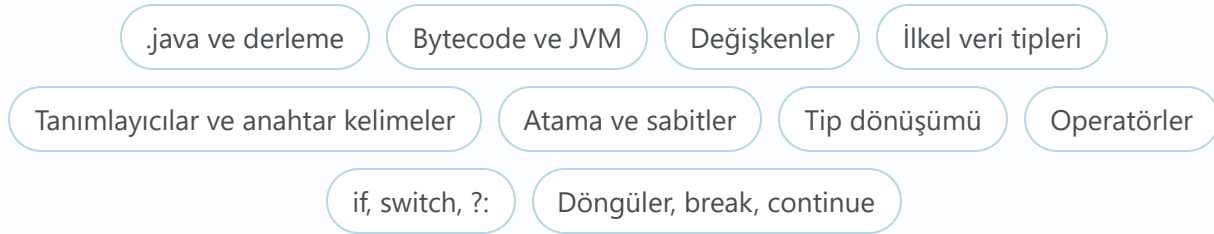


Bölüm 2: Söz Dizimi Kuralları

Java ile Nesne Yönelimli Programlama



Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

Öğrenme Çıktıları

Derleme ve çalıştırma

Bir `.java` dosyasının `javac` ile bytecode'a, `java` ile çalışan programa dönüşmesini anlatmak; JVM'in platform bağımsızlığını nasıl sağladığını açıklamak.

Değişkenler ve tipler

Sekiz ilkel veri tipini boyut ve aralıklarıyla tanımak; değişken bildirmek, değer atamak, taşıma ve ondalık hassasiyet sorunlarını fark etmek.

Kurallar ve adlandırma

Geçerli tanımlayıcıyı geçersizden ayırmak, anahtar kelimeleri tanımak, camelCase / PascalCase / BÜYÜK_HARF kurallarına uymak.

İfadeler ve akış

Operatör önceliğine göre ifadeleri hesaplamak, tip dönüşümü yapmak;

`if` , `switch` , `?:` , `for` , `while` , `do-while` , `break` , `continue` ile kısa programlar yazıp çıktısını önceden bulmak.

Java ile Yazılım Geliştirme

- Java kodu **düz metin** (plain text) dosyalarına yazılır; Word gibi biçimli belgeler kullanılmaz.
- Kaynak kod (source code) dosyalarının uzantısı `.java` 'dır. Derleyici ve geliştirme ortamları dosyayı bu uzantıdan tanır.
- Bir dosyada `public` bir sınıf varsa dosyanın adı sınıfın adıyla **aynı** olmalıdır: `MerhabaDunya`
→ `MerhabaDunya.java` .

Okunaklı

Düz metin insanlar için okunaklıdır, kod kolayca anlaşılır ve paylaşılır.

Taşınabilir

Aynı `.java` dosyası her işletim sisteminde ve her geliştirme ortamında açılır.

Kolay düzeltilir

Hataları düzeltmek, kodu güncellemek için herhangi bir metin düzenleyici yeterlidir.

Dosya adında büyük-küçük harf de önemlidir: `merhabadunya.java` dosyasındaki `public class MerhabaDunya` derlenmez.

Programlama Araçları

Araç	Ne işe yarar?	Örnekler
Metin düzenleyici	Kaynak kodu düz metin olarak yazmak	Notepad++, VS Code
JDK (Java Development Kit)	Derleyici, çalıştırıcı ve diğer geliştirme araçları	<code>javac</code> , <code>java</code> , <code>javap</code> , <code>jshell</code>
IDE (Integrated Development Environment, tümleşik geliştirme ortamı)	Düzenleyici + derleme + çalıştırma + hata ayıklama tek pencerede; yazarken hataları altını çizerek gösterir	Eclipse, IntelliJ IDEA, NetBeans, VS Code (Java eklentileriyle)
JShell (Java 9+)	Tek satırlık Java ifadelerini sınıf yazmadan denemek	<code>jshell> 7 / 2 → 3</code>

Başlangıçta bir kez komut satırında `javac` ve `java` kullanmak, IDE'nin "Çalıştır" düğmesinin arkasında ne olduğunu anlamayı sağlar.

Örnek .java Dosyası: MerhabaDunya.java

```
public class MerhabaDunya {  
    public static void main(String[] args) {  
        System.out.println("Merhaba, Dünya!");  
    }  
}
```

public class MerhabaDunya

Her Java programı en az bir **sınıf** içerir. Sınıfın gövdesi `{ }` süslü parantezleri arasındadır.

public static void main(String[] args)

Programın **giriş noktası**: JVM çalışmaya bu metottan başlar. Yazımı birebir böyle olmalıdır.

System.out.println(...)

Parantez içindeki metni konsola yazar ve alt satıra geçer. Metinler çift tırnak içindedir.

Noktalı virgül ;

Her deyim (statement) `;` ile biter. Süslü parantezle kapanan bloklardan sonra `;` gerekmez.

Temel Söz Dizimi (Syntax) Kuralları

```
/* Çok satırlı açıklama:  
   derleyici bunu yok sayar. */  
public class Kurallar {  
    // Tek satırlık açıklama  
    public static void main(String[] args) {  
        int sayi = 5; // ; ile biter  
        int Sayi = 7; // sayi ≠ Sayi  
        System.out.println(sayi  
                             + Sayi); // satır bölünebilir  
    }  
}
```

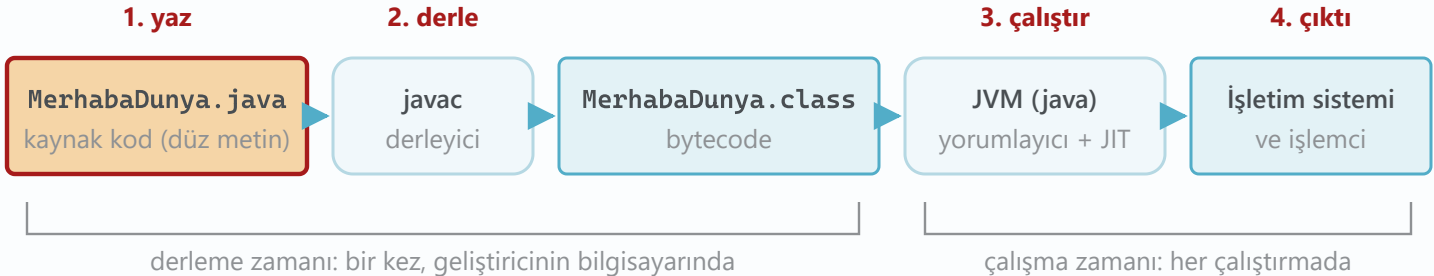
ÇIKTI

12

- **Söz dizimi** (syntax): dilin yazım kuralları. Kurala uymayan kod **derlenmez**.
- Her deyim `;` ile biter.
- Bloklar `{` ile açılır, `}` ile kapanır; her açılan kapanmalıdır.
- Java büyük-küçük harfe duyarlıdır (case-sensitive): `sayi` ≠ `Sayi`, `System` ≠ `system`.
- Boşluk, sekme ve satır sonu derleyici için önemsizdir; okunabilirlik için girinti kullanılır.
- Açıklamalar (comments): `//` satır sonuna kadar, `/* ... */` çok satırlı. Derleyici bunları yok sayar.

Derleme (Compilation)

- Bilgisayar `.java` metnini doğrudan çalıştıramaz; önce **derlenmesi** gerekir. Derleme komutu: `javac`.
- Java derleyicisi kaynak kodu işlemcinin makine diline değil, JVM'in anladığı **bytecode'a** çevirir ve `.class` dosyasına yazar. Bytecode'u çalıştırırken makine koduna çeviren JVM'dir.



Programcı kaynak kodu düz metin olarak yazar ve `MerhabaDunya.java` adıyla kaydeder.

Derleme (Compilation)

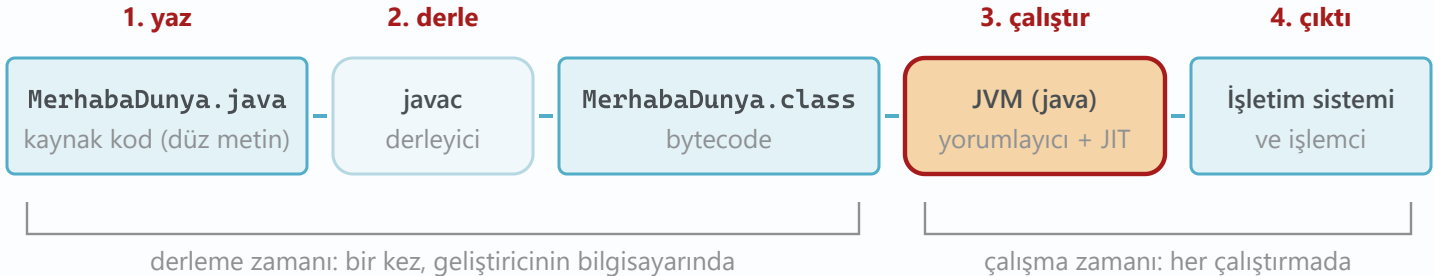
- Bilgisayar `.java` metnini doğrudan çalıştıramaz; önce **derlenmesi** gerekir. Derleme komutu: `javac`.
- Java derleyicisi kaynak kodu işlemcinin makine diline değil, JVM'in anladığı **bytecode'a** çevirir ve `.class` dosyasına yazar. Bytecode'u çalıştırırken makine koduna çeviren JVM'dir.



`javac MerhabaDunya.java` : derleyici söz dizimini ve tipleri denetler; hata yoksa `MerhabaDunya.class` oluşur.

Derleme (Compilation)

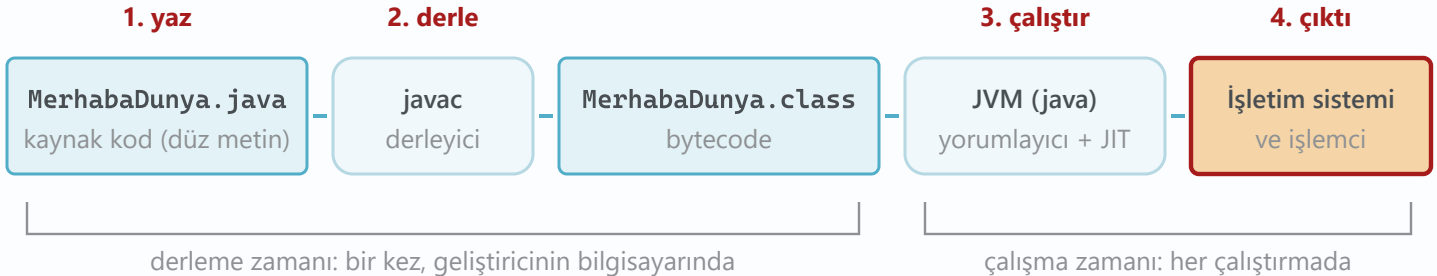
- Bilgisayar `.java` metnini doğrudan çalıştıramaz; önce **derlenmesi** gerekir. Derleme komutu: `javac`.
- Java derleyicisi kaynak kodu işlemcinin makine diline değil, JVM'in anladığı **bytecode'a** çevirir ve `.class` dosyasına yazar. Bytecode'u çalıştırırken makine koduna çeviren JVM'dir.



`java MerhabaDunya` : başlatıcı (launcher) JVM'i açar, sınıfı yükler; JVM bytecode'u yorumlar, sık çalışan kısmı JIT ile makine koduna çevirir.

Derleme (Compilation)

- Bilgisayar `.java` metnini doğrudan çalıştıramaz; önce **derlenmesi** gerekir. Derleme komutu: `javac`.
- Java derleyicisi kaynak kodu işlemcinin makine diline değil, JVM'in anladığı **bytecode'a** çevirir ve `.class` dosyasına yazar. Bytecode'u çalıştırırken makine koduna çeviren JVM'dir.



İşletim sistemi ve işlemci asıl komutları yürütür; ekranda `Merhaba, Dünya!` görünür.

Derleme Neden Önemli?

- `MerhabaDunya.java` derlenince aynı klasörde `MerhabaDunya.class` oluşur.
- Derleyici, program çalışmadan **önce** hataları yakalar: eksik `;`, kapanmamış parantez, yanlış yazılmış ad, uyumsuz tip...
- Hata varsa `.class` dosyası oluşmaz; mesajdaki **dosya adı:satır** bilgisi hatanın yerini gösterir.
- Derleme aşaması ve JVM'in çalışma zamanındaki iyileştirmeleri (JIT) programın daha hızlı çalışmasına katkı sağlar.

3. satırdaki `;` unutulursa:

```
System.out.println("Merhaba, Dünya!")
```

```
MerhabaDunya.java:3: error: ';' expected
    System.out.println("Merhaba, Dünya!")
                        ^
1 error
```

Java ile Temel İşlem Adımları

1. **Derleme:** `.java` dosyası `.class` dosyasına dönüştürülür.
`javac MerhabaDunya.java`
2. **Çalıştırma:** `java` komutuna **uzantısız** sınıf adı verilir.
`java MerhabaDunya`
3. **Çıktı:** program ekrana `Merhaba, Dünya!` yazar.

KOMUT SATIRI

```
> javac MerhabaDunya.java
> dir /b
MerhabaDunya.class
MerhabaDunya.java
> java MerhabaDunya
Merhaba, Dünya!
```

`dir` (Windows) / `ls` (Linux, macOS) klasördeki dosyaları listeler. `javac` bir şey yazmadıysa derleme başarılıdır.



Sık hata: `java MerhabaDunya.class`

Çalıştırırken uzantı yazılmaz; JVM bir **sınıf adı** bekler.

Java 11+ kısayolu

Tek dosyalık programlar `java` `MerhabaDunya.java` ile derlenip doğrudan çalıştırılabilir (diske `.class` yazılmaz).

Bytecode (Ara Kod)

- `.class` dosyaları işlemcinin doğrudan anlayabileceği makine kodu **içermez**.
- İçlerinde Java sanal makinesinin anladığı **bytecode** bulunur: bytecode, JVM'in "makine dilidir".
- Her komutun işlem kodu (opcode) tek bir bayttır (adı buradan gelir); gerekirse ardından işlenen baytlar gelir.
- Bytecode bir işlemciye ya da işletim sistemine bağlı değildir: **aynı** `.class` dosyası Windows, Linux ve macOS'ta çalışır.

Benzetme

Bytecode, her ülkede aynı okunan bir **nota** gibidir; her bilgisayardaki JVM ise o notayı kendi çalgısında çalan müzisyendir.

Görmek için

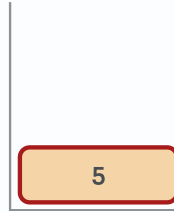
JDK'daki `javap -c Sinif` komutu bir `.class` dosyasındaki bytecode komutlarını okunur biçimde listeler.

Bytecode Örneği: Adım Adım

`int x = 5; System.out.println(x);` satırlarının derlenmiş hâli (`javap -c` , JDK 21).
JVM bir **yığın makinesidir**: komutlar değerleri işlenen yığına iter ve oradan alır.

```
▶ 0: iconst_5
2 1: istore_1
3 2: getstatic    #7    // System.out
4 5: iload_1
5 6: invokevirtual #13    // println(int)
6 9: return
```

işlenen yığını



yerel değişkenler

0: args = dizi

1: x = -

`iconst_5` : sabit 5 işlenen yığına itilir. (Kaynak: `int x = 5; )`

Adım 1 / 6

DEĞİŞKENLER

x = -

yığın = []

ÇIKTI



Bytecode Örneği: Adım Adım

`int x = 5; System.out.println(x);` satırlarının derlenmiş hâli (`javap -c` , JDK 21).
JVM bir **yığın makinesidir**: komutlar değerleri işlenen yığına iter ve oradan alır.

```
1 0: iconst_5
▶ 1: istore_1
3 2: getstatic    #7    // System.out
4 5: iload_1
5 6: invokevirtual #13    // println(int)
6 9: return
```

işlenen yığını



yerel değişkenler

0: args = dizi

1: x = 5

`istore_1` : yığının üstü alınır, 1 numaralı yerel değişkene (`x`) yazılır.

Adım 2 / 6

DEĞİŞKENLER

x = 5

yığın = []

ÇIKTI



Bytecode Örneği: Adım Adım

`int x = 5; System.out.println(x);` satırlarının derlenmiş hâli (`javap -c` , JDK 21).
JVM bir **yığın makinesidir**: komutlar değerleri işlenen yığına iter ve oradan alır.

```
1 0: iconst_5
2 1: istore_1
▶ 2: getstatic    #7    // System.out
4 5: iload_1
5 6: invokevirtual #13   // println(int)
6 9: return
```

işlenen yığını

yerel değişkenler

out

0: args = dizi

1: x = 5

`getstatic #7 : System.out` nesnesine başvuru yığına
itilir. #7, sabitler tablosundaki kayıt numarasıdır.

Adım 3 / 6

DEĞİŞKENLER

x = 5

yığın = [System.out]

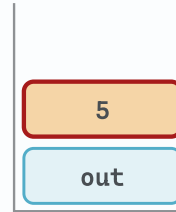
ÇIKTI

Bytecode Örneği: Adım Adım

`int x = 5; System.out.println(x);` satırlarının derlenmiş hâli (`javap -c` , JDK 21).
JVM bir **yığın makinesidir**: komutlar değerleri işlenen yığına iter ve oradan alır.

```
1 0: iconst_5
2 1: istore_1
3 2: getstatic    #7    // System.out
▶ 5: iload_1
5 6: invokevirtual #13    // println(int)
6 9: return
```

işlenen yığını



yerel değişkenler

0: args = dizi

1: x = 5

`iload_1` : `x` 'in değeri (5) yığına kopyalanır. (Kaynak: `System.out.println(x);`)

Adım 4 / 6

DEĞİŞKENLER

x = 5

yığın = [System.out, 5]

ÇIKTI



Bytecode Örneği: Adım Adım

`int x = 5; System.out.println(x);` satırlarının derlenmiş hâli (`javap -c` , JDK 21).
JVM bir **yığın makinesidir**: komutlar değerleri işlenen yığına iter ve oradan alır.

```
1 0: iconst_5
2 1: istore_1
3 2: getstatic    #7    // System.out
4 5: iload_1
5 6: invokevirtual #13   // println(int)
6 9: return
```

işlenen yığını

yerel değişkenler

0: args = dizi

1: x = 5

`invokevirtual #13 : println(int)` çağrılır, iki değer yığından alınır; ekrana **5** yazılır.

Adım 5 / 6

DEĞİŞKENLER

x = 5

yığın = []

ÇIKTI

5

Bytecode Örneği: Adım Adım

`int x = 5; System.out.println(x);` satırlarının derlenmiş hâli (`javap -c` , JDK 21).
JVM bir **yığın makinesidir**: komutlar değerleri işlenen yığına iter ve oradan alır.

```
1 0: iconst_5
2 1: istore_1
3 2: getstatic    #7    // System.out
4 5: iload_1
5 6: invokevirtual #13   // println(int)
▶ 9: return
```

işlenen yığını



yerel değişkenler

0: args = dizi

1: x = 5

`return` : `main` biter, program sonlanır.

Adım 6 / 6

DEĞİŞKENLER

x = 5

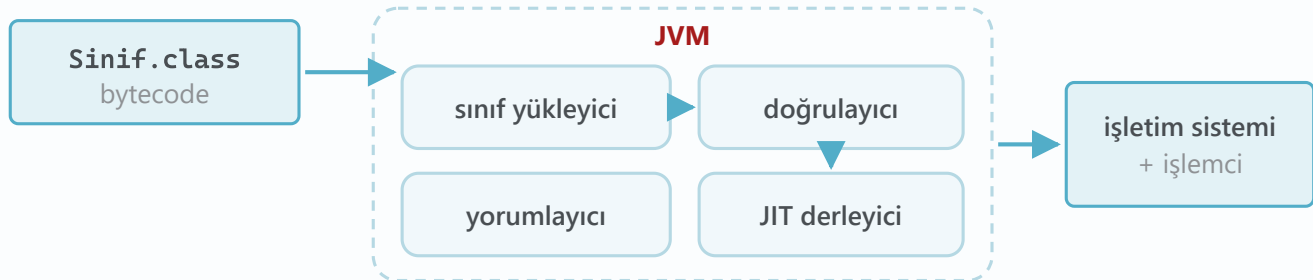
yığın = []

ÇIKTI

5

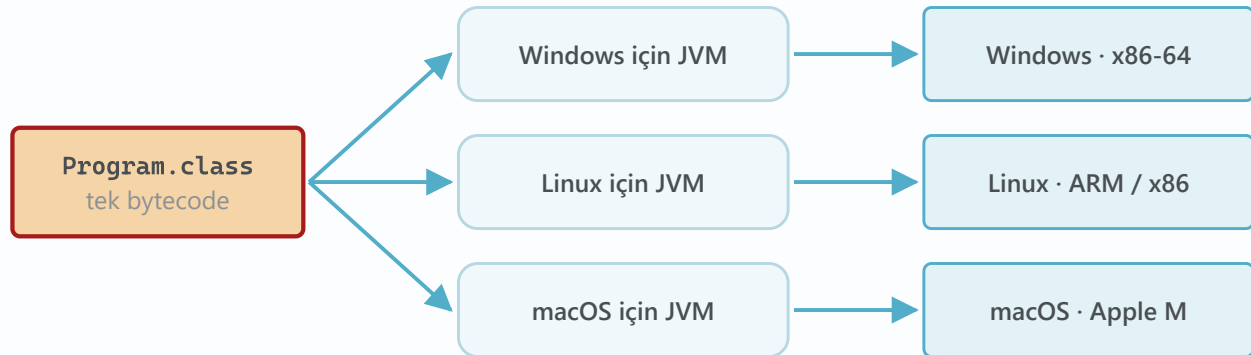
Java Sanal Makinesi (JVM)

- JVM (Java Virtual Machine), Java uygulamalarını çalıştıran temel bileşendir: bytecode'u **yorumlar** ve bulunduğu işlemciye uygun biçimde yürütür.
- Sık çalışan kod parçalarını **JIT** (Just-In-Time) derleyicisi çalışma sırasında makine koduna çevirir; program bu yüzden zamanla hızlanır.
- Java başlatıcı aracı** (launcher tool) `java` komutudur: JVM'i başlatır, sınıfı yükler ve `main` metodunu çağırır.



Platform Bağımsızlık

- Java'nın temel özelliklerinden biridir: **“Bir kez yaz, her yerde çalıştır”** (Write Once, Run Anywhere).
- Program bir kez derlenir; aynı bytecode farklı işletim sistemlerinde çalıştırılır.
- Bunu mümkün kılan, JVM'in birçok farklı işletim sistemi ve işlemci için ayrı ayrı yazılmış olmasıdır.
- C gibi dillerde ise her platform için ayrı derleme gerekir.



Java Platformu

Platform: bir programın çalıştığı donanım ya da yazılım ortamı. Java platformunun iki ana bileşeni vardır:

- **JVM** (Java Virtual Machine): uygulamaları çalıştırır; farklı donanımlara taşınabilirlik sağlar.
- **API** (Application Programming Interface): hazır sınıf kütüphaneleri. Grafik arayüz, veri tabanı, ağ iletişimi, dosya işlemleri gibi birçok işlevi hazır sunar; geliştirmeyi hızlandırır.

Program geliştirmek için **JDK** kurulur. Java 11'den beri JRE ayrı bir paket olarak dağıtılmaz; JDK hepsini içerir.

JDK: geliştirme seti

javac · javap · jshell · jar ...

Çalışma ortamı (JRE)

API (hazır kütüphaneler)

java.lang · java.util · java.io ...

JVM

bytecode'u çalıştırır

Temel Programlama Unsurları

Değişkenler, tipler ve ifadeler

Verileri saklamak ve işlemek için değişkenler gerekir. Her değişkenin bir **tipi** vardır; değerler **ifadelerle** (expressions) hesaplanır: `sayi > 0`, `toplam + 1`.

Kontrol akışı (control flow)

Programın hangi sırayla ve nasıl çalışacağını belirler:

- **Dallanma** (branching): koşula göre farklı kısımları çalıştırır, ör. `if-else`.
- **Döngüler** (loops): bir işlemi tekrarlar, ör. `for`, `while`.

Örnek kod parçası: değişken + koşul + çıktı

```
int sayi = 5;
if (sayi > 0) {
    System.out.println("Sayı pozitif.");
} else {
    System.out.println("Sayı negatif veya sıfır.");
}
```

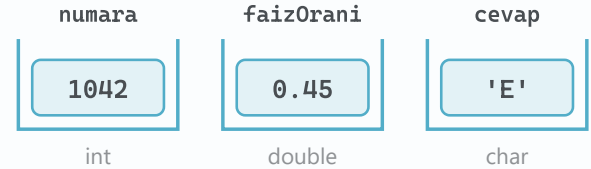
ÇIKTI

Sayı pozitif.

`sayi` 5 olduğu için koşul doğrudur; `else` bloğu atlanır.

Değişkenler: Veri Saklamanın Temel Taşları

- **Değişken** (variable): bellekte bir değeri saklayan, **adı** ve **tipi** olan alandır. Üzerinde ad yazan bir **kutu** gibi düşünülebilir.
- Her değişken kullanılmadan önce **bildirilmelidir** (declaration): önce tip, sonra ad yazılır; bildirim `;` ile biter.
- Aynı tipten birden çok değişken virgülle ayrılarak tek satırda bildirilebilir.
- Adlar açıklayıcı ve anlamlı seçilmelidir: `x` yerine `cekSayisi`.



```
int numara, cekSayisi, mevcutSayisi;  
double miktar, faizOrani;  
char cevap;
```

Söz dizimi: `tür ad;` · `tür ad1, ad2, ad3;` · `tür ad = ilkDeğer;`

Bildirim, Atama, Güncelleme

```
► int yas;  
2 yas = 25;  
3 double boy = 1.78;  
4 yas = yas + 1;  
5 char harf = 'K';  
6 System.out.println(yas + " " + boy + " " +  
    harf);
```

yas

?

int

`int yas;` bildirimini bellekte `yas` adlı bir kutu ayırır; içi henüz **boş** (değer atanmamış).

Adım 1 / 6

DEĞİŞKENLER

yas = ?

ÇIKTI



Bildirim, Atama, Güncelleme

```
1 int yas;  
▶ yas = 25;  
3 double boy = 1.78;  
4 yas = yas + 1;  
5 char harf = 'K';  
6 System.out.println(yas + " " + boy + " " +  
    harf);
```

yas

25

int

Atama: kutuya 25 konur. Artık **yas** okunabilir.

Adım 2 / 6

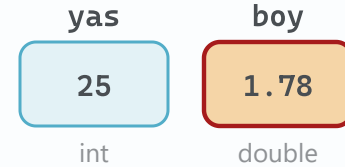
DEĞİŞKENLER

yas = 25

ÇIKTI

Bildirim, Atama, Güncelleme

```
1 int yas;  
2 yas = 25;  
▶ double boy = 1.78;  
4 yas = yas + 1;  
5 char harf = 'K';  
6 System.out.println(yas + " " + boy + " " +  
  harf);
```



Bildirim ve ilk değer verme tek satırda:

```
double boy = 1.78;
```

Adım 3 / 6

DEĞİŞKENLER

yas = 25

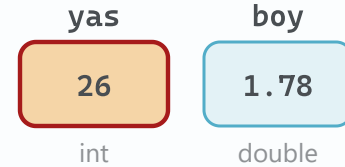
boy = 1.78

ÇIKTI



Bildirim, Atama, Güncelleme

```
1 int yas;  
2 yas = 25;  
3 double boy = 1.78;  
4 ► yas = yas + 1;  
5 char harf = 'K';  
6 System.out.println(yas + " " + boy + " " +  
  harf);
```



Önce sağ taraf hesaplanır: $25 + 1 = 26$; sonuç `yas` 'a yazılır, eski değer kaybolur.

Adım 4 / 6

DEĞİŞKENLER

`yas = 26`

`boy = 1.78`

ÇIKTI

Bildirim, Atama, Güncelleme

```
1 int yas;  
2 yas = 25;  
3 double boy = 1.78;  
4 yas = yas + 1;  
5 ▶ char harf = 'K';  
6 System.out.println(yas + " " + boy + " " +  
    harf);
```

yas

26

int

boy

1.78

double

harf

'K'

char

char değerleri tek tırnak içinde yazılır.

Adım 5 / 6

DEĞİŞKENLER

yas = 26

boy = 1.78

harf = 'K'

ÇIKTI

Bildirim, Atama, Güncelleme

```
1 int yas;  
2 yas = 25;  
3 double boy = 1.78;  
4 yas = yas + 1;  
5 char harf = 'K';  
▶ System.out.println(yas + " " + boy + " " +  
  harf);
```

yas

26

int

boy

1.78

double

harf

'K'

char

Değerler **+** ile metne eklenip yazdırılır.

Adım 6 / 6

DEĞİŞKENLER

yas = 26

boy = 1.78

harf = 'K'

ÇIKTI

26 1.78 K

DEĞİŞKENLER VE VERİ TİPLERİ

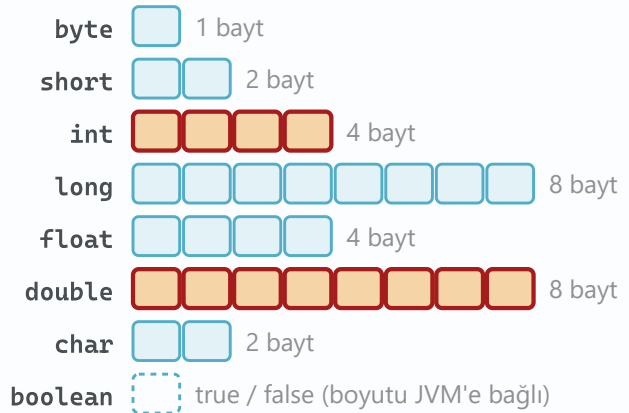
İlkel (Primitive) Veri Tipleri

- Java'da **sekiz** ilkel veri tipi vardır; her birinin kendine özgü **bellek boyutu** ve **değer aralığı** vardır.
- Tam sayı:** `byte`, `short`, `int`, `long`
- Kesirli (ondalıklı) sayı:** `float`, `double`
- Karakter:** `char`
- Mantıksal:** `boolean`

Günlük kullanımda tam sayılar için

`int`, ondalıklı sayılar için `double`

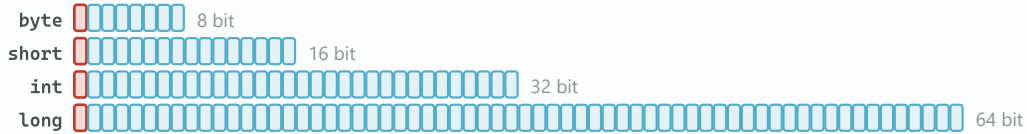
varsayılan seçimdir.



DEĞİŞKENLER VE VERİ TİPLERİ

Tam Sayı (Integer) Tipleri

Tip	Bellek	En küçük değer	En büyük değer	Örnek
<code>byte</code>	1 bayt (8 bit)	-128	127	<code>byte b = 100;</code>
<code>short</code>	2 bayt (16 bit)	-32.768	32.767	<code>short s = 32000;</code>
<code>int</code>	4 bayt (32 bit)	-2.147.483.648	2.147.483.647	<code>int n = 1_000_000;</code>
<code>long</code>	8 bayt (64 bit)	-9.223.372.036.854.775.808	9.223.372.036.854.775.807	<code>long l = 30000000000L;</code>



- Hepsi **işaretlidir** (signed): n bitlik tipin aralığı $-2^{n-1} \dots 2^{n-1} - 1$ 'dir. Sınırlar `Byte.MIN_VALUE` , `Integer.MAX_VALUE` gibi sabitlerle okunur.
- Kodda yazılan tam sayı (ör. `42`) `int` kabul edilir; `int` aralığını aşan bir `long` değerin sonuna `L` eklenir.

Ondalıklı (Floating-point) Sayı Tipleri

Tip	Bellek	En küçük pozitif	En büyük ($\pm$)	Anlamalı basamak
<code>float</code>	4 bayt (32 bit)	$\approx 1,4 \times 10^{-45}$	$\approx 3,4028235 \times 10^{38}$	≈ 7
<code>double</code>	8 bayt (64 bit)	$\approx 4,9 \times 10^{-324}$	$\approx 1,7976931348623157 \times 10^{308}$	$\approx 15-16$

`float` $\pm$ üs 8 kesir (anlamalı basamaklar) 23

`double` $\pm$ üs 11 kesir (anlamalı basamaklar) 52

- Kodda yazılan ondalıklı sayı (ör. `3.14`) `double` kabul edilir; `float` için sonuna `f` eklenir: `3.14f`.
- "En küçük" sütunu, sıfırdan farklı en küçük **pozitif** değerdir (`Float.MIN_VALUE` , `Double.MIN_VALUE`); negatif sınırlar en büyük değer in eksilisidir.
- `double` çift duyarlılıktır (double precision): `float` 'tan hem daha geniş hem daha hassastır.

char ve boolean

char : tek karakter

2 bayt (16 bit), **işaretsiz**. Değer aralığı 0 ... 65.535: her değer bir **Unicode** karakter kodudur. Tek tırnakla yazılır: `'A'`, `'Ş'`, `'7'`.

`'A'` = 65, `'a'` = 97, `'0'` = 48. Kodlar [ASCII tablosunda](#) görülebilir.

```
char harf = 'A';
boolean dogruMu = true;
System.out.println(harf);
System.out.println(dogruMu);
System.out.println(!dogruMu);
```

boolean : mantıksal değer

Yalnızca iki değeri vardır: `true` (doğru) ve `false` (yanlış). Bir bitlik bilgi taşır; bellekte kaç bayt tutacağını JVM belirler.

Koşullarda ve karşılaştırma sonuçlarında kullanılır: `boolean gectiMi = puan >= 50;`

ÇIKTI

```
A
true
false
```

Tiplerin Sınırlarını Program ile Görmek

```
System.out.println("byte : " + Byte.MIN_VALUE + " .. " + Byte.MAX_VALUE);
System.out.println("short : " + Short.MIN_VALUE + " .. " + Short.MAX_VALUE);
System.out.println("int : " + Integer.MIN_VALUE + " .. " + Integer.MAX_VALUE);
System.out.println("long : " + Long.MIN_VALUE + " .. " + Long.MAX_VALUE);
System.out.println("float : " + Float.MIN_VALUE + " .. " + Float.MAX_VALUE);
System.out.println("double: " + Double.MIN_VALUE + " .. " + Double.MAX_VALUE);
System.out.println("char : " + (int) Character.MIN_VALUE + " .. "
                    + (int) Character.MAX_VALUE);
```

ÇIKTI

```
byte : -128 .. 127
short : -32768 .. 32767
int : -2147483648 .. 2147483647
long : -9223372036854775808 .. 9223372036854775807
float : 1.4E-45 .. 3.4028235E38
double: 4.9E-324 .. 1.7976931348623157E308
char : 0 .. 65535
```

4.9E-324 bilimsel gösterimdir: $4,9 \times 10^{-324}$. **Integer**, **Double** gibi sınıflar ilkel tiplerin **sarmalayıcı** (wrapper) sınıflarıdır.

DEĞİŞKENLER VE VERİ TİPLERİ

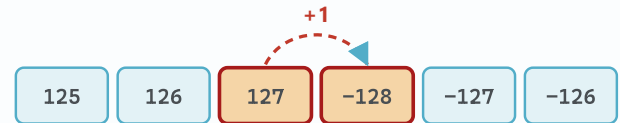
Taşma (Overflow)

```
int enBuyuk = Integer.MAX_VALUE;
System.out.println(enBuyuk);
System.out.println(enBuyuk + 1);    // taşma!
byte sayi1 = 80;
byte sayi2 = 50;
byte toplam = (byte) (sayi1 + sayi2);
System.out.println(toplam);          // 130 değil
long dogru = (long) enBuyuk + 1;
System.out.println(dogru);
```

ÇIKTI

```
2147483647
-2147483648
-126
2147483648
```

- Bir tipin sınırını aşan sonuç **hata vermez**: değer aralığın öbür ucuna “sarar” (saat 12'den sonra 1'e dönmek gibi).
- $80 + 50 = 130$, **byte** sınırı 127 $\rightarrow 130 - 256 = -126$.
- Çözüm: daha geniş bir tip kullanmak (**long**); işlemden **önce** dönüştürmek gerekir.



byte: en büyük değere 1 eklenince en küçüğe “sarar”

b (byte) = 125

0	1	1	1	1	1	0	1
-128	64	32	16	8	4	2	1

Number line: -128, 0, 127

Adım 1 / 8

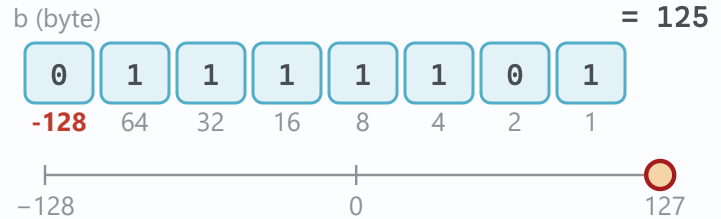
b = 125

ÇIKTI



Taşma: byte Sayacı Adım Adım

```
1 byte b = 125;  
▶ for (int i = 1; i <= 4; i++) {  
3     b++;  
4     System.out.println(b);  
5 }
```



i = 1 : 1 <= 4 doğru, döngü gövdesine girilir.

Adım 2 / 8

DEĞİŞKENLER

b = 125

i = 1

ÇIKTI



b (byte) = 126

0	1	1	1	1	1	1	0
-128	64	32	16	8	4	2	1

-128 0 127

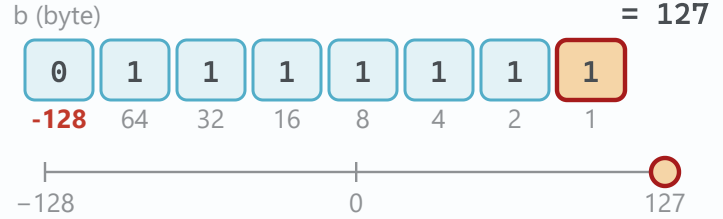
Adım 3 / 8

b = 126 (**i = 1**

126

Taşma: byte Sayacı Adım Adım

```
1 byte b = 125;  
2 for (int i = 1; i <= 4; i++) {  
    ▶ b++;  
    ▶ System.out.println(b);  
5 }
```



b++ → 127 (01111111): byte için en büyük değer.
Yazdırılır.

Adım 4 / 8

DEĞİŞKENLER

b = 127

i = 2

ÇIKTI

126

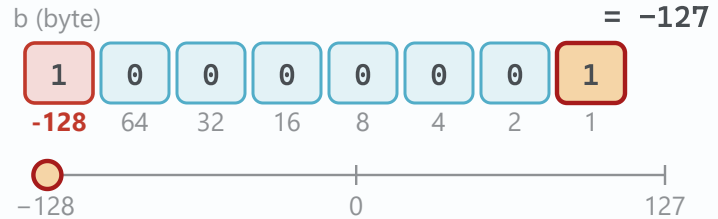
127

ÇIKTI

ÇIKTI

Taşma: byte Sayacı Adım Adım

```
1 byte b = 125;  
2 for (int i = 1; i <= 4; i++) {  
    ▶ b++;  
    ▶ System.out.println(b);  
5 }
```



b++ → -127 (10000001): sayım artık -128'den yukarı doğru sürer.

Adım 7 / 8

DEĞİŞKENLER

b = -127

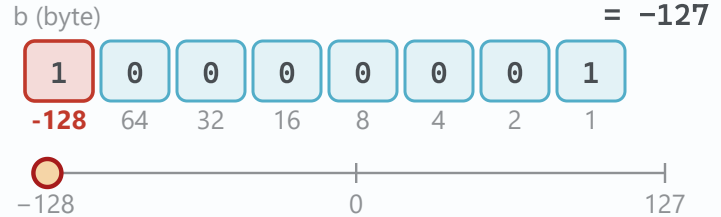
i = 4

ÇIKTI

```
126  
127  
-128  
-127
```

Taşma: byte Sayacı Adım Adım

```
1 byte b = 125;  
▶ for (int i = 1; i <= 4; i++) {  
3     b++;  
4     System.out.println(b);  
5 }
```



i = 5 : 5 <= 4 yanlış, döngü biter. Çıktı 126, 127, -128, -127.

Adım 8 / 8

DEĞİŞKENLER

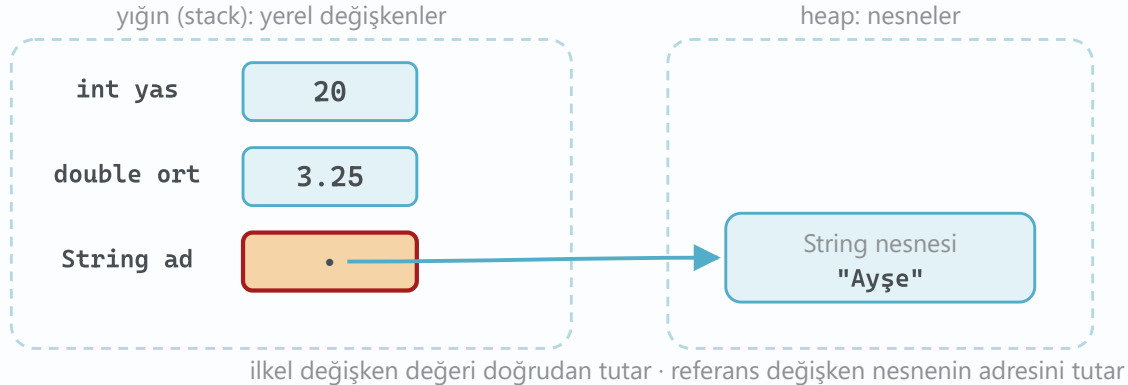
b = -127

i = 5

ÇIKTI

```
126  
127  
-128  
-127
```

İlkel Tipler ve Referans Tipleri



- **İlkel tipler** (8 tane): değerin kendisini saklar; adları küçük harfle başlar.
- **Referans tipleri**: bir **nesneyi** gösterir; değişkende nesnenin kendisi değil, yeri (başvurusu) tutulur.
- Örnekler: `String` (karakter dizisi), diziler, sınıflar, arayüzler, `enum` 'lar. Nesneler genellikle `new` ile oluşturulur.
- Hiçbir nesneyi göstermeyen referansın değeri `null` 'dır.

Tanımlayıcılar (Identifiers)

- Değişkenlere, sınıflara, metotlara verilen adlara **tanımlayıcı** denir. Tanımlayıcılar özel kurallara tabidir:
- Yalnızca **harf**, **rakam** (0–9), alt çizgi `_` ve dolar işareti `$` içerebilir. Boşluk, `-`, `@`, `.` gibi karakterler olmaz.
- İlk karakter harf, `_` ya da `$` olmalıdır; **rakamla başlayamaz**.
- Anahtar kelimeler ve `true`, `false`, `null` tanımlayıcı olamaz.
- Uzunluk sınırı yoktur.
- Java büyük-küçük harfe duyarlıdır: `kisiAdi` ile `kisiadi` **farklı** değişkenlerdir.

Geçerli örnekler

`kullaniciAdi`, `toplamPuan`,
`ogrenci_adi`, `veril`, `_sayac`

İpucu

Java harf olarak Unicode harflerini de kabul eder (`öğrenci` derlenir), ama farklı sistemlerde sorun çıkmaması için adlarda yalnızca İngilizce harfler kullanılması önerilir.

Hangi Tanımlayıcılar Geçerli?

Ad		Neden?
ogrenciNo	✓	harf ile başlıyor
sayi_2	✓	harf, rakam ve _
\$fiyat	✓	\$ izinli (ama önerilmez)
2yas	✗	rakamla başlıyor
maas hesabi	✗	boşluk içeriyor
ad-soyad	✗	- çıkarma operatörüdür
@isim	✗	@ izinli değil
class	✗	anahtar kelime
Class	✓	büyük C: anahtar kelime değil

Rakamla başlayan bir ad derleyiciyi şaşırtır; hata mesajı adı değil, deyimini işaret eder:

```
int 2yas = 20;
```

Tanim.java:3: error: not a statement

```
    int 2yas = 20;
      ^
```

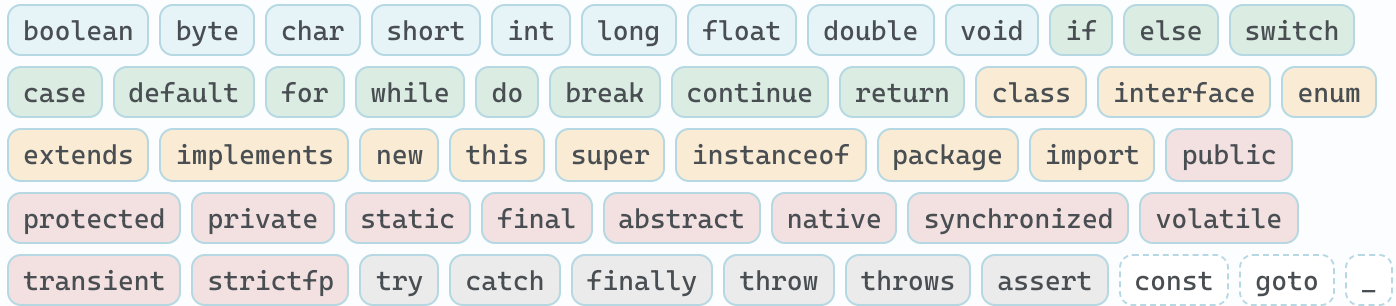
Tanim.java:3: error: ';' expected

```
    int 2yas = 20;
      ^
```

2 errors

Anahtar Kelimeler (Reserved Words)

- Dilde belirli görevleri olan, **ayrılmış** kelimelerdir. Hepsi küçük harfle yazılır.
- Değişken, sınıf ya da metod adı olarak **kullanılamazlar**. Java 21'de 51 anahtar kelime vardır:



□ tipler □ kontrol akışı □ sınıf ve nesne □ düzenleyiciler □ istisnalar □ ayrılmış, kullanılmıyor

`true`, `false`, `null` anahtar kelime değil **sabit değerdir** (literal), ama onlar da ad olamaz. `var` (Java 10+), `record` (16+), `yield` (14+) gibi **bağlamsal** kelimeler yalnızca belirli yerlerde özel anlam taşır.

Anahtar Kelimeler: Veri Tipleri

Kelime	Görevi
boolean	Yalnızca iki değer: <code>true</code> ve <code>false</code> . Koşullarda ve mantıksal işlemlerde kullanılır.
byte	8 bit (1 bayt) işaretli tam sayı; küçük tam sayılar için.
short	16 bit tam sayı; <code>int</code> 'ten daha küçük sayılar için, daha az bellek kullanır.
int	32 bit tam sayı; hesaplamalarda varsayılan tam sayı tipi.
long	64 bit tam sayı; <code>int</code> 'in yetersiz kaldığı büyük tam sayılar için.
float	32 bit tek duyarlıklı ondalıklı sayı; <code>double</code> 'dan daha az hassas.
double	64 bit çift duyarlıklı ondalıklı sayı; büyük ve hassas ondalıklı değerler için.
char	16 bit tek bir Unicode karakter.
void	Metodun değer döndürmediğini belirtir: <code>public static void main</code> .

Anahtar Kelimeler: Kontrol Akışı

Kelime	Görevi
if / else	Koşulu denetler; doğruysa <code>if</code> bloğu, değilse <code>else</code> bloğu çalışır.
switch	Bir değişkenin farklı değerlerine göre farklı işlemler yapar.
case	<code>switch</code> içinde bir durumu (değeri) temsil eder.
default	<code>switch</code> içinde hiçbir durum eşleşmediğinde yapılacak işlem.
for	Bir işlemi belirli bir koşul altında, sayaçla tekrarlar.
while	Koşul sağlandığı sürece işlemi tekrarlar.
do	<code>do-while</code> döngüsü: gövde en az bir kez çalışır, koşul sonda denetlenir.
break	Döngüden ya da <code>switch</code> bloğundan hemen çıkar.
continue	Döngünün o turunun kalanını atlar, bir sonraki tura geçer.
return	Metodu bitirir; gerekiyorsa bir sonuç döndürür.

Anahtar Kelimeler: Sınıflar ve Nesneler

Kelime	Görevi
class	Sınıf tanımlar. Sınıflar nesnelerin şablonudur; Java programlarının temel yapı taşıdır.
interface	Arayüz tanımlar: bir sınıfın sağlaması gereken metotları belirleyen sözleşme.
enum	Sabit değerlerden oluşan bir tip: <code>enum Renk { KIRMIZI, YESIL }</code> .
extends	Kalıtım (inheritance): bir sınıfın başka bir sınıfın alan ve metotlarını kalıtımla alması.
implements	Bir sınıfın bir arayüzü gerçekleştirmesi; arayüzün metotlarını sağlar.
new	Nesne oluşturur: bellekte yer ayırır ve ilk değerleri verir.
this	Nesnenin kendisine başvuru; aynı adlı alan ile yerel değişkeni ayırt eder.
super	Üst sınıfın metotlarına, alanlarına ve yapıcısına erişir.
instanceof	Bir nesnenin belirli bir sınıf ya da arayüz tipinden olup olmadığını denetler.
package / import	<code>package</code> sınıfın paketini (isim uzayını) belirler; <code>import</code> başka paketteki sınıfları kullanıma açar.

Anahtar Kelimeler: Düzenleyiciler (Modifiers)

Kelime	Görevi
public	Her yerden, tüm sınıflardan erişilebilir.
protected	Aynı paketten ve alt sınıflardan erişilebilir.
private	Yalnızca aynı sınıfın içinden erişilebilir.
static	Üye nesneye değil sınıfa aittir; tüm nesneler arasında paylaşılır.
final	Değişken: değeri değişmez · metot: alt sınıfta yeniden yazılamaz · sınıf: kalıtılamaz.
abstract	Soyut sınıf (nesnesi oluşturulamaz) ya da gövdesiz metot; alt sınıf metodu gerçekleştirmelidir.
native	Gövdesi Java dışında, platforma özgü kodla (C/C++) yazılmış metot; JNI ile bağlanır.
synchronized	Çoklu iş parçacığında (multithreading) aynı anda tek iş parçacığının girmesini sağlar.
volatile	Değişkenin her iş parçacığı tarafından her zaman güncel değeriyle okunmasını sağlar.
transient	Nesne serileştirilirken (kaydedilirken) bu alanın dikkate alınmamasını sağlar.
strictfp	Ondalıklı işlemlerin her platformda aynı sonucu vermesi. Java 17'den beri bu zaten varsayılan; kelimenin etkisi kalmadı.

Anahtar Kelimeler: İstisnalar ve Diğerleri

Kelime	Görevi
try	Hata verebilecek kodu çevreler; oluşan istisnalar (exceptions) yakalanabilir.
catch	<code>try</code> bloğunda oluşan istisnayı yakalar ve işler.
finally	<code>try-catch</code> sonunda her durumda çalışan blok (ör. dosyayı kapatmak).
throw	Bir istisna nesnesi fırlatır; normal akış kesilir.
throws	Metot başlığında, metodun fırlatabileceği istisna türlerini listeler.
assert	Bir koşulun doğru olduğunu denetler; hata ayıklama ve doğrulamada kullanılır (<code>java -ea</code> ile etkin).
const, goto	Ayrılmıştır ama hiçbir işlevi yoktur; ad olarak kullanılamaz.
_	Java 9'dan beri tek başına alt çizgi ad olamaz (Java 22+ ile "kullanılmayan değişken" anlamı kazandı).

```
int new = 5;
```

```
Anahtar.java:3: error: not a statement
    int new = 5;
    ^
... (toplam 4 hata)
```

İsimlendirme Kuralları (Naming Conventions)

Ne?	Kural	Örnekler
Sınıf ve arayüz tipleri	PascalCase, büyük harfle başlar	<code>String</code> , <code>Ogrenci</code> , <code>MerhabaDunya</code>
Değişken, alan, metot	camelCase, küçük harfle başlar	<code>firstName</code> , <code>classAverage</code> , <code>hesapla()</code>
İlkel veri tipleri	küçük harf (anahtar kelimedir)	<code>int</code> , <code>float</code> , <code>boolean</code>
İsimlendirilmiş sabit	BÜYÜK_HARF, kelimeler <code>_</code> ile	<code>PI</code> , <code>GUNLER_HAFTADA</code> , <code>MAX_SAYI</code>
Paket	tamamen küçük harf	<code>syntax</code> , <code>tr.edu.giresun</code>

camelCase: ilk kelime küçük, sonrakiler büyük harfle başlar (deve hörgücü).

PascalCase: her kelime büyük harfle başlar. Bunlar derleyici kuralı değil, **uzlaşım**dır: uyulmazsa kod derlenir ama okuması zorlaşır.

```
class OgrenciKaydi { }    // sınıf
String isimSoyisim;      // değişken
boolean evliMi;          // evet/hayır
final int MAX_SAYI = 100; // sabit
```

Atama İfadeleri

- Değişkene değer vermek için `=` **atama operatörü** kullanılır: `değişken = ifade;`
- Önce **sağ taraftaki** ifade hesaplanır, sonuç **sol taraftaki** değişkene yazılır.
- İfade bir sabit değer, başka bir değişken ya da bir işlem olabilir.
- Matematikteki eşitlik değildir: `x = x + 1` "x'in yeni değeri, eski değerinin bir fazlası" demektir.
- Sol tarafta yalnızca bir değişken olabilir: `5 = sayi;` derlenmez.

```
int sayi, sonuc;  
int sayi1 = 4, sayi2 = 6;  
String isim;
```

```
sayi = 10; // sabit değer  
sonuc = sayi1 + sayi2; // toplam: 10  
isim = "Ahmet"; // metin  
sayi = sayi * 2; // 10 → 20
```

sayi = sayi * 2;



Atamada tipler uyumlu olmalıdır:

String değişkene sayı, **int** değişkene metin atanamaz.

Bileşik Atama Operatörleri: İzleme

`+=` `-=` `*=` `/=` `%=` : `x op= y` ↔ `x = x op y` . Kısa yazılır, aynı işi yapar.

```
► int x = 5;  
2 int y = 10;  
3 x = x + y;  
4 x -= y;  
5 x *= 2;  
6 x /= 4;  
7 x %= 3;  
8 System.out.println("x = " + x + ", y = " + y);
```

x
5
int

x 5 ile başlar.

Adım 1 / 8

DEĞİŞKENLER

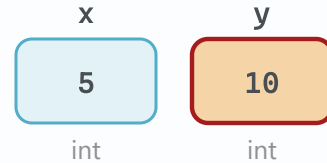
x = 5

ÇIKTI

Bileşik Atama Operatörleri: İzleme

`+=` `-=` `*=` `/=` `%=` : `x op= y` ↔ `x = x op y` . Kısa yazılır, aynı işi yapar.

```
1 int x = 5;  
▶ int y = 10;  
3 x = x + y;  
4 x -= y;  
5 x *= 2;  
6 x /= 4;  
7 x %= 3;  
8 System.out.println("x = " + x + ", y = " + y);
```



y 10 ile başlar.

Adım 2 / 8

DEĞİŞKENLER

x = 5

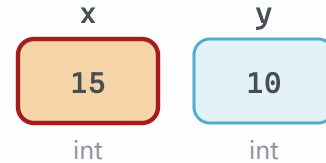
y = 10

ÇIKTI

Bileşik Atama Operatörleri: İzleme

`+=` `-=` `*=` `/=` `%=` : `x op= y` ↔ `x = x op y` . Kısa yazılır, aynı işi yapar.

```
1 int x = 5;  
2 int y = 10;  
▶ x = x + y;  
4 x -= y;  
5 x *= 2;  
6 x /= 4;  
7 x %= 3;  
8 System.out.println("x = " + x + ", y = " + y);
```



Sağ taraf: $5 + 10 = 15 \rightarrow x = 15$.

Adım 3 / 8

DEĞİŞKENLER

`x = 15`

`y = 10`

ÇIKTI



Bileşik Atama Operatörleri: İzleme

`+=` `-=` `*=` `/=` `%=` : `x op= y` ↔ `x = x op y` . Kısa yazılır, aynı işi yapar.

```
1 int x = 5;
2 int y = 10;
3 x = x + y;
4 ► x -= y;
5 x *= 2;
6 x /= 4;
7 x %= 3;
8 System.out.println("x = " + x + ", y = " + y);
```



`x -= y` kısaltması `x = x - y` demektir:
 $15 - 10 = 5$.

Adım 4 / 8

DEĞİŞKENLER

`x = 5`

`y = 10`

ÇIKTI

Bileşik Atama Operatörleri: İzleme

`+=` `-=` `*=` `/=` `%=` : `x op= y` ↔ `x = x op y` . Kısa yazılır, aynı işi yapar.

```
1 int x = 5;
2 int y = 10;
3 x = x + y;
4 x -= y;
5 ▶ x *= 2;
6 x /= 4;
7 x %= 3;
8 System.out.println("x = " + x + ", y = " + y);
```



`x = x * 2` : $5 \times 2 = 10$.

Adım 5 / 8

DEĞİŞKENLER

x = 10

y = 10

ÇIKTI



Bileşik Atama Operatörleri: İzleme

`+=` `-=` `*=` `/=` `%=` : `x op= y` ↔ `x = x op y` . Kısa yazılır, aynı işi yapar.

```
1 int x = 5;
2 int y = 10;
3 x = x + y;
4 x -= y;
5 x *= 2;
6 x /= 4;
7 x %= 3;
8 System.out.println("x = " + x + ", y = " + y);
```



`x = x / 4` : $10 / 4 = 2$ (tam sayı bölmesi, kesir atılır).

Adım 6 / 8

DEĞİŞKENLER

`x = 2`

`y = 10`

ÇIKTI

Bileşik Atama Operatörleri: İzleme

`+=` `-=` `*=` `/=` `%=` : `x op= y` ↔ `x = x op y` . Kısa yazılır, aynı işi yapar.

```
1 int x = 5;
2 int y = 10;
3 x = x + y;
4 x -= y;
5 x *= 2;
6 x /= 4;
7 x %= 3;
8 System.out.println("x = " + x + ", y = " + y);
```



`x = x % 3` : 2'nin 3'e bölümünden kalan = 2.

Adım 7 / 8

DEĞİŞKENLER

x = 2

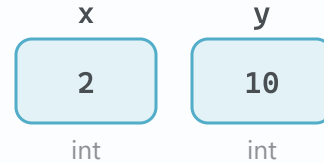
y = 10

ÇIKTI

Bileşik Atama Operatörleri: İzleme

`+=` `-=` `*=` `/=` `%=` : `x op= y` ↔ `x = x op y` . Kısa yazılır, aynı işi yapar.

```
1 int x = 5;
2 int y = 10;
3 x = x + y;
4 x -= y;
5 x *= 2;
6 x /= 4;
7 x %= 3;
▶ System.out.println("x = " + x + ", y = " + y);
```



Son değerler yazdırılır.

Adım 8 / 8

DEĞİŞKENLER

x = 2

y = 10

ÇIKTI

x = 2, y = 10

Değişkenlere İlk Değer Verme

- Bir değişken değer atanmadan bildirilebilir; ilk değer bildirimle birlikte de verilebilir: `int yas = 25;`
- Yerel değişkenler** (metot içindikiler) varsayılan değer **almaz**: değer atanmadan okunursa program derlenmez.
- Sınıf **alanları** otomatik varsayılan değer alır: sayılar 0, `boolean` `false`, `char` `'\u0000'`; referans tipleri (ör. `String`) `null`.
- Yine de varsayılan değerlere güvenmemek, değeri açıkça vermek iyi bir alışkanlıktır.

```
int sayac;  
System.out.println(sayac);
```

```
Ilk.java:4: error: variable sayac  
might not have been initialized
```

Alan tipi	Varsayılan
<code>int</code> , <code>long</code> , ...	<code>0</code>
<code>double</code> , <code>float</code>	<code>0.0</code>
<code>char</code>	<code>'\u0000'</code> (kod 0)
<code>boolean</code>	<code>false</code>
<code>String</code> ve diğer referanslar	<code>null</code>

Sabit Değerler (Literals)

Kodda doğrudan yazılan `2`, `3.7`, `'y'`, `"Ali"` gibi değerlere **sabit değer** (literal) denir.

- **Tam sayı:** `+` ya da `-` ile başlayabilir, **virgöl içeremez:** `+42`, `-17`. Okunabilirlik için alt çizgi kullanılabilir (Java 7+): `1_000_000`. Onaltılık `0x1F`, ikilik `0b1010`.
- **Ondalıklı:** nokta ile yazılır, `e` gösterimi de olur: `2.5e3` = $2,5 \times 10^3$, `7.68e-2` = $7,68 \times 10^{-2}$.
- **Karakter:** `'y'`; özel karakterler: `'\n'` yeni satır, `'\t'` sekme, `'\'`, `'\"'`.
- **Metin:** `"Merhaba"` · **mantıksal:** `true`, `false`

```
System.out.println(2.5e3);
System.out.println(7.68e-2);
System.out.println(1_000_000);
System.out.println(0x1F);
System.out.println(0b1010);
System.out.println(3000000000L);
System.out.println("Ali\tVeli\n\"Java\"");
```

ÇIKTI

```
2500.0
0.0768
1000000
31
10
3000000000
Ali      Veli
"Java"
```

Ondalık Sayıların Hassaslığı

- Ondalık sayılar sınırlı sayıda bitle, **ikili** (binary) tabanda saklanır.
- $1/3$ ya da 0,1 gibi değerler ikili tabanda sonsuz basamaklıdır, bu yüzden **yaklaşık** saklanır: `1.0/3.0` tam olarak $1/3$ değildir.
- Küçük yuvarlama hataları birikebilir: 0,1'i on kez toplamak tam 1 etmez. Bazen hatalar birbirini götürür: üç tane `1.0/3.0` 'ın toplamı `1.0` yazdırır.
- Bu yüzden ondalıklı sayılar `==` ile karşılaştırılmaz; farkın çok küçük olup olmadığına bakılır. Para hesabında `BigDecimal` kullanılır.

```
System.out.println(1.0 / 3.0);
System.out.println(1.0/3 + 1.0/3 + 1.0/3);
System.out.println(0.1 + 0.2);
System.out.println(0.1 + 0.2 == 0.3);
double t = 0.0;
for (int i = 0; i < 10; i++) t += 0.1;
System.out.println(t);
double fark = (0.1 + 0.2) - 0.3;
System.out.println(Math.abs(fark) < 1e-9);
```

ÇIKTI

```
0.3333333333333333
1.0
0.30000000000000004
false
0.9999999999999999
true
```

İsimlendirilmiş Sabitler (Named Constants)

- `final` ile bildirilen değişkenin değeri bir kez verilir, sonra **değiştirilemez**. Kalıp: `public static final Tür SABİT_ADI = değer;`
- Adlar BÜYÜK harfle, kelimeler `_` ile ayrılarak yazılır. Kodda "sihirli sayılar" yerine anlamlı ad kullanılır: `7` yerine `GUNLER_HAFTADA` ; değer değişirse tek yerden güncellenir.

```
public class Sabitler {  
    public static final double PI = 3.14159;  
    public static final int GUNLER_HAFTADA = 7;  
  
    public static void main(String[] args) {  
        double r = 2.0;  
        double alan = PI * r * r;  
        System.out.println("Alan: " + alan);  
        int gun = 4 * GUNLER_HAFTADA;  
        System.out.println(gun + " gün");  
    }  
}
```

ÇIKTI

```
Alan: 12.56636  
28 gün
```

Sabiti değiştirmeye çalışmak ya da

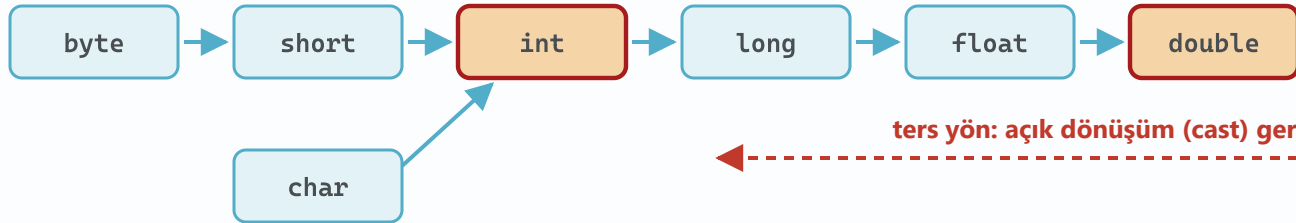
`double` sonucu `float` 'a atamak derlenmez:

```
PI = 3.14;  
float alan = PI * r * r;
```

```
error: cannot assign a value to  
      static final variable PI  
error: incompatible types: possible  
      lossy conversion from double to float
```

Atama Uyumluluğu

örtük (otomatik) genişletme →



- Bir değer, zincirde **daha sağdaki** bir tipin değişkenine doğrudan atanabilir (genişletme, widening).
- `char` değeri `int` 'e (ve sağındakilere) atanabilir; tersi derlenmez.
- `long` → `float` izinlidir ama çok büyük sayılarda bazı basamaklar yuvarlanabilir.

```
char karakter = 'A';
int tamsayi;
tamsayi = karakter;    // mümkün: 65
karakter = tamsayi;    // hata
```

```
Uyum.java:5: error: incompatible types:
    possible lossy conversion from int to char
    karakter = tamsayi;
                ^
```

Veri Tipi Dönüşümü (Casting)

Örtük dönüşüm (implicit)

Küçük tipten büyük tipe Java **kendiliğinden** dönüştürür; veri kaybı olmaz.

Açık dönüşüm (explicit)

Büyük tipten küçük tipe (daraltma, narrowing) dönüşüm, hedef tip **parantez içinde** yazılarak istenir: `(int) d`.

Programcı olası kaybı kabul etmiş olur:

- ondalıktan tam sayıya: kesirli kısım **atılır** (yuvarlanmaz),
- sığmayan tam sayı: taşar (`(byte) 130` → -126).

```
double ondalikDegisken = 5;    // örtük: 5.0
System.out.println(ondalikDegisken);
double d = 5.0;
int tamSayiDegisken = (int) d; // açık: 5
System.out.println(tamSayiDegisken);
System.out.println((int) 3.99);
System.out.println((int) -3.99);
System.out.println(Math.round(3.99));
System.out.println((byte) 130);
```

ÇIKTI

```
5.0
5
3
-3
4
-126
```

Tam Sayı Bölmesi ve Dönüşümün Yeri

İfade	Sonuç	Neden?
<code>7 / 2</code>	<code>3</code>	iki <code>int</code> → tam sayı bölmesi, kesir atılır
<code>7 / 2.0</code>	<code>3.5</code>	<code>2.0</code> <code>double</code> → işlem <code>double</code> yapılır
<code>(double) 7 / 2</code>	<code>3.5</code>	önce <code>7</code> → <code>7.0</code> , sonra bölme
<code>(double) (7 / 2)</code>	<code>3.0</code>	önce tam sayı bölmesi (<code>3</code>), sonra <code>3</code> → <code>3.0</code> : geç kalınmış dönüşüm
<code>(int) 3.99</code>	<code>3</code>	kesirli kısım atılır
<code>Math.round(3.99)</code>	<code>4</code>	en yakın tam sayıya yuvarlar (<code>long</code> döndürür)

İki `int` arasındaki işlemin sonucu `int` 'tir. Bir işlenen `double` ise diğeri de `double` 'a yükseltilir (sayısal yükseltme, numeric promotion). `byte` ve `short` değerler işleme girerken önce `int` 'e yükseltilir: `byte + byte` sonucu `int` 'tir.

Tip Dönüşümü: Kodu Adım Adım İzle

```
► int a = 9, b = 4;  
2 double d1 = a / b;  
3 double d2 = (double) a / b;  
4 double d3 = (double) (a / b);  
5 int k = (int) 9.75;  
6 long r = Math.round(9.75);  
7 byte y = (byte) 200;  
8 System.out.println(d1 + " " + d2 +  
  " " + d3);  
9 System.out.println(k + " " + r + "  
  " + y);
```

a

9

int

b

4

int

İki `int` değişken: `a = 9`, `b = 4`.

Adım 1 / 9

DEĞİŞKENLER

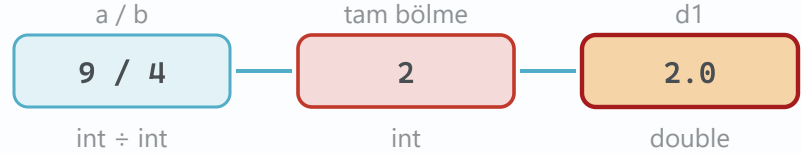
a = 9

b = 4

ÇIKTI

Tip Dönüşümü: Kodu Adım Adım İzle

```
1 int a = 9, b = 4;  
2 double d1 = a / b;  
3 double d2 = (double) a / b;  
4 double d3 = (double) (a / b);  
5 int k = (int) 9.75;  
6 long r = Math.round(9.75);  
7 byte y = (byte) 200;  
8 System.out.println(d1 + " " + d2 +  
  " " + d3);  
9 System.out.println(k + " " + r + "  
  " + y);
```



İki `int` bölünür: sonuç **2** (kesir atılır). Dönüşüm **sonra** olur: $2 \rightarrow 2.0$. Hedefin `double` olması bölmeyi kurtarmaz.

Adım 2 / 9

DEĞİŞKENLER

a = 9

b = 4

d1 = 2.0

ÇIKTI

Tip Dönüşümü: Kodu Adım Adım İzle

```
1 int a = 9, b = 4;
2 double d1 = a / b;
3 ▶ double d2 = (double) a / b;
4 double d3 = (double) (a / b);
5 int k = (int) 9.75;
6 long r = Math.round(9.75);
7 byte y = (byte) 200;
8 System.out.println(d1 + " " + d2 +
  " " + d3);
9 System.out.println(k + " " + r + "
  " + y);
```



Önce `(double) a` = 9.0; bir işlenen `double` olduğu için `b` de 4.0'a yükseltir: $9.0 / 4.0 = 2.25$.

Adım 3 / 9

DEĞİŞKENLER

a = 9

b = 4

d1 = 2.0

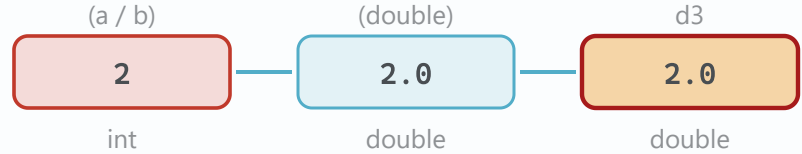
d2 = 2.25

ÇIKTI

```
2.0 2.25 2.0
9 4 200
```

Tip Dönüşümü: Kodu Adım Adım İzle

```
1 int a = 9, b = 4;  
2 double d1 = a / b;  
3 double d2 = (double) a / b;  
▶ 4 double d3 = (double) (a / b);  
5 int k = (int) 9.75;  
6 long r = Math.round(9.75);  
7 byte y = (byte) 200;  
8 System.out.println(d1 + " " + d2 +  
  " " + d3);  
9 System.out.println(k + " " + r + "  
  " + y);
```



Parantez önce hesaplanır: `a / b` = 2; ardından `2` → `2.0`.
Geç kalınmış dönüşüm.

Adım 4 / 9

DEĞİŞKENLER

`a = 9`

`b = 4`

`d1 = 2.0`

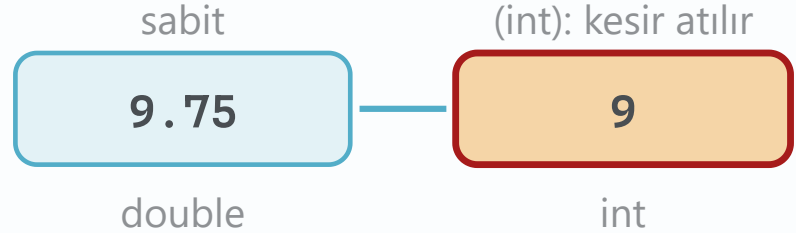
`d2 = 2.25`

`d3 = 2.0`

ÇIKTI

Tip Dönüşümü: Kodu Adım Adım İzle

```
1 int a = 9, b = 4;  
2 double d1 = a / b;  
3 double d2 = (double) a / b;  
4 double d3 = (double) (a / b);  
▶ int k = (int) 9.75;  
6 long r = Math.round(9.75);  
7 byte y = (byte) 200;  
8 System.out.println(d1 + " " + d2 +  
  " " + d3);  
9 System.out.println(k + " " + r + "  
  " + y);
```



(int) 9.75 : kesirli kısım **atılır**, yuvarlanmaz → 9.

Adım 5 / 9

DEĞİŞKENLER

d1 = 2.0

d2 = 2.25

d3 = 2.0

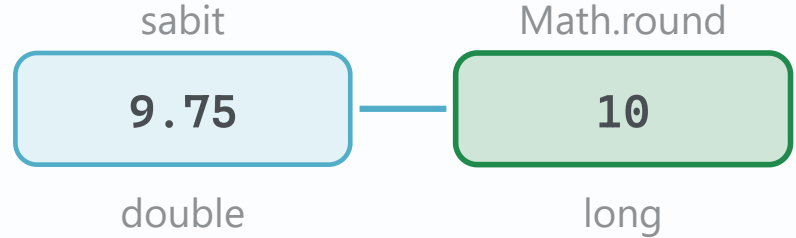
k = 9

ÇIKTI

```
2.0 2.25 2.0  
9 200 200
```

Tip Dönüşümü: Kodu Adım Adım İzle

```
1 int a = 9, b = 4;
2 double d1 = a / b;
3 double d2 = (double) a / b;
4 double d3 = (double) (a / b);
5 int k = (int) 9.75;
▶ long r = Math.round(9.75);
7 byte y = (byte) 200;
8 System.out.println(d1 + " " + d2 +
  " " + d3);
9 System.out.println(k + " " + r + "
  " + y);
```



`Math.round(9.75)` en yakın tam sayıyı verir: **10**
(`long` tipinde).

Adım 6 / 9

DEĞİŞKENLER

d1 = 2.0

d2 = 2.25

d3 = 2.0

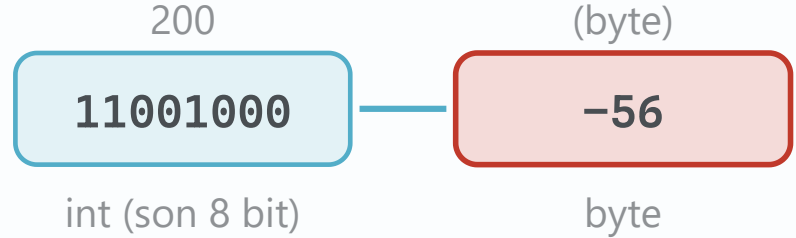
k = 9

r = 10

ÇIKTI

Tip Dönüşümü: Kodu Adım Adım İzle

```
1 int a = 9, b = 4;  
2 double d1 = a / b;  
3 double d2 = (double) a / b;  
4 double d3 = (double) (a / b);  
5 int k = (int) 9.75;  
6 long r = Math.round(9.75);  
7 byte y = (byte) 200;  
8 System.out.println(d1 + " " + d2 +  
  " " + d3);  
9 System.out.println(k + " " + r + "  
  " + y);
```



200 `byte` 'a sığmaz: son 8 bit 11001000 alınır; işaret biti 1 → $200 - 256 = -56$.

Adım 7 / 9

DEĞİŞKENLER

d1 = 2.0

d2 = 2.25

d3 = 2.0

k = 9

r = 10

y = -56

ÇIKTI

Tip Dönüşümü: Kodu Adım Adım İzle

```
1 int a = 9, b = 4;  
2 double d1 = a / b;  
3 double d2 = (double) a / b;  
4 double d3 = (double) (a / b);  
5 int k = (int) 9.75;  
6 long r = Math.round(9.75);  
7 byte y = (byte) 200;  
► System.out.println(d1 + " " + d2 +  
  " " + d3);  
9 System.out.println(k + " " + r + "  
  " + y);
```

d1	d2	d3	k	r	y
2.0	2.25	2.0	9	10	-56
double	double	double	int	long	byte
d1	d2	d3	yazdırılır: 2.0 2.25 2.0 .		

Adım 8 / 9

DEĞİŞKENLER

d1 = 2.0

d2 = 2.25

d3 = 2.0

k = 9

r = 10

y = -56

ÇIKTI

2.0 2.25 2.0

Tip Dönüşümü: Kodu Adım Adım İzle

```
1 int a = 9, b = 4;  
2 double d1 = a / b;  
3 double d2 = (double) a / b;  
4 double d3 = (double) (a / b);  
5 int k = (int) 9.75;  
6 long r = Math.round(9.75);  
7 byte y = (byte) 200;  
8 System.out.println(d1 + " " + d2 +  
  " " + d3);  
► System.out.println(k + " " + r + "  
  " + y);
```

d1	d2	d3	k	r	y
2.0	2.25	2.0	9	10	-56
double	double	double	int	long	byte
k	r	y	yazdırılır: 9 10 -56		

Adım 9 / 9

DEĞİŞKENLER

d1 = 2.0

d2 = 2.25

d3 = 2.0

k = 9

r = 10

y = -56

ÇIKTI

2.0 2.25 2.0

9 10 -56

char ↔ int Dönüşümü

- Her `char` aslında bir sayıdır: karakterin Unicode kodu. `'A'` = 65, `'B'` = 66, `'a'` = 97.
- `char` → `int` ötürür; `int` → `char` için `(char)` gerekir.
- İki `char` + ile toplanırsa **sayı** çıkar; metin elde etmek için önce bir `String` ile başlanır.
- Büyük ve küçük İngilizce harfler arasında 32 fark vardır.

Karakter kodlarının tamamı: [ASCII tablosu](#).

```
char karakter = 'A';
int kod = karakter;           // 65
System.out.println(kod);
char sonraki = (char) (kod + 1);
System.out.println(sonraki);
System.out.println('B' + 'C');
System.out.println("" + 'B' + 'C');
System.out.println((char) (karakter + 32));
System.out.println((int) 'ş');
```

ÇIKTI

```
65
B
133
BC
a
351
```

char ↔ int: Kodu Adım Adım İzle

```
► char c = 'd';  
2 int kod = c;  
3 char buyuk = (char) (c - 32);  
4 int fark = 'z' - c;  
5 char sonraki = c;  
6 sonraki++;  
7 System.out.println(kod + " " + buyuk +  
  " " + fark + " " + sonraki);  
8 System.out.println(c + 1);  
9 System.out.println("'" + c + 1);
```

c

'd'

char · 100

c bir char : içinde 'd' karakterinin Unicode kodu **100** saklanır.

Adım 1 / 8

DEĞİŞKENLER

c = 'd'

ÇIKTI

char ↔ int: Kodu Adım Adım İzle

```
1 char c = 'd';  
2 int kod = c;  
3 char buyuk = (char) (c - 32);  
4 int fark = 'z' - c;  
5 char sonraki = c;  
6 sonraki++;  
7 System.out.println(kod + " " + buyuk +  
  " " + fark + " " + sonraki);  
8 System.out.println(c + 1);  
9 System.out.println("'" + c + 1);
```

c	kod
'd'	100
char · 100	int

char → int örtük dönüşümdür: kod = 100 .
Dönüşüm işareti gerekmez.

Adım 2 / 8

DEĞİŞKENLER

c = 'd' kod = 100

ÇIKTI

char ↔ int: Kodu Adım Adım İzle

```
1 char c = 'd';
2 int kod = c;
3 ► char buyuk = (char) (c - 32);
4 int fark = 'z' - c;
5 char sonraki = c;
6 sonraki++;
7 System.out.println(kod + " " + buyuk +
  " " + fark + " " + sonraki);
8 System.out.println(c + 1);
9 System.out.println("'" + c + 1);
```

c	kod	buyuk
'd'	100	'D'
char · 100	int	char · 68

c - 32 bir int ifadesidir: 100 - 32 = 68.
int 'ten char'a geçmek için (char) gerekir:
68 = 'D'.

Adım 3 / 8

DEĞİŞKENLER

c = 'd'

kod = 100

buyuk = 'D'

ÇIKTI

char ↔ int: Kodu Adım Adım İzle

```
1 char c = 'd';
2 int kod = c;
3 char buyuk = (char) (c - 32);
4 int fark = 'z' - c;
5 char sonraki = c;
6 sonraki++;
7 System.out.println(kod + " " + buyuk +
  " " + fark + " " + sonraki);
8 System.out.println(c + 1);
9 System.out.println("'" + c + 1);
```

c	kod	buyuk	fark
'd'	100	'D'	22
char · 100	int	char · 68	int

İki `char` çıkarılınca `int` elde edilir: `'z' - 'd'`
= 122 - 100 = 22.

Adım 4 / 8

DEĞİŞKENLER

c = 'd'

kod = 100

buyuk = 'D'

fark = 22

ÇIKTI

char ↔ int: Kodu Adım Adım İzle

```
1 char c = 'd';
2 int kod = c;
3 char buyuk = (char) (c - 32);
4 int fark = 'z' - c;
5 ▶ char sonraki = c;
6 ▶ sonraki++;
7 System.out.println(kod + " " + buyuk +
8   " " + fark + " " + sonraki);
9 System.out.println(c + 1);
10 System.out.println("'" + c + 1);
```

c	kod	buyuk	fark	sonraki
'd'	100	'D'	22	'e'
char · 100	int	char · 68	int	char · 101

sonraki önce 'd' olur, sonraki++ ile 101 = 'e'. ++, char değişkende dönüşüm işareti istemez.

Adım 5 / 8

DEĞİŞKENLER

c = 'd'

kod = 100

buyuk = 'D'

fark = 22

sonraki = 'e'

ÇIKTI

```
100 D 22 e
d
'd'
```

char ↔ int: Kodu Adım Adım İzle

```
1 char c = 'd';
2 int kod = c;
3 char büyük = (char) (c - 32);
4 int fark = 'z' - c;
5 char sonraki = c;
6 sonraki++;
7 System.out.println(kod + " " + büyük +
8 " " + fark + " " + sonraki);
9 System.out.println(c + 1);
10 System.out.println("'" + c + 1);
```

c	kod	buyuk	fark	sonraki
'd'	100	'D'	22	'e'
char · 100	int	char · 68	int	char · 101

Birleştirmede " " metni olduğu için değerler metne eklenir: 100 D 22 e .

Adım 6 / 8

DEĞİŞKENLER

c = 'd'

kod = 100

buyuk = 'D'

fark = 22

sonraki = 'e'

ÇIKTI

100 D 22 e

char ↔ int: Kodu Adım Adım İzle

```
1 char c = 'd';
2 int kod = c;
3 char büyük = (char) (c - 32);
4 int fark = 'z' - c;
5 char sonraki = c;
6 sonraki++;
7 System.out.println(kod + " " + büyük +
  " " + fark + " " + sonraki);
8 ▶ System.out.println(c + 1);
9 System.out.println("" + c + 1);
```

c	kod	buyuk	fark	sonraki
'd'	100	'D'	22	'e'
char · 100	int	char · 68	int	char · 101

$c + 1 \rightarrow 100 + 1 = 101$ (int)

$c + 1$: char sayıya yükseltilir, $100 + 1 = 101$ yazılır (harf değil).

Adım 7 / 8

DEĞİŞKENLER

c = 'd'

kod = 100

buyuk = 'D'

fark = 22

sonraki = 'e'

ÇIKTI

```
100 D 22 e
101
```

char ↔ int: Kodu Adım Adım İzle

```
1 char c = 'd';
2 int kod = c;
3 char buyuk = (char) (c - 32);
4 int fark = 'z' - c;
5 char sonraki = c;
6 sonraki++;
7 System.out.println(kod + " " + buyuk +
  " " + fark + " " + sonraki);
8 System.out.println(c + 1);
▶ System.out.println("'" + c + 1);
```

c	kod	buyuk	fark	sonraki
'd'	100	'D'	22	'e'
char · 100	int	char · 68	int	char · 101

"" + c + 1 → "d" + 1 → "d1"

"" + c önce "d" metnini verir; ardından + 1
metne eklenir → **d1**.

Adım 8 / 8

DEĞİŞKENLER

c = 'd'

kod = 100

buyuk = 'D'

fark = 22

sonraki = 'e'

ÇIKTI

```
100 D 22 e
101
d1
```

OPERATÖRLER

Aritmetik Operatörler

Operatör	Anlamı	$x = 17, y = 5$	Sonuç
+	toplama	$x + y$	22
-	çıkarma	$x - y$	12
*	çarpma	$x * y$	85
/	bölme	x / y	3
%	kalan (mod)	$x \% y$	2
		$-17 / 5$	-3
		$-17 \% 5$	-2
		$17.0 / 5$	3.4

- / iki tam sayı arasında **tam sayı bölmesi** yapar.
- % (mod) bölümden kalanı verir; işareti bölünenin işaretidir.
- Tek/çift testi: $n \% 2 == 0$ ise çift.
- Son basamak: $n \% 10$; son basamağı atmak: $n / 10$.

○○○○○ ○○○○○ ○○○○○ ○○

$17 / 5 = 3$ (tam grup) $17 \% 5 = 2$

beşerli 3 grup, artan 2

+ bir işleneni String ise **birleştirme** (concatenation) yapar:

"Yaş: " + 20 → Yaş: 20 .

Artırma (++) ve Azaltma (--): İzleme

`x++` ve `++x` ikisi de `x` 'i 1 artırır; fark, ifadenin **değerindedir**: öncek eski değeri, önek yeni değeri verir. Tek başına `i++;` yazıldığında fark yoktur.

```
► int a = 5;  
2 int b = a++;  
3 System.out.println(a + " " + b);  
4 int c = ++a;  
5 System.out.println(a + " " + c);  
6 int d = a--;  
7 System.out.println(a + " " + d);
```

a
5
int

a 5 ile başlar.

Adım 1 / 7

DEĞİŞKENLER

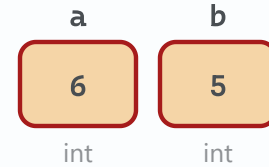
a = 5

ÇIKTI

Artırma (++) ve Azaltma (--): İzleme

`x++` ve `++x` ikisi de `x` 'i 1 artırır; fark, ifadenin **değerindedir**: önce eski değeri, önce yeni değeri verir. Tek başına `i++;` yazıldığında fark yoktur.

```
1 int a = 5;  
▶ int b = a++;  
3 System.out.println(a + " " + b);  
4 int c = ++a;  
5 System.out.println(a + " " + c);  
6 int d = a--;  
7 System.out.println(a + " " + d);
```



Sonek `a++` : önce eski değeri (5) kullanılır → `b` = 5 , **sonra** `a` 6 olur.

Adım 2 / 7

DEĞİŞKENLER

a = 6 b = 5

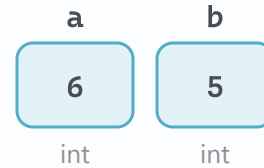
ÇIKTI



Artırma (++) ve Azaltma (--): İzleme

`x++` ve `++x` ikisi de `x` 'i 1 artırır; fark, ifadenin **değerindedir**: önceki eski değeri, önceki yeni değeri verir. Tek başına `i++;` yazıldığında fark yoktur.

```
1 int a = 5;  
2 int b = a++;  
▶ System.out.println(a + " " + b);  
4 int c = ++a;  
5 System.out.println(a + " " + c);  
6 int d = a--;  
7 System.out.println(a + " " + d);
```



Yazdırılır: `6 5`.

Adım 3 / 7

DEĞİŞKENLER

`a = 6` `b = 5`

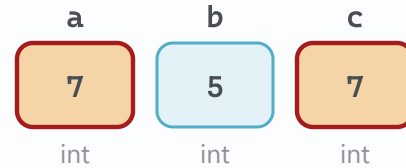
ÇIKTI

`6 5`

Artırma (++) ve Azaltma (--): İzleme

`x++` ve `++x` ikisi de `x` 'i 1 artırır; fark, ifadenin **değerindedir**: önce eski değeri, önek yeni değeri verir. Tek başına `i++;` yazıldığında fark yoktur.

```
1 int a = 5;
2 int b = a++;
3 System.out.println(a + " " + b);
▶ 4 int c = ++a;
5 System.out.println(a + " " + c);
6 int d = a--;
7 System.out.println(a + " " + d);
```



Önek ++a : önce `a` 7 olur, sonra yeni değeri kullanılır → `c = 7`.

Adım 4 / 7

DEĞİŞKENLER

`a = 7` `b = 5` `c = 7`

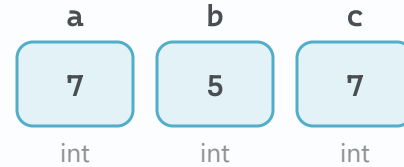
ÇIKTI

6 5

Artırma (++) ve Azaltma (--): İzleme

`x++` ve `++x` ikisi de `x` 'i 1 artırır; fark, ifadenin **değerindedir**: önce eski değeri, önek yeni değeri verir. Tek başına `i++;` yazıldığında fark yoktur.

```
1 int a = 5;  
2 int b = a++;  
3 System.out.println(a + " " + b);  
4 int c = ++a;  
5 System.out.println(a + " " + c);  
6 int d = a--;  
7 System.out.println(a + " " + d);
```



Yazdırılır: `7 7`.

Adım 5 / 7

DEĞİŞKENLER

a = 7 b = 5 c = 7

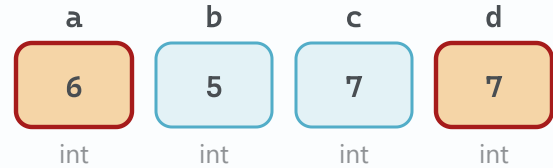
ÇIKTI

```
6 5  
7 7
```

Artırma (++) ve Azaltma (--): İzleme

`x++` ve `++x` ikisi de `x` 'i 1 artırır; fark, ifadenin **değerindedir**: önce eski değeri, önek yeni değeri verir. Tek başına `i++;` yazıldığında fark yoktur.

```
1 int a = 5;
2 int b = a++;
3 System.out.println(a + " " + b);
4 int c = ++a;
5 System.out.println(a + " " + c);
▶ int d = a--;
7 System.out.println(a + " " + d);
```



`a--` : önce eski değer (7) → `d = 7` , sonra `a` 6 olur.

Adım 6 / 7

DEĞİŞKENLER

a = 6 b = 5 c = 7 d = 7

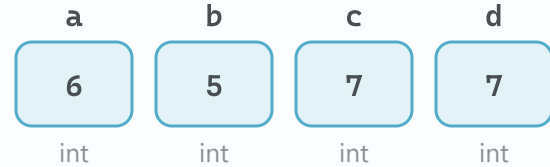
ÇIKTI

```
6 5
7 7
```

Artırma (++) ve Azaltma (--): İzleme

`x++` ve `++x` ikisi de `x` 'i 1 artırır; fark, ifadenin **değerindedir**: önceki eski değeri, önceki yeni değeri verir. Tek başına `i++;` yazıldığında fark yoktur.

```
1 int a = 5;  
2 int b = a++;  
3 System.out.println(a + " " + b);  
4 int c = ++a;  
5 System.out.println(a + " " + c);  
6 int d = a--;  
▶ System.out.println(a + " " + d);
```



Yazdırılır: `6 7`.

Adım 7 / 7

DEĞİŞKENLER

a = 6 b = 5 c = 7 d = 7

ÇIKTI

```
6 5  
7 7  
6 7
```

OPERATÖRLER

Operatörler ve Öncelik Sırası

Öncelik	Operatörler	Yön
1 (en yüksek)	sonek <code>x++</code> <code>x--</code>	→
2	tekli: <code>++x</code> <code>--x</code> <code>!</code> <code>++x</code> <code>--x</code> · tip dönüşümü <code>(int)</code>	←
3	çarpma, bölme, mod: <code>*</code> <code>/</code> <code>%</code>	→
4	toplama, çıkarma: <code>+</code> <code>-</code>	→
5	karşılaştırma: <code><</code> <code>></code> <code><=</code> <code>>=</code>	→
6	eşitlik: <code>==</code> <code>!=</code>	→
7	mantıksal VE: <code>&&</code>	→
8	mantıksal VEYA: <code> </code>	→
9	üçlü: <code>?:</code>	←
10 (en düşük)	atama: <code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> ...	←

Eşit öncelikte **ikili** operatörlerde soldaki önce (→): `10 - 4 - 3` = 3. **Tekli** operatörlerde ve atamada sağdaki önce (←): `a = b = 0`. Sırayı değiştirmek ya da okunurluğu artırmak için **parantez** kullanılır.

OPERATÖRLER

Bir İfadenin Adım Adım Hesaplanması

```
System.out.println(2 + 3 * 4 - 6 / 2);
```

$$2 + 3 * 4 - 6 / 2$$

`*` ve `/`, `+` ve `-` 'den önce gelir; ikisinden **soldaki** önce: $3 * 4 = 12$.

Adım 1 / 5

$$(2 + 3) * 4$$

$$= 5 * 4 = 20$$

$$10 - 4 - 3$$

$$= 6 - 3 = 3 \text{ (soldan)}$$

$$20 / 4 * 2$$

$$= 5 * 2 = 10 \text{ (soldan)}$$

$$-2 * -3$$

$$\text{tekli eksi önce: } 6$$

OPERATÖRLER

Bir İfadenin Adım Adım Hesaplanması

```
System.out.println(2 + 3 * 4 - 6 / 2);
```

$$2 + 12 - 6 / 2$$

Sırada kalan yüksek öncelikli işlem: $6 / 2 = 3$.

Adım 2 / 5

$$(2 + 3) * 4$$

$$= 5 * 4 = 20$$

$$10 - 4 - 3$$

$$= 6 - 3 = 3 \text{ (soldan)}$$

$$20 / 4 * 2$$

$$= 5 * 2 = 10 \text{ (soldan)}$$

$$-2 * -3$$

tekli eksi önce: 6

OPERATÖRLER

Bir İfadenin Adım Adım Hesaplanması

```
System.out.println(2 + 3 * 4 - 6 / 2);
```

$$2 + 12 - 3$$

Kalan $+$ ve $-$ eşit öncelikli $\rightarrow$ soldan sağa: $2 + 12 = 14$.

Adım 3 / 5

$$(2 + 3) * 4$$

$$= 5 * 4 = 20$$

$$10 - 4 - 3$$

$$= 6 - 3 = 3 \text{ (soldan)}$$

$$20 / 4 * 2$$

$$= 5 * 2 = 10 \text{ (soldan)}$$

$$-2 * -3$$

tekli eksi önce: 6

OPERATÖRLER

Bir İfadenin Adım Adım Hesaplanması

```
System.out.println(2 + 3 * 4 - 6 / 2);
```

14 - 3

14 - 3 = 11.

Adım 4 / 5

(2 + 3) * 4

= 5 * 4 = **20**

10 - 4 - 3

= 6 - 3 = **3** (soldan)

20 / 4 * 2

= 5 * 2 = **10** (soldan)

-2 * -3

tekli eksi önce: **6**

OPERATÖRLER

Bir İfadenin Adım Adım Hesaplanması

```
System.out.println(2 + 3 * 4 - 6 / 2);
```

11

Sonuç **11**. Parantezle sıra değişir: $(2 + 3) * 4 - 6 / 2 = 20 - 3 = 17$.

Adım 5 / 5

$(2 + 3) * 4$

$= 5 * 4 = 20$

$10 - 4 - 3$

$= 6 - 3 = 3$ (soldan)

$20 / 4 * 2$

$= 5 * 2 = 10$ (soldan)

$-2 * -3$

tekli eksi önce: **6**

OPERATÖRLER

Karşılaştırma (İlişkisel) Operatörleri

Operatör	Anlamı	x = 10, y = 20
<code>==</code>	eşit mi?	<code>x == y</code> → <code>false</code>
<code>!=</code>	eşit değil mi?	<code>x != y</code> → <code>true</code>
<code>></code>	soldaki büyük mü?	<code>y > x</code> → <code>true</code>
<code><</code>	soldaki küçük mü?	<code>x < y</code> → <code>true</code>
<code>>=</code>	büyük veya eşit mi?	<code>x >= 30</code> → <code>false</code>
<code><=</code>	küçük veya eşit mi?	<code>x <= y</code> → <code>true</code>

- Değerler arasındaki ilişkiyi değerlendirir; sonuç her zaman bir `boolean` 'dır: `true` ya da `false`.
- Sonuç bir değişkende saklanabilir: `boolean buyukMu = y > x;`
- `=` atamadır, `==` karşılaştırmadır. `if (x = 5)` derlenmez:

error: incompatible types: int cannot be converted to boolean

OPERATÖRLER

Mantıksal Operatörler

- **VE (AND)** `&&` : iki koşul da `true` ise `true` . `A && B` yalnızca A ve B doğruyken doğrudur.
- **VEYA (OR)** `||` : en az biri `true` ise `true` .
- **DEĞİL (NOT)** `!` : değeri tersine çevirir:
`!true` → `false` .

A	B	A && B	A B	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

```
int x = 10, y = 20, z = 30;  
System.out.println(x < y && y < z);  
System.out.println(x > y || y < z);  
System.out.println(!(x < y));  
int yas = 20;  
boolean ehliyet = true;  
System.out.println(yas >= 18 && ehliyet);
```

ÇIKTI

```
true  
true  
false  
true
```

Kısa Devre (Short-circuit) Değerlendirme

- `&&` 'de sol taraf `false` ise sonuç zaten `false` 'tur: sağ taraf **hiç hesaplanmaz**.
- `||` 'de sol taraf `true` ise sağ taraf hesaplanmaz.
- Bu, tehlikeli bir işlemi korumak için kullanılır: önce "bölen sıfır değil mi?" diye bakılır, ancak öyleyse bölme yapılır.
- Sıra önemlidir: `sayi / bolen > 1 && bolen != 0` yazılıyorsa program sıfıra bölme hatasıyla (`ArithmeticException`) dururdu.

```
int bolen = 0;
int sayi = 10;
if (bolen != 0 && sayi / bolen > 1) {
    System.out.println("Bölüm 1'den büyük");
} else {
    System.out.println("Bölünemez ya da küçük");
}
```

ÇIKTI

Bölünemez ya da küçük

KONTROL AKIŞI

"if" Deyimi

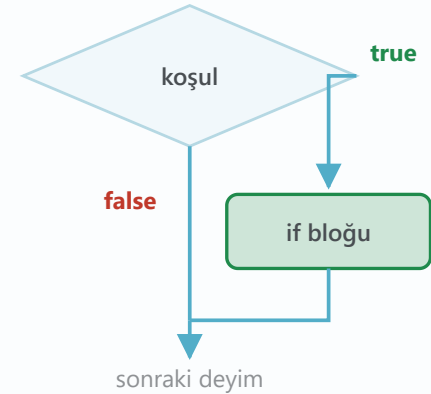
Bir koşulu değerlendirir; koşul `true` ise bloğun içindeki kodu çalıştırır, değilse bloğu atlar. Koşul mutlaka `boolean` olmalıdır.

```
if (koşul) {  
    // Koşul doğruysa buradaki kod çalışır.  
}
```

```
int puan = 72;  
if (puan >= 50) {  
    System.out.println("Geçti");  
}  
System.out.println("Bitti");
```

ÇIKTI

```
Geçti  
Bitti
```



KONTROL AKIŞI

"if-else" Deyimi

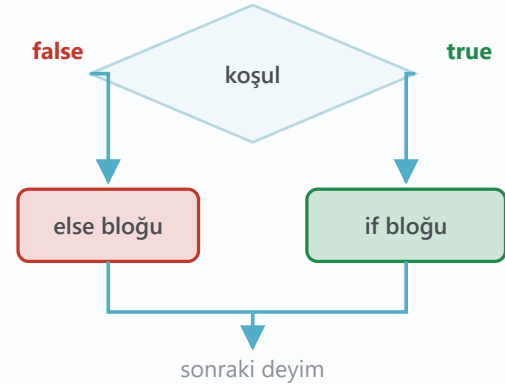
Koşul doğru ise `if` bloğu, aksi takdirde `else` bloğu çalışır: iki bloktan **tam olarak biri** çalışır.

```
if (koşul) {  
    // Koşul doğruysa buradaki kod çalışır.  
} else {  
    // Koşul yanlışsa buradaki kod çalışır.  
}
```

```
int sayi = 7;  
if (sayi % 2 == 0) {  
    System.out.println(sayi + " çift");  
} else {  
    System.out.println(sayi + " tek");  
}
```

ÇIKTI

7 tek



"switch" Deyimi

Bir değerin durumlarına (case) göre farklı kod bloklarını çalıştırır. Değer `int`, `char`, `String` ya da `enum` olabilir (`double`, `boolean` olamaz).

```
switch (değer) {  
    case durum1:  
        // Durum 1 için kod  
        break;  
    case durum2:  
        // Durum 2 için kod  
        break;  
    default:  
        // Hiçbir durum uymuyorsa  
}
```

- `break` o durumun sonunu belirtir; `switch` 'ten çıkılır.
- `default` hiçbir durum uymadığında çalışır (isteğe bağlı).

```
int gun = 3;  
switch (gun) {  
    case 1:  
        System.out.println("Pazartesi");  
        break;  
    case 2:  
        System.out.println("Salı");  
        break;  
    case 3:  
        System.out.println("Çarşamba");  
        break;  
    default:  
        System.out.println("Diğer gün");  
}
```

ÇIKTI

Çarşamba

"break" Unutulursa: Düşme (Fall-through)

```
► int gun = 2;  
2 switch (gun) {  
3     case 1:  
4         System.out.println("Pazartesi");  
5     case 2:  
6         System.out.println("Salı");  
7     case 3:  
8         System.out.println("Çarşamba");  
9     default:  
10        System.out.println("Diğer gün");  
11 }
```

case 1

case 2

case 3

default

gun = 2 .

Adım 1 / 6

DEĞİŞKENLER

gun = 2

ÇIKTI



"break" Unutulursa: Düşme (Fall-through)

```
1 int gun = 2;
2 switch (gun) {
3     case 1:
4         System.out.println("Pazartesi");
5     case 2:
6         System.out.println("Salı");
7     case 3:
8         System.out.println("Çarşamba");
9     default:
10        System.out.println("Diğer gün");
11 }
```

case 1

case 2

case 3

default

`switch` değeri durumlarla karşılaştırır ve doğrudan `case 2` 'ye **atlar**.

Adım 2 / 6

DEĞİŞKENLER

gun = 2

ÇIKTI



"break" Unutulursa: Düşme (Fall-through)

```
1 int gun = 2;  
2 switch (gun) {  
3     case 1:  
4         System.out.println("Pazartesi");  
5     case 2:  
6         System.out.println("Salı");  
7     case 3:  
8         System.out.println("Çarşamba");  
9     default:  
10        System.out.println("Diğer gün");  
11 }
```

case 1

case 2

case 3

default

Salı yazılır. Ama **break** yok!

Adım 3 / 6

DEĞİŞKENLER

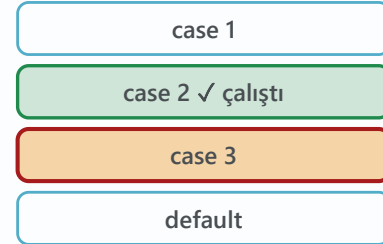
gun = 2

ÇIKTI

Salı

"break" Unutulursa: Düşme (Fall-through)

```
1 int gun = 2;
2 switch (gun) {
3     case 1:
4         System.out.println("Pazartesi");
5     case 2:
6         System.out.println("Salı");
7     case 3:
8         System.out.println("Çarşamba");
9     default:
10        System.out.println("Diğer gün");
11 }
```



break olmadığı için akış bir sonraki duruma **düşer** (fall-through): koşula bakılmadan Çarşamba yazılır.

Adım 4 / 6

DEĞİŞKENLER

gun = 2

ÇIKTI

```
Salı
Çarşamba
```

"break" Unutulursa: Düşme (Fall-through)

```
1 int gun = 2;
2 switch (gun) {
3     case 1:
4         System.out.println("Pazartesi");
5     case 2:
6         System.out.println("Salı");
7     case 3:
8         System.out.println("Çarşamba");
9     default:
10        System.out.println("Diğer gün");
11 }
```

case 1

case 2 ✓ çalıştı

case 3 ✓ çalıştı

default

Yine düşer: default da çalışır.

Adım 5 / 6

DEĞİŞKENLER

gun = 2

ÇIKTI

```
Salı
Çarşamba
Diğer gün
```

"break" Unutulursa: Düşme (Fall-through)

```
1 int gun = 2;
2 switch (gun) {
3     case 1:
4         System.out.println("Pazartesi");
5     case 2:
6         System.out.println("Salı");
7     case 3:
8         System.out.println("Çarşamba");
9     default:
10        System.out.println("Diğer gün");
11 }
```

case 1

case 2 ✓ çalıştı

case 3 ✓ çalıştı

default ✓ çalıştı

switch biter: tek bir gün yerine **üç satır** yazıldı. Her **case** 'in sonuna **break** konmalıdır.

Adım 6 / 6

DEĞİŞKENLER

gun = 2

ÇIKTI

```
Salı
Çarşamba
Diğer gün
```

Ok Söz Dizimli switch (Java 14+)

- Java 14 ile `case ... ->` biçimi geldi: her durum **yalnızca kendi** kodunu çalıştırır, `break` gerekmez, düşme olmaz.
- `switch` bir **değer üreten ifade** olarak kullanılabilir; sonuç doğrudan değişkene atanır.
- Bu derste klasik biçim esas alınır; yeni biçimi gördüğünüzde tanıyın.

```
int gun = 3;
String ad = switch (gun) {
    case 1 -> "Pazartesi";
    case 2 -> "Salı";
    case 3 -> "Çarşamba";
    default -> "Diğer gün";
};
System.out.println(ad);
```

ÇIKTI

Çarşamba

Üçlü Operatör (Ternary Operator)

Koşula göre iki değerden birini seçen kısa bir ifadedir: koşul doğruysa **değer1** , değilse **değer2** .

```
sonuç = (koşul) ? değer1 : değer2;
```

Örnek: bir sayının pozitif ya da negatif olduğunu belirleme.

```
int sayi = -5;  
String sonuc = (sayi > 0) ? "Pozitif" : "Negatif";  
System.out.println("Sayı " + sonuc);
```

ÇIKTI

```
Sayı Negatif
```

Aynı işi yapan **if-else** :

```
String sonuc;  
if (sayi > 0) {  
    sonuc = "Pozitif";  
} else {  
    sonuc = "Negatif";  
}
```

?: bir **değer** üretir, bu yüzden atamanın sağında ya da **println** içinde kullanılır. İki seçeneğin tipi uyumlu olmalıdır.

İç İçe Üçlü Operatörler

Üçlü operatörler iç içe kullanılabilir; daha karmaşık koşullar kısa yazılır. Örnek: sayı sıfır mı, pozitif mi, negatif mi?

```
int sayi = -5;  
String sonuc = (sayi == 0) ? "Sıfır" : (sayi > 0) ? "Pozitif" : "Negatif";  
System.out.println("Sayı " + sonuc);
```

sayi	Sonuç
0	Sıfır
7	Pozitif
-5	Negatif

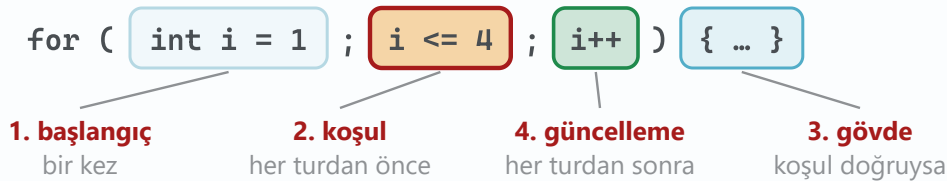
Okunuşu: "sıfır mı? evet → Sıfır; hayır → pozitif mi? evet → Pozitif; hayır → Negatif".
İkinci `?:` birincinin `değer2` 'sidir (sağdan gruplanır).

İkiden fazla iç içe `?:` okunurluğu bozar; o durumda `if-else if` tercih edilir.

"for" Döngüsü

- Bir işlemi belirli bir aralıkta, **kaç kez** yapılacağı bilindiğinde tekrarlamak için kullanılır.
- Başlangıç değeri, koşul ve artırma/azaltma işlemi döngünün çalışmasını kontrol eder.
- Koşul dikkatle yazılmalıdır: hiç `false` olmayan bir koşul **sonsuz döngü** oluşturur (ör. `i--` yazıp `i <= 4` beklemek).

```
for (başlangıç; koşul; artırma/azaltma) {  
    // Döngü içinde yapılacak işlem  
}
```



for Döngüsü: 1'den 4'e Toplam

```
► int toplam = 0;
2 for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```

1 2 3 4

toplam 0 ile başlar.

Adım 1 / 11

DEĞİŞKENLER

toplam = 0

ÇIKTI

for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 ▶ for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```



Başlangıç: `i = 1` . Koşul `1 <= 4` → **true**,
gövdeye girilir.

Adım 2 / 11

DEĞİŞKENLER

toplam = 0

i = 1

ÇIKTI



for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```



`toplam = 0 + 1 = 1` , satır yazdırılır.

Adım 3 / 11

DEĞİŞKENLER

`toplam = 1`

`i = 1`

ÇIKTI

1 → 1

for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 ▶ for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```



`i++` → `i = 2` . Koşul `2 <= 4` → **true**,
gövdeye girilir.

Adım 4 / 11

DEĞİŞKENLER

`toplam = 1`

`i = 2`

ÇIKTI

1 → 1

for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```



toplam = 1 + 2 = 3 , satır yazdırılır.

Adım 5 / 11

DEĞİŞKENLER

toplam = 3

i = 2

ÇIKTI

1 → 1

2 → 3

for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 ▶ for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```



`i++` → `i = 3` . Koşul `3 <= 4` → **true**,
gövdeye girilir.

Adım 6 / 11

DEĞİŞKENLER

`toplam = 3`

`i = 3`

ÇIKTI

```
1 → 1
2 → 3
```

for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```



`toplam = 3 + 3 = 6` , satır yazdırılır.

Adım 7 / 11

DEĞİŞKENLER

`toplam = 6`

`i = 3`

ÇIKTI

```
1 → 1
2 → 3
3 → 6
```

for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 ▶ for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```



`i++` → `i = 4` . Koşul `4 <= 4` → **true**,
gövdeye girilir.

Adım 8 / 11

DEĞİŞKENLER

`toplam = 6`

`i = 4`

ÇIKTI

```
1 → 1
2 → 3
3 → 6
```

for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```



toplam = 6 + 4 = 10 , satır yazdırılır.

Adım 9 / 11

DEĞİŞKENLER

toplam = 10

i = 4

ÇIKTI

```
1 → 1
2 → 3
3 → 6
4 → 10
```

for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 ▶ for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
6 System.out.println("Toplam: " + toplam);
```



`i++` → 5. Koşul `5 <= 4` → **false**: döngü biter. (`i` yalnızca döngü içinde tanımlıdır.)

Adım 10 / 11

DEĞİŞKENLER

`toplam = 10`

`i = 5`

ÇIKTI

```
1 → 1
2 → 3
3 → 6
4 → 10
```

for Döngüsü: 1'den 4'e Toplam

```
1 int toplam = 0;
2 for (int i = 1; i <= 4; i++) {
3     toplam = toplam + i;
4     System.out.println(i + " → " + toplam);
5 }
▶ System.out.println("Toplam: " + toplam);
```



Döngüden sonraki satır çalışır: $1 + 2 + 3 + 4 = 10$.

Adım 11 / 11

DEĞİŞKENLER

toplam = 10

ÇIKTI

```
1 → 1
2 → 3
3 → 6
4 → 10
Toplam: 10
```

"while" Döngüsü

Koşul sağlandığı **sürece** işlemi tekrarlar; koşul her turdan **önce** denetlenir. Kaç kez döneceği önceden bilinmediğinde kullanılır. Söz dizimi: `while (koşul) { ... }`

```
► int sayi = 4071;
► int basamak = 0;
3 while (sayi > 0) {
4     sayi = sayi / 10;
5     basamak++;
6 }
7 System.out.println("Basamak sayısı: " + basamak);
```

4 0 7 1 → sayi = 4071

Sayının basamaklarını `/ 10` ile tek tek atarak sayacağız.

Adım 1 / 7

DEĞİŞKENLER

sayi = 4071

basamak = 0

ÇIKTI

“while” Döngüsü

Koşul sağlandığı **sürece** işlemi tekrarlar; koşul her turdan **önce** denetlenir. Kaç kez döneceği önceden bilinmediğinde kullanılır. Söz dizimi: `while (koşul) { ... }`

```
1 int sayi = 4071;
2 int basamak = 0;
▶ while (sayi > 0) {
▶     sayi = sayi / 10;
▶     basamak++;
6 }
7 System.out.println("Basamak sayısı: " + basamak);
```

4 0 7 1 → sayi = 407

Koşul `sayi > 0` doğru → son basamak atılır: `sayi = 407`, `basamak = 1`.

Adım 2 / 7

DEĞİŞKENLER

sayi = 407 basamak = 1

ÇIKTI

“while” Döngüsü

Koşul sağlandığı **sürece** işlemi tekrarlar; koşul her turdan **önce** denetlenir. Kaç kez döneceği önceden bilinmediğinde kullanılır. Söz dizimi: `while (koşul) { ... }`

```
1 int sayi = 4071;
2 int basamak = 0;
▶ while (sayi > 0) {
▶     sayi = sayi / 10;
▶     basamak++;
6 }
7 System.out.println("Basamak sayısı: " + basamak);
```

4 0 7 1 → sayi = 40

Koşul `sayi > 0` doğru → son basamak atılır: `sayi = 40`, `basamak = 2`.

Adım 3 / 7

DEĞİŞKENLER

sayi = 40 basamak = 2

ÇIKTI

“while” Döngüsü

Koşul sağlandığı **sürece** işlemi tekrarlar; koşul her turdan **önce** denetlenir. Kaç kez döneceği önceden bilinmediğinde kullanılır. Söz dizimi: `while (koşul) { ... }`

```
1 int sayi = 4071;
2 int basamak = 0;
▶ while (sayi > 0) {
▶     sayi = sayi / 10;
▶     basamak++;
6 }
7 System.out.println("Basamak sayısı: " + basamak);
```

4 0 7 1 → sayi = 4

Koşul `sayi > 0` doğru → son basamak atılır: `sayi = 4`, `basamak = 3`.

Adım 4 / 7

DEĞİŞKENLER

sayi = 4 basamak = 3

ÇIKTI

“while” Döngüsü

Koşul sağlandığı **sürece** işlemi tekrarlar; koşul her turdan **önce** denetlenir. Kaç kez döneceği önceden bilinmediğinde kullanılır. Söz dizimi: `while (koşul) { ... }`

```
1 int sayi = 4071;
2 int basamak = 0;
▶ while (sayi > 0) {
▶     sayi = sayi / 10;
▶     basamak++;
6 }
7 System.out.println("Basamak sayısı: " + basamak);
```

4 0 7 1 → sayi = 0

Koşul `sayi > 0` doğru → son basamak atılır: `sayi = 0`, `basamak = 4`.

Adım 5 / 7

DEĞİŞKENLER

sayi = 0 basamak = 4

ÇIKTI

“while” Döngüsü

Koşul sağlandığı **sürece** işlemi tekrarlar; koşul her turdan **önce** denetlenir. Kaç kez döneceği önceden bilinmediğinde kullanılır. Söz dizimi: `while (koşul) { ... }`

```
1 int sayi = 4071;
2 int basamak = 0;
▶ while (sayi > 0) {
4     sayi = sayi / 10;
5     basamak++;
6 }
7 System.out.println("Basamak sayısı: " + basamak);
```

4 0 7 1 → sayi = 0

Koşul `0 > 0` → **false**: döngü biter. Kaç tur döneceği önceden bilinmiyordu; koşul belirledi.

Adım 6 / 7

DEĞİŞKENLER

sayi = 0

basamak = 4

ÇIKTI

“while” Döngüsü

Koşul sağlandığı **sürece** işlemi tekrarlar; koşul her turdan **önce** denetlenir. Kaç kez döneceği önceden bilinmediğinde kullanılır. Söz dizimi: `while (koşul) { ... }`

```
1 int sayi = 4071;
2 int basamak = 0;
3 while (sayi > 0) {
4     sayi = sayi / 10;
5     basamak++;
6 }
▶ System.out.println("Basamak sayısı: " + basamak);
```

4 0 7 1 → sayi = 0

Sonuç yazdırılır.

Adım 7 / 7

DEĞİŞKENLER

sayi = 0

basamak = 4

ÇIKTI

Basamak sayısı: 4

"do-while" Döngüsü

- Gövde **en az bir kez** çalışır; koşul turun **sonunda** denetlenir.
- Kullanıcıdan geçerli bir giriş isteme gibi senaryolar için uygundur.
- `while (koşul);` sonundaki `;` unutulmamalıdır.

```
do {  
    // Döngü içinde yapılacak işlem  
} while (koşul);
```

```
// dosyanın başında: import java.util.Scanner;  
Scanner giris = new Scanner(System.in);  
int sayi;  
do {  
    System.out.print("1-10 arası bir sayı girin: ");  
    sayi = giris.nextInt();  
} while (sayi < 1 || sayi > 10);  
System.out.println("Girilen: " + sayi);
```

ÖRNEK ÇALIŞMA (GİRİŞLER: 0, 15, 7)

```
1-10 arası bir sayı girin: 0  
1-10 arası bir sayı girin: 15  
1-10 arası bir sayı girin: 7  
Girilen: 7
```

do-while ile Basamak Sayma

```
▶ int n = 507;
▶ int basamak = 0;

3 do {
4     basamak++;
5     n = n / 10;
6 } while (n > 0);
7 System.out.println("Basamak: " +
    basamak);
```

507'nin basamakları

5

0

7

n = 507

basamak = 0

silik basamaklar n / 10 ile atıldı

n = 507, basamak = 0. Döngü koşulu en sonda denetlenecek.

Adım 1 / 8

DEĞİŞKENLER

n = 507

basamak = 0

ÇIKTI

do-while ile Basamak Sayma

```
1 int n = 507;
2 int basamak = 0;
3 do {
4     basamak++;
5     n = n / 10;
6 } while (n > 0);
7 System.out.println("Basamak: " + basamak);
```

507'nin basamakları



$n = 50$

$\text{basamak} = 1$

silik basamaklar $n / 10$ ile atıldı

Gövde koşula bakılmadan çalışır: basamak 1 olur, $n / 10$ son basamağı (7) atar $\rightarrow 50$.

Adım 2 / 8

DEĞİŞKENLER

$n = 50$

$\text{basamak} = 1$

ÇIKTI



do-while ile Basamak Sayma

```
1 int n = 507;
2 int basamak = 0;
3 do {
4     basamak++;
5     n = n / 10;
6 } while (n > 0);
7 System.out.println("Basamak: " + basamak);
```

507'nin basamakları



n = 50

basamak = 1

50 > 0: true

silik basamaklar n / 10 ile atıldı

Tur sonunda koşul: 50 > 0 doğru → başa dönülür.

Adım 3 / 8

DEĞİŞKENLER

n = 50

basamak = 1

ÇIKTI



do-while ile Basamak Sayma

```
1 int n = 507;
2 int basamak = 0;
3 do {
4     basamak++;
5     n = n / 10;
6 } while (n > 0);
7 System.out.println("Basamak: " + basamak);
```

507'nin basamakları

5

~~0~~

~~7~~

n = 5

basamak = 2

silik basamaklar n / 10 ile atıldı

basamak 2; $50 / 10 = 5$.

Adım 4 / 8

DEĞİŞKENLER

n = 5

basamak = 2

ÇIKTI

do-while ile Basamak Sayma

```
1 int n = 507;
2 int basamak = 0;
3 do {
4     basamak++;
5     n = n / 10;
6 } while (n > 0);
7 System.out.println("Basamak: " + basamak);
```

507'nin basamakları



n = 5

basamak = 2

5 > 0: true

silik basamaklar n / 10 ile atıldı

5 > 0 doğru → bir tur daha.

Adım 5 / 8

DEĞİŞKENLER

n = 5

basamak = 2

ÇIKTI

do-while ile Basamak Sayma

```
1 int n = 507;  
2 int basamak = 0;  
3 do {  
4     basamak++;  
5     n = n / 10;  
6 } while (n > 0);  
7 System.out.println("Basamak: " +  
    basamak);
```

507'nin basamakları

~~5~~

~~0~~

~~7~~

n = 0

basamak = 3

silik basamaklar n / 10 ile atıldı

basamak 3; $5 / 10 = 0$ (tam sayı bölmesi).

Adım 6 / 8

DEĞİŞKENLER

n = 0

basamak = 3

ÇIKTI

do-while ile Basamak Sayma

```
1 int n = 507;
2 int basamak = 0;
3 do {
4     basamak++;
5     n = n / 10;
6 } while (n > 0);
7 System.out.println("Basamak: " + basamak);
```

507'nin basamakları



n = 0

basamak = 3

0 > 0: false

silik basamaklar n / 10 ile atıldı

0 > 0 yanlış → döngüden çıkılır.

Adım 7 / 8

DEĞİŞKENLER

n = 0

basamak = 3

ÇIKTI



do-while ile Basamak Sayma

```
1 int n = 507;
2 int basamak = 0;
3 do {
4     basamak++;
5     n = n / 10;
6 } while (n > 0);
▶ System.out.println("Basamak: " + basamak);
```

507'nin basamakları



n = 0

basamak = 3

silik basamaklar n / 10 ile atıldı

Sonuç yazılır: 3 basamak. **n = 0** ile başlansaydı gövde yine bir kez çalışır ve doğru cevap **1** çıkardı; **while** ile 0 çıkardı.

Adım 8 / 8

DEĞİŞKENLER

n = 0

basamak = 3

ÇIKTI

Basamak: 3

for, while, do-while: Hangisi Ne Zaman?

	for	while	do-while
Koşul ne zaman denetlenir?	her turdan önce	her turdan önce	her turdan sonra
En az kaç kez çalışır?	0	0	1
Tipik kullanım	tekrar sayısı belli: 1'den n'e, dizinin her ögesi	tekrar sayısı belli değil: basamak sayma, dosya sonuna kadar okuma	önce yap, sonra kontrol et: menü, giriş doğrulama
Örnek	<pre>for (int i = 0; i < n; i++)</pre>	<pre>while (sayi > 0)</pre>	<pre>do { ... } while (secim != 0);</pre>

Her `for` döngüsü bir `while` ile yazılabilir: `başlangıç; while (koşul) {
gövde; güncelleme; }`. Döngüler bir sonraki bölümde ayrıntılı işlenecektir.

“break” ile Döngüden Çıkmak

`break` içinde bulunduğu döngüden ya da `switch` bloğundan hemen çıkar. Aranılan değer bulunduğunda aramayı bitirmek gibi durumlarda kullanılır.

```
1 for (int i = 1; i <= 10; i++) {  
2     if (i == 5) {  
3         break; // Döngüyü sonlandır  
4     }  
5     System.out.println(i);  
6 }  
7 System.out.println("Döngü bitti");
```

1 2 3 4 5 6 7 8 9 10

1 == 5 yanlış → 1 yazdırılır.

Adım 1 / 6

DEĞİŞKENLER

i = 1

ÇIKTI

1

“break” ile Döngüden Çıkmak

`break` içinde bulunduğu döngüden ya da `switch` bloğundan hemen çıkar. Aranılan değer bulunduğunda aramayı bitirmek gibi durumlarda kullanılır.

```
1 for (int i = 1; i <= 10; i++) {  
2     if (i == 5) {  
3         break; // Döngüyü sonlandır  
4     }  
5     System.out.println(i);  
6 }  
7 System.out.println("Döngü bitti");
```

1 2 3 4 5 6 7 8 9 10

2 == 5 yanlış → 2 yazdırılır.

Adım 2 / 6

DEĞİŞKENLER

i = 2

ÇIKTI

1
2

“break” ile Döngüden Çıkmak

`break` içinde bulunduğu döngüden ya da `switch` bloğundan hemen çıkar. Aranılan değer bulunduğunda aramayı bitirmek gibi durumlarda kullanılır.

```
1 for (int i = 1; i <= 10; i++) {  
2     if (i == 5) {  
3         break; // Döngüyü sonlandır  
4     }  
5     System.out.println(i);  
6 }  
7 System.out.println("Döngü bitti");
```



3 == 5 yanlış → 3 yazdırılır.

Adım 3 / 6

DEĞİŞKENLER

i = 3

ÇIKTI

```
1  
2  
3
```

“break” ile Döngüden Çıkmak

`break` içinde bulunduğu döngüden ya da `switch` bloğundan hemen çıkar. Aranılan değer bulunduğunda aramayı bitirmek gibi durumlarda kullanılır.

```
1 for (int i = 1; i <= 10; i++) {  
2     if (i == 5) {  
3         break; // Döngüyü sonlandır  
4     }  
5     System.out.println(i);  
6 }  
7 System.out.println("Döngü bitti");
```



4 == 5 yanlış → 4 yazdırılır.

Adım 4 / 6

DEĞİŞKENLER

i = 4

ÇIKTI

```
1  
2  
3  
4
```

“break” ile Döngüden Çıkmak

`break` içinde bulunduğu döngüden ya da `switch` bloğundan hemen çıkar. Aranılan değer bulunduğunda aramayı bitirmek gibi durumlarda kullanılır.

```
1 for (int i = 1; i <= 10; i++) {  
    ▶ if (i == 5) {  
        ▶ break; // Döngüyü sonlandır  
4    }  
5    System.out.println(i);  
6 }  
7 System.out.println("Döngü bitti");
```



`i == 5` doğru → `break` : döngü **hemen** sonlanır; 5 yazılmaz, 6–10 hiç denenmez.

Adım 5 / 6

DEĞİŞKENLER

`i = 5`

ÇIKTI

```
1  
2  
3  
4
```

“break” ile Döngüden Çıkmak

break içinde bulunduğu döngüden ya da **switch** bloğundan hemen çıkar. Aranılan değer bulunduğunda aramayı bitirmek gibi durumlarda kullanılır.

```
1 for (int i = 1; i <= 10; i++) {  
2     if (i == 5) {  
3         break; // Döngüyü sonlandır  
4     }  
5     System.out.println(i);  
6 }  
▶ System.out.println("Döngü bitti");
```



Akış döngüden sonraki ilk satırdan devam eder.

Adım 6 / 6

DEĞİŞKENLER

—

ÇIKTI

```
1  
2  
3  
4  
Döngü bitti
```

"continue" ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.
Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
2     if (i % 2 == 0) {  
3         continue; // Çift sayıları atla  
4     }  
5     System.out.println(i);  
6 }
```



$1 \% 2 = 1 \rightarrow$ koşul yanlış, **1** yazdırılır.

Adım 1 / 11

DEĞİŞKENLER

i = 1

ÇIKTI

1

“continue” ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
    ▶ if (i % 2 == 0) {  
        ▶ continue; // Çift sayıları atla  
4    }  
5    System.out.println(i);  
6 }
```



`2 % 2 == 0` doğru → `continue` :

`println` atlanır, `i++` ile sonraki tura geçilir.

Adım 2 / 11

DEĞİŞKENLER

`i = 2`

ÇIKTI

1

"continue" ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
    ▶ if (i % 2 == 0) {  
        3 continue; // Çift sayıları atla  
        4 }  
    ▶ System.out.println(i);  
    6 }
```



$3 \% 2 = 1 \rightarrow$ koşul yanlış, 3 yazdırılır.

Adım 3 / 11

DEĞİŞKENLER

i = 3

ÇIKTI

1
3

“continue” ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
    ▶ if (i % 2 == 0) {  
        ▶ continue; // Çift sayıları atla  
4    }  
5    System.out.println(i);  
6 }
```



4 % 2 == 0 doğru → continue :

println atlanır, i++ ile sonraki tura geçilir.

Adım 4 / 11

DEĞİŞKENLER

i = 4

ÇIKTI

1
3

“continue” ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
  ▶ if (i % 2 == 0) {  
    3     continue; // Çift sayıları atla  
    4   }  
  ▶ System.out.println(i);  
  6 }
```



$5 \% 2 = 1 \rightarrow$ koşul yanlış, 5 yazdırılır.

Adım 5 / 11

DEĞİŞKENLER

i = 5

ÇIKTI

```
1  
3  
5
```

“continue” ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
    ▶ if (i % 2 == 0) {  
        ▶ continue; // Çift sayıları atla  
4    }  
5    System.out.println(i);  
6 }
```



6 % 2 == 0 doğru → continue :

println atlanır, i++ ile sonraki tura geçilir.

Adım 6 / 11

DEĞİŞKENLER

i = 6

ÇIKTI

1
3
5

"continue" ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
▶   2 if (i % 2 == 0) {  
3       3 continue; // Çift sayıları atla  
4   }  
▶   5 System.out.println(i);  
6 }
```



$7 \% 2 = 1 \rightarrow$ koşul yanlış, 7 yazdırılır.

Adım 7 / 11

DEĞİŞKENLER

i = 7

ÇIKTI

```
1  
3  
5  
7
```

“continue” ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
    ▶ if (i % 2 == 0) {  
        ▶ continue; // Çift sayıları atla  
4    }  
5    System.out.println(i);  
6 }
```



8 % 2 == 0 doğru → continue :

println atlanır, i++ ile sonraki tura geçilir.

Adım 8 / 11

DEĞİŞKENLER

i = 8

ÇIKTI

1
3
5
7

“continue” ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
  ▶ if (i % 2 == 0) {  
    3     continue; // Çift sayıları atla  
    4   }  
  ▶ System.out.println(i);  
  6 }
```



$9 \% 2 = 1 \rightarrow$ koşul yanlış, 9 yazdırılır.

Adım 9 / 11

DEĞİŞKENLER

i = 9

ÇIKTI

```
1  
3  
5  
7  
9
```

“continue” ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
1 for (int i = 1; i <= 10; i++) {  
    ▶ if (i % 2 == 0) {  
        ▶ continue; // Çift sayıları atla  
4    }  
5    System.out.println(i);  
6 }
```



10 % 2 == 0 doğru → continue :

println atlanır, i++ ile sonraki tura geçilir.

Adım 10 / 11

DEĞİŞKENLER

i = 10

ÇIKTI

```
1  
3  
5  
7  
9
```

“continue” ile Tur Atlamak

Koşul sağlandığında turun geri kalanını atlar ve döngüyü bir sonraki değerden sürdürür.

Örnek: çift sayıları atlayıp yalnızca tek sayıları yazdırma.

```
▶ for (int i = 1; i <= 10; i++) {  
2     if (i % 2 == 0) {  
3         continue; // Çift sayıları atla  
4     }  
5     System.out.println(i);  
6 }
```



11 <= 10 yanlış → döngü biter. **continue** döngüyü bitirmez, yalnızca o turu atlar.

Adım 11 / 11

DEĞİŞKENLER

i = 11

ÇIKTI

```
1  
3  
5  
7  
9
```

ÖZET

Özet

Konu	Hatırlanacaklar
Derleme	<code>javac Ad.java</code> → <code>Ad.class</code> (bytecode) · <code>java Ad</code> → JVM çalıştırır; aynı bytecode her platformda
Söz dizimi	deyim <code>;</code> ile biter · bloklar <code>{ }</code> · büyük-küçük harf duyarlı · <code>//</code> ve <code>/* */</code> açıklama
İlkel tipler	<code>byte</code> 1, <code>short</code> 2, <code>int</code> 4, <code>long</code> 8 · <code>float</code> 4, <code>double</code> 8 · <code>char</code> 2 bayt · <code>boolean</code> ; sınırı aşan değer taşar
Adlar	harf, rakam, <code>_</code> , <code>\$</code> ; rakamla başlamaz; anahtar kelime olmaz · Sınıf PascalCase, değişken camelCase, SABIT_AD
Atama	sağ taraf önce hesaplanır · <code>+=</code> ... kısaltmaları · yerel değişkene okumadan önce değer verilmeli · <code>final</code> sabit
Dönüşüm	<code>byte</code> → <code>short</code> → <code>int</code> → <code>long</code> → <code>float</code> → <code>double</code> otomatik; ters yön (tür) ile, kesir atılır · <code>int / int</code> tam sayı bölmesi
Operatörler	tekli > <code>*</code> / <code>%</code> > <code>+</code> - > karşılaştırma > <code>&&</code> > <code> </code> > <code>?:</code> > atama · <code>&&</code> <code> </code> kısa devre
Akış	<code>if/else</code> , <code>switch</code> (+ <code>break</code>), <code>?:</code> · <code>for</code> , <code>while</code> (0+ tur), <code>do-while</code> (1+ tur) · <code>break</code> çıkar, <code>continue</code> atlar

İlgili Sayfalar

ASCII Tablosu

`char` ↔ kod dönüşümü için
karakterlerin sayısal kodları: `'A'` = 65,
`'a'` = 97, `'0'` = 48 ...

ÖZET

Sık Yapılan Hatalar (1)

Eksik noktalı virgül

`error: ';' expected` . Hata çoğu zaman bir **önceki** satırın sonundadır.

Dosya adı ≠ sınıf adı

`class MerhabaDunya is public, should be declared in a file named MerhabaDunya.java`

Büyük-küçük harf

`system.out.println` , `string` , `Int` derlenmez: `error: cannot find symbol` . Doğrusu `System` , `String` , `int` .

Değersiz yerel değişken

`variable sayac might not have been initialized` : yerel değişkenler varsayılan değer almaz.

Sonek olmadan sabit

`float f = 3.14;` → `possible lossy conversion from double to float` (`3.14f` yazın) . `long l = 30000000000;` → `integer number too large` (`L` ekleyin).

Daraltmada cast unutmak

`int` → `char` , `double` → `int` atamaları `incompatible types: possible lossy conversion` verir; `(char)` , `(int)` gerekir.

ÖZET

Sık Yapılan Hatalar (2): Derlenir ama Yanlış

Tam sayı bölmesi

`double ort = (3 + 4) / 2;` → 3.0.
Doğrusu `(3 + 4) / 2.0` → 3.5.

= ile ==

`if (x = 5)` → `int cannot be converted to boolean`. Karşılaştırma `==` ile yapılır.

switch 'te break yok

Eşleşen durumdan sonraki tüm durumlar da çalışır (fall-through).

if 'ten sonra ;

`if (x > 0); { ... }` : boş deyim koşula bağlanır, blok **her zaman** çalışır.

Sonsuz döngü

`for (int i = 1; i <= 10; i--)` ya da gövdede koşul değişkenini güncellemeyi unutan `while`.

Ondalıkta ==

`0.1 + 0.2 == 0.3` → `false`. Farkın küçük olup olmadığına bakın: `Math.abs(a - b) < 1e-9`.

ÖZET

Örnek Problemler

Her satırın çıktısı nedir? (`int a = 7, b = 2;`)

1. `System.out.println(a / b + " " + a % b + " " + (double) a / b);`
2. `System.out.println(1 + 2 + "3" + 4 + 5);`
3. `int i = 3; int j = i++ * 2;` → `i` ve `j` kaçtır?
4. `byte k = (byte) 200;` → `k` kaçtır?
5. `for (int m = 10; m > 0; m -= 3) System.out.print(m + " ");`
6. `System.out.println('a' + 1);` ve `System.out.println((char) ('a' + 1));`

► Cevapları göster

ÖZET

Alıştırmalar

1. Santigradı Fahrenheitya çevirin: $F = C \times 9 / 5 + 32$. $C = 37$ için sonuç nedir? `int` ile `9 / 5 * c + 32` yazılırsa ne olur?
2. 3725 saniyeyi `/` ve `%` ile saat, dakika, saniyeye ayırın.
3. Bir sayının tek mi çift mi olduğunu **üçlü operatörle** yazdırın.
4. 1'den 100'e kadar olan sayılardan 3'ün katı **olmayanların** toplamını `continue` ile bulun.
5. `char harf = 'g';` karakterini, kod farkını kullanarak büyük harfe çevirin.
6. Şu adlardan hangileri geçerli bir tanımlayıcıdır: `_toplam` , `3boyut` , `ogrenciNo` , `final` , `ad` `soyad` , `Final` ?

► Cevaplar / ipuçları

Kaynaklar

- Oracle, *The Java Tutorials: "Language Basics"* (değişkenler, operatörler, ifadeler, kontrol akışı), docs.oracle.com/javase/tutorial.
- J. Gosling vd., *The Java Language Specification*, Java SE 21 Edition (Bölüm 3: söz dizimi, anahtar kelimeler, sabit değerler; Bölüm 4: tipler; Bölüm 5: dönüşümler).
- T. Lindholm vd., *The Java Virtual Machine Specification*, Java SE 21 Edition (bytecode ve JVM).
- P. Deitel ve H. Deitel, *Java: How to Program*, 11. baskı, Pearson, 2017 (Bölüm 2–5).
- K. Sierra ve B. Bates, *Head First Java*, 3. baskı, O'Reilly, 2022.
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 13. baskı, 2024 (Bölüm 3).
- A. B. Downey ve C. Mayfield, *Think Java*, 2. baskı, O'Reilly, 2019.

Son

Bölüm 2: Söz Dizimi Kuralları

Sonraki bölüm: Bölüm 3: Kontrol Yapıları · [Terim sözlüğü](#)