Bölüm 9: Metotlar

Bilgisayar ve Programlamaya Giriş

Tanım | Parametre | Dönüş | Kapsam | Örnekler

Sercan KÜLCÜ · Giresun Üniversitesi

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/computer/slides/Bolum_09_Metotlar.html

GİRİŞ

Öğrenme Çıktıları

Tanımlama

Metot tanımlaması ve çağrısını yazmak.

Veri akışı

Parametre ve dönüş değerini kullanmak.

Kapsam

Yerel değişkenlerin yaşam alanını açıklamak.

Tasarım

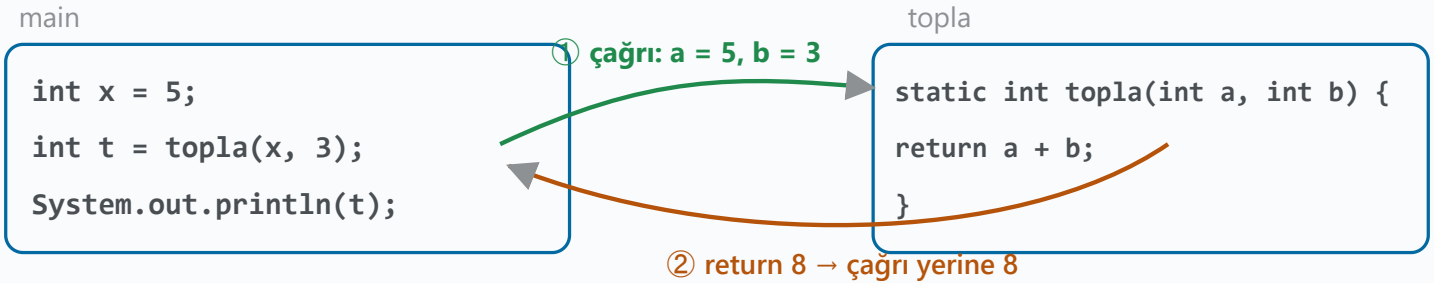
Kısa, odaklı ve test edilebilir metotlar kurmak.

Fonksiyon Nedir?

- Java'da fonksiyonlar sınıf içinde **metot** olarak tanımlanır.
- Belirli bir görevi adlandırır ve tekrar kullanılabilir bir birim oluşturur.
- Girdi **parametrelerle** alınır; sonuç **dönüş değeriyle** verilir.
- Örnek: `Math.sqrt(16)` → 4.0; nasıl hesapladığını bilmemiz gerekmez.



Metot Çalışması



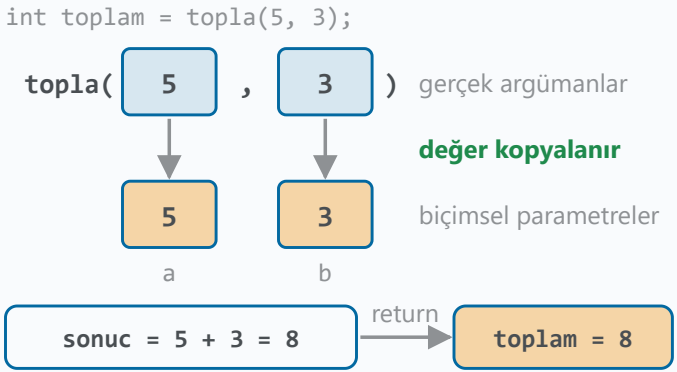
Argümanları değerlendir → **Parametrelere kopyala** → **Gövdeyi çalıştır** → **Değer döndür** → **Çağrıya devam et**

- Her çağrı kendi parametre ve yerel değişkenleriyle çalışır.

Metot Tanımı ve Çağrısı

```
static int topla(int a, int b) {
    int sonuc = a + b;
    return sonuc;
}

int toplam = topla(5, 3);
System.out.println(toplam);
```



- int dönüş tipi; a ve b biçimsel parametrelerdir.
- 5 ve 3 gerçek argümanlardır.
- Dönen değer değişkene atanabilir veya başka ifadede kullanılabilir.

Kod İzleme: `topla(5, 3)`

main() — toplam = topla(5, 3)

main çalışıyor. Sağ taraf önce hesaplanmalı: `topla(5, 3)` çağrılır.

Adım 1 / 6

```
1 static int topla(int a, int b) {
2     int sonuc = a + b;
3     return sonuc;
4 }
5
6 // main içinde:
▶ int toplam = topla(5, 3);
8 System.out.println(toplam);
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `topla(5, 3)`



Yeni **çerçeve** açılır: argümanlar 5 ve 3, parametreler **a** ve **b**'ye kopyalanır.

Adım 2 / 6

```
▶ static int topla(int a, int b) {
2     int sonuc = a + b;
3     return sonuc;
4 }
5
6 // main içinde:
7 int toplam = topla(5, 3);
8 System.out.println(toplam);
```

DEĞİŞKENLER

`a = 5` `b = 3`

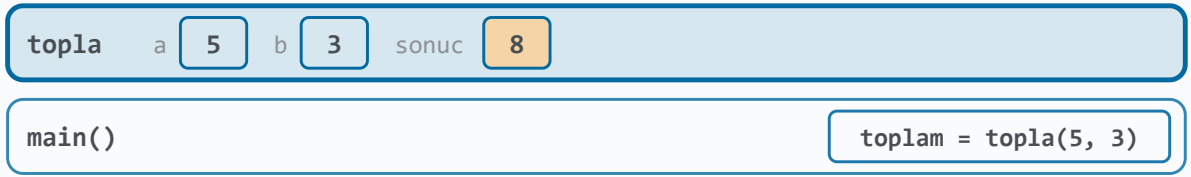
ÇAĞRI YIĞINI

`topla(5, 3)`

`main()`

ÇIKTI

Kod İzleme: `topla(5, 3)`



Yerel değişken **sonuc = 5 + 3 = 8**; yalnızca topla'nın çerçevesinde yaşar.

Adım 3 / 6

```
1 static int topla(int a, int b) {
▶     int sonuc = a + b;
3     return sonuc;
4 }
5
6 // main içinde:
7 int toplam = topla(5, 3);
8 System.out.println(toplam);
```

DEĞİŞKENLER

`a = 5` `b = 3` `sonuc = 8`

ÇAĞRI YIĞINI

`topla(5, 3)`

`main()`

ÇIKTI

Kod İzleme: `topla(5, 3)`



`return sonuc`: 8 çağırana gönderilir, topla'nın çerçevesi kapanır.

Adım 4 / 6

```
1 static int topla(int a, int b) {
2     int sonuc = a + b;
▶     return sonuc;
4 }
5
6 // main içinde:
7 int toplam = topla(5, 3);
8 System.out.println(toplam);
```

DEĞİŞKENLER

`a = 5` `b = 3` `sonuc = 8`

ÇAĞRI YIĞINI

`topla(5, 3)`

`main()`

ÇIKTI

Kod İzleme: `topla(5, 3)`



Dönen değer **çağrının yerine** geçer: `toplam = 8` . `a` , `b` , `sonuc` artık yok.

Adım 5 / 6

```
1 static int topla(int a, int b) {
2     int sonuc = a + b;
3     return sonuc;
4 }
5
6 // main içinde:
▶ int toplam = topla(5, 3);
8 System.out.println(toplam);
```

DEĞİŞKENLER

`toplam = 8`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `topla(5, 3)`



8 yazılır.

Adım 6 / 6

```
1 static int topla(int a, int b) {
2     int sonuc = a + b;
3     return sonuc;
4 }
5
6 // main içinde:
7 int toplam = topla(5, 3);
▶ System.out.println(toplam);
```

DEĞİŞKENLER

`toplam = 8`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
8
```

Çağrı Yığını: metot metodu çağırınca

main() o = ortalama(9, 7)

main, `ortalama(9, 7)`'yi çağırır. Yığında şimdilik tek çerçeve var.

Adım 1 / 9

```
1 static int topla(int a, int b) {
2     return a + b;
3 }
4 static int ortalama(int x, int y) {
5     int t = topla(x, y);
6     return t / 2;
7 }
▶ int o = ortalama(9, 7);  // main içinde
9 System.out.println(o);
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

main()

ÇIKTI

Çağrı Yığını: metot metodu çağırınca



ortalama çerçevesi main'in üstüne konur: x = 9, y = 7.

Adım 2 / 9

```
1 static int topla(int a, int b) {
2     return a + b;
3 }
▶ static int ortalama(int x, int y) {
5     int t = topla(x, y);
6     return t / 2;
7 }
8 int o = ortalama(9, 7);  // main içinde
9 System.out.println(o);
```

DEĞİŞKENLER

x = 9 y = 7

ÇAĞRI YIĞINI

ortalama(9, 7)

main()

ÇIKTI

Çağrı Yığını: metot metodu çağırınca



ortalama da bir metot çağırıyor: `topla(x, y)` → x ve y'nin **değerleri** (9, 7) gönderilir.

Adım 3 / 9

```
1 static int topla(int a, int b) {
2     return a + b;
3 }
4 static int ortalama(int x, int y) {
▶     int t = topla(x, y);
6     return t / 2;
7 }
8 int o = ortalama(9, 7);  // main içinde
9 System.out.println(o);
```

DEĞİŞKENLER

`x = 9` `y = 7`

ÇAĞRI YIĞINI

`ortalama(9, 7)`

`main()`

ÇIKTI

Çağrı Yığını: metot metodu çağırınca



topla en üste: 3 çerçeve. Yalnız en üstteki çalışır.

Adım 4 / 9

```
▶ static int topla(int a, int b) {
2     return a + b;
3 }
4 static int ortalama(int x, int y) {
5     int t = topla(x, y);
6     return t / 2;
7 }
8 int o = ortalama(9, 7);  // main içinde
9 System.out.println(o);
```

DEĞİŞKENLER

a = 9 b = 7

ÇAĞRI YIĞINI

- topla(9, 7)
- ortalama(9, 7)
- main()

ÇIKTI

Çağrı Yığını: metot metodu çağırınca



return 16: topla'nın çerçevesi kapanır (son giren ilk çıkar).

Adım 5 / 9

```
1 static int topla(int a, int b) {
▶     return a + b;
3 }
4 static int ortalama(int x, int y) {
5     int t = topla(x, y);
6     return t / 2;
7 }
8 int o = ortalama(9, 7);  // main içinde
9 System.out.println(o);
```

DEĞİŞKENLER

a = **9** b = **7**

ÇAĞRI YIĞINI

- **topla(9, 7)**
- ortalama(9, 7)
- main()

ÇIKTI

Çağrı Yığını: metot metodu çağırınca



16 çağrı yerine geçer: **t = 16**. ortalama kaldığı yerden sürer.

Adım 6 / 9

```
1 static int topla(int a, int b) {
2     return a + b;
3 }
4 static int ortalama(int x, int y) {
▶     int t = topla(x, y);
6     return t / 2;
7 }
8 int o = ortalama(9, 7);  // main içinde
9 System.out.println(o);
```

DEĞİŞKENLER

x = 9 y = 7 t = 16

ÇAĞRI YIĞINI

ortalama(9, 7)

main()

ÇIKTI

Çağrı Yığını: metot metodu çağırınca



return t / 2 → 16 / 2 = 8 (tam sayı bölmesi). ortalama'nın çerçevesi kapanır.


```
1 static int topla(int a, int b) {
2     return a + b;
3 }
4 static int ortalama(int x, int y) {
5     int t = topla(x, y);
▶     return t / 2;
7 }
8 int o = ortalama(9, 7);  // main içinde
9 System.out.println(o);
```

DEĞİŞKENLER

`x = 9` `y = 7` `t = 16`

ÇAĞRI YIĞINI

- `ortalama(9, 7)`
- `main()`

ÇIKTI

Çağrı Yığını: metot metodu çağırınca



main'e dönüldü: **o = 8**. x, y, t, a, b hiçbiri artık yok.


```
1 static int topla(int a, int b) {
2     return a + b;
3 }
4 static int ortalama(int x, int y) {
5     int t = topla(x, y);
6     return t / 2;
7 }
▶ int o = ortalama(9, 7);  // main içinde
9 System.out.println(o);
```

DEĞİŞKENLER

o = 8

ÇAĞRI YIĞINI

main()

ÇIKTI

Çağrı Yığını: metot metodu çağırınca



8 yazılır. Çerçeveler yığın (stack) gibi: son açılan ilk kapanır.

Adım 9 / 9

```
1 static int topla(int a, int b) {
2     return a + b;
3 }
4 static int ortalama(int x, int y) {
5     int t = topla(x, y);
6     return t / 2;
7 }
8 int o = ortalama(9, 7);  // main içinde
▶ System.out.println(o);
```

DEĞİŞKENLER

o = 8

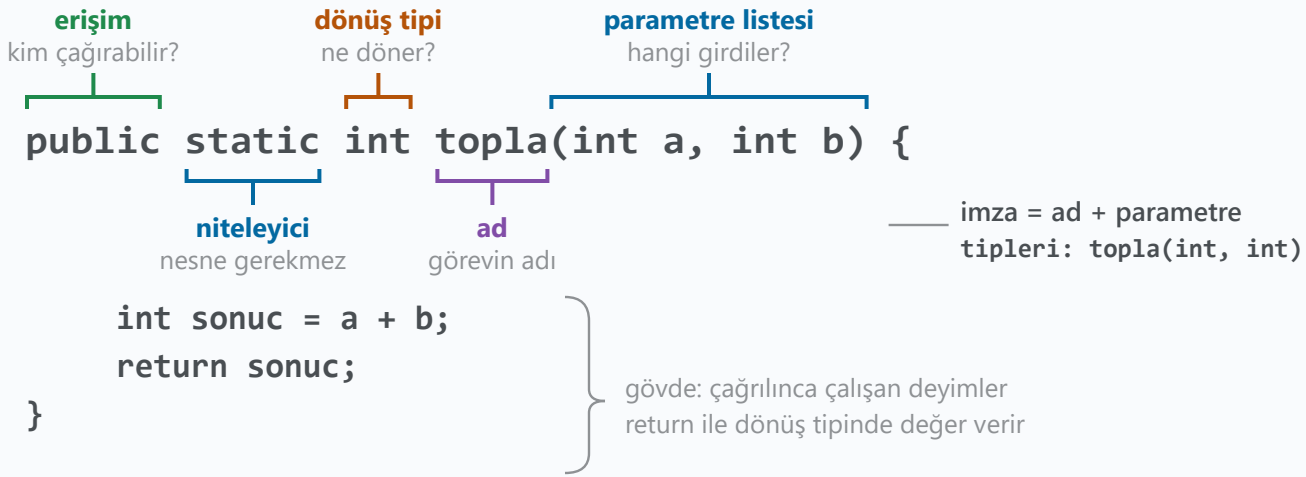
ÇAĞRI YIĞINI

main()

ÇIKTI

```
8
```

Metot Tanımlamasının Parçaları



Örneklerde metotlar `main` ile aynı sınıfta ve `static` yazılır.

`main` Metodunun İmzası ve `static`

`public static void main(String[] args)`
`public` : JVM dışarıdan çağırabilsin
`static` : nesne üretmeden çağrılsın
`void` : değer döndürmez
`main` : JVM'nin aradığı ad
`String[] args` : komut satırı argümanları

```
void selamla() { }
public static void main(String[]
args) {
    selamla();   // hata
}
```

```
error: non-static method selamla() cannot be referenced from a static context
```

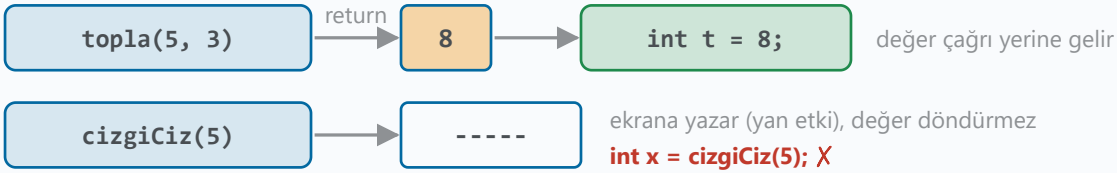
- **static** metot sınıfa aittir; nesne olmadan çağrılır. static olmayan metot bir nesneye aittir.
- static metot, static olmayan metoda ya da alana doğrudan erişemez; çözüm o metodu da `static` yapmaktır.
- JVM programı başlatırken henüz hiç nesne yoktur; bu yüzden `main` static olmalıdır.

DÖNÜŞ

void Metotlar

```java
static void cizgiCiz(int n) {
    for (int i = 0; i < n; i++)
        System.out.print("-");
    System.out.println();
}
// main içinde:
cizgiCiz(5);           // -----
int x = cizgiCiz(5);   // HATA: void değer vermez
```

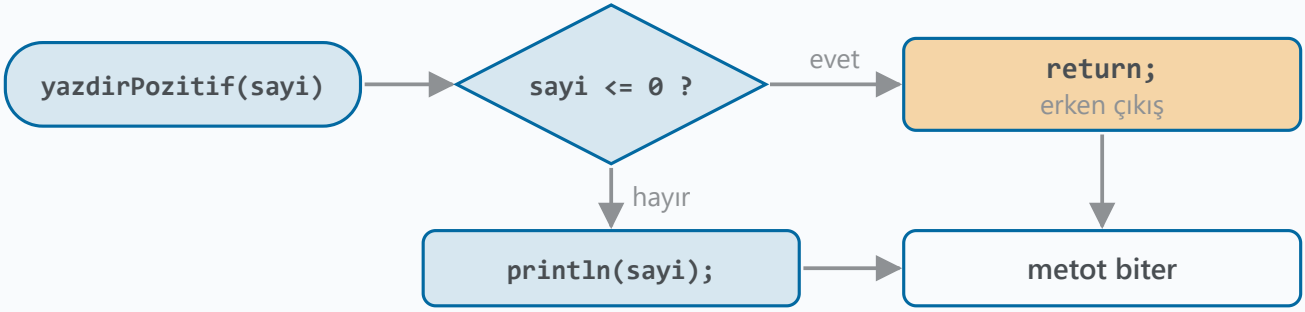
- `void` : metot çağrıya bir değer döndürmez; iş yan etkidir (ekrana yazmak gibi).
- Bu yüzden void çağrı bir atamanın ya da ifadenin parçası olamaz.
- `return;` (değersiz) void metottan erken çıkmak için kullanılabilir.



Erken Dönüş

```
static void yazdirPozitif(int sayi) {
    if (sayi <= 0) return;
    System.out.println(sayi);
}
```

- Özel durumu başta ele almak iç içeliği azaltabilir.
- Değer döndüren metotta her yol uyumlu bir değer döndürmelidir.



`yazdirPozitif(-3)` → çıktı yok

`yazdirPozitif(7)` → 7

Kod İzleme: `yazdirPozitif(-3)` ve `(7)`



İlk çağrı: argüman −3.

Adım 1 / 8

```
1 static void yazdirPozitif(int sayi) {
2     if (sayi <= 0) return;
3     System.out.println(sayi);
4 }
5 // main içinde:
▶ yazdirPozitif(-3);
7 yazdirPozitif(7);
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

DÖNÜŞ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `yazdirPozitif(-3)` ve `(7)`



Çerçeve açılır: **sayi = −3**.

Adım 2 / 8

```
▶ static void yazdirPozitif(int sayi) {
2     if (sayi <= 0) return;
3     System.out.println(sayi);
4 }
5 // main içinde:
6 yazdirPozitif(-3);
7 yazdirPozitif(7);
```

DEĞİŞKENLER

`sayi = -3`

ÇAĞRI YIĞINI

`yazdirPozitif(-3)`

`main()`

ÇIKTI

Kod İzleme: `yazdirPozitif(-3)` ve `(7)`



−3 <= 0 → **true** → `return;` Metot hemen biter; println'e hiç gelinmez.

Adım 3 / 8

```
1 static void yazdirPozitif(int sayi) {
▶     if (sayi <= 0) return;
3     System.out.println(sayi);
4 }
5 // main içinde:
6 yazdirPozitif(-3);
7 yazdirPozitif(7);
```

DEĞİŞKENLER

`sayi = -3`

ÇAĞRI YIĞINI

`yazdirPozitif(-3)`

`main()`

ÇIKTI

Kod İzleme: `yazdirPozitif(-3)` ve `(7)`



main sonraki satırdan sürer: ikinci çağrı, argüman 7.

Adım 4 / 8

```
1 static void yazdirPozitif(int sayi) {
2     if (sayi <= 0) return;
3     System.out.println(sayi);
4 }
5 // main içinde:
6 yazdirPozitif(-3);
▶ yazdirPozitif(7);
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

DÖNÜŞ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `yazdirPozitif(-3)` ve `(7)`



Yepyeni bir çerçeve: **sayi = 7** (önceki çağrının `sayi`'sı çoktan yok oldu).

Adım 5 / 8

```
▶ static void yazdirPozitif(int sayi) {
2     if (sayi <= 0) return;
3     System.out.println(sayi);
4 }
5 // main içinde:
6 yazdirPozitif(-3);
7 yazdirPozitif(7);
```

DEĞİŞKENLER

`sayi = 7`

ÇAĞRI YIĞINI

`yazdirPozitif(7)`

`main()`

ÇIKTI

Kod İzleme: `yazdirPozitif(-3)` ve `(7)`



7 <= 0 → **false** → erken dönüş yok, devam.

Adım 6 / 8

```
1 static void yazdirPozitif(int sayi) {
▶     if (sayi <= 0) return;
3     System.out.println(sayi);
4 }
5 // main içinde:
6 yazdirPozitif(-3);
7 yazdirPozitif(7);
```

DEĞİŞKENLER

`sayi = 7`

ÇAĞRI YIĞINI

`yazdirPozitif(7)`

`main()`

ÇIKTI

Kod İzleme: `yazdirPozitif(-3)` ve `(7)`



7 yazılır.

Adım 7 / 8

```
1 static void yazdirPozitif(int sayi) {
2     if (sayi <= 0) return;
▶     System.out.println(sayi);
4 }
5 // main içinde:
6 yazdirPozitif(-3);
7 yazdirPozitif(7);
```

DEĞİŞKENLER

`sayi = 7`

ÇAĞRI YIĞINI

`yazdirPozitif(7)`

`main()`

ÇIKTI

```
7
```

Kod İzleme: `yazdirPozitif(-3)` ve `(7)`



Kapanış süslü parantezine gelindi: void metot burada kendiliğinden döner.

Adım 8 / 8

```
1 static void yazdirPozitif(int sayi) {
2     if (sayi <= 0) return;
3     System.out.println(sayi);
▶ }
5 // main içinde:
6 yazdirPozitif(-3);
7 yazdirPozitif(7);
```

DEĞİŞKENLER

`sayi = 7`

ÇAĞRI YIĞINI

`yazdirPozitif(7)`

`main()`

ÇIKTI

```
7
```

Derleme Hataları: `return` ve Tip

```
static int yaz(int n) {
    if (n > 0) {
        return 1;
    }
}
```

`error: missing return statement` (kapanan `}` satırında)

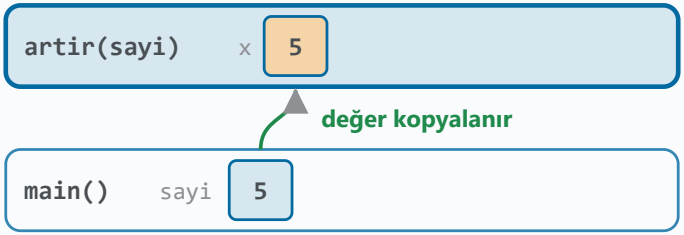
```
static int f() {
    return 2.5;
}
```

`error: incompatible types: possible lossy conversion from double to int`

- void olmayan metotta **her yol** return ile bitmelidir; burada if'in false dalı eksik. Çözüm: sona `return 0;` eklemek.
- Değer, dönüş tipine sığmalıdır: `return (int) 2.5;` derlenir (2 döner) ya da dönüş tipi `double` yapılır.
- `void` metotta `return 1;` → `error: incompatible types: unexpected return value`

Parametre Aktarımı

- Java her zaman **değer ile** aktarır (pass by value).
- İlkel tipte değerin **kopyası** parametreye yazılır: iki ayrı bellek kutusu.
- Metot kendi kopyasını değiştirse de çağıranın değişkeni değişmez.
- Nesnelerde (ör. dizilerde) referansın kopyası aktarılır.



Değer ile Aktarım: `artir(sayi)`



main'de **sayi = 5**: main çerçevesinde bir kutu.

Adım 1 / 6

```
1 static void artir(int x) {
2     x++;
3 }
4 // main içinde:
▶ int sayi = 5;
6 artir(sayi);
7 System.out.println(sayi);   // 5
```

DEĞİŞKENLER

`sayi = 5`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Değer ile Aktarım: `artir(sayi)`



`artir(sayi)` : gönderilen **değişken değil, değeri** (5).

Adım 2 / 6

```
1 static void artir(int x) {
2     x++;
3 }
4 // main içinde:
5 int sayi = 5;
▶ artir(sayi);
7 System.out.println(sayi);   // 5
```

DEĞİŞKENLER

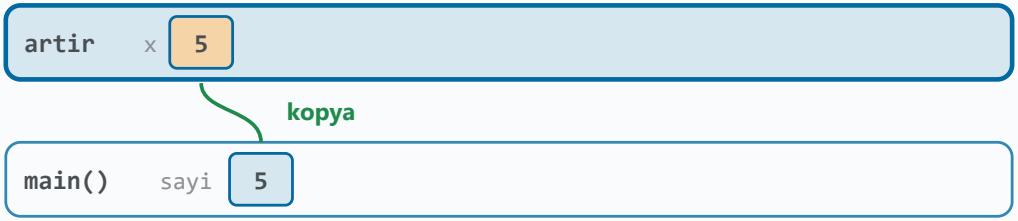
`sayi = 5`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Değer ile Aktarım: `artir(sayi)`



Yeni çerçevede **ayrı bir kutu**: x = 5 (kopya).

Adım 3 / 6

```
▶ static void artir(int x) {
2     x++;
3 }
4 // main içinde:
5 int sayi = 5;
6 artir(sayi);
7 System.out.println(sayi);   // 5
```

DEĞİŞKENLER

x = 5

ÇAĞRI YIĞINI

artir(5)

main()

ÇIKTI

Değer ile Aktarım: `artir(sayi)`



x++ → x = 6. Yalnızca artir'ın kutusu değişti; main'deki `sayi` hâlâ 5.

Adım 4 / 6

```
1 static void artir(int x) {
▶     x++;
3 }
4 // main içinde:
5 int sayi = 5;
6 artir(sayi);
7 System.out.println(sayi);   // 5
```

DEĞİŞKENLER

x = 6

ÇAĞRI YIĞINI

artir(5)

main()

ÇIKTI

Değer ile Aktarım: `artir(sayi)`



Metot biter; çerçeve ve içindeki x (6) silinir. Değer geri gönderilmez (void).

Adım 5 / 6

```
1 static void artir(int x) {
2     x++;
▶ }
4 // main içinde:
5 int sayi = 5;
6 artir(sayi);
7 System.out.println(sayi);   // 5
```

DEĞİŞKENLER

`x = 6`

ÇAĞRI YIĞINI

`artir(5)`

`main()`

ÇIKTI

Değer ile Aktarım: `artir(sayi)`

main() sayi 5

5 yazılır. Değiştirmek isteseydik: `static int artir(int x) { return x + 1; }` ve `sayi = artir(sayi);`

Adım 6 / 6

```
1 static void artir(int x) {
2     x++;
3 }
4 // main içinde:
5 int sayi = 5;
6 artir(sayi);
▶ System.out.println(sayi);   // 5
```

DEĞİŞKENLER

sayi = 5

ÇAĞRI YIĞINI

main()

ÇIKTI

```
5
```

Değer ile Aktarım Tuzağı: `swap(x, y)`



main'de **x = 3** ve **y = 5**: main çerçevesinde iki ayrı kutu.

Adım 1 / 7

```
1 static void swap(int a, int b) {
2     int t = a;
3     a = b;
4     b = t;
5 }
6 // main içinde:
▶ int x = 3, y = 5;
8 swap(x, y);
9 System.out.println(x + " " + y);    // 3 5
```

DEĞİŞKENLER

`x = 3` `y = 5`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Değer ile Aktarım Tuzağı: `swap(x, y)`



`swap(x, y)` : gönderilen **değişkenlerin kendisi değil, değerleri** (3 ve 5).

Adım 2 / 7

```
1 static void swap(int a, int b) {
2     int t = a;
3     a = b;
4     b = t;
5 }
6 // main içinde:
7 int x = 3, y = 5;
▶ swap(x, y);
9 System.out.println(x + " " + y);    // 3 5
```

DEĞİŞKENLER

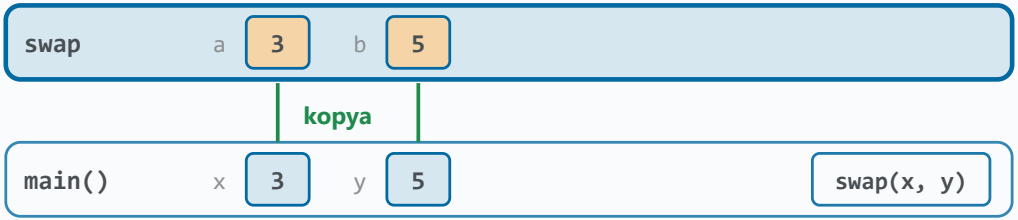
`x = 3` `y = 5`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Değer ile Aktarım Tuzağı: `swap(x, y)`



swap'in çerçevesinde **iki yeni kutu**: a = 3 ve b = 5 (kopyalar). x ve y ile bağlantı yok.

Adım 3 / 7

```
▶ static void swap(int a, int b) {
2     int t = a;
3     a = b;
4     b = t;
5 }
6 // main içinde:
7 int x = 3, y = 5;
8 swap(x, y);
9 System.out.println(x + " " + y);    // 3 5
```

DEĞİŞKENLER

a = 3 b = 5

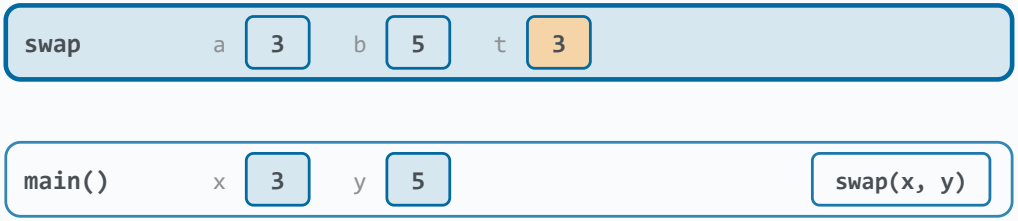
ÇAĞRI YIĞINI

swap(3, 5)

main()

ÇIKTI

Değer ile Aktarım Tuzağı: `swap(x, y)`



t = a → t = 3: geçici kutu a'nın değerini saklar.

Adım 4 / 7

```
1 static void swap(int a, int b) {
▶     int t = a;
3     a = b;
4     b = t;
5 }
6 // main içinde:
7 int x = 3, y = 5;
8 swap(x, y);
9 System.out.println(x + " " + y);   // 3 5
```

DEĞİŞKENLER

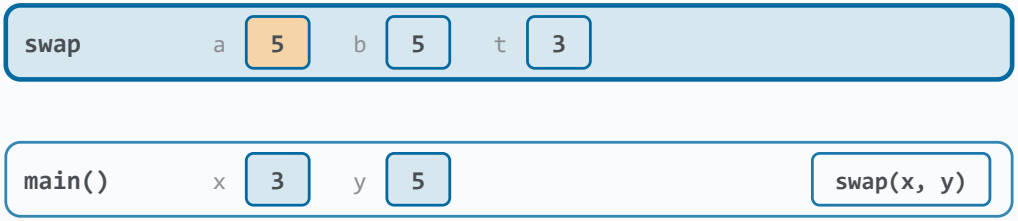
a = 3 | b = 5 | t = 3

ÇAĞRI YIĞINI

- swap(3, 5)
- main()

ÇIKTI

Değer ile Aktarım Tuzağı: `swap(x, y)`



a = b → a = 5. Eski a değeri artık yalnızca t'de.

Adım 5 / 7

```
1 static void swap(int a, int b) {
2     int t = a;
▶     a = b;
4     b = t;
5 }
6 // main içinde:
7 int x = 3, y = 5;
8 swap(x, y);
9 System.out.println(x + " " + y);   // 3 5
```

DEĞİŞKENLER

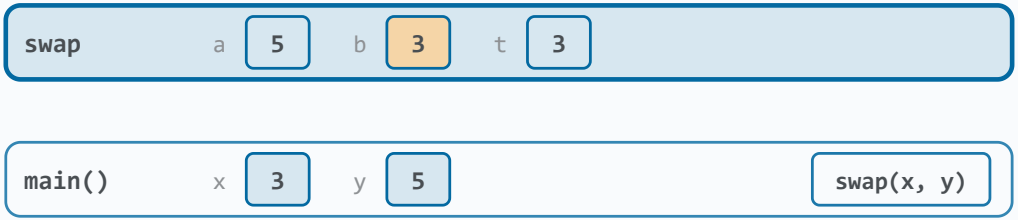
a = 5 | b = 5 | t = 3

ÇAĞRI YIĞINI

- `swap(3, 5)`
- `main()`

ÇIKTI

Değer ile Aktarım Tuzağı: `swap(x, y)`



b = t → b = 3. swap'in **kopyaları** yer değiştirdi (a = 5, b = 3); ama main'deki x ve y'ye hiç dokunulmadı.

Adım 6 / 7

```
1 static void swap(int a, int b) {
2     int t = a;
3     a = b;
▶     b = t;
5 }
6 // main içinde:
7 int x = 3, y = 5;
8 swap(x, y);
9 System.out.println(x + " " + y);   // 3 5
```

DEĞİŞKENLER

`a = 5` `b = 3` `t = 3`

ÇAĞRI YIĞINI

`swap(3, 5)`

`main()`

ÇIKTI

Değer ile Aktarım Tuzağı: `swap(x, y)`



swap bitince çerçevesi (a, b, t) silinir. main'de hâlâ **x = 3, y = 5**: yalnızca kopyalar takas edildi. Gerçek takas için sonucu döndürmek ya da dizi kullanmak gerekir.

Adım 7 / 7

```
1 static void swap(int a, int b) {
2     int t = a;
3     a = b;
4     b = t;
5 }
6 // main içinde:
7 int x = 3, y = 5;
8 swap(x, y);
▶ System.out.println(x + " " + y);   // 3 5
```

DEĞİŞKENLER

x = 3 y = 5

ÇAĞRI YIĞINI

main()

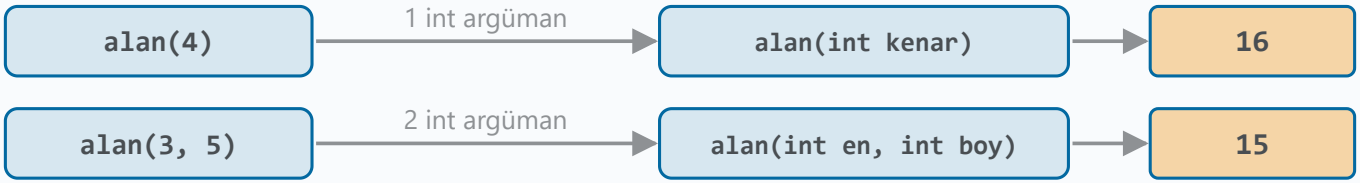
ÇIKTI

```
3 5
```

Metot Aşırı Yükleme

```
static int alan(int kenar) {
    return kenar * kenar;
}
static int alan(int en, int boy) {
    return en * boy;
}
// HATA: yalnız dönüş tipi farklı
static double alan(int kenar) { … }
```

- Aynı ad, **farklı parametre listesi** (sayı, tip veya sıra).
- Yalnızca dönüş tipini değiştirmek yetmez: çağrıdan hangisi olduğu anlaşılmaz.
- Derleyici çağrıdaki argümanlara bakarak uygun imzayı seçer.
- Tanıdık örnek: `println(int)`, `println(double)`, `println(String)`.



Derleyici argümanların sayı ve tipine bakar; dönüş tipine bakmaz.

Kod İzleme: derleyici hangi `alan`'ı seçer?



`alan(4)` : **derleyici** argümanlara bakar: 1 tane int.

Adım 1 / 7

```
1 static int alan(int kenar) {
2     return kenar * kenar;
3 }
4 static int alan(int en, int boy) {
5     return en * boy;
6 }
▶ System.out.println(alan(4));      // main
8 System.out.println(alan(3, 5));   // main
```

DEĞİŞKENLER

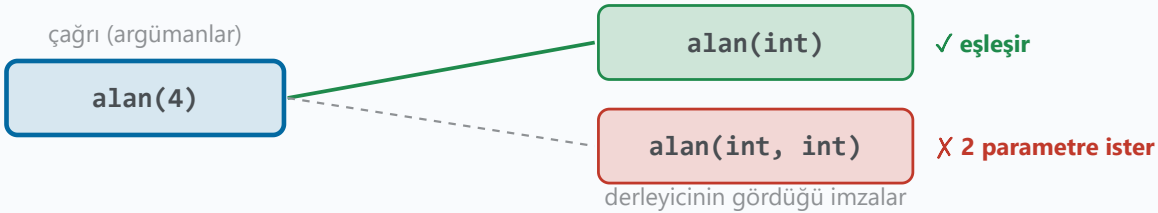
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: derleyici hangi `alan`'ı seçer?



Aynı ada sahip iki metot var; parametre sayısı/tipi uyan **alan(int)** seçilir. Bu karar program çalışmadan, derlemede verilir.

Adım 2 / 7

```
1 static int alan(int kenar) {
2     return kenar * kenar;
3 }
4 static int alan(int en, int boy) {
5     return en * boy;
6 }
▶ System.out.println(alan(4));      // main
8 System.out.println(alan(3, 5));   // main
```

DEĞİŞKENLER

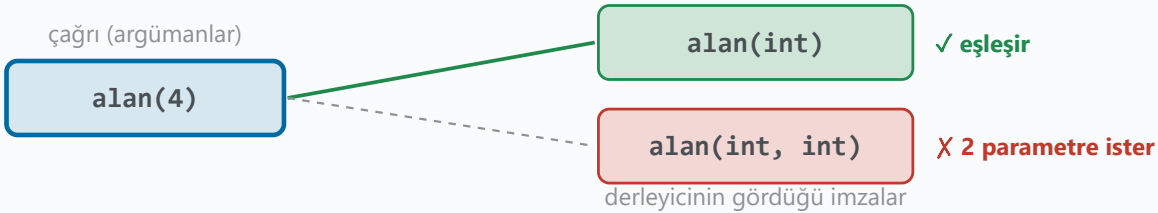
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: derleyici hangi `alan`'ı seçer?



Çalışma zamanında alan(int) çağrılır: kenar = 4 → return 16.

Adım 3 / 7

```
▶ static int alan(int kenar) {
▶     return kenar * kenar;
3 }
4 static int alan(int en, int boy) {
5     return en * boy;
6 }
7 System.out.println(alan(4));      // main
8 System.out.println(alan(3, 5));   // main
```

DEĞİŞKENLER

`kenar = 4`

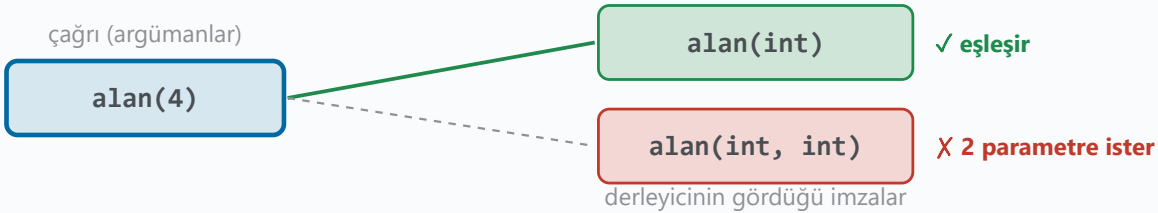
ÇAĞRI YIĞINI

`alan(4)`

`main()`

ÇIKTI

Kod İzleme: derleyici hangi `alan`'ı seçer?



16 yazılır.

Adım 4 / 7

```
1 static int alan(int kenar) {
2     return kenar * kenar;
3 }
4 static int alan(int en, int boy) {
5     return en * boy;
6 }
▶ System.out.println(alan(4));      // main
8 System.out.println(alan(3, 5));   // main
```

DEĞİŞKENLER

—

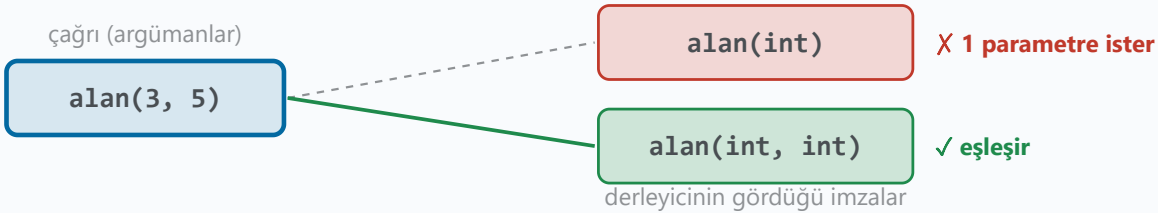
ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
16
```

Kod İzleme: derleyici hangi `alan`'ı seçer?



`alan(3, 5)` : 2 int argüman → **alan(int, int)** seçilir.

Adım 5 / 7

```
1 static int alan(int kenar) {
2     return kenar * kenar;
3 }
4 static int alan(int en, int boy) {
5     return en * boy;
6 }
7 System.out.println(alan(4));      // main
▶ System.out.println(alan(3, 5));   // main
```

DEĞİŞKENLER

—

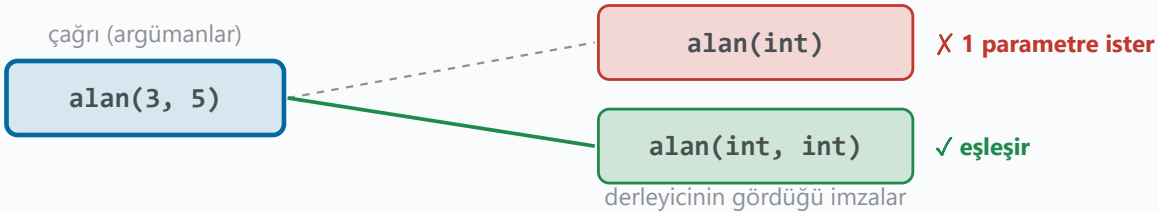
ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
16
```

Kod İzleme: derleyici hangi `alan`'ı seçer?



en = 3, boy = 5 → return 15.

Adım 6 / 7

```
1 static int alan(int kenar) {
2     return kenar * kenar;
3 }
▶ static int alan(int en, int boy) {
▶     return en * boy;
6 }
7 System.out.println(alan(4));      // main
8 System.out.println(alan(3, 5));   // main
```

DEĞİŞKENLER

`en = 3` `boy = 5`

ÇAĞRI YIĞINI

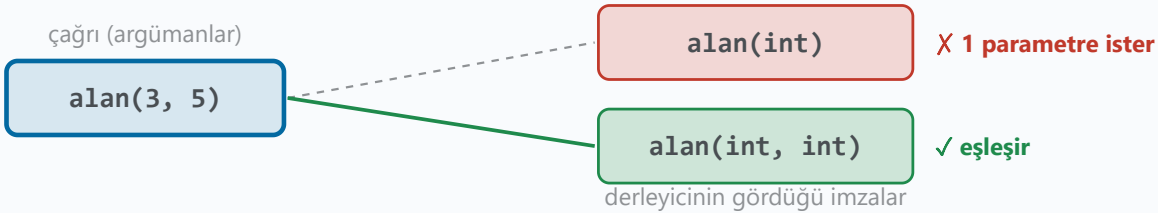
`alan(3, 5)`

`main()`

ÇIKTI

```
16
```

Kod İzleme: derleyici hangi `alan`'ı seçer?



15 yazılır. `alan(2.5)` yazılsaydı hiçbir imza uymaz → **derleme hatası** (double → int otomatik olmaz).

Adım 7 / 7

```
1  static int alan(int kenar) {
2      return kenar * kenar;
3  }
4  static int alan(int en, int boy) {
5      return en * boy;
6  }
7  System.out.println(alan(4));      // main
▶  System.out.println(alan(3, 5));   // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
16
15
```

PARAMETRELER

Değişken Sayıda Argüman (varargs)

```
static int topla(int... s) {
    int t = 0;
    for (int v : s) t += v;
    return t;
}
topla(1, 2, 3)   // 6
topla()          // 0
topla(new int[]{4, 5})   // 9
```

- `int... s` : istenen sayıda argüman; metot içinde **s bir int[] dizisidir**.
- Derleyici çağrıdaki değerleri bir diziye paketler; argüman yoksa **boş dizi** (s.length == 0) olur.
- Varargs parametresi listenin **en sonunda** olmalı ve metotta yalnız bir tane bulunabilir.
- Tanıdık örnek: `printf(String, Object...)` .

Değişken Sayıda Argüman: `topla(1, 2, 3)`



`topla(1, 2, 3)` : derleyici üç değeri **new int[]{1, 2, 3}** dizisine paketler.

Adım 1 / 8

```
1 static int topla(int... s) {
2     int t = 0;
3     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
▶ System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

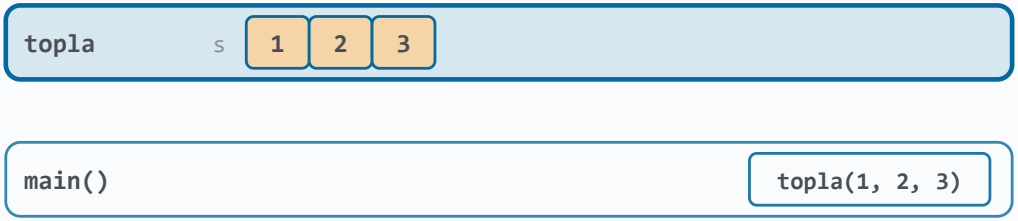
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Değişken Sayıda Argüman: `topla(1, 2, 3)`



s bir **int[]** dizisidir: s = {1, 2, 3}, s.length = 3.

Adım 2 / 8

```
▶ static int topla(int... s) {
2     int t = 0;
3     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = [1, 2, 3]`

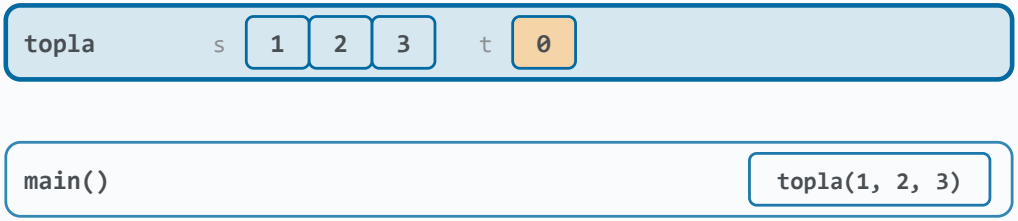
ÇAĞRI YIĞINI

`topla(1, 2, 3)`

`main()`

ÇIKTI

Değişken Sayıda Argüman: `topla(1, 2, 3)`



t = 0: toplam için yerel kutu.

Adım 3 / 8

```
1 static int topla(int... s) {
▶     int t = 0;
3     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = [1, 2, 3]` `t = 0`

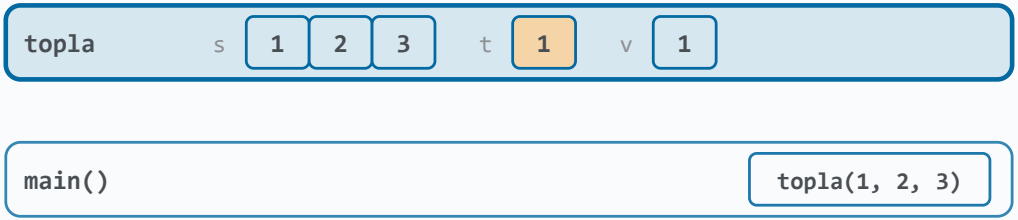
ÇAĞRI YIĞINI

`topla(1, 2, 3)`

`main()`

ÇIKTI

Değişken Sayıda Argüman: `topla(1, 2, 3)`



Gelişmiş for, 1. tur: **v = 1** → t = 0 + 1 = **1**.

Adım 4 / 8

```
1 static int topla(int... s) {
2     int t = 0;
▶     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = [1, 2, 3]` `t = 1` `v = 1`

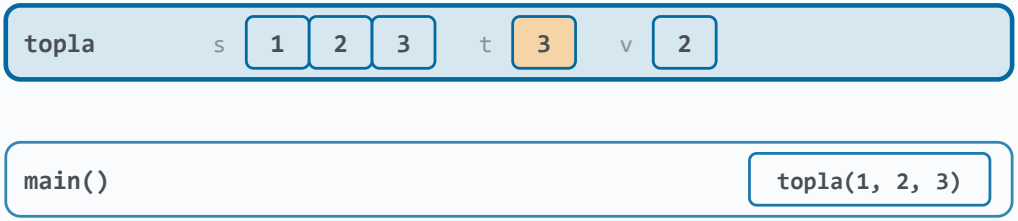
ÇAĞRI YIĞINI

`topla(1, 2, 3)`

`main()`

ÇIKTI

Değişken Sayıda Argüman: `topla(1, 2, 3)`



2. tur: **v = 2** → t = 1 + 2 = **3**.

Adım 5 / 8

```
1 static int topla(int... s) {
2     int t = 0;
▶     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = [1, 2, 3]` `t = 3` `v = 2`

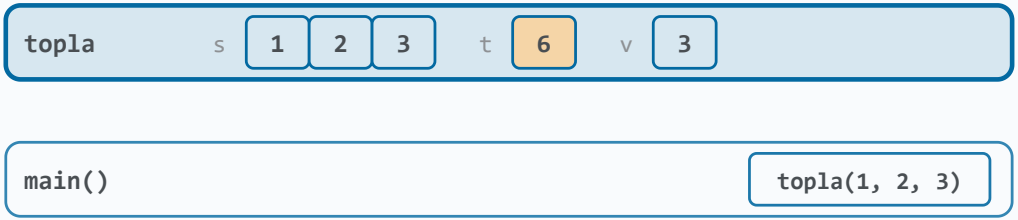
ÇAĞRI YIĞINI

`topla(1, 2, 3)`

`main()`

ÇIKTI

Değişken Sayıda Argüman: `topla(1, 2, 3)`



3. tur: **v = 3** → t = 3 + 3 = **6**. Dizide öğe kalmadı.

Adım 6 / 8

```
1 static int topla(int... s) {
2     int t = 0;
▶     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = [1, 2, 3]` `t = 6` `v = 3`

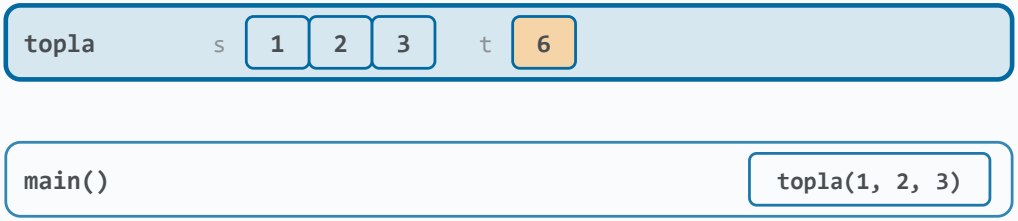
ÇAĞRI YIĞINI

`topla(1, 2, 3)`

`main()`

ÇIKTI

Değişken Sayıda Argüman: `topla(1, 2, 3)`



return t → 6 döner. v döngüyle birlikte kapsam dışında.

Adım 7 / 8

```
1 static int topla(int... s) {
2     int t = 0;
3     for (int v : s) t += v;
▶     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = [1, 2, 3]` `t = 6`

ÇAĞRI YIĞINI

`topla(1, 2, 3)`

`main()`

ÇIKTI

Değişken Sayıda Argüman: `topla(1, 2, 3)`

`main()`

6 yazılır. topla çerçevesi (s, t) silindi.

Adım 8 / 8

```
1 static int topla(int... s) {
2     int t = 0;
3     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
▶ System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
6
```

Boş Çağrı: `topla()`



`topla()` : hiç argüman yok → derleyici **boş dizi** (new int[0]) geçirir; null değil.

Adım 1 / 6

```
1 static int topla(int... s) {
2     int t = 0;
3     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
▶ System.out.println(topla());
```

DEĞİŞKENLER

—

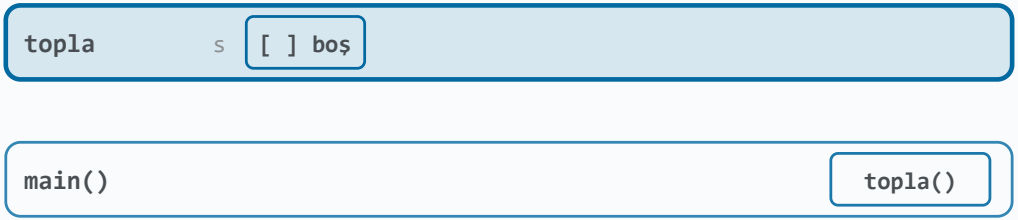
ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
6
```

Boş Çağrı: `topla()`



s boş bir dizi: s.length = 0.

Adım 2 / 6

```
▶ static int topla(int... s) {
2     int t = 0;
3     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = []`

ÇAĞRI YIĞINI

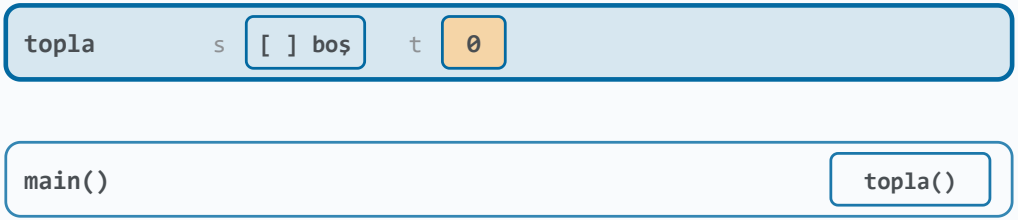
`topla()`

`main()`

ÇIKTI

```
6
```

Boş Çağrı: `topla()`



t = 0.

Adım 3 / 6

```
1 static int topla(int... s) {
▶     int t = 0;
3     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = []` `t = 0`

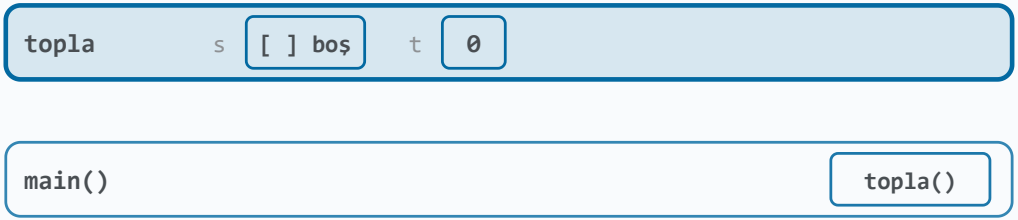
ÇAĞRI YIĞINI

- **topla()**
- main()

ÇIKTI

```
6
```

Boş Çağrı: `topla()`



Dizi boş: gövde **hiç çalışmaz**, t 0 kalır.

Adım 4 / 6

```
1 static int topla(int... s) {
2     int t = 0;
▶     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = []` `t = 0`

ÇAĞRI YIĞINI

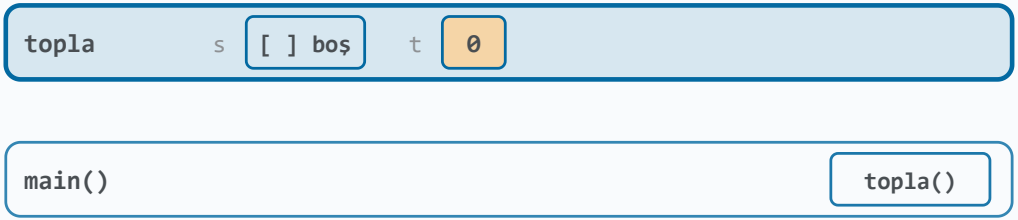
`topla()`

`main()`

ÇIKTI

```
6
```

Boş Çağrı: `topla()`



return t → 0 döner.

Adım 5 / 6

```
1 static int topla(int... s) {
2     int t = 0;
3     for (int v : s) t += v;
▶     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
8 System.out.println(topla());
```

DEĞİŞKENLER

`s = []` `t = 0`

ÇAĞRI YIĞINI

`topla()`

`main()`

ÇIKTI

```
6
```

Boş Çağrı: `topla()`

`main()`

0 yazılır. Boş çağrı hata değildir; sonuç toplamanın etkisiz elemanı olan 0'dır.

Adım 6 / 6

```
1 static int topla(int... s) {
2     int t = 0;
3     for (int v : s) t += v;
4     return t;
5 }
6 // main içinde:
7 System.out.println(topla(1, 2, 3));
▶ System.out.println(topla());
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

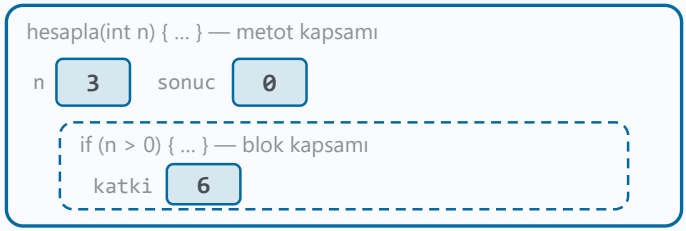
ÇIKTI

```
6
0
```

KAPSAM

Yerel Kapsam

- Parametreler ve metot içinde bildirilen değişkenler **yereldir**.
- Bir blokta (`{ }`) bildirilen değişken yalnızca o blok ve iç bloklarda görünür.
- Aynı kapsamda aynı adlı iki yerel değişken bildirilemez.
- Başka metodun yerel değişkenine erişilemez.



Kapsam Örneği: `hesapla(3)`

hesapla(int n) { ... } — metot kapsamı

main, `hesapla(3)`'ü çağırır.

Adım 1 / 9

```
1 static int hesapla(int n) {
2     int sonuc = 0;
3     if (n > 0) {
4         int katki = n * 2;
5         sonuc += katki;
6     }
7     return sonuc;
8 }
▶ System.out.println(hesapla(3));  // main
```

DEĞİŞKENLER

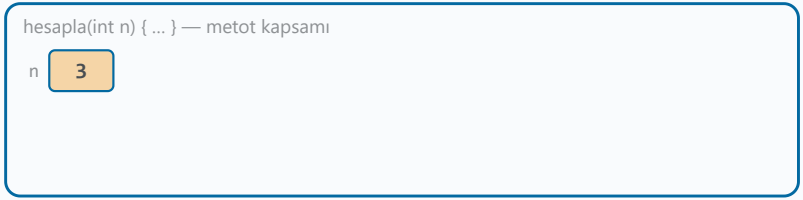
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kapsam Örneği: `hesapla(3)`



Metot kapsamı açılır: parametre **n = 3**.

Adım 2 / 9

```
▶ static int hesapla(int n) {
2     int sonuc = 0;
3     if (n > 0) {
4         int katki = n * 2;
5         sonuc += katki;
6     }
7     return sonuc;
8 }
9 System.out.println(hesapla(3));  // main
```

DEĞİŞKENLER

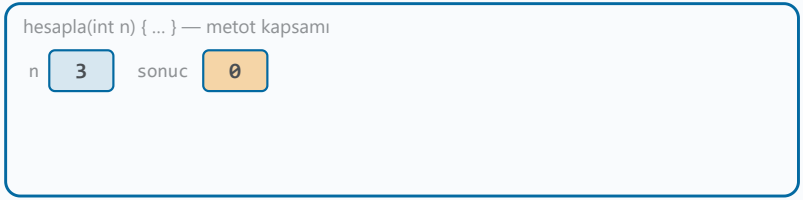
`n = 3`

ÇAĞRI YIĞINI

- `hesapla(3)`
- `main()`

ÇIKTI

Kapsam Örneği: `hesapla(3)`



`sonuc` metot kapsamında: metodun sonuna kadar görünür.

Adım 3 / 9

```
1 static int hesapla(int n) {
▶     int sonuc = 0;
3     if (n > 0) {
4         int katki = n * 2;
5         sonuc += katki;
6     }
7     return sonuc;
8 }
9 System.out.println(hesapla(3));  // main
```

DEĞİŞKENLER

`n = 3` `sonuc = 0`

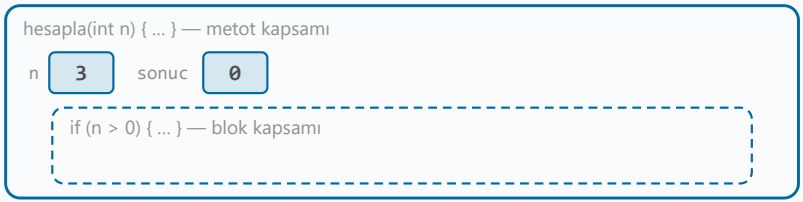
ÇAĞRI YIĞINI

`hesapla(3)`

`main()`

ÇIKTI

Kapsam Örneği: `hesapla(3)`



3 > 0 → **true**: if bloğuna girilir, yeni bir iç kapsam açılır.

Adım 4 / 9

```
1 static int hesapla(int n) {
2     int sonuc = 0;
▶     if (n > 0) {
4         int katki = n * 2;
5         sonuc += katki;
6     }
7     return sonuc;
8 }
9 System.out.println(hesapla(3));  // main
```

DEĞİŞKENLER

`n = 3` `sonuc = 0`

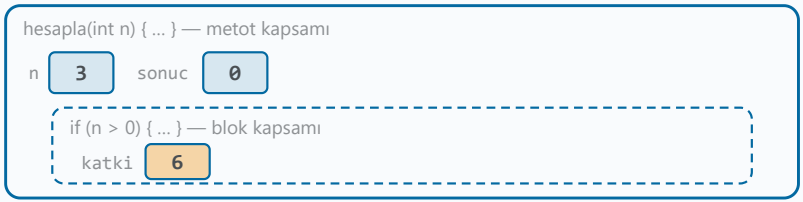
ÇAĞRI YIĞINI

`hesapla(3)`

`main()`

ÇIKTI

Kapsam Örneği: `hesapla(3)`



katki = 3 × 2 = 6: yalnızca if bloğunun içinde var.

Adım 5 / 9

```
1 static int hesapla(int n) {
2     int sonuc = 0;
3     if (n > 0) {
▶         int katki = n * 2;
5         sonuc += katki;
6     }
7     return sonuc;
8 }
9 System.out.println(hesapla(3));  // main
```

DEĞİŞKENLER

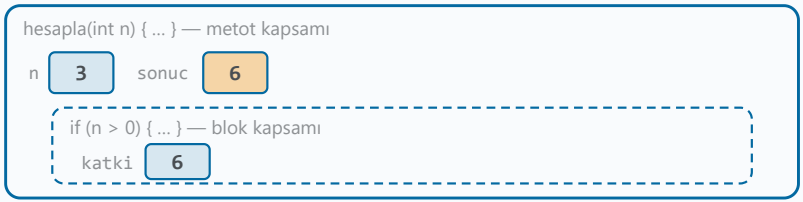
`n = 3` `sonuc = 0` `katki = 6`

ÇAĞRI YIĞINI

- `hesapla(3)`
- `main()`

ÇIKTI

Kapsam Örneği: `hesapla(3)`



İç bloktan dış kapsamdaki `sonuc`'a erişilebilir: sonuc = 0 + 6 = **6**.

Adım 6 / 9

```
1 static int hesapla(int n) {
2     int sonuc = 0;
3     if (n > 0) {
4         int katki = n * 2;
▶         sonuc += katki;
6     }
7     return sonuc;
8 }
9 System.out.println(hesapla(3));  // main
```

DEĞİŞKENLER

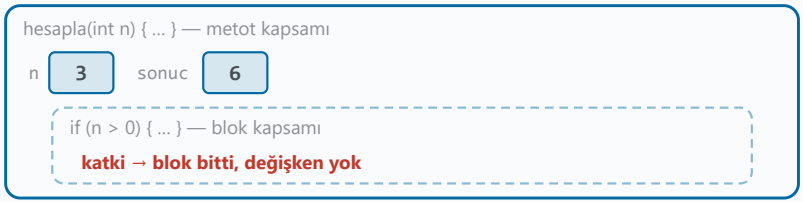
`n = 3` `sonuc = 6` `katki = 6`

ÇAĞRI YIĞINI

- `hesapla(3)`
- `main()`

ÇIKTI

Kapsam Örneği: `hesapla(3)`



Blok kapandı: **katki yok oldu**. Burada `katki` yazmak derleme hatası olurdu.

Adım 7 / 9

```
1 static int hesapla(int n) {
2     int sonuc = 0;
3     if (n > 0) {
4         int katki = n * 2;
5         sonuc += katki;
▶     }
7     return sonuc;
8 }
9 System.out.println(hesapla(3));  // main
```

DEĞİŞKENLER

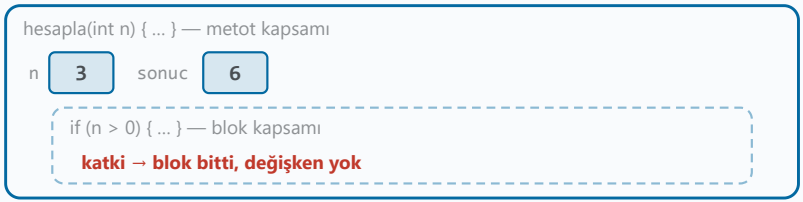
`n = 3` `sonuc = 6`

ÇAĞRI YIĞINI

- **hesapla(3)**
- main()

ÇIKTI

Kapsam Örneği: `hesapla(3)`



return 6: metot kapsamı da kapanır (`n` ve `sonuc` silinir).

Adım 8 / 9

```
1 static int hesapla(int n) {
2     int sonuc = 0;
3     if (n > 0) {
4         int katki = n * 2;
5         sonuc += katki;
6     }
▶     return sonuc;
8 }
9 System.out.println(hesapla(3));  // main
```

DEĞİŞKENLER

`n = 3` `sonuc = 6`

ÇAĞRI YIĞINI

- **hesapla(3)**
- main()

ÇIKTI

Kapsam Örneği: `hesapla(3)`

hesapla(int n) { ... } — metot kapsamı

Dönen 6 println'e argüman olur ve yazılır.

Adım 9 / 9

```
1 static int hesapla(int n) {
2     int sonuc = 0;
3     if (n > 0) {
4         int katki = n * 2;
5         sonuc += katki;
6     }
7     return sonuc;
8 }
▶ System.out.println(hesapla(3));  // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

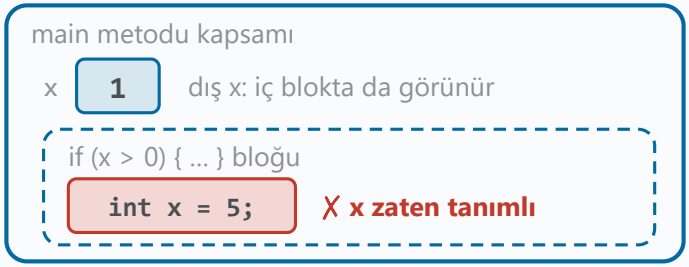
`main()`

ÇIKTI

```
6
```

Kapsam: Aynı Adlı Değişken

```
int x = 1;
if (x > 0) {
    int x = 5;    // hata
}
```




```
error: variable x is already defined in method main(String[])
```

- İç blokta dış kapsamdaki yerel değişkenle aynı ad kullanılamaz; yerel değişken yerel değişkeni **gölgelemez**.
- Parametre de yereldir: `topla(int a, int b)` içinde `int a = 3;` aynı hatayı verir.
- Blok bitince değişken silinir: `for (int i …) { }` sonrası `int i = 7;` yazılabilir.
- static alanla aynı adlı **yerel** değişken derlenir; yerel olan görünür, alan değişmez.

KÜTÜPHANE

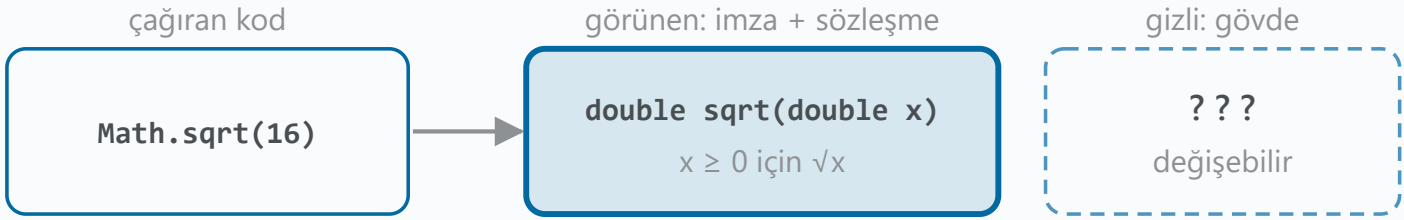
Sık Kullanılan Hazır Metotlar: Math

Çağrı	Sonuç	Dönüş tipi	Not
`Math.abs(-7)`	7	int	mutlak değer (aşırı yüklü: int, long, double…)
`Math.max(3, 9)`	9	int	büyük olanı döndürür
`Math.sqrt(16)`	4.0	double	karekök; sonuç her zaman double
`Math.pow(2, 10)`	1024.0	double	üs; int'e atamak için (int) gerekir
`Math.round(2.5)`	3	long	en yakın tam sayıya yuvarlar

- Belgelerde parametre, dönüş tipi ve özel durumlar okunmalı (ör. `Math.sqrt(-1)` → NaN).

TASARIM

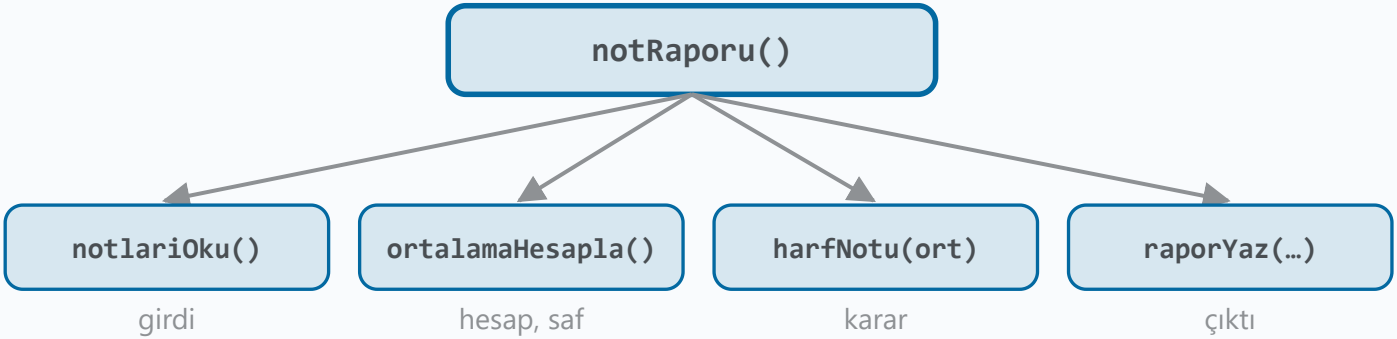
Soyutlama



- Metot, çağıran koddan uygulama ayrıntısını saklar.
- İyi adlandırılmış metot **ne** yapıldığını söyler; **nasıl** yapıldığı gövdede kalır.
- Gövde değişse bile imza aynı kaldıkça çağıranlar etkilenmez.

TASARIM

Ayrıştırma



- Büyük görev tek sorumluluklu alt görevlere ayrılır.
- Her alt görev için girdi, çıktı ve hata davranışı belirlenir.
- Üst düzey metot bu görevleri anlamlı sırada birleştirir; okuması içindekiler tablosu gibi olur.

TASARIM

İyi Metot Tasarımı

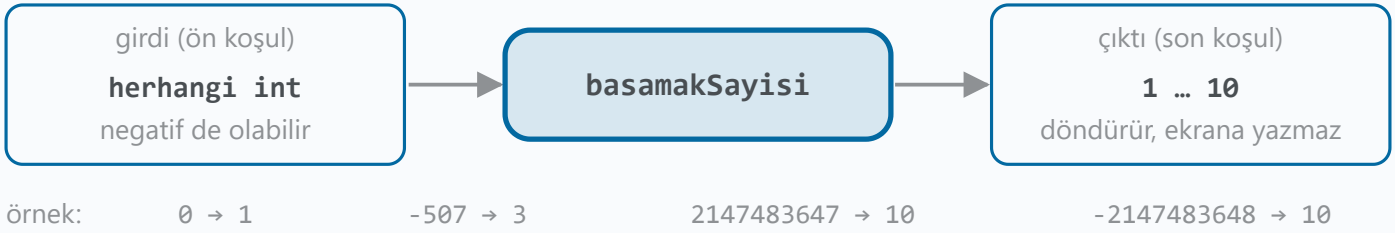
İlke	Soru
Tek görev	Metot tek cümleyle açıklanabiliyor mu?
Anlamlı ad	Ad sonuç veya eylemi söylüyor mu?
Az parametre	Girdilerin hepsi gerekli mi?
Belirgin sonuç	Dönüş ve yan etkiler açık mı?
Test edilebilirlik	Metot bağımsız örneklerle denenebilir mi?

ÖRNEK

Önce Metot Sözleşmesi

- Metot adı yaptığı işi açıkça belirtmelidir.
- Parametrelerin geçerli değer aralığı tanımlanmalıdır.
- Dönüş değerinin anlamı ve hata davranışı bilinmelidir.

```
// Görev : sayının onluk basamak sayısını döndürür.
// Girdi : herhangi bir int (negatif de olabilir).
// Çıktı : 1..10; 0 için 1, -507 için 3.
static int basamakSayisi(int sayi) { … }
```



ÖRNEK

Tek / Çift

```
static boolean ciftMi(int sayi) {
    return sayi % 2 == 0;
}
```

- boolean sonuç kararlarda doğrudan kullanılabilir.
- Metot ekrana yazmak yerine sonucu döndürür.

sayi	7	10	-4	-3
sayi % 2	1	0	0	-1
ciftMi(sayi)	false	true	true	false

Java'da negatif sayının kalanı negatif olabilir (-3 % 2 = -1); bu yüzden == 0 ile bakılır.

Kod İzleme: `ciftMi(7)`

main() if (ciftMi(7))

if koşulunu bulmak için önce `ciftMi(7)` çağrılır.

Adım 1 / 5

```
1 static boolean ciftMi(int sayi) {
2     return sayi % 2 == 0;
3 }
4 // main içinde:
▶ if (ciftMi(7)) System.out.println("çift");
6 else System.out.println("tek");
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `ciftMi(7)`



Çerçeve açılır: sayi = 7.

Adım 2 / 5

```
▶ static boolean ciftMi(int sayi) {
2     return sayi % 2 == 0;
3 }
4 // main içinde:
5 if (ciftMi(7)) System.out.println("çift");
6 else System.out.println("tek");
```

DEĞİŞKENLER

`sayi = 7`

ÇAĞRI YIĞINI

`ciftMi(7)`

`main()`

ÇIKTI

Kod İzleme: `ciftMi(7)`



7 % 2 = 1 → 1 == 0 → **false**. return false.

Adım 3 / 5

```
1 static boolean ciftMi(int sayi) {
▶     return sayi % 2 == 0;
3 }
4 // main içinde:
5 if (ciftMi(7)) System.out.println("çift");
6 else System.out.println("tek");
```

DEĞİŞKENLER

`sayi = 7`

ÇAĞRI YIĞINI

`ciftMi(7)`

`main()`

ÇIKTI

Kod İzleme: `ciftMi(7)`

main() if (false)

Dönüş değeri çağrının yerine geçer: `if (false)` → then dalı atlanır.

Adım 4 / 5

```
1 static boolean ciftMi(int sayi) {
2     return sayi % 2 == 0;
3 }
4 // main içinde:
▶ if (ciftMi(7)) System.out.println("çift");
6 else System.out.println("tek");
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `ciftMi(7)`

`main()`

else dalı: **tek** yazılır. Metot yalnızca hesapladı; yazma kararı çağırana ait.

Adım 5 / 5

```
1 static boolean ciftMi(int sayi) {
2     return sayi % 2 == 0;
3 }
4 // main içinde:
5 if (ciftMi(7)) System.out.println("çift");
▶ else System.out.println("tek");
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
tek
```

ÖRNEK

Basamak Sayısı

```
static int basamakSayisi(int sayi) {
    long kalan = Math.abs((long) sayi);
    if (kalan == 0) return 1;
    int adet = 0;
    while (kalan > 0) {
        adet++;
        kalan /= 10;
    }
    return adet;
}
System.out.println(basamakSayisi(-507));
```

- `(long)` önce dönüştürür: `Math.abs(-2147483648)` int'te yine negatif kalırdı.
- 0 için erken dönüş: 1 basamak.
- Sonuç ekrana yazılmaz, **döndürülür**; çağıran istediği gibi kullanır.

Kod İzleme: `basamakSayisi(-507)`

`main()` `println(basamakSayisi(-507))`

println'in argümanı için önce `basamakSayisi(-507)` çağrılır.

Adım 1 / 11

```
1  static int basamakSayisi(int sayi) {
2      long kalan = Math.abs((long) sayi);
3      if (kalan == 0) return 1;
4      int adet = 0;
5      while (kalan > 0) {
6          adet++;
7          kalan /= 10;
8      }
9      return adet;
10 }
▶  System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`



Çerçeve açılır: sayi = −507.

Adım 2 / 11

```
▶ static int basamakSayisi(int sayi) {
2     long kalan = Math.abs((long) sayi);
3     if (kalan == 0) return 1;
4     int adet = 0;
5     while (kalan > 0) {
6         adet++;
7         kalan /= 10;
8     }
9     return adet;
10 }
11 System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

`sayi = -507`

ÇAĞRI YIĞINI

`basamakSayisi(-507)`

`main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`



Math.abs((long) −507) → **kalan = 507**. (Hazır metot da bir çağrıdır: değer döndürür.)

Adım 3 / 11

```
1  static int basamakSayisi(int sayi) {
▶      long kalan = Math.abs((long) sayi);
3      if (kalan == 0) return 1;
4      int adet = 0;
5      while (kalan > 0) {
6          adet++;
7          kalan /= 10;
8      }
9      return adet;
10 }
11 System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

`sayi = -507` `kalan = 507`

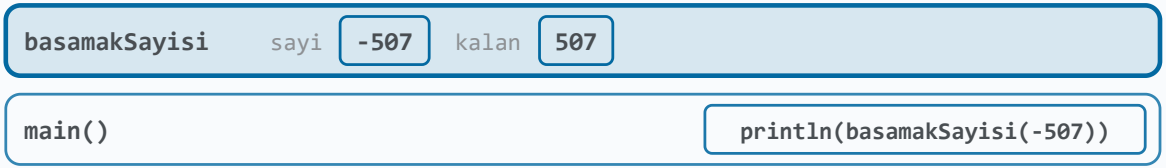
ÇAĞRI YIĞINI

`basamakSayisi(-507)`

`main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`



507 == 0 → false: erken dönüş yok.

Adım 4 / 11

```
1  static int basamakSayisi(int sayi) {
2      long kalan = Math.abs((long) sayi);
▶      if (kalan == 0) return 1;
4      int adet = 0;
5      while (kalan > 0) {
6          adet++;
7          kalan /= 10;
8      }
9      return adet;
10 }
11 System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

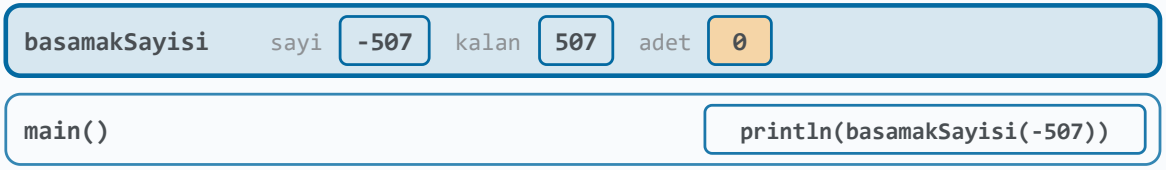
`sayi = -507` `kalan = 507`

ÇAĞRI YIĞINI

- `basamakSayisi(-507)`
- `main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`



Sayaç adet = 0.

Adım 5 / 11

```
1  static int basamakSayisi(int sayi) {
2      long kalan = Math.abs((long) sayi);
3      if (kalan == 0) return 1;
▶      int adet = 0;
5      while (kalan > 0) {
6          adet++;
7          kalan /= 10;
8      }
9      return adet;
10 }
11 System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

`sayi = -507` `kalan = 507` `adet = 0`

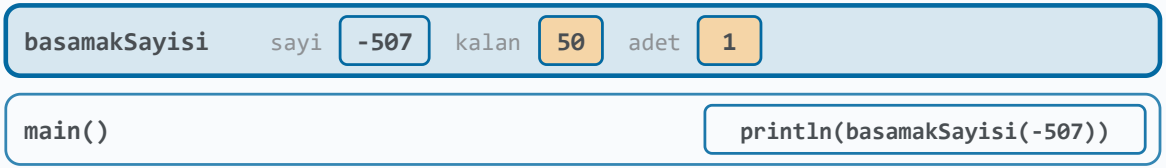
ÇAĞRI YIĞINI

`basamakSayisi(-507)`

`main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`



Tur 1: 507 > 0 → adet = 1, kalan = 507 / 10 = 50.

Adım 6 / 11

```
1  static int basamakSayisi(int sayi) {
2      long kalan = Math.abs((long) sayi);
3      if (kalan == 0) return 1;
4      int adet = 0;
▶      while (kalan > 0) {
▶          adet++;
▶          kalan /= 10;
8      }
9      return adet;
10 }
11 System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

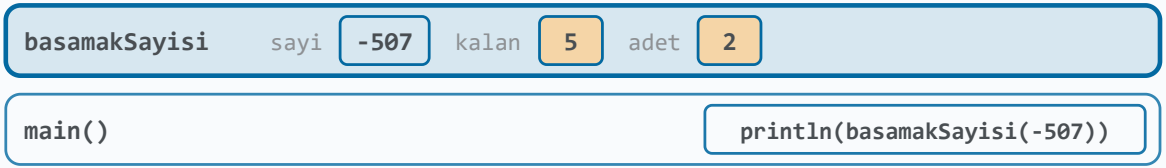
`sayi = -507` `kalan = 50` `adet = 1`

ÇAĞRI YIĞINI

- `basamakSayisi(-507)`
- `main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`



Tur 2: 50 > 0 → adet = 2, kalan = 50 / 10 = 5.

Adım 7 / 11

```
1  static int basamakSayisi(int sayi) {
2      long kalan = Math.abs((long) sayi);
3      if (kalan == 0) return 1;
4      int adet = 0;
▶      while (kalan > 0) {
▶          adet++;
▶          kalan /= 10;
8      }
9      return adet;
10 }
11 System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

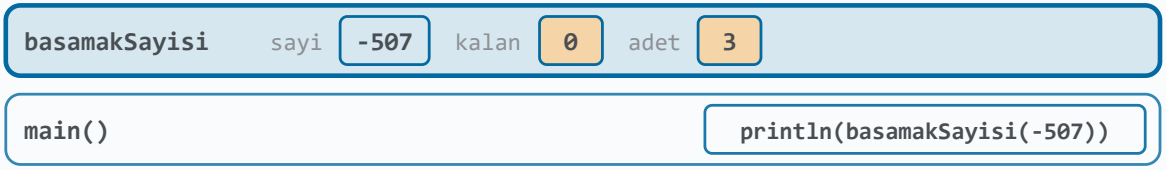
`sayi = -507` `kalan = 5` `adet = 2`

ÇAĞRI YIĞINI

- `basamakSayisi(-507)`
- `main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`



Tur 3: 5 > 0 → adet = 3, kalan = 5 / 10 = 0.

Adım 8 / 11

```
1  static int basamakSayisi(int sayi) {
2      long kalan = Math.abs((long) sayi);
3      if (kalan == 0) return 1;
4      int adet = 0;
▶      while (kalan > 0) {
▶          adet++;
▶          kalan /= 10;
8      }
9      return adet;
10 }
11 System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

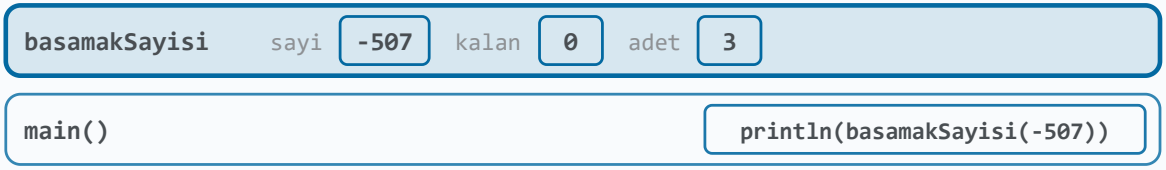
`sayi = -507` `kalan = 0` `adet = 3`

ÇAĞRI YIĞINI

- `basamakSayisi(-507)`
- `main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`



0 > 0 → false: döngü biter.

Adım 9 / 11

```
1  static int basamakSayisi(int sayi) {
2      long kalan = Math.abs((long) sayi);
3      if (kalan == 0) return 1;
4      int adet = 0;
▶      while (kalan > 0) {
6          adet++;
7          kalan /= 10;
8      }
9      return adet;
10 }
11 System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

`sayi = -507` `kalan = 0` `adet = 3`

ÇAĞRI YIĞINI

- `basamakSayisi(-507)`
- `main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`



return 3: çerçeve kapanır.

Adım 10 / 11

```
1  static int basamakSayisi(int sayi) {
2      long kalan = Math.abs((long) sayi);
3      if (kalan == 0) return 1;
4      int adet = 0;
5      while (kalan > 0) {
6          adet++;
7          kalan /= 10;
8      }
▶      return adet;
10 }
11 System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

`sayi = -507` `kalan = 0` `adet = 3`

ÇAĞRI YIĞINI

- `basamakSayisi(-507)`
- `main()`

ÇIKTI

Kod İzleme: `basamakSayisi(-507)`

`main()` `println(3)`

Çağrının yerine 3 geçer: `println(3)` → **3** yazılır.

Adım 11 / 11

```
1  static int basamakSayisi(int sayi) {
2      long kalan = Math.abs((long) sayi);
3      if (kalan == 0) return 1;
4      int adet = 0;
5      while (kalan > 0) {
6          adet++;
7          kalan /= 10;
8      }
9      return adet;
10 }
▶  System.out.println(basamakSayisi(-507));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
3
```

ÖRNEK

Asal mı?

```
static boolean asalMi(int n) {
    if (n < 2) return false;
    for (int i = 2; i <= n / i; i++)
        if (n % i == 0) return false;
    return true;
}
```

- İlk bölen bulunduğunda erken dönüş yapılır.

asalMi(37): yalnızca i ≤ n / i olan i değerleri denenir (√37 ≈ 6,08)

i	2	3	4	5	6	7
n / i	18	12	9	7	6	5
37 % i	1	1	1	2	1	denenmez

Hiçbir kalan 0 değil; i = 7 için 7 > 37 / 7 = 5 → döngü biter → return true

Kod İzleme: `asalMi(15)` ve `asalMi(7)`



`asalMi(15)` çağrılır.

Adım 1 / 11

```
1 static boolean asalMi(int n) {
2     if (n < 2) return false;
3     for (int i = 2; i <= n / i; i++)
4         if (n % i == 0) return false;
5     return true;
6 }
▶ System.out.println(asalMi(15));  // main
8 System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `asalMi(15)` ve `asalMi(7)`



n = 15: 15 < 2 → false.

Adım 2 / 11

```
1 static boolean asalMi(int n) {
▶     if (n < 2) return false;
3     for (int i = 2; i <= n / i; i++)
4         if (n % i == 0) return false;
5     return true;
6 }
7 System.out.println(asalMi(15));  // main
8 System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

`n = 15`

ÇAĞRI YIĞINI

`asalMi(15)`

`main()`

ÇIKTI

Kod İzleme: `asalMi(15)` ve `asalMi(7)`



2 <= 15/2 = 7 → true; 15 % 2 = 1 → bölmüyor, i++.

Adım 3 / 11

```
1 static boolean asalMi(int n) {
2     if (n < 2) return false;
▶     for (int i = 2; i <= n / i; i++)
▶         if (n % i == 0) return false;
5     return true;
6 }
7 System.out.println(asalMi(15));  // main
8 System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

`n = 15` `i = 2`

ÇAĞRI YIĞINI

`asalMi(15)`

`main()`

ÇIKTI

Kod İzleme: `asalMi(15)` ve `asalMi(7)`



3 <= 5 → true; 15 % 3 = 0 → **return false**: kalan turlar hiç çalışmaz (erken dönüş).

Adım 4 / 11

```
1 static boolean asalMi(int n) {
2     if (n < 2) return false;
▶     for (int i = 2; i <= n / i; i++)
▶         if (n % i == 0) return false;
5     return true;
6 }
7 System.out.println(asalMi(15));  // main
8 System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

`n = 15` `i = 3`

ÇAĞRI YIĞINI

`asalMi(15)`

`main()`

ÇIKTI

Kod İzleme: `asalMi(15)` ve `asalMi(7)`

`main()` `println(false)`

Dönen **false** println'e verilir ve yazılır.

Adım 5 / 11

```
1 static boolean asalMi(int n) {
2     if (n < 2) return false;
3     for (int i = 2; i <= n / i; i++)
4         if (n % i == 0) return false;
5     return true;
6 }
▶ System.out.println(asalMi(15));  // main
8 System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
false
```

Kod İzleme: `asalMi(15)` ve `asalMi(7)`

`main()` `println(asalMi(7))`

`asalMi(7)` çağrılır.

Adım 6 / 11

```
1 static boolean asalMi(int n) {
2     if (n < 2) return false;
3     for (int i = 2; i <= n / i; i++)
4         if (n % i == 0) return false;
5     return true;
6 }
7 System.out.println(asalMi(15));  // main
▶ System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
false
```

Kod İzleme: `asalMi(15)` ve `asalMi(7)`



n = 7: 7 < 2 → false.

Adım 7 / 11

```
1 static boolean asalMi(int n) {
▶     if (n < 2) return false;
3     for (int i = 2; i <= n / i; i++)
4         if (n % i == 0) return false;
5     return true;
6 }
7 System.out.println(asalMi(15));  // main
8 System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

`n = 7`

ÇAĞRI YIĞINI

`asalMi(7)`

`main()`

ÇIKTI

```
false
```

Kod İzleme: `asalMi(15)` ve `asalMi(7)`



2 <= 7/2 = 3 → true; 7 % 2 = 1 → bölmüyor, i++.

Adım 8 / 11

```
1 static boolean asalMi(int n) {
2     if (n < 2) return false;
▶     for (int i = 2; i <= n / i; i++)
▶         if (n % i == 0) return false;
5     return true;
6 }
7 System.out.println(asalMi(15));  // main
8 System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

`n = 7` `i = 2`

ÇAĞRI YIĞINI

`asalMi(7)`

`main()`

ÇIKTI

```
false
```

Kod İzleme: `asalMi(15)` ve `asalMi(7)`



3 <= 7/3 = 2 → **false**: √7'ye kadar bölen yok.

Adım 9 / 11

```
1 static boolean asalMi(int n) {
2     if (n < 2) return false;
▶     for (int i = 2; i <= n / i; i++)
4         if (n % i == 0) return false;
5     return true;
6 }
7 System.out.println(asalMi(15));  // main
8 System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

`n = 7` `i = 3`

ÇAĞRI YIĞINI

`asalMi(7)`

`main()`

ÇIKTI

```
false
```

Kod İzleme: `asalMi(15)` ve `asalMi(7)`



Döngü erken dönüş olmadan bitti → **return true**.

Adım 10 / 11

```
1 static boolean asalMi(int n) {
2     if (n < 2) return false;
3     for (int i = 2; i <= n / i; i++)
4         if (n % i == 0) return false;
▶       return true;
6 }
7 System.out.println(asalMi(15));  // main
8 System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

`n = 7`

ÇAĞRI YIĞINI

`asalMi(7)`

`main()`

ÇIKTI

```
false
```

Kod İzleme: `asalMi(15)` ve `asalMi(7)`



Dönen **true** println'e verilir ve yazılır.

Adım 11 / 11

```
1  static boolean asalMi(int n) {
2      if (n < 2) return false;
3      for (int i = 2; i <= n / i; i++)
4          if (n % i == 0) return false;
5      return true;
6  }
7  System.out.println(asalMi(15));  // main
▶  System.out.println(asalMi(7));   // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
false
true
```

ebob: Öklid algoritması (48, 18)

main() println(ebob(48, 18))

Öklid algoritması en büyük ortak böleni (EBOB) bulur: ebob(a, b) = ebob(b, a % b); b = 0 olunca cevap a.

Adım 1 / 11

```
1 static int ebob(int a, int b) {
2     while (b != 0) {
3         int r = a % b;
4         a = b;
5         b = r;
6     }
7     return a;
8 }
▶ System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)



Çerçeve açılır: a = 48, b = 18.


```
▶ static int ebob(int a, int b) {
2     while (b != 0) {
3         int r = a % b;
4         a = b;
5         b = r;
6     }
7     return a;
8 }
9 System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

a = 48 b = 18

ÇAĞRI YIĞINI

ebob(48, 18)

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)



Koşul: 18 != 0 → **true**.


```
1 static int ebob(int a, int b) {
▶       while (b != 0) {
3           int r = a % b;
4           a = b;
5           b = r;
6       }
7       return a;
8   }
9   System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

a = 48 b = 18

ÇAĞRI YIĞINI

ebob(48, 18)

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)



r = 48 % 18 = 12; a = 18, b = 12. (ebob(48, 18) = ebob(18, 12))

Adım 4 / 11

```
1 static int ebob(int a, int b) {
2     while (b != 0) {
▶         int r = a % b;
▶         a = b;
▶         b = r;
6     }
7     return a;
8 }
9 System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

a = 18 b = 12 r = 12

ÇAĞRI YIĞINI

ebob(48, 18)

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)



Koşul: 12 != 0 → **true**.

Adım 5 / 11

```
1 static int ebob(int a, int b) {
▶       while (b != 0) {
3           int r = a % b;
4           a = b;
5           b = r;
6       }
7       return a;
8   }
9   System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

a = 18 b = 12

ÇAĞRI YIĞINI

ebob(48, 18)

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)



r = 18 % 12 = 6; a = 12, b = 6. (ebob(18, 12) = ebob(12, 6))

Adım 6 / 11

```
1 static int ebob(int a, int b) {
2     while (b != 0) {
▶         int r = a % b;
▶         a = b;
▶         b = r;
6     }
7     return a;
8 }
9 System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

a = 12 b = 6 r = 6

ÇAĞRI YIĞINI

ebob(48, 18)

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)



Koşul: 6 != 0 → **true**.

Adım 7 / 11

```
1 static int ebob(int a, int b) {
▶       while (b != 0) {
3           int r = a % b;
4           a = b;
5           b = r;
6       }
7       return a;
8   }
9   System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

a = 12 b = 6

ÇAĞRI YIĞINI

ebob(48, 18)

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)



r = 12 % 6 = 0; a = 6, b = 0. (ebob(12, 6) = ebob(6, 0))

Adım 8 / 11

```
1 static int ebob(int a, int b) {
2     while (b != 0) {
          int r = a % b;
          a = b;
          b = r;
6     }
7     return a;
8 }
9 System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

a = 6 b = 0 r = 0

ÇAĞRI YIĞINI

ebob(48, 18)

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)



Koşul: 0 != 0 → **false**.

Adım 9 / 11

```
1 static int ebob(int a, int b) {
▶       while (b != 0) {
3           int r = a % b;
4           a = b;
5           b = r;
6       }
7       return a;
8   }
9   System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

a = 6 b = 0

ÇAĞRI YIĞINI

ebob(48, 18)

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)



return 6.

Adım 10 / 11

```
1 static int ebob(int a, int b) {
2     while (b != 0) {
3         int r = a % b;
4         a = b;
5         b = r;
6     }
▶     return a;
8 }
9 System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

a = 6 b = 0

ÇAĞRI YIĞINI

ebob(48, 18)

main()

ÇIKTI

ebob: Öklid algoritması (48, 18)

main() println(6)

6 yazılır: 48 = 6 × 8, 18 = 6 × 3. Yalnızca 3 tur; 1'den 18'e tüm sayıları denemekten çok hızlı.

Adım 11 / 11

```
1 static int ebob(int a, int b) {
2     while (b != 0) {
3         int r = a % b;
4         a = b;
5         b = r;
6     }
7     return a;
8 }
▶ System.out.println(ebob(48, 18));  // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

main()

ÇIKTI

```
6
```

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

ekok(12, 18): metot içinden metot çağrısı



main, `ekok(12, 18)` 'i çağırır.

Adım 1 / 7

```
1 static int ekok(int a, int b) {
2     return a / ebob(a, b) * b;
3 }
4 // ebob: Öklid algoritmasıyla EBOB
▶ System.out.println(ekok(12, 18));  // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

ekok(12, 18): metot içinden metot çağrısı



ekok çerçevesi: a = 12, b = 18.

Adım 2 / 7

```
▶ static int ekok(int a, int b) {
2     return a / ebob(a, b) * b;
3 }
4 // ebob: Öklid algoritmasıyla EBOB
5 System.out.println(ekok(12, 18));  // main
```

DEĞİŞKENLER

a = 12 b = 18

ÇAĞRI YIĞINI

ekok(12, 18)

main()

ÇIKTI

ekok(12, 18): metot içinden metot çağrısı



İfadeyi hesaplamak için ekok, **ebob(a, b)** metodunu çağırır: yığında 3 çerçeve.

Adım 3 / 7

```
1 static int ekok(int a, int b) {
▶     return a / ebob(a, b) * b;
3 }
4 // ebob: Öklid algoritmasıyla EBOB
5 System.out.println(ekok(12, 18));  // main
```

DEĞİŞKENLER

a = 12 b = 18

ÇAĞRI YIĞINI

- ebob(12, 18)
- ekok(12, 18)
- main()

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

ekok(12, 18): metot içinden metot çağrısı



ebob kendi döngüsünü çalıştırır: (12, 18) → (18, 12) → (12, 6) → (6, 0) → return **6**. Adları aynı olsa da ebob'un a, b'si ayrı kutulardır.

Adım 4 / 7

```
1 static int ekok(int a, int b) {
▶     return a / ebob(a, b) * b;
3 }
4 // ebob: Öklid algoritmasıyla EBOB
5 System.out.println(ekok(12, 18));  // main
```

DEĞİŞKENLER

`a = 6` `b = 0`

ÇAĞRI YIĞINI

- `ebob(12, 18)`
- `ekok(12, 18)`
- `main()`

ÇIKTI

ekok(12, 18): metot içinden metot çağrısı



Çağrının yerine 6 geçer: 12 / 6 * 18 = 2 * 18 = **36**. Önce bölmek, a * b'nin taşmasını önler.

Adım 5 / 7

```
1 static int ekok(int a, int b) {
▶     return a / ebob(a, b) * b;
3 }
4 // ebob: Öklid algoritmasıyla EBOB
5 System.out.println(ekok(12, 18));  // main
```

DEĞİŞKENLER

`a = 12` `b = 18`

ÇAĞRI YIĞINI

`ekok(12, 18)`

`main()`

ÇIKTI

ekok(12, 18): metot içinden metot çağrısı



return 36: ekok'un çerçevesi kapanır.

Adım 6 / 7

```
1 static int ekok(int a, int b) {
▶     return a / ebob(a, b) * b;
3 }
4 // ebob: Öklid algoritmasıyla EBOB
5 System.out.println(ekok(12, 18));  // main
```

DEĞİŞKENLER

a = 12 b = 18

ÇAĞRI YIĞINI

ekok(12, 18)

main()

ÇIKTI

ekok(12, 18): metot içinden metot çağrısı

main() println(36)

36 yazılır. Metot başka metodu çağırarak işini kısaltır: kod tekrarı yok.

Adım 7 / 7

```
1 static int ekok(int a, int b) {
2     return a / ebob(a, b) * b;
3 }
4 // ebob: Öklid algoritmasıyla EBOB
▶ System.out.println(ekok(12, 18));  // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

main()

ÇIKTI

```
36
```

Yeniden kullanım: `enBuyuk3(4, 9, 7)`



main, üç sayının en büyüğünü ister.

Adım 1 / 9

```
1 static int enBuyuk(int a, int b) {
2     if (a > b) return a;
3     return b;
4 }
5 static int enBuyuk3(int a, int b, int c) {
6     return enBuyuk(enBuyuk(a, b), c);
7 }
▶ System.out.println(enBuyuk3(4, 9, 7));  // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Yeniden kullanım: `enBuyuk3(4, 9, 7)`



enBuyuk3 çerçevesi: a = 4, b = 9, c = 7.

Adım 2 / 9

```
1 static int enBuyuk(int a, int b) {
2     if (a > b) return a;
3     return b;
4 }
▶ static int enBuyuk3(int a, int b, int c) {
6     return enBuyuk(enBuyuk(a, b), c);
7 }
8 System.out.println(enBuyuk3(4, 9, 7));  // main
```

DEĞİŞKENLER

`a = 4` `b = 9` `c = 7`

ÇAĞRI YIĞINI

`enBuyuk3(4, 9, 7)`

`main()`

ÇIKTI

Yeniden kullanım: `enBuyuk3(4, 9, 7)`



İç içe çağrıda önce **içteki** hesaplanır: `enBuyuk(4, 9)`.

Adım 3 / 9

```
1 static int enBuyuk(int a, int b) {
2     if (a > b) return a;
3     return b;
4 }
5 static int enBuyuk3(int a, int b, int c) {
▶     return enBuyuk(enBuyuk(a, b), c);
7 }
8 System.out.println(enBuyuk3(4, 9, 7));  // main
```

DEĞİŞKENLER

`a = 4` `b = 9` `c = 7`

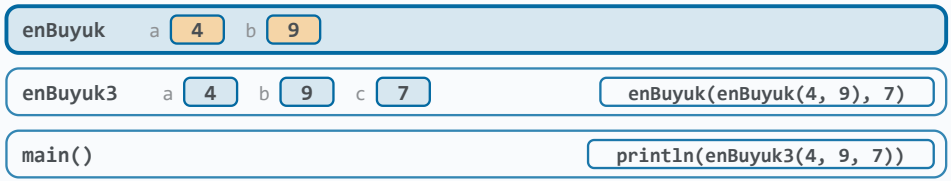
ÇAĞRI YIĞINI

`enBuyuk3(4, 9, 7)`

`main()`

ÇIKTI

Yeniden kullanım: `enBuyuk3(4, 9, 7)`



Yeni çerçeve (a = 4, b = 9): 4 > 9 → false.

Adım 4 / 9

```
▶ static int enBuyuk(int a, int b) {
▶     if (a > b) return a;
3     return b;
4 }
5 static int enBuyuk3(int a, int b, int c) {
6     return enBuyuk(enBuyuk(a, b), c);
7 }
8 System.out.println(enBuyuk3(4, 9, 7));  // main
```

DEĞİŞKENLER

`a = 4` `b = 9`

ÇAĞRI YIĞINI

- `enBuyuk(4, 9)`
- `enBuyuk3(4, 9, 7)`
- `main()`

ÇIKTI

Yeniden kullanım: `enBuyuk3(4, 9, 7)`



return b → 9.

Adım 5 / 9

```
1 static int enBuyuk(int a, int b) {
2     if (a > b) return a;
▶     return b;
4 }
5 static int enBuyuk3(int a, int b, int c) {
6     return enBuyuk(enBuyuk(a, b), c);
7 }
8 System.out.println(enBuyuk3(4, 9, 7));  // main
```

DEĞİŞKENLER

`a = 4` `b = 9`

ÇAĞRI YIĞINI

- `enBuyuk(4, 9)`
- `enBuyuk3(4, 9, 7)`
- `main()`

ÇIKTI

Yeniden kullanım: `enBuyuk3(4, 9, 7)`



İfade sadeleşti: `enBuyuk(9, 7)`. Aynı metot ikinci kez çağrılır.

Adım 6 / 9

```
1 static int enBuyuk(int a, int b) {
2     if (a > b) return a;
3     return b;
4 }
5 static int enBuyuk3(int a, int b, int c) {
▶     return enBuyuk(enBuyuk(a, b), c);
7 }
8 System.out.println(enBuyuk3(4, 9, 7));  // main
```

DEĞİŞKENLER

`a = 4` `b = 9` `c = 7`

ÇAĞRI YIĞINI

`enBuyuk3(4, 9, 7)`

`main()`

ÇIKTI

Yeniden kullanım: `enBuyuk3(4, 9, 7)`



Yeni çerçeve (a = 9, b = 7): 9 > 7 → true → **return a** → 9.

Adım 7 / 9

```
1 static int enBuyuk(int a, int b) {
▶     if (a > b) return a;
3     return b;
4 }
5 static int enBuyuk3(int a, int b, int c) {
6     return enBuyuk(enBuyuk(a, b), c);
7 }
8 System.out.println(enBuyuk3(4, 9, 7));  // main
```

DEĞİŞKENLER

`a = 9` `b = 7`

ÇAĞRI YIĞINI

- `enBuyuk(9, 7)`
- `enBuyuk3(4, 9, 7)`
- `main()`

ÇIKTI

Yeniden kullanım: `enBuyuk3(4, 9, 7)`



enBuyuk3 de 9 döndürür.

Adım 8 / 9

```
1 static int enBuyuk(int a, int b) {
2     if (a > b) return a;
3     return b;
4 }
5 static int enBuyuk3(int a, int b, int c) {
▶     return enBuyuk(enBuyuk(a, b), c);
7 }
8 System.out.println(enBuyuk3(4, 9, 7));  // main
```

DEĞİŞKENLER

`a = 4` `b = 9` `c = 7`

ÇAĞRI YIĞINI

`enBuyuk3(4, 9, 7)`

`main()`

ÇIKTI

Yeniden kullanım: `enBuyuk3(4, 9, 7)`



9 yazılır. Karşılaştırma mantığı tek yerde (enBuyuk): hata varsa tek yerde düzeltilir.

Adım 9 / 9

```
1 static int enBuyuk(int a, int b) {
2     if (a > b) return a;
3     return b;
4 }
5 static int enBuyuk3(int a, int b, int c) {
6     return enBuyuk(enBuyuk(a, b), c);
7 }
▶ System.out.println(enBuyuk3(4, 9, 7));  // main
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
9
```

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

double döndüren metotlar: sıcaklık dönüşümü

main() f = fahrenheit(25)

`fahrenheit(25)` : int argüman 25, double parametreye **25.0** olarak genişletilir.

Adım 1 / 8

```
1 static double fahrenheit(double c) {
2     return c * 9 / 5 + 32;
3 }
4 static double celsius(double f) {
5     return (f - 32) * 5 / 9;
6 }
▶ double f = fahrenheit(25);          // main
8 System.out.println(f);
9 System.out.println(celsius(f));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

main()

ÇIKTI

double döndüren metotlar: sıcaklık dönüşümü



c = 25.0.

Adım 2 / 8

```
▶ static double fahrenheit(double c) {
2     return c * 9 / 5 + 32;
3 }
4 static double celsius(double f) {
5     return (f - 32) * 5 / 9;
6 }
7 double f = fahrenheit(25);          // main
8 System.out.println(f);
9 System.out.println(celsius(f));
```

DEĞİŞKENLER

c = 25.0

ÇAĞRI YIĞINI

fahrenheit(25.0)

main()

ÇIKTI

double döndüren metotlar: sıcaklık dönüşümü



25.0 × 9 = 225.0; / 5 = 45.0; + 32 = **77.0** → return.

Adım 3 / 8

```
1 static double fahrenheit(double c) {
▶     return c * 9 / 5 + 32;
3 }
4 static double celsius(double f) {
5     return (f - 32) * 5 / 9;
6 }
7 double f = fahrenheit(25);          // main
8 System.out.println(f);
9 System.out.println(celsius(f));
```

DEĞİŞKENLER

`c = 25.0`

ÇAĞRI YIĞINI

`fahrenheit(25.0)`

`main()`

ÇIKTI

double döndüren metotlar: sıcaklık dönüşümü



f = 77.0 (double).

Adım 4 / 8

```
1 static double fahrenheit(double c) {
2     return c * 9 / 5 + 32;
3 }
4 static double celsius(double f) {
5     return (f - 32) * 5 / 9;
6 }
▶ double f = fahrenheit(25);          // main
8 System.out.println(f);
9 System.out.println(celsius(f));
```

DEĞİŞKENLER

`f = 77.0`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

double döndüren metotlar: sıcaklık dönüşümü



77.0 yazılır (double olduğu için .0 ile).

Adım 5 / 8

```
1 static double fahrenheit(double c) {
2     return c * 9 / 5 + 32;
3 }
4 static double celsius(double f) {
5     return (f - 32) * 5 / 9;
6 }
7 double f = fahrenheit(25);          // main
▶ System.out.println(f);
9 System.out.println(celsius(f));
```

DEĞİŞKENLER

f = 77.0

ÇAĞRI YIĞINI

main()

ÇIKTI

```
77.0
```

double döndüren metotlar: sıcaklık dönüşümü



`celsius(f)` : f'nin değeri (77.0) kopyalanır.

Adım 6 / 8

```
1 static double fahrenheit(double c) {
2     return c * 9 / 5 + 32;
3 }
4 static double celsius(double f) {
5     return (f - 32) * 5 / 9;
6 }
7 double f = fahrenheit(25);          // main
8 System.out.println(f);
▶ System.out.println(celsius(f));
```

DEĞİŞKENLER

`f = 77.0`

ÇAĞRI YIĞINI

`celsius(77.0)`

`main()`

ÇIKTI

```
77.0
```

double döndüren metotlar: sıcaklık dönüşümü



(77.0 − 32) × 5 / 9 = 45.0 × 5 / 9 = **25.0**. `5 / 9 * (f - 32)` yazılsaydı 5 / 9 tam sayı bölmesiyle **0** olurdu!

Adım 7 / 8

```
1 static double fahrenheit(double c) {
2     return c * 9 / 5 + 32;
3 }
4 static double celsius(double f) {
▶     return (f - 32) * 5 / 9;
6 }
7 double f = fahrenheit(25);          // main
8 System.out.println(f);
9 System.out.println(celsius(f));
```

DEĞİŞKENLER

`f = 77.0`

ÇAĞRI YIĞINI

`celsius(77.0)`

`main()`

ÇIKTI

```
77.0
```

double döndüren metotlar: sıcaklık dönüşümü



25.0 yazılır: iki metot birbirinin tersi (test için güzel bir kontrol).

Adım 8 / 8

```
1 static double fahrenheit(double c) {
2     return c * 9 / 5 + 32;
3 }
4 static double celsius(double f) {
5     return (f - 32) * 5 / 9;
6 }
7 double f = fahrenheit(25);          // main
8 System.out.println(f);
▶ System.out.println(celsius(f));
```

DEĞİŞKENLER

f = 77.0

ÇAĞRI YIĞINI

main()

ÇIKTI

```
77.0
25.0
```

ÖRNEK

Saf Metot ve Yan Etki

- Aynı girdi için aynı sonucu döndüren ve dış durumu değiştirmeyen metot daha kolay test edilir.
- Hesaplama metodu ile ekrana yazma metodu ayrılabilir.
- Gerektiğinde yan etki açıkça adlandırılmalı ve sınırlandırılmalıdır.

Saf

```
static int kare(int x) { return x * x; }
```

kare(4) her zaman 16; her yerde kullanılabilir.

Yan etkili

```
static void kareYaz(int x) { System.out.println(x * x); }
```

Yalnızca ekrana yazar; sonuç başka hesapta kullanılamaz.

ÖZET

Bölüm Özeti

- Metot bir görevi adlandırır, parametrelerle girdi alır ve sonuç döndürebilir.
- Java'da argümanlar her zaman değer ile aktarılır.
- Aşırı yüklemede aynı ad farklı parametre listeleriyle kullanılır.
- Kapsam, değişkenin nerede erişilebilir olduğunu belirler.
- Metot başka metodu çağırabilir (ekok → ebob, enBuyuk3 → enBuyuk); her çağrı yığında kendi çerçevesini açar.
- Sonucu ekrana yazmak yerine döndüren metot daha kolay test edilir ve yeniden kullanılır.

ÖZET

Kontrol Soruları

1. Parametre ile argüman arasındaki fark nedir?
2. void metot hangi durumda kullanılır?
3. `artir(sayi)` çağrısından sonra `sayi` neden değişmez?
4. Aşırı yüklemede hangi bölüm farklı olmalıdır? `int f(int)` ile `double f(int)` neden birlikte olamaz?
5. `ciftMi` neden ekrana yazmak yerine boolean döndürür?

ÖZET

Alıştırmalar

1. `static boolean artikYil(int yil)` yazın (4/100/400 kuralı) ve 1900, 2000, 2024 ile deneyin.
2. `static long faktoriyel(int n)` ve bunu kullanan `static long kombinasyon(int n, int r)` = n! / (r! (n − r)!) yazın.
3. `static int tersCevir(int sayi)` ve onu çağıran `static boolean palindromMu(int sayi)` yazın.
4. `static double ortalama(int a, int b, int c)` : neden `(a + b + c) / 3` yanlış sonuç verebilir? Düzeltin.
5. Aşırı yükleme: `enBuyuk(double, double)` ekleyin; `enBuyuk(3, 4.5)` hangisini çağırır?
6. `asalMi` ile 1..100 arasındaki ikiz asalları (5–7, 11–13 …) listeleyen bir `main` yazın.

ÖZET

Kaynaklar

- Deitel ve Deitel, Java: How to Program, Early Objects, 11. baskı, Pearson, 2018.
- Liang, Introduction to Java Programming and Data Structures, 13. baskı, Pearson, 2024.
- Oracle, The Java Tutorials: **Defining Methods**, **Passing Information to a Method**.
- Oracle, Java Language Specification SE 21: **Bölüm 8, Classes** (§8.4 Method Declarations).
- Sedgewick ve Wayne, Computer Science: An Interdisciplinary Approach, Addison-Wesley, 2016: **2.1 Static Methods**.
- Downey ve Mayfield, Think Java, 2. baskı, O'Reilly, 2019 (**ücretsiz çevrim içi sürüm**).

Son

Metotlar

Sonraki sunum: Diziler · Çalışma soruları · Simülatör: Metot Çağrı Yığını · Terim sözlüğü