Bölüm 2.1: Programlama

Java ile Nesne Yönelimli Programlama

Programlama dilleri | Derleme ve yorumlama | Paradigmalar | Söz dizimi ve anlam | Hata türleri | İfadeler ve öncelik | Değişkenler ve atama | Statik ve dinamik tipleme | 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

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

Değişkenler ve tipler

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

Algoritmadan programa

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

GİRİŞ

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.

Programlama Dili Nedir?

- **Programlama dili** (programming language): bir makinenin yapabileceği 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.
- Soyutlama düzeyine göre: makine dili ve assembly **düşük düzeyli**; Java, Python **yüksek düzeyli**.
- Dil seçimi **projenin ihtiyaçlarına** göre yapılır.
- Örnekler: C, C++, Java, Python, Prolog, Haskell, Scala…



PROGRAMLAMA DİLLERİ

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

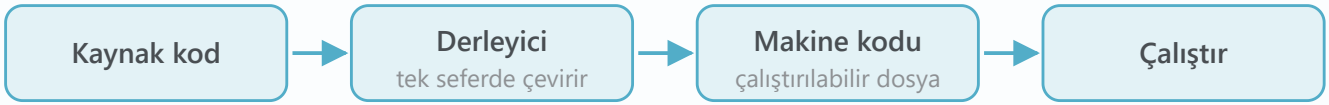
```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
```

PROGRAMLAMA DİLLERİ

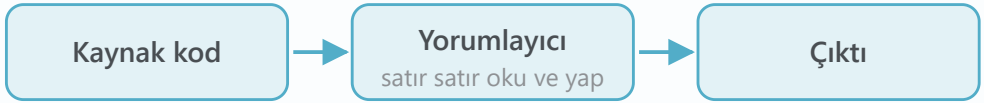
Derlenen ve Yorumlanan Diller

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

DERLENEN DİL (C, C++, GO, RUST)



YORUMLANAN DİL (PYTHON, JAVASCRİPT)



	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 ikisini birleştirir: `javac` → **bytecode** (`.class`) → JVM (yorum + JIT).

PROGRAMLAMA DİLLERİ

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)



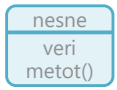
Hesaplamayı matematiksel **fonksiyonların değerlendirilmesi** olarak ele alır; durum değiştirmekten kaçınır. Lisp, Scheme, Haskell...

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.

```java
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);
    }
}
```

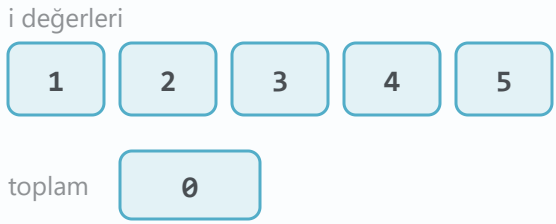
ÇIKTI

```
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);
```



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;
▶ for (int i = 1; i <= 5; i++) {
▶     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

1	2	3	4	5

toplam **1** `0 + 1 = 1`

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

Adım 2 / 10

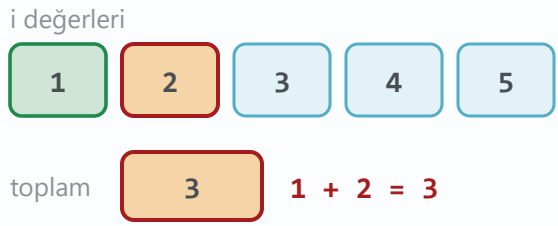
DEĞİŞKENLER

`toplam = 1` `i = 1`

ÇIKTI

Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
▶ for (int i = 1; i <= 5; i++) {
▶     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 = 2` : 2 <= 5 doğru → toplam = 1 + 2 = **3**.

Adım 3 / 10

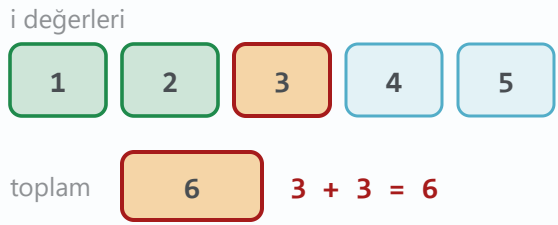
DEĞİŞKENLER

`toplam = 3` `i = 2`

ÇIKTI

Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
▶ for (int i = 1; i <= 5; i++) {
▶     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 = 3` : 3 <= 5 doğru → toplam = 3 + 3 = **6**.

Adım 4 / 10

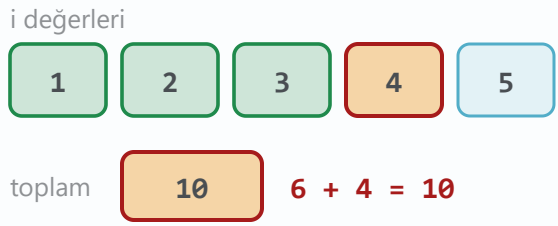
DEĞİŞKENLER

`toplam = 6` `i = 3`

ÇIKTI

Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
▶ for (int i = 1; i <= 5; i++) {
▶     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 = 4` : 4 <= 5 doğru → toplam = 6 + 4 = **10**.

Adım 5 / 10

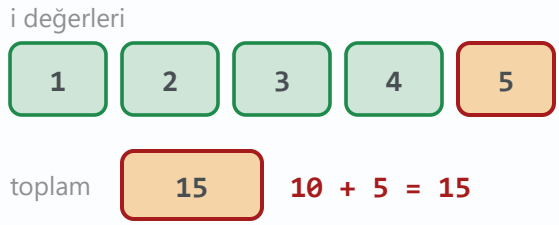
DEĞİŞKENLER

`toplam = 10` `i = 4`

ÇIKTI

Buyruksal ve Fonksiyonel: Kod İzleme

```
1 int toplam = 0;
▶ for (int i = 1; i <= 5; i++) {
▶     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 = 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;
▶ 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

1	2	3	4	5

toplam **15**

`i` 6 oldu: 6 <= 5 **yanlış** → döngü biter.

Adım 7 / 10

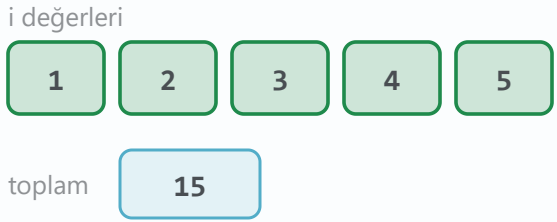
DEĞİŞKENLER

`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 }
▶ System.out.println("Buyruksal: " + toplam);
6
7 int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```



Sonuç yazdırılır: beş turda biriken toplam 15.

Adım 8 / 10

DEĞİŞKENLER

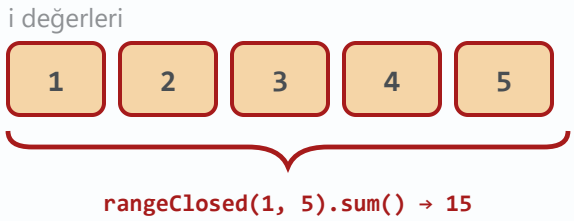
`toplam = 15`

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
▶ int sonuc = IntStream.rangeClosed(1, 5).sum();
8 System.out.println("Fonksiyonel: " + sonuc);
```



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

Adım 9 / 10

DEĞİŞKENLER

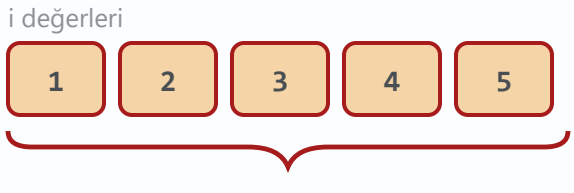
`toplam = 15` `sonuc = 15`

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);
```



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
```

PROGRAMLAMA DİLLERİ

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) {
▶          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

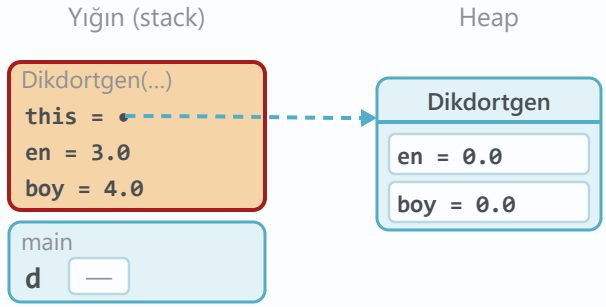
DEĞİŞKENLER

d = –

ÇIKTI

Nesne Yönelimli: Kod İzleme

```
1  public class Dikdortgen {
2      double en, boy;
▶      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 }
```



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

Adım 2 / 7

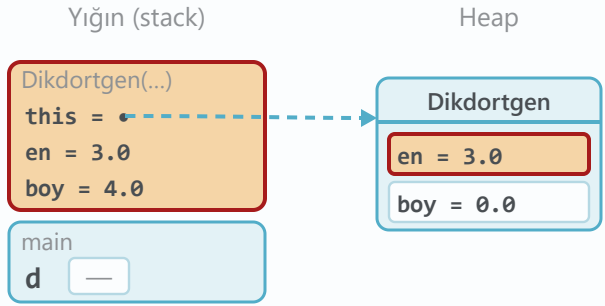
DEĞİŞKENLER

`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) {
▶          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 }
```



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

Adım 3 / 7

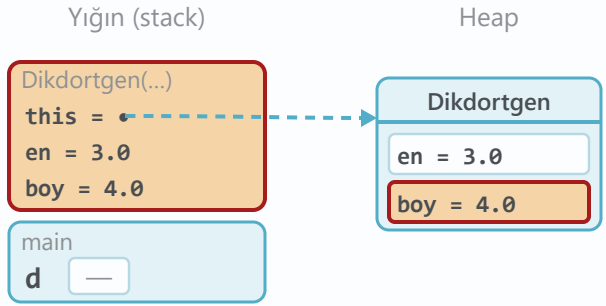
DEĞİŞKENLER

`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 }
```



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

Adım 4 / 7

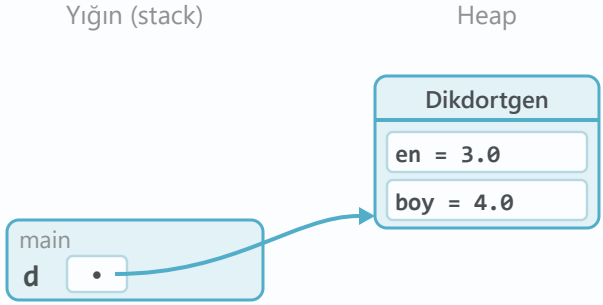
DEĞİŞKENLER

`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) {
▶          Dikdortgen d = new Dikdortgen(3, 4);
12         System.out.println("Alan: " + d.alan());
13     }
14 }
```



`new` ifadesinin değeri nesnenin **referansı**dır; `d` artık bu nesneyi gösterir.

Adım 5 / 7

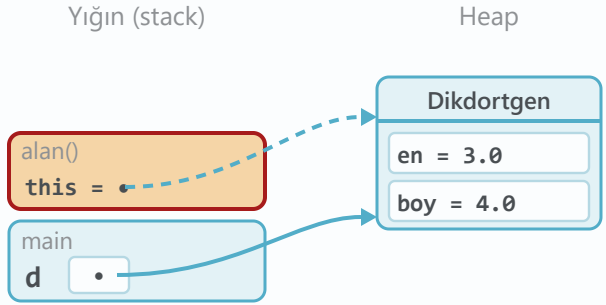
DEĞİŞKENLER

`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() {
▶          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 }
```



`d.alan()` : `this` ile `d` aynı nesneyi gösterir. `en * boy` = 3.0 × 4.0 = **12.0** döndürülür.

Adım 6 / 7

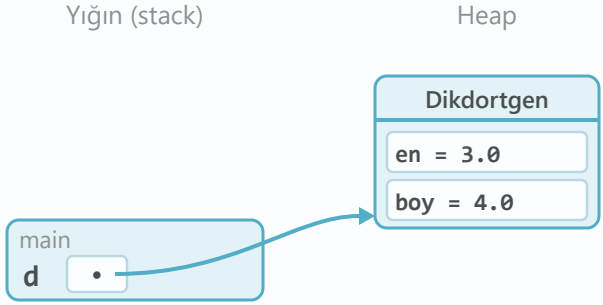
DEĞİŞKENLER

`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);
▶          System.out.println("Alan: " + d.alan());
13     }
14 }
```



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
```

SÖZ DİZİMİ VE ANLAM

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 işleçler (`+ - * /`), anahtar sözcükler
Söz dizimi (syntax)	"kedi köpek çocuk": kelimeler doğru ama cümle kuruluşu yok → **geçersiz**	hangi karakter ve sembol dizilerinin **düzgün** olduğunu belirler

Söz dizimi dilin **kurallarını** belirler ve dilin anlaşılırlığını 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

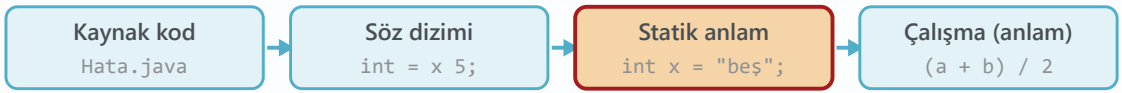
- **Statik anlam** (static semantics): söz dizimi doğru olan bir ifadenin **anlamlı** olup olmadığını, program çalışmadan **önce** denetler.
- İngilizce: "I are big" biçim olarak `<isim> <fiil> <isim>` kalıbına uyar; ama "I" tekil, "are" çoğuldur → dil açısından **geçersiz**.
- Java: `<değer> <işleç>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

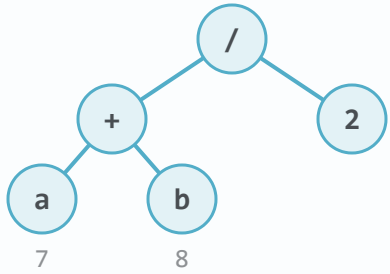
ÇIKTI

```
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

```
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
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

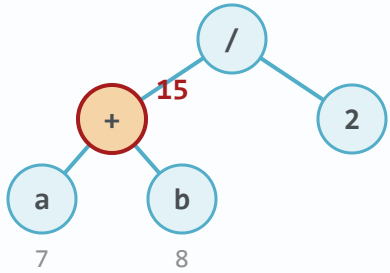
DEĞİŞKENLER

`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

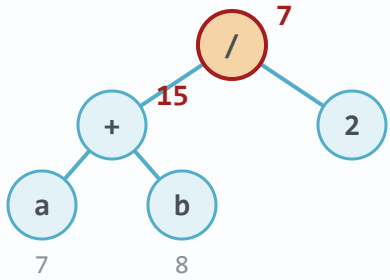
DEĞİŞKENLER

`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

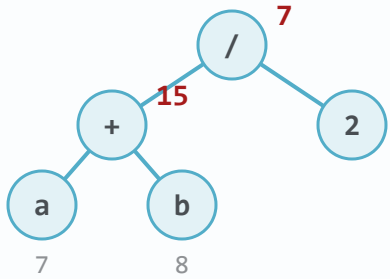
DEĞİŞKENLER

`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

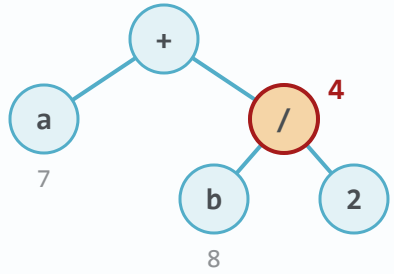
`a = 7` `b = 8` `ortalama = 7`

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 `+` işlecinden yüksek. Önce `b / 2` = 4.

Adım 5 / 7

DEĞİŞKENLER

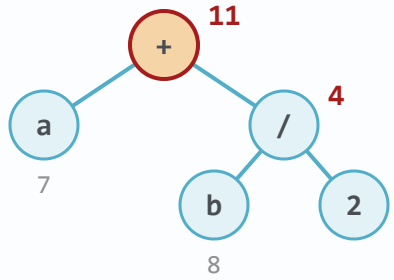
`a = 7` `b = 8` `ortalama = 7`

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

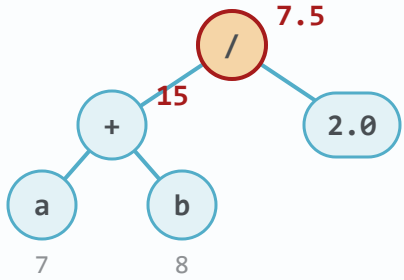
`a = 7` `b = 8` `ortalama = 7`

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)	`System.out.println("Merhaba")`	`';' expected`
Statik anlam	derlemede (tip denetimi)	`int x = "beş";`	`incompatible types`
Çalışma zamanı	program çalışırken	`a / b` (b sıfırken)	çökme: `ArithmeticException`
Mantık	derleyici fark etmez; ancak testle	`(a + b) / 2` 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 oluştuğu satır.
- İstisna `try` - `catch` ile yakalanmazsa program bu satırda sonlanır.

Birden çok hata listelendiğinde düzeltmeye **ilkinden** başlanır; sonrakiler çoğu zaman ilk hatanın yan etkisidir.

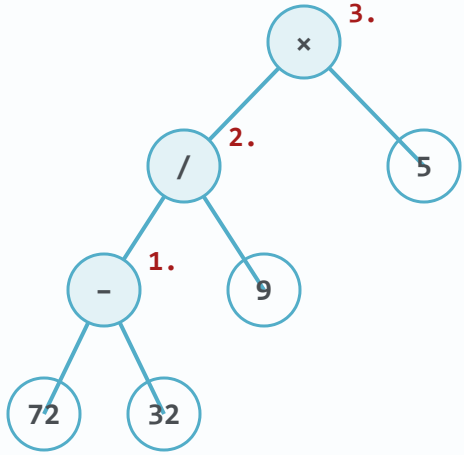
İfade Değerlendirme Adım Adım

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.

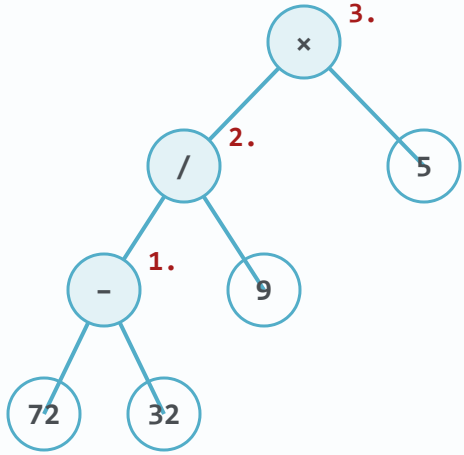
İfade Değerlendirme Adım Adım

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.

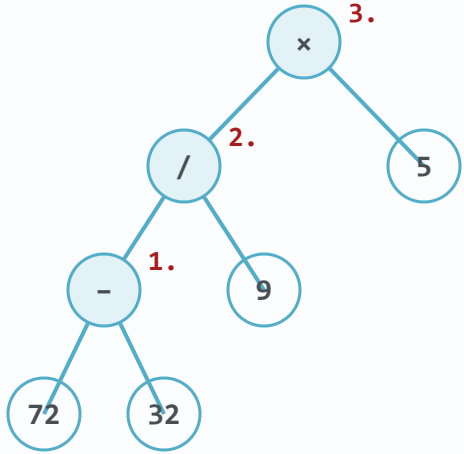
İfade Değerlendirme Adım Adım

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.

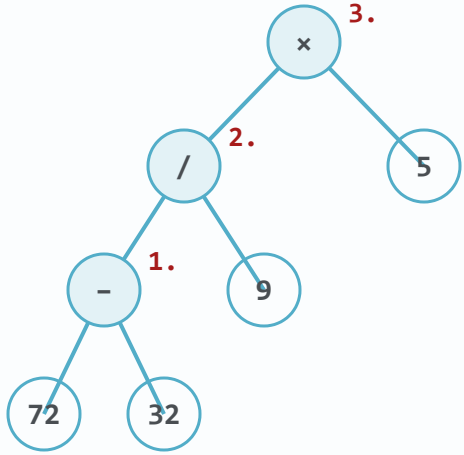
İfade Değerlendirme Adım Adım

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.

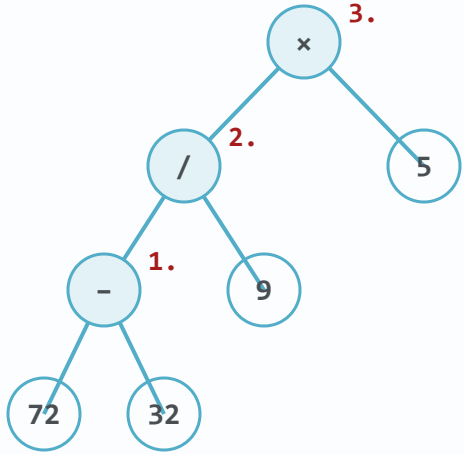
İfade Değerlendirme Adım Adım

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



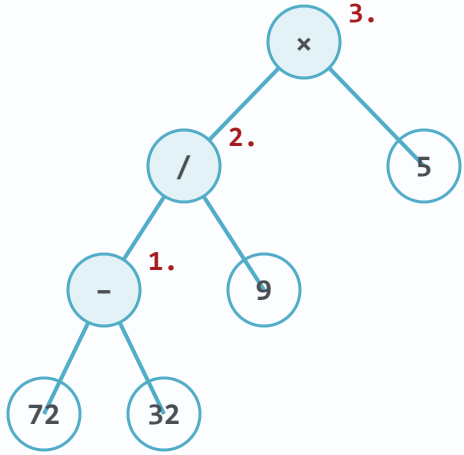
İfade ağacı: alttaki işlem önce yapılır; numaralar işlem sırasıdır.

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: 4.444444444444445.



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

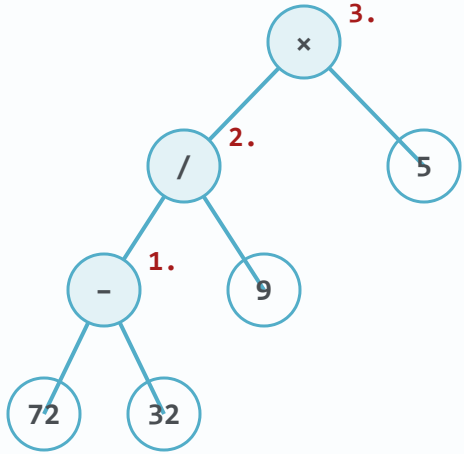
İfade Değerlendirme Adım Adım

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

```
4.444444444444445 * 5
```

Çarpma: `double` × `int` → `double`.

Adım 7 / 8



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

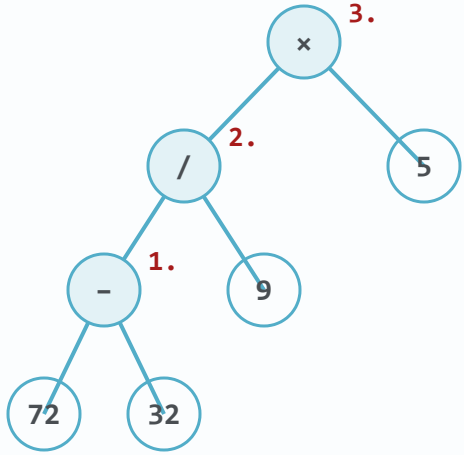
İfade Değerlendirme Adım Adım

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.

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.

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.

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
```

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.

Farklı Tipler Karşılaştırılmamalıdır

- Farklı veri tiplerini doğrudan karşılaştırmak **hataya** yol açar: bir tam sayı ile bir dizgi karşılaştırılamaz.
- 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.

Farklı Tiplerde İşlemler

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

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

PROGRAM VE ALGORİTMA · ADIM ADIM KOD İZLEME

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;
▶  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;
▶  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;
▶ 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;
▶  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;
▶  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
```

PROGRAM VE ALGORİTMA

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 zor olduğundan tercih edilmez.

POPÜLER DİLLER

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

Konu	Hatırlanacaklar
Dil sınıfları	düşük / yüksek düzey; genel amaçlı / hedefe odaklı; derlenen / yorumlanan
Java	`javac` → bytecode (`.class`) → 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 → `* / %` → `+ -`, soldan sağa; `int / int` tam sayı bölmesi; üs için `Math.pow`
Değişken ve atama	tip + ad; önce sağ taraf hesaplanır, sonra yazılır; `=` atama, `==` karşılaştırma
Tipler	Java statik tipli; `int` + `double` → `double` ; 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`. javac, `^` ile ; eksik satırın **sonunu** gösterir.

= 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` gerekir.

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 Fahrenhayt 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, işleçler, 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 2.1: Programlama

Sonraki bölüm: Bölüm 2.2: Söz Dizimi Kuralları · Terim sözlüğü