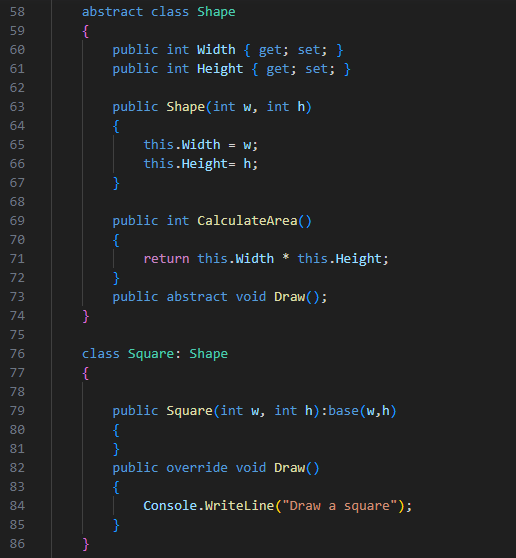
**Kalıtım**

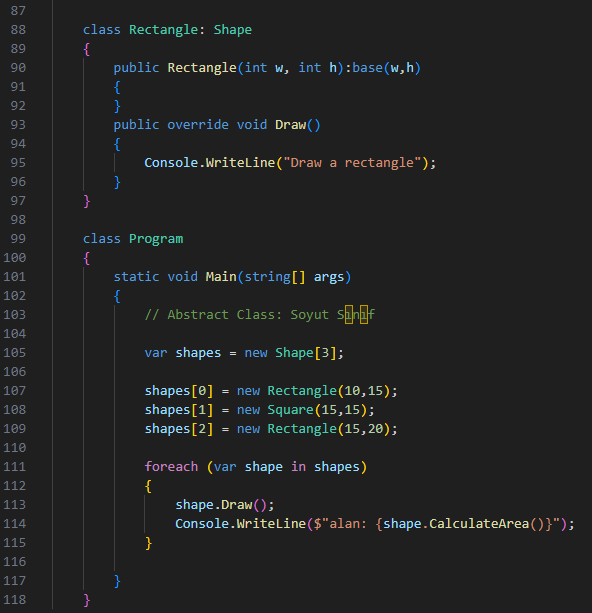




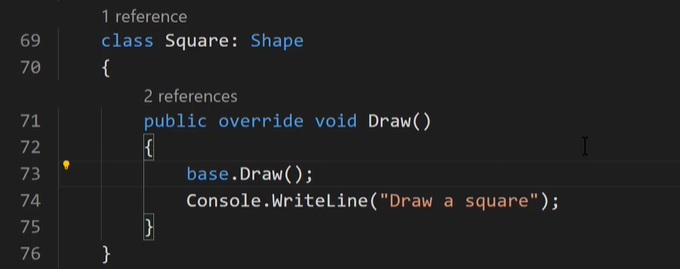
* Kalıtım özelliği ile örnek bir üniversitede kişi diye bir class tanımlanır ve her akademisyen,öğrenci,personelin isim soyismi vardır. Ortak özellikleri daha sonra oluşturulan öğrenci veya akademisyen sınıfında tekrar tanımlamaya gerek yoktur. Kalıtım yolu ile aktarım mümkündür.
* 4. Satırdaki Person classı bizim anasınıfımızdır, 16. Satırda Intro isminde bir metot bulunmaktadır. Bu metot alt classlarda tanımlandığında alt sınıftaki Intro ezmesi için virtual yazılır.
* 22. Satırda Student:Person ile kalıtım gerçekleşir, 25.. satırda base içerisine kalıtım yolu ile alınacak isim ve soyismi aldık, öğrencilerin extra öğrenci numarası olduğundan 24. Satırda öğrenci numarası tanımlandı.
* 31. Satırda override ile Intro metodu yukarıdaki virtual Intro metodunu ezer ve öğrenci isim soyismin yanında okul numarasını da yazar.
* Kalıtımın çalışma prensibi örneğimizdeki Person classı (Ana Class) ve Student (Child Class) bulunmaktadır. Student sınıfı person sınıfından kalıtım ile aldığı içi bir student nesnesi oluşturulduğunda ilk olarak person sınıfı çalıştırılır sonra student sınıfı çalıştırılır.

**Abstract Sınıflar**



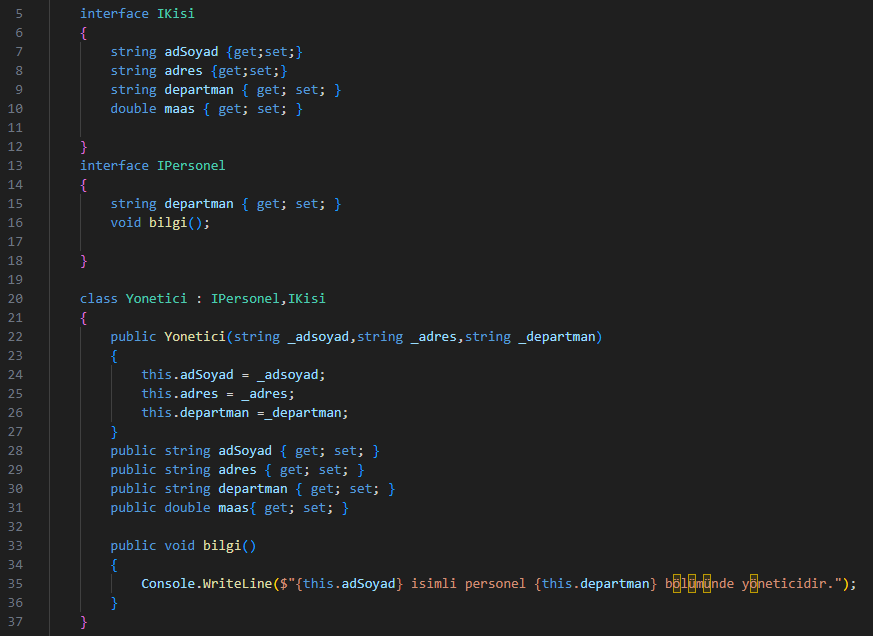


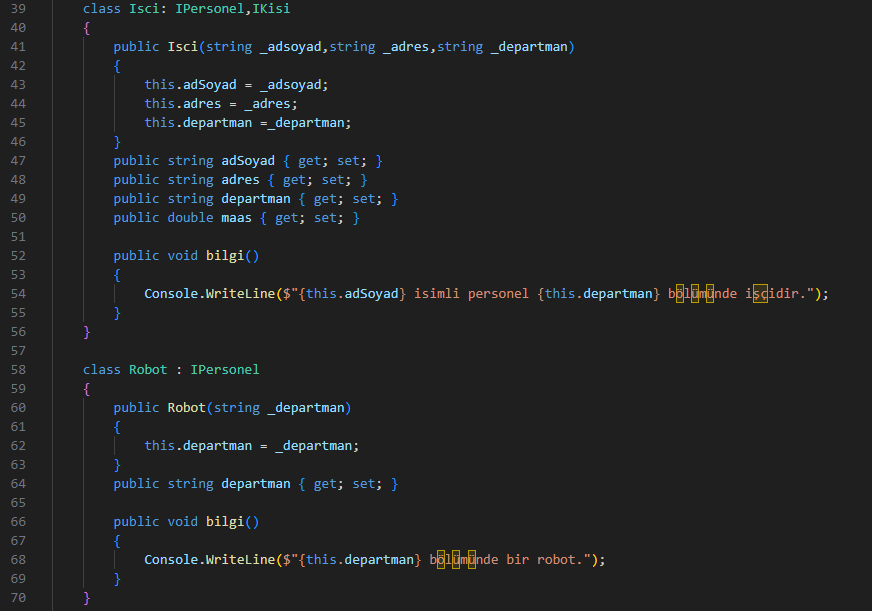
* Abstract sınıflar ve metotlar ile kalıtım yolu ile alan alt sınıflar için belirli özellikleri doldurmasını zorunlu hale getirebiliriz.
* 58. Satırda abstract class tanımlandı ve 73. Satırdaki Draw metodu abstract olarak tanımlandı. Bu yüzden 76. Ve 88. Satırdaki sınıflar Draw metodunu kullanmak zorundalar.
* Abstract sınıflar içerisinde normal metotlar oluşturulabilir lakin abstract sınıflar nesne oluşturamaz. Alt sınıflar kullanabilirler. Örnek 114. Satırda CalculateArea metodu kullanıldı.

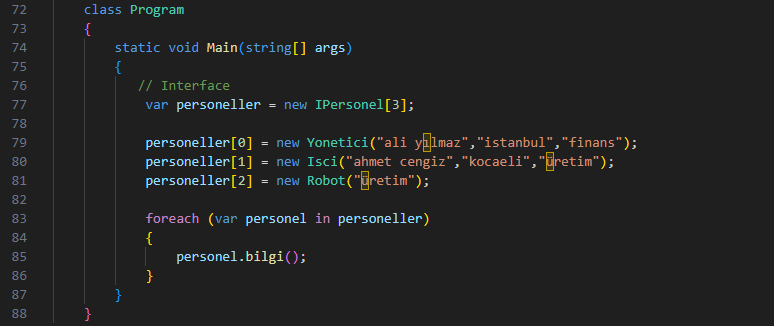


* Eğer üst sınıftaki bir metot çağrılmak istenirse, base ile çağırabilirsiniz. (73.satır).
* Ekstra bir bilgidir. Kodun orijinali yukarıdaki gibidir.

**Interface**

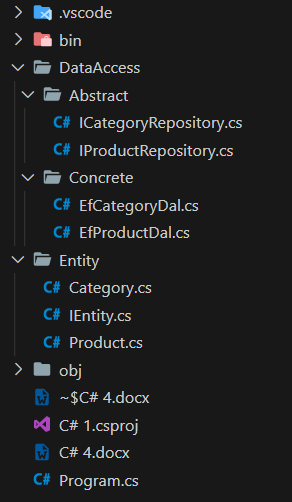




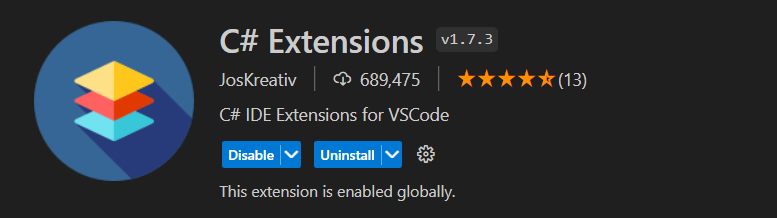


* İnterface bir yapı olarak düşünülebilir. Bir string yapısı gibi. Kullanıcılar bir nesne veya değişken tanımladıklarında, tip olarak seçili interface yapısını gösterirse interface içerisindeki yapıları zorunlu kılar.
* 5-13. Satırlarda iki tane interface tanımlanmıştır. “I” harfi ile başlaması zorunlu değil lakin yazılımcılar bu şekilde tanımların interface olduğunu anlar.
* Interface tanımlamalarında değişken tanımlarken (7-10. Satırlar) başında public vs yazılmaz.
* Kişi interface şirketteki insan personelleri temsil eder, Personel interface ise insan + robotları temsil etmektedir.
* 20. Satırda 2 tane interface kullanılmıştır. İnterface tanımlı değişkenlerde 28-31. Satırlardaki aynı değişkenler tanımlanmak zorundadır.

**Interface Uygulama 1 (Repository Pattern)**

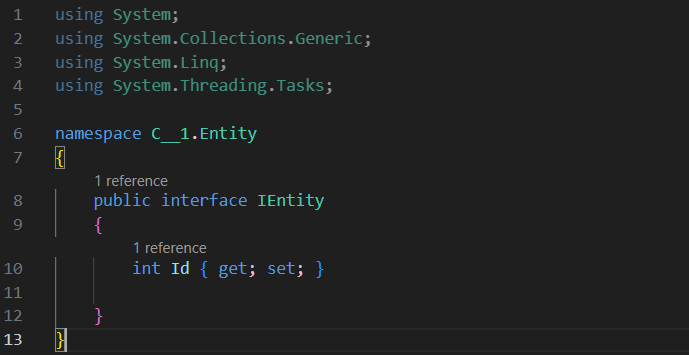


* Dosyalarımız şuanda bu şekilde oluştururken klasör isimlerini kendiniz oluşturun. Daha sonra verdiğim eklenti ile sağ tık yaptığınıca C# class veya ınterface dosyası oluşturun.
* Abstract içerisindeki dosyalar ve Entity içerisindeki IEntity.cs dosyası interface dosyasıdır.



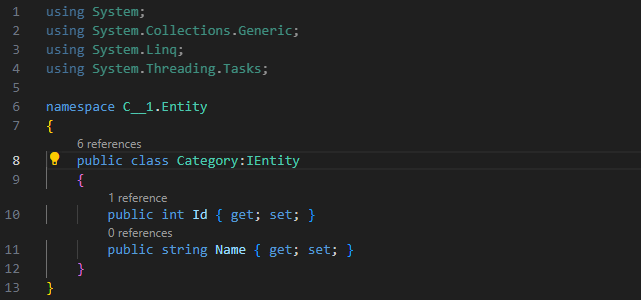
* Kurulacak eklenti yukarıdaki eklentidir.

**Entity/IEntitiy.cs**



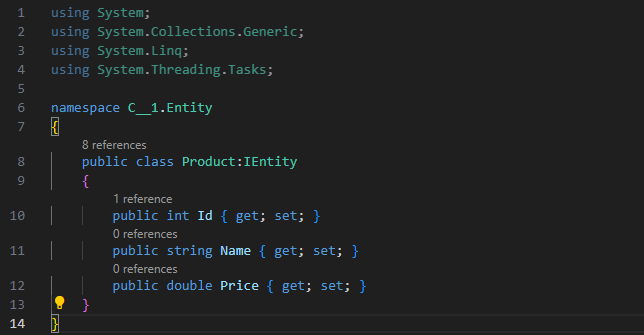
* Bu dosya ile category ve product dosyasının temel yapısı id tanımlandı. (Interface)
* Entity klasörü içerisindeki dosyalar ile veri taşıncak. Örnek product içerisinde sınıf tanımlanıp ilerleyen zamanda nesne oluşturup paketleyip göndercez. Burada nesnelerin ana tipleri.

**Entity/Category.cs**



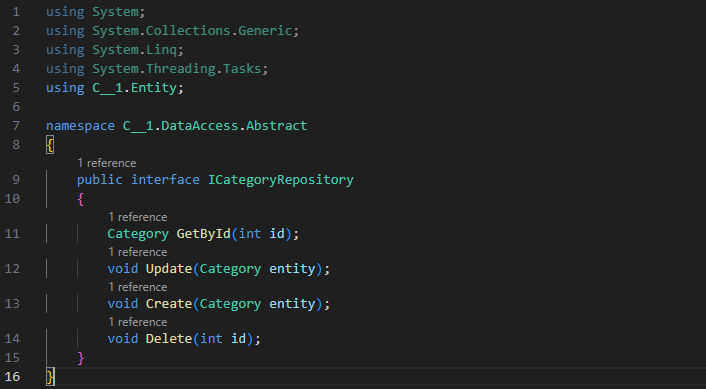
* Bu dosya ile Category classının ayarlamasını yaptık.

**Entity/Product.cs**



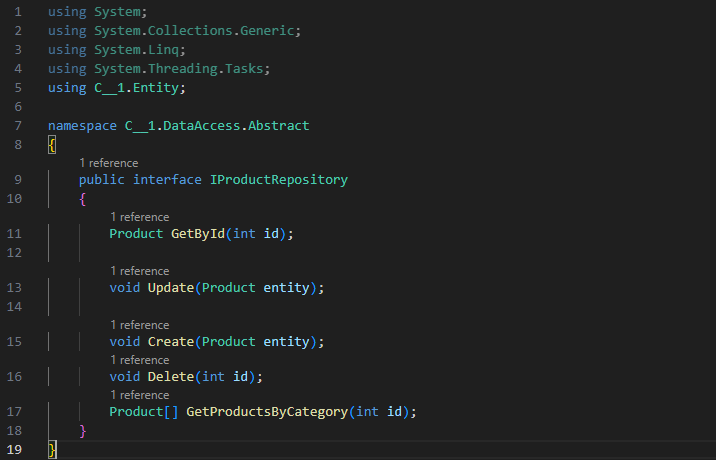
* Bu dosya ile product classının ayarlamasını yaptık.

**DataAccess/Abstract/ICategoryRepository.cs**



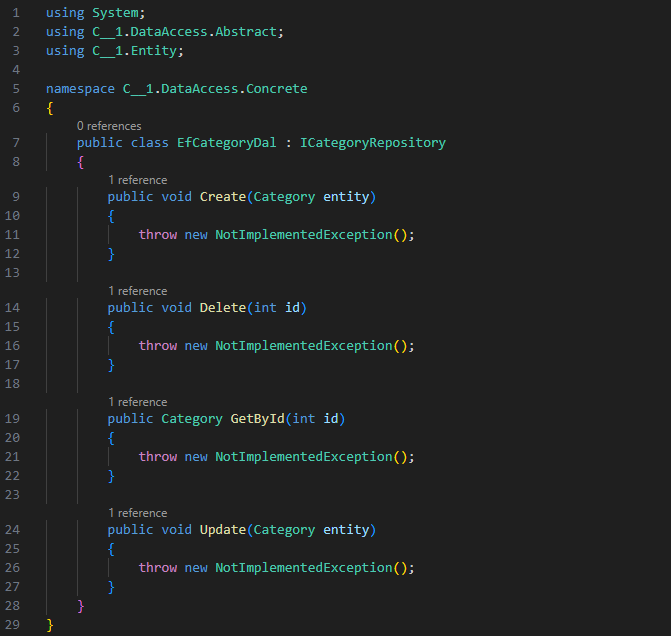
* DataAccess içerisinde veri tabanından veri çekme, gönderme gibi pek çok işlem bu klasör içerisinde yer alacaktır.
* Bu interface, **Category** **metotlarının yapıları** tanımlanmıştır.

**DataAccess/Abstract/IProductRepository.cs**



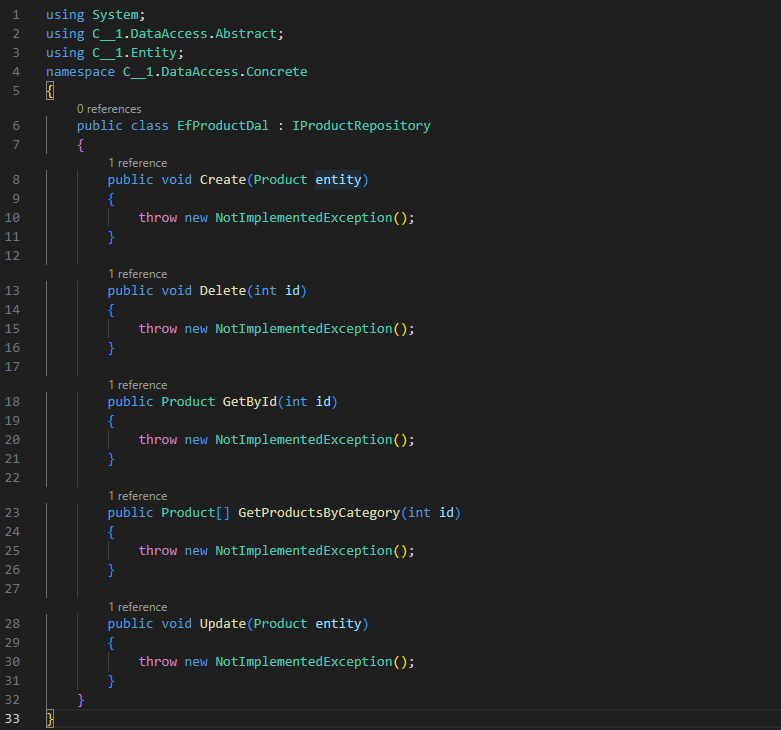
* Bu interface, **Product** **metotlarının yapıları** tanımlanmıştır.

**DataAccess/Concrete/EfCategoryDal.cs**



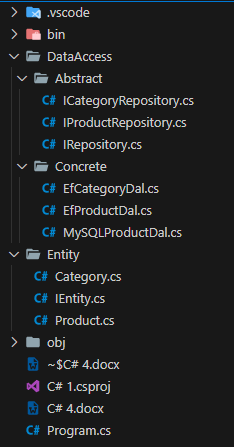
* Buraya interface de tanımlanan matotların içerisi doldurulmaktadır. (Category)

**DataAccess/Concrete/EfProductDal.cs**



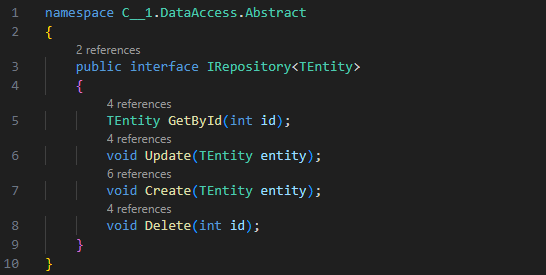
* Product metotları bu bölüm içerisinde doldurulmaktadır.
* Uygulama olduğu için genel yapıyı anlamaya çalışalım. Şuan veri tabanı ile bir çalışma işlemi söz konusu değildir. Sadece genel şema hazırlanmaktadır.

**Generic Interface Uygulama 2 (Repository Pattern)**



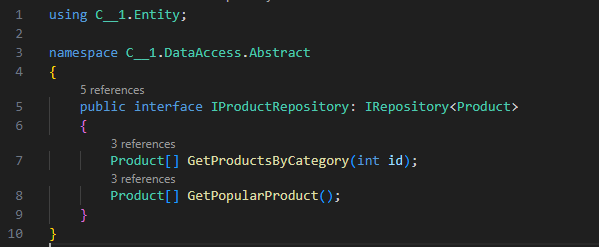
* Abstract / IRepository.cs ve Entity/MySQLProductdal.cs eklendi.

**DataAccess/Abstract/IRepository.cs**

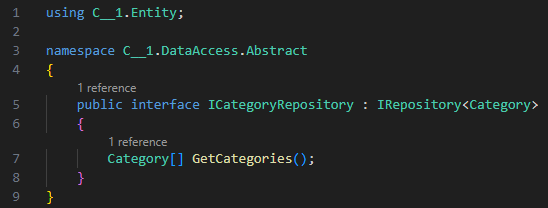


* IProductRepository ve ICategoryRepository içerisindeki ortak metotları buraya aldık. 3. Satırdaki TEntity yerine yazmamız yeterli. 5,6,7. Satırda TEntity yerine normalde Category veya Product yazmaktaydı.

**DataAccess/Abstract/IProductRepository.cs**

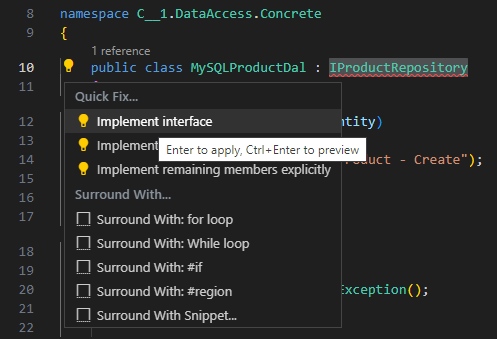


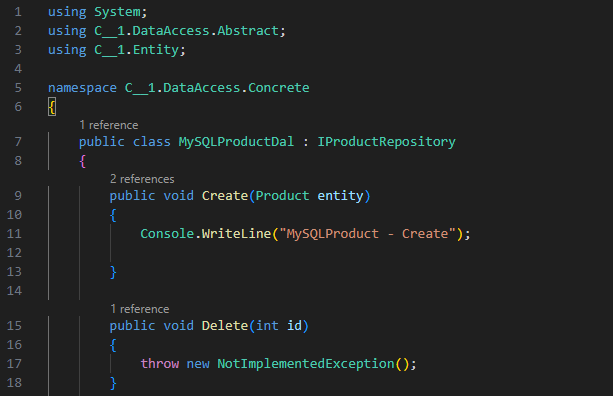
**DataAccess/Abstract/ICategoryRepository.cs**



* IReposity dosyası içerisinde tanımladığımız metotları tekrar yazmak zorunda kalmadık. 5. Satırdaki gibi <> içerisine Category veya Product yerleştirilmesi yeterli. Yanlızca ekstra metot yazılır.

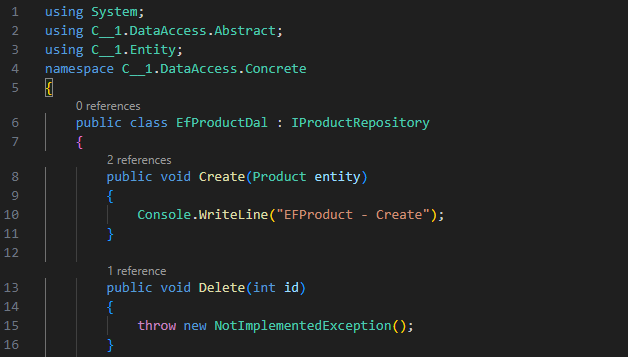
**DataAccess/Concrete/MySQLProductDal.cs**





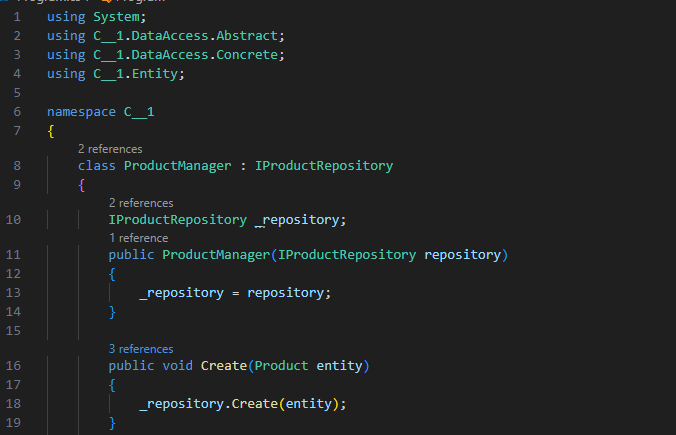
* 7. Satırdaki :IProductRepository üzerine çift tıklayıp, ampule tıkladığınızda ImpLement interface diyinde otomatik metotlar yerleşecektir.
* 9. Satırdaki metot üzerinde değişiklik yapıldı. (Create).

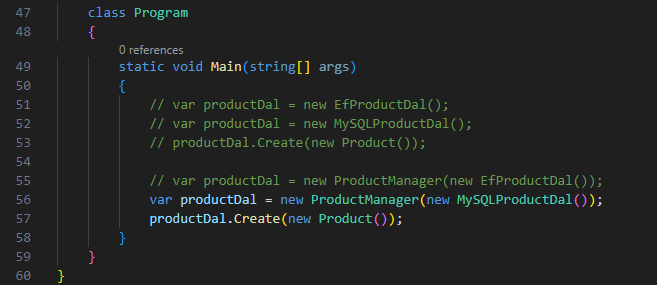
**DataAccess/Concrete/EfProductDal.cs**



* 8. Satırdaki metot değiştirilecek. (Create).

**Program.cs**





* 8. Satırdaki class’ı tanımlamadan 50-53. Satırlardaki gibi kullanılabilir. MySQL teknolojisi veya Ef teknolojisi kullanılabilir. Ki ileriki derslerde daha anlaşılır olacaktır.
* 8.satırda Class ismini yazdıktan sonra IProductReposity ekleyip implement edin. Daha sonra 10-19. Satırları düzenleyin, 55-57. Satırlardaki gibi kullanabilirsiniz