Bölüm 3.2: Fonksiyonlar

Java ile Nesne Yönelimli Programlama

Tanım ve çağrı | Parametre ve dönüş değeri | Çağrı yığını | Aşırı yükleme | Hazır Java metotları | Kapsam | Soyutlama ve tasarım

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Tanımlamak ve çağırmak

Erişim belirleyici, dönüş tipi, ad ve parametre listesiyle bir metot yazmak; argümanlarla çağırıp dönüş değerini kullanmak.

Çalışmayı izlemek

Çağrı yığınında (call stack) çerçevelerin nasıl açılıp kapandığını, argümanların **değer olarak** kopyalandığını adım adım göstermek.

Aşırı yükleme

Aynı adlı metotları parametre listesiyle ayırmak; derleyicinin hangi sürümü seçtiğini argüman tiplerinden öngörmek.

Hazır metotlar

`System.out` ve `Math` metotlarını doğru kullanmak; Java kütüphanesinin sık kullanılan sınıflarını ve paketlerini tanımak.

Kapsam

Yerel değişkenlerin, sınıf alanlarının ve iç içe blokların kapsamını ve yaşam süresini açıklamak.

Tasarım

Soyutlama, genelleme, DRY ve tek sorumluluk ilkeleriyle iyi metot tasarlamak; problemi metotlara bölmek.

FONKSİYON KAVRAMI

Fonksiyon Nedir?

Fonksiyon (function), bir işi yapan, **adı olan** ve tekrar tekrar kullanılabilen kod parçasıdır. Java'da fonksiyonlar her zaman bir sınıfın içinde yazılır ve **metot** (method) adını alır.

- Aldığı **parametreler** ve **dönüş değeri tipi** ile tanımlanır.
- Kodun okunabilirliğini artırır, kod tekrarını azaltır.
- Modüler ve düzenli kod yazmayı teşvik eder.
- Programı küçük, yönetilebilir parçalara ayırır.

Benzetme: yemek tarifi

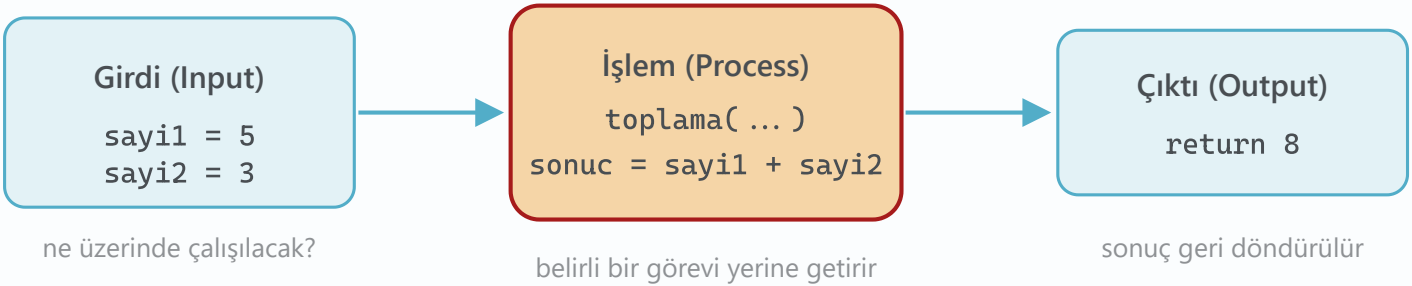
"Çay demle" bir tariftir: bir kez yazılır, istendiği kadar uygulanır. Malzemeler **parametre**, ortaya çıkan çay **dönüş değeri**dir. Tarifi kullanan, her adımı yeniden düşünmek zorunda kalmaz.

Şimdiye kadar yazdığımız `main` da bir metottur; `System.out.println(...)` ise hazır bir metot **çağrısıdır**.

FONKSİYON KAVRAMI

Girdi – İşlem – Çıktı

Her fonksiyon üç aşamada düşünülebilir:



Girdi
Fonksiyona veri sağlanır: **parametreler**. Fonksiyonun ne üzerinde çalışacağını belirler.

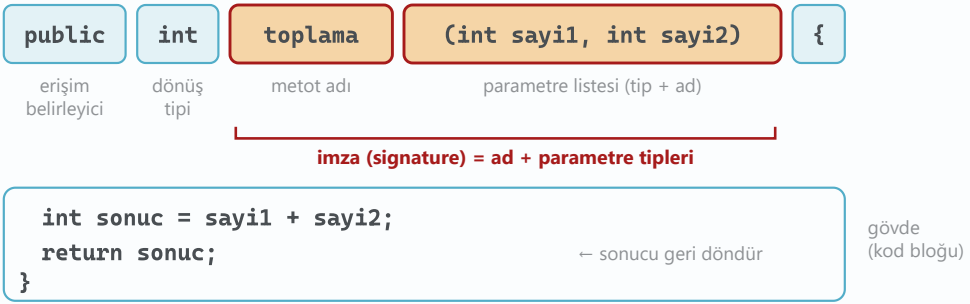
İşlem
Fonksiyon, gövdesindeki deyimlerle girdiyi işler ve belirli bir görevi yerine getirir.

Çıktı
İşlem sonunda bir sonuç üretilir ve `return` ile çağırana **geri döndürülür**.

Matematikteki $f(x) = x^2$ gibi: x girdi, kare alma işlem, x^2 çıktıdır. Bazı fonksiyonların girdisi ya da çıktısı olmayabilir (`void`).

TANIM VE ÇAĞRI

Fonksiyon Tanımı



`public`	**Erişim belirleyici** (access modifier): metodu kimlerin çağırabileceğini söyler.
`int`	**Dönüş değeri tipi**: metodun geri vereceği değerin tipi; değer döndürmüyorsa `void`.
`toplama`	**Metot adı**: fiil ya da fiil öbeği, küçük harfle başlar (camelCase).
`(int sayi1, int sayi2)`	**Parametreler**: metodun girdileri; her birinin tipi ve adı yazılır.
`{ ... }`	**Kod bloğu** (gövde): metot çağrıldığında çalışacak deyimler.

toplama metodu, sayi1 ve sayi2 adlı iki girdi parametresi alır, bunları toplar ve sonucu geri döndürür.

TANIM VE ÇAĞRI

Fonksiyon Çağırma

```java
public class ToplamaOrnegi {
    public static int toplama(int sayi1, int sayi2) {
        int sonuc = sayi1 + sayi2;
        return sonuc;
    }

    public static void main(String[] args) {
        int sonuc = toplama(5, 3);
        System.out.println("Toplam: " + sonuc);
    }
}
```

ÇIKTI

```
Toplam: 8
```

- `toplama(5, 3)` bir **çağrıdır** (call): program akışı metoda gider, metot bitince çağrının yapıldığı yere geri döner.
- Çağrıdaki `5` ve `3` **argümandır** (argument); tanımdaki `sayi1`, `sayi2` ise **parametredir** (parameter).
- Çağrı **8** değerini döndürür; bu değer `sonuc` değişkenine atanır.
- Argüman sayısı, sırası ve tipleri parametrelerle uyuşmalıdır.
- `static` : metot nesne oluşturmadan, doğrudan `main` 'den çağrılabilir.

Çağrı ve Dönüş: Adım Adım

```
1 static int toplama(int sayi1, int sayi2) {
2   int sonuc = sayi1 + sayi2;
3   return sonuc;
4 }
5
▶ public static void main(String[] args) {
7   int sonuc = toplama(5, 3);
8   System.out.println("Toplam: " + sonuc);
9 }
```

Yığın (stack)



Program her zaman `main` metodundan başlar. Yığında yalnızca `main` çerçevesi vardır.

Adım 1 / 6

ÇIKTI

ÇAĞRI YIĞINI

`main()`

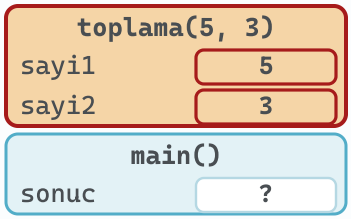
DEĞİŞKENLER

`args = []`

Çağrı ve Dönüş: Adım Adım

```
1 static int toplama(int sayi1, int sayi2) {
2   int sonuc = sayi1 + sayi2;
3   return sonuc;
4 }
5
6 public static void main(String[] args) {
▶   int sonuc = toplama(5, 3);
8   System.out.println("Toplam: " + sonuc);
9 }
```

Yığın (stack)



`toplama(5, 3)` çağrılır: yeni bir **çerçeve** (frame) açılır, argümanlar parametrelere **kopyalanır**.

Adım 2 / 6

ÇIKTI

DEĞİŞKENLER

`sayi1 = 5` `sayi2 = 3`

ÇAĞRI YIĞINI

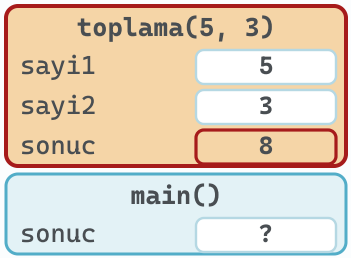
`toplama(5, 3)`

`main()`

Çağrı ve Dönüş: Adım Adım

```
1 static int toplama(int sayi1, int sayi2) {
▶   int sonuc = sayi1 + sayi2;
3   return sonuc;
4 }
5
6 public static void main(String[] args) {
7   int sonuc = toplama(5, 3);
8   System.out.println("Toplam: " + sonuc);
9 }
```

Yığın (stack)



Gövde çalışır: `sonuc = 5 + 3 = 8`. Bu `sonuc`, toplama'nın **yerel** değişkenidir.

Adım 3 / 6

ÇIKTI

DEĞİŞKENLER

`sayi1 = 5` `sayi2 = 3` `sonuc = 8`

ÇAĞRI YIĞINI

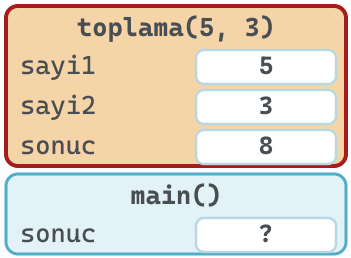
`toplama(5, 3)`

`main()`

Çağrı ve Dönüş: Adım Adım

```
1 static int toplama(int sayi1, int sayi2) {
2   int sonuc = sayi1 + sayi2;
▶   return sonuc;
4 }
5
6 public static void main(String[] args) {
7   int sonuc = toplama(5, 3);
8   System.out.println("Toplam: " + sonuc);
9 }
```

Yığın (stack)



`return sonuc;` 8 değerini çağırana gönderir ve metodu bitirir; çerçeve yığından **silinir**.

Adım 4 / 6

ÇIKTI

DEĞİŞKENLER

`sayi1 = 5` `sayi2 = 3` `sonuc = 8`

`dönüş = 8`

ÇAĞRI YIĞINI

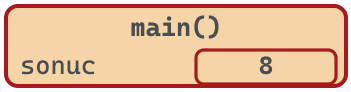
`toplama(5, 3)`

`main()`

Çağrı ve Dönüş: Adım Adım

```
1 static int toplama(int sayi1, int sayi2) {
2   int sonuc = sayi1 + sayi2;
3   return sonuc;
4 }
5
6 public static void main(String[] args) {
▶   int sonuc = toplama(5, 3);
8   System.out.println("Toplam: " + sonuc);
9 }
```

Yığın (stack)



Akış çağrının yapıldığı satıra döner: dönen 8, `main` 'in `sonuc` değişkenine atanır. İki `sonuc` farklı değişkenlerdir.

Adım 5 / 6

ÇIKTI

ÇAĞRI YIĞINI

`main()`

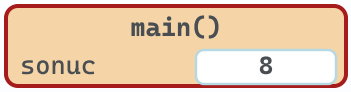
DEĞİŞKENLER

`args = []` `sonuc = 8`

Çağrı ve Dönüş: Adım Adım

```
1 static int toplama(int sayi1, int sayi2) {
2   int sonuc = sayi1 + sayi2;
3   return sonuc;
4 }
5
6 public static void main(String[] args) {
7   int sonuc = toplama(5, 3);
▶   System.out.println("Toplam: " + sonuc);
9 }
```

Yığın (stack)



`println` de bir metot çağrısıdır: ekrana **Toplam: 8** yazılır.

Adım 6 / 6

ÇIKTI

```
Toplam: 8
```

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`args = []` `sonuc = 8`

TANIM VE ÇAĞRI

Örnek Fonksiyonlar

```java
public class OrnekFonksiyonlar {
    // Toplama fonksiyonu
    public static int toplama(int sayi1, int sayi2) {
        return sayi1 + sayi2;
    }
    // Kare hesaplama fonksiyonu
    public static double kareHesapla(double sayi) {
        return sayi * sayi;
    }
    // Merhaba dünya fonksiyonu
    public static void merhabaDunya() {
        System.out.println("Merhaba, Dünya!");
    }

    public static void main(String[] args) {
        System.out.println(toplama(2, 7));
        System.out.println(kareHesapla(1.5));
        merhabaDunya();
    }
}
```

ÇIKTI

```
9
2.25
Merhaba, Dünya!
```

Metot	Girdi	Çıktı
`toplama`	2 × int	int
`kareHesapla`	1 × double	double
`merhabaDunya`	yok	yok (void)

Değer döndüren metodun sonucu bir ifadede kullanılır; `void` metot ise tek başına bir deyim olarak çağrılır.

TANIM VE ÇAĞRI

return ve void

- `return ifade;` değeri çağırana verir ve metodu **hemen bitirir**; sonraki satırlar çalışmaz.
- Dönen değerin tipi, bildirilen dönüş tipiyle uyumlu olmalıdır (`int` → `double` genişlemesi serbesttir).
- Değer döndüren metotta **her yol** bir `return` ile bitmelidir; yoksa derleyici `missing return statement` der.
- `void` metot değer döndürmez; erken çıkmak için tek başına `return;` yazılabilir.

```
public class Erken {
    static void bolumYazdir(int bolunen, int bolen) {
        if (bolen == 0) {
            System.out.println("Sıfıra bölünemez!");
            return;                        // metottan hemen çık
        }
        System.out.println(bolunen / bolen);
    }
    public static void main(String[] args) {
        bolumYazdir(10, 2);
        bolumYazdir(7, 0);
    }
}
```

ÇIKTI

```
5
Sıfıra bölünemez!
```

TANIM VE ÇAĞRI

static ve Nesne Üzerinden Çağrı

static metot

```
static int toplama(int sayi1, int sayi2) {
    return sayi1 + sayi2;
}
// main içinde, doğrudan:
int sonuc = toplama(5, 3);
```

`static` metot sınıfa aittir; nesne gerekmez. Başka sınıftan `SinifAdi.metot()` biçiminde çağrılır: `Math.sqrt(16)` gibi.

nesne (örnek) metodu

```
public class Hesap {
    int toplama(int sayi1, int sayi2) {
        return sayi1 + sayi2;
    }
    public static void main(String[] args) {
        Hesap hesap = new Hesap();
        int sonuc = hesap.toplama(5, 3);
        System.out.println(sonuc);  // 8
    }
}
```

`static` olmayan metot bir **nesne** üzerinden çağrılır: önce `new` ile nesne oluşturulur, sonra `hesap.toplama(5, 3)`.

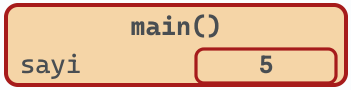
Tanımda `static` yokken `main`'den doğrudan `toplama(5, 3)` yazmak derleme hatasıdır:
`non-static method toplama(int,int) cannot be referenced from a static context`

Değer ile Aktarım (Pass by Value)

Metot, kendisine verilen değişkeni **değiştirebilir mi?**

```
1  static void arttir(int sayi) {
2    sayi = sayi + 10;
3    System.out.println("arttir: " + sayi);
4  }
5
6  public static void main(String[] args) {
▶    int sayi = 5;
8    arttir(sayi);
9    System.out.println("main: " + sayi);
10 }
```

Yığın (stack)



`main` 'de `sayi = 5` .

Adım 1 / 5

ÇIKTI

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

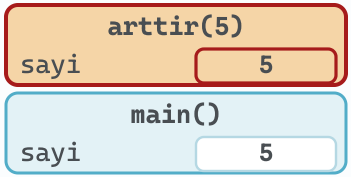
`sayi = 5`

Değer ile Aktarım (Pass by Value)

Metot, kendisine verilen değişkeni **değiştirebilir mi?**

```
1  static void arttir(int sayi) {
2    sayi = sayi + 10;
3    System.out.println("arttir: " + sayi);
4  }
5
6  public static void main(String[] args) {
7    int sayi = 5;
▶    arttir(sayi);
9    System.out.println("main: " + sayi);
10 }
```

Yığın (stack)



Çağrıda **değerin kopyası** (5) gönderilir. arttir'in `sayi` 'si ayrı bir kutudur; adlarının aynı olması önemsizdir.

Adım 2 / 5

ÇIKTI

DEĞİŞKENLER

`sayi = 5`

ÇAĞRI YIĞINI

`arttir(5)`

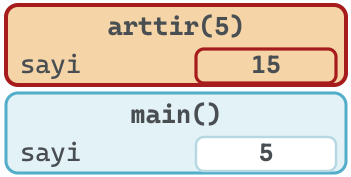
`main()`

Değer ile Aktarım (Pass by Value)

Metot, kendisine verilen değişkeni **değiştirebilir mi?**

```
1  static void arttir(int sayi) {
2    sayi = sayi + 10;
3    System.out.println("arttir: " + sayi);
4  }
5
6  public static void main(String[] args) {
7    int sayi = 5;
8    arttir(sayi);
9    System.out.println("main: " + sayi);
10 }
```

Yığın (stack)



Yalnızca **kopya** değişir: arttir'deki `sayi` 15 olur, main'deki hâlâ 5.

Adım 3 / 5

ÇIKTI

DEĞİŞKENLER

`sayi = 15`

ÇAĞRI YIĞINI

`arttir(5)`

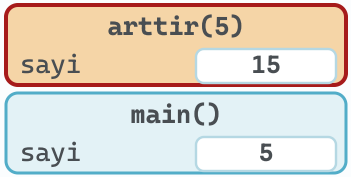
`main()`

Değer ile Aktarım (Pass by Value)

Metot, kendisine verilen değişkeni **değiştirebilir mi?**

```
1  static void arttir(int sayi) {
2    sayi = sayi + 10;
▶    System.out.println("arttir: " + sayi);
4  }
5
6  public static void main(String[] args) {
7    int sayi = 5;
8    arttir(sayi);
9    System.out.println("main: " + sayi);
10 }
```

Yığın (stack)



arttir içinde 15 yazılır.

Adım 4 / 5

ÇIKTI

```
arttir: 15
```

DEĞİŞKENLER

`sayi = 15`

ÇAĞRI YIĞINI

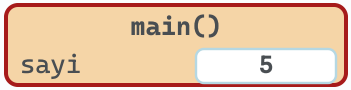
- **arttir(5)**
- main()

Değer ile Aktarım (Pass by Value)

Metot, kendisine verilen değişkeni **değiştirebilir mi?**

```
1  static void arttir(int sayi) {
2    sayi = sayi + 10;
3    System.out.println("arttir: " + sayi);
4  }
5
6  public static void main(String[] args) {
7    int sayi = 5;
8    arttir(sayi);
▶    System.out.println("main: " + sayi);
10 }
```

Yığın (stack)



arttir bitti, çerçevesi ve kopyası silindi. main'deki `sayi` **5** kaldı: Java argümanları **değer ile aktarır** (pass by value).

Adım 5 / 5

ÇIKTI

```
arttir: 15
main: 5
```

ÇAĞRI YIĞINI

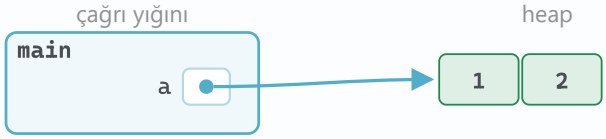
`main()`

DEĞİŞKENLER

`sayi = 5`

Referansın Kopyası: Kodu Adım Adım İzle

```
1  static void degistir(int[] d, int x) {
2      d[0] = 50;
3      x = 50;
4      d = new int[2];
5      d[1] = 50;
6  }
7
8  public static void main(String[] args) {
▶      int[] a = {1, 2};
10     int x = 1;
11     degistir(a, x);
12     System.out.println(a[0] + " " + a[1] +
   " " + x);
13 }
```



`a` bir **referans**: dizi nesnesi heap'te durur, `a` onun adresini tutar.

Adım 1 / 9

ÇIKTI

ÇAĞRI YIĞINI

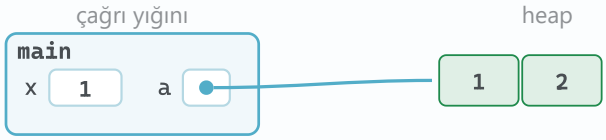
`main()`

DEĞİŞKENLER

—

Referansın Kopyası: Kodu Adım Adım İzle

```
1  static void degistir(int[] d, int x) {
2      d[0] = 50;
3      x = 50;
4      d = new int[2];
5      d[1] = 50;
6  }
7
8  public static void main(String[] args) {
9      int[] a = {1, 2};
▶      int x = 1;
11     degistir(a, x);
12     System.out.println(a[0] + " " + a[1] +
   " " + x);
13 }
```



`x = 1` : ilkel değer doğrudan çerçevede saklanır.

Adım 2 / 9

ÇIKTI

ÇAĞRI YIĞINI

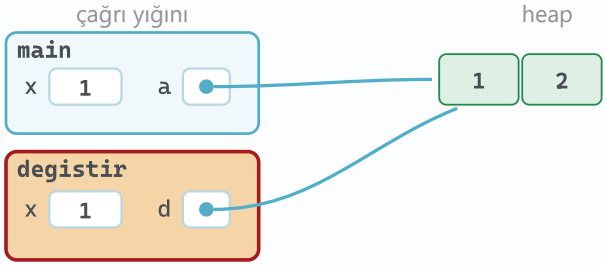
`main()`

DEĞİŞKENLER

`x = 1`

Referansın Kopyası: Kodu Adım Adım İzle

```
▶ static void degistir(int[] d, int x) {
2     d[0] = 50;
3     x = 50;
4     d = new int[2];
5     d[1] = 50;
6 }
7
8 public static void main(String[] args) {
9     int[] a = {1, 2};
10    int x = 1;
11    degistir(a, x);
12    System.out.println(a[0] + " " + a[1] +
  " " + x);
13 }
```



Çağrı: `d` 'ye `a` 'daki **referansın kopyası**, `x` 'e 1'in kopyası verilir. `d` ve `a` aynı diziyi gösterir.

Adım 3 / 9

ÇIKTI

DEĞİŞKENLER

`x = 1`

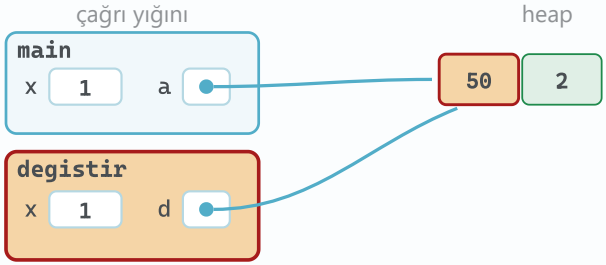
ÇAĞRI YIĞINI

`degistir(a, x)`

`main()`

Referansın Kopyası: Kodu Adım Adım İzle

```
1  static void degistir(int[] d, int x) {
▶      d[0] = 50;
3      x = 50;
4      d = new int[2];
5      d[1] = 50;
6  }
7
8  public static void main(String[] args) {
9      int[] a = {1, 2};
10     int x = 1;
11     degistir(a, x);
12     System.out.println(a[0] + " " + a[1] +
   " " + x);
13 }
```



`d[0] = 50` : ortak dizi değişir; `main` da bu değişikliği görecek.

Adım 4 / 9

ÇIKTI

DEĞİŞKENLER

`x = 1`

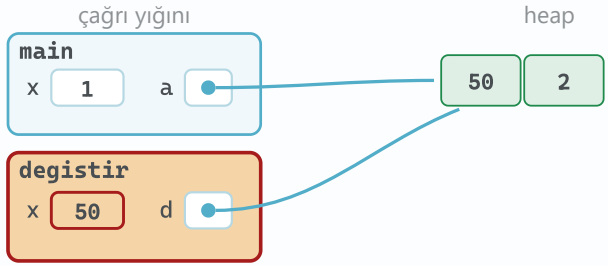
ÇAĞRI YIĞINI

`degistir(a, x)`

`main()`

Referansın Kopyası: Kodu Adım Adım İzle

```
1  static void degistir(int[] d, int x) {
2      d[0] = 50;
▶      x = 50;
4      d = new int[2];
5      d[1] = 50;
6  }
7
8  public static void main(String[] args) {
9      int[] a = {1, 2};
10     int x = 1;
11     degistir(a, x);
12     System.out.println(a[0] + " " + a[1] +
   " " + x);
13 }
```



`x = 50` : yalnızca `degistir` 'in kopyası değişir.

Adım 5 / 9

ÇIKTI

DEĞİŞKENLER

`x = 50`

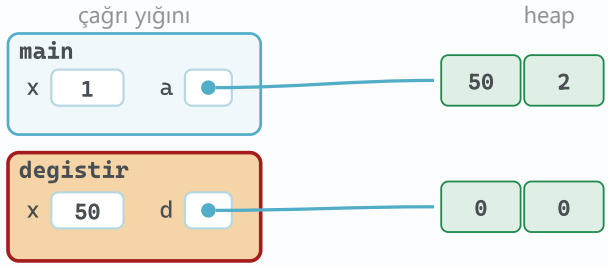
ÇAĞRI YIĞINI

`degistir(a, x)`

`main()`

Referansın Kopyası: Kodu Adım Adım İzle

```
1  static void degistir(int[] d, int x) {
2      d[0] = 50;
3      x = 50;
▶      d = new int[2];
5      d[1] = 50;
6  }
7
8  public static void main(String[] args) {
9      int[] a = {1, 2};
10     int x = 1;
11     degistir(a, x);
12     System.out.println(a[0] + " " + a[1] +
   " " + x);
13 }
```



`d = new int[2]` : `d` artık **yeni** bir diziyi gösterir. `a` etkilenmez; bağ koptu.

Adım 6 / 9

ÇIKTI

DEĞİŞKENLER

`x = 50`

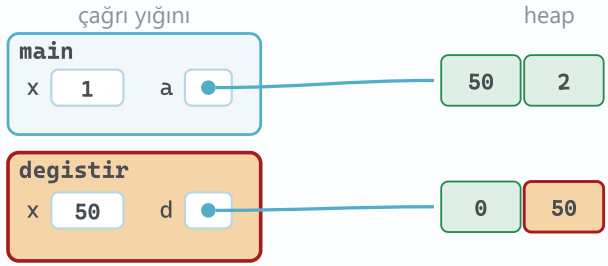
ÇAĞRI YIĞINI

`degistir(a, x)`

`main()`

Referansın Kopyası: Kodu Adım Adım İzle

```
1  static void degistir(int[] d, int x) {
2      d[0] = 50;
3      x = 50;
4      d = new int[2];
▶      d[1] = 50;
6  }
7
8  public static void main(String[] args) {
9      int[] a = {1, 2};
10     int x = 1;
11     degistir(a, x);
12     System.out.println(a[0] + " " + a[1] +
   " " + x);
13 }
```



`d[1] = 50` yeni diziye yazılır; `main` 'in dizisi değişmez.

Adım 7 / 9

ÇIKTI

DEĞİŞKENLER

`x = 50`

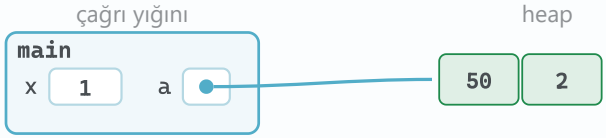
ÇAĞRI YIĞINI

`degistir(a, x)`

`main()`

Referansın Kopyası: Kodu Adım Adım İzle

```
1  static void degistir(int[] d, int x) {
2      d[0] = 50;
3      x = 50;
4      d = new int[2];
5      d[1] = 50;
6  }
7
8  public static void main(String[] args) {
9      int[] a = {1, 2};
10     int x = 1;
▶      degistir(a, x);
12     System.out.println(a[0] + " " + a[1] +
   " " + x);
13 }
```



Metot bitti; çerçevesi silindi. Yeni dizi artık hiçbir referansla gösterilmiyor (çöp toplayıcı temizleyecek).

Adım 8 / 9

ÇIKTI

ÇAĞRI YIĞINI

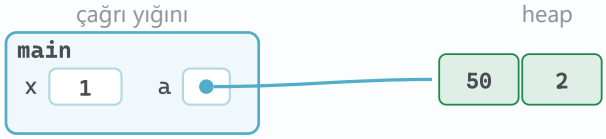
```
main()
```

DEĞİŞKENLER

`x = 1`

Referansın Kopyası: Kodu Adım Adım İzle

```
1  static void degistir(int[] d, int x) {
2      d[0] = 50;
3      x = 50;
4      d = new int[2];
5      d[1] = 50;
6  }
7
8  public static void main(String[] args) {
9      int[] a = {1, 2};
10     int x = 1;
11     degistir(a, x);
▶      System.out.println(a[0] + " " + a[1] +
   " " + x);
13 }
```



`50 2 1` : dizinin içeriği değişti, `x` değişmedi. Java her şeyi değerle aktarır; referans da kopyalanan bir değerdir.

Adım 9 / 9

ÇIKTI

```
50 2 1
```

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`x = 1`

TANIM VE ÇAĞRI

Metot Aşırı Yükleme

```java
public class AsiriYukleme {
    static int topla(int a, int b) {
        return a + b;
    }
    static double topla(double a, double b) {
        return a + b;
    }
    static int topla(int a, int b, int c) {
        return a + b + c;
    }

    public static void main(String[] args) {
        System.out.println(topla(2, 3));
        System.out.println(topla(2.5, 1.2));
        System.out.println(topla(1, 2, 3));
    }
}
```

ÇIKTI

```
5
3.7
6
```

Aşırı yükleme (overloading): aynı adlı birden fazla metot, **parametre listeleri farklı** olduğu sürece aynı sınıfta bulunabilir.

- Farklılık: parametre **sayısı**, **tipi** ya da **sırası**.
- Derleyici, çağrıdaki argümanlara bakarak hangi metodun çalışacağına **derleme zamanında** karar verir.
- Yalnızca dönüş tipi farklı olan iki metot aşırı yükleme **değildir**: `method topla(int,int) is already defined` hatası verir.
- `println` da aşırı yüklenmiştir: `println(int)`, `println(String)`, `println(double)` ...

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
▶      System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

`topla(4, 6)` — **argümanlar: int, int**

- `topla(int, int)` — ✓ tam eşleşme
- `topla(double, double)` — ✓ genişletmeyle uyar
- `topla(int, int, int)` — ✗ sayı tutmuyor

Aynı adlı üç `topla` var. İlk çağrının argümanları `(int, int)` : üç parametreli sürüm elenir; `(int, int)` **tam eşleşme** olduğu için o seçilir.

Adım 1 / 12

ÇIKTI

ÇAĞRI YIĞINI

`main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
▶      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

topla(4, 6) — **argümanlar: int, int**

`topla(int, int)`	`a = 4, b = 6`
`topla(double, double)`	✓ genişletmeyle uyar
`topla(int, int, int)`	✗ sayı tutmuyor

`topla(int, int)` çalışır: `a = 4, b = 6`.

Adım 2 / 12

ÇIKTI

ÇAĞRI YIĞINI

- `topla(int, int)`
- `main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
▶      System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

topla(4, 6) — **argümanlar: int, int**

- **topla(int, int)** → **dönüş: 10**
- **topla(double, double)** ✓ genişletmeyle uyar
- **topla(int, int, int)** ✗ sayı tutmuyor

10 döner ve yazdırılır.

Adım 3 / 12

ÇIKTI

```
10
```

ÇAĞRI YIĞINI

`main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
▶      System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

`topla(4, 2.5)` — **argümanlar: int, double**

- `topla(int, int)` — ✗ 2.5 int olamaz
- `topla(double, double)` — ✓ 4 → 4.0
- `topla(int, int, int)` — ✗ sayı tutmuyor

`(int, double)`: 2.5 `int`'e örtük dönüşemez, `(int, int)` elenir. 4, 4.0'a genişletilir → `topla(double, double)`.

Adım 4 / 12

ÇIKTI

```
10
```

ÇAĞRI YIĞINI

`main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
▶      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

topla(4, 2.5) — **argümanlar: int, double**

- `topla(int, int)` — ✗ 2.5 int olamaz
- **`topla(double, double)`** — `a = 4.0, b = 2.5`
- `topla(int, int, int)` — ✗ sayı tutmuyor

`a = 4.0, b = 2.5` : parametreler `double`.

Adım 5 / 12

ÇIKTI

```
10
```

ÇAĞRI YIĞINI

- **`topla(double, double)`**
- `main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
▶      System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

topla(4, 2.5) — **argümanlar: int, double**

- `topla(int, int)` — ✗ 2.5 int olamaz
- `topla(double, double)` — **dönüş: 6.5**
- `topla(int, int, int)` — ✗ sayı tutmuyor

6.5 döner ve yazdırılır.

Adım 6 / 12

ÇIKTI

```
10
6.5
```

ÇAĞRI YIĞINI

`main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
▶      System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

topla(7, 1, 2) — **argümanlar: int, int, int**

- `topla(int, int)` — ✗ sayı tutmuyor
- `topla(double, double)` — ✗ sayı tutmuyor
- `topla(int, int, int)` — ✓ tam eşleşme

Üç argüman: yalnızca üç parametreli sürüm uyar.

Adım 7 / 12

ÇIKTI

```
10
6.5
```

ÇAĞRI YIĞINI

`main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
▶      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

topla(7, 1, 2) — **argümanlar: int, int, int**

- `topla(int, int)` — ✗ sayı tutmuyor
- `topla(double, double)` — ✗ sayı tutmuyor
- `topla(int, int, int)` — `a = 7, b = 1, c = 2`

`a = 7, b = 1, c = 2` .

Adım 8 / 12

ÇIKTI

```
10
6.5
```

ÇAĞRI YIĞINI

- `topla(int, int, int)`
- `main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
▶      System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

topla(7, 1, 2) — **argümanlar: int, int, int**

- `topla(int, int)` — ✗ sayı tutmuyor
- `topla(double, double)` — ✗ sayı tutmuyor
- `topla(int, int, int)` — **dönüş: 10**

10 döner ve yazdırılır.

Adım 9 / 12

ÇIKTI

```
10
6.5
10
```

ÇAĞRI YIĞINI

`main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
▶      System.out.println(topla('A', 1));
15 }
```

`topla('A', 1)` — **argümanlar: char, int**

`topla(int, int)`	✓ 'A' → 65
`topla(double, double)`	✓ genişletmeyle uyar
`topla(int, int, int)`	✗ sayı tutmuyor

`(char, int)` : `char` hem `int` 'e hem `double` 'a genişleyebilir; derleyici **en özel** olanı, `(int, int)` 'i seçer.

Adım 10 / 12

ÇIKTI

```
10
6.5
10
```

ÇAĞRI YIĞINI

`main()`

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
14     System.out.println(topla('A', 1));
15 }
```

topla('A', 1) — **argümanlar: char, int**

topla(int, int)	a = 65, b = 1
topla(double, double)	✓ genişletmeyle uyar
topla(int, int, int)	✗ sayı tutmuyor

'A' → 65: a = 65, b = 1 .

Adım 11 / 12

ÇIKTI

```
10
6.5
10
```

ÇAĞRI YIĞINI

- topla(int, int)
- main()

Hangi topla Çalışır? Adım Adım

```
1  static int topla(int a, int b) {
2      return a + b;
3  }
4  static double topla(double a, double b) {
5      return a + b;
6  }
7  static int topla(int a, int b, int c) {
8      return a + b + c;
9  }
10 public static void main(String[] args) {
11     System.out.println(topla(4, 6));
12     System.out.println(topla(4, 2.5));
13     System.out.println(topla(7, 1, 2));
▶      System.out.println(topla('A', 1));
15 }
```

topla('A', 1) — **argümanlar: char, int**

- **topla(int, int)** — **dönüş: 66**
- **topla(double, double)** — ✓ genişletmeyle uyar
- **topla(int, int, int)** — ✗ sayı tutmuyor

66 döner ve yazdırılır. Seçim derleme zamanında, yalnızca argüman tiplerine bakılarak yapılır.

Adım 12 / 12

ÇIKTI

```
10
6.5
10
66
```

ÇAĞRI YIĞINI

`main()`

Fonksiyon İçinde Fonksiyon

```java
public class IcIce {
    public static int kareAl(int sayi) {
        return sayi * sayi;
    }
    public static int topla(int sayi1, int sayi2) {
        return sayi1 + sayi2;
    }
    public static int ikiKareToplami(int sayi1, int sayi2) {
        return topla(kareAl(sayi1), kareAl(sayi2));
    }
    public static void main(String[] args) {
        int sonuc = ikiKareToplami(3, 4);
        System.out.println(sonuc);
    }
}
```

ÇIKTI

```
25
```

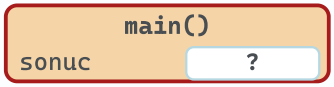
- Bir fonksiyon, **başka bir fonksiyonu çağırabilir**.
- Karmaşık görev küçük parçalara bölünür: `ikiKareToplami`, işi `kareAl` ve `topla`'ya yaptırır.
- Kod okunabilir ve yönetilebilir olur; her parça ayrı ayrı test edilebilir.
- `topla(kareAl(sayi1), kareAl(sayi2))` ifadesinde önce **argümanlar** hesaplanır (soldan sağa), sonra dış çağrı yapılır.

$3^2 + 4^2 = 9 + 16 = \mathbf{25}$.

ikiKareToplami(3, 4): Çağrı Yığını

```
1  static int kareAl(int sayi) {
2    return sayi * sayi;
3  }
4  static int topla(int sayi1, int sayi2) {
5    return sayi1 + sayi2;
6  }
7  static int ikiKareToplami(int sayi1,
8                            int sayi2) {
9    return topla(kareAl(sayi1), kareAl(sayi2));
10 }
11 public static void main(String[] args) {
▶    int sonuc = ikiKareToplami(3, 4);
13   System.out.println(sonuc);
14 }
```

Yığın (stack)



`main` , `ikiKareToplami(3, 4)` çağrısını yapar.

Adım 1 / 9

ÇIKTI

ÇAĞRI YIĞINI

`main()`

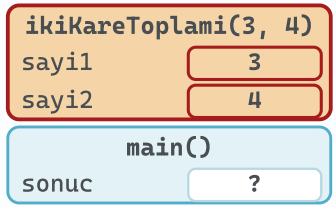
DEĞİŞKENLER

`sonuc = ?`

ikiKareToplami(3, 4): Çağrı Yığını

```
1  static int kareAl(int sayi) {
2    return sayi * sayi;
3  }
4  static int topla(int sayi1, int sayi2) {
5    return sayi1 + sayi2;
6  }
7  static int ikiKareToplami(int sayi1,
8                            int sayi2) {
▶    return topla(kareAl(sayi1), kareAl(sayi2));
10 }
11 public static void main(String[] args) {
12   int sonuc = ikiKareToplami(3, 4);
13   System.out.println(sonuc);
14 }
```

Yığın (stack)



`topla` çağrılmadan önce argümanları hesaplanmalı: önce `kareAl(sayi1)`.

Adım 2 / 9

ÇIKTI

DEĞİŞKENLER

`sayi1 = 3` `sayi2 = 4`

ÇAĞRI YIĞINI

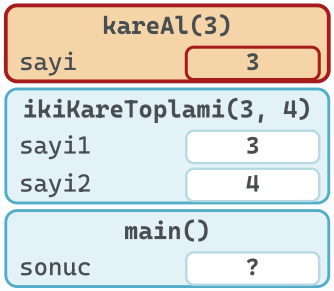
`ikiKareToplami(3, 4)`

`main()`

ikiKareToplami(3, 4): Çağrı Yığını

```
1  static int kareAl(int sayi) {
▶    return sayi * sayi;
3  }
4  static int topla(int sayi1, int sayi2) {
5    return sayi1 + sayi2;
6  }
7  static int ikiKareToplami(int sayi1,
8                            int sayi2) {
9    return topla(kareAl(sayi1), kareAl(sayi2));
10 }
11 public static void main(String[] args) {
12   int sonuc = ikiKareToplami(3, 4);
13   System.out.println(sonuc);
14 }
```

Yığın (stack)



Üçüncü çerçeve açılır: `kareAl(3)` → 3 × 3 = **9** döner.

Adım 3 / 9

ÇIKTI

DEĞİŞKENLER

`sayi = 3` `dönüş = 9`

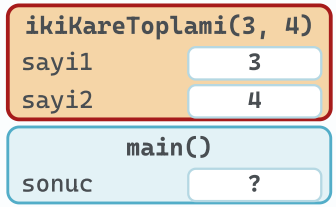
ÇAĞRI YIĞINI

- `kareAl(3)`
- `ikiKareToplami(3, 4)`
- `main()`

ikiKareToplami(3, 4): Çağrı Yığını

```
1  static int kareAl(int sayi) {
2    return sayi * sayi;
3  }
4  static int topla(int sayi1, int sayi2) {
5    return sayi1 + sayi2;
6  }
7  static int ikiKareToplami(int sayi1,
8                            int sayi2) {
▶    return topla(kareAl(sayi1), kareAl(sayi2));
10 }
11 public static void main(String[] args) {
12   int sonuc = ikiKareToplami(3, 4);
13   System.out.println(sonuc);
14 }
```

Yığın (stack)



kareAl çerçevesi kapandı; ilk argüman (`arg1`) = 9. Sıradaki: `kareAl(sayi2)` .

Adım 4 / 9

ÇIKTI

DEĞİŞKENLER

`sayi1 = 3` `sayi2 = 4` `arg1 = 9`

ÇAĞRI YIĞINI

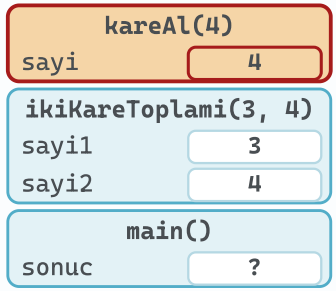
`ikiKareToplami(3, 4)`

`main()`

ikiKareToplami(3, 4): Çağrı Yığını

```
1  static int kareAl(int sayi) {
2    return sayi * sayi;
3  }
4  static int topla(int sayi1, int sayi2) {
5    return sayi1 + sayi2;
6  }
7  static int ikiKareToplami(int sayi1,
8                            int sayi2) {
9    return topla(kareAl(sayi1), kareAl(sayi2));
10 }
11 public static void main(String[] args) {
12   int sonuc = ikiKareToplami(3, 4);
13   System.out.println(sonuc);
14 }
```

Yığın (stack)



`kareAl(4)` → 4 × 4 = **16** döner.

Adım 5 / 9

ÇIKTI

DEĞİŞKENLER

`sayi = 4` `dönüş = 16`

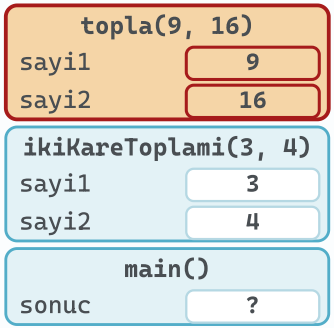
ÇAĞRI YIĞINI

- `kareAl(4)`
- `ikiKareToplami(3, 4)`
- `main()`

ikiKareToplami(3, 4): Çağrı Yığını

```
 1 static int kareAl(int sayi) {
 2   return sayi * sayi;
 3 }
 4 static int topla(int sayi1, int sayi2) {
 5   return sayi1 + sayi2;
 6 }
 7 static int ikiKareToplami(int sayi1,
 8                            int sayi2) {
 9   return topla(kareAl(sayi1), kareAl(sayi2));
10 }
11 public static void main(String[] args) {
12   int sonuc = ikiKareToplami(3, 4);
13   System.out.println(sonuc);
14 }
```

Yığın (stack)



Argümanlar hazır: `topla(9, 16)` çağrılır → **25**.
Buradaki `sayi1` = 9; ikiKareToplami'nınki hâlâ 3.

Adım 6 / 9

ÇIKTI

DEĞİŞKENLER

`sayi1 = 9` `sayi2 = 16` `dönüş = 25`

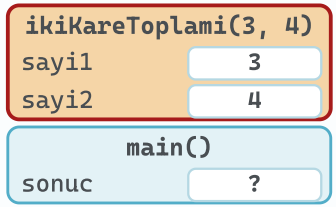
ÇAĞRI YIĞINI

- `topla(9, 16)`
- `ikiKareToplami(3, 4)`
- `main()`

ikiKareToplami(3, 4): Çağrı Yığını

```
1  static int kareAl(int sayi) {
2    return sayi * sayi;
3  }
4  static int topla(int sayi1, int sayi2) {
5    return sayi1 + sayi2;
6  }
7  static int ikiKareToplami(int sayi1,
8                            int sayi2) {
▶    return topla(kareAl(sayi1), kareAl(sayi2));
10 }
11 public static void main(String[] args) {
12   int sonuc = ikiKareToplami(3, 4);
13   System.out.println(sonuc);
14 }
```

Yığın (stack)



topla'nın sonucu, ikiKareToplami'nın dönüş değeri olur: **25** döner.

Adım 7 / 9

ÇIKTI

DEĞİŞKENLER

`sayi1 = 3` `sayi2 = 4` `dönüş = 25`

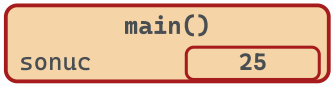
ÇAĞRI YIĞINI

- `ikiKareToplami(3, 4)`
- `main()`

ikiKareToplami(3, 4): Çağrı Yığını

```
1  static int kareAl(int sayi) {
2    return sayi * sayi;
3  }
4  static int topla(int sayi1, int sayi2) {
5    return sayi1 + sayi2;
6  }
7  static int ikiKareToplami(int sayi1,
8                            int sayi2) {
9    return topla(kareAl(sayi1), kareAl(sayi2));
10 }
11 public static void main(String[] args) {
▶    int sonuc = ikiKareToplami(3, 4);
13   System.out.println(sonuc);
14 }
```

Yığın (stack)



main'e dönüldü: `sonuc = 25`.

Adım 8 / 9

ÇIKTI

ÇAĞRI YIĞINI

`main()`

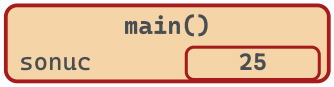
DEĞİŞKENLER

`sonuc = 25`

ikiKareToplami(3, 4): Çağrı Yığını

```
1  static int kareAl(int sayi) {
2    return sayi * sayi;
3  }
4  static int topla(int sayi1, int sayi2) {
5    return sayi1 + sayi2;
6  }
7  static int ikiKareToplami(int sayi1,
8                            int sayi2) {
9    return topla(kareAl(sayi1), kareAl(sayi2));
10 }
11 public static void main(String[] args) {
12   int sonuc = ikiKareToplami(3, 4);
▶    System.out.println(sonuc);
14 }
```

Yığın (stack)



Ekrana **25** yazılır. En fazla 3 çerçeve aynı anda yığındaydı.

Adım 9 / 9

ÇIKTI

```
25
```

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`sonuc = 25`

HAZIR METOTLAR

En Sık Kullanılan Metotlar (Java API)

Alan	Sınıflar	Örnek metotlar	Paket
Ekrana yazma	`System.out`	`print`, `println`, `printf`	java.lang
Matematik	`Math`	`sqrt`, `abs`, `max`, `pow`	java.lang
Metin	`String`	`length`, `charAt`, `substring`, `equals`	java.lang
Dizi	`Arrays`	`sort`, `copyOf`, `binarySearch`, `equals`	java.util
Kullanıcı girişi	`Scanner`	`nextInt`, `nextLine`, `close`	java.util
Dosya	`File`, `FileWriter`	`createNewFile`, `write`, `close`	java.io
Veri tabanı	`DriverManager`, `Statement`	`getConnection`, `executeQuery`, `executeUpdate`	java.sql
Veri yapıları	`ArrayList`	`add`, `remove`, `get`, `size`	java.util
Zaman	`LocalDateTime`, `DateTimeFormatter`	`now`, `ofPattern`, `format`	java.time
Web / grafik	`URI`, `URL`, `JFrame`	`URI.create`, `openConnection`, `setVisible`	java.net, javax.swing

`java.lang` otomatik içe aktarılır; diğerleri için `import` gerekir, ör. `import java.util.Scanner;`

HAZIR METOTLAR

Ekrana Yazdırma

```
public class Yazdirma {
    public static void main(String[] args) {
        System.out.println("Merhaba, Dünya!");
        System.out.print("Bir ");
        System.out.print("satır");
        System.out.println();
        int yas = 20;
        System.out.println("Yaş: " + yas);
        System.out.printf("Pi yaklaşık %.2f%n", Math.PI);
    }
}
```

ÇIKTI

```
Merhaba, Dünya!
Bir satır
Yaş: 20
Pi yaklaşık 3,14
```

- `System.out.println(x)` : x'i yazar, **alt satıra** geçer.
- `System.out.print(x)` : x'i yazar, aynı satırda kalır.
- `+` ile metin ve değer birleştirilir: `"Yaş: " + yas` .
- `printf` : biçimli yazma; `%.2f` 2 ondalık basamak, `%d` tam sayı, `%s` metin, `%n` satır sonu.
- `printf` yerel ayarı kullanır: Türkçe sistemde ondalık ayırıcı **virgüldür** (3,14).

HAZIR METOTLAR

Math Sınıfı

```
double kareKok = Math.sqrt(16);
double mutlakDeger = Math.abs(-17.2);
double maksimum = Math.max(3, 5);
System.out.println(kareKok);
System.out.println(mutlakDeger);
System.out.println(maksimum);
System.out.println(Math.pow(2, 10));
System.out.println(Math.round(2.6));
System.out.println(Math.min(-4, 7));
System.out.println(Math.max(3, 5));
```

ÇIKTI

```
4.0
17.2
5.0
1024.0
3
-4
5
```

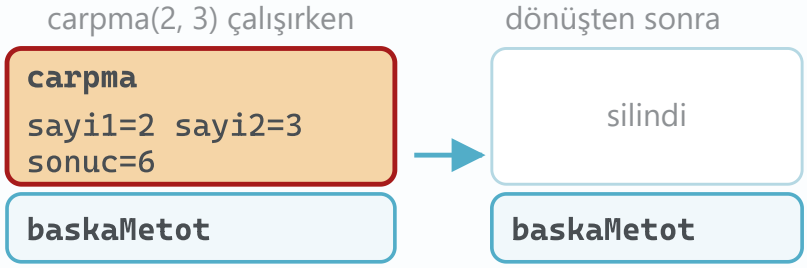
- `Math` sınıfı matematiksel işlemler içindir; metotları `static` olduğundan `Math.sqrt(16)` biçiminde çağrılır.
- `Math.max(3, 5)` `int` döndürür (5); `double` değişkene atanınca **5.0** olur.
- `sqrt` ve `pow` her zaman `double` döndürür.

KAPSAM

Yerel Değişken

```java
public int carpma(int sayi1, int sayi2) {
    int sonuc = sayi1 * sayi2; // 'sonuc' bir yerel değişkendir
    return sonuc;
}
public void baskaMetot() {
    carpma(2, 3);
    System.out.println(sonuc);  // HATA
}
```

```
error: cannot find symbol
        System.out.println(sonuc);
                           ^
  symbol:   variable sonuc
```

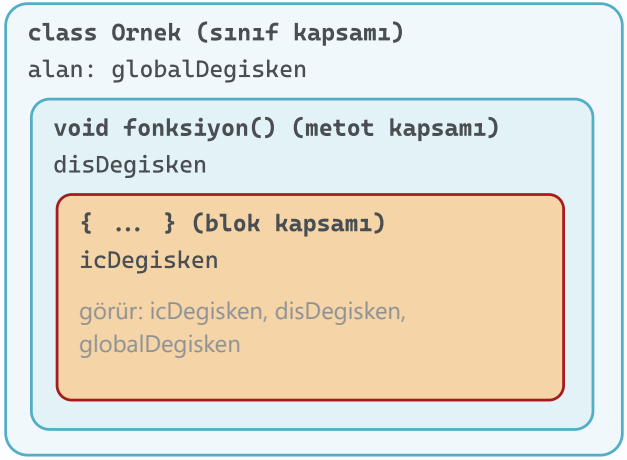


- **Yerel değişken** (local variable): metodun ya da bloğun içinde tanımlanır; parametreler de yereldir.
- Yalnızca fonksiyon çalışırken var olur; **fonksiyon dışından erişilemez**.
- `sonuc`, sadece `carpma` yürütülürken vardır; metot dönünce silinir.
- Değeri dışarı taşımanın yolu `return`: `int s = carpma(2, 3);`

Kapsam

Kapsam (scope), bir değişkenin tanımlandığı ve kullanılabildiği kod bölgesidir.

- Her değişkenin bir kapsamı vardır: Java'da kapsam, değişkenin tanımlandığı **süslü parantez bloğudur**.
- Kapsam, değişkenin **erişilebilirliğini** ve **yaşam süresini** belirler.
- Yerel ve genel (global) kapsamlar farklı erişim kurallarına sahiptir.
- İç içe kapsamda iç kapsam, üst kapsamdaki değişkenlere erişebilir; dış kapsam içtekini göremez.
- Aynı adın farklı kapsamlarda kullanılması **isim çakışmasına** yol açabilir; doğru kullanılmayan kapsam hatalara neden olur.



KAPSAM

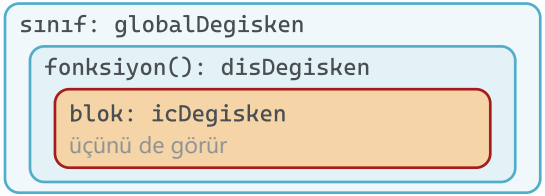
Yerel ve Global Kapsam

```java
public class IcKapsam {
    static int globalDegisken = 10; // global (sınıf alanı)
    static void fonksiyon() {
        int disDegisken = 10;       // yerel kapsam
        {
            int icDegisken = 5;
            // disDegisken ve icDegisken burada erişilebilir
            System.out.println(disDegisken + icDegisken
                    + globalDegisken);
        }
        // icDegisken burada YOK
    }
    public static void main(String[] args) {
        fonksiyon();
    }
}
```

ÇIKTI

```
25
```

10 + 5 + 10 = 25: iç blok, hem metodun hem sınıfın değişkenlerini görür.



Yerel kapsam

Metot ya da blok içinde; yaşam süresi o bloğun çalışma süresi kadar.

Global kapsam

Her yerden erişilir, program boyunca yaşar. Java'da bu rolü sınıf **alanları** üstlenir.

İç içe kapsam

Kapsam içinde kapsam; iç kapsam dıştakinin değişkenlerine erişir.

KAPSAM

İsim Çakışması

Bloğun içinde aynı ad: hata

```
public void fonksiyon() {
    int disDegisken = 10;
    {
        int disDegisken = 5;  // HATA
    }
}
```

```
error: variable disDegisken is already
       defined in method fonksiyon()
```

Java, iç bloğun dıştaki yerel değişkenle aynı adı kullanmasına **izin vermez**.

Yerel değişken alanı gölgeler

```
public class Golge {
    static int deger = 10;  // sınıf alanı
    static void goster() {
        int deger = 5;      // alanı gölgeler
        System.out.println(deger);
        System.out.println(Golge.deger);
    }
    public static void main(String[] args) {
        goster();
        System.out.println(deger);
    }
}
```

ÇIKTI

```
5
10
10
```

Yerel `deger`, aynı adlı alanı **gölgeler** (shadowing); alana `Golge.deger` ile ulaşılır.

KAPSAM · ADIM ADIM KOD İZLEME

Gölgeleme: Kodu Adım Adım İzle

```
1  public class Golge {
2      static int deger = 10;  // sınıf alanı
3      static void goster() {
4          int deger = 5;      // alanı gölgeler
5          System.out.println(deger);
6          System.out.println(Golge.deger);
7      }
8      public static void main(String[] args) {
▶          goster();
10         System.out.println(deger);
11     }
12 }
```

sınıf alanı
`Golge.deger = 10`

`main`

Sınıf alanı `deger = 10` program başlarken oluşur ve program boyunca yaşar. `main`, `goster()`'i çağırır.

Adım 1 / 6

ÇIKTI

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`Golge.deger = 10`

Gölgeleme: Kodu Adım Adım İzle

```
1  public class Golge {
2      static int deger = 10;  // sınıf alanı
▶      static void goster() {
4          int deger = 5;      // alanı gölgeler
5          System.out.println(deger);
6          System.out.println(Golge.deger);
7      }
8      public static void main(String[] args) {
9          goster();
10         System.out.println(deger);
11     }
12 }
```



`goster` için yeni bir çerçeve açılır; henüz yerel değişkeni yok.

Adım 2 / 6

ÇIKTI

DEĞİŞKENLER

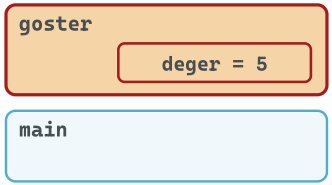
`Golge.deger = 10`

ÇAĞRI YIĞINI

- `goster()`
- `main()`

Gölgeleme: Kodu Adım Adım İzle

```
1  public class Golge {
2      static int deger = 10;  // sınıf alanı
3      static void goster() {
▶          int deger = 5;      // alanı gölgeler
5          System.out.println(deger);
6          System.out.println(Golge.deger);
7      }
8      public static void main(String[] args) {
9          goster();
10         System.out.println(deger);
11     }
12 }
```





Yerel `deger = 5` tanımlanır. Aynı ad artık bu metot içinde yereli gösterir: alan **gölgelendi**.

Adım 3 / 6

ÇIKTI

DEĞİŞKENLER

`Golge.deger = 10` `deger (yerel) = 5`

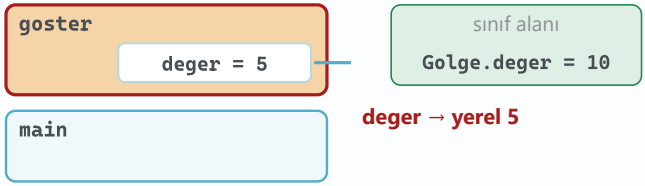
ÇAĞRI YIĞINI

`goster()`

`main()`

Gölgeleme: Kodu Adım Adım İzle

```
1  public class Golge {
2      static int deger = 10;  // sınıf alanı
3      static void goster() {
4          int deger = 5;      // alanı gölgeler
▶          System.out.println(deger);
6          System.out.println(Golge.deger);
7      }
8      public static void main(String[] args) {
9          goster();
10         System.out.println(deger);
11     }
12 }
```



`deger` adı önce en içteki kapsamda aranır → yerel 5 yazdırılır.

Adım 4 / 6

ÇIKTI

```
5
```

DEĞİŞKENLER

`Golge.deger = 10` `deger (yerel) = 5`

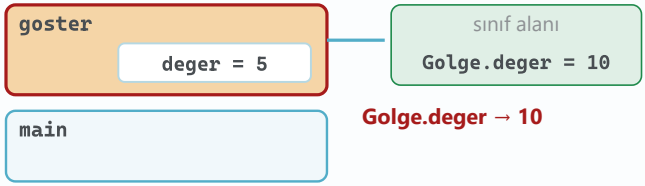
ÇAĞRI YIĞINI

- `goster()`
- `main()`

KAPSAM · ADIM ADIM KOD İZLEME

Gölgeleme: Kodu Adım Adım İzle

```
1  public class Golge {
2      static int deger = 10;  // sınıf alanı
3      static void goster() {
4          int deger = 5;      // alanı gölgeler
5          System.out.println(deger);
▶          System.out.println(Golge.deger);
7      }
8      public static void main(String[] args) {
9          goster();
10         System.out.println(deger);
11     }
12 }
```



`Golge.deger` sınıf adıyla alana ulaşır → 10.

Adım 5 / 6

ÇIKTI

```
5
10
```

ÇAĞRI YIĞINI

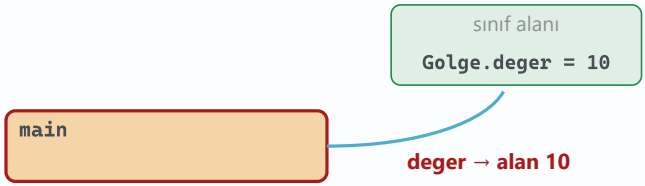
- `goster()`
- `main()`

DEĞİŞKENLER

`Golge.deger = 10` `deger (yerel) = 5`

Gölgeleme: Kodu Adım Adım İzle

```
1  public class Golge {
2      static int deger = 10;  // sınıf alanı
3      static void goster() {
4          int deger = 5;      // alanı gölgeler
5          System.out.println(deger);
6          System.out.println(Golge.deger);
7      }
8      public static void main(String[] args) {
9          goster();
▶          System.out.println(deger);
11     }
12 }
```



`goster` bitti, yerel `deger` silindi. `main` 'de yerel `deger` yok: ad alanı gösterir → 10.

Adım 6 / 6

ÇIKTI

```
5
10
10
```

ÇAĞRI YIĞINI

`main()`

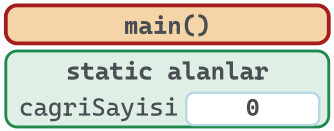
DEĞİŞKENLER

`Golge.deger = 10`

Değişkenler Ne Zaman Doğar, Ne Zaman Ölür?

```
1  static int cagriSayisi = 0;  // sınıf alanı
2  static int carpma(int sayi1, int sayi2) {
3    cagriSayisi++;
4    int sonuc = sayi1 * sayi2;
5    return sonuc;
6  }
7  public static void main(String[] args) {
▶    int ilk = carpma(2, 3);
9    {
10     int ikinci = carpma(ilk, 4);
11     System.out.println(ikinci);
12   }
13   System.out.println(ilk + " " + cagriSayisi);
14 }
```

Bellek



Sınıf alanı `cagriSayisi` program başında oluşur ve program boyunca yaşar. `carpma(2, 3)` çağrılıyor.

Adım 1 / 8

ÇIKTI

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`ilk = ?`

Değişkenler Ne Zaman Doğar, Ne Zaman Ölür?

```
1  static int cagriSayisi = 0;  // sınıf alanı
2  static int carpma(int sayi1, int sayi2) {
▶    cagriSayisi++;
4    int sonuc = sayi1 * sayi2;
5    return sonuc;
6  }
7  public static void main(String[] args) {
8    int ilk = carpma(2, 3);
9    {
10     int ikinci = carpma(ilk, 4);
11     System.out.println(ikinci);
12   }
13   System.out.println(ilk + " " + cagriSayisi);
14 }
```

Bellek



Metot, sınıf alanına erişebilir: `cagriSayisi` 1 olur.

Adım 2 / 8

ÇIKTI

DEĞİŞKENLER

`sayi1 = 2` `sayi2 = 3`

ÇAĞRI YIĞINI

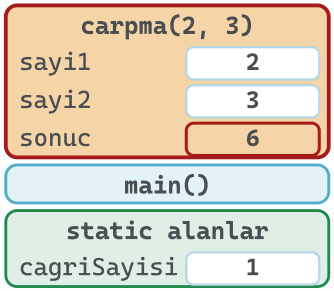
`carpma(2, 3)`

`main()`

Değişkenler Ne Zaman Doğar, Ne Zaman Ölür?

```
1  static int cagriSayisi = 0;  // sınıf alanı
2  static int carpma(int sayi1, int sayi2) {
3    cagriSayisi++;
▶    int sonuc = sayi1 * sayi2;
5    return sonuc;
6  }
7  public static void main(String[] args) {
8    int ilk = carpma(2, 3);
9    {
10     int ikinci = carpma(ilk, 4);
11     System.out.println(ikinci);
12   }
13   System.out.println(ilk + " " + cagriSayisi);
14 }
```

Bellek



Yerel `sonuc` **şimdi** doğar: 6.

Adım 3 / 8

ÇIKTI

DEĞİŞKENLER

`sayi1 = 2` `sayi2 = 3` `sonuc = 6`

ÇAĞRI YIĞINI

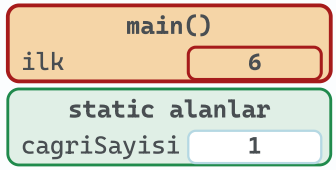
`carpma(2, 3)`

`main()`

Değişkenler Ne Zaman Doğar, Ne Zaman Ölür?

```
1  static int cagriSayisi = 0;  // sınıf alanı
2  static int carpma(int sayi1, int sayi2) {
3    cagriSayisi++;
4    int sonuc = sayi1 * sayi2;
5    return sonuc;
6  }
7  public static void main(String[] args) {
▶    int ilk = carpma(2, 3);
9    {
10     int ikinci = carpma(ilk, 4);
11     System.out.println(ikinci);
12   }
13   System.out.println(ilk + " " + cagriSayisi);
14 }
```

Bellek



carpma döndü: `sayi1`, `sayi2`, `sonuc` **yok oldu**. Değer, `ilk` 'e atandı.

Adım 4 / 8

ÇIKTI

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`ilk = 6`

Değişkenler Ne Zaman Doğar, Ne Zaman Ölür?

```
1  static int cagriSayisi = 0;  // sınıf alanı
2  static int carpma(int sayi1, int sayi2) {
3    cagriSayisi++;
▶    int sonuc = sayi1 * sayi2;
5    return sonuc;
6  }
7  public static void main(String[] args) {
8    int ilk = carpma(2, 3);
9    {
10     int ikinci = carpma(ilk, 4);
11     System.out.println(ikinci);
12   }
13   System.out.println(ilk + " " + cagriSayisi);
14 }
```

Bellek



Yeni çerçeve: `sonuc` = 24, sayaç 2.

Adım 5 / 8

ÇIKTI

DEĞİŞKENLER

`sayi1 = 6` `sayi2 = 4` `sonuc = 24`

ÇAĞRI YIĞINI

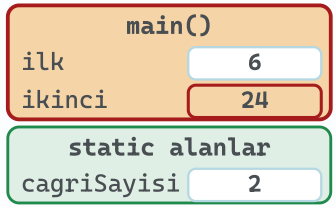
`carpma(6, 4)`

`main()`

Değişkenler Ne Zaman Doğar, Ne Zaman Ölür?

```
1  static int cagriSayisi = 0;  // sınıf alanı
2  static int carpma(int sayi1, int sayi2) {
3    cagriSayisi++;
4    int sonuc = sayi1 * sayi2;
5    return sonuc;
6  }
7  public static void main(String[] args) {
8    int ilk = carpma(2, 3);
9    {
▶      int ikinci = carpma(ilk, 4);
11     System.out.println(ikinci);
12   }
13   System.out.println(ilk + " " + cagriSayisi);
14 }
```

Bellek



`ikinci` yalnızca iç blokta yaşayan bir yerel değişkendir.

Adım 6 / 8

ÇIKTI

ÇAĞRI YIĞINI

`main()`

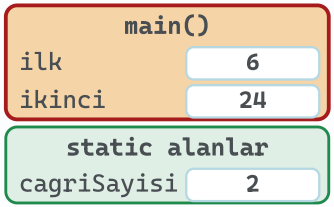
DEĞİŞKENLER

`ilk = 6` `ikinci = 24`

Değişkenler Ne Zaman Doğar, Ne Zaman Ölür?

```
1  static int cagriSayisi = 0;  // sınıf alanı
2  static int carpma(int sayi1, int sayi2) {
3    cagriSayisi++;
4    int sonuc = sayi1 * sayi2;
5    return sonuc;
6  }
7  public static void main(String[] args) {
8    int ilk = carpma(2, 3);
9    {
10     int ikinci = carpma(ilk, 4);
▶      System.out.println(ikinci);
12   }
13   System.out.println(ilk + " " + cagriSayisi);
14 }
```

Bellek



Blok içinde `ikinci` erişilebilir: **24** yazılır.

Adım 7 / 8

ÇIKTI

```
24
```

ÇAĞRI YIĞINI

`main()`

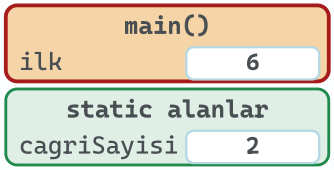
DEĞİŞKENLER

`ilk = 6` `ikinci = 24`

Değişkenler Ne Zaman Doğar, Ne Zaman Ölür?

```
1  static int cagriSayisi = 0;  // sınıf alanı
2  static int carpma(int sayi1, int sayi2) {
3    cagriSayisi++;
4    int sonuc = sayi1 * sayi2;
5    return sonuc;
6  }
7  public static void main(String[] args) {
8    int ilk = carpma(2, 3);
9    {
10     int ikinci = carpma(ilk, 4);
11     System.out.println(ikinci);
12   }
▶  System.out.println(ilk + " " + cagriSayisi);
14 }
```

Bellek



Blok bitti, `ikinci` yok oldu: **6 2**.

Adım 8 / 8

ÇIKTI

```
24
6 2
```

ÇAĞRI YIĞINI

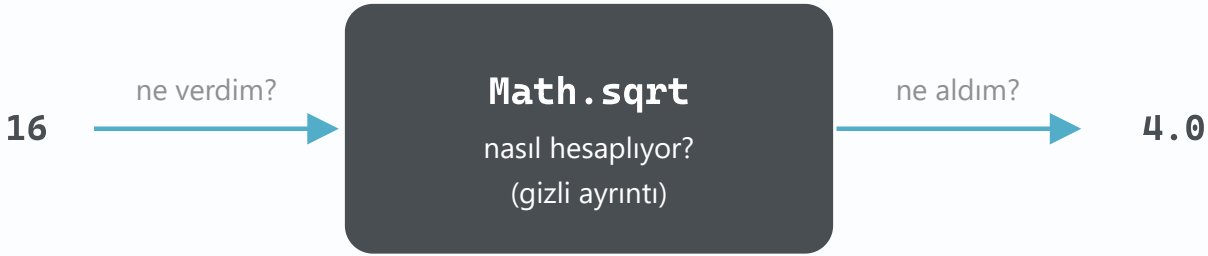
`main()`

DEĞİŞKENLER

`ilk = 6`

Soyutlama

Soyutlama (abstraction), karmaşıklığı azaltmak için **ayrıntıları yok saymaktır**.



Neden?

Kodun okunabilirliğini artırır, işlemi basitleştirir; büyük ve karmaşık programları anlaşılır kılar.

Nasıl?

Kodu parçalara ayırır; her parça ayrı ayrı düşünülür. Bir metodu kullanan, onun içini bilmek zorunda değildir.

Günlük benzetme: arabayı sürmek için motorun içini bilmek gerekmez; direksiyon ve pedallar bir "arayüzdür".

Fonksiyonel Soyutlama

Hesaplamalar üzerinde soyutlama, **işlevsel** ya da **yordamsal soyutlama** (procedural abstraction) olarak da bilinir.

- Bir fonksiyonun **nasıl** hesapladığından çok **ne** hesapladığını bilmek önemlidir.
- Görevi bilindiği sürece, nasıl yaptığı umursanmaz.
- Fonksiyonun gerçeklemesi (gövdesi) hakkında endişelenilmez.
- Gövde değişse bile (daha hızlı bir algoritma), çağıran kod aynı kalır.

```
System.out.println("Merhaba");
```

Fonksiyon, verilen metni konsola yazar; bunu nasıl yaptığını (karakter kodlama, işletim sistemi çağrıları...) bilmek gerekmez.

```
// Kullanan yalnızca imzayı bilir:
static boolean asalMi(int sayi)
// "sayi asal ise true döner"
if (asalMi(17)) { ... }
```



kullanıcı yalnızca girdiyi ve çıktıyı bilir

Genelleme

Bağlam (context):

- bir sayının, bir denklem içindeki diğer unsurlarla ilişkisi,
- bir kelimenin anlamının cümledeki yeri ve durumuyla ilişkisi,
- bir kod bloğu içindeki durum ve etkileşimler.

Bir ifadenin anlamı, çevresindeki diğer kelimelerden etkilenebilir.

Genelleme (generalization):

- bir bağlama ait bir çözümü başka bağlamlara uygun hâle getirmek,
- bir ürünü daha genel ve çok amaçlı kullanılabilir yapmak.

Kodda genellemenin aracı **parametredir**: sabit değerler parametreye dönüşür.

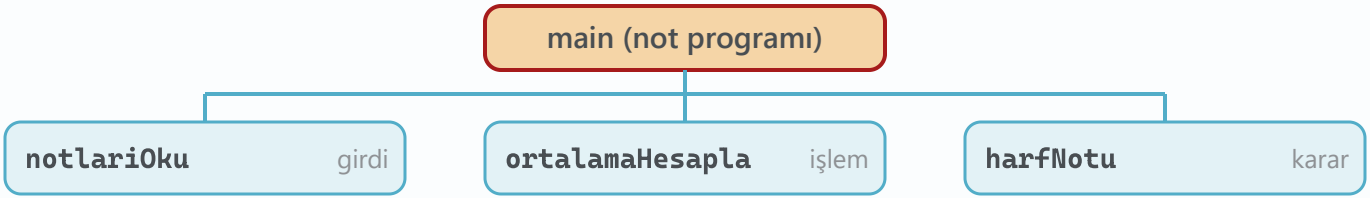
```java
public class Genelleme {
    // Özel: yalnızca 10 tane '-' yazar
    static void cizgiCiz() {
        System.out.println("----------");
    }
    // Genel: uzunluğu ve karakteri parametre
    static void cizgiCiz(int uzunluk, char karakter) {
        for (int i = 0; i < uzunluk; i++) {
            System.out.print(karakter);
        }
        System.out.println();
    }
    public static void main(String[] args) {
        cizgiCiz();
        cizgiCiz(5, '*');
        cizgiCiz(12, '=');
    }
}
```

ÇIKTI

```
----------
*****
============
```

Ayrıştırma

Problemi küçük parçalara ayırmak (decomposition), programlamanın temel adımıdır. Her parça bir fonksiyon olur: kod düzenli, okunur, **yeniden kullanılabilir** ve sürdürülebilir olur.




```java
// notlariOku ve harfNotu da ayrı metotlardır
static double ortalamaHesapla(int[] notlar) {
    int toplam = 0;
    for (int n : notlar) {
        toplam += n;
    }
    return (double) toplam / notlar.length;
}
public static void main(String[] args) {
    Scanner giris = new Scanner(System.in);
    int[] notlar = notlariOku(giris, 3);
    double ort = ortalamaHesapla(notlar);
    System.out.println("Ortalama: " + ort + " -> " + harfNotu(ort));
}
```

ÇIKTI (GİRDİ: 80 90 75)

```
Ortalama:
81.66666666666667 -> BB
```

`main` bir "içindekiler" gibi okunur: ne yapıldığı görünür, nasıl yapıldığı alt fonksiyonlardadır.

SOYUTLAMA VE TASARIM

Ne Zaman Fonksiyon?

Bir kural: **DRY** (*Don't Repeat Yourself*, kendini tekrar etme). Kodu kopyalayıp yapıştırma ihtiyacı doğduğu an, o kod bir fonksiyon olmaya adaydır.

Tekrarlı kod

```
double ort1 = (70 + 80 + 90) / 3.0;
System.out.println(ort1);
double ort2 = (55 + 60 + 100) / 3.0;
System.out.println(ort2);
// formül değişirse her yerde
// ayrı ayrı düzeltmek gerekir
```

Fonksiyonla

```
static double ortalama(int a, int b, int c) {
    return (a + b + c) / 3.0;
}
System.out.println(ortalama(70, 80, 90));
System.out.println(ortalama(55, 60, 100));
```

ÇIKTI

```
80.0
71.66666666666667
```

Tekrar kullanılabilirlik ve bakım kolaylığı için önemlidir. İyi bir fonksiyon **tek bir işi** yapar ve okunaklıdır.

SOYUTLAMA VE TASARIM

İyi Fonksiyon Tasarımı

Tek sorumluluk

Her fonksiyon yalnızca bir işi yapmalı. "ve" ile anlatılıyorsa (oku *ve* hesapla *ve* yazdır) bölünmeli.

Açıklayıcı isim

Ne yaptığını anlatan fiil: `ortalamaHesapla`, `asalMi`, `notlariOku`. `islem2` gibi isimler bir şey anlatmaz.

Parametreler

Gerekli verileri parametrelerle almalı; gizlice dış değişkenlere bağlı olmamalı. Az sayıda parametre okunurluğu artırır.

Dönüş değeri

Sonucu açıkça `return` ile döndürmeli. Hesaplayan metot yazdırmamalı; yazdırma işini çağıran yapsın.

Kısa ve odaklı

Bir ekrana sığan, birkaç satırlık fonksiyonlar anlaşılır ve kolay test edilir.

Yerinde çağrı

Fonksiyon yalnızca gerektiği yerde çağrılmalı; aynı sonucu döngüde tekrar tekrar hesaplamak gereksiz iştir.

Değişken Adlandırma

Her değişken tek bir şeyi temsil etmeli

- Bir değişken yalnızca bir nesneyi ya da bir veriyi tutmalı; aynı değişkenin önce yaş, sonra toplam için kullanılması kafa karıştırır.
- Kodun okunurluğunu ve yeniden kullanılabilirliğini artırır, mantıksal hataları azaltır.

İyi isim

- Açıklayıcı ve anlamlı olmalı; temsil ettiği verinin doğasını yansıtmalı.
- İçeriği ve kullanımı anlatmalı; anlamsız ya da tek harfli isimler okunurluğu düşürür (döngü sayacı `i` ve matematikteki `x` istisnadır).
- Java geleneği: **camelCase** (`ogrenciSayisi`), sınıflar büyük harfle (`Kitap`), sabitler `MAKS_DEGER` .

```
int ogrenciSayisi;      // iyi bir adlandırma
int a;                  // anlamsız bir isim
int yas = 25;
String isim = "Ahmet";
double ortalama = 85.5;
String kitapAdi;
```

Kötü	İyi
`int x1, x2;`	`int vizeNotu, finalNotu;`
`double d;`	`double daireAlani;`
`boolean f;`	`boolean asalMi;`

ÖZET

Özet

Konu	Hatırlanacaklar
Tanım	`erişim [static] dönüşTürü ad(parametreler) { gövde }`; imza = ad + parametre tipleri
Çağrı	argümanlar parametrelere **kopyalanır** (değer ile aktarım); her çağrı yığında yeni bir çerçeve açar
return / void	`return` değeri verir ve metodu bitirir; `void` değer döndürmez
static	static metot doğrudan / `Sinif.metot()`; değilse `nesne.metot()`
Aşırı yükleme	aynı ad, farklı parametre listesi; seçim derleme zamanında
Hazır metotlar	`System.out`, `Math` (`static`: `Math.sqrt(16)`); `java.lang` dışındakiler `import` ister
Kapsam	değişken tanımlandığı blokta yaşar; iç blok dışı görür, tersi değil; alanlar program boyunca
Tasarım	soyutlama (ne, nasıl değil), genelleme (parametre), ayrıştırma, DRY, tek sorumluluk

ÖZET

İlgili Sayfalar

Değer ile Aktarım Laboratuvarı

Simülatör: metoda ilkel değer, dizi ve nesne referansı geçirmek; çağrı yığını çerçeveleri ve heap nesneleri adım adım.

Çalışma Soruları

Asal sayı, kombinasyon, öğrenci numarası çözme gibi problemler; çağrı yığını ve gölgeleme izlemeleri.

ÖZET

Sık Yapılan Hatalar (1)

static olmayan metodu main'den çağırmak

```
non-static method toplama(int,int) cannot be referenced from a static context
```

Çözüm: metot `static` yapılır ya da bir nesne oluşturulup onun üzerinden çağrılır.

Bazı yollarda return unutmak

```
missing return statement
```

if/else if zincirinde her olasılık bir `return` ile bitmeli; sona varsayılan bir `return` konur.

void metodun sonucunu kullanmak

```
incompatible types: void cannot be converted to int
```

`int x = selamla();` olmaz: void metot değer üretmez.

Eksik / fazla argüman

```
method toplama in class ... cannot be applied to given types; required: int,int found: int
```

Argüman sayısı ve tipleri imzayla aynı olmalı.

ÖZET

Sık Yapılan Hatalar (2)

Yerel değişkeni dışarıda kullanmak

`cannot find symbol … variable sonuc`

Değer `return` ile döndürülür.

İç blokta aynı adla değişken

`variable disDegisken is already defined in method fonksiyon()`

Takas metodunun çalışacağını sanmak

`yerDegistir(int x, int y)` yalnızca kopyaları takas eder; `yerDegistir(dizi[0], dizi[2])` sonrası dizi aynı kalır. Dizi hücreleri için `yerDegistir(int[] d, int i, int j)`.

Dönüş tipiyle aşırı yüklemek

`int f(int a)` ve `double f(int a)` birlikte olamaz:

`method f(int) is already defined`

. İmzayı yalnız ad ve parametre tipleri belirler.

Argümanın değişeceğini sanmak

`arttir(sayi)` sonrası `sayi` aynı kalır: kopya değişir. Yeni değer `return` ile döner.

Math sonucunun tipini unutmak

`int k = Math.sqrt(16);` →

`possible lossy conversion from double to int`

. `sqrt` ve `pow` `double` döndürür.

ÖZET

Örnek Problemler (1)

```java
static int f(int x) {
    x = x * 2;
    return x + 1;
}
public static void main(String[] args) {
    int a = 3;
    int b = f(a);
    System.out.println(a + " " + b);
    System.out.println(f(f(1)));
}
```

```java
static int kare(int n) { return n * n; }
static int kup(int n) { return n * kare(n); }
// main içinde:
System.out.println(kup(2) + kare(3));
```

1. Soldaki program ne yazdırır? `a` neden değişmedi?
2. Sağdaki satır ne yazdırır? `kup(2)` çalışırken yığında en çok kaç çerçeve olur (main dahil)?

▶ **Cevapları göster**

ÖZET

Örnek Problemler (2)

```
static int topla(int a, int b) { return a + b; }
static double topla(double a, double b) { return a + b; }
// main içinde:
System.out.println(topla(2, 3.0));
System.out.println(topla(2, 3));
System.out.println(Math.max(Math.abs(-8), Math.sqrt(49)));
```

1. İlk iki `println` hangi `topla` metodunu çağırır, ne yazdırır?
2. Son satır neden `8` değil de `8.0` yazdırır?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. `boolean ciftMi(int sayi)` : sayı çiftse `true` döndürsün.
2. `int enBuyuk(int a, int b, int c)` : üç sayının en büyüğünü `Math.max` ile bulsun.
3. `int usAl(int taban, int us)` : döngüyle üs hesaplasın (`Math.pow` kullanmadan).
4. `int basamakToplami(int n)` : sayının rakamları toplamını döngüyle bulsun.
5. `double daireAlani(double r)` yazın; sonucu `printf` ile 2 basamak yazdırın.
6. Aşırı yüklenmiş `double alan(double r)` (daire) ve `double alan(double en, double boy)` (dikdörtgen).

▶ **Cevaplar / ipuçları**

Kaynaklar

- Oracle, *The Java Tutorials*: "Defining Methods", "Passing Information to a Method or a Constructor".
- J. Gosling ve diğ., *The Java Language Specification, Java SE 21 Edition* (Bölüm 8.4 Method Declarations, 6.3 Scope).
- P. Deitel ve H. Deitel, *Java: How to Program, Early Objects*, 11. baskı, Pearson, 2017 (Bölüm 6: Methods).
- K. Sierra ve B. Bates, *Head First Java*, 3. baskı, O'Reilly, 2022.
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, 2022.
- A. B. Downey ve C. Mayfield, *Think Java*, 2. baskı, O'Reilly, 2019 (Bölüm 4: Methods and Testing).

Son

Bölüm 3.2: Fonksiyonlar

Sonraki bölüm: Bölüm 3.3: Diziler · Terim sözlüğü