**1) ALGORİTMA**

🡪 Algoritma bir problemin çözümü için uygulanması ya da takip edilmesi gereken yönergeler olarak tanımlanabilir. Bir dizi işlemi kesin olarak tanımlayan bir dizi kuraldır.

🡪 Karşılaşılan herhangi bir durumda kararsız kalmadan en kısa yoldan ve en az maliyetle sonuca ulaşılmasını amaç edinmektedir.

Örnek:

**Çay Hazırlama Algoritması**

1. Çaydanlığın altına yeteri kadar su koyun.
2. Ocağın altını yakın.
3. Su kaynayana kadar bekleyin.
4. Su kaynadıysa çaydanlığın üstüne 3 kaşık çay ekleyelim.
5. Kaynar suyu ekleyelim.
6. Ocağın altını kısalım.
7. 15 dakika demlenmesini bekleyelim.
8. Çayı servis edelim.

🡪 Algoritmalar sonlu adımlar kümesinden oluşur. Her bir algoritmanın bir başlangıç ve bir bitiş noktası vardır.

🡪 Algoritmalar teknik kavramlar içermemelidir. Kullanılan ifadeler sade ve anlaşılır olmalıdır.

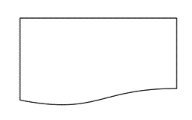
🡪 Eğer bir algoritma çıktı olarak geriye değer döndürüyorsa bir **fonksiyon** olarak; void olarak tanımlanmış ise yani geriye değer döndürmüyorsa **prosedür** olarak ifade edilirler.

**AKIŞ ŞEMALARI**

🡪 Diyagramın en üstünde başlatma sembolü, en altında ise durdurma sembolü yer almalıdır.

🡪 Dikdörtgen, diyagramda herhangi bir işlemi ifade etmek için kullanılır. Özellikle değişken tanımlama, değişkene değer atama ya da hesaplama yaparken kullanılır.

🡪 Ekrana çıktı vermek için aşağıdaki şekil kullanılır;

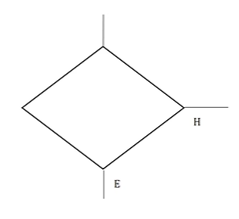


🡪 Kullanıcıdan veri girişi beklendiğinde aşağıdaki şekil kullanılır;

dikdörtgen, taslak, metin, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

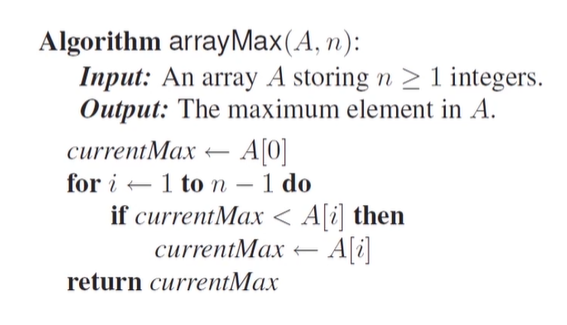
🡪Duruma göre akışın belirlenmesini sağlamak amacıyla aşağıdaki şekil kullanılır;



**KABA KOD**

🡪 Bir kaba kod yazarken bu kodu makinelerin değil, insanların okuyacağı unutulmamalıdır. Bu nedenle anlaşılırlık yüksek seviyede tutulmalı, teknik detaylardan kaçınılmalıdır.

Örnek:

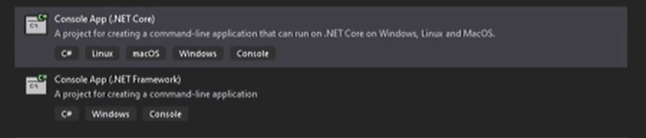


**2) TEMEL PROGRAMLAMA**

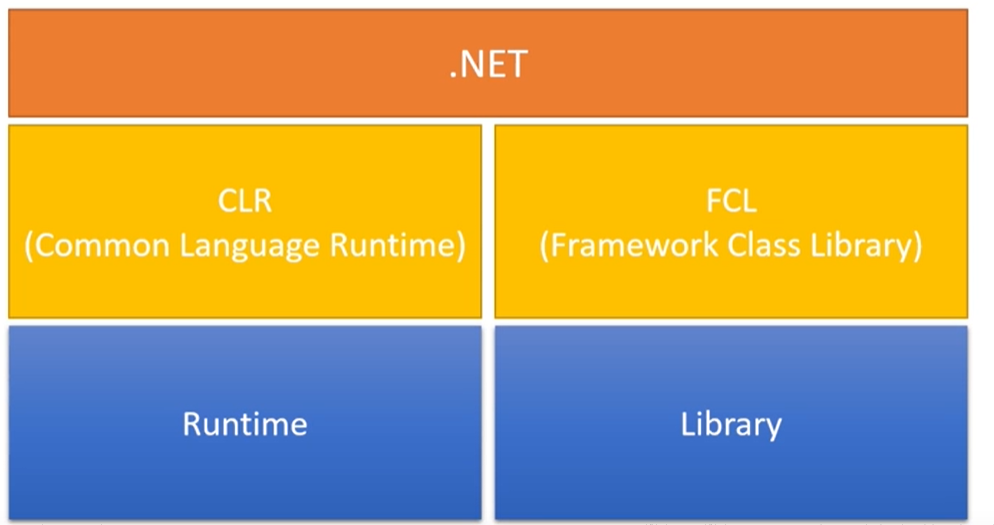
🡪 IDE = Integrated development environment (Tümleşik geliştirme ortamı)

🡪 .NET Framework = Sadece Windows uyumlu programlar yazılır.

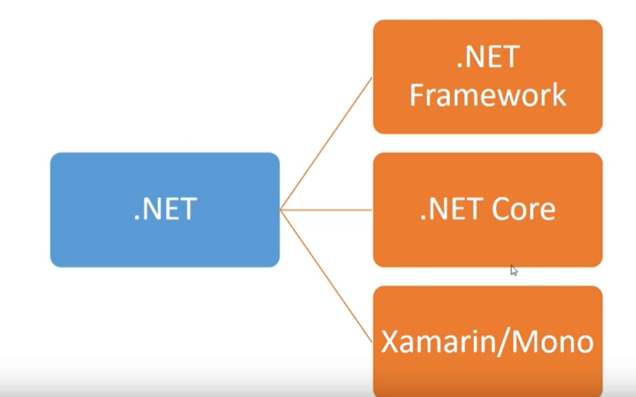
🡪 .NET Core = Windows, Mac ve Linux uyumlu programlar yazılır. Açık kaynak kodludur.



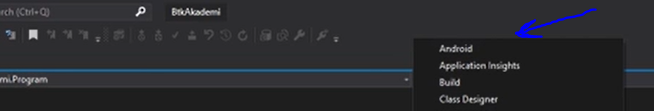
🡪 Sol taraf .NET Framework, sağ taraf .NET Core



🡪 Xamarin/Mono android ve ios platformlar için kullanılır.



🡪 Visual Studio’da yukarıya eklemek istediğimiz araçları(toolları), yukarıdaki bara sağ tıklayıp seçebiliriz.

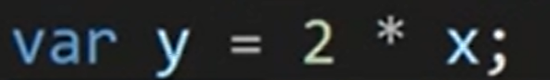


Soru: Neden framework’lere ihtiyaç duyarız?

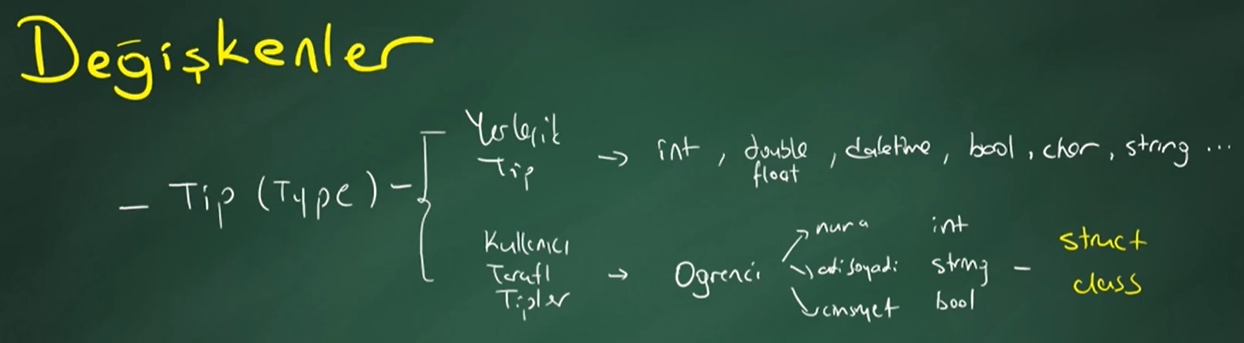
Cevap: Bir geliştiricinin ihtiyaç duyduğu veya duyabileceği pek çok yapının hazır olarak geliştiricinin kullanımına sunulmasını sağlıyor.

🡪 Derlenen kodlar .dll veya .exe uzantısı ile saklanırlar.

🡪 var anahtar kelimesi o değişkenin tipinin olmadığını belirtir. Bu şekilde tanımlama yapmaya implict(örtülü) tanımlama denir. Örneğin aşağıdaki örnekte x değişkeninin tipi int ise y’nin de tipi int olacaktır. Eğer x değişkeninin tipi double ise y’nin de tipi double olacaktır.



🡪 Değişken tipleri 2 ye ayrılır.



**Refactoring:** Kod üzerinde yapılan iyileştirmelerdir. Örnek olarak bir kod bloğunu method’a çevirmek veya bazı kodları başka dosyalara taşımak verilebilir.

🡪 **\a** bir uyarı sesi çıkarır.

🡪 **Verbatim(@),** backslash(\) kullanmadan stringleri yazmamızı sağlar. Örneğin aşağıda hem backslash hem de verbatim kullanılarak oluşturulan bir string örneği var.

* Console.WriteLine(@”c:\users\ridvanks\appdata”)
* Console.WriteLine(“c:\\users\\ridvanks\\appdata”)

İkiside aynı çıktıyı verir.

🡪 String concatanate, birden fazla stringi birleştirmektir.

* string isim = “Rıdvan”

string metin = “Merhaba benim adım ” + isim + “ ” + “Karasubaşı”;

Çıktı => Merhaba benim adım Rıdvan Karasubaşı

🡪 **String interpolation**, birden fazla tekil değeri bir sabit değere atamaya denir. Yapısı verbatim’e benzer fakat interpolation da dolar işareti($) kullanılır. Aşağıdaki örnek birinci ve ikinci değişkenlerini birleştiriyor ve yanına bir metin daha ekliyor. Tekil değerleri yazarken süslü parantez kullanılır.

metin, multimedya yazılımı, yazılım, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡪 Verbatim tipindeki sabitleri interpolation şeklinde yazabilmek için ilk önce interpolation işareti olan $ işareti eklenir ardından yanına verbatim işareti olan @ işareti eklenir ve içerisine metin yazılır.

metin, yazılım, multimedya yazılımı, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡪 Casting tip dönüşümü demektir.

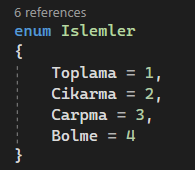
* int sayi1 = 35;

int sayi2 = 34;

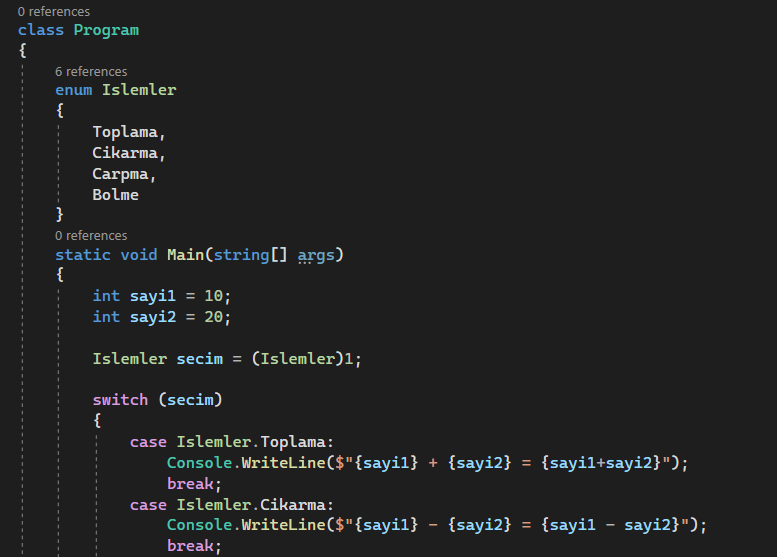
float sonuç = (float)sayi1 / sayi2;

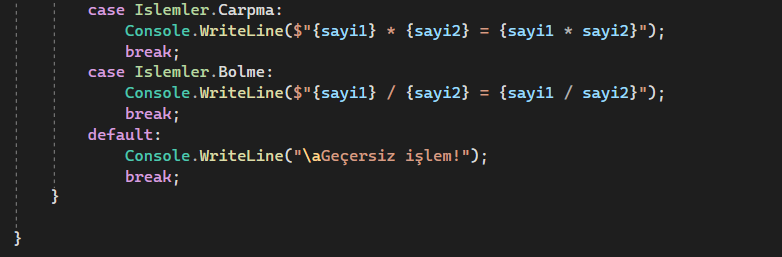
ifadesinde sayi1 değişkeninin tipini float’a çeviriyoruz.

🡪 Enum yapısı aslında bir liste oluşturuyor gibi düşünebiliriz. 1 değeri string olduğu için aşağıdaki kodlarda Casting yaptık ve onu “Islemler” enumunun 1.elemanına çevirdik. Enum’da indeksler 0 dan başlar. Fakat istersek bu durumu aşağıdaki işlemi yaparak değiştirebiliriz.



Enum Kodları:





🡪 Aşağıdaki ifadede 0 değeri sayac değerini tutar 5 değeri ise sayac değeri için ayrılacak alanı belirtir. Eğer ayrılacak alan pozitif ise sayac değerlerini sağa yaslar, negatif ise sola yaslar.



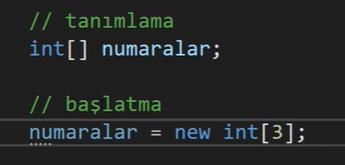
🡪 Burada 5 değeri pozitif olduğu için sayac değerleri sağa yaslandı.



🡪 Aşağıdaki örnekte ise bir formatlama var. 0’dan sonra kaç tane sayı geleceğini aşağıdaki kod ile yapıyoruz. Kaç tane “#” işareti varsa o kadar sayı 0’dan sonra gelir.



🡪 Diziler sabit boyutludur. Dizileri tanımlarken boyutlarını da belirtmeliyiz. Diziler referans tiplidir. Referans tipli demek bir ifadeyi tanımladıktan sonra onu başlatmamız gerektiği anlamına gelir. Referans tipli ifadeleri başlatmak için ise “new” anahtar kelimesine ihtiyaç duyarız.



🡪 Dizilerde aynı satırda hem tanımlama hem başlatma hem de atama yapabiliriz.



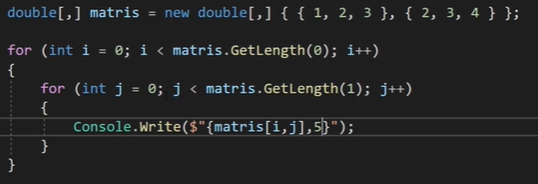
🡪 Eğer atamayı tanımlama adımında yapacaksak “new” anahtar kelimesini kullanmamıza gerek yok.



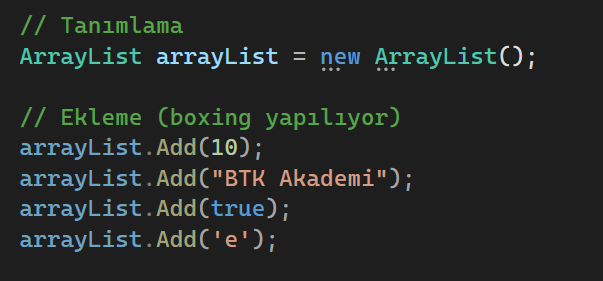
🡪 2 boyutlu diziler aşağıdaki gibi tanımlanır.



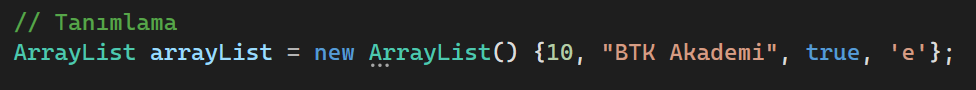
🡪 Aşağıdaki örnekte satırın uzunluğunu almak için “matris.GetLength(0)”, sütunun uzunluğunu almak için ise “matris.GetLength(1)” ifadeleri kullanılır.



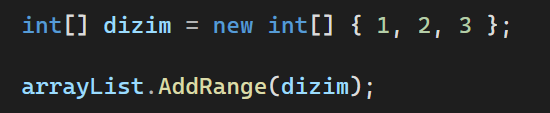
🡪 ArrayList’ler “using System.Collections” kütüphanesi ile gelir. ArrayList’te farklı türdeki değerleri tutabiliriz. Çünkü ArrayList’lerde “Boxing” vardır. Boxing kutulama demektir. ArrayList’e eklenilen her değer bir obje olarak tutulur. Bu sayede farklı türlerdeki değerleri aynı ArrayList’te tutabiliriz.



🡪 Dizilerde olduğu gibi tanımlama yaparken atama yapabiliyoruz.



🡪 Eğer bir ArrayList’e bir dizi atamak istiyorsak Add fonksiyonunu değil de “AddRange” methodunu kullanmalıyız.

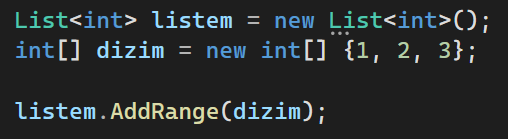


🡪 ArrayList’ten bir eleman silmek için “Remove” methodu kullanılır. Remove methodu içerisine değeri alır. “RemoveAt” methodu ise içerisine indeks değeri alır. “RemoveRange” methodu ise içerisine 2 parametre alır. İlk parametre hangi indeksten silinmeye başlanılacağı, 2.parametre ise kaç tane eleman silineceği. Aşağıdaki örnekte 1.satırda 10 değeri silinirken 2.satırda 1.indeksteki değer silinir, 3.satırda ise 3.indeksten başlayarak 3-4-5. indekslerdeki elemanları siler.

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡪 Liste methodları parantez içerisindeki türe göre çalışır. Listelerin tipi, aynı dizilerdeki gibi tanımlanırken belirtilmelidir. Bu sayede tip güvenliğini sağlamış oluyoruz. Biz “int” tutan bir listeye “string” değer ekleyemeyiz. ArrayList’ler de olduğu gibi “AddRange” methodu ile başka bir diziyi ekleyebiliriz. Burada dikkat edilmesi gereken bir nokta var. Eklenecek dizi ile listemizim türü aynı olmalıdır.



🡪 Listelerde araya ekleme yapmak için “Insert” methodu kullanılır. Bu method 2 tane parametre alır. İlki hangi indekse ekleneceği. İkincisi ise hangi elemanın ekleneceği. Araya birden fazla eleman eklemek istiyorsak “InsertRange” methodu kullanılır. Bu method da 2 tane parametre alır. İlki hani indeksten başlayarak ekleme yapılacağı. İkincisi ise hangi elemanların ekleneceği. İkinci parametreye bir dizi verebiliriz. Aşağıdaki örnekte 1.satırda 2.indekse 4 değeri atanır. 2.satırda 5.indeksten başlayarak 4,5 ve 6 değerleri sırası ile eklenir. Yani bu durumda 5.indekse 4 değeri, 6.indekse 5 değeri ve 7.indekse 6 değeri atanır. Bu arada atanır derken o indeksteki değerler silinmez. Onlar da kaç eleman geldiyse o kadar kaydırılır.

yazı tipi, metin, ekran görüntüsü, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

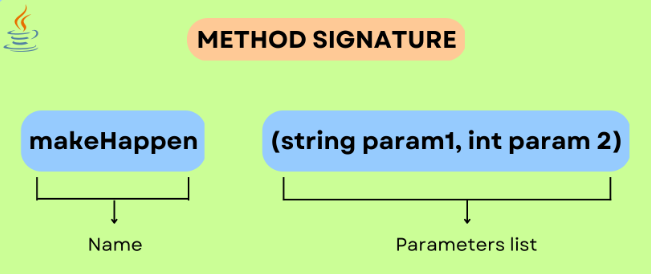
🡪 Eğer listeden bir eleman silmek istersek ArrayList’lerde olduğu gibi “Remove” methodlarını kullanabiliriz. Eğer bir değerin indeksini bilmiyorsak “IndexOf” methodunu kullanarak o değerin indeksine ulaşabiliriz.

metin, yazı tipi, ekran görüntüsü, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡪 Listelerin değerleri Heap’de, adresleri Stack’de tutulur.

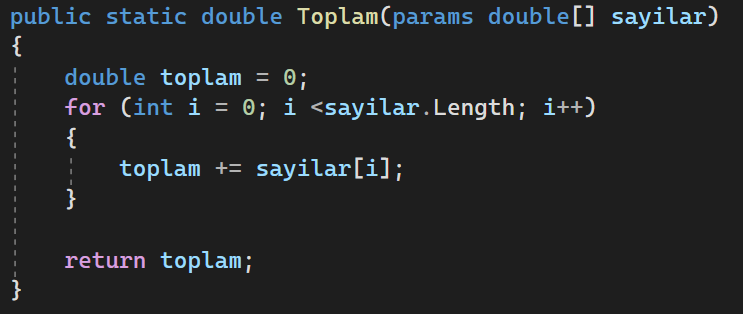
🡪 Methodun adının ve parametrelerinin olduğu kısma method imzası denir.



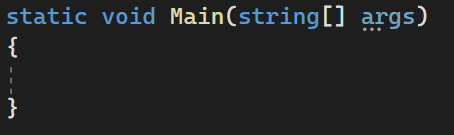
🡪 Kısa if aşağıdaki gibi yazılır. x eğer y den büyükse x dönecek, küçükse y dönecek.



🡪 Methodumuzu dinamik hale getirmek istiyorsak aşağıdaki gibi parametre göndermeliyiz. Bu sayede method istediği kadar parametre alabilir.



🡪 Eğer bir methodun referans tipi(public, private, void,) belirtilmezse default olarak “internal” olur.



🡪 Methodlarda parametrelerin değerlerini default olarak atayabiliyoruz. Bu sayede eğer kullanıcı parametre olarak bir değer göndermeyi unutursa bizim default olarak girdiğimiz değer kullanılır.

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

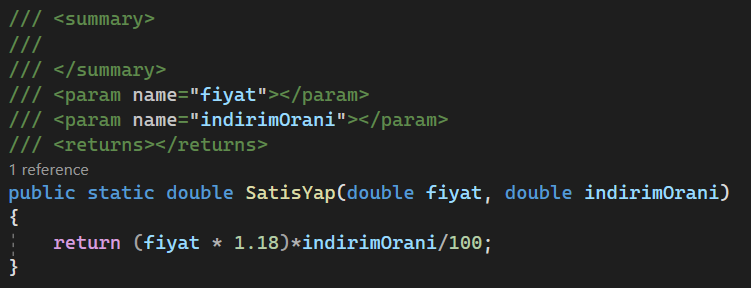
Açıklama otomatik olarak oluşturuldu

🡪 Method overloading, methodun sadece parametrelerini değiştirerek gönderilen parametre sayısına göre davranmasını sağlar.

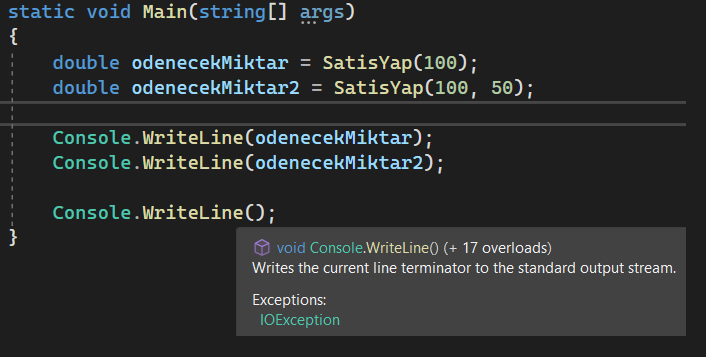
metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡪 Eğer Visual Studio’da bir methodun üzerine **3** tane “/” işareti koyarsak methodu özetlememiz için otomatik olarak bize bir yorum satırı şablonu açar. Bu sayede methodumuzun açıklamasını yazabiliriz. Bu, methodumuzu kullanacak kişiye bilgi vermemizi sağlar.

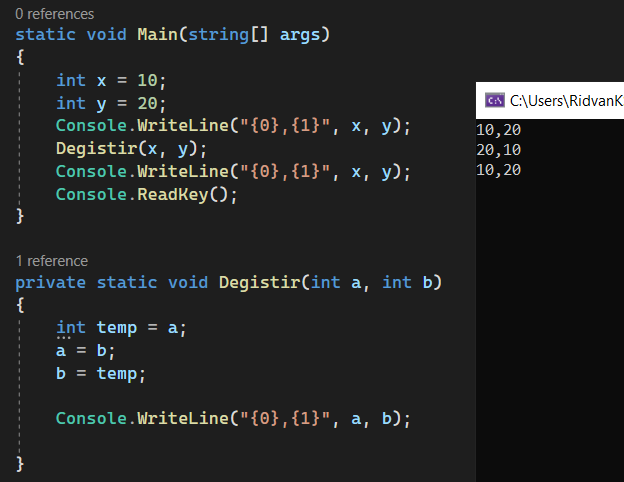


🡪 Örneğin aşağıda “Console.WriteLine()” methodunun açıklaması bulunmaktadır.



**Değer ve Referans Tip**

🡪 Aşağıdaki resim değer tipin nasıl çalıştığının bir örneğidir. Her değişkenin bellek üzerinde bir adresi vardır. Biz bu adreslere değişken isimlerini kullanarak ulaşıyoruz. Eğer bir parametre adrese göre işlem yapmıyorsa değer tiplidir.

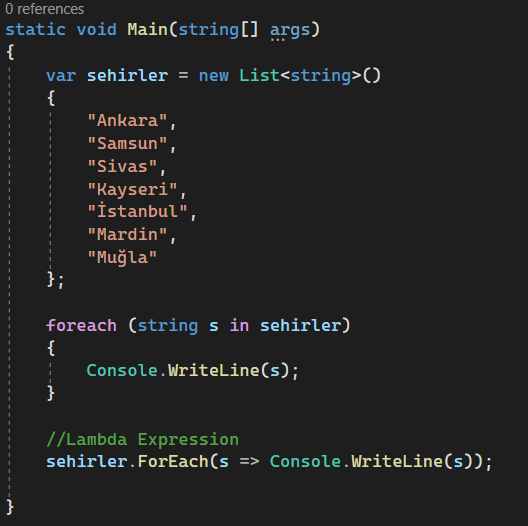


🡪 Aşağıdaki resim referans tipin nasıl çalıştığının bir örneğidir. Biz “ref” anahtar kelimesini kullanarak bu parametrenin adresine göre işlem yapılacağını belirtiyoruz. Örneğin aşağıdaki örnekte “ref x” demek x’in adresine göre işlem yapılacağı anlamına gelmektedir. Yani a değişirse x’de değişir. Referans tip adrese göre çalışır. Adrese göre çalışıyor demek parametre olarak gönderilen ifadenin adres bilgisini tutuyor demektir. Bu örnekte a x’in adres bilgisini, b ise y’nin adres bilgisini tutuyor. Yani herhangi bir değişiklik olduğu zaman “ref” anahtar kelimesinden dolayı adresteki değerin içeriğini değiştiriyor.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

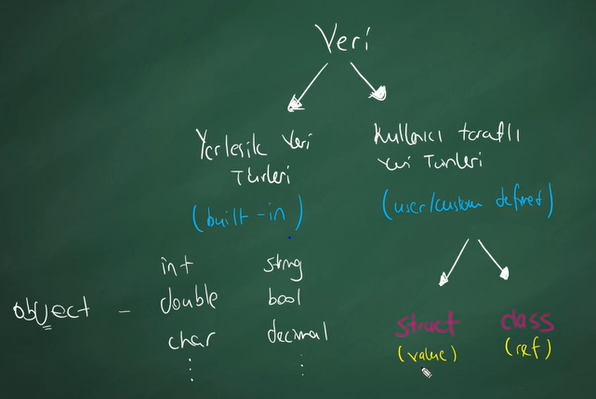
Açıklama otomatik olarak oluşturuldu

🡪 Lambda(Lamda) gösterimi:

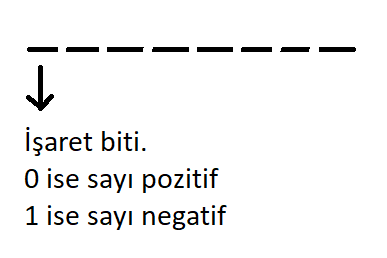


**3) TEMEL VERİ YAPILARI**

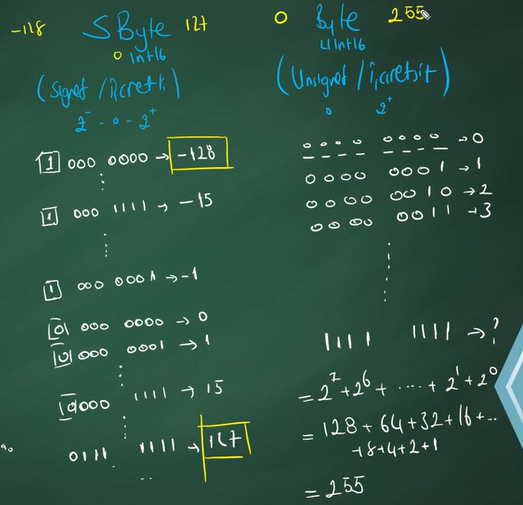
🡪 Verinin 2 farklı türü vardır. Yerleşik veri türleri; int, double, char, string vb. Kullanıcı taraflı veri türleri ise yazılımcının, yerleşik veri türlerinden 1 veya daha fazlasını kullanarak oluşturduğu veri türüdür. Struct’lar değer tipli, class’lar ise referans tiplidir.



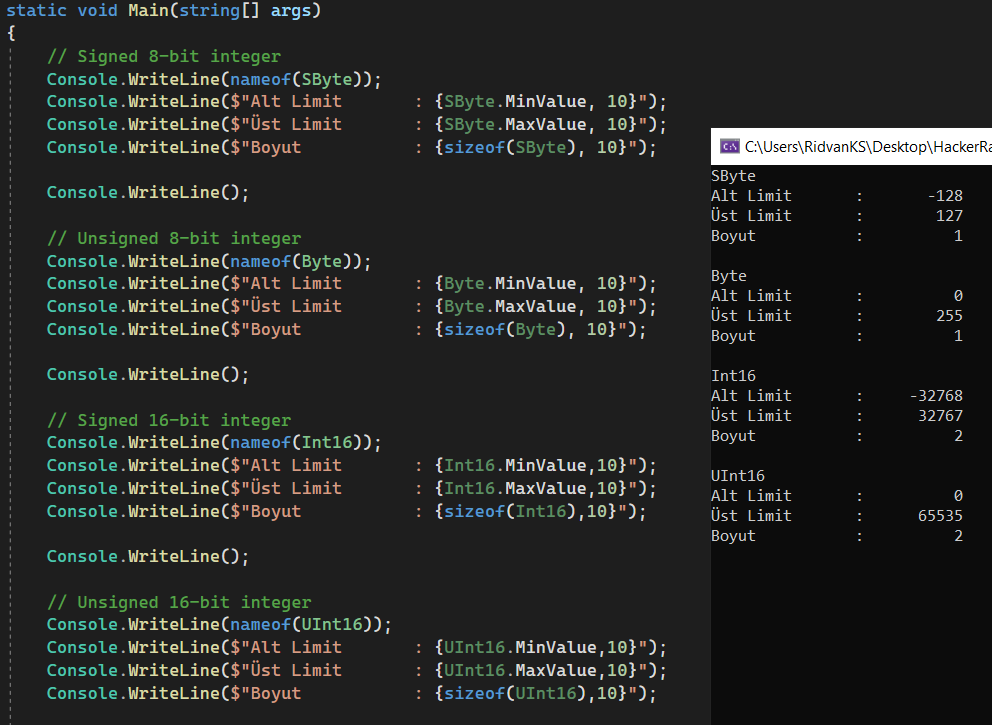
🡪 Int16 = 2 byte, Int32 = 4 byte, Int64 = 8 byte, SByte = 1 byte, Byte = 1 byte (1 byte 8 bit’tir.). Bir sayıyı bit şeklinde yazdığımızda en baştaki bit işaret bitidir. Eğer işaret biti 0 ise sayı pozitif, 1 ise sayı negatiftir. Yukarıdaki ifadelerde yer alan SByte işaretli (signed), Byte ise işaretsizdir (unsigned). İşaretli demek negatif tam sayıları, 0’ı ve pozitif tamsayıları tutabileceği anlamına gelmektedir. İşaretsiz ise sadece 0’ı ve pozitif tamsayıları tutar.



🡪 Int16 işaretli UInt16 işaretsizdir.

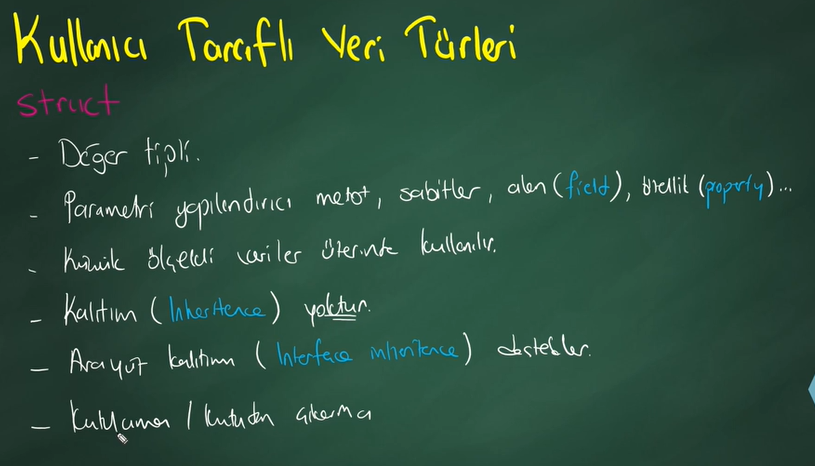


🡪 Aşağıdaki görselde yukarıda anlatılanlar görselleştirilmiştir. C#’da NameOf operatörü, bir değişkenin, sınıfın (class) veya metodun adını almak için kullanılır. Sonuç olarak bize string bir ifade döndürür. SizeOf, belirtilen veri türünün byte olarak ölçülmesini sağlar. Örneğin SByte’ın boyutu 1 byte çıktı. Yani bellekte 1 byte yer kaplıyor.



**STRUCT**

🡪 Struct’lar değer tipli bir veri türüdür. Yapılandırıcı methodları vardır. Default yapılandırıcı methodları yoktur (yani parametresiz yapılandırıcı methodu yoktur.).



🡪 Aşağıdaki örnekte bulunan get ve set ifadelerinin anlamları şunlardır. Get okuma yaparken, set ise yazma yaparken kullanılır. Buna kapsülleme denir.



🡪 Bir struct aşağıdaki gibi tanımlanır. Numara, Ad, Soyad, Cinsiyet ise birer property’dir.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡪 Yapıcı methodların ismi Struct ismi ile aynı olmalıdır. Yapıcı Methodlarda geriye dönen bir ifade bulunmaz (return ifadesi).

metin, ekran görüntüsü, yazılım içeren bir resim

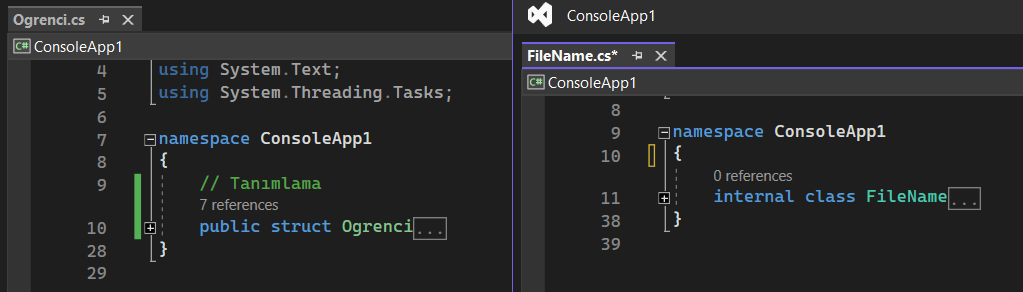
Açıklama otomatik olarak oluşturuldu

🡪Struct’lar aşağıdaki gibi kullanılır. 3 farklı kullanım türü vardır.

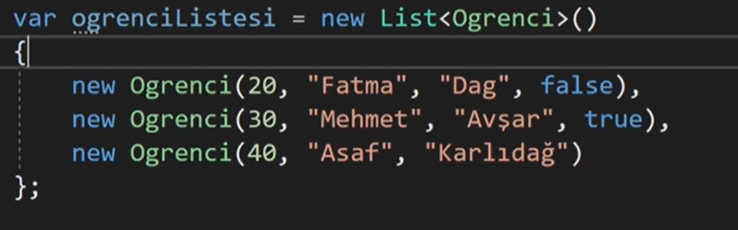
metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡪 Struct’lar farklı dosyalara yazılsalar bile aynı namespace’leri kullanıyorlarsa program sorunsuz çalışacaktır.



🡪 Öğrenci objelerini bir listede tutabiliriz.



🡪 Listemizdeki öğrencileri de aşağıdaki gibi yazdırabiliriz.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

🡪 ToString methodunu aşağıdaki gibi override ettik. Artık bir öğrenci objesini ekrana bastırmak istediğimizde bize aşağıdaki değerleri basacak.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

ÖNEMLİ İFADELER

Casting: Değişkenler arasında tür dönüşümü yapılması.

Boxing: ArrayList’lere eklenen değerlerin obje olarak tutulması

Unboxing: ArrayList’ten çıkarılan ifadelerin cast işlemi(tür dönüşümü) yapılarak çıkarılması.

new(): Referans tipli ifadeleri başlatmak için kullanılır.

Tip güvenliği: Listelerin veya dizilerin tiplerini en başta belirmektir.

Code snippet: Visual Studio’da “Tab” tuşuna 2 kere basına ifadeleri otomatik tamamlaması.

Örtülü değişken(implict): “var” anahtar kelimesini kullanmaya denir. Bu sayede değişkenin belli bir türü olmaz. Gelen ifadeye göre türü belirlenir.

Static: Herhangi bir nesneye ihtiyaç duymadan ilgili methodun doğrudan çalışmasına olanak sağlar.

Method imzası(Method signature): Methodun adının ve parametrelerinin olduğu kısma method imzası denir.

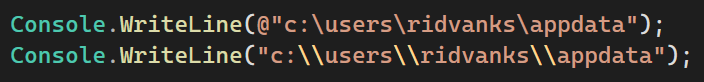
Public: herkes erişebilir.

Private: Sadece aynı class’tan erişilebilir.

Internal: Sadece aynı programdan erişilir. Başka bir programdan erişilemez.

Refactoring: Kod üzerinde yapılan iyileştirmelerdir. Örnek olarak bir kod bloğunu method’a çevirmek veya bazı kodları başka dosyalara taşımak verilebilir.

Verbatim(@): Backslash(\) kullanmadan stringleri yazmamızı sağlar. Örneğin aşağıda hem backslash hem de verbatim kullanılarak oluşturulan bir string örneği var.



String interpolation: Birden fazla tekil değeri bir sabit değere atamaya denir. Yapısı verbatim’e benzer fakat interpolation da dolar işareti($) kullanılır. Tekil değerleri yazarken süslü parantez kullanılır.

metin, yazı tipi, ekran görüntüsü içeren bir resim

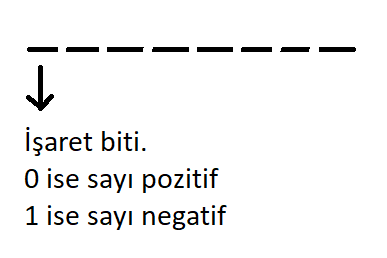
Açıklama otomatik olarak oluşturuldu

Method overloading: Methodun sadece parametrelerini değiştirerek gönderilen parametre sayısına göre davranmasını sağlar.

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

İşaret biti: Sayıyı 2 lik sayı sisteminde çizdiğimizde en baştaki bit işaret bitidir.



Kapsülleme: Get ve set kullanımı.