

Bölüm 2: Programlama

Java ile Nesne Yönelimli Programlama

Programlama dilleri

Derleme ve yorumlama

Paradigmalar

Söz dizimi ve anlam

Hata türleri

İfadeler ve öncelik

Değişkenler ve atama

Koşullu ifadeler

Dizgiler ve tipler

Algoritmadan programa

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Programlama dilleri

Programlama dilini tanımlamak; dilleri soyutlama düzeyi, uygulama alanı, yürütme biçimi ve paradigmaya göre sınıflandırmak.

Java'nın çalışması

Kaynak kodun `javac` ile bytecode'a derlenip JVM'de çalıştırılmasını açıklamak.

Söz dizimi ve anlam

Söz dizimi, statik anlam ve anlam kavramlarını ayırmak; derleme, çalışma zamanı ve mantık hatalarını tanımak.

İfadeler

İşlem önceliği, soldan sağa değerlendirme ve tam sayı bölmesiyle bir ifadenin değerini bulmak.

Değişkenler ve tipler

Değişken tanımlamak, atamayı iki adımda izlemek; statik ve dinamik tiplmeyi, tipler arası işlemleri açıklamak.

Algoritmadan programa

İfade ile deyim ayırt etmek; bir algoritmayı Java programına çevirmek ve deyim sırasının önemini göstermek.

Bu Bölümün Amacı

1

Dilin yapı taşları

Bir programlama dilinin temel yapı taşlarını, **söz dizimini** (syntax) ve **anlamını** (semantics) öğrenmek.

2

Tarifi çevirmek

Bir problemin çözüm tarifini, bu yapı taşlarıyla **bilgisayarın anlayacağı** bir biçime, yani programa dönüştürmek.

3

Hesaplamalı düşünme

Bir problemi çözmek için ayrıştırma, örüntü bulma, soyutlama ve algoritma tasarımı gibi **yöntemleri** bir arada kullanabilmek.

Bu bölüm, her programlama dilinde ortak olan kavramları **Java örnekleriyle** tanıtır. Java'nın ayrıntılı söz dizimi kuralları (tanımlayıcılar, veri tipleri, operatörler) bir sonraki bölümün konusudur.

Programlama Dili Nedir?

- **Programlama dili** (programming language): bir makinenin gerçekleştirebileceği hesaplamaları ifade etmek için tasarlanmış **yapay bir dildir**.
- Makinenin davranışını kontrol eden **programlar** yazmak için kullanılır:
 - **algoritmaları** açık ve kesin biçimde ifade etmek için,
 - programcılar arasında bir **iletişim aracı** olarak.
- Bilgisayarlara, telefonlara ve gömülü cihazlara **işlevsellik** kazandırır.
- Algoritmaların ve mantıksal süreçlerin ifadesidir.
- Dil seçimi **projenin ihtiyaçlarına** göre yapılır.
- Örnekler: C, C++, Java, Python, Prolog, Haskell, Scala...



Soyutlama Seviyeleri

- Diller, **düşük seviyeden yüksek seviyeye** farklı **soyutlama** (abstraction) düzeylerindedir.
- **Düşük seviye** diller (ör. Assembly) neredeyse makine düzeyinde çalışır; komutları çok basittir: "belleği oku", "topla", "yaz".
- **Yüksek seviye** diller (ör. C, Java, Python) zengin bir komut kümesi sunar; programcı yazmaçları (register) ve bellek adreslerini düşünmez.
- Soyutlama arttıkça kod **kısılır ve okunur**; dönüştürme işini derleyici yapar.

YÜKSEK SEVİYE · JAVA

```
int toplam = a + b;
```

DÜŞÜK SEVİYE · X86 ASSEMBLY

```
mov eax, [ebp-8]    ; a'yı yazmaca yükle  
add eax, [ebp-4]    ; b'yi ekle  
mov [ebp-12], eax   ; sonucu toplam'a yaz
```

MAKİNE KODU · AYNI ÜÇ KOMUTUN BAYTLARI (ONALTILIK)

```
8B 45 F8  
03 45 FC  
89 45 F4
```

Genel Amaçlı ve Hedefe Odaklı

- Diller, uygulama alanına göre **genel amaçlı** (general-purpose) ya da **hedefe odaklı** (targeted, domain-specific) olabilir.

Genel amaçlı

- + Çok çeşitli uygulamaları destekleyen geniş bir temel komut kümesi.
- + Java, C, C++, Python: masaüstü, web, mobil, gömülü sistem...

Hedefe odaklı

- Belirli bir uygulama türüne odaklanır, o işlemleri kolaylaştırır.
- **MATLAB**: sayısal hesaplama; matris ve vektör işlemleri optimize. **SQL**: veri tabanı sorguları.

MATLAB · MATRİS ÇARPIMI

```
C = A * B;    % tek satır
```

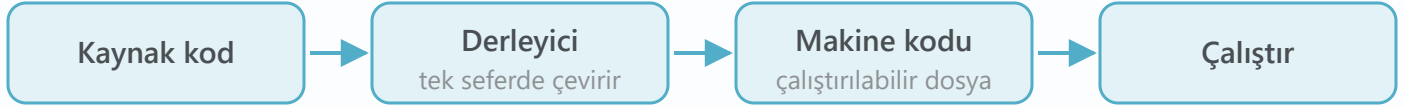
JAVA · AYNI İŞ, ELLE YAZILAN DÖNGÜLER

```
for (int i = 0; i < n; i++)  
    for (int j = 0; j < n; j++)  
        for (int k = 0; k < n; k++)  
            C[i][j] += A[i][k] * B[k][j];
```

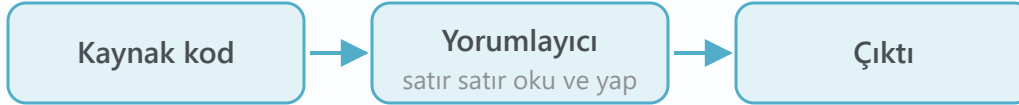
Derlenen ve Yorumlanan Diller

Diller, kodun **yürütölme** biçimine göre **derlenen** (compiled) ve **yorumlanan** (interpreted) olarak sınıflandırılır.

DERLENEN DİL (C, C++, GO, RUST)

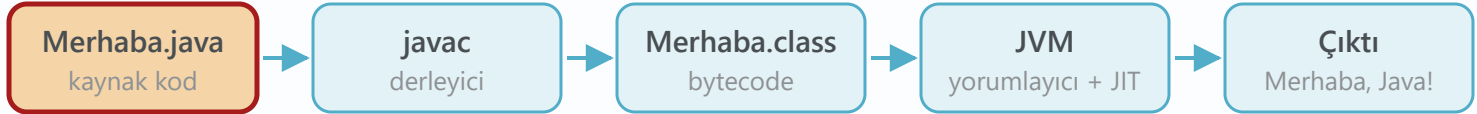


YORUMLANAN DİL (PYTHON, JAVASCRIPT)



	Derlenen	Yorumlanan
Nasıl?	Tüm kaynak kod, çalıştırılmadan önce derleyiciyle nesne (object) / makine koduna çevrilir.	Yorumlayıcı kaynak kodu okur ve komutları sırayla o anda çalıştırır.
Hız	Hızlı ve optimize yürütme	Genellikle daha yavaş
Hatalar	Birçoğu derleme sırasında bulunur	Çoğu ancak o satır çalışınca görülür

Java: Önce Derlenir, Sonra JVM'de Çalışır



Programcı kaynak kodu (source code) yazar. Dosyanın adı, içindeki **public** sınıfın adıyla aynı olmalıdır.

Adım 1 / 5

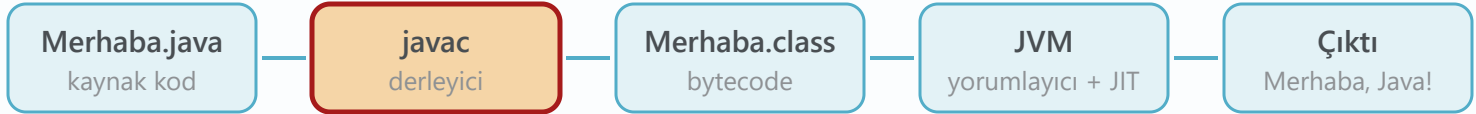
MERHABA.JAVA

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

KOMUT SATIRI

```
> javac Merhaba.java  
> java Merhaba  
Merhaba, Java!
```

Java: Önce Derlenir, Sonra JVM'de Çalışır



`javac` derleyicisi kodu denetler: söz dizimi ve tip hataları **bu aşamada** bulunur. Hata varsa çıktı dosyası üretilmez.

Adım 2 / 5

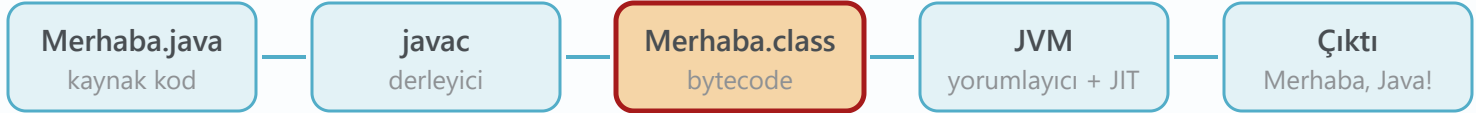
MERHABA.JAVA

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

KOMUT SATIRI

```
> javac Merhaba.java  
> java Merhaba  
Merhaba, Java!
```

Java: Önce Derlenir, Sonra JVM'de Çalışır



Sonuç makine kodu değil, **bytecode**'dur: hiçbir işlemciye özgü olmayan, Java Sanal Makinesi (JVM) için ara kod.

Adım 3 / 5

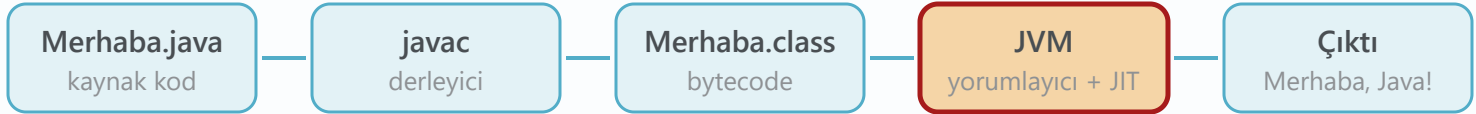
MERHABA.JAVA

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

KOMUT SATIRI

```
> javac Merhaba.java  
> java Merhaba  
Merhaba, Java!
```

Java: Önce Derlenir, Sonra JVM'de Çalışır



`java Merhaba` komutu JVM'i başlatır. JVM bytecode'u yorumlar; sık çalışan kısımları JIT (Just-In-Time) derleyicisiyle makine koduna çevirir.

Adım 4 / 5

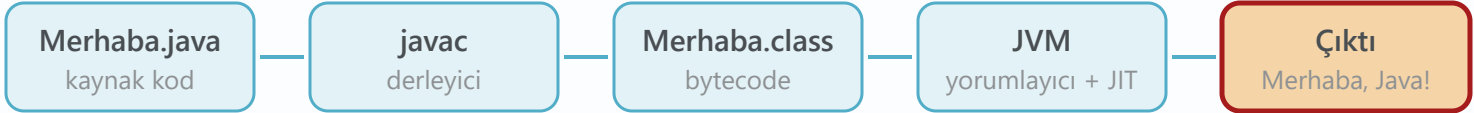
MERHABA.JAVA

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

KOMUT SATIRI

```
> javac Merhaba.java  
> java Merhaba  
Merhaba, Java!
```

Java: Önce Derlenir, Sonra JVM'de Çalışır



Aynı `.class` dosyası Windows, Linux ya da macOS'taki her JVM'de çalışır: **“Bir kez yaz, her yerde çalıştır”** (WORA).

Adım 5 / 5

MERHABA.JAVA

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

KOMUT SATIRI

```
> javac Merhaba.java  
> java Merhaba  
Merhaba, Java!
```

Programlama Paradigmaları

Paradigma (paradigm): bir problemi çözerken izlenen **temel yaklaşım**. Diller, problemi farklı yaklaşımlarla çözebilir.

Buyruksal (imperative)

Programcı makineye, durumunu **nasıl** değiştireceğini adım adım söyler. FORTRAN, BASIC, Pascal, C...



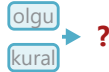
Fonksiyonel (functional)

Hesaplamayı matematiksel **fonksiyonların değerlendirilmesi** olarak ele alır; durum değiştirmekten kaçınır. Lisp, Scheme, Haskell...



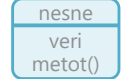
Mantıksal (logic)

Hesaplamanın **mantığını** (olgular ve kurallar) açıklar; kontrol akışını ayrıntılandırmaz. Prolog



Nesne yönelimli (object-oriented)

İlişkili **veri ve metotları** bir **nesne** altında toplar; komutlar, üzerinde çalıştıkları durumla birlikte gruplanır. C++, Java, C#, Python...



Java temelde **nesne yönelimli** ve buyruksaldır; Java 8'den beri fonksiyonel öğeler (lambda, akışlar) de içerir. Birden çok paradigmayı destekleyen dillere **çok paradigmalı** denir.

Aynı Problem: Buyruksal ve Fonksiyonel

Problem: 1'den 5'e kadar sayıların toplamını bul.

```
import java.util.stream.IntStream;

public class Paradigma {
    public static void main(String[] args) {
        // Buyruksal: adım adım durum değiştir
        int toplam = 0;
        for (int i = 1; i <= 5; i++) {
            toplam = toplam + i;
        }
        System.out.println("Buyruksal: " + toplam);

        // Fonksiyonel (Java 8+): ne istediğini söyle
        int sonuc = IntStream.rangeClosed(1, 5).sum();
        System.out.println("Fonksiyonel: " + sonuc);
    }
}
```

ÇIKTI

Buyruksal: 15
Fonksiyonel: 15

- **Buyruksal:** `toplam` değişkeni her adımda **değiştirilir**; "nasıl" yapılacağı yazılır.
- **Fonksiyonel:** "1..5 aralığının toplamı" **istenir**; değişken güncellenmez.

Buyruksal ve Fonksiyonel: Kod İzleme

```
► int toplam = 0;
2 for (int i = 1; i <= 5; i++) {
3     toplam = toplam + i;
4 }
5 System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



toplam



Buyruksal yaklaşımda **toplam** adlı bir **durum** değişkeni 0 ile başlar.

Adım 1 / 10

DEĞİŞKENLER

toplam = 0

ÇIKTI



Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
2 for (int i = 1; i <= 5; i++) {
3     toplam = toplam + i;
4 }
5 System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



toplam



i = 1 : 1 <= 5 doğru → toplam = 0 + 1 = 1. Durum her turda değişir.

Adım 2 / 10

DEĞİŞKENLER

toplam = 1

i = 1

ÇIKTI



Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
2 for (int i = 1; i <= 5; i++) {
3     toplam = toplam + i;
4 }
5 System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



toplam



i = 2 : 2 <= 5 doğru → toplam = 1 + 2 = 3.

Adım 3 / 10

DEĞİŞKENLER

toplam = 3

i = 2

ÇIKTI



Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
2 for (int i = 1; i <= 5; i++) {
3     toplam = toplam + i;
4 }
5 System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



toplam



i = 3 : 3 <= 5 doğru → toplam = 3 + 3 = 6.

Adım 4 / 10

DEĞİŞKENLER

toplam = 6

i = 3

ÇIKTI



Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
2 ▶ for (int i = 1; i <= 5; i++) {
3   ▶   toplam = toplam + i;
4 }
5 System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



toplam

10

6 + 4 = 10

i = 4 : 4 <= 5 doğru → toplam = 6 + 4 = 10.

Adım 5 / 10

DEĞİŞKENLER

toplam = 10

i = 4

ÇIKTI



Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
2 for (int i = 1; i <= 5; i++) {
3     toplam = toplam + i;
4 }
5 System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



toplam

15

10 + 5 = 15

i = 5 : 5 <= 5 doğru → toplam = 10 + 5 = 15.

Adım 6 / 10

DEĞİŞKENLER

toplam = 15

i = 5

ÇIKTI



Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
2 ▶ for (int i = 1; i <= 5; i++) {
3     toplam = toplam + i;
4 }
5 System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



toplam

15

i 6 oldu: $6 \leq 5$ **yanlış** → döngü biter.

Adım 7 / 10

DEĞİŞKENLER

toplam = 15

i = 6

ÇIKTI



Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
2 for (int i = 1; i <= 5; i++) {
3     toplam = toplam + i;
4 }
5 ▶ System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



toplam

15

Sonuç yazdırılır: beş turda biriken toplam 15.

Adım 8 / 10

DEĞİŞKENLER

toplam = 15

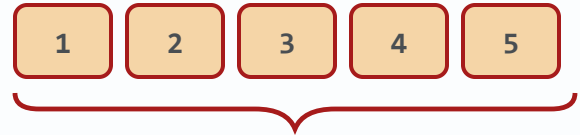
ÇIKTI

Buyruksal: 15

Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
2 for (int i = 1; i <= 5; i++) {
3     toplam = toplam + i;
4 }
5 System.out.println("Buyruksal: " + toplam);
6
7 ► int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



`rangeClosed(1, 5).sum()` → 15

Fonksiyonel: `rangeClosed(1, 5)` 1...5 sayılarını üretir, `sum()` hepsini toplar. Hiçbir değişken adım adım güncellenmez.

Adım 9 / 10

DEĞİŞKENLER

`toplam = 15`

`sonuc = 15`

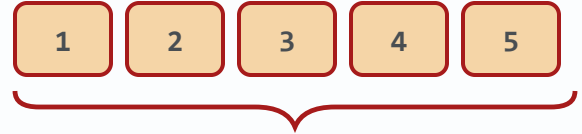
ÇIKTI

Buyruksal: 15

Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
2 for (int i = 1; i <= 5; i++) {
3     toplam = toplam + i;
4 }
5 System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
▶ System.out.println("Fonksiyonel: " + sonuc);
```

i değerleri



`rangeClosed(1, 5).sum() → 15`

İki yaklaşım aynı sonucu verir; fark, **nasıl** yapılacağını mı yoksa **ne** istendiğini mi yazdığımızdadır.

Adım 10 / 10

DEĞİŞKENLER

toplam = 15

sonuc = 15

ÇIKTI

```
Buyruksal: 15
Fonksiyonel: 15
```

Mantıksal ve Nesne Yönelimli

MANTIKSAL · PROLOG

```
% olgular
ebeveyn(ali, ayse).
ebeveyn(ayse, can).
% kural
dede(X, Z) :- ebeveyn(X, Y),
              ebeveyn(Y, Z).
% sorgu
?- dede(ali, can).
true.
```

Yalnızca **olgular** ve bir **kural** verilir; sorunun nasıl çözüleceğine (arama sırasına) Prolog kendisi karar verir.

Dikdortgen nesnesi hem **veriyi** (**en** , **boy**) hem de o veri üzerinde çalışan **metodu** (**alan()**) taşır.

NESNE YÖNELİMLİ · JAVA

```
public class Dikdortgen {
    double en, boy;           // veri (durum)

    Dikdortgen(double en, double boy) {
        this.en = en;
        this.boy = boy;
    }

    double alan() {           // davranış
        return en * boy;
    }

    public static void main(String[] args) {
        Dikdortgen d = new Dikdortgen(3, 4);
        System.out.println("Alan: " + d.alan());
    }
}
```

ÇIKTI

```
Alan: 12.0
```

Nesne Yönelimli: Kod İzleme

```
1 public class Dikdortgen {  
2     double en, boy;  
3     Dikdortgen(double en, double boy) {  
4         this.en = en;  
5         this.boy = boy;  
6     }  
7     double alan() {  
8         return en * boy;  
9     }  
10    public static void main(String[] args) {  
11        ▶ Dikdortgen d = new Dikdortgen(3, 4);  
12        System.out.println("Alan: " + d.alan());  
13    }  
14 }
```

Yığın (stack)

Heap

main

d

main çalışmaya başlar. Yığında main çerçevesi var; d henüz bir nesneyi göstermiyor.

Adım 1 / 7

DEĞİŞKENLER

d = -

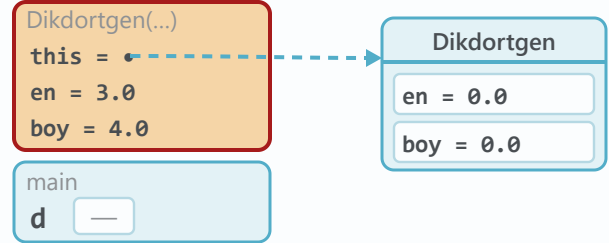
ÇIKTI

Nesne Yönelimli: Kod İzleme

```
1 public class Dikdortgen {  
2     double en, boy;  
3     ▶ Dikdortgen(double en, double boy) {  
4         this.en = en;  
5         this.boy = boy;  
6     }  
7     double alan() {  
8         return en * boy;  
9     }  
10    public static void main(String[] args) {  
11        Dikdortgen d = new Dikdortgen(3, 4);  
12        System.out.println("Alan: " + d.alan());  
13    }  
14 }
```

Yığın (stack)

Heap



new heap'te nesneye yer ayırır; alanlar varsayılan **0.0**. Yapıcı çağrılır: 3 ve 4, **double** 'a genişletilir.

Adım 2 / 7

DEĞİŞKENLER

this = • en = 3.0 boy = 4.0

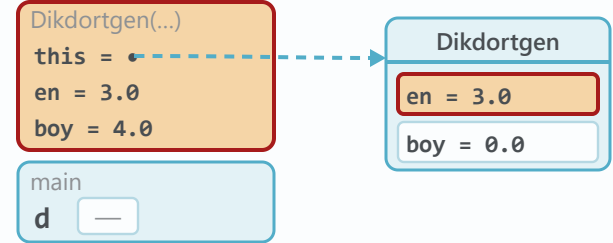
ÇIKTI

Nesne Yönelimli: Kod İzleme

```
1 public class Dikdortgen {  
2     double en, boy;  
3     Dikdortgen(double en, double boy) {  
4         ▶ this.en = en;  
5         this.boy = boy;  
6     }  
7     double alan() {  
8         return en * boy;  
9     }  
10    public static void main(String[] args) {  
11        Dikdortgen d = new Dikdortgen(3, 4);  
12        System.out.println("Alan: " + d.alan());  
13    }  
14 }
```

Yığın (stack)

Heap



`this.en` nesnenin alanıdır, sağdaki `en` parametredir: alan 3.0 olur.

Adım 3 / 7

DEĞİŞKENLER

`this = •` `en = 3.0` `boy = 4.0`

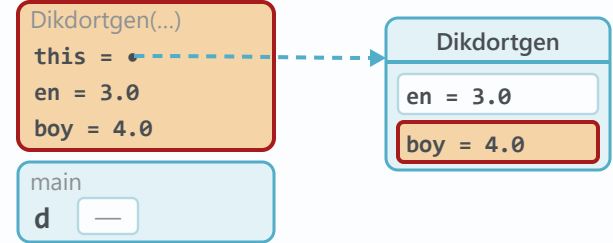
ÇIKTI

Nesne Yönelimli: Kod İzleme

```
1 public class Dikdortgen {  
2     double en, boy;  
3     Dikdortgen(double en, double boy) {  
4         this.en = en;  
5         this.boy = boy;  
6     }  
7     double alan() {  
8         return en * boy;  
9     }  
10    public static void main(String[] args) {  
11        Dikdortgen d = new Dikdortgen(3, 4);  
12        System.out.println("Alan: " + d.alan());  
13    }  
14 }
```

Yığın (stack)

Heap



boy alanı 4.0 olur. Yapıcı biter, çerçevesi yığından kalkar.

Adım 4 / 7

DEĞİŞKENLER

this = •

en = 3.0

boy = 4.0

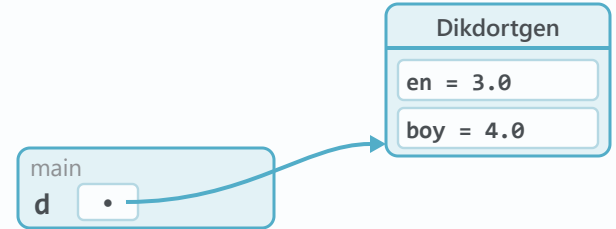
ÇIKTI

Nesne Yönelimli: Kod İzleme

```
1 public class Dikdortgen {  
2     double en, boy;  
3     Dikdortgen(double en, double boy) {  
4         this.en = en;  
5         this.boy = boy;  
6     }  
7     double alan() {  
8         return en * boy;  
9     }  
10    public static void main(String[] args) {  
11        ▶ Dikdortgen d = new Dikdortgen(3, 4);  
12        System.out.println("Alan: " + d.alan());  
13    }  
14 }
```

Yığın (stack)

Heap



new ifadesinin değeri nesnenin referansıdır; **d** artık bu nesneyi gösterir.

Adım 5 / 7

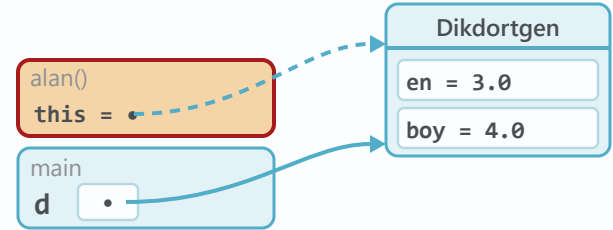
DEĞİŞKENLER**d = • (Dikdortgen)****ÇIKTI**

Nesne Yönelimli: Kod İzleme

```
1 public class Dikdortgen {  
2     double en, boy;  
3     Dikdortgen(double en, double boy) {  
4         this.en = en;  
5         this.boy = boy;  
6     }  
7     double alan() {  
8         return en * boy;  
9     }  
10    public static void main(String[] args) {  
11        Dikdortgen d = new Dikdortgen(3, 4);  
12        System.out.println("Alan: " + d.alan());  
13    }  
14 }
```

Yığın (stack)

Heap



d.alan() : this ile d aynı nesneyi gösterir. $en * boy = 3.0 \times 4.0 = 12.0$ döndürülür.

Adım 6 / 7

DEĞİŞKENLER

this = •

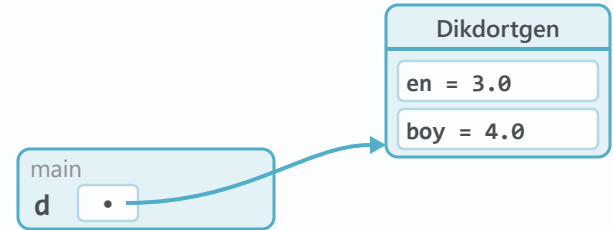
ÇIKTI

Nesne Yönelimli: Kod İzleme

```
1 public class Dikdortgen {  
2     double en, boy;  
3     Dikdortgen(double en, double boy) {  
4         this.en = en;  
5         this.boy = boy;  
6     }  
7     double alan() {  
8         return en * boy;  
9     }  
10    public static void main(String[] args) {  
11        Dikdortgen d = new Dikdortgen(3, 4);  
12        System.out.println("Alan: " + d.alan());  
13    }  
14 }
```

Yığın (stack)

Heap



Dönen 12.0 dizgiyle birleştirilir: **Alan: 12.0**. Veri (en, boy) ve davranış (alan) aynı nesnede durur.

Adım 7 / 7

DEĞİŞKENLER

d = • (Dikdortgen)

ÇIKTI

Alan: 12.0

Temel Yapılar ve Söz Dizimi

Programlama dilleri ile doğal diller birçok ortak yön paylaşır.

	Türkçe (doğal dil)	Java (programlama dili)
Temel yapılar	kelimeler, noktalama işaretleri	sayılar, dizgiler, basit operatörler (+ - * /), anahtar kelimeler
Söz dizimi (syntax)	"kedi köpek çocuk": kelimeler doğru ama cümle kurulumu yok → geçersiz	hangi karakter ve sembol dizilerinin düzgün olduğunu belirler

Söz dizimi dilin **kurallarını** belirler ve dilin anlaşılabilirliğini etkiler. Derleyici, kurala uymayan bir sembol dizisini kabul etmez:

✓ GEÇERLİ

```
int x = 5;
```

✗ GEÇERSİZ: SIRALAMA YANLIŞ

```
int = x 5;
```

Hata.java:3: error: not a statement

```
    int = x 5;
```

```
    ^
```

```
...
```

```
4 errors
```

Statik Anlam

- **Statik anlam** (static semantics): söz dizimi doğru olan bir ifadenin **anlamlı** olup olmadığını, program çalışmadan **önce** denetler.
- İngilizce: "I are big" biçim olarak `<isim> <fiil> <isim>` kalıbına uyar; ama "I" tekil, "are" çoğuldur → dil açısından **geçersiz**.
- Java: `<değer> <işleç> <değer>` geçerli bir kalıptır; ama bir sayıyı bir dizgiye bölmenin anlamı yoktur → **statik anlam hatası**.

```
double sonuc = 2.3 / "abc";
```

```
error: bad operand types for binary operator '/'  
first type: double  
second type: String
```

```
int x = "beş";
```

```
error: incompatible types:  
String cannot be converted to int
```



Java **statik tipli** bir dildir: derleyici her operatörün işlenenlerinin tipini denetler. Bu tür hatalar program hiç çalışmadan yakalanır.

Anlam (Semantics)

- **Anlam** (semantics): söz dizimi doğru bir ifadenin **ne anlama geldiğini** belirler.
- Doğal dillerde **çok anlamlılık** olabilir: "They saw the man with the telescope."
 - adamı teleskopla gördüler, ya da
 - teleskobu olan adamı gördüler.
- Programlama dilinde her geçerli ifadenin **tek bir anlamı** vardır.
- Ancak bu anlam, programcının **beklediği** şey olmayabilir: yanlış bir işlem beklenmeyen bir sonuç üretir.

```
int a = 7, b = 8;  
int ortalama = (a + b) / 2;  
System.out.println(ortalama);           // 7.5 beklenirken  
System.out.println(a + b / 2);          // öncelik  
System.out.println((a + b) / 2.0);      // doğru
```

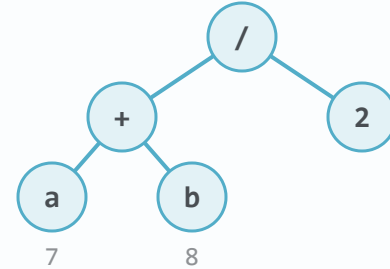
ÇIKTI

```
7  
11  
7.5
```

7 ile 8'in ortalaması 7.5'tir. İlk satır **tam sayı bölmesi** yapar, ikinci satırda `/` işlemi `+` 'dan önce yapılır. Derleyici bu satırlarda hata görmez.

Anlam Hatası: Kod İzleme

```
► int a = 7, b = 8;  
2 int ortalama = (a + b) / 2;  
3 System.out.println(ortalama);           // 7.5 değil  
4 System.out.println(a + b / 2);           // öncelik  
5 System.out.println((a + b) / 2.0);       // doğru
```



`a` = 7, `b` = 8; ikisi de `int`. Ağaç, `(a + b) / 2` ifadesinin yapısını gösterir.

Adım 1 / 7

DEĞİŞKENLER

`a = 7`

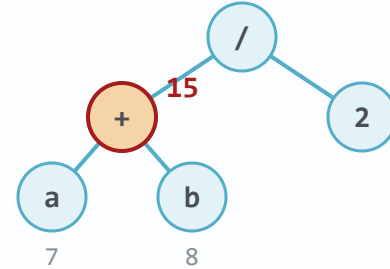
`b = 8`

ÇIKTI



Anlam Hatası: Kod İzleme

```
1 int a = 7, b = 8;  
▶ int ortalama = (a + b) / 2;  
3 System.out.println(ortalama);           // 7.5 değil  
4 System.out.println(a + b / 2);         // öncelik  
5 System.out.println((a + b) / 2.0);     // doğru
```



Parantez önce: **a + b** = 15 (int).

Adım 2 / 7

DEĞİŞKENLER

a = 7

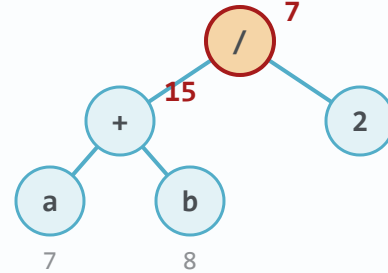
b = 8

ÇIKTI



Anlam Hatası: Kod İzleme

```
1 int a = 7, b = 8;  
▶ int ortalama = (a + b) / 2;  
3 System.out.println(ortalama);           // 7.5 değil  
4 System.out.println(a + b / 2);         // öncelik  
5 System.out.println((a + b) / 2.0);     // doğru
```



15 / 2 : iki işlenen de `int` → **tam sayı bölmesi**, kesir atılır: 7.

Adım 3 / 7

DEĞİŞKENLER

a = 7

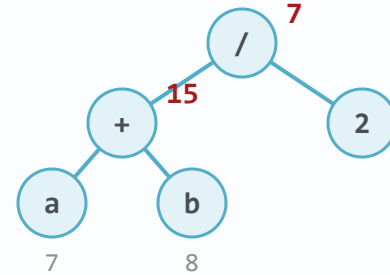
b = 8

ortalama = 7

ÇIKTI

Anlam Hatası: Kod İzleme

```
1 int a = 7, b = 8;  
2 int ortalama = (a + b) / 2;  
▶ System.out.println(ortalama); // 7.5 değil  
4 System.out.println(a + b / 2); // öncelik  
5 System.out.println((a + b) / 2.0); // doğru
```



7 yazılır. Program hatasız çalıştı ama anlamı beklenenden farklı.

Adım 4 / 7

DEĞİŞKENLER

a = 7

b = 8

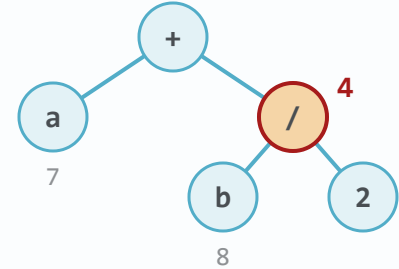
ortalama = 7

ÇIKTI

7

Anlam Hatası: Kod İzleme

```
1 int a = 7, b = 8;  
2 int ortalama = (a + b) / 2;  
3 System.out.println(ortalama); // 7.5 değil  
▶ System.out.println(a + b / 2); // öncelik  
5 System.out.println((a + b) / 2.0); // doğru
```



Parantez yok: `/` önceliği `+` operatöründen yüksek. Önce `b / 2 = 4`.

Adım 5 / 7

DEĞİŞKENLER

a = 7

b = 8

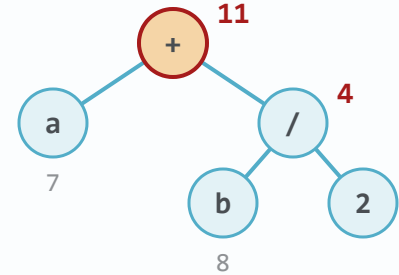
ortalama = 7

ÇIKTI

7

Anlam Hatası: Kod İzleme

```
1 int a = 7, b = 8;  
2 int ortalama = (a + b) / 2;  
3 System.out.println(ortalama);           // 7.5 değil  
▶ System.out.println(a + b / 2);         // öncelik  
5 System.out.println((a + b) / 2.0);     // doğru
```



Sonra `7 + 4` = 11 yazılır.

Adım 6 / 7

DEĞİŞKENLER

a = 7

b = 8

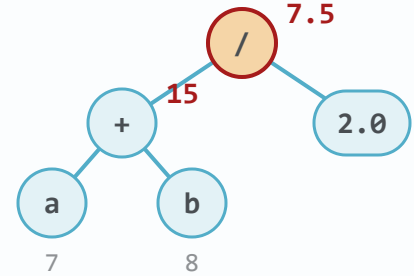
ortalama = 7

ÇIKTI

```
7  
11
```

Anlam Hatası: Kod İzleme

```
1 int a = 7, b = 8;  
2 int ortalama = (a + b) / 2;  
3 System.out.println(ortalama);           // 7.5 değil  
4 System.out.println(a + b / 2);         // öncelik  
▶ System.out.println((a + b) / 2.0);    // doğru
```



2.0 bir double : 15 de double'a genişletilir, $15 / 2.0 = 7.5$.

Adım 7 / 7

DEĞİŞKENLER

a = 7

b = 8

ortalama = 7

ÇIKTI

```
7  
11  
7.5
```

Neler Yanlış Gidebilir?

Hata türü	Ne zaman fark edilir?	Java örneği	Belirti
Söz dizimi	derlemede (derleyici)	<code>System.out.println("Merhaba")</code>	<code>';' expected</code>
Statik anlam	derlemede (tip denetimi)	<code>int x = "beş";</code>	<code>incompatible types</code>
Çalışma zamanı	program çalışırken	<code>a / b</code> (b sıfırken)	çökme: <code>ArithmeticException</code>
Mantık	derleyici fark etmez; ancak testle	<code>(a + b) / 2</code> ile ortalama	yanlış sonuç, sonsuz döngü

- **Söz dizimi hataları** yazım hataları gibi açıkça tanımlanmış kurallara aykırılıktır; bilgisayar tarafından **kolayca** tespit edilir.
- **Statik anlam hataları** program çalışmadan önce dile özgü denetimlerle aranır. Bu denetimi zayıf olan dillerde yakalanmayan hatalar programın davranışını **öngörülemez** kılar.
- Söz dizimi ve statik anlam açısından **kusursuz** bir program bile beklenen sonucu üretmeyebilir: çöker, **sonsuz döngüye** girer ya da istenmeyen bir sonuç verir.

Hata Mesajlarını Okumak

DERLEME HATASI

```
public class Selam {  
    public static void main(String[] args) {  
        System.out.println("Merhaba")  
    }  
}
```

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

System.out.println("Merhaba")

^

1 error

- `Selam.java:3` : dosya ve **satır** numarası
- `error: ';' expected` : ne bekleniyordu
- `^` işareti: hatanın **konumu**
- Hata varsa `.class` dosyası oluşmaz; program hiç çalışmaz.

ÇALIŞMA ZAMANI HATASI (İSTİSNA)

```
public class Bolme {  
    public static void main(String[] args) {  
        int a = 10, b = 0;  
        System.out.println(a / b);  
    }  
}
```

Exception in thread "main"

java.lang.ArithmeticException: / by zero

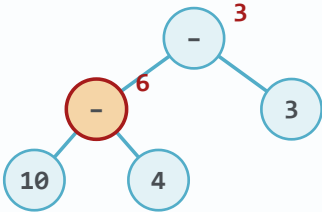
at Bolme.main(Bolme.java:4)

- Kod derlenir, ama çalışırken **istisna** (exception) fırlatılır.
- `/ by zero` : tam sayı sıfıra bölünemez.
- `(Bolme.java:4)` : hatanın olduğu satır.
- İstisnaları yakalamak Bölüm 6'nın konusudur.

İpucu: Birden çok hata listelenirse **ilkinden** başlayın; sonrakiler çoğu zaman ilk hatanın yan etkisidir.

İfadeler ve İşlem Önceliği

- **İfade** (expression): bir hesaplamayı tanımlayan, bir **değer** üreten yapıdır: `3 + 4`, `x * 2`.
- İfadeler belirli kurallarla değerlendirilir; operatörlerin **önceliği** (precedence) ve **yönü** (associativity) tanımlıdır.
- Çarpma, bölme ve kalan; toplama ve çıkarmadan **önce** yapılır.
- Aynı öncelikteki operatörler **soldan sağa** işlenir:
`10 - 4 - 3 = 3`.
- Öncelikleri değiştirmek için **parantez** kullanılır:
`(2 + 3) * 4 = 20`.



önce soldaki: $(10 - 4) - 3 = 3$

Öncelik	Operatör	Yön
1	() parantez	içten dışa
2	* / %	soldan sağa
3	+ -	soldan sağa
4	= atama	sağdan sola

Üs alma: matematikte (ve Python'daki `**` operatöründe) üs **sağdan sola** işlenir: $2^{3^2} = 2^9 = 512$. Java'da üs operatörü **yoktur**; `Math.pow(2, 3)` kullanılır ve sonuç `double` 'dır (8.0).

İFADELER VE DEĞİŞKENLER

Örnek İfadeler

İfade	Java'da yazımı	Sonuç	Not
5	5	5	tek bir değer de bir ifadedir
3 + 4	3 + 4	7	
44 / 2	44 / 2	22	
2 ³	Math.pow(2, 3)	8.0	Java'da 2 ** 3 derlenmez
34 + 56	34 + 56	90	
(72 - 32) / 9 × 5	(72 - 32) / 9 * 5	20	tam sayı bölmesi: 40 / 9 = 4
(72 - 32) / 9 × 5	(72.0 - 32) / 9 * 5	22.22222222222222	bir işlenen double ise sonuç da double
7 / 2	7 / 2 · 7 / 2.0	3 · 3.5	kesir kısmı atılır / korunur
17 mod 5	17 % 5	2	kalan operatörü

(72 - 32) / 9 × 5, Fahrenheit'tan (72 °F) Celsius'a dönüşümdür: doğru cevap ≈ 22.2 °C. Tam sayılarla yazılınca sessizce 20 çıkar.

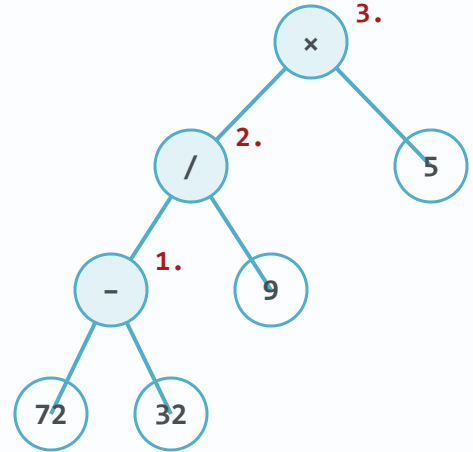
İfade Değerlendirme Adım Adım

Bilgisayar bir ifadeyi tek seferde değil, öncelik kurallarına göre **adım adım** sadeleştirir.

$$(72 - 32) / 9 * 5$$

Önce **parantez** içi hesaplanır: $72 - 32 = 40$.

Adım 1 / 8



İfade ağacı: alttaki işlem önce yapılır; numaralar işlem sırasıdır.

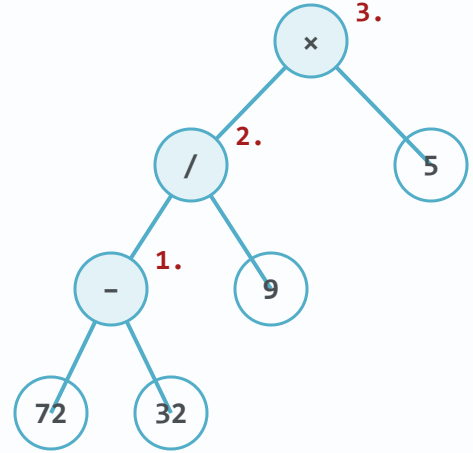
İfade Değerlendirme Adım Adım

Bilgisayar bir ifadeyi tek seferde değil, öncelik kurallarına göre **adım adım** sadeleştirir.

40 / 9 * 5

/ ile * aynı önceliktedir → **soldan sağa**: önce 40 / 9.

Adım 2 / 8



İfade ağacı: alttaki işlem önce yapılır; numaralar işlem sırasıdır.

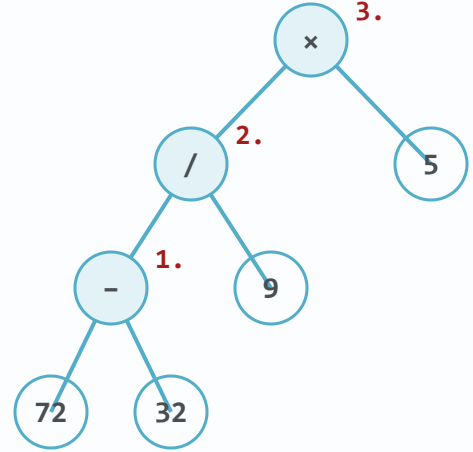
İfade Değerlendirme Adım Adım

Bilgisayar bir ifadeyi tek seferde değil, öncelik kurallarına göre **adım adım** sadeleştirir.

4 * 5 (40 / 9 = 4, kalan 4 atıldı)

İki işlenen de `int` → **tam sayı bölmesi**: 4.44... değil,
4.

Adım 3 / 8



İfade ağacı: alttaki işlem önce yapılır; numaralar işlem sırasıdır.

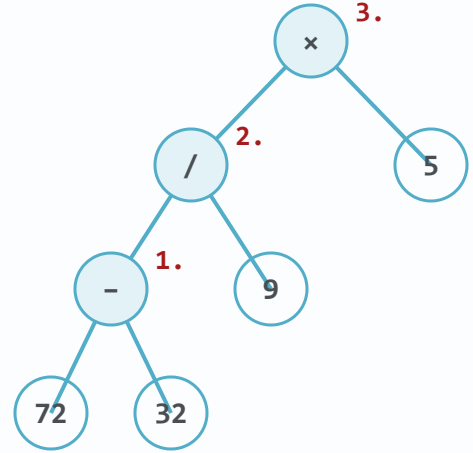
İfade Değerlendirme Adım Adım

Bilgisayar bir ifadeyi tek seferde değil, öncelik kurallarına göre **adım adım** sadeleştirir.

20

Sonuç **20**. Matematiksel olarak beklenen 22.22... idi:
bir **anlam** (mantık) hatası.

Adım 4 / 8



İfade ağacı: alttaki işlem önce yapılır; numaralar işlem sırasıdır.

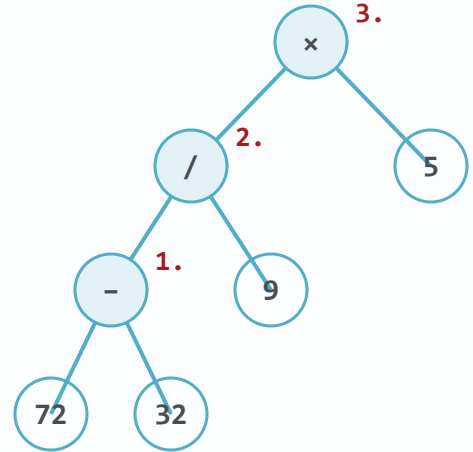
İfade Değerlendirme Adım Adım

Bilgisayar bir ifadeyi tek seferde değil, öncelik kurallarına göre **adım adım** sadeleştirir.

(72.0 - 32) / 9 * 5

Şimdi 72.0 yazalım: `double` - `int` işleminde
`int` önce `double` 'a genişletilir; sonuç 40.0.

Adım 5 / 8



İfade ağacı: alttaki işlem önce yapılır; numaralar işlem sırasıdır.

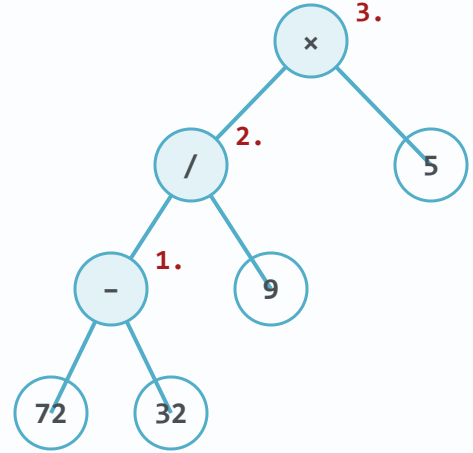
İfade Değerlendirme Adım Adım

Bilgisayar bir ifadeyi tek seferde değil, öncelik kurallarına göre **adım adım** sadeleştirir.

$$40.0 / 9 * 5$$

40.0 / 9 artık **gerçek bölme**dir: 4.4444444444444445.

Adım 6 / 8



İfade ağacı: alttaki işlem önce yapılır; numaralar işlem sırasıdır.

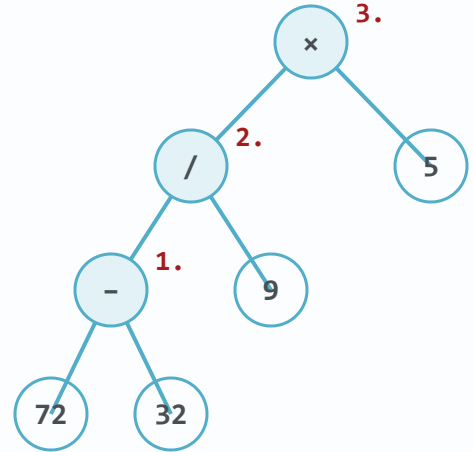
İfade Değerlendirme Adım Adım

Bilgisayar bir ifadeyi tek seferde değil, öncelik kurallarına göre **adım adım** sadeleştirir.

4.4444444444444445 * 5

Çarpma: double × int → double .

Adım 7 / 8



İfade ağacı: alttaki işlem önce yapılır; numaralar işlem sırasıdır.

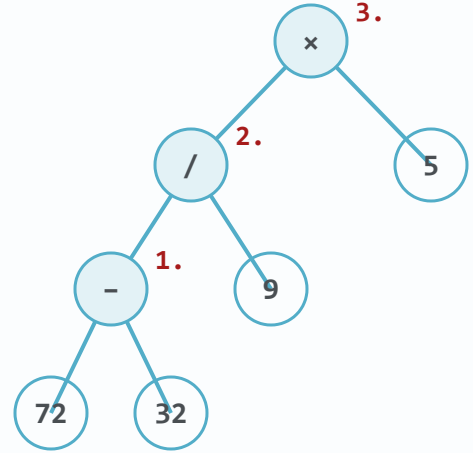
İfade Değerlendirme Adım Adım

Bilgisayar bir ifadeyi tek seferde değil, öncelik kurallarına göre **adım adım** sadeleştirir.

22.22222222222222

Doğru sonuç. Ondalıklı sonuç gerekiyorsa işlenenlerden en az biri `double` olmalıdır.

Adım 8 / 8

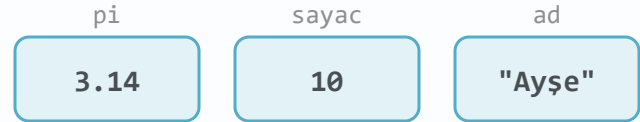


İfade ağacı: alttaki işlem önce yapılır; numaralar işlem sırasıdır.

Değişkenler

- **Değişken** (variable): programda değerleri **saklamak ve yönetmek** için kullanılan, adı olan bir bellek alanıdır.
- Değişkenler program içindeki verilere bir **başvuru** (ad) sağlar: değeri her yerde adıyla kullanırız.
- Değer atamak için `değişkenAdı = ifade` yazılır. Java'da değişken önce **tipiyle tanımlanır**:

```
double pi = 3.14;      // tanımla ve ata
int sayac = 10;
String ad = "Ayşe";
sayac = sayac + 1;     // içerik değişir: 11
```



`double pi = 3.14;` → `pi` adlı, `double` tipinde bir kutu oluşturulur ve içine 3.14 yazılır. Kutunun tipi sonradan değişmez, **içeriği** değişebilir.

Değişken Adlandırma Kuralları

- Tek bir **kelime** olmalıdır (boşluk içermez).
- Harf, rakam ve alt çizgi** (`_`) içerebilir; Java `$` işaretine de izin verir.
- Rakamla başlayamaz.**
- Java'nın **anahtar kelimeleri** (`int` , `class` , `if` ...) ad olamaz.
- Büyük/küçük harf **duyarlıdır**: `Sayi` ile `sayi` farklıdır.
- Gelenek: **camelCase** (`ortalamaNot`), anlamlı adlar; Türkçe karakterden kaçının.

```
int Sayi = 5;  
System.out.println(sayi);
```

Ad	Geçerli?	Neden / derleyici
<code>toplam2</code>	✓	rakam başta değil
<code>_sayac</code> , <code>\$fiyat</code>	✓	izinli ama alışılmadık
<code>ortalamaNot</code>	✓	önerilen camelCase
<code>2sayi</code>	✗	rakamla başlar → <code>not a statement</code>
<code>ogrenci sayisi</code>	✗	boşluk → <code> ';' expected</code>
<code>class</code>	✗	anahtar kelime → <code><identifier> expected</code>

```
error: cannot find symbol  
symbol:   variable sayi
```

Dinamik ve Statik Tipleme

Dinamik tipli diller (Python, JavaScript)

- Değişkenin tipi önceden **belirtilmez**; değer atanınca belirlenir.
- Tip **çalışma zamanında** değişebilir.
- Esneklik sağlar, ama hatalar ancak **çalışırken** ortaya çıkar.

PYTHON

```
x = 5           # x şimdi int
x = "beş"       # artık str: sorun yok
print(x + 1)    # çalışırken:
# TypeError: can only concatenate
#   str (not "int") to str
```

Statik tipli diller (Java, C, C#)

- Değişkenin tipi **açıkça** belirtilir ve değişmez.
- Dizgi ile tam sayı gibi uyumsuz işlemler **derleme zamanında** hata olarak bulunur.
- Daha güvenli; derleyici hızlı kod üretebilir.

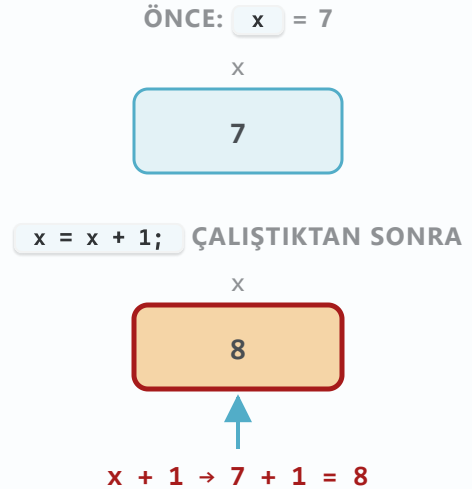
JAVA

```
int x = 5;
x = "beş";    // derlenmez:
// error: incompatible types:
//   String cannot be converted to int
```

Java 10+: `var adet = 3;` tipi yazmadan tanımlar, ama tip yine derlemede (`int` olarak) sabitlenir; Java statik tipli kalır.

Atama İşlemi

- **Atama** (assignment) bir değişkenin değerini **değiştirmek** için kullanılır.
- Atama **iki adımda** gerçekleşir:
 1. **Sağ taraftaki ifade** hesaplanır.
 2. Değişkenin içeriği hesaplanan **yeni değerle** güncellenir; eski değer kaybolur.
- `x = 5;` → sağdaki 5 değerlendirilir ve `x` adlı değişkende saklanır.
- `=` matematikteki **eşitlik değildir**: `x = x + 1` cebirde çelişkidir, programda ise "x'i bir artır" demektir.
- Sol tarafta **yalnızca bir değişken** olabilir: `5 = x;` derlenmez (`unexpected type`).



Atama: Kodu Adım Adım İzle

```
1 public class Atama {  
2     public static void main(String[] args)  
3     {  
4         ▶ int x = 5;  
5         x = x + 3;  
6         int y = x * 2;  
7         x = y - x;  
8         System.out.println("x = " + x);  
9         System.out.println("y = " + y);  
10    }  
}
```



x için bellekte yer ayrılır ve içine 5 yazılır.

Adım 1 / 8

DEĞİŞKENLER

x = 5

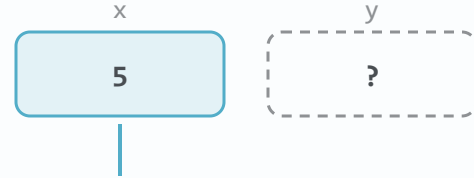
y = -

ÇIKTI



Atama: Kodu Adım Adım İzle

```
1 public class Atama {  
2     public static void main(String[] args)  
3     {  
4         int x = 5;  
5         x = x + 3;  
6         int y = x * 2;  
7         x = y - x;  
8         System.out.println("x = " + x);  
9         System.out.println("y = " + y);  
10    }  
}
```



$$x + 3 \rightarrow 5 + 3 = 8$$

1. adım: **sağ taraf** hesaplanır. `x` 'in o anki değeri (5) okunur: $5 + 3 = 8$.

Adım 2 / 8

DEĞİŞKENLER

x = 5

y = -

ÇIKTI



Atama: Kodu Adım Adım İzle

```
1 public class Atama {  
2     public static void main(String[] args)  
3     {  
4         int x = 5;  
5         x = x + 3;  
6         int y = x * 2;  
7         x = y - x;  
8         System.out.println("x = " + x);  
9         System.out.println("y = " + y);  
10    }  
}
```



2. adım: sonuç **x** 'e yazılır; eski değer 5 kaybolur.

Adım 3 / 8

DEĞİŞKENLER

x = 8

y = -

ÇIKTI



Atama: Kodu Adım Adım İzle

```
1 public class Atama {  
2     public static void main(String[] args)  
3     {  
4         int x = 5;  
5         x = x + 3;  
6         int y = x * 2;  
7         x = y - x;  
8         System.out.println("x = " + x);  
9         System.out.println("y = " + y);  
10    }  
}
```

x

8

y

16

$x * 2 = 8 \times 2 = 16$ hesaplanır ve yeni değişken **y** 'ye yazılır. **x** değişmez.

Adım 4 / 8

DEĞİŞKENLER

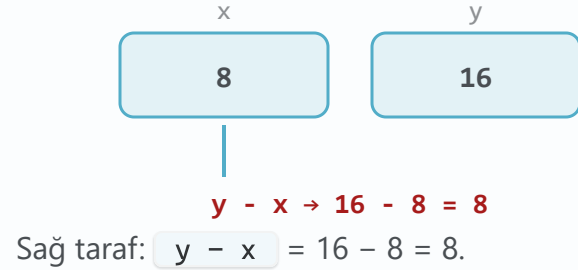
x = 8

y = 16

ÇIKTI

Atama: Kodu Adım Adım İzle

```
1 public class Atama {  
2     public static void main(String[] args)  
3     {  
4         int x = 5;  
5         x = x + 3;  
6         int y = x * 2;  
7         x = y - x;  
8         System.out.println("x = " + x);  
9         System.out.println("y = " + y);  
10    }  
}
```



Adım 5 / 8

DEĞİŞKENLER

x = 8

y = 16

ÇIKTI



Atama: Kodu Adım Adım İzle

```
1 public class Atama {  
2     public static void main(String[] args)  
3     {  
4         int x = 5;  
5         x = x + 3;  
6         int y = x * 2;  
7         x = y - x;  
8         System.out.println("x = " + x);  
9         System.out.println("y = " + y);  
10    }  
}
```



Sonuç `x` 'e yazılır. Değer yine 8: atama yapıldı ama içerik değişmedi.

Adım 6 / 8

DEĞİŞKENLER

x = 8

y = 16

ÇIKTI



Atama: Kodu Adım Adım İzle

```
1 public class Atama {  
2     public static void main(String[] args)  
3     {  
4         int x = 5;  
5         x = x + 3;  
6         int y = x * 2;  
7         x = y - x;  
8         System.out.println("x = " + x);  
9         System.out.println("y = " + y);  
10    }  
}
```

x

8

y

16

"x = " + x dizgi birleştirmedir: x = 8 yazılır.

Adım 7 / 8

DEĞİŞKENLER

x = 8

y = 16

ÇIKTI

x = 8

Atama: Kodu Adım Adım İzle

```
1 public class Atama {  
2     public static void main(String[] args)  
3     {  
4         int x = 5;  
5         x = x + 3;  
6         int y = x * 2;  
7         x = y - x;  
8         System.out.println("x = " + x);  
9         System.out.println("y = " + y);  
10    }  
}
```

x

8

y

16

Program biter. Son değerler: x = 8, y = 16.

Adım 8 / 8

DEĞİŞKENLER

x = 8

y = 16

ÇIKTI

x = 8

y = 16

Koşullu İfadeler

- **Koşullu ifade** (boolean expression) programın **hangi yolu** izleyeceğini belirler.
- Sonucu yalnızca `true` (doğru) ya da `false` (yanlış) olabilir; Java'da tipi `boolean` 'dır.
- **Atama bir koşul değildir:** `x = 100` değer verir, `x == 100` karşılaştırır.
- **Karşılaştırma operatörleri** değerleri karşılaştırır: `x >= 5`, x'in 5 veya daha büyük olup olmadığını denetler.
- **Mantıksal operatörler** koşulları birleştirir ya da tersine çevirir: `!true` → `false`.
- **Birleşik koşullar:** `3 < 4 && 5 < 6` yalnızca iki karşılaştırma da doğruysa `true` 'dur.

Operatör	Anlam	Örnek (x = 7)
<code>==</code> <code>!=</code>	eşit / eşit değil	<code>x == 100</code> → false
<code><</code> <code>></code>	küçük / büyük	<code>x < 10</code> → true
<code><=</code> <code>>=</code>	küçük-eşit / büyük-eşit	<code>x >= 5</code> → true
<code>&&</code>	ve (AND)	<code>x > 5 && x < 6</code> → false
<code> </code>	veya (OR)	<code>x < 0 x > 5</code> → true
<code>!</code>	değil (NOT)	<code>!(x == 7)</code> → false

```
if (x = 100) { }
```

error: incompatible types:
int cannot be converted to boolean

Koşullar: Kodu Adım Adım İzle

```
► int x = 7;
2 boolean buyukMu = x >= 5;
3 System.out.println(buyukMu);
4 System.out.println(x == 100);
5 System.out.println(!true);
6 System.out.println(3 < 4 && 5 < 6);
7 if (x >= 5 && x < 10) {
8     System.out.println("x 5 ile 9 arasında");
9 } else {
10     System.out.println("koşul sağlanmadı");
11 }
```

int x = 7

x değişkenine 7 atanır.

Adım 1 / 8

DEĞİŞKENLER

x = 7

ÇIKTI

Koşullar: Kodu Adım Adım İzle

```
1 int x = 7;
2 boolean buyukMu = x >= 5;
3 System.out.println(buyukMu);
4 System.out.println(x == 100);
5 System.out.println(!true);
6 System.out.println(3 < 4 && 5 < 6);
7 if (x >= 5 && x < 10) {
8     System.out.println("x 5 ile 9 arasında");
9 } else {
10     System.out.println("koşul sağlanmadı");
11 }
```

`x >= 5` → `7 >= 5` → `true`

Karşılaştırmanın sonucu bir `boolean` değerdir; `buyukMu` 'da saklanır.

Adım 2 / 8

DEĞİŞKENLER

`x = 7`

`buyukMu = true`

ÇIKTI



Koşullar: Kodu Adım Adım İzle

```
1 int x = 7;
2 boolean buyukMu = x >= 5;
3 ► System.out.println(buyukMu);
4 System.out.println(x == 100);
5 System.out.println(!true);
6 System.out.println(3 < 4 && 5 < 6);
7 if (x >= 5 && x < 10) {
8     System.out.println("x 5 ile 9 arasında");
9 } else {
10     System.out.println("koşul sağlanmadı");
11 }
```

buyukMu → true

true yazdırılır.

Adım 3 / 8

DEĞİŞKENLER

x = 7

buyukMu = true

ÇIKTI

true

Koşullar: Kodu Adım Adım İzle

```
1 int x = 7;
2 boolean buyukMu = x >= 5;
3 System.out.println(buyukMu);
4 ► System.out.println(x == 100);
5 System.out.println(!true);
6 System.out.println(3 < 4 && 5 < 6);
7 if (x >= 5 && x < 10) {
8     System.out.println("x 5 ile 9 arasında");
9 } else {
10     System.out.println("koşul sağlanmadı");
11 }
```

`x == 100` → `7 == 100` → **false**

`==` karşılaştırır, `x` 'i değiştirmez.

Adım 4 / 8

DEĞİŞKENLER

`x = 7`

`buyukMu = true`

ÇIKTI

true
false

Koşullar: Kodu Adım Adım İzle

```
1 int x = 7;
2 boolean buyukMu = x >= 5;
3 System.out.println(buyukMu);
4 System.out.println(x == 100);
5 ▶ System.out.println(!true);
6 System.out.println(3 < 4 && 5 < 6);
7 if (x >= 5 && x < 10) {
8     System.out.println("x 5 ile 9 arasında");
9 } else {
10     System.out.println("koşul sağlanmadı");
11 }
```

!true → **false**

! mantıksal değeri tersine çevirir.

Adım 5 / 8

DEĞİŞKENLER

x = 7

buyukMu = **true**

ÇIKTI

```
true
false
false
```

Koşullar: Kodu Adım Adım İzle

```
1 int x = 7;
2 boolean buyukMu = x >= 5;
3 System.out.println(buyukMu);
4 System.out.println(x == 100);
5 System.out.println(!true);
6 ► System.out.println(3 < 4 && 5 < 6);
7 if (x >= 5 && x < 10) {
8     System.out.println("x 5 ile 9 arasında");
9 } else {
10     System.out.println("koşul sağlanmadı");
11 }
```

3 < 4 && 5 < 6 → true

Önce iki karşılaştırma (yüksek öncelik),
sonra `&&` : ikisi de doğru → `true` .

Adım 6 / 8

DEĞİŞKENLER

x = 7

buyukMu = true

ÇIKTI

```
true
false
false
true
```

Koşullar: Kodu Adım Adım İzle

```
1 int x = 7;
2 boolean buyukMu = x >= 5;
3 System.out.println(buyukMu);
4 System.out.println(x == 100);
5 System.out.println(!true);
6 System.out.println(3 < 4 && 5 < 6);
7 ► if (x >= 5 && x < 10) {
8     System.out.println("x 5 ile 9 arasında");
9 } else {
10     System.out.println("koşul sağlanmadı");
11 }
```

7 >= 5 && 7 < 10 → true

if koşulu doğru → **ilk blok** seçilir;
else bloğu atlanacak.

Adım 7 / 8

DEĞİŞKENLER

x = 7

buyukMu = true

ÇIKTI

```
true
false
false
true
```

Koşullar: Kodu Adım Adım İzle

```
1 int x = 7;
2 boolean buyukMu = x >= 5;
3 System.out.println(buyukMu);
4 System.out.println(x == 100);
5 System.out.println(!true);
6 System.out.println(3 < 4 && 5 < 6);
7 if (x >= 5 && x < 10) {
8     System.out.println("x 5 ile 9 arasında");
9 } else {
10     System.out.println("koşul sağlanmadı");
11 }
```

yol: **if bloğu**

Satır 10 hiç çalışmaz.

Adım 8 / 8

DEĞİŞKENLER

x = 7

buyukMu = true

ÇIKTI

```
true
false
false
true
x 5 ile 9 arasında
```

DİZGİLER VE TİPLER

Dizgiler (Strings)

```
String ad = "Java";  
System.out.println(ad.length());           // uzunluk  
System.out.println("Ad\tSoyad");           // sekme  
System.out.println("Satır 1\nSatır 2");     // yeni satır  
System.out.println("Dedi ki: \"Merhaba\"");  
System.out.println(ad + " ile programlama"); // birleştirme  
System.out.println(ad.contains("a"));       // 'a' var mı?  
System.out.println(ad.indexOf('v'));        // konumu (0'dan)  
System.out.println(ad.toUpperCase());
```

Kaçış dizisi	Anlamı	Kaçış dizisi	Anlamı
\t	sekme (tab)	\"	çift tırnak
\n	yeni satır (newline)	\\	ters eğik çizgi

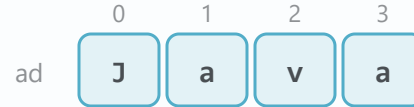
ÇIKTI

```
4  
Ad      Soyad  
Satır 1  
Satır 2  
Dedi ki: "Merhaba"  
Java ile programlama  
true  
2  
JAVA
```

- **Dizgi** (string) metin verisini temsil eder; Java'da **String** tipidir, **" "** içinde yazılır.
- **Kaçış** (escape) karakteri **** özel karakterleri yazmayı sağlar.
- Uzunluk bulma, **birleştirme** (concatenation), **arama** (search) gibi işlemler yapılır.

Dizgi Metotları: Kod İzleme

```
► String ad = "Java";  
2 int n = ad.length();  
3 int k = ad.indexOf('v');  
4 char c = ad.charAt(0);  
5 boolean varMi = ad.contains("a");  
6 String b = ad + " ile programlama";  
7 String u = ad.toUpperCase();  
8 System.out.println(n + " " + k + " " + c);  
9 System.out.println(b);  
10 System.out.println(u + " " + ad);
```



`ad` bir `String` nesnesini gösterir.
Karakterlerin indisi **0**'dan başlar.

Adım 1 / 10

DEĞİŞKENLER

ad = "Java"

ÇIKTI



Dizgi Metotları: Kod İzleme

```
1 String ad = "Java";  
2 ▶ int n = ad.length();  
3 int k = ad.indexOf('v');  
4 char c = ad.charAt(0);  
5 boolean varMi = ad.contains("a");  
6 String b = ad + " ile programlama";  
7 String u = ad.toUpperCase();  
8 System.out.println(n + " " + k + " " + c);  
9 System.out.println(b);  
10 System.out.println(u + " " + ad);
```

ad 0 1 2 3
 J a v a length() = 4

length() karakter sayısını verir: 4. Son indis 3'tür.

Adım 2 / 10

DEĞİŞKENLER

ad = "Java"

n = 4

ÇIKTI

Dizgi Metotları: Kod İzleme

```
1 String ad = "Java";
2 int n = ad.length();
3 ▶ int k = ad.indexOf('v');
4 char c = ad.charAt(0);
5 boolean varMi = ad.contains("a");
6 String b = ad + " ile programlama";
7 String u = ad.toUpperCase();
8 System.out.println(n + " " + k + " " + c);
9 System.out.println(b);
10 System.out.println(u + " " + ad);
```

ad

0	1	2	3
J	a	v	a

'v' → 2

`indexOf('v')` ilk 'v'nin indisini döndürür: 2.
Bulunamasa -1 dönerdi.

Adım 3 / 10

DEĞİŞKENLER

ad = "Java"

n = 4

k = 2

ÇIKTI

Dizgi Metotları: Kod İzleme

```
1 String ad = "Java";
2 int n = ad.length();
3 int k = ad.indexOf('v');
4 ▶ char c = ad.charAt(0);
5 boolean varMi = ad.contains("a");
6 String b = ad + " ile programlama";
7 String u = ad.toUpperCase();
8 System.out.println(n + " " + k + " " + c);
9 System.out.println(b);
10 System.out.println(u + " " + ad);
```

ad 0 1 2 3
 J a v a **charAt(0)**

charAt(0) 0. karakteri **char** olarak verir:
'J'.

Adım 4 / 10

DEĞİŞKENLER

ad = "Java"

n = 4

k = 2

c = 'J'

ÇIKTI

Dizgi Metotları: Kod İzleme

```
1 String ad = "Java";
2 int n = ad.length();
3 int k = ad.indexOf('v');
4 char c = ad.charAt(0);
5 ▶ boolean varMi = ad.contains("a");
6 String b = ad + " ile programlama";
7 String u = ad.toUpperCase();
8 System.out.println(n + " " + k + " " + c);
9 System.out.println(b);
10 System.out.println(u + " " + ad);
```

ad

0	1	2	3
J	a	v	a

"a" var

`contains("a")` : dizgide "a" geçiyor mu?
Evet → `true` .

Adım 5 / 10

DEĞİŞKENLER

ad = "Java"

n = 4

k = 2

c = 'J'

varMi = true

ÇIKTI

Dizgi Metotları: Kod İzleme

```
1 String ad = "Java";
2 int n = ad.length();
3 int k = ad.indexOf('v');
4 char c = ad.charAt(0);
5 boolean varMi = ad.contains("a");
6 ▶ String b = ad + " ile programlama";
7 String u = ad.toUpperCase();
8 System.out.println(n + " " + k + " " + c);
9 System.out.println(b);
10 System.out.println(u + " " + ad);
```

ad

0	1	2	3
J	a	v	a

b

"Java ile programlama"

+ birleştirme **yeni** bir dizgi üretir; **ad** değişmez.

Adım 6 / 10

DEĞİŞKENLER

ad = "Java"

n = 4

k = 2

c = 'J'

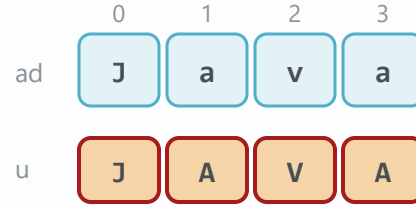
varMi = true

b = "Java ile programlama"

ÇIKTI

Dizgi Metotları: Kod İzleme

```
1 String ad = "Java";
2 int n = ad.length();
3 int k = ad.indexOf('v');
4 char c = ad.charAt(0);
5 boolean varMi = ad.contains("a");
6 String b = ad + " ile programlama";
7 String u = ad.toUpperCase();
8 System.out.println(n + " " + k + " " + c);
9 System.out.println(b);
10 System.out.println(u + " " + ad);
```



`toUpperCase()` de yeni bir dizgi döndürür: "JAVA". `ad` hâlâ "Java".

Adım 7 / 10

DEĞİŞKENLER

`ad = "Java"`

`n = 4`

`k = 2`

`c = 'J'`

`varMi = true`

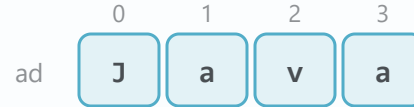
`u = "JAVA"`

ÇIKTI



Dizgi Metotları: Kod İzleme

```
1 String ad = "Java";
2 int n = ad.length();
3 int k = ad.indexOf('v');
4 char c = ad.charAt(0);
5 boolean varMi = ad.contains("a");
6 String b = ad + " ile programlama";
7 String u = ad.toUpperCase();
8 ▶ System.out.println(n + " " + k + " " + c);
9 System.out.println(b);
10 System.out.println(u + " " + ad);
```



n, k, c boşluklarla birleştirilip yazılır: 4
2 J .

Adım 8 / 10

DEĞİŞKENLER

ad = "Java"

n = 4

k = 2

c = 'J'

ÇIKTI

4 2 J

Dizgi Metotları: Kod İzleme

```
1 String ad = "Java";
2 int n = ad.length();
3 int k = ad.indexOf('v');
4 char c = ad.charAt(0);
5 boolean varMi = ad.contains("a");
6 String b = ad + " ile programlama";
7 String u = ad.toUpperCase();
8 System.out.println(n + " " + k + " " + c);
9 System.out.println(b);
10 System.out.println(u + " " + ad);
```

ad

0	1	2	3
J	a	v	a

b

"Java ile programlama"

b yazılır.

Adım 9 / 10

DEĞİŞKENLER

ad = "Java"

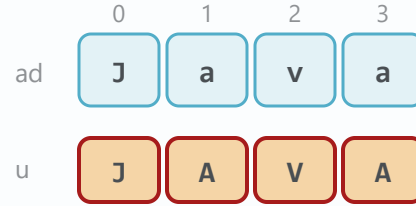
b = "Java ile programlama"

ÇIKTI

```
4 2 J
Java ile programlama
```

Dizgi Metotları: Kod İzleme

```
1 String ad = "Java";
2 int n = ad.length();
3 int k = ad.indexOf('v');
4 char c = ad.charAt(0);
5 boolean varMi = ad.contains("a");
6 String b = ad + " ile programlama";
7 String u = ad.toUpperCase();
8 System.out.println(n + " " + k + " " + c);
9 System.out.println(b);
▶ System.out.println(u + " " + ad);
```



JAVA Java : metotlar yeni dizgi döndürdü,
ad hiç değişmedi. **String** nesneleri
değiştirilemez.

Adım 10 / 10

DEĞİŞKENLER

`ad = "Java"`

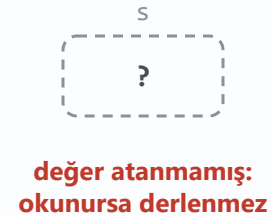
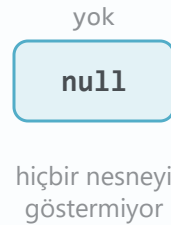
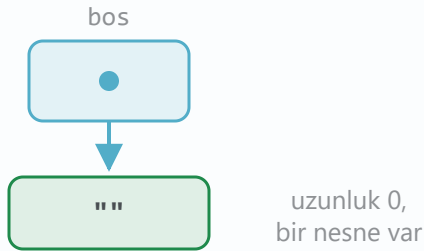
`u = "JAVA"`

ÇIKTI

```
4 2 J
Java ile programlama
JAVA Java
```

Boş Dizgi, null ve Atanmamış

Boş dizgi (`""`), değer atanmamış bir değişkenle **aynı şey değildir**.



```
String bos = "";  
String yok = null;  
System.out.println(bos.length()); // 0  
System.out.println(yok); // null  
  
String s;  
System.out.println(s); // derlenmez!
```

ÇIKTI

```
0  
null
```

```
error: variable s might not have been  
initialized
```

DİZGİLER VE TİPLER

Farklı Tipler Karşılaştırılmamalıdır

- Farklı veri tiplerini doğrudan karşılaştırmak **hataya** yol açar: bir tam sayı ile bir dizgi karşılaştırılmaz.
- Diller, tiplerin karşılaştırılması için **kurallar** ve **tip dönüşümü** (type conversion) yolları tanımlar.

✗ DOĞRUDAN KARŞILAŞTIRMA

```
int sayi = 5;  
String metin = "5";  
boolean b = sayi == metin;
```

```
error: bad operand types for binary operator '=='  
first type:  int  
second type: String
```

✓ ÖNCE AYNI TİPE DÖNÜŞTÜR

```
int sayi = 5;  
String metin = "5";  
System.out.println(Integer.parseInt(metin) == sayi);  
System.out.println(metin.equals(String.valueOf(sayi)));  
System.out.println(metin.equals(sayi));
```

```
true  
true  
false
```

`metin.equals(sayi)` derlenir ama her zaman `false` döndürür: `String` hiçbir zaman bir `Integer` 'a eşit değildir. Dizgiler `==` ile değil `equals` ile karşılaştırılır.

DİZGİLER VE TİPLER

Farklı Tiplerde İşlemler

İşlemler farklı tipler üzerinde **farklı davranır**: dizgiler üzerinde `+` toplama değil **birleştirmedir** (concatenation).

İfade	Java sonucu	Tipi	Açıklama
<code>3.0 + 4.0</code>	<code>7.0</code>	<code>double</code>	iki ondalıklı sayı
<code>3 + 4</code>	<code>7</code>	<code>int</code>	iki tam sayı
<code>3 + 4.0</code>	<code>7.0</code>	<code>double</code>	<code>int</code> otomatik olarak <code>double</code> 'a genişletilir (widening)
<code>"3" + "4"</code>	<code>"34"</code>	<code>String</code>	birleştirme
<code>3 + "4"</code>	<code>"34"</code>	<code>String</code>	bir taraf dizgiyse diğeri dizgiye çevrilir. Hata değildir (Python'da <code>TypeError</code>)
<code>3 + true</code>	derlenmez	—	<code>bad operand types for binary operator '+'</code>
<code>'a' + 1</code>	<code>98</code>	<code>int</code>	<code>char</code> bir sayısal koddur ('a' = 97)

Farklı tipler arasında işlem, dilin kuralları bilinmiyorsa **beklenmeyen sonuçlara** ya da hatalara yol açar. Java'da bu kurallar kesindir ve derleyici uygular.

Birleştirme Soldan Sağa

Bir ifadede hem sayı hem dizgi varsa sonuç, işlemlerin **sırasına** bağlıdır.

1 + 2 + "3"

+ işlemleri **soldan sağa** yapılır. İlk işlem iki `int` arasında: **toplama**.

Adım 1 / 8

Deyim	Çıktı	Neden?
<code>System.out.println(1 + 2 + "3");</code>	<code>33</code>	önce sayısal toplama, sonra birleştirme
<code>System.out.println("3" + 1 + 2);</code>	<code>312</code>	baştan itibaren dizgi: hep birleştirme
<code>System.out.println("3" + (1 + 2));</code>	<code>33</code>	parantez toplamayı öne alır
<code>System.out.println("" + 'a' + 1);</code>	<code>a1</code>	boş dizgiyle başlamak her şeyi birleştirir

Birleştirme Soldan Sağa

Bir ifadede hem sayı hem dizgi varsa sonuç, işlemlerin **sırasına** bağlıdır.

3 + "3"

Şimdi bir taraf dizgi: 3 dizgiye çevrilir ve **birleştirilir**.

Adım 2 / 8

Deyim	Çıktı	Neden?
<code>System.out.println(1 + 2 + "3");</code>	33	önce sayısal toplama, sonra birleştirme
<code>System.out.println("3" + 1 + 2);</code>	312	baştan itibaren dizgi: hep birleştirme
<code>System.out.println("3" + (1 + 2));</code>	33	parantez toplamayı öne alır
<code>System.out.println("" + 'a' + 1);</code>	a1	boş dizgiyle başlamak her şeyi birleştirir

Birleştirme Soldan Sağa

Bir ifadede hem sayı hem dizgi varsa sonuç, işlemlerin **sırasına** bağlıdır.

"33"

1 + 2 + "3" → "33"

Adım 3 / 8

Deyim	Çıktı	Neden?
<code>System.out.println(1 + 2 + "3");</code>	33	önce sayısal toplama, sonra birleştirme
<code>System.out.println("3" + 1 + 2);</code>	312	baştan itibaren dizgi: hep birleştirme
<code>System.out.println("3" + (1 + 2));</code>	33	parantez toplamayı öne alır
<code>System.out.println("" + 'a' + 1);</code>	a1	boş dizgiyle başlamak her şeyi birleştirir

Birleştirme Soldan Sağa

Bir ifadede hem sayı hem dizgi varsa sonuç, işlemlerin **sırasına** bağlıdır.

"3" + 1 + 2

Bu kez soldaki işlenen **dizgi**: 1 dizgiye çevrilir → "31" .

Adım 4 / 8

Deyim	Çıktı	Neden?
<code>System.out.println(1 + 2 + "3");</code>	33	önce sayısal toplama, sonra birleştirme
<code>System.out.println("3" + 1 + 2);</code>	312	baştan itibaren dizgi: hep birleştirme
<code>System.out.println("3" + (1 + 2));</code>	33	parantez toplamayı öne alır
<code>System.out.println("" + 'a' + 1);</code>	a1	boş dizgiyle başlamak her şeyi birleştirir

Birleştirme Soldan Sağa

Bir ifadede hem sayı hem dizgi varsa sonuç, işlemlerin **sırasına** bağlıdır.

"31" + 2

Sonuç yine dizgi; 2 de sona eklenir. Toplama hiç yapılmaz!

Adım 5 / 8

Deyim	Çıktı	Neden?
<code>System.out.println(1 + 2 + "3");</code>	33	önce sayısal toplama, sonra birleştirme
<code>System.out.println("3" + 1 + 2);</code>	312	baştan itibaren dizgi: hep birleştirme
<code>System.out.println("3" + (1 + 2));</code>	33	parantez toplamayı öne alır
<code>System.out.println("" + 'a' + 1);</code>	a1	boş dizgiyle başlamak her şeyi birleştirir

Birleştirme Soldan Sağa

Bir ifadede hem sayı hem dizgi varsa sonuç, işlemlerin **sırasına** bağlıdır.

"312"

"3" + 1 + 2 → "312"

Adım 6 / 8

Deyim	Çıktı	Neden?
<code>System.out.println(1 + 2 + "3");</code>	33	önce sayısal toplama, sonra birleştirme
<code>System.out.println("3" + 1 + 2);</code>	312	baştan itibaren dizgi: hep birleştirme
<code>System.out.println("3" + (1 + 2));</code>	33	parantez toplamayı öne alır
<code>System.out.println("" + 'a' + 1);</code>	a1	boş dizgiyle başlamak her şeyi birleştirir

Birleştirme Soldan Sağa

Bir ifadede hem sayı hem dizgi varsa sonuç, işlemlerin **sırasına** bağlıdır.

"3" + (1 + 2)

Toplama isteniyorsa **parantez** kullanılır: önce $1 + 2 = 3$.

Adım 7 / 8

Deyim	Çıktı	Neden?
<code>System.out.println(1 + 2 + "3");</code>	33	önce sayısal toplama, sonra birleştirme
<code>System.out.println("3" + 1 + 2);</code>	312	baştan itibaren dizgi: hep birleştirme
<code>System.out.println("3" + (1 + 2));</code>	33	parantez toplamayı öne alır
<code>System.out.println("" + 'a' + 1);</code>	a1	boş dizgiyle başlamak her şeyi birleştirir

Birleştirme Soldan Sağa

Bir ifadede hem sayı hem dizgi varsa sonuç, işlemlerin **sırasına** bağlıdır.

"33"

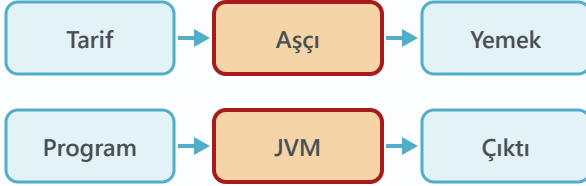
"3" + (1 + 2) → "33" . Aynı tuzak: "Toplam: " + 2 + 3 → Toplam: 23 .

Adım 8 / 8

Deyim	Çıktı	Neden?
System.out.println(1 + 2 + "3");	33	önce sayısal toplama, sonra birleştirme
System.out.println("3" + 1 + 2);	312	baştan itibaren dizgi: hep birleştirme
System.out.println("3" + (1 + 2));	33	parantez toplamayı öne alır
System.out.println("" + 'a' + 1);	a1	boş dizgiyle başlamak her şeyi birleştirir

Program Bir Tariftir

- **Programlama** bir problemi çözme sürecidir; **program** bu sürecin sonucudur.
- Program, bilgisayara bir görevi **nasıl** yerine getireceğini anlatan bir **tariftir** (recipe).
- Programlar bilgisayarın **anlayabileceği** biçimde yazılır ve **adım adım** talimatlar içerir.
- Bilgisayar bu talimatlara **harfiyen** uyar: eksik ya da belirsiz bir adımı tahmin etmez.



talimatlar harfiyen, sırayla uygulanır

Yemek tarifi	Program
malzemeler (un, şeker)	veriler, değişkenler
ölçüler (2 su bardağı)	değerler ve tipler
adımlar (karıştır, pişir)	deyimler
"kabarana kadar çırp"	döngü
"tuzsuzsa tuz ekle"	koşul (if)
aşçı	bilgisayar (JVM)
ortaya çıkan yemek	çıktı

Kodlamadan Önce Algoritmayı Tasarla

- Program yazılmadan önce problemin **nasıl çözüleceği** düşünülür ve bir **algoritma** tasarlanır.
- **Algoritmik düşünme**: problemi çözme ve işlemleri doğru **sıralama** yeteneği. İyi bir algoritma programın verimli çalışmasını sağlar.
- **Kodlama**, algoritmayı bilgisayarın anlayacağı söz dizimine çevirir ve onu gerçek bir programa dönüştürür.

ALGORİTMA: İKİ SAYININ ORTALAMASI

1. Birinci sayıyı iste ve oku.
2. İkinci sayıyı iste ve oku.
3. $\text{ortalama} \leftarrow (\text{sayi1} + \text{sayi2}) / 2$
4. Ortalamayı yazdır.

```
import java.util.Scanner;

public class Ortalama {
    public static void main(String[] args) {
        Scanner giris = new Scanner(System.in);
        System.out.print("Birinci sayı: ");    // 1
        double sayi1 = giris.nextDouble();
        System.out.print("İkinci sayı: ");    // 2
        double sayi2 = giris.nextDouble();
        double ortalama = (sayi1 + sayi2) / 2; // 3
        System.out.println("Ortalama: " + ortalama); // 4
        giris.close();
    }
}
```

Her algoritma adımı bir ya da iki Java satırına karşılık gelir.

Ortalama: Kodu Adım Adım İzle

```
1 import java.util.Scanner;
2
3 public class Ortalama {
4     public static void main(String[] args) {
5         ▶ Scanner giris = new Scanner(System.in);
6         System.out.print("Birinci sayı: ");
7         double sayi1 = giris.nextDouble();
8         System.out.print("İkinci sayı: ");
9         double sayi2 = giris.nextDouble();
10        double ortalama = (sayi1 + sayi2) / 2;
11        System.out.println("Ortalama: " + ortalama);
12        giris.close();
13    }
14 }
```



Klavyeden okumak için bir **Scanner** nesnesi oluşturulur.

Adım 1 / 8

DEĞİŞKENLER

sayi1 = -

sayi2 = -

ortalama = -

ÇIKTI



Ortalama: Kodu Adım Adım İzle

```
1 import java.util.Scanner;
2
3 public class Ortalama {
4     public static void main(String[] args) {
5         Scanner giris = new Scanner(System.in);
6         ▶ System.out.print("Birinci sayı: ");
7         double sayi1 = giris.nextDouble();
8         System.out.print("İkinci sayı: ");
9         double sayi2 = giris.nextDouble();
10        double ortalama = (sayi1 + sayi2) / 2;
11        System.out.println("Ortalama: " + ortalama);
12        giris.close();
13    }
14 }
```



print alt satıra geçmez; imleç istemin yanında bekler.

Adım 2 / 8

DEĞİŞKENLER

sayi1 = -

sayi2 = -

ortalama = -

ÇIKTI

Birinci sayı:

Ortalama: Kodu Adım Adım İzle

```
1 import java.util.Scanner;
2
3 public class Ortalama {
4     public static void main(String[] args) {
5         Scanner giris = new Scanner(System.in);
6         System.out.print("Birinci sayı: ");
7         ▶ double sayi1 = giris.nextDouble();
8         System.out.print("İkinci sayı: ");
9         double sayi2 = giris.nextDouble();
10        double ortalama = (sayi1 + sayi2) / 2;
11        System.out.println("Ortalama: " + ortalama);
12        giris.close();
13    }
14 }
```

sayi1

7.0

sayi2

?

ortalama

?

Kullanıcı **7** yazıp Enter'a basar.

`nextDouble()` bunu 7.0 olarak okur.

Adım 3 / 8

DEĞİŞKENLER

`sayi1 = 7.0`

`sayi2 = -`

`ortalama = -`

ÇIKTI

Birinci sayı: 7

Ortalama: Kodu Adım Adım İzle

```
1 import java.util.Scanner;
2
3 public class Ortalama {
4     public static void main(String[] args) {
5         Scanner giris = new Scanner(System.in);
6         System.out.print("Birinci sayı: ");
7         double sayi1 = giris.nextDouble();
8         ▶ System.out.print("İkinci sayı: ");
9         double sayi2 = giris.nextDouble();
10        double ortalama = (sayi1 + sayi2) / 2;
11        System.out.println("Ortalama: " + ortalama);
12        giris.close();
13    }
14 }
```

sayi1

7.0

sayi2

?

ortalama

?

İkinci istem yazdırılır.

Adım 4 / 8

DEĞİŞKENLER

sayi1 = 7.0

sayi2 = -

ortalama = -

ÇIKTI

Birinci sayı: 7

İkinci sayı:

Ortalama: Kodu Adım Adım İzle

```
1 import java.util.Scanner;
2
3 public class Ortalama {
4     public static void main(String[] args) {
5         Scanner giris = new Scanner(System.in);
6         System.out.print("Birinci sayı: ");
7         double sayi1 = giris.nextDouble();
8         System.out.print("İkinci sayı: ");
9         ▶ double sayi2 = giris.nextDouble();
10        double ortalama = (sayi1 + sayi2) / 2;
11        System.out.println("Ortalama: " + ortalama);
12        giris.close();
13    }
14 }
```

sayi1

7.0

sayi2

8.0

ortalama

?

Kullanıcı **8** girer → **sayi2** = 8.0.

Adım 5 / 8

DEĞİŞKENLER

sayi1 = 7.0

sayi2 = 8.0

ortalama = -

ÇIKTI

Birinci sayı: 7

İkinci sayı: 8

Ortalama: Kodu Adım Adım İzle

```
1 import java.util.Scanner;
2
3 public class Ortalama {
4     public static void main(String[] args) {
5         Scanner giris = new Scanner(System.in);
6         System.out.print("Birinci sayı: ");
7         double sayi1 = giris.nextDouble();
8         System.out.print("İkinci sayı: ");
9         double sayi2 = giris.nextDouble();
10        ► double ortalama = (sayi1 + sayi2) / 2;
11        System.out.println("Ortalama: " + ortalama);
12        giris.close();
13    }
14 }
```

sayi1

7.0

sayi2

8.0

ortalama

7.5

$(7.0 + 8.0) / 2 = 15.0 / 2 = 7.5$: işlenenler

`double` olduğu için kesir korunur.

Adım 6 / 8

DEĞİŞKENLER

sayi1 = 7.0

sayi2 = 8.0

ortalama = 7.5

ÇIKTI

Birinci sayı: 7

İkinci sayı: 8

Ortalama: Kodu Adım Adım İzle

```
1 import java.util.Scanner;
2
3 public class Ortalama {
4     public static void main(String[] args) {
5         Scanner giris = new Scanner(System.in);
6         System.out.print("Birinci sayı: ");
7         double sayi1 = giris.nextDouble();
8         System.out.print("İkinci sayı: ");
9         double sayi2 = giris.nextDouble();
10        double ortalama = (sayi1 + sayi2) / 2;
11        ▶ System.out.println("Ortalama: " + ortalama);
12        giris.close();
13    }
14 }
```

sayi1

7.0

sayi2

8.0

ortalama

7.5

Sonuç yazdırılır.

Adım 7 / 8

DEĞİŞKENLER

sayi1 = 7.0

sayi2 = 8.0

ortalama = 7.5

ÇIKTI

Birinci sayı: 7

İkinci sayı: 8

Ortalama: 7.5

Ortalama: Kodu Adım Adım İzle

```
1 import java.util.Scanner;
2
3 public class Ortalama {
4     public static void main(String[] args) {
5         Scanner giris = new Scanner(System.in);
6         System.out.print("Birinci sayı: ");
7         double sayi1 = giris.nextDouble();
8         System.out.print("İkinci sayı: ");
9         double sayi2 = giris.nextDouble();
10        double ortalama = (sayi1 + sayi2) / 2;
11        System.out.println("Ortalama: " + ortalama);
12        ▶ giris.close();
13    }
14 }
```

sayi1

7.0

sayi2

8.0

ortalama

7.5

Scanner kapatılır; program biter.
Deyimler yazıldıkları **sırayla** çalıştı.

Adım 8 / 8

DEĞİŞKENLER

sayi1 = 7.0

sayi2 = 8.0

ortalama = 7.5

ÇIKTI

```
Birinci sayı: 7
İkinci sayı: 8
Ortalama: 7.5
```

Bir Program Nedir? Sıra Önemlidir

Program bir **talimatlar dizisidir**; bilgisayar talimatları **birer birer** ve **sırayla** işler. Bu yüzden işlem sırasını doğru belirlemek önemlidir.

```
► int a = 3, b = 5;  
2 a = b;  
3 b = a;  
4 System.out.println(a + " " + b);  
5  
6 int c = 3, d = 5;  
7 int gecici = c;  
8 c = d;  
9 d = gecici;  
10 System.out.println(c + " " + d);
```



Amaç: **a** ile **b** 'nin değerlerini **takas** etmek (yer değiştirmek).

Adım 1 / 9

DEĞİŞKENLER

a = 3 b = 5

ÇIKTI



Bir Program Nedir? Sıra Önemlidir

Program bir **talimatlar dizisidir**; bilgisayar talimatları **birer birer** ve **sırayla** işler. Bu yüzden işlem sırasını doğru belirlemek önemlidir.

```
1 int a = 3, b = 5;  
2 a = b;  
3 b = a;  
4 System.out.println(a + " " + b);  
5  
6 int c = 3, d = 5;  
7 int gecici = c;  
8 c = d;  
9 d = gecici;  
10 System.out.println(c + " " + d);
```



a = b : a'ya 5 yazılır. a'nın eski değeri 3 **kayboldu!**

Adım 2 / 9

DEĞİŞKENLER

a = 5

b = 5

ÇIKTI



Bir Program Nedir? Sıra Önemlidir

Program bir **talimatlar dizisidir**; bilgisayar talimatları **birer birer** ve **sırayla** işler. Bu yüzden işlem sırasını doğru belirlemek önemlidir.

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



b = a : b'ye a'nın **şimdiki** değeri (5) yazılır.

Adım 3 / 9

DEĞİŞKENLER

a = 5

b = 5

ÇIKTI



Bir Program Nedir? Sıra Önemlidir

Program bir **talimatlar dizisidir**; bilgisayar talimatları **birer birer** ve **sırayla** işler. Bu yüzden işlem sırasını doğru belirlemek önemlidir.

```
1 int a = 3, b = 5;
2 a = b;
3 b = a;
▶ System.out.println(a + " " + b);
5
6 int c = 3, d = 5;
7 int gecici = c;
8 c = d;
9 d = gecici;
10 System.out.println(c + " " + d);
```



Çıktı **5 5** : takas başarısız. Deyimlerin **sırası** sonucu belirler.

Adım 4 / 9

DEĞİŞKENLER

a = 5

b = 5

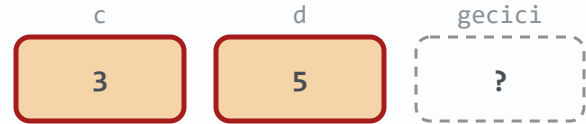
ÇIKTI

5 5

Bir Program Nedir? Sıra Önemlidir

Program bir **talimatlar dizisidir**; bilgisayar talimatları **birer birer** ve **sırayla** işler. Bu yüzden işlem sırasını doğru belirlemek önemlidir.

```
1 int a = 3, b = 5;
2 a = b;
3 b = a;
4 System.out.println(a + " " + b);
5
▶ int c = 3, d = 5;
7 int gecici = c;
8 c = d;
9 d = gecici;
10 System.out.println(c + " " + d);
```



Doğru yöntem: üçüncü bir **geçici** değişken.

Adım 5 / 9

DEĞİŞKENLER

c = 3

d = 5

gecici = -

ÇIKTI

5 5

Bir Program Nedir? Sıra Önemlidir

Program bir **talimatlar dizisidir**; bilgisayar talimatları **birer birer** ve **sırayla** işler. Bu yüzden işlem sırasını doğru belirlemek önemlidir.

```
1 int a = 3, b = 5;
2 a = b;
3 b = a;
4 System.out.println(a + " " + b);
5
6 int c = 3, d = 5;
7 int gecici = c;
8 c = d;
9 d = gecici;
10 System.out.println(c + " " + d);
```



c'nin değeri önce **gecici** 'de **saklanır**.

Adım 6 / 9

DEĞİŞKENLER

c = 3

d = 5

gecici = 3

ÇIKTI

5 5

Bir Program Nedir? Sıra Önemlidir

Program bir **talimatlar dizisidir**; bilgisayar talimatları **birer birer** ve **sırayla** işler. Bu yüzden işlem sırasını doğru belirlemek önemlidir.

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



Artık c'nin üzerine yazılabilir: c = 5.

Adım 7 / 9

DEĞİŞKENLER

c = 5

d = 5

gecici = 3

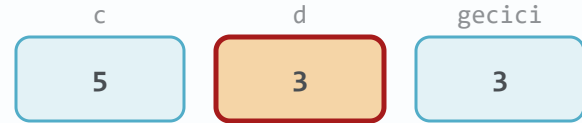
ÇIKTI

5 5

Bir Program Nedir? Sıra Önemlidir

Program bir **talimatlar dizisidir**; bilgisayar talimatları **birer birer** ve **sırayla** işler. Bu yüzden işlem sırasını doğru belirlemek önemlidir.

```
1 int a = 3, b = 5;
2 a = b;
3 b = a;
4 System.out.println(a + " " + b);
5
6 int c = 3, d = 5;
7 int gecici = c;
8 c = d;
9 d = gecici;
10 System.out.println(c + " " + d);
```



d'ye saklanan eski c değeri (3) yazılır.

Adım 8 / 9

DEĞİŞKENLER

c = 5

d = 3

gecici = 3

ÇIKTI

5 5

Bir Program Nedir? Sıra Önemlidir

Program bir **talimatlar dizisidir**; bilgisayar talimatları **birer birer** ve **sırayla** işler. Bu yüzden işlem sırasını doğru belirlemek önemlidir.

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



Çıktı **5 3** : takas başarılı.

Adım 9 / 9

DEĞİŞKENLER

c = 5

d = 3

gecici = 3

ÇIKTI

5 5

5 3

İfadeler, Deyimler ve Programlar

İfade (expression) → değer

Bir değeri hesaplar ve sonucunu döndürür: `3 + 4` ifadesinin değeri 7'dir.

Deyim (statement) → etki

Bir etki yaratır: `pi = 3.14159;` değişkene değer atar, `System.out.println(pi);` ekrana yazar. Java'da deyimler `;` ile biter.

İfadeler iç içe geçebilir

Bir ifade, başka ifadelerin ve deyimlerin içinde yer alabilir: `double alan = pi * r * r;`

Program = deyimler dizisi

Bir program, sırayla çalışan birçok deyimden oluşur.

Bir deyim, bir ifadenin içinde yer alamaz. `println` ekrana yazar (etki) ama bir değer döndürmez (`void`):

```
double d = 3 + System.out.println(pi);
```

error: 'void' type not allowed here

```
3 + 4;           // değer üretir ama etkisi yok
```

error: not a statement

Java'da atama da bir değer üretir: `int y = (x = 4) + 1;` → `x = 4`, `y = 5`. Derlenir ama okunması zordur; kullanmayın.

Popüler Programlama Dilleri

- Programlama dilleri farklı görevler için **farklı avantajlar** sunar.
- Programcılar, projenin ihtiyaçlarına göre **en uygun dili** seçer.
- Dillerin popülerliği her yıl anketler ve dizinlerle (Stack Overflow anketi, TIOBE, IEEE Spectrum) değerlendirilir.
- 2023'te öne çıkan dillerden bazıları yandaki tabloda.

Dil	Öne çıktığı alan
Python	veri analizi, yapay zekâ, web geliştirme
Java	büyük ölçekli kurumsal uygulamalar, Android
JavaScript	web tarayıcıları, web geliştirme
C#	Windows uygulamaları, oyun geliştirme
C++	oyun, sistem programlama, performans odaklı yazılım
Rust	güvenli ve hızlı sistem programlama
Swift	Apple platformlarında uygulama geliştirme
Kotlin	Android uygulama geliştirme (JVM üzerinde çalışır)
Go (Golang)	basit ve etkili; ağ ve bulut uygulamaları

ÖZET

Özet

Konu	Hatırlanacaklar
Dil sınıfları	düşük / yüksek seviye; genel amaçlı / hedefe odaklı; derlenen / yorumlanan
Java	<code>javac</code> → bytecode (<code>.class</code>) → JVM (yorum + JIT); "bir kez yaz, her yerde çalıştır"
Paradigmalar	buyruksal, fonksiyonel, mantıksal, nesne yönelimli; Java: nesne yönelimli + buyruksal
Söz dizimi / anlam	söz dizimi: biçim; statik anlam: derlemede tip denetimi; anlam: tek ve kesin, ama beklenen olmayabilir
Hata türleri	söz dizimi ve statik anlam (derleme), çalışma zamanı (istisna), mantık (yanlış sonuç)
İfadeler	parantez → <code>*</code> / <code>%</code> → <code>+</code> - , soldan sağa; <code>int</code> / <code>int</code> tam sayı bölmesi; üs için <code>Math.pow</code>
Değişken ve atama	tip + ad; önce sağ taraf hesaplanır, sonra yazılır; <code>=</code> atama, <code>==</code> karşılaştırma
Tipler	Java statik tipli; <code>int</code> + <code>double</code> → <code>double</code> ; dizgi + herhangi → birleştirme
Program	önce algoritma, sonra kod; ifade değer üretir, deyim etki yaratır; deyimler sırayla çalışır

ÖZET

Sık Yapılan Hatalar

Noktalı virgülü unutmak

`System.out.println("Merhaba")` → `' ; '`
expected . Hata, bir **sonraki** satırda gösterilebilir.

= ile == karıştırmak

`if (x = 100)` → `int cannot be converted to boolean` . Karşılaştırma `==` 'dir.

Tam sayı bölmesi

`(a + b) / 2` ile 7 ve 8'in ortalaması 7 çıkar. Gerçek bölme için `/ 2.0` yazın.

Üs için ** ya da ^

`2 ** 3` → `illegal start of expression`; `2 ^ 3` derlenir ama XOR'dur (1). Doğrusu `Math.pow(2, 3)` .

Dizgi + sayı sırası

`"Toplam: " + 2 + 3` → `Toplam: 23` . Toplama için `"Toplam: " + (2 + 3)` .

Büyük/küçük harf

`Sayi tanımlayıp sayi kullanmak` → `cannot find symbol` . Java harf duyarlıdır.

Değer atamadan kullanmak

`int s; System.out.println(s);` → `variable s might not have been initialized` .

Ondalık ayırıcı

Türkçe bölge ayarlı bilgisayarda `nextDouble()` girdisi `7,5` biçimindedir; `7.5` yazmak `InputMismatchException` verir.

ÖZET

Örnek Problemler

1. Değerlerini bulun: (a) `10 + 6 / 4 * 2` (b) `(10 + 6) / 4 * 2` (c) `10 + 6 / 4.0 * 2`
2. Çıktıları nedir? `System.out.println("Toplam: " + 2 + 3);` ve `System.out.println(2 + 3 + " toplam");`
3. Kod çalıştıktan sonra `a` ve `b` kaçtır? `int a = 4; int b = a * 3; a = b - a; b = b / a;`
4. Hata türünü söyleyin: (a) `int n = 4.5;` (b) `System.out.println("Merhaba");` (c) dairenin alanı için `alan = 2 * pi * r;`
5. `double c = (f - 32) * 5 / 9;` ile `f = 100` için sonuç nedir? `f` ve `c` `int` olsaydı?

► Cevapları göster

ÖZET

Alıştırmalar

1. Değerlerini ve tiplerini yazın: `7 % 3` , `-7 / 2` , `1 / 3 * 3` , `1.0 / 3 * 3` , `Math.pow(2, 10)` .
2. Hangileri geçerli değişken adıdır? `toplam2` , `2toplam` , `_sayac` , `ogrenci sayisi` , `$fiyat` , `int` , `ortalamaNot`
3. Kullanıcıdan Fahrenheit değeri okuyup Celsius'a çeviren programın önce algoritmasını, sonra Java kodunu yazın.
4. Şunları ifade ya da deyim olarak sınıflandırın: `x * 2` , `x = 5;` , `x > 3` , `System.out.println(x);`
5. `5 / 9 * (f - 32)` ifadesi `int f = 100` için neden 0 verir? İki farklı biçimde düzeltin.

► Cevaplar / ipuçları

Kaynaklar

- Oracle, *The Java Tutorials: "Language Basics"* (değişkenler, operatörler, ifadeler, deyimler ve bloklar).
- J. Gosling ve diğerleri, *The Java Language Specification, Java SE 21 Edition*, Oracle, 2023 (Bölüm 3: Sözcük yapısı, Bölüm 15: İfadeler).
- P. Deitel ve H. Deitel, *Java: How to Program, Early Objects*, 11. baskı, Pearson, 2017 (Bölüm 1–2).
- A. B. Downey ve C. Mayfield, *Think Java: How to Think Like a Computer Scientist*, 2. baskı, O'Reilly, 2019 (Bölüm 1–2).
- K. Sierra ve B. Bates, *Head First Java*, 3. baskı, O'Reilly, 2022.
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, Pearson, 2022 (Bölüm 2–3).
- J. Guttag, *Introduction to Computation and Programming Using Python*, 3. baskı, MIT Press, 2021 (Bölüm 2: söz dizimi, statik anlam ve anlam).

Son

Bölüm 2: Programlama

Sonraki bölüm: Bölüm 2: Söz Dizimi Kuralları · [Terim sözlüğü](#)