**MEO# Programlama Dili Kullanıcı Rehberi**

**İçindekiler**

1. Giriş
2. Kütüphane Kullanımı
3. Bildirimler
   1. Sabit Bildirimleri
   2. Değişken Bildirimleri
   3. Sınıf Bildirimleri
   4. Metot Bildirimleri
4. Koşullu İfadeler
   1. Eğer İfadeleri
   2. Durum İfadeleri
5. Döngüler
6. Dönüş İfadeleri
7. Sınıf ve Metot Çağrıları
8. Veri Tipleri
   1. Temel Veri Tipleri
   2. Liste Veri Tipi
9. İfadeler
   1. Mantıksal İfadeler
   2. Aritmetik İfadeler
   3. Karşılaştırma İfadeleri
   4. Temel İfadeler
10. Sözcükler ve Semboller
11. **Giriş**

Bu kullanıcı rehberi, programlama öğrenmek isteyen Türk öğrenciler için MEO# programlama dilini öğrenme sürecinde rehberlik etmek amacıyla hazırlanmıştır.

Rehberimiz, MEO# dilinin temel özelliklerini, kullanımını ve pratik uygulamalarını adım adım açıklar. Dilin basit ve anlaşılır yapısı, programlamaya yeni başlayan öğrencilerin hızlı bir şekilde ilerlemelerini sağlar.

Rehberimizde, dilin yapısal öğeleri, veri tipleri, koşullar ve döngüler, metotlar, sınıflar ve nesne yönelimli programlama (OOP) gibi konulara ayrıntılı bir şekilde yer verilir. Ayrıca, örnekler ve pratikler aracılığıyla MEO# dilini anlama ve kendi projelerinizi geliştirme becerilerinizi geliştirme fırsatı bulacaksınız.

Önemli bir nokta, MEO# dilinde OOP öğelerini kullanmak zorunlu değildir. Bu rehber, OOP konusuna ilgi duyan öğrenciler için temel bir giriş sağlar ve daha ileri düzeydeki konulara geçiş yapmak isteyenlere bir platform sunar.

MEO# dilini öğrenmek için bu rehberi takip ederek adım adım ilerleyebilirsiniz. Kendi projelerinizi gerçekleştirerek ve daha fazla pratik yaparak programlama becerilerinizi geliştirebilirsiniz.

1. **Kütüphane Kullanımı**

Programlama dillerinde, kütüphaneler veya modüller kullanarak hazır fonksiyonları ve bileşenleri projemize dahil edebiliriz. Kütüphane içe aktarma, başka bir kaynaktan veya harici bir dosyadan bir kütüphaneyi projemize dahil etmek anlamına gelir. Bu sayede, kütüphanenin sağladığı özellikleri ve fonksiyonları kullanabiliriz.

MEO# dilinde kütüphaneler sınıfları veya metotları içeren modüller olarak yer alır. Dildeki temel aktiviteler daha sonrasında detaylıca ele alacağımız metot kavramı üzerinden gerçekleştirilir. Bu sebeple bir program yazımı sırasında kendi yazacağımız metotların yanı sıra hazırda yazılmış olan metot ve sınıfların kullanımı için kütüphanelerin içe aktarımına ihtiyaç vardır.

Örnek olarak dilin standart kütüphanesindeki “girdi” ve “çıktı” metotlarını ele alabiliriz. “girdi” metodu standart kütüphanede bulunan ve kullanıcının klavyeden yazdığı veriyi alıp işleyen bir metottur. “çıktı” metodu ise parametre olarak aldığı veriyi ekrana yazdıran bir metot olarak kütüphanede yer alır. Temel program yazımı için hayati önem arz eden bu metotların kullanımını kütüphane kullanımını örneklemek için aşağıda vermiş bulunuyoruz:

kullan ; standart\_kütüphane .

genel değişken metin için kullanıcı\_girdisi : girdi .

{kullanıcı\_girdisi} için çıktı .

Örnekte verilmiş olduğu üzere standart kütüphane çağrılmış ve kütüphanedeki “girdi” ve “çıktı” fonksiyonları kullanılarak kullanıcının girdiği metni ekrana yazdıran basit bir program yazılmıştır.

MEO# dili minimal bir tasarıma sahip olduğu için standart kütüphanesinde sadece veri giriş çıkışları ile ilgili metotlara sahiptir. Aritmetik işlemler ve ilkel sınıflar dilin kendisinde tanımlı olup herhangi bir kütüphaneye bağlı değildir. Standart kütüphane yukarıda örneği verilmiş olan “girdi” ve “çıktı” metotlarını ve dosya okuma ve yazma işlerinde kullanılan “dosya” sınıfını içerir. “dosya” sınıfında temel olarak kullanılan “oku” ve “yaz metotları vardır ve dosya formatı sınırı olmayıp doğrudan dosyanın içeriği ele alınır.

Standart kütüphane içeriği ve kullanımı belirtildiğine göre kendi kütüphanenizi yazma işine gelebiliriz. Fakat, bunun için öncelikle bildirimler (3.bölüm) incelenmelidir. Kütüphane oluşturma şu şekildedir:

erişim\_cinsi sınıf sınıf\_1 için :

#Sınıf içeriği#

erişim\_cinsi sınıf sınıf\_2 için :

#Sınıf içeriği#

.

.

.

erişim\_cinsi metot type metot1 {type parameter1, type parameter2} için :

#Metot içeriği#

erişim\_cinsi metot type metot2 {type parameter1, type parameter2} için :

#Metot içeriği#

.

.

.

Sadece sınıf ve metotlardan oluşan kütüphane kaydedildiği dosya adıyla çağırılarak kullanılır.

1. **Bildirimler**

MEO# dilinde program yazımına başlarken kullanılacak unsurların kullanımdan önce bildirilmesi gereklidir. Bu unsurlar sınıflar, metotlar, değişkenler ve sabitler olmak üzere gruplandırılır. Bildirim programın, sınıfın veya metodun başlangıcında yapılmalıdır. Bu bölümde bildirim nasıl yapılır inceleyeceğiz.

* 1. **Sabit Bildirimleri**

Sabitler, değeri programın akışı içerisinde değiştirilemeyen, bildirildiğinde bir değer alan ve o değeri kullanım esnasında koruyan unsurlardır. Sabit bildirimi esnasında tanımlanan sabitin erişim cinsi, sabitin tipi ve değeri bildirilir ve sabit, program içerisinde veya bildirildiği sınıf veya metodun içerisinde kullanımından önce belirtilmek zorundadır. Sabit bildirimi örnek olarak şu şekilde gösterilebilir:

genel sabit tam\_sayı için SAYI\_1 : 50 .

genel sınıf sınıf\_1 için :

özel sabit metin için METİN\_1: “Selamlar!” .

.

.

.

!

genel metot boş metot\_1 için :

özel sabit liste için LİSTE\_1: {15, 10, 1} .

.

.

.

!

Örnekte daha sonra 3.3. ve 3.4. bölümlerde ele alacığımız sınıf ve metot bildirimleri kullanılarak sınıf ve metot içinde sabit bildirimi örneklendirilmiştir. “özel” ve “genel” ifadeleri erişim cinsini belirtmektedir.

!!! Erişim cinsi, bildirilen bir öğenin programın hangi bloklarında erişilebileceğini gösterir. “özel” olan bir öğe sadece bildirildiği blokta kullanılabilirken, “genel” olan bir öğe ise programın tamamında kullanılabilir. Bunun amacı donanımın hafızasını etkili bir şekilde kullanabilmektir. !!!

Örnekte görüldüğü şekilde sabit bildirimi yapılmakta olup sabite atanmış olan değerler sadece bildirim yapıldığı satırda değiştirilebilir. Yani sadece bir defa değer ataması yapılabilir.

* 1. **Değişken Bildirimleri**

Değişken bildirimleri, sabit bildirimleri ile çok benzer yapıdadır. Değişkenler, sabitlerden farklı olarak daha sonradan farklı değer atamaları ile değerleri değişebilen öğelerdir. Değişken bildirimi için bir örnek verecek olursak:

genel değişken metin metin\_1 : “Selamlar!” .

genel sınıf sınıf\_1 için :

özel değişken gerçek\_sayı için sayı\_1: 1,2 .

.

.

.

!

genel metot boş metot\_1 için :

özel değişken liste için liste\_1: {metin\_1, 10, (25,2)} .

metin\_1 : “Güle Güle!” .

.

.

!

Örnekte görülmüş olduğu gibi erişim cinsi ve türü belirtilen değişken değer atanarak bildirilmiştir ve daha sonra “metot\_1” de görüldüğü gibi bildirilen “metin\_1” değeri farklı bir değer atanarak değiştirilebilmiştir.

Değişkenlerin değer atanarak bildirilmesi zorunlu değildir. Aşağıdaki gibi bir bildirim de mümkündür:

genel değişken tam\_sayı için sayı\_1 .

Bu değişkene daha sonra kullanım esnasında değer atanabilir.

* 1. **Sınıf Bildirimleri**

Sınıflar, nesne yönelimi programlamada (OOP) kullanılan kendi içerisinde unsurlar barındıran ve obje olarak çağırılıp kullanılabilen öğelerdir. Bu bölümde MEO# dilinde nasıl sınıf bildirimi yapılır ve sınıf içeriği nasıl yazılır bundan bahsedeceğiz.

Sınıf bildirimi önceki bölümlerden de aşina olduğunuz gibi erişim çeşidi sınıf tanımlayıcısından oluşur ve sınıf içi unsurları içeren bir blokla sürdürülür. Örnek olarak:

genel sınıf hayvanlar için:

özel değişken metin için hayvan\_sınıfı .

özel değişken metin için hayvan\_türü .

özel değişken metin için hayvan\_ismi .

genel metot hayvanlar {metin hs, metin ht, metin hi} için :

hayvan\_sınıfı : hs .

hayvan\_türü : ht .

hayvan\_ismi : hi . !

genel metot metin ismi\_al için :

hayvan\_ismi döndür . !

genel metot metin sınıfı\_al için :

hayvan\_sınıfı döndür . !

genel metot metin türü\_al için :

hayvan\_türü döndür . !

genel metot boş ismi\_belirle {metin hi} için :

hayvan\_ismi : hi . !

genel metot boş sınıfı\_belirle {metin hs} için :

hayvan\_sınıfı : hs . !

genel metot boş türü\_belirle {metin ht} için :

hayvan\_türü : ht . !

genel metot metin metne\_çevir için :

hayvan\_sınıfı + “,” + hayvan\_türü + “,” + “hayvan\_ismi” döndür . !

!

Örnekte “hayvanlar” isimli bir sınıf tanımlanmıştır ve sınıf içinde kendi değişken ve metotlarını içermektedir. Tipi belirtilmemiş olan sınıf ile aynı isimdeki metot sınıfın yapıcı metodudur ve sınıfın taşıdığı unsurları girilen parametreler ile bu metot belirler. Sınıf metotlarının kullanımını daha sonra sınıf ve metod çağırımlarını anlatan bölümlerde ele alacağız. Aynı örnek üzerinden devam edersek genellikle sınıf yazımında yapıcı metot, alma ve belirleme metotları ve bunun dışında kullanım ihtiyacına göre başka metotlar da tanımlanır.

* 1. **Metot Bildirimleri**

Metotlar sınıf içinde veya doğrudan programın içinde tanımlanan, çağırımı sırasında içerisndeki satırları çalıştıran ve döndürülmesi istenmişse istenen değeri döndüren öğelerdir. Metotların bildirimi erişim cinsi ve tip belirlenmesi ve isteğe bağlı parametre belirlenmesi ile olur. Önceki örneklerden aşina olduğunuz üzere metot çağırımı için örnek vermek gerekirse aşağıdaki gibidir:

genel metot boş yap\_bişeyler için :

{“Bir şeyler yaptım!”} için çıktı . !

genel metot tam\_sayı topla {tam\_sayı say1, tam\_sayı say2} için :

özel değişken tam\_sayı için sonuç : say1 + say2 .

sonuç döndür . !

Örnekte görüldüğü gibi method içerisine istenilen görev ve isteğe bağlı olarak döndürülecek değer belirlenerek metotlar yazılabilir.

1. **Koşullu İfadeler**
   1. **Eğer İfadeleri**

Eğer ifadeleri belirli bir koşula bağlı olarak gerçekleşecek eylemleri belirtir. Bu koşullar karşılaştırma ifadelerinden oluşur ve bu ifadeler “doğru” ya da “yanlış” olan boolean değerleri belirtmektedir. Eğer girilen koşul doğruysa ifade gerçekleşir, değilse isteğe bağlı olarak sonrasında girilen ifade gerçekleşir ya da sadece ifade gerçekleşmemiş olur. Örnek yazım aşağıdaki gibidir:

genel değişken tam\_sayı için say1 : 4 .

genel değişken tam\_sayı için say2 : 5 .

say1 = say2 ise ;

{“sayılar eşittir”} için çıktı .

değil say1 < say2 ise ;

{“1.sayı 2.sayıdan küçüktür”} için çıktı .

değil ise ;

{“2.sayı 1.sayıdan küçüktür”} için çıktı . !

Örnekte görüldüğü üzere 3 adet koşul belirtilmiştir. Bu koşullara göre çıktımız “1.sayı 2.sayıdan küçüktür” olacaktır. Yani sırasıyla okunup doğru olan koşuldaki istenen eylem gerçekleşmektedir.

* 1. **Durum İfadeleri**

Durum ifadeleri de eğer ifadeleri gibi koşullu ifadelerden biridir. Durum ifadelerinde başlangıçta belirtilmiş olan bir değişkenin aldığı değerler üzerinden koşul sağlanır. Bu ifadelerden değişkenin aldığı değer eşleşmesi sonucundaki durum gerçekleşir.

Örnek yazımı aşağıdaki gibidir:

genel değişken tam\_sayı için sayı : 5 .

sayı için ;

1 -> {“1”} için çıktı .

2 -> {“2”} için çıktı .

5 -> {“5”} için çıktı . !

Örnekten anlaşılacağı üzere çıktımız “5” olacaktır. Verilen değişken farklı değerler aldıkça sonuç değişecektir.

1. **Döngüler**

Proglamadaki en önemli ifadelerden biri de döngülerdir. MEO# dilinde döngü ifadeleri iki şekilde yazılmaktadır.

İlk olarak döngü içerisinde koşullu ifade yazılarak belirli bir koşulda döngünün devamı sağlanır. Bu yazım şeklinde diğer dillerden farklı olarak koşullu ifadedeki her bir koşul önemlidir. İfadelerden herhangi biri sağlandığı sürece döngü ona göre devam eder. Hiçbir koşul sağlanmıyorsa döngüden çıkılır.

Örnek yazım şekli aşağıdaki gibidir:

genel değişken metin için selam : “Merhaba!” .

döngü ;

selam = “Merhaba!” ise ;

{“Merhaba!”} için çıktı .

selam : “Selam!” .

değil selam = “Selam!” ise ;

{“Selam!”} için çıktı .

selam : “Hello!” . ! !

Verilen örnekte bir selam değişkeni bildirilmiş olup “Merhaba!” değeri atanmıştır. Döngüdeki koşullara göre öncelikle ilk koşula girilir ve “Merhaba!” yazdırılır. Daha sonra selam değeri “Selam!” olarak değiştirilir ve döngünün ikinci dönüşünde ikinci koşul gerçekleşir ve “Selam!” yazdırılır ve değer “Hello!” olarak değişir. Döngüye bir sonraki girişte iki koşul da sağlanmadığından döngüden çıkılır.

MEO# dilindeki bir diğer döngü uygulaması ise belirli bir sayıdan belirli aralıklarla artış sonucu ulaşılan sayıya kadar devam eden bir döngü biçimidir. Aşağıdaki örnekte bu daha iyi ifade edilmektedir:

döngü;

0 -> 5 -> 1 ++ ;

{“Hello Word!”} için çıktı . !

Bu döngü yazım biçiminde alacağımız çıktı 5 defa “Hello World!” olacaktır. Girilen ilk sayı kaçtan başlayacağını, ikinci sayı kaça kadar devam edeceğini, üçüncü sayı ise artış miktarını belirtmektedir. Bu döngü yapısı belirli bir miktarda gerçekleştirilecek eylemler için çok faydalıdır.

1. **Dönüş İfadeleri**

Dönüş ifadeleri metot bildirimleri bölümünden aşina olacağınız, bir metot bitiminde kullanılan ve metodun belirtilen tipte bir değer döndürmesini sağlayan ifadelerdir.

Örnek kullanımı aşağıdaki gibidir:

genel metot tam\_sayı çarpma\_işlemi {tam\_sayı say1, tam\_sayı say2} için:

özel değişken tam\_sayı için sonuç : 0 .

döngü;

0 -> say2 -> 1 ++ ;

sonuç : sonuç + say1 . !

sonuç döndür . !

Örnekte, bildirilen metot kendi içerisinde bir döngü ile hesaplama yapmakta ve sonucu döndürmektedir.

1. **Sınıf ve Metot Çağrıları**

Önceki bölümlerde sınıf ve metotların nasıl bildirildiklerini ve ne olduklarını gördük. Sıra geldi nasıl kullanıldıklarına.

Öncelikle bir sınıfın çağrılabilmesi için önceden bildirilmiş olduğunu biliyoruz. Önceden bildirilmiş olan tanımlı sınıf bir obje oluşturularak ya da içerisindeki bir metot çağrılarak kullanılabilir.

Metotlar ise bir cümle olarak çağrılabilir ya da bir değişkene döndürdüğü değeri vermek için de kullanılabilir.

Ayrıca sınıfa bir yapıcı metot yazıldıysa yapıcı metottaki parametreler kullanılarak sınıfın içeriği belirlenebilir.

Sınıf çağırımı ve sınıf içindeki metot kullanımları örnek olarak şu şekildedir:

genel değişken sınıf\_adı için değişken\_adı : {parametre\_değeri\_1, parametre\_değeri\_2} için yeni sınıf\_adı .

değişken\_adı | sınıfiçi\_metot\_1 .

değişken\_adı | {parametre\_değeri} için sınıfiçi\_metot\_2 .

metot\_1 .

genel değişken tip değişken\_2 : {parametre\_değeri} için metot\_2 .

Verilen örnekte ilk cümlemiz önceden tanımlı bir sınıftan verilen parametreler ile bir obje oluşturmaktadır. Daha sonra bu objeden sınıf içindeki metotlara erişerek bu metotlar o obje için kullanılır. “metot\_1” sınıf içinde tanımlı olmayan bir metodun doğrudan kullanımını göstermektedir. Son cümlede ise “metot\_2” döndürdüğü değeri değişkene atamaktadır.

Özetle sınıf ve metot çağrımlarını görmüş olduk.

1. **Veri Tipleri**
   1. **Temel Veri Tipleri**

Bu bölümde MEO# programlama dilinin içerisinde tanımlanmış olan temel veri tiplerini inceleyeceğiz.

Temel veri tipleri, “tam\_sayı”, “gerçek\_sayı”, “metin” ve “bool” olarak tanımlanmıştır. “tam\_sayı” veri tipi girilen tam sayı ifadelerini kapsarken “gerçek\_sayı” veri tipi gerçek sayıları yani “,” ile ayrılan basamağa sahip sayıları ifade eder. “metin” ler ise tırnak işaretiyle belirtilen ve her türlü karakteri kapsayan veri yapılarıdır. “bool” veri yapısı Boolean anlamına gelen sadece “doğru” ve “yanlış” değerlerini alan veri yapısıdır.

Bu veri yapıları değişken, sabit ve metot tanımlarında kullanılır. Kendi içlerinde işleme tabi olabilirler. Örneğin “metin” ile “metin” toplanabilir, “bool” ile “bool” mantıksal operatörler ile işleme tabi olabilir. Kısacası her veri tipi kendi içinde işleme tabidir, fakat birbirleri ile işleme giremezler.

Bu duruma istisna olarak “tam\_sayı” ve “gerçek\_sayı” tipleri birbirleri ile aritmetik işleme tabi olabilirler ve “metin” tipi her türlü temel veri tipiyle işleme tabi olabilir.

* 1. **Liste Veri Tipi**

MEO# dilinde liste veri tipi diğer programlama dillerine kıyasla daha kolay işlenmiştir.

Listeler çoğu programlama dilinde aynı tipte verileri içerirken, MEO# dilinde listelerin içerebileceği veri tipi açısından bir kısıtlama yoktur. Örneğin, bir liste hem “metin” hem de “tam\_sayı” içerebilir.

Ayrıca listeler farklı boyut özellikleri de taşıyabilirler. Bu durum listelerin eleman olarak liste içermesi durumuyla meydana gelir.

Listelerde işlemler ve indeksleme eylemleri önemli hususlardır. Önceden tanımlanmış bir listeye eleman ekleme “+” operatörü ile gerçekleşir ve eleman çıkarma durumu da indeks ile belirtilmiş bir eleman ile “-“ operatörü uygulanması ile gerçekleşir.

Aşağıda liste tanımlama ve işlemlerine dair bir örnek verilmiştir:

genel değişken liste için liste\_1 : {12, “on iki”, doğru, (15,4)} .

genel değişken tam\_sayı için on\_iki : liste\_1(0) .

liste\_1 – on\_iki .

liste\_1 + 12 .

Verilen örnekte 4 farklı tipte elemandan oluşan bir liste tanımlanmıştır. Daha sonra listenin ilk elemanı bir değişkene atanmış ve değişken yardımıyla atanan eleman listeden çıkarılmıştır. Daha sonra listeye yeni eleman eklenmiştir. Son durumda listenin ilk elemanı olan “12” sayısı çıkarılıp sonradan eklendiği için listenin son elemanı olmuştur.

Bu örnekler çoğaltılabilir.

Unutulmaması gerekir ki liste indekslenirken ilk eleman her zaman “0” olarak belirtilir.

MEO# dilinde diğer programlama dillerinde hazır verilen liste uzunluğu bulma ya da liste üzerindeki diğer işlemler hazırda bulunmamaktadır. Dil bu gibi durumların kullanıcı tarafında algoritma yazımı geliştirebilmesi açısından kullanıcıya bırakmıştır. Gerekli durumlarda metotlar yazılabilir.

1. **İfadeler**
   1. **Mantıksal İfadeler**

Mantıksal ifadeler temel mantık operatörleri ile kurulmuş ifadelerdir. MEO# dilinde bu operatörler “ve”(AND), “veya”(OR), “ya\_da”(XOR) ve “değil”(NOT) ifadelerinden oluşur.

Bu ifadeler “bool” ibarelerini ve koşulları birleştirmek veya tersine çevirmek için kullanılır.

Örneğin:

değil (1 = 2 ve 2 = 3 veya 4<5) ya da 5>=8 .

* 1. **Aritmetik İfadeler**

Aritmetik ifadeler temel matematik işlemlerinden oluşan ifadelerdir.Fakat, burada sadece matematiksel işlemlerde değil, diğer veri tipleri için de işlemlerde kullanılabilir. Örneğin:

“Hello” + “World” .

1 + 2 .

1 + 1,5 .

liste + 1,5 – liste(9) .

gibi.

* 1. **Karşılaştırma İfadeleri**

Koşullu ifadelerden hatırlayacağınız üzere, karşılaştırma ifadeleri “bool” değer döndüren ve aynı türden ifadeleri kıyaslayan ifadelerdir. Örneğin:

1=2 .

“Hello” = “Selam” .

1<2 .

5<= 5 .

Gibi.

* 1. **Temel İfadeler**

Temel ifadeler, veri tipleri ve objelerden oluşan ifadelerdir. Bütün veri tiplerini ve objeleri kapsar.

1. **Sözcükler ve Semboller**

MEO# dilinde değişken, sabit, metot veya sınıfları ifade etmek için kullanılan sözcükler harfle başlar ve boşluk içeremez. Bunun dışında bir karakter kısıtlaması yoktur.

Dildeki karakterler geçerli bütün karakterleri kapsar.

Aritmetik operatörler: - , \* , / , +

Karşılaştırma operatörleri : < , > , <= , >= , =

Satır sonu: “ . “

Blok sonu: “ ! “

Yorum başı ve sonu: “ # “

MEO# rehberimiz bu kadardır. Programala dilinin daha çok kitle kazanması ve öğrencilere sağlam temeller oluşturmasında katkı sunabilmesi umuduyla…

**Appendix**

**EBNF Definition**

<program> ::= [<library\_import>] [<declaration\_part>] <statement\_part>

<library\_import> ::= “kullan” “;” <identifier> {“,” <identifier>} <end\_of\_line>

<declaration\_part> ::= <acces\_modifier> (<constant\_declaration> | <variable\_declaration> | <class\_declaration> | <method\_declaration>) {<declaration\_part>}

<constant\_declaration> ::= “sabit” <type> “için” <identifier> “:” <constant> <end\_of\_line>

<variable\_declaration> ::= “değişken” <type> “için” <identifier> [“:” <expression>] <end\_of\_line>

<class\_declaration> ::= [<class\_inheritance>] “sınıf” <identifier> “için” “:” <class\_block> <end\_of\_block>

<method\_declaration> ::= “metot” <type> <identifier> [“{“ <parameter\_list> “}”] “için” “:” <method\_block> <end\_of\_block>

<class\_inheritance> ::= <identifier> “genişlet” “;”

<class\_block> ::= <declaration\_part>

<method\_block> ::= [{<variable\_declaration> | <constant\_declaration>}] <statement\_part>

<parameter\_list> ::= <type> <identifier> {“,” <type> <identifier>}

<statement\_part> ::= {<statement>}

<statement> ::= <assignment\_statement> | <method\_call\_statement> | <conditional\_statement> | <loop\_statement> | <return\_statement> | <break\_statement>

<assignment\_statement> ::= <identifier> “:” <expression> <end\_of\_line>

<method\_call\_statement> ::= <method\_call> <end\_of\_line>

<conditional\_statement> ::= <if\_statement> <case\_statement>

<if\_statement> ::= <condition> “ise” “;” <statement\_part> {“değil” [<condition>] “ise” “;” <statement\_part>} <end\_of\_block>

<condition> ::= (<expression> <relational\_operator> <expression> | <boolean>) {<logical\_operator> <condition>} <end\_of\_block>

<case\_statement> ::= <identifier> “için” “;” {<expression> “->” <statement\_part>}

<loop\_statement> ::= “döngü” “;” <conditional\_statement> <end\_of\_block> | “döngü” <expression> “->” <expression> “->” <expression> “++” “;” <statement\_part> <end\_of\_block>

<return\_statement> ::= <expression> “döndür” <end\_of\_line>

<break\_statement> ::= “sonlandır” <end\_of\_line>

<method\_call> ::= [<identifier> “|”] [“{“ <actual\_parameter\_list> “}” “için”] <identifier>

<actual\_parameter\_list> ::= <expression> {“,” <expression>}

<type> ::= “metin” | “tam\_sayı” | “gerçek\_sayı” | “bool” | “liste” | <identifier> | “boş”

<access\_modifier> ::= “özel” | “genel

<expression> ::= <logical\_expression>

<logical\_expression> ::= <comparison\_expression> {<logical\_operator> <comparison\_expression>}

<comparison\_expression> ::= <additive\_expression> {<comparison\_operator> <additive\_expression>}

<additive\_expression> ::= <multiplicative\_expression> {<additive\_operator> <multiplicative\_expression>}

<multiplicative\_expression> ::= <primary\_expression> {<multiplicative\_operator> <primary\_expression>}

<primary\_expression> ::= <identifier> | <constant> | <method\_call> | “(“ <expression> “)” | <object>

<object> ::= [“{“ <actual\_parameter\_list> “}” “için”] “yeni” <identifier>

<constant> ::= <number> | <string> | <boolean> | <list>

<list> ::= “{“ <expression> {“,” <expression>} “}”

<indexing> ::= <identifier> “(“ <integer> “)”

<number> ::= <integer> | <float>

<integer> ::= <digit> {<digit>}

<float> ::= <integer> “,” <integer>

<string> ::= “”” <character> {<character>} “””

<boolean> ::= “doğru” | “yanlış”

<identifier> ::= <letter> {<letter\_or\_digit> | <special>}

<letter> ::= “a” | “b” | ... | “z” | “A” | “B” | ... | “Z”

<digit> ::= “0” | “1” | ... | “9”

<letter\_or\_digit> ::= <letter> | <digit>

<special> ::= “!” | “#” | . . . | “?”

<logical\_operator> ::= “ve” | “veya” | “ya\_da” | “değil”

<additive\_operator> ::= “+” | “-“

<multiplicative\_operator> ::= “\*” | “/”

<comparison\_operator> ::= “=” | “<” | “>” | “<=” | “>=”

<end\_of\_line> ::= “.”

<end\_of\_block> ::= “!”

<comment> ::= “#” <character> {<character>} “#”