# NESNELER VE SİNİFLAR

Nesneye dayalı programlama bütün hesaplamaların nesneler bağlamında yapıldığı bir programlama yaklaşımıdır. Nesneye dayalı programlama diliyle yazılmış bir program verilen görevi gerçekleştirmek için biraraya gelmiş nesneler topluluğu olarak görülebilir.
Nesne, onun şimdiki durumunu tanımlayan veriler ile dış dunyaya servisler vermesini ve şimdiki durumunu değiştirmesini sağlayan yöntemleri kapsayan bir varlıktır. Benzer nesneler kümesine tek bir sınıfın örnekleridir denir. Bir sınıfı aynı türden nesneler için ayrintili bir tasarım olarak düşünebilirsiniz.
Java programlama dilinde bir sınıf **class** anahtar kelimesini kullanarak tanımlanabilir. Örneklerin durumları üye değişkenler olarak tanımlanabilir. Bir üye değişken basitçe bir tür ve ve bir isim belirtilerek tanımlanabilir. Ayrıca üye değişkenleri tanımlarken ilklendirmek de mümkündür. Yöntemler, sıfır veya daha fazla parametreli ve dönme değerli fonksiyonlar olarak tanımlanabilir. Bir methodun dönme değeri, hiç birşey dönülmez anlamındaki **void** türünde olabilir. Ayrıca, farklı parametreler aldığı sürece aynı isimde birden fazla yöntem tanımlamak da mümkündür.
Bir sınıfın üye değişkinlerine ve yöntemlerine erişimi, **public**, **protected**, ve **private** anahtar kelimelerini kullanarak sınırlandırmak mümkündür. Sadece genel üye değişkenler ve genel yöntemler sınıfın tanımı dışından erişilebilir. Bu erişim kısıtlama yöntemleri daha sonraki bölümlerde ayrıntılı olarak anlatılacaktır.
Aşağıda, bir sınıf tanımlama şablonu verilmiştir.

```
class SinifAdi{
        public degiskenTuru1 degiskenAdi1;
        public degiskenTuru2 degiskenAdi2=ilkDeger;
        public donmeTuru YontemAdi1(parametreTuru1
parametreAdi1,
                                    ..., parametreTuruN
parametreAdiN){
                ...
                return donmeDegeri;
        }
        public void YontemAdi2(parametreTuru1 parametreAdi1,
                                    ..., parametreTuruN
parametreAdiN){
                ...
        }
}
```

# NESNELER YARATMA VE KULLANMA

Her nesne kullanılmadan önce açıkça yaratılmalıdır. Bir nesne yaratmak için, ilgili sınıfın **constructor** (yapılandırıcı) olarak adlandırılan özel yöntemi çağrılmalıdır. Kurucu nesne için gerekli kaynakları ayırır ve yaratılan nesnenin bir örneğini döndürür. Java proglamlama dilinde, yapılandırıcıyı, sınıfla aynı isimde bir yöntem yaratarak kendiniz tanımlayabilirsiniz. Farklı parametreler aldıkları sürece birden fazla kurucu tanımlanabilir.

```
class SinifAdi{
        public SinifAdi (sıfır veya daha fazla parametre){
                ...
        }
}
```

Bir sınıfın bir örneğini yaratmak için **new** işletmeni (operator) kullanılır. Bu örnek, referans türünün değişkeni olarak tutulabilir, bu da aşağıdaki gibi tanımlanır ve ilklendirilebilir.

```
SinifAdi DegiskenAdi = new SinifAdi(<parametreler>);
```

Burada **<parametreler>** eğer varsa, kurucu tarafından istenen parametrelerdir. Bir örneğin üye değişkenlerine ve yöntemlerine sınıf tanımı dışından, direk isimlerini kullanarak veya referans değişkeni yoluyla erişilebilir:

```
DegiskenAdi.UyeDegiskenAdi
DegiskenAdi.YontemAdi(<parametreler>)
```

# Sınıf Değişkenleri ve Sınıf Yöntemleri

Her örnek farklı durumlara sahip olabileceğinden, üye değişkenler bir sınıfın farklı örneklerinde farklı değerler tutabilirler. Fakat, `static` anahtar kelimesini kullanarak, sınıf değişkeni denilen, değeri bu sınıfın bütün örnekleri boyunca aynı olan, özel bir çeşit üye değişken tanımlamak mümkün. Örneğin:

```
class SinifAdi{
        public DegiskenTuru DegiskenAdi = IlkDeger;
        ...
}
```

Bir sınıf değişkenine, diğer herhangi bir üye değişkene ulaşıldığı gibi ulaşılabilir veya sınıf adını kullanarak `SinifAdi.DegiskenAdi` şeklinde de ulaşılabilir. Benzer şekilde bir sınıf yöntemi de `static` anahtar kelimesi kullanilarak tanımlanabilir:

```
class SinifAdi{
        public static DondurmeTuru YontemAdi(parametreler){
                ...
        }
        ...
}
```

Sınıf değişkenleri gibi sınıf yöntemleri de referans değişken yoluyla veya bir örnek istemeden direk sınıf adıyla çağrılabilir. Yani, sınıf yöntemleri bir örnek yaratmadan kullanılabilir. Bunlar örneksiz çağırılabildiklerinden, sınıf değişkenleri dışındaki üye değişkenleri değiştiremezler.

# JAVA PROGRAMİ

Java programşama dili tamamen nesneye dayalı bir dil olduğundan, bir programın içindeki herşey sınıf tanımlamalarında içerilmelidir. Bir program içindeki birçok sınıf istenilen görevi yerine getirmek için birbirlerinin yöntemlerini çağırarak işbirliği yapabilirler.
En az bir tane sınıf tanımlaması içeren bir program kaynak dosyası yaratarak bağımsız bir uygulama oluşturabilirsiniz. Her program yürütülmek için, ilk ifade için bir giriş noktasına ihdiyaç duyar. Bu nedenle, bir program için giriş noktasını sınıfın bir yöntemi olarak açıkça belirtmek zorundasınız. Bu amaç için **`main`** özel yöntemi kullanılır. Sınıf yöntemleri bir örnek çağırılmasına ihdiyaç duymadıklarından, ana (main) yöntemini içeren sınıfı belirtmek programı yürütmek için yeterli olacaktır.
Ana (main) yöntemin biçimi aşağıdaki gibidir:

```
public static void main(String[] args){
...
}
```

Bu **`main`** yöntemi tek bir parametre, **`args`**, alır ve hiçbir değer döndürmez. Bu **`args`** parametresi, **`String`** dizisi (array) türündedir ve program çalıştırıldığında komut satırına girilen parametreleri içerir. Ana yöntem (**`main`**), sınıf dışından örneksiz çağırılabilen bir yöntem olmak zorunda olduğundan, **`public`** ve **`static`** anahtar kelimeleri kullanılır.

# AÇIKLAMALAR

Java proglamlama dili üç tür açıklamayı destekler: tek satır açıklamalar `//` ile, çok-satır açıklamalar `"/*` ve `*/"` ile, ve `javadoc` açıklamalar `"/**` ve `*/"` ile gösterilir. İlk iki tür basit açıklamalar, ve üçüncüsü `javadoc` aracı ile işlenen programın gömülmüş belgelemesi için kullanılır.
Tek satır açıklamalar `//` şekli kullanılarak, programın herhangi bir yerine konabilir. Açıklama ilk `//` ile başlar ve satır sonuna kadar devam eder. Aşağıdakiler tek satır açıklamalara örnektir:

```
int i=7; // bir tek satır açıklama
i=i*7;
// başka bir tek satır açıklama
i=1;
```

Çok-satır açıklama, bir programda açıklama birden fazla satıra yayıldığında kullanılır. Derleyici başlangıç biçim imi `/*` ile ilk bitiş imi `*/` arasındaki herşeyi atacaktır. Aşağıdakiler çok-satır açıklamalara örnektir:

```
int i=7;
/* bir
açıklama */
i=i*7; /*
başka bir
açıklama
*/
i=i+1;
```

`Javadoc` açıklamalar birçok satıra yayılabilir ve derleyici başlangıç biçim imi `/**` ile ilk bitiş imi `*/` arasındaki herşeyi atacaktır. `Javadoc` özelliği `javadoc` açıklamalarını okuyup HTML belgesine çevirebilir.
Sınıf ve arayüz seviyesinde `javadoc` açıklamalar ekleyebilirsiniz, yöntem, kurucu ve alan seviyesinde ekleyebileğiniz gibi. Her açıklama ilişkin varlıktan hemen önce görünür ve bir veya daha fazla imle izlenen bir tanımdan oluşur. Eğer isterseniz `javadoc` açıklamalarınızda HTML biçimlendirmesi kullanabilirsiniz. `Javadoc` açıklamasının genel biçimi aşağıdaki gibidir:

```
/**
  * Bir sınıf betimlemesi
  *
  * @imadı betimleyici metin
  * ...
  * @imadı betimleyici metin
  */
public class DocTest {
  /** Bir değişken betimlemesi*/
  public int i;
/**
  * Bir yöntem betimlemesi
  *
  * @imadı betimleyici metin
```

```
   * ...
   * @imadı betimleyici metin
   */
  public void f() {}
}
```

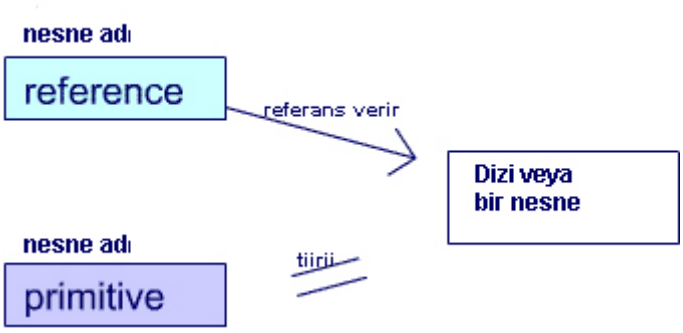
`Javadoc` açıklamları için tanımlanmış çok farklı çeşitte im (tag) vardır, `@param değiştirgeç_adı değiştirgeç_betimlemesi`,`@author yazar_adı`, `@version sürüm_bilgisi` gibi. İmlerin tüm listesini için Java belgerine başvurabilirsiniz.

# DEĞİŞKENLER

Bir değişken veri (ve nesneler) için isimlendirilmiş bir kapsayıcıdır. Bir değişken için çalıştırma zamanında bellek alanı ayrılır ve adı, değerini almak yada belirlemek için kullanılır. Yani, değişkenin adını bellek yerini okumak yada yazmak için kullanabilirsiniz.
Bir değişkeni kullanmadan önce bildirmelisiniz. Yani, bir değişkeni adına ve türüne bir isim vererek bildirmelisiniz. Javada bütün değişkenlerin türü olmalıdır, bu nedenle derleyici değişkenin içerdiği veriyi nasıl yorumlayacağını bilir. Bir değişken şu şekilde bildirilebilir: `TurAdi DegiskenAdi`.
Bir değişken ilkel (primitive) yada ilgi (reference) türünden olabilir. İlkel türünden bir değişken yalnız tamsayı (`byte`, `short`, `int`, `long`), ondalık sayı(`double`, `float`), tek yada unicode karakter (`char`), yada tek aç/kapa (on/off) durumu (`boolean` `true` yada `false`) içerir. İlgi türünden bir değişken bir `sınıf`'ın yada `dizi`'nin elemanını işaret eder.

Bir değişkenin ismi geçerli bir `tanıtıcı (identifier)` olmalıdır.Yani, harfle başlayan tek bir kelime olmalıdır (içinde boşluk yok). Aslında para birimi sembolü yada altçizgi (_) ile de başlayabilir, fakat en iyisi harf ile başlatmaktır. Harf, sayı, altçizgi, para birimi sembolü (örneğin `$`) içerebilir, fakat başka özel karakter içeremez. Java büyük küçük harfe duyarlı (case-sensitive) olduğu için, bir değişken bildirildikten sonra ismi aynı formda kullanılmalıdır.

Aşağıdaki şekil referans türünde değişkenlerin ilkel türden farkını gösterir:



# DEĞİŞKEN İLKLENDİRME

Yöntem parametreleri ve kural dışı durum parametreleri kendilerini çağıran yöntem tarafından ilklendirilirler. Bir üye yada yerel değişken kendi bildiriminde atama işletmeni `=` kullanarak şu şekilde ilklendirilebilir:

```
turadi
tanitici = baslangic_degeri;
```

Yada değeri bildiriminden sonra belirlenebilir:

```
turadi tanitici;
...
tanitici = baslangic_degeri
...
```

Bir üye değişkenin varsayılan değer alması -ilklendirilmese bile- kesindir (sayısal değişkenler için sıfır, `boolean` değişkenler için `false` ve karakterler için unicode karakter sıfır ve basvuru türleri için `null`). Fakat, yerel değişkenleri kullanmadan önce ilklendirmeniz gerekir.
Ayrıca bir değişkeni değeri ilklendirldikten sonra değişmeyen `final` değişken olarak bildirebilirsiniz. Bir `final` değişkeni her bağlamda şu şekilde bildirebilirsiniz:

```
final float PI = 3.14; //bildirim sırasında ilklendirme
...
final char firstLetter; // boş final
...
firstLetter = 'A' // ertelenmiş ilklendirme
```

`final` değişkenler aslında diğer dillerdeki sabitlerin karşılığıdır. `final` değişkenlerin ilklendirildikten sonra değerini değiştirmek için yapılan girişim derleme zaman hatası ile sonuçlanacaktır.Örneğin, `PI` değişkeninin değeri bildiriminden sonra değiştirilemez ve `firstLetter`'ın değeri ilklendirildikten sonra değiştirilemez.

# KAPSAM

Bir değişkene kendi kapsamı içerisinde erişilebilir.Bir değişkenin kapsamı onun bildirilmesinin bağlamına bağlıdır. Bir değişken esas olarak dört bağlamda bildirilebilir: bir sınıf içinde üye değişkeni olarak (herhangi bir yöntem dışında), bir yöntem bildirimi içinde yöntem parametresi olarak, bir yöntem içinde yerel parametre olarak, ve kural dışı durum işleme(exception handler) içinde kural dışı durum işleme parametresi olarak bildirilebilir. (Kural dışı durum işleme (exception handler) bu bölüm içinde daha sonra açıklanacaktır).

Bir üye değişkenin kapsamı bir sınıfın bütün bildirimini kaplar. Bir yöntem parametresinin kapsamı bir yöntemi tam olarak kaplar. Bir yerel parametrenin kapsamı parametrenin bildiriminden onu saran kod bloğunun sonuna kadardır. (Bir kod bloğu, bu bölüm içinde daha sonra tanımlanacağı gibi, sol { ve sağ } kıvırcık kaşlı ayıraçların (curly braces) arasındaki herşeydir).

# İLKEL VERİ TÜRLERİ

İlkel türde bir değişken önceden tanımlanmış tek boyut ve biçim değerini içerebilir. İlkel türde bir değişkenin biçim ve boyutu programın üstünde çalıştığı sistemle değişmez. Bu da Java programlama dilinde yazılmış programların taşınabilirliğine katkı sağlar.
Üç çeşit ilkel veri türü vardır: sayısal (tamsayı ve ondalık), bool, ve karakter. Sağdaki tablo, Java proglamlama dilinin desteklediği ilkel veri türlerini listeliyor.
Bütün tamsayı veri türlenin işaretli ve bool veri türünün boyutunun belirtilmediğine dikkat ediniz. Değişkeniniz için değerlerin erimine göre uygun bir tamsayı veri türü seçebilirsiniz. Aksi takdirde, hesaplamalarınız sonucu doğru olmayabilir (taşma (overflow) olabilir). Benzer şekilde, eğer uygun bir ondalık sayı türü seçmezseniz, duyarlık kaybedebilirsiniz veya hesaplamalar `sonsuz (infinite)` sonucu verebilir.

**İlkel Veri Türleri**

| Kategori | Tür Adı | Boyut | Biçim / Değerlerin Erimi (range) |
|---|---|---|---|
| Tamsayılar | `byte` | 1 byte tamsayı | 2'nin tümleyicisi `-128` den `127` e |
| | `short` | 2 byte tamsayı | 2'nin tümleyicisi $-2^{15}$ den $2^{15}-1$ e |
| | `int` | 4 byte tamsayı | 2'nin tümleyicisi $-2^{31}$ den $2^{31}-1$ e |
| | `long` | 8 byte tamsayı | 2'nin tümleyicisi $-2^{63}$ den $2^{63}-1$ e |
| Ondalık Sayılar | `float` | 4 byte kayan-noktalı sayı | IEEE 754 |
| | `double` | 8 byte gerçek sayı | IEEE 754 |
| Karakterler | `char` | 2 byte Unicode karakter/td> | Unicode `0` dan Unicode $2^{16}-1$ e |
| Bool'lar | `boolean` | - | `doğru` veya `yanlış` |

Bir ilkel veri türünün değeri örtülü olarak derleyiciyle veya açık olarak tür-değiştirme yöntemiyle, bir diğer ilkel veri türüne dönüştürülebilir. Dönüşüm türünü genişletmek (örneğin `byte` dan `int` e, `int` den `float` e, v.s.) derleyici ile örtülü olarak yapılabilir. Fakat dönüşüm türünü daraltmak (örneğin `double` dan `int` e, `int` den `short` a, v.s.) açık değişim gerektirir. Açık tür değişimi aşağıdaki forma sahiptir:

```
(tür) değer
```

Burada, `tür`, `char` a veya herhangi bir sayısal veri türüne ilişkindir, ve `değer` ilkel veri türünde değere sahip olan herhangi birşey (direk yazılmış bir değer- hazır bilgi, bir değişken, veya değer döndüren bir ifade, `bool` haricinde) olabilir.

# HAZIR BİLGİLER (LİTERALS)

Bir hazır bilgi direkt olarak bir programda kullanılabilen bir sayının, bir karakterin, bir bool, yada bir dizginin gerçek sunumudur. Yandaki tablo her çeşit hazır bilgi biçim özetlerini ve örnekleri göstermektedir. Bir sayısal hazır bilginin önüne `-` işareti koyarak onun negatif değere sahip olacağına, ve `e` (or `E`)'den sonra `-` koyarak ondalık sayıların üst kısmı negatif yapılabileceğine dikkat etmelidir. Karakter hazır bilgileri (Character literals) tek tırnak ile `' '`, dizgi hazır bilgileri (String literals) `" "` ile çevrelenir.

**Hazır Bilgiler**

| Tür | Biçim | Örnekler |
|---|---|---|
| `int` | Basamaklar dizisi `0-9` (onaltılı için karakterler `0-9,A,B,C,D,E,F` ve sekizli için karakterler `0-7`) | `123`<br>`0123` (sekizli taban 8)<br>`0x123` (onaltılı-taban 16) |
| `long` | Basamaklar dizisi `l` yada `L` harfi olarak devam eder. | `123L` |
| `double` | Bir '.' sembollü ve/veya `.` `e` (yada `E`) harfli basamaklar dizisi. Hazır bilgi `d` yada `D` harfi ile devam edebilir. | `123D`<br>`1.23`<br>`1.23D`<br>`1.2E-3`<br>`12.3E4D` |
| `float` | >Bir '.' sembollü ve/veya `.` `e` (yada `E`) harfli basamaklar dizisi. Hazır bilgi `f` yada `F` harfi ile devam etmelidir. | `123F`<br>`1.23F`<br>`1.2E-3F`<br>`1.2E3F` |
| `boolean` | `true` yada `false` | `true`<br>`false` |
| `char` | Tek tırnak içindeki tek karakter | `'a'` |
| `Dizgi` | Çift tırnak içindeki karakterler | `"abc"` |

**Kaçma Karakterleri**

| Karakter | Kaçma Karakteri |
|---|---|
| Ters eğik çizgi (Backslash) | `\\` |
| Geri (Backspace) | `\b` |
| Satır başı (Carriage Return) | `\r` |
| Tek tırnak (Single Quote) | `\'` |
| Çift tırnak (Double Quote) | `\"` |

Yazdırılamayan karakterler için, kaçma karakteri karakter yada dizgi hazır bilgisinde kullanılabilir. Soldaki tablo bazı kaçma karakterlerini listeler. Örnekler: `'\''` tek

| | |
|---|---|
| Form besleme (Form Feed) | `\f` |
| Yatay sekme (Horizontal Tab) | `\t` |
| Yeni satır (New Line) | `\n` |
| Unicode karakter (`X 0-9,A,B,C,D,E,F` lerden biri) | `\uXXXX` |
| Sekizli (Octal) Karakter (`D 0-7`lerden biri) | `\DDD` |

tırnak içeren karakter hazır bilgisi, `'\n'` yeni satır içeren karakter hazır bilgisi , `"asd\n"` a,s,d ve yeni satır karakterlerini içeren hazır bilgi , ve `"asd\nfgh"` içerisinde yeni satırı içeren dizgi hazır bilgisi.

# İŞLETMENLER

İşletmenler bir değer üretmek için işlenenleri üzerinde bir fonksiyon gerçekleştirirler. Java programlama dilinde işletmenler önceden tanımlanmış fonksiyonlar gibi çalışırlar. Örneğin, `A` ve `B` işletenlerinden ve `+` işletmeninden oluşturulmuş `A+B` anlatımı; eğer `A` ve `B` nin ikisi de tamsayı (`int`) türündeyse, `A+B` bunların toplamını döndürür. Bir işletmen bir, iki veya üç işlenen isteyebilir. Bir işlenen isteyen işletmene birli işletmen, iki işlenen isteyene ikili işletmen ve üç işlenen isteyene üçlü işletmen denir.
İşletmenlerin çoğu ilkel veri türleri döndüren anlatımlar üzerinde işlerler. Burada *anlatım* değişkenler, hazır bilgiler, yöntemlerin döndürdüğü ilkel türler veya işletmen tarafından şekillendirilen diğer anlatımlar anlamındadır. Sadece `=`, `==`, and `!=` işletmenleri ayrıca ilgi türleriyle de çalışırlar ve `+` ve `+=` ayrıca `Dizgi` nesneleri (sonraki bölümlerde anlatılacak) üzerinde de çalışırlar. Altı çeşit temel işletmen vardır: aritmetik, mantıksal, bit bazında, atama, karşılaştırma ve üçlü eğer-diğer (if-else) işletmenleri. Ayrica işletmenlerle şekillendirilmiş anlatımları, birden fazla işletmenli birleşik anlatımlar oluşturmak üzere birleştirmek mümkün.
Birden fazla işletmen kullanılarak oluşturulmuş ifadeleri değerlendirme sırası öncelik kuralları uygulanarak belirlenir. Örneğin, * ve / , + ve - dan önce değerlendirilir. genellikle diğer öncelik kurallarını hatırlamak kolay değildir. Bu yüzden benzer anlatımları parantezle kaplamak tercih edilir. Bu yolla anlatımı değerlendirmedeki belirsizliği azaltmış olursunuz. Örneğin, aşağıdaki satrilarda `i` ve `j` nin değerleri eşit görünmelerine rağmen farklı olacaktır.

```
i=a+b+c/d+e;
j=a+(b+c)/(d+e);
```

# ARİTMETİK İŞLETMENLER

Aritmetik işletmenler sayısal veri türleri (tamsayı ve ondalık sayılar) üzerindeki matematiksel fonksiyonları gerçekleştirebilirler. Aşağıdaki tablo Java programlama dili tarafından desteklenen aritmetik işletmenleri listelemektedir.
`++` ve `--` işletmenlerinin işlemden sonra ayrıca kendi işlenenlerinin değerini de değiştirdiğine dikkat ediniz. İşlemin sonucu işletmenin konumuna göre belirlenir.

| Anlatım | Fonksiyon |
|---|---|
| `++A` | Ön artırma: Bu anlatım `A=A+1` e eşittir ve sonuç `A+1` a eşittir. |
| `--A` | Ön azaltma: Bu anlatım `A=A-1` e eşittir ve sonuç `A-1` a eşittir. |
| `A++` | Sonradan artırma: Sonuç A'ya eşittir, fakat anlatım değerlendirildikten sonra `A` nın değeri `A+1` olarak belirlenir. |
| `A--` | Sonradan azaltma: Sonuç A'ya eşittir, fakat anlatım değerlendirildikten sonra `A` nın değeri `A-1` olarak belirlenir. |
| `+A` | Yükseltme: Eğer `A` `byte` veya `short` türünde ise, sonuç `int` türünde `A` olacaktır. |
| `-A` | Negatiflik: Sonuç `A` nın negatifidir. |
| `A+B` | Sonuç `A` ve `B` nin toplamıdır. |
| `A-B` | Sonuç `B` nin `A` dan çıkarılmasıdır. |
| `A*B` | Sonuç `A` ile `B` nin çarpımıdır. |
| `A/B` | Sonuç `A` nın `B` ye bölümüdür. Eğer `A` ve `B` nin ikisi de tamsayı ise, tamsayı bölümü gerçekleştirilir. (Sonuç `A/B` işleminin tamsayı kısmıdır). |
| `A%B` | Sonuç `A/B` bölme işleminde kalandır. |

# KARŞILAŞTIRMA İŞLETMENLERİ

Karşılaştırma işletmenleri iki işleneni karşılaştırı ve bir `bool` sonuç üretir. Aşağıdaki tablo Java programlama dilinin desteklediği karşılaştırma işletmenlerini listeler.

| Anlatım | Fonksiyon |
|---|---|
| `A==B` | Eşittir: Eğer `A` nın değeri `B` ye eşitse sonuç `doğru` olacaktır, yoksa `yanlış` olacaktır. |
| `A!=B` | Eşit değildir: Eğer `A` nın değeri `B` ye eşit değilse sonuç `doğru` olacaktır, yoksa `yanlış` olacaktır. |
| `A<B` | Daha az: Eğer `A` nın değeri `B` den daha az ise sonuç `doğru` olacaktır, yoksa `yanlış` olacaktır. |
| `A<=B` | Daha az veya eşit: Eğer `A` nın değeri `B` den daha az veya eşitse sonuç `doğru` olacaktır, yoksa `yanlış` olacaktır. |
| `A>B` | Daha fazla: Eğer `A` nın değeri `B` den daha fazla ise sonuç `doğru` olacaktır, yoksa `yanlış` olacaktır. |
| `A>=B` | Daha fazla veya eşittir: Eğer `A` nın değeri `B` den daha fazla veya eşitse sonuç `doğru` olacaktır, yoksa `yanlış` olacaktır. |

# MANTIKSAL İŞLETMENLER

Mantıksal işletmenler, DEĞİL, VE, VEYA, DIŞLAYAN VEYA (XOR) gibi bool işlemlerini gerçekleştirmek için bool değerler üzerinde işlerler. Yandaki tablo Java programlama dilinin desteklediği mantıksal işletmenleri listeler.
Koşullu VE ve koşullu VEYA işletmenleri, değeri brinci işlenene bağlı olan ikinci işlenenleri değerlendirmeyebilirler.

| Anlatım | Fonksiyon |
|---|---|
| `!A` | DEĞİL: Eğer `A doğru` ise sonuç `yanlış` olacaktır, Eğer `A yanlış` ise sonuç `doğru` olacaktır. |
| `A&B` | VE: Eğer `A` ve `B` nin ikisi de `doğru` ise sonuç `doğru` olacaktir, yoksa sonuç `yanlış` olacaktır. |
| `A&&B` | Koşullu VE: Eğer `A doğru` ise sonuç `B` nin değeri olacaktır, yoksa sonuç `yanlış` olacaktır (`B` değerlendirilmeyecektir) . |
| `A\|B` | VEYA:Eğer `A` ve `B` nin ikisi de `yanlış` ise sonuç `yanlış` olacaktir, yoksa sonuç `doğru` olacaktır. |
| `A\|\|B` | Koşollu VEYA: Eğer `A yanlış` ise sonuç `B` nin değeri olacaktır, yoksa sonuç `true` olacaktır (`B` değerlendirilmeyecektir) . |
| `A^B` | DIŞLAYAN VEYA: Eğer `A` veya `B` den yalnız biri `doğru` is sonuç `doğru` olacaktır, eğer her ikisi de `doğru` veya her ikisi de `yanlış` ise sonuç `yanlış` olacaktır. |

# ÜÇLÜ İŞLETMEN

Java programlama dilinde yalnız bir tane üçlü işletmen vardır `?:`. `A?B:C` şeklindeki anlatım `A` nın değerine bağlı olarak, `B` nin veya `C` nin değerini döndürür. Birinci işlenen `A`, `bool` değeri döndüren bir anlatım olmalıdır ve eğer `A doğru` ise `B` değerlendirilir ve sonuç onun değeri olur. Eğer `A yanlış` ise `C` değerlendirilir ve sonuç onun değeri olur. `A` nın değerine bağlı olarak `B` veya `C` işlenenlerinden birisinin değerlendirilmediğine dikkat ediniz.

# BİT BAZINDA İŞLETMENLER

Bit bazında işletmenler DEĞİL (`~`), VE (`&`), VEYA (`|`), DIŞLAMALI VEYA (XOR) (`^`), veya kaydırma (sağ yada sol) gibi işlenenler üzerindeki işlemleri bit bit gerçekleştiriler. Bu işlemlerde işlenenler tamsayı olmalıdır (`&` ve `|` daha önce de gördüğümüz gibi, bool'lar üzerinde de işletilebilirler) ve işlemler işlenenlerin değerlerinin ikili temsilleri üzerinde yerine getirilir.

Eğer bir değerin bit-n'i, o değerin ikilik gösterimindeki n. konumdaki biti ise,`~A`,`A&B`, `A|B`, `A^B` için sonuçların bit-n'i şuna göre değerlendirilir:

| A nın bit-n'i | B nin bit-n'i | ~A nın bit-n'i | A&B nın bit-n'i | A\|B nın bit-n'i | A^B nin bit-n'i |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |

Kaydırma işletmenleri ilk işlenenin bitlerini ikinci işlenende belirtildiği kadar sağa yada sola kaydırır ve boş bitleri `0` ile doldurur (sağa kaydırma `>>` işletmeni hariç). Java programlama dili tarafından desteklenen kaydırma işletmenleri yandadır.

| Anlatım | Fonksiyon |
|---|---|
| A<<B | Sola kaydırma: A nın bitlerini B nin konumu kadar sola kaydırır, 0 lar sağdan kaydırırlır (yüksek degerli bitler kaybedilir) s |
| A>>B | İşarteli sağa kaydırma: A nın bitlerini B nin konumu kadar sağa kaydırır. Eğer A negatif ise 1 ler soldan kaydırılır; eğer positif ise 0 lar kaydırılır (A nın işareti korunur). |
| A>>>B | İşaretsiz sağa kaydırma: A nın bitlerini B nin konumu kadar sağa kaydırır, 0 lar sağdan kaydırılır. |

# ATAMA İŞLETMENLERİ

Temel atama işletmeni; `=`, sol işlenene sağ işlenenin değerini atar. Örneğin, `A=B` anlatımı `A`'nın değerini `B`'nin değeri olarak belirler. Bu ifadede, sağ işlenen `B`, değer üreten herhangi bir bir anlatım olabilir. Fakat, sol işlenen değişken olmalıdır.
Aritmetik, mantıksal ve bit bazında işletmenler, bir işlemi gerçekleştirmek ve sol işlenene değer atamak için `=` işletmeni ile kullanılabilirler. Örneğin, `A` ve `B` işlenenlerini toplamak ve sonucu `A`' ya atamak için, `+=` işletmeni `A+=B` şeklinde kullanılabilir. Bu durumda, `=` işletmeni gibi sağ işlenen bir değer üreten herhangi bir anlatım olabilir ve sol işlenen bir değişken olmalıdır.
Yandaki tablo bu tür kısa atama işletmenlerini ve onların eşdeğerlerini listelemektedir.

| Anlatım | Eşdeğer Anlatım |
|---|---|
| A+=B | A=A+B |
| A-=B | A=A-B |
| A*=B | A=A*B |
| A/=B | A=A/B |
| A%=B | A=A%B |
| A&=B | A=A&B |
| A\|=B | A=A\|B |
| A^=B | A=A^B |
| A<<=B | A=A<<B |
| A>>=B | A=A>>B |
| A>>>=B | A=A>>>B |

# REFERANS VERİ TÜRLERİ

Referans veri türünde bir değişken bir nesneye veya bir diziye referans olabilir. (İleriki derslerde anlatılacağı gibi ayrıca arayüzlere ve sınıflara da referans olabilir). Referans veri türündeki değişkenler sadece birer referans olduklarından, bunların kullanımı ilkel veri türlerinden oldukça farklıdır. Referans türünde bir değişkenin bildirilmesi ilkel türde bir değişkenin bildirilmesiyle cok benzerdir.

```
TurAdi degiskenAdi
```

Burada, ***TurAdi*** referans edilen varlığın türünün adıdır, ***degiskenAdi*** geçerli bir tanıtıcı olmalıdır. İlkel veri türleri gibi yöntem parametreleri çağırıcı ile ilklendirilir. Üye değişkenler `null` değerle ilklendirilir, yani boş bir referansla, ve bunlar daha sonra ilklendirilebilir. Fakat yerel değişkenleri kullanmadan önce açıkca ilklendirmelisiniz.
Değişkeni bildirerek sadece değişken için fakat referans verilecek varlık için değil, basitçe bir yer tutucu (bir ilgi depolama bölgesi) yaratmış olursunuz. Bu nedenle basit bir ilklendirme varlığın referansını başka bir değişkenden `=` işletmenini kullanarak kopyalayabilir veya `new` işletmenini kullanarak yeni bir varlık yaratılabilir ve onun referansı değişkene `=` işletmeni kullanılarak geçirilebilir. Aşağıdaki örnek bu değişkenleri bildirme ve ilklendirmelerini gösterir.

```
BenimSinifim x = new BenimSinifim();
BenimSinifim y = x;
```

Bu örnekte `BenimSinif` türünden `x` değişkeni `BenimSinif` ın bir örneği yaratılarak bildirilmiş ve ilklendirilmiştir. Varlığın bir örneğini yaratmak için `new` işletmeni kullanılmıştır. Bundan sonra diğer değişken `y`, `x`'i kullanarak bildirilmiş ve ilklendirilmiştir. Eğer varlık `sınıf` türünden ise `new` işletmenini kullanarak aslında bu sınıfın yapılandırıcısını çağırmış oluyorsunuz. Yapılandırıcı nesne için gerekli kaynakları ayırır ve nesnenin bir örneğini döndürür. Atama işletmeni `=` ve karşılaştırma işletmeni `==` ve `!=` de ilgi türünde dağişkenlerle kullanılabilir. Fakat bu referans değişkenlerinin sadece gerçek değişkenlere referans içerdikleri unutulmamalıdır. Bu yuzden `=` işletmeni sadece bir varlığın referansını başka bir değişkene kopyalar, ve `==` ve `!=` işletmenleri sadece referansları (içeriklerini değil) karşılaştırır ve eğer eşitlerse `doğru` döndürür.

# Anlamsiz Verİ Toplayici

`New` işletmeni kullanarak bir varlık oluşturduğunuzda kaynaklar ayrılır ve bu nesne için referenslar döndürülür. Diğer birçok programlama dilinde, açıkça yaratılan fakat artık ihtiyaç duyulmayan varlıkları temizlemek (yada yok etmek) zorundasınız. Fakat Javada, `new` işletmeni ile yaratılan varlıkları yok etmeye ihtiyaç duymazsınız. Artık ihtiyaç duyulmayan varlıkları, anlamsız veri toplama mekanizması onları sizin yerinize yok eder. Bir nesneyi referans gösteren hiçbir değişken yoksa, nesne anlamsız veri toplayıcıya konur. Yani bir varlığı referans gösteren bütün değişkenler kapsam dışına çıkarsa, farklı bir varlığa yada `null` değerine atanırsa varlık otomatik olarak yok edilir.

# SARICI SINIFLAR

İlkel türler, diziler ve dizgiler için sarıcı sınıflar programlarda sık sık kullanılan basit referans veri türleridir.

Bazı durumlarda ilkel veri türünün bir değerini içeren bir nesneye ihdiyaç duyarsınız (sarıcı sınıfların bir kullanım alanı da koleksiyonlar konusunda daha sonra anlatılacaktır). Aşağıdaki tablo her bir ilkel veri türünün sarıcı sınıfını gösterir. Bu sınıfların değişkenleri sıradan referans değişkenler gibi bildirilir ve ilklendirilir. Herhangi bir sarıcı sınıfın nesneleri `new` işletmeni kullanarak ve ilgili türün bir değeri yapılandırıcıya geçilerek yaratılabilir.

| İlkel veri türü | Sarıcı sınıf |
|---|---|
| `byte` | `Byte` |
| `short` | `Short` |
| `int` | `Integer` |
| `long` | `Long` |
| `float` | `Float` |
| `double` | `Double` |
| `char` | `Character` |
| `boolean` | `Boolean` |

Bir nesne içindeki sarıcı sınıf türünden değeri kullanmak için nesnenin `xxxValue()` yöntemini çağırabilirsiniz, burada *xxx* ilgili ilkel veri türünün adıdır, örneğin `byteValue()`, `intValue()`, `booleanValue()` gibi.
Örneğin aşağıdaki kod `Integer` türünde iki nesne yaratır ve onları iki değere, `i`, ve `k` ve `s`, atar:

```
Integer i = new Integer(5);
int j = 5;
Integer k = new Integer(j);
boolean b = i==k;
Integer s = k;
boolean e = s==k;
j = i.intValue() + k.intValue();
```

Yukarıdaki örnekte aynı değeri, `5`, içermelerine rağmen `i` and `k` değişkenleri birbirine eşit değildir (örneğin `b`, `false`'dur). Fakat aynı nesneye ilgi olduklarından `s` ve `k` değişkenleri birbirine eşittir (örneğin `e` `true`'dur) .

# DİZİLER

Bir dizi aynı veri türünden değerler grubu içerebilir. Bir diziyi bildirmek için, ya değişken adından sonra yada veri türünden sonra köşeli parantez `[]` konur. Örneğin,

```
int[] j
int k[]
```

Burada ilkel tür `int`'in elemanı olan bir boyutlu `j`, ve `k` bildirilir. Diğer referans türleri gibi, bir dizinin bildirimi o dizi için bellek ayırmaz. Dizi kullanılmadan önce yaratılmalıdır (ki değişken ilklendirilsin). Ayrıca dizi `new` işletmeni ile de şöyle yaratılır: `new elementType[size]`. Örneğin,

```
int[] j = new int[10];
int k[];
k = new int[5];
```

Büyüklüğü `10` ve `5` diziler yaratın ve bunlara `i` ve `j` değişkenlerini sırayla referans olarak atayın. Dizinin elemanlarına ayrıca parantez kullanılarak da şöyle `variableName[index]`erişilebilir. `index` `0` 'dan başlar ve dizinin son elemanı `size-1`'dir. Aşağıdaki örnekte, büyüklüğü `3` olan `char` dizisi yaratılır ve üyeleri ilklendirilir:

```
char[] c = new char[3];
c[0] = 'a';
c[1] = 'b';
c[2] = c[1];
```

`{}` parantezi kullanılarak bir dizi yaratmanın ve dizi üyelerini ilklendirmenin bir kısa yolu da vardır. Aşağıdaki örnek kısayol ilklendirmeyi gösterir.

```
char[] c = {'a', 'b', 'b'};
```

Bir dizinin büyüklüğü `length` değişkeni kullanılarak öğrenilebilir. Örneğin yukarıdaki örnekte `c.length`, `3`'ü döndürür.
Çok-boyutlu diziler ve referans türünde elemenlar içeren diziler yaratmak da mümkün. Diziler daha sonra detaylı olarak anlatılacaktır.

# DİZGİLER

Java programalama dilinde `String` (dizgi) nesneleri karakter serilerini tutmak için kullanılır. Diğer programlama dillerinin aksine `String` nesnesinde içerilen dizgiler atamadan sonra değiştirilemezler. Yani dizgiler yaratıldığında sabittirler. Bunun yerine dizgi türünde bir değişkene yeni bir referans atayabilirsiniz. Aşağıdaki örnek dizgi nesnesi yaratmanın ve değer atamanın farklı yollarını gösteriyor:

```
String s = new String("abc"); // Yapılandırıcı kullanarak
yaratma
String t = "abc"; // kısayol ilklendirme
s = "abccd"; // kısayol atama
```

`+` ve `+=` işletmenleri yeni bir dizgi yaratmak üzere dizgi üzerinde de çalışır. `+` işletmeni her türdeki işletenle kullanılabilir. Örneğin aşağıdaki kod yürütüldüğünde `s`'nin değeri `"abc105"` olacaktır.

```
String s;
s = "abc";
s += 1;
s += '0';
int j=5;
s = s+j;
```

İki dizgi değişkeninin değerini karşılaştırmak için dizginin `compareTo()` yöntemini kullanmalısınız (değişkenleri direk karşılaştırmak demek sadece referansları karşılaştırmak demektir değerleri değil). Örneğin dizgi türündeki `A` ve `B` deki değerleri karşılaştırmak için `A.compareTo(B)`'i çağırabilirsiniz. Bu yöntem, eğer aynı dizgiyi içeriyorlarsa `0`, eğer `A<B` ise `-1`, ve eğer `A>B` ise `1` döndürür.
İlkel veri türleri ve dizgiler `String` sınıfının ve ilkel türlerin sarıcı sınıflarının uygun yöntemleri kullanılarak birbirlerine çevrilebilirler. İlkel türler `String`'e ilkel türünün sarıcı sınıfının `toString(değer)` yöntemiyle çevrilebilir (bu yöntem bir sınıf yöntemi olduğundan sınıf adını direk kullanarak hiçbir örnek yaratmadan kullanılabilir). Bir dizgi ilkel türe dizginin `valueOf(dizgi)` kullanılarak çevrilebilir. Örneğin:

```
int i = 5;
String s = Integer.toString(i);
i = String.valueOf(s);
```

# DEYİMLER

Bir deyim bir programın içindeki tek komuttur. Bir tek deyim noktalı virgülle `";"` biter. Tek bir deyim birden fazla satırdan oluşan kodları kaplayabilir, fakat noktalı virgüle kadar olan herşey tek bir komutmuş gibi işlenir. Bir deyim:

- Değişken bildirebilir,
- Referans veri türü yaratabilir,
- Bir değişkene değer atayabilir,
- Bir değişkeni arttırabilir veya azaltabilir (`++` veya `--` işletmenlerini kullanarak)
- Bir yöntem çağırabilir,
- Bir yöntemin yürütülmesini tamamlayabilir (`return döndürme_değeri;`).

Çoklu deyimler birleşik deyimler oluşturmak için (`{` ve `}`) paranteziyle kapsanabilir. Bu deyimler sadece bir deyime izin verilen yerlerde (yürütme kontrolu deyimlerinde açıklanacak) kullanılabilirler. Aşağıdaki örnek basit bir deyim bloğunu göstermektedir.

```
...
{
   int i = 0;
   i += i+1;
   String s = i.toString();
   s = "2*2=" + s;
}
...
```

Program yürütülürken, deyimler yazıldıkları sırayla yürütülür. Fakat, deyimlerin yürütülmesini akış kontrolu deyimleri kullanarak bazı koşullara göre kontrol etmek mümkündür. Üç grup akış kontrolu deyimi vardır: Döngü deyimler, koşullu deyimler, ve dallanan deyimler. Döngü ve koşullu deyimler kendileri zaten deyim oldukları için istenilen derinlikte yuvalanabilirler (gövdelerinde başka akış kontrolu içerebilirler).

# DÖNGÜ DEYİMLER

Döngü deyimler aynı deyimin (veya deyimlerin), verilen koşul anlatımına gore, sıfır veya daha fazla kez yürütülmesini sağlar. Java programlama dilinde üç çeşit döngü deyimi vardır: `while`, `do-while`, and `for`.

`while` deyimi verilen koşul `doğru` olduğu sürece tek bir deyimi veya deyim öbeğini (birden fazla deyimi yürütmek isterseniz) yürütür. Koşul deyimi bir `bool` değer üretmelidir. `while` deyiminin biçimi aşağıdaki gibidir:

```
...
while (kosul) deyim;
// veya
while (kosul){
  deyim 1;
  ...
  deyim n;
}
```

`Do-while` deyimi benzer yürütme yapar. Verilen koşul `doğru` olduğu sürece deyimi ya da deyimleri yürütür. Fakat `while` deyiminden farklı olarak, `while` ifadesini takip eden deyimler koşul ifadesine bağlı olarak hiç yürütülmeyebilirken, bunda `do` ifadesini takip eden deyimler en az bir kez yürütülür. `Do-while` deyimini biçimi aşağıdaki gibidir: ... do deyim while(kosul); // veya do { deyim 1; ... deyim n; } while (kosul); ...

`For` deyimi değerlerin erimi boyunca yinelemek için kullanılır. `For` deyiminin biçimi aşağıdaki gibidir:

```
...
for (ilklendirme;kosul;yineleme) deyim;
// veya
for (ilklendirme;kosul;yineleme){
  deyim 1;
  ...
  deyim n;
};
...
```

Burada *ilklendirme* başlangıçta bir kez yürütülen bir deyimdir. Bu bir değişkenin bildirilmesini ve ilklendirmesini içerebilir veya sadece bir kontrol değişkenini ilklendirebilir. *Yineleme* deyimi takip eden deyimler her yürütüldüğünde, *koşul* anlatımı `yanlış` değerlendirilmedikçe yürütülür.

# KOŞULLU DEYİMLER

Koşullu deyimler hangi deyim ya da deyimlerin verilen koşul anlatımına göre yürütüleceğine karar verilmesini sağlar. Java programlama dilinde üç çeşit döngü deyimi vardır: **eğer (if), eğer-ya da (if-else),** ve **anahtar (switch).**
**If** deyimi bir deyimi ya da deyim öbeğini yürütme yada yürütmeme kararını vermek için kullanılır. Eğer koşul **doğru** ise deyim yürütülür, yoksa yürütme onu takip eden bir sonraki deyimle devam eder. **If** deyiminin biçimi aşağıdaki gibidir:

```
...
if(koşul) deyim;
// veya
if(koşul){
  deyim 1;
  ...
  deyim n;
}
...
```

Koşul anlatımı bir **bool** değer döndürmelidir. **Do-while** deyimi de benzer yürütme yapar.
Bir **if-else** deyimi hangi deyimi ya da deyim öbeğini yürütme kararını vermek için kullanılır. Eğer koşul **doğru** ise birinci deyim ya da deyim öbeği yürütülür, yoksa ikinci deyim ya da deyim öbeği yürütülür. **If-else** deyiminin biçimi aşağıdaki gibidir:

```
if(kosul) deyim1; else deyim2;
// veya
if(kosul){
  deyim 1_1;
  ...
  deyim 1_n;
} else {
  deyim 2_1;
  ...
  deyim 2_m;
}
```

**Else** bölümü başka **if** deyimleri de içerebilir:

```
 if (kosul 1){
  deyim 1_1;
  ...
  deyim 1_n;
} else if (kosul 2){
  deyim 2_1;
  ...
  deyim 2_m;
} else if (kosul 3) {
  deyim 3_1;
  ...
  deyim 3_k;
} else {
```

```
  deyim 4_1;
  ...
  deyim 4_t;
}
```

Bir **switch** deyimi verilen tamsayı veya karakter değerine göre (bir değişkenden veya anlatımdan gelen) yürütülecek bir deyimi ya da deyimleri seçmede kullanılır. **switch** deyiminin biçimi aşağıdaki gibidir:

```
switch (anlatim){
case <deger1>:
  deyim 1;
  break;
case <deger2>:
  deyim 2;
  break;
...
case <deger n>:
  deyim n;
  break;
default:
  deyim n+1;
}
```

Burada *anlatım* bir tamsayı veya karakter değeri üretir ve yürütülecek deyim onun değerine göre belirlenir. Örneğin, eğer onun değeri <değer 2> ye eşitse **case <değer 2>** bölümü **break** deyimine kadar yürütülür. **break**) deyimim **switch** öbeğinin sona ermesine sebep olur ve yürütme **switch** öbeğini takip eden deyimle devam eder. Eğer hiçbir **case <değer i>** satırı eşleşmezse varsayılan (**default**) bölümü yürütülür (istenmezse varsayılan bölümü atlanabilir).
Birçok **case** bölümünü şu şekilde birleştirmek mümkün:

```
switch (anlatim){
case <deger1>:
case <deger2>:
case <deger3>:
  deyim 1;
  break;
case <deger4>:
case <deger5>
  deyim 2;
  break;
...
case <deger n>:
  deyim n;
  break;
default:
  deyim n+1;
}
```

Bu durumda, eğer *anlatim*, <*deger1*>, <*deger2*> veya <*deger3*> den birine eşitse *deyim 1* yürütülür.

# DALLANAN DEYİMLER

Dallanan deyimlerden `break`, ve `continue` deyimleri `while`, `do-while`, ve `for` blokları içindeki deyimlerin yürütülmesini kontrol edebilirler. Bu deyimlerin iki biçimi vardır: düz ve etiketli. Düz biçim en içteki deyim bloğunun yürütülmesini kontrol etmek için kullanılır, ve etiketli biçim yuvalanmış deyim bloklarının yürütülmesini kontrol etmek için kullanılır. Bir deyim bloğuna etiket şöyle verilir:

```
etiket:
akış_kontrolu_deyimi;
```

`break` deyimi eğer düz biçimde kullanılmışsa en içteki bloğu sonlandırır, ve eğer etiketli biçimde kullanılmışsa belirtilen bloğu sonlandırır.Örneğin;

```
...
for_dongusu:
for(int i=0; true; i++) {
  ...
  int j=0;
  while(true){
    j++;
    ...
    if(i==100) break for_dongusu;
    ...
    if(j>=200) break;
    ...
  }
}
```

Bu örnekte `break for_dongusu;` deyimi dıştaki for döngüsünü sonlandırır ve düz `break;` deyimi içteki while döngüsünü sonlandırır (for döngüsü yürütülmeye devam eder).
`continue` deyimi `while`, `do-while`, veya `for` döngüsünün şu anki yinelemesini atlamak için kullanılır. Düz biçimde kullanılır ve en içteki bloğun şu anki yinelemesini atlamak için kullanılır. Eğer etiketli biçimde kullanılırsa belirtilen bloğun o anki yinelemesini atlar. Yürütme, deyim kontrol bölümüyle ve koşullu anlatımın değerine göre devam eder, daha fazla yinelemeler devam edebilir yada etmeyebilir.