Bölüm 1.4: Akış Diyagramları

Java ile Nesne Yönelimli Programlama

Algoritma ve akış diyagramı | Temel semboller | Çizim kuralları | Sıralı akış | Koşullu akış | Tekrarlı akış | Döngü sembolü | Alt program | Diyagramdan Java'ya

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Kavram

Algoritma ile akış diyagramı arasındaki ilişkiyi açıklamak; akış diyagramının neden ve nerede kullanıldığını söylemek.

Semboller

Başla/Bitir, Giriş/Çıkış, İşlem, Karar, Döngü, Yazdır, bağlantı noktası ve alt program sembollerini tanımak ve doğru yerde kullanmak.

Kontrol yapıları

Sıralı, koşullu ve tekrarlı akışı ayırt etmek; her biri için akış diyagramı çizmek.

İzleme

Bir diyagramı verilen girdiyle adım adım izleyip çıktıyı bulmak; sınır hatalarını ve sonsuz döngüleri fark etmek.

Diyagramdan Java'ya

Her sembolün Java karşılığını bilmek: `nextInt()`, atama, `if/else`, `while`, `for`, `println`.

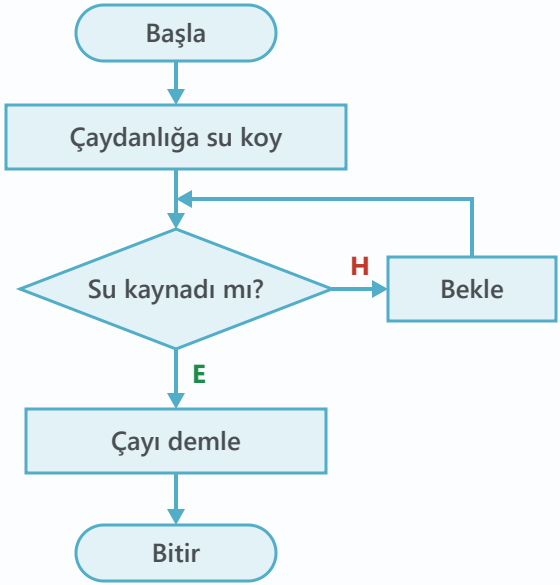
Tasarım

Basit bir problemi (toplam, tek/çift, sayma) önce diyagramla planlayıp sonra programa dönüştürmek.

Akış Diyagramları (Flowcharts)

- **Algoritma (Algorithm):** bir problemi çözmek veya belirli bir görevi yerine getirmek için adım adım talimatlar içeren plan.
- **Akış diyagramı (Flowchart):** algoritmanın adımlarını, adımlar arasındaki bağlantıları ve kontrol mekanizmalarını **görsel** olarak gösterir.
- Her adım bir **sembolle**, adımların sırası **oklarla** (akış çizgileri) çizilir.

Benzetme: Yemek tarifi bir algoritmadır; aynı tarifin kutular ve oklarla çizilmiş hâli akış diyagramıdır. Program yazmadan önce çözümü kâğıt üzerinde planlamanın en kolay yoludur.



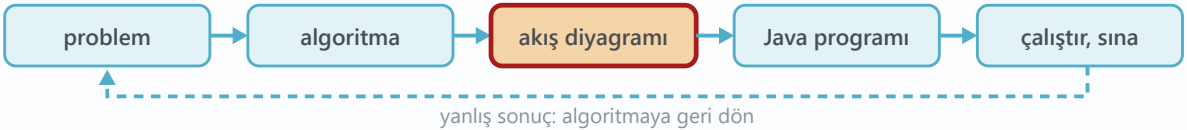
Faydaları ve Sınırlılıkları

Faydaları

- \+ Karmaşık süreçleri **basitleştirir**: çözümün tamamı tek bakışta görülür.
- \+ **Hataları** ve iyileştirme noktalarını kod yazmadan önce belirlemeye yardımcı olur.
- \+ Süreçleri **belgelemede** ve raporlamada kullanılır.
- \+ Programlama dilinden **bağımsızdır**; ekip içinde ortak bir dil sağlar.

Sınırlılıkları

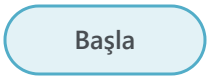
- – Büyük programlarda diyagram çok büyür ve okunması zorlaşır.
- – Algoritma değiştiğinde diyagramın yeniden çizilmesi gerekir.
- – Ayrıntı arttıkça sembol sayısı artar; bu durumda **sözde kod** (pseudocode) daha pratiktir.



Tipik çalışma sırası: **problem** → **algoritma** (adımlar) → **akış diyagramı** → **Java programı** → çalıştırıp sınama.

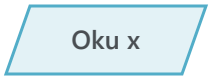
SEMBOLLER VE KURALLAR

Temel Öğeler (Semboller)



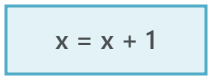
Başlangıç / Bitiş

Oval. Algoritmanın başladığı ve bittiği yer.



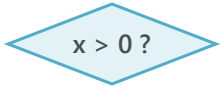
Giriş / Çıkış

Paralelkenar. Klavyeden veri okuma (girdi).



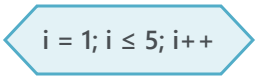
İşlem

Dikdörtgen. Hesaplama, işlem, atama.



Karar

Eşkenar dörtgen. Evet/Hayır sorusu, iki çıkış.



Döngü

Altıgen. Belirli sayıda tekrar.



Yazdır

Dalgalı alt kenar. Sonucu ekrana / kâğıda yazma.



Bağlantı noktası

Küçük çember. Çizgilerin birleştiği nokta.



Akış çizgisi

Ok. Adımların sırası; okun yönünde ilerlenir.

Semboller ISO 5807 standardına dayanır. Bazı kaynaklar giriş ve çıkışın ikisi için de paralelkenar kullanır; ekrana yazma için "Yazdır" sembolü de yaygındır.

SEMBOLLER VE KURALLAR

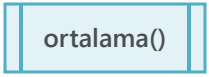
Her Sembolün Java Karşılığı

Sembol	Diyagramdaki yazı	Java karşılığı
 Başla / Bitir	Başla … Bitir	`public static void main(String[] args) { … }`
 Giriş	Oku sayı	`int sayi = giris.nextInt();`
 İşlem	sonuç = sayı1 + sayı2	`int sonuc = sayi1 + sayi2;`
 Karar	sonuç = 0 ?	`if (sonuc == 0) { … } else { … }`
 Karar + geri ok	sayaç ≤ 10 ?	`while (sayac <= 10) { … }`
 Döngü	i = 1; i ≤ 10; i++	`for (int i = 1; i <= 10; i++) { … }`
 Yazdır	Yaz sonuç	`System.out.println(sonuc);`

Diyagramdaki işlem kutusunda `=` **atama**, karar kutusunda `=` **karşılaştırmadır**. Java'da atama `=`, karşılaştırma `==` ile yazılır. Diyagramdaki "mod" Java'da `%` işlecidir.

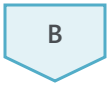
SEMBOLLER VE KURALLAR

Diğer Semboller



Alt program

İki yanı çizgili dikdörtgen. Ayrıca çizilmiş bir adım dizisini çağırır. Java: metot çağrısı.



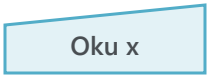
Sayfa dışı bağlantı

Ucu aşağı bakan beşgen. Diyagram başka sayfada sürer; iki uçta aynı harf yazılır.



Veri deposu

Silindir. Dosyaya ya da veri tabanına kaydetme ve oradan okuma.



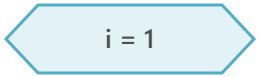
Elle giriş

Üst kenarı eğik dörtgen. Bazı kaynaklar klavyeden girişi bununla çizer.



Açıklama

Açık köşeli parantez. Akışa katılmayan not. Java: `//` yorumu.



Hazırlık

Altıgen. ISO 5807'deki adı hazırlıktır; döngünün başlangıç ayarı için kullanılır.

Temel semboller çoğu diyagram için yeterlidir. Ek semboller uzun diyagramları bölmeye (alt program, sayfa dışı bağlantı) ve verinin nereye yazıldığını göstermeye yarar.

Akış Diyagramı Nasıl Oluşturulur?

Adımlar

1. İşlemler adım adım belirlenir: girdiler, yapılacak hesaplar, çıktılar.
2. Adımlar mantıklı bir sırayla yerleştirilir.
3. Tüm işlemler **akış çizgileri** ile birbirine bağlanır.
4. Koşullu ifadelerle **karar noktaları** belirlenir.

Çizim kuralları

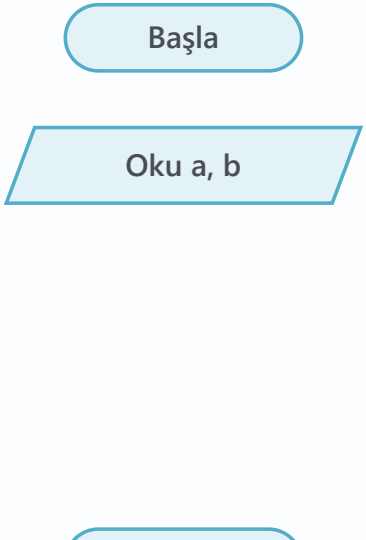
- Tek bir **Başla** ve en az bir **Bitir** bulunur.
- Akış yukarıdan aşağıya, soldan sağa okunur.
- Her sembolün bir çıkışı vardır; yalnızca **Karar** sembolünün iki çıkışı (E/H) olur.
- Karar çıkışları mutlaka **Evet/Hayır** diye etiketlenir.
- Çizgiler kesişmemeli; gerekirse bağlantı noktası kullanılır.
- Kutulara kısa ve açık ifadeler yazılır.

Adım Adım Çizim: Dikdörtgenin Alanı



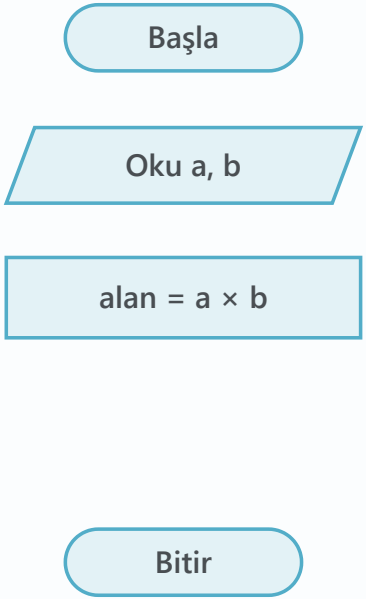
Problem: **dikdörtgenin alanını** bulmak. Önce iskelet: bir **Başla** ve bir **Bitir**.

Adım Adım Çizim: Dikdörtgenin Alanı



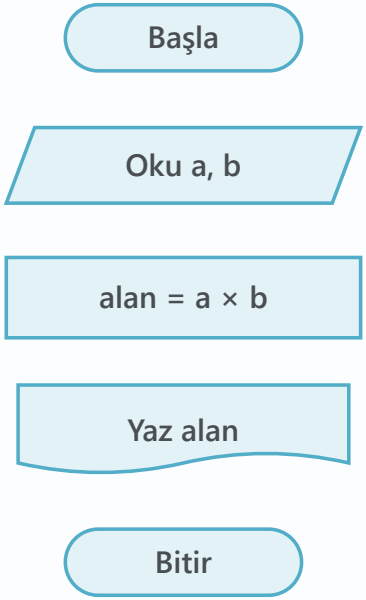
Girdiler belirlenir: kenar uzunlukları a ve b klavyeden okunur (paralelkenar).

Adım Adım Çizim: Dikdörtgenin Alanı



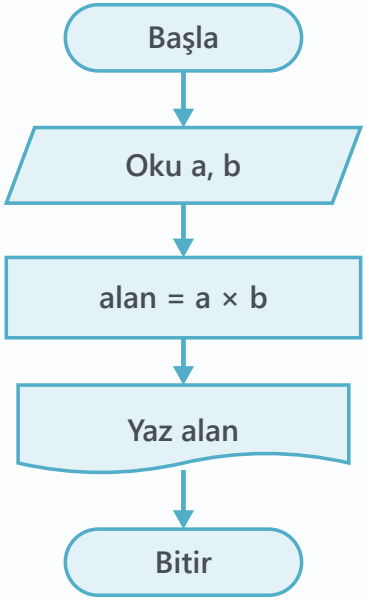
İşlem eklenir: `alan = a × b` hesaplanır ve alan değişkenine atanır (dikdörtgen).

Adım Adım Çizim: Dikdörtgenin Alanı



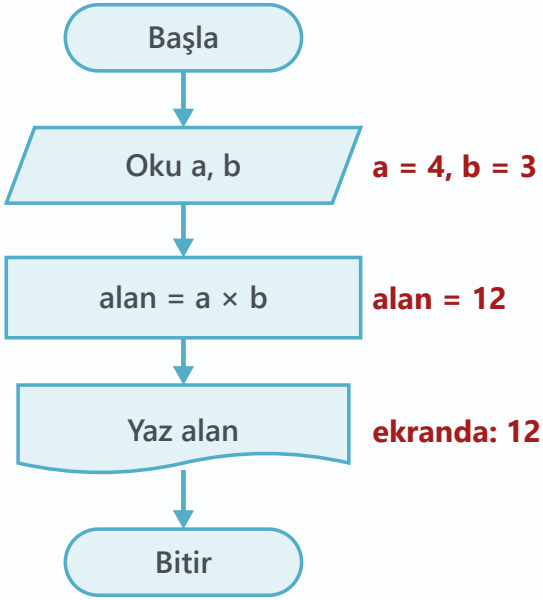
Çıktı eklenir: hesaplanan alan ekrana yazılır (Yazdır).

Adım Adım Çizim: Dikdörtgenin Alanı



Semboller mantıklı sırayla **akış çizgileri** ile bağlanır. Bu problemde karar noktası yok: akış **sıralıdır**.

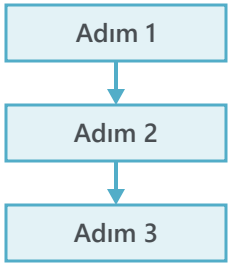
Adım Adım Çizim: Dikdörtgenin Alanı



Sınama: a = 4, b = 3 girilirse alan = 12 hesaplanır ve ekranda **12** görülür. Diyagram doğru çalışıyor.

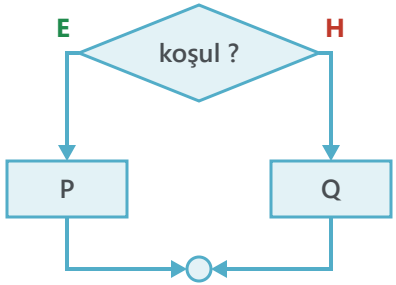
Kontrol Mekanizmaları

Sıralı akış



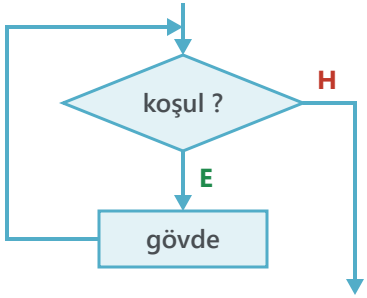
İşlem adımları sırayla, birer kez gerçekleşir.

Koşullu akış



Koşula bağlı olarak farklı işlem adımlarına yönelinir.

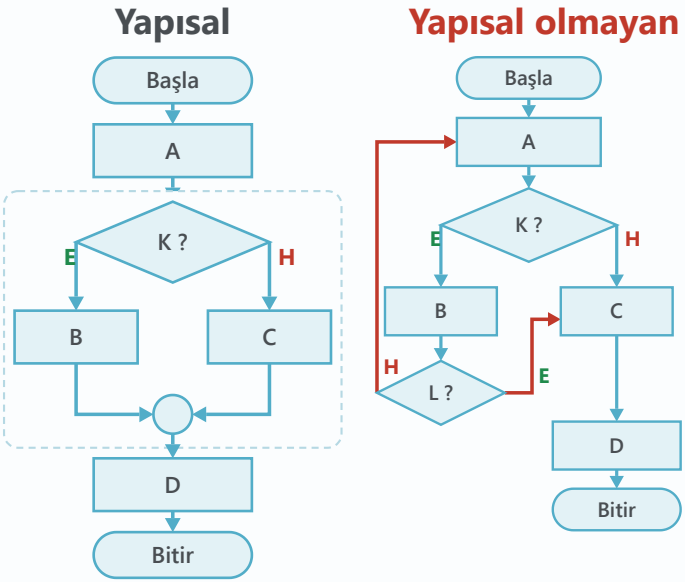
Tekrarlı akış



Koşul sağlandığı sürece bazı adımlar tekrarlanır.

Yapısal programlama teoremi (Böhm ve Jacopini, 1966): her algoritma yalnızca bu üç yapı (sıra, seçim, tekrar) iç içe kullanılarak yazılabilir. Java'da karşılıkları: art arda deyimler, `if/else`, `while` / `for`.

Yapısal ve Yapısal Olmayan Diyagram

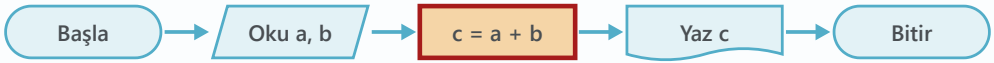


- **Yapısal** diyagramda her kontrol yapısının **tek girişi ve tek çıkışı** vardır (kesikli çerçeve). Yapılar ya art arda dizilir ya da birbirinin içine yerleştirilir.
- **Yapısal olmayan** diyagramda oklar istenen yere atlar: L kararının Evet oku öteki kolun ortasına (C) girer, Hayır oku geriye, A'ya döner. Böyle "spagetti" akış izlemeyi ve hata ayıklamayı zorlaştırır.
- Eski dillerde bu atlamalar `goto` ile yazılırdı. Java'da `goto` ayrılmış bir sözcüktür ama kullanılamaz.
- `break` ve `continue` sınırlı atlamalardır: yalnızca bulundukları döngüyü bitirir ya da sonraki tura geçer.
- Böhm–Jacopini teoremi gereği her diyagram, gerekirse bir bayrak değişkeniyle, yapısal biçime dönüştürülebilir.

SIRALI AKIŞ

Sıralı Akış (Sequential Flow)

Yapılacak işlerin (komutların) art arda, mantıksal bir sıra ile dizilmesidir. Akış yukarıdan aşağıya ilerler.



Basit
Sade ve anlaşılır bir akışı temsil eder; dallanma ve tekrar yoktur.

Kolay bakım
Hata ayıklamak ve kodu anlamak kolaydır: adımlar yazıldığı sırayla izlenir.

Sıra önemlidir
Bir adım öncekinin sonucunu kullanıyorsa yer değiştiremez. Birbirinden bağımsız adımların (ör. iki ayrı okuma) sırası ise değiştirilebilir.

Geri dönüş yok
Her adım tam olarak **bir kez** çalışır; önceki bir adıma geri dönülmez, hiçbir adım atlanmaz.

Yanlış sıra örneği: `sonuç = sayı1 + sayı2` adımı sayılar okunmadan önce yazılırsa, toplama henüz bilinmeyen değerlerle yapılmaya çalışılır. Java'da değişkenler tanımlanmış ama değer almamışsa derleyici bunu yakalar: `variable sayi1 might not have been initialized`.

SIRALI AKIŞ

Örnek: İki Sayının Toplamı

Problem: klavyeden girilen iki sayının toplamını ekrana yazmak.

Algoritma

1. Başla
2. İlk sayıyı oku (sayı1)
3. İkinci sayıyı oku (sayı2)
4. İki sayıyı topla, sonuç'a ata
5. Sonuç'u ekrana yaz
6. Bitir

Her algoritma adımı diyagramda bir sembole karşılık gelir.



Diyagramdan Java'ya: Toplam Programı

```
1  import java.util.Scanner;
2
3  public class IkiSayiToplami {
▶      public static void main(String[] args) {
▶          Scanner giris = new Scanner(System.in);
6          System.out.print("İlk sayı: ");
7          int sayi1 = giris.nextInt();
8          System.out.print("İkinci sayı: ");
9          int sayi2 = giris.nextInt();
10         int sonuc = sayi1 + sayi2;
11         System.out.println("Sonuç: " + sonuc);
12         giris.close();
13     }
14 }
```

DEĞİŞKENLER

`sayi1 = –` `sayi2 = –` `sonuc = –`

ÇIKTI



Program `main` 'den başlar (**Başla**). Klavyeden okumak için bir `Scanner` oluşturulur.

Diyagramdan Java'ya: Toplam Programı

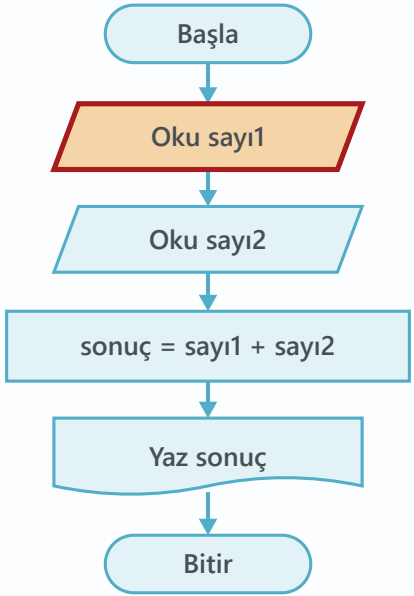
```
1  import java.util.Scanner;
2
3  public class IkiSayiToplami {
4      public static void main(String[] args) {
5          Scanner giris = new Scanner(System.in);
           System.out.print("İlk sayı: ");
           int sayi1 = giris.nextInt();
8          System.out.print("İkinci sayı: ");
9          int sayi2 = giris.nextInt();
10         int sonuc = sayi1 + sayi2;
11         System.out.println("Sonuç: " + sonuc);
12         giris.close();
13     }
14 }
```

DEĞİŞKENLER

`sayi1 = 7` `sayi2 = –` `sonuc = –`

ÇIKTI

```
İlk sayı: 7
```



Mesaj yazılır, kullanıcı **7** girer: `nextInt()` değeri `sayi1`'e atar.

Diyagramdan Java'ya: Toplam Programı

```
1  import java.util.Scanner;
2
3  public class IkiSayiToplami {
4      public static void main(String[] args) {
5          Scanner giris = new Scanner(System.in);
6          System.out.print("İlk sayı: ");
7          int sayi1 = giris.nextInt();
▶          System.out.print("İkinci sayı: ");
▶          int sayi2 = giris.nextInt();
10         int sonuc = sayi1 + sayi2;
11         System.out.println("Sonuç: " + sonuc);
12         giris.close();
13     }
14 }
```

DEĞİŞKENLER

`sayi1 = 7` `sayi2 = 5` `sonuc = –`

ÇIKTI

```
İlk sayı: 7
İkinci sayı: 5
```



Kullanıcı **5** girer: `sayi2 = 5`.

Diyagramdan Java'ya: Toplam Programı

```
1  import java.util.Scanner;
2
3  public class IkiSayiToplami {
4      public static void main(String[] args) {
5          Scanner giris = new Scanner(System.in);
6          System.out.print("İlk sayı: ");
7          int sayi1 = giris.nextInt();
8          System.out.print("İkinci sayı: ");
9          int sayi2 = giris.nextInt();
▶          int sonuc = sayi1 + sayi2;
11         System.out.println("Sonuç: " + sonuc);
12         giris.close();
13     }
14 }
```

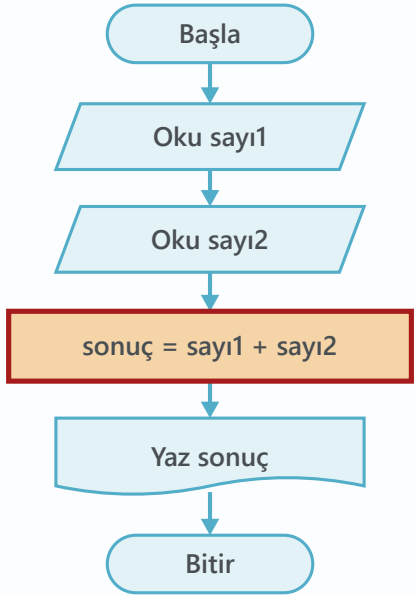
DEĞİŞKENLER

`sayi1 = 7` `sayi2 = 5`

`sonuc = 12`

ÇIKTI

```
İlk sayı: 7
İkinci sayı: 5
```



İşlem kutusu: `sonuc = 7 + 5 = 12`.

Diyagramdan Java'ya: Toplam Programı

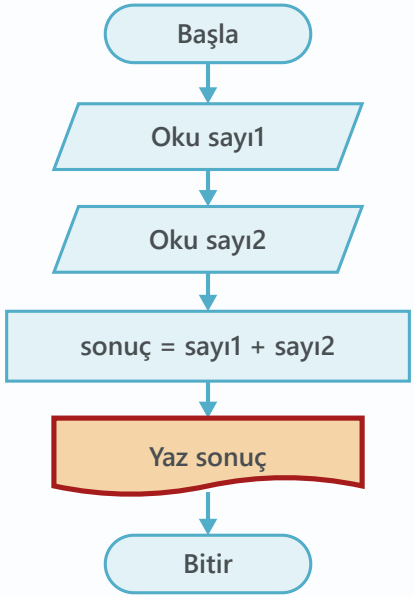
```
1  import java.util.Scanner;
2
3  public class IkiSayiToplami {
4      public static void main(String[] args) {
5          Scanner giris = new Scanner(System.in);
6          System.out.print("İlk sayı: ");
7          int sayi1 = giris.nextInt();
8          System.out.print("İkinci sayı: ");
9          int sayi2 = giris.nextInt();
10         int sonuc = sayi1 + sayi2;
▶          System.out.println("Sonuç: " + sonuc);
12         giris.close();
13     }
14 }
```

DEĞİŞKENLER

`sayi1 = 7` `sayi2 = 5` `sonuc = 12`

ÇIKTI

```
İlk sayı: 7
İkinci sayı: 5
Sonuç: 12
```



Yazdır kutusu: `println` sonucu ekrana yazar.

Diyagramdan Java'ya: Toplam Programı

```
1  import java.util.Scanner;
2
3  public class IkiSayiToplami {
4      public static void main(String[] args) {
5          Scanner giris = new Scanner(System.in);
6          System.out.print("İlk sayı: ");
7          int sayi1 = giris.nextInt();
8          System.out.print("İkinci sayı: ");
9          int sayi2 = giris.nextInt();
10         int sonuc = sayi1 + sayi2;
11         System.out.println("Sonuç: " + sonuc);
           giris.close();
       }
14 }
```

DEĞİŞKENLER

`sayi1 = 7` `sayi2 = 5` `sonuc = 12`

ÇIKTI

```
İlk sayı: 7
İkinci sayı: 5
Sonuç: 12
```



`Scanner` kapatılır, `main` biter (**Bitir**). Her satır tam bir kez çalıştı.

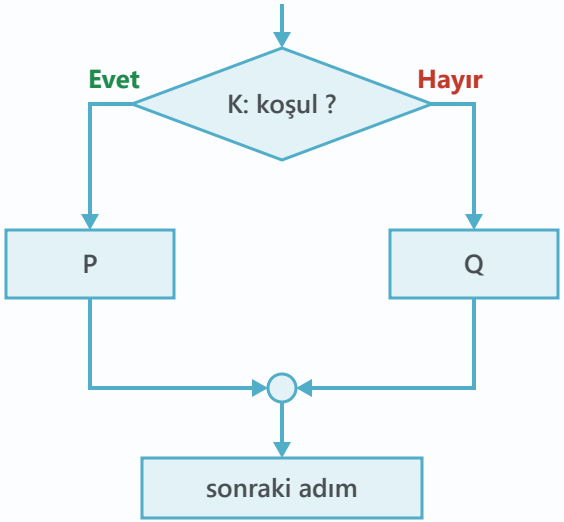
KOŞULLU AKIŞ

Koşullu Akış (Conditional Flow)

İki (veya daha çok) ihtimalden **birinin seçildiği** akış türüdür.

- **Karar noktası (K):** bir koşul sorusu bulunur; cevabı Evet ya da Hayır'dır.
- **Evet'e göre akış (P):** cevap evet ise P işlemi yönünde ilerlenir.
- **Hayır'a göre akış (Q):** cevap hayır ise Q işlemi yönünde ilerlenir.
- **Esneklik** (koşullara göre farklı davranış), **karar verme yetisi** (koşula bağlı işlem) ve **kod kontrolü** (durumu denetleyip ona göre işlem) sağlar.

```
if (K) {
    P;   // koşul doğru
} else {
    Q;   // koşul yanlış
}
```



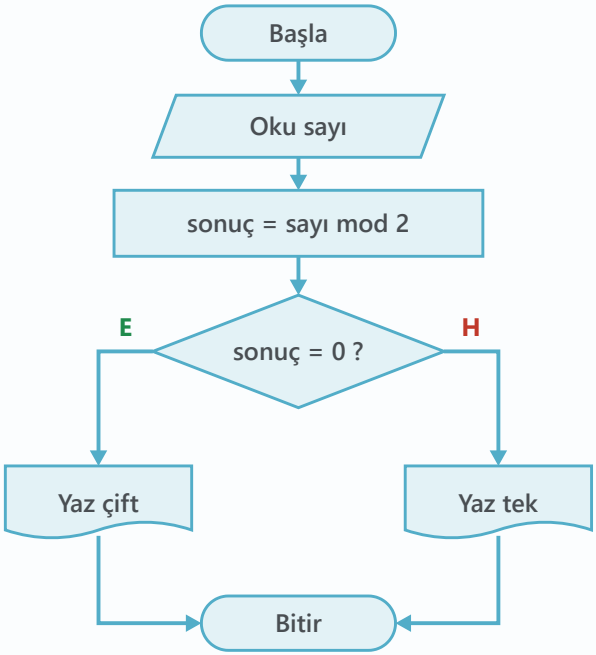
KOŞULLU AKIŞ

Örnek: Sayı Tek mi Çift mi?

Problem: verilen sayının tek mi çift mi olduğunu ekrana yazmak.

1. Başla
2. Sayıyı oku
3. Sayı mod 2'yi al, sonuç'a ata
4. Sonuç 0 ise 5'e git, değilse 6'ya git
5. Ekrana "çift" yaz, 7'ye git
6. Ekrana "tek" yaz
7. Bitir

mod: bölümden kalan. 7 mod 2 = 1, 8 mod 2 = 0. Çift sayının 2'ye bölümünden kalan 0'dır. Java'da `sayi % 2`.



Diyagramdan Java'ya: Tek mi Çift mi?

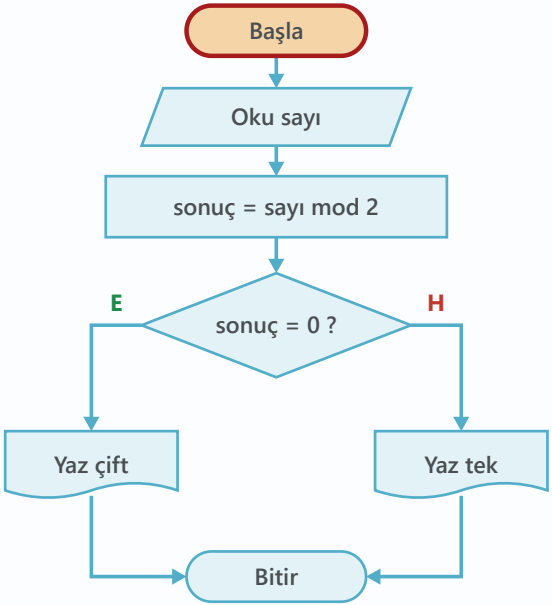
```
1  import java.util.Scanner;
2
3  public class TekMiCiftMi {
4      public static void main(String[] args) {
5          Scanner giris = new
   Scanner(System.in);
6          System.out.print("Sayı: ");
7          int sayi = giris.nextInt();
8          int sonuc = sayi % 2;
9          if (sonuc == 0) {
10             System.out.println("çift");
11         } else {
12             System.out.println("tek");
13         }
14         giris.close();
15     }
16 }
```

DEĞİŞKENLER

`sayi = –`

`sonuc = –`

ÇIKTI

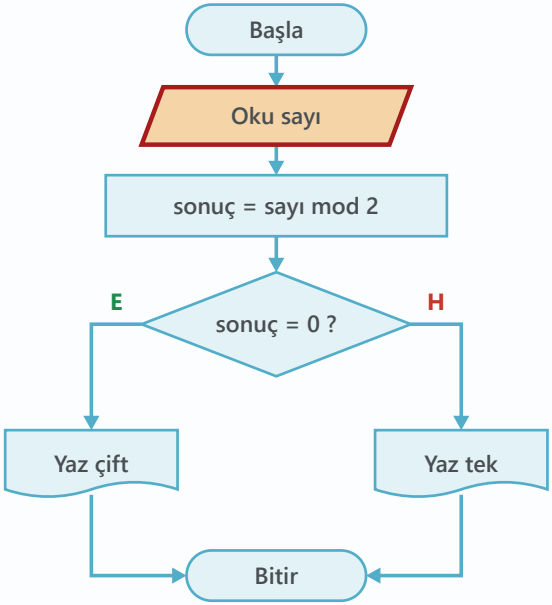


Başla: `Scanner` oluşturulur.

Adım 1 / 6

Diyagramdan Java'ya: Tek mi Çift mi?

```
1  import java.util.Scanner;
2
3  public class TekMiCiftMi {
4      public static void main(String[] args) {
5          Scanner giris = new
   Scanner(System.in);
           System.out.print("Sayı: ");
           int sayi = giris.nextInt();
8          int sonuc = sayi % 2;
9          if (sonuc == 0) {
10             System.out.println("çift");
11         } else {
12             System.out.println("tek");
13         }
14         giris.close();
15     }
16 }
```



Kullanıcı **7** girer: `sayi = 7` .

DEĞİŞKENLER

`sayi = 7`

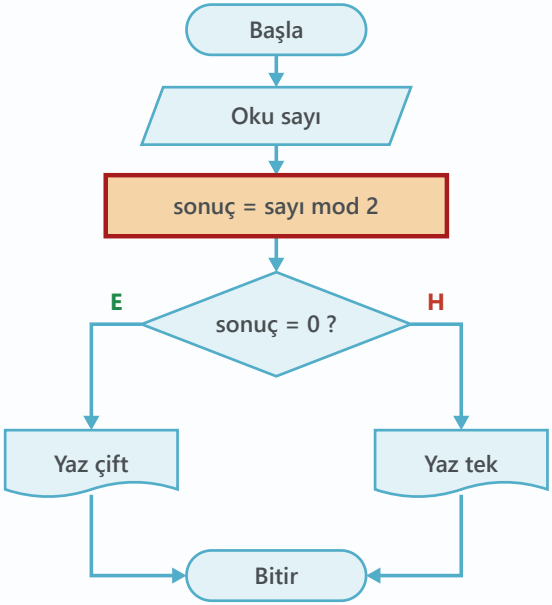
`sonuc = –`

ÇIKTI

```
Sayı: 7
```

Diyagramdan Java'ya: Tek mi Çift mi?

```
1  import java.util.Scanner;
2
3  public class TekMiCiftMi {
4      public static void main(String[] args) {
5          Scanner giris = new
   Scanner(System.in);
6          System.out.print("Sayı: ");
7          int sayi = giris.nextInt();
▶          int sonuc = sayi % 2;
9          if (sonuc == 0) {
10             System.out.println("çift");
11         } else {
12             System.out.println("tek");
13         }
14         giris.close();
15     }
16 }
```



`sonuc = 7 % 2 = 1` (7 = 3 × 2 + 1).

DEĞİŞKENLER

`sayi = 7`

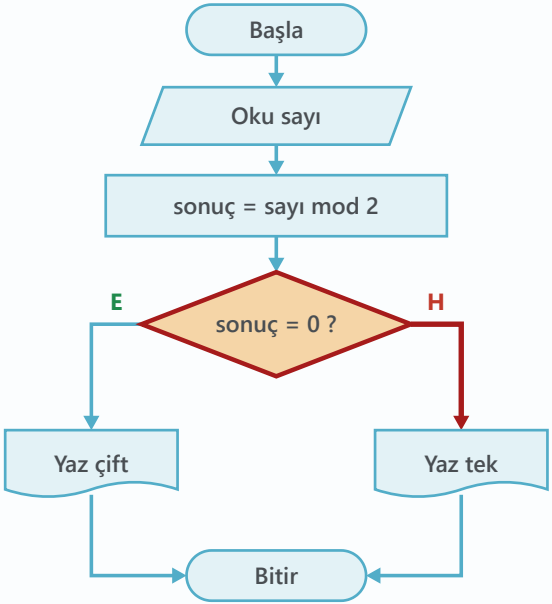
`sonuc = 1`

ÇIKTI

```
Sayı: 7
```

Diyagramdan Java'ya: Tek mi Çift mi?

```
1  import java.util.Scanner;
2
3  public class TekMiCiftMi {
4      public static void main(String[] args) {
5          Scanner giris = new
   Scanner(System.in);
6          System.out.print("Sayı: ");
7          int sayi = giris.nextInt();
8          int sonuc = sayi % 2;
▶          if (sonuc == 0) {
10             System.out.println("çift");
11         } else {
12             System.out.println("tek");
13         }
14         giris.close();
15     }
16 }
```



Karar: `1 == 0` **yanlış** → Hayır (H) yolu, `else` bloğu.

DEĞİŞKENLER

`sayi = 7`

`sonuc = 1`

ÇIKTI

```
Sayı: 7
```

Diyagramdan Java'ya: Tek mi Çift mi?

```
1  import java.util.Scanner;
2
3  public class TekMiCiftMi {
4      public static void main(String[] args) {
5          Scanner giris = new
   Scanner(System.in);
6          System.out.print("Sayı: ");
7          int sayi = giris.nextInt();
8          int sonuc = sayi % 2;
9          if (sonuc == 0) {
10             System.out.println("çift");
           } else {
               System.out.println("tek");
13         }
14         giris.close();
15     }
16 }
```

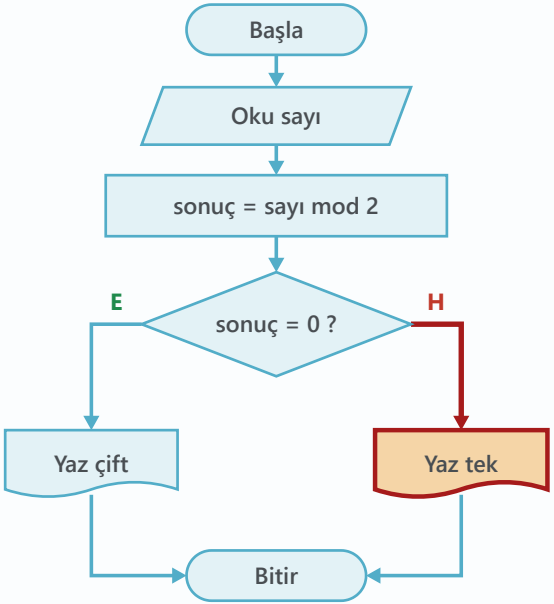
DEĞİŞKENLER

`sayi = 7`

`sonuc = 1`

ÇIKTI

```
Sayı: 7
tek
```



"tek" yazılır. `if` bloğundaki 10. satır **atlandı**.

Diyagramdan Java'ya: Tek mi Çift mi?

```
1  import java.util.Scanner;
2
3  public class TekMiCiftMi {
4      public static void main(String[] args) {
5          Scanner giris = new
   Scanner(System.in);
6          System.out.print("Sayı: ");
7          int sayi = giris.nextInt();
8          int sonuc = sayi % 2;
9          if (sonuc == 0) {
10             System.out.println("çift");
11         } else {
12             System.out.println("tek");
13         }
▶          giris.close();
▶      }
16 }
```

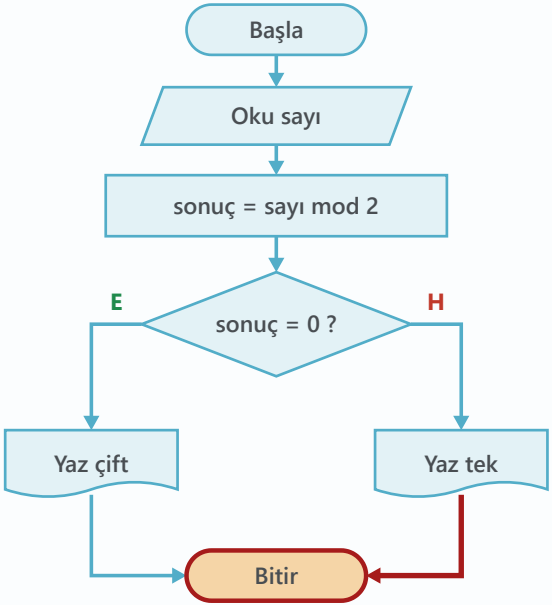
DEĞİŞKENLER

`sayi = 7`

`sonuc = 1`

ÇIKTI

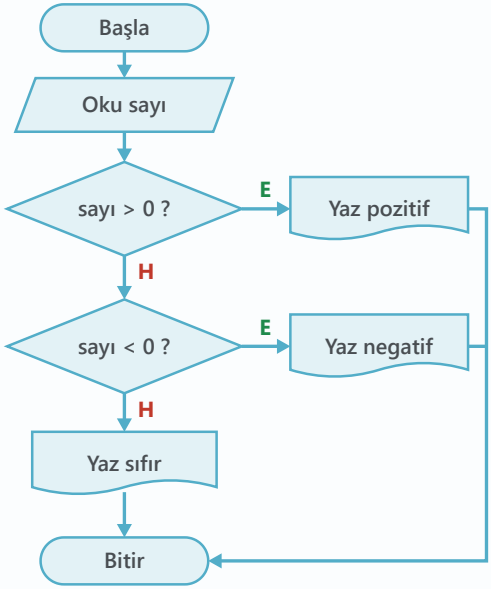
```
Sayı: 7
tek
```



İki yol **Bitir**'de birleşir. 8 girilseydi `sonuc = 0` olur, E yolundan "çift" yazılırdı.

KOŞULLU AKIŞ

İç İçe Karar: Pozitif, Negatif, Sıfır



Üç ihtimal varsa iki karar kutusu art arda bağlanır: ilk sorunun **Hayır** çıkışı ikinci soruya gider.

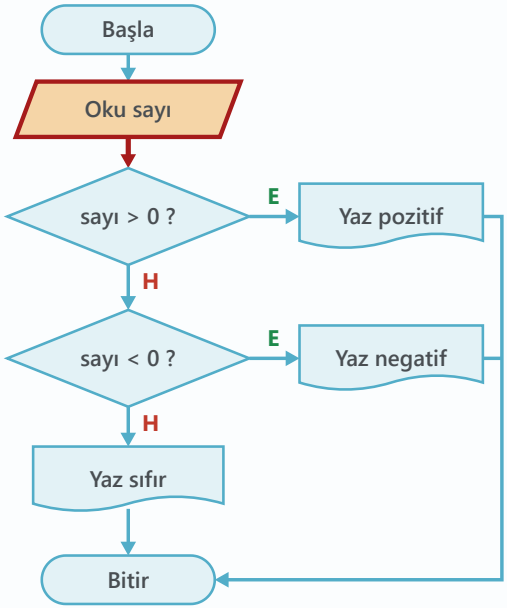
```
int sayi = giris.nextInt();
if (sayi > 0) {
    System.out.println("pozitif");
} else if (sayi < 0) {
    System.out.println("negatif");
} else {
    System.out.println("sıfır");
}
```

girdi	izlenen yol	çıktı
5	1. karar E	`pozitif`
−2	1. karar H, 2. karar E	`negatif`
0	1. karar H, 2. karar H	`sıfır`

Tek yönlü karar (yalnız Evet kolu): Java'da `else`'siz `if`.

İç İçe Karar: Kod İzleme

```
1 Scanner giris = new Scanner(System.in);
▶ int sayi = giris.nextInt();
3 if (sayi > 0) {
4     System.out.println("pozitif");
5 } else if (sayi < 0) {
6     System.out.println("negatif");
7 } else {
8     System.out.println("sıfır");
9 }
```



girdi: −4

DEĞİŞKENLER

`sayi = -4`

ÇIKTI

1. çalıştırma: kullanıcı −4 girer, `sayi` = −4.

İç İçe Karar: Kod İzleme

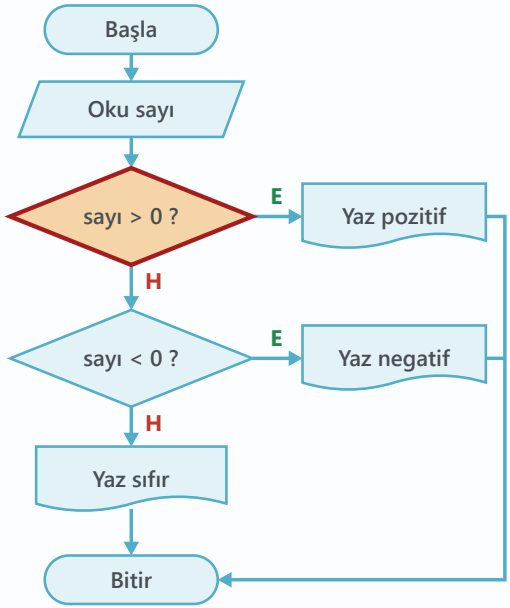
```
1 Scanner giris = new Scanner(System.in);
2 int sayi = giris.nextInt();
▶ if (sayi > 0) {
4     System.out.println("pozitif");
5 } else if (sayi < 0) {
6     System.out.println("negatif");
7 } else {
8     System.out.println("sıfır");
9 }
```

DEĞİŞKENLER

`sayi = -4`

ÇIKTI





−4 > 0 → yanlış

1. karar: `−4 > 0` **yanlış** → Hayır çıkışı.

İç İçe Karar: Kod İzleme

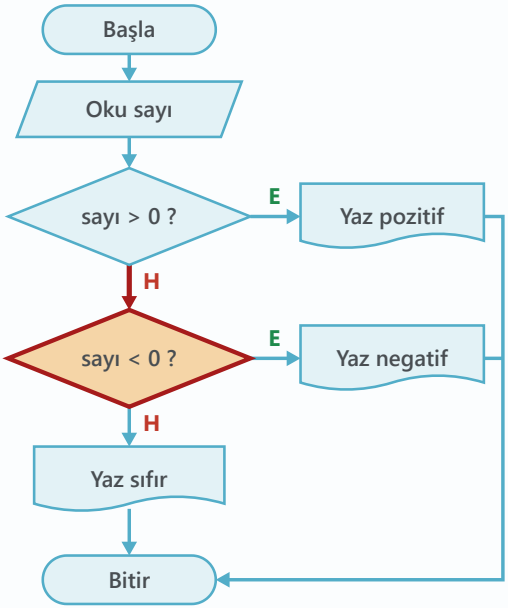
```
1 Scanner giris = new Scanner(System.in);
2 int sayi = giris.nextInt();
3 if (sayi > 0) {
4     System.out.println("pozitif");
▶ } else if (sayi < 0) {
6     System.out.println("negatif");
7 } else {
8     System.out.println("sıfır");
9 }
```

DEĞİŞKENLER

`sayi = -4`

ÇIKTI





−4 < 0 → doğru

Hayır çıkışı 2. karara gider: `−4 < 0` **doğru** → Evet.

İç İçe Karar: Kod İzleme

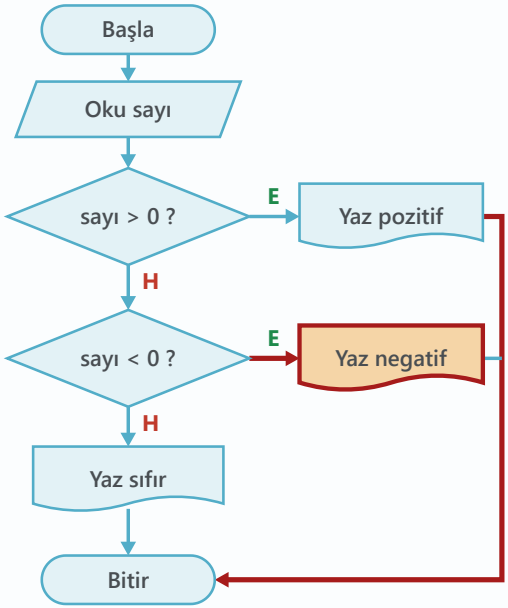
```
1 Scanner giris = new Scanner(System.in);
2 int sayi = giris.nextInt();
3 if (sayi > 0) {
4     System.out.println("pozitif");
5 } else if (sayi < 0) {
      System.out.println("negatif");
7 } else {
8     System.out.println("sıfır");
9 }
```

DEĞİŞKENLER

`sayi = -4`

ÇIKTI

```
negatif
```



çıktı: negatif

`negatif` yazılır; `else` kolu atlanır, akış Bitir'e gider.

İç İçe Karar: Kod İzleme

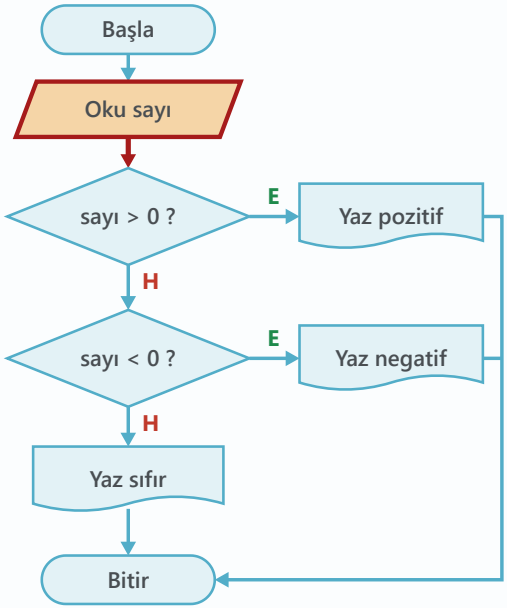
```
1 Scanner giris = new Scanner(System.in);
▶ int sayi = giris.nextInt();
3 if (sayi > 0) {
4     System.out.println("pozitif");
5 } else if (sayi < 0) {
6     System.out.println("negatif");
7 } else {
8     System.out.println("sıfır");
9 }
```

DEĞİŞKENLER

`sayi = 0`

ÇIKTI



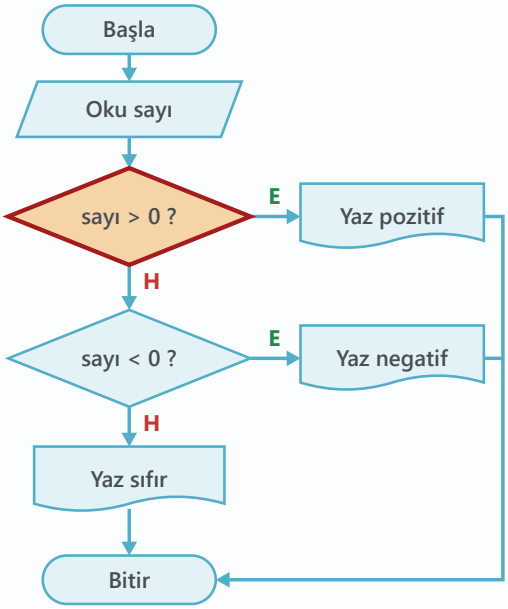


girdi: 0

2. çalıştırma: bu kez kullanıcı 0 girer (çıktı paneli yeni çalıştırmayı gösterir).

İç İçe Karar: Kod İzleme

```
1 Scanner giris = new Scanner(System.in);
2 int sayi = giris.nextInt();
▶ if (sayi > 0) {
4     System.out.println("pozitif");
5 } else if (sayi < 0) {
6     System.out.println("negatif");
7 } else {
8     System.out.println("sıfır");
9 }
```



DEĞİŞKENLER

`sayi = 0`

ÇIKTI

0 > 0 → yanlış

`0 > 0` **yanlış**: 0 pozitif değildir.

İç İçe Karar: Kod İzleme

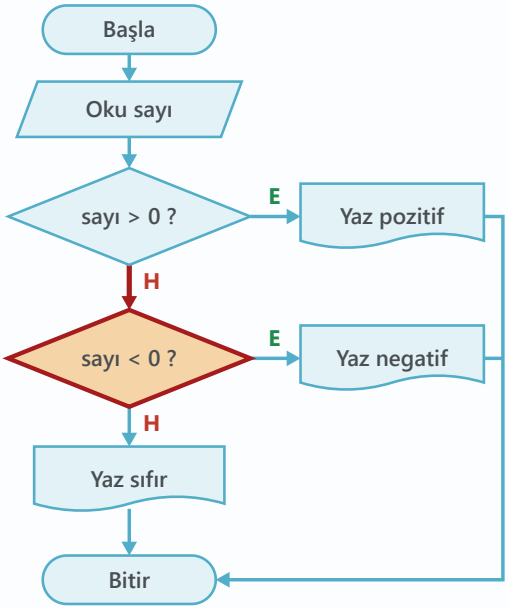
```
1 Scanner giris = new Scanner(System.in);
2 int sayi = giris.nextInt();
3 if (sayi > 0) {
4     System.out.println("pozitif");
▶ } else if (sayi < 0) {
6     System.out.println("negatif");
7 } else {
8     System.out.println("sıfır");
9 }
```

DEĞİŞKENLER

`sayi = 0`

ÇIKTI





0 < 0 → yanlış

`0 < 0` da **yanlış** → ikinci kararın Hayır çıkışı.

İç İçe Karar: Kod İzleme

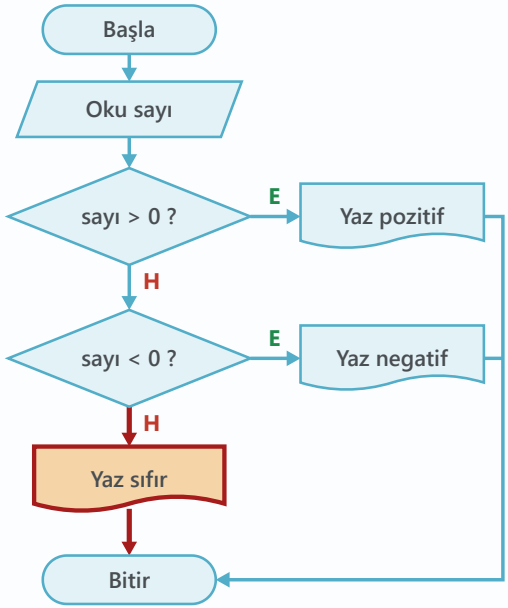
```
1 Scanner giris = new Scanner(System.in);
2 int sayi = giris.nextInt();
3 if (sayi > 0) {
4     System.out.println("pozitif");
5 } else if (sayi < 0) {
6     System.out.println("negatif");
7 } else {
▶     System.out.println("sıfır");
9 }
```

DEĞİŞKENLER

`sayi = 0`

ÇIKTI

```
sıfır
```

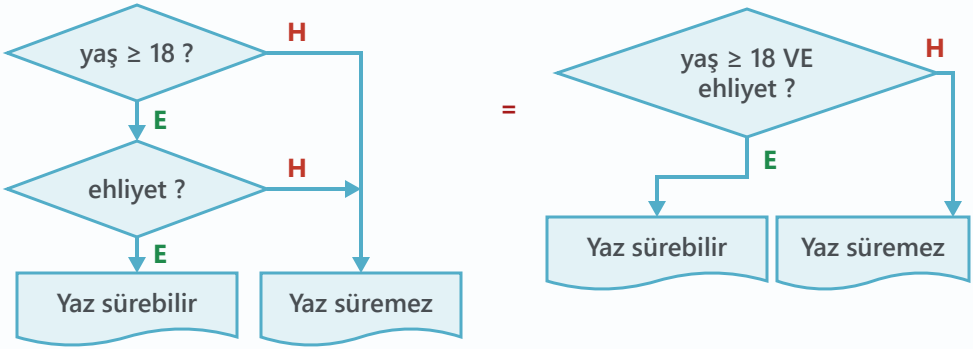


çıktı: sıfır

İki koşul da yanlış: son `else` çalışır, `sıfır` yazılır. Üç koldan her seferinde yalnız **biri** çalışır.

KOŞULLU AKIŞ

Birleşik Koşullar: VE, VEYA



yaş	ehliyet	yaş ≥ 18 VE ehliyet	yaş ≥ 18 VEYA ehliyet
20	var	Evet → sürebilir	Evet
20	yok	Hayır → süremez	Evet
16	var	Hayır → süremez	Evet
16	yok	Hayır → süremez	Hayır

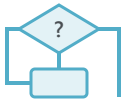
- **VE**: iki koşul da doğruysa Evet. İki kararla çizilirse ilk kararın **Hayır** çıkışı doğrudan "süremez"e gider; ikinci karar hiç sorulmaz.
- **VEYA**: biri doğruysa yeter. İki kararla çizilirse ilk kararın **Evet** çıkışı doğrudan sonuca gider.
- Java: `if (yas >= 18 && ehliyet)` , `||` . Java da ikinci koşula yalnızca gerekirse bakar.

TEKRARLI AKIŞ

Tekrarlı Akış (Döngü, Loop)

İşlemlerin **birden çok kez** tekrar ettiği akış türüdür. Diyagramda geriye doğru giden bir ok, karar kutusuna döner.

Koşula bağlı döngü



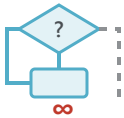
Koşul sağlandığı sürece tekrar eder. Kaç kez döneceği önceden bilinmeyebilir (ör. kullanıcı 0 girene kadar oku). Java: `while`.

Belirli sayıda tekrar



Tekrar sayısı baştan bellidir (ör. 1'den 10'a kadar). Java: `for`.

Sonsuz döngü



Koşul hep sağlandığı için işlemler durmadan tekrarlanır. Çoğunlukla bir **hatadır**; sunucu gibi sürekli çalışan programlarda bilinçli kullanılır.

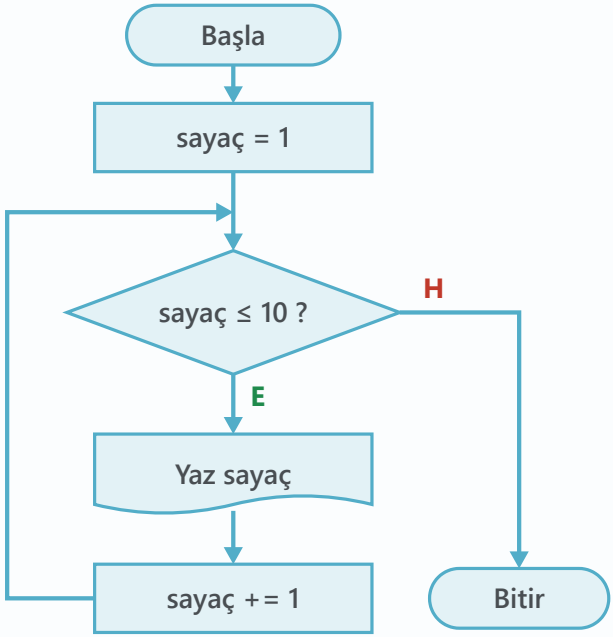
- **Verimli:** aynı işlem, kod tekrar tekrar yazılmadan istenen sayıda yapılır.
- **Dinamik:** tekrar sayısı girdiye göre değişebilir; program değişen koşullara uyum sağlar.
- **Kod tekrarını önler:** 1000 sayıyı yazdırmak için 1000 satır değil, 3–4 satır yeterlidir.

Örnek: 1'den 10'a Kadar Sayıları Yazdırma

Algoritma

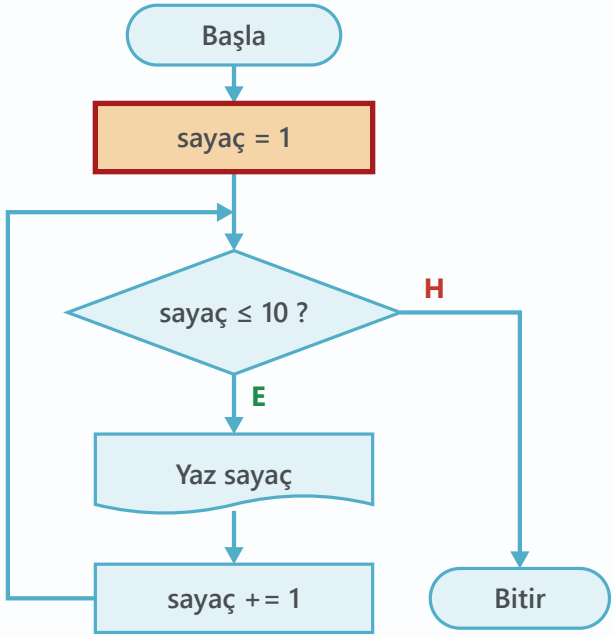
1. Başla
2. Sayaç = 1
3. Sayaç ≤ 10 ise 4'e git, değilse 7'ye git
4. Sayaç'ı yaz
5. Sayaç'ı 1 artır
6. 3'e git
7. Bitir

6. adımdaki "3'e git", diyagramda karar kutusunun üstüne dönen **geri oktur**. Bu ok döngüyü oluşturur; karar kutusu döngüden çıkış kapısıdır.



Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
          int sayac = 1;
4         while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

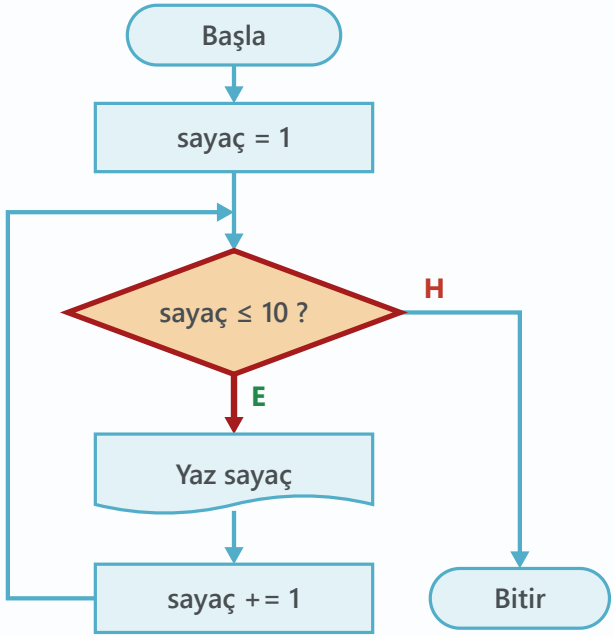
`sayac = 1`

ÇIKTI

`sayac = 1` : döngü değişkenine başlangıç değeri verilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
          while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

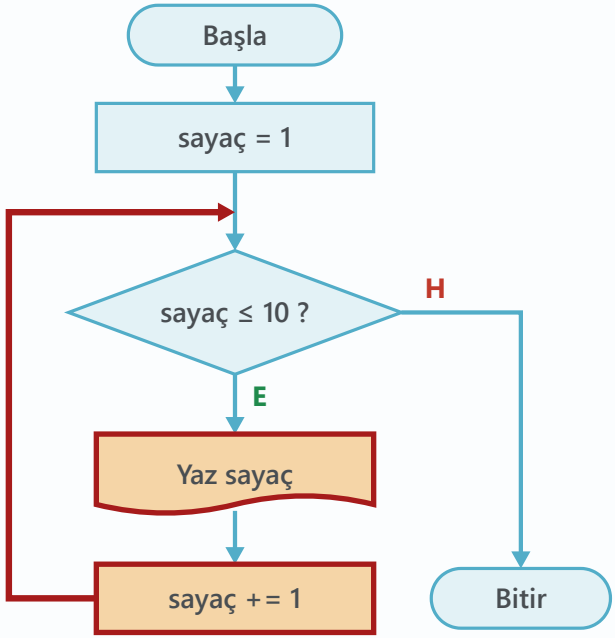
`sayac = 1`

ÇIKTI

Koşul: `1 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
▶             System.out.print(sayac + " ");
▶             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 2`

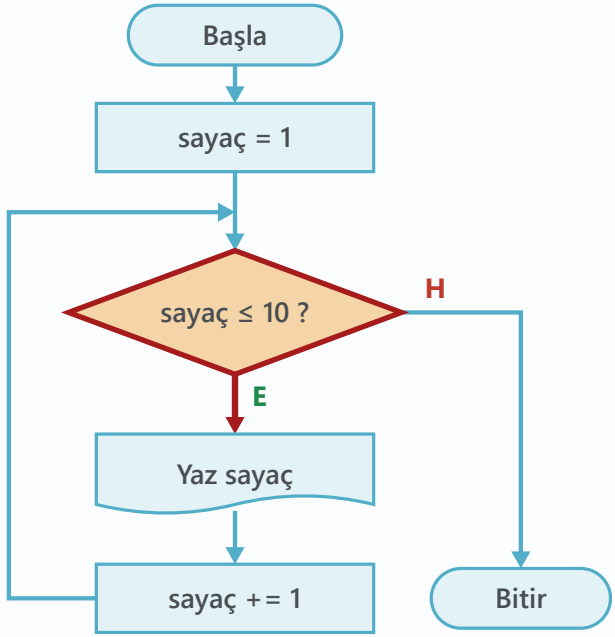
ÇIKTI

```
1
```

1 yazılır, `sayac` 2 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
▶         while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 2`

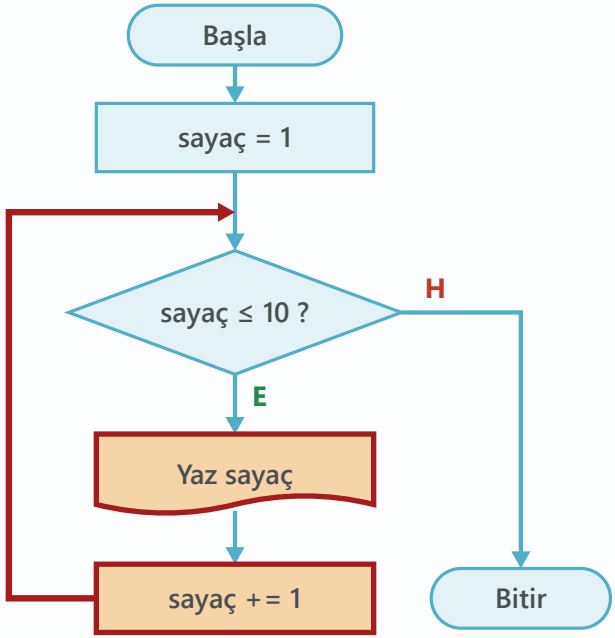
ÇIKTI

```
1
```

Koşul: `2 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
▶             System.out.print(sayac + " ");
▶             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 3`

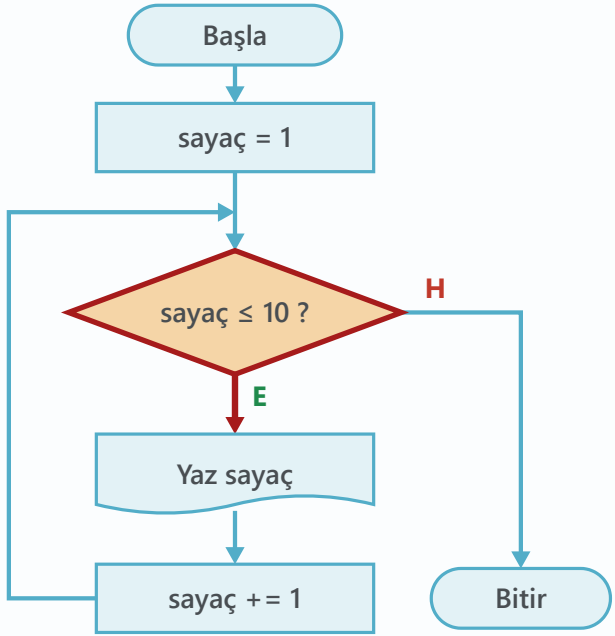
ÇIKTI

```
1 2
```

2 yazılır, `sayac` 3 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
▶         while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 3`

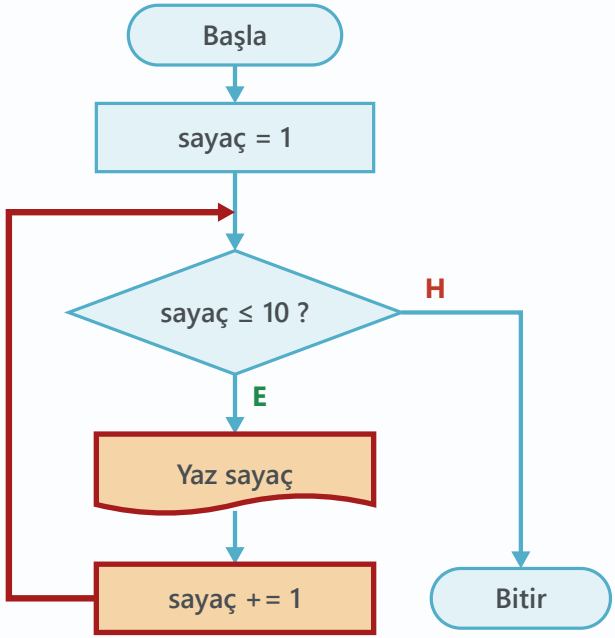
ÇIKTI

```
1 2
```

Koşul: `3 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
▶             System.out.print(sayac + " ");
▶             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 4`

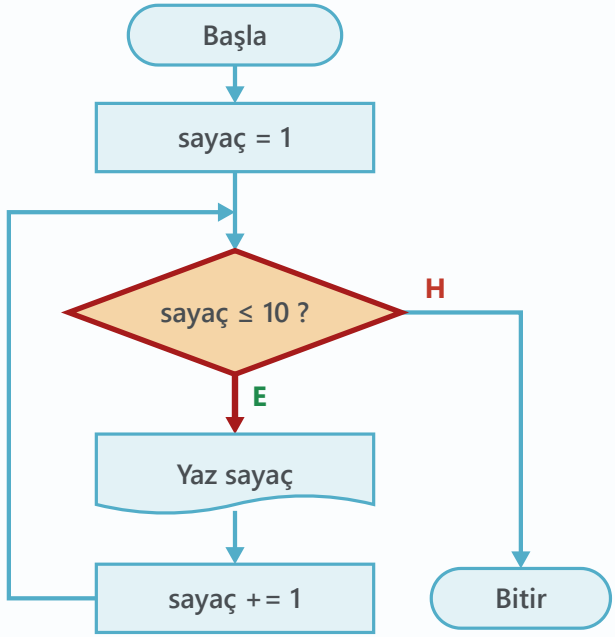
ÇIKTI

```
1 2 3
```

3 yazılır, `sayac` 4 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
▶         while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 4`

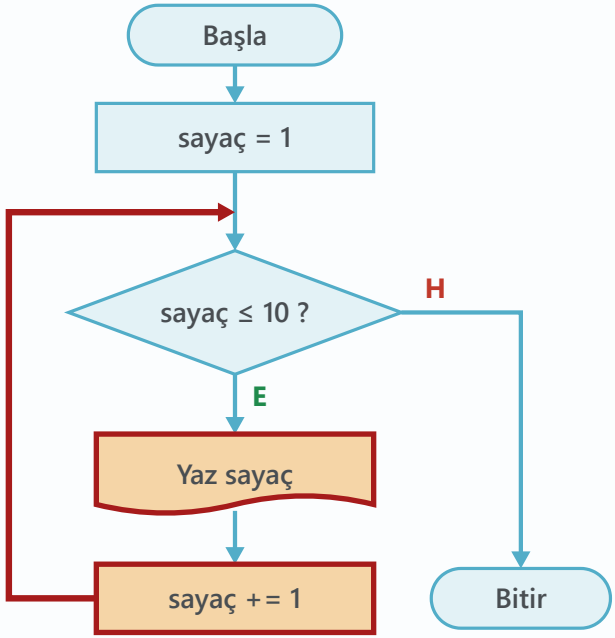
ÇIKTI

```
1 2 3
```

Koşul: `4 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
▶             System.out.print(sayac + " ");
▶             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 5`

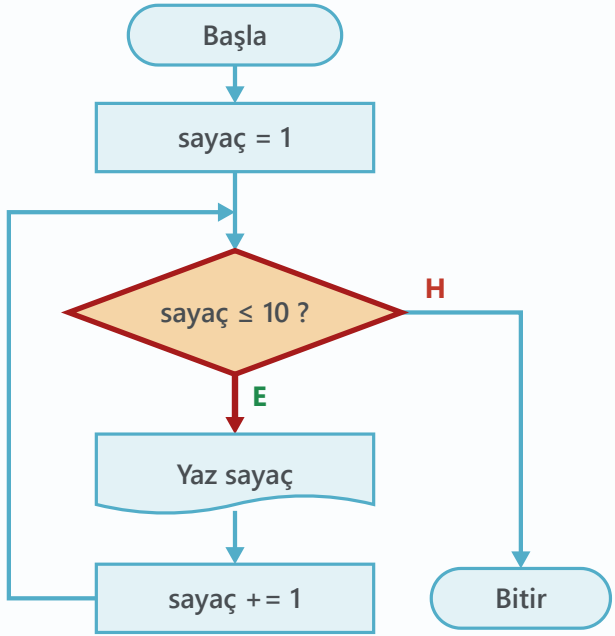
ÇIKTI

```
1 2 3 4
```

4 yazılır, `sayac` 5 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
▶         while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 5`

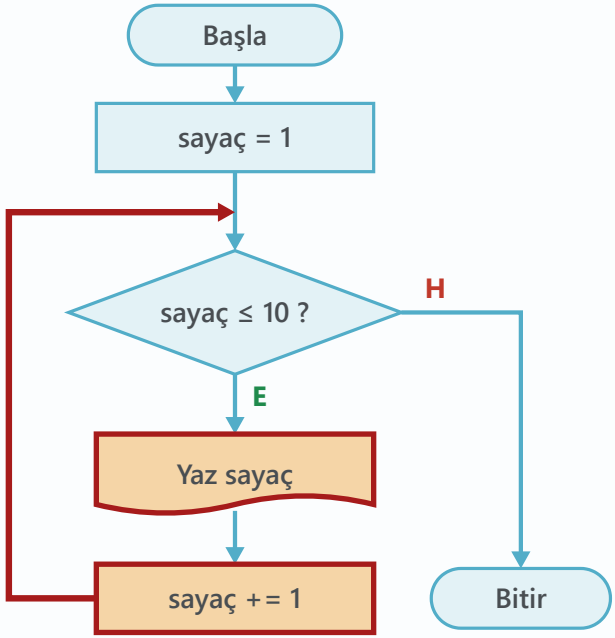
ÇIKTI

```
1 2 3 4
```

Koşul: `5 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
              System.out.print(sayac + " ");
              sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 6`

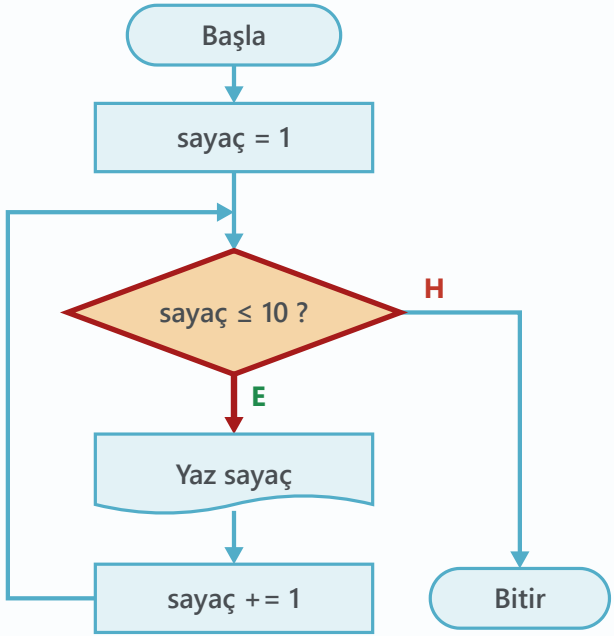
ÇIKTI

```
1 2 3 4 5
```

5 yazılır, `sayac` 6 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
          while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 6`

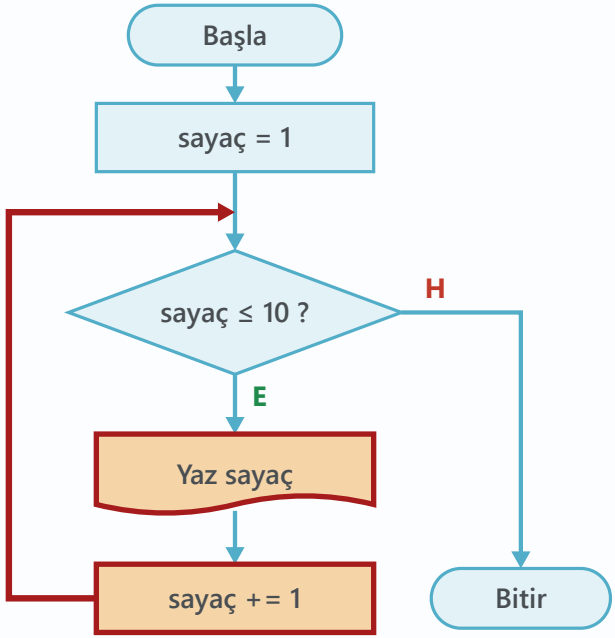
ÇIKTI

```
1 2 3 4 5
```

Koşul: `6 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
▶             System.out.print(sayac + " ");
▶             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 7`

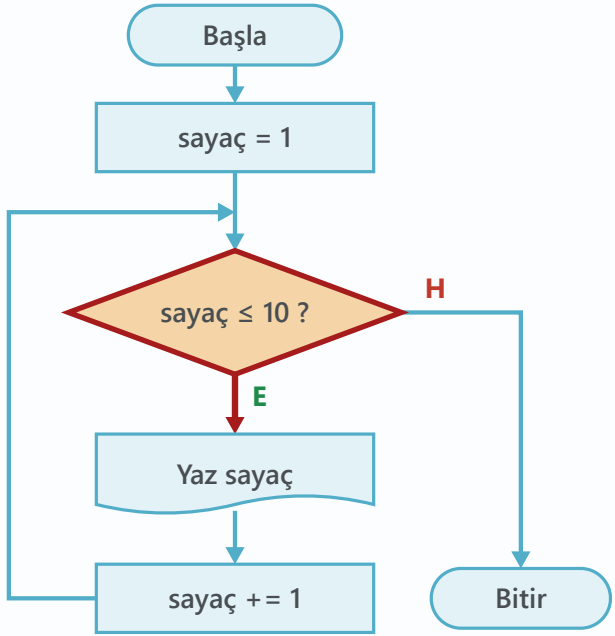
ÇIKTI

```
1 2 3 4 5 6
```

6 yazılır, `sayac` 7 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
▶         while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 7`

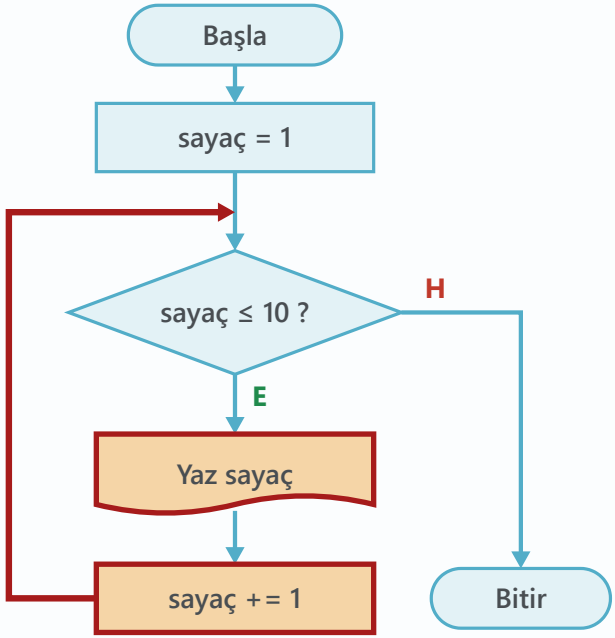
ÇIKTI

```
1 2 3 4 5 6
```

Koşul: `7 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
              System.out.print(sayac + " ");
              sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 8`

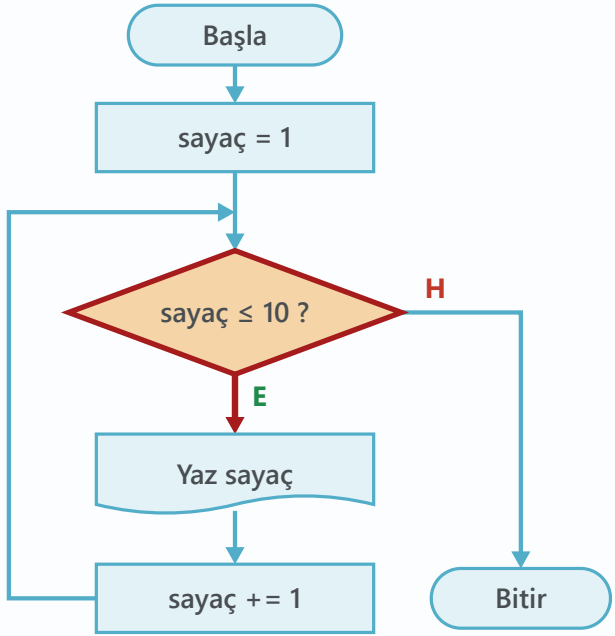
ÇIKTI

```
1 2 3 4 5 6 7
```

7 yazılır, `sayac` 8 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
▶         while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 8`

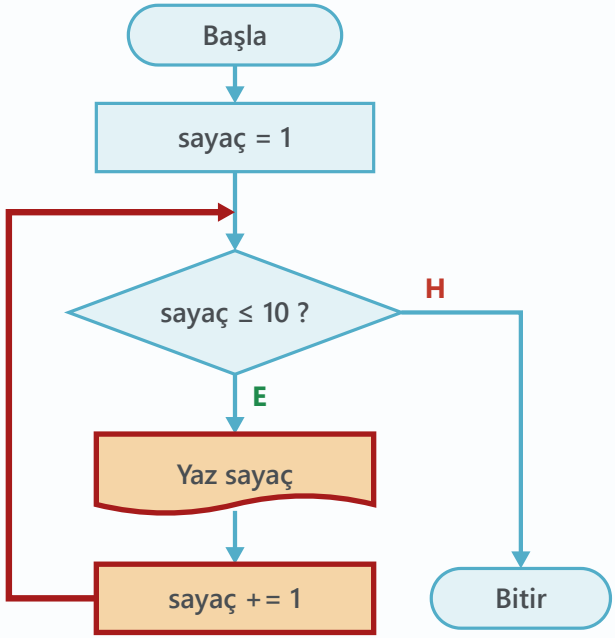
ÇIKTI

```
1 2 3 4 5 6 7
```

Koşul: `8 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
              System.out.print(sayac + " ");
              sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 9`

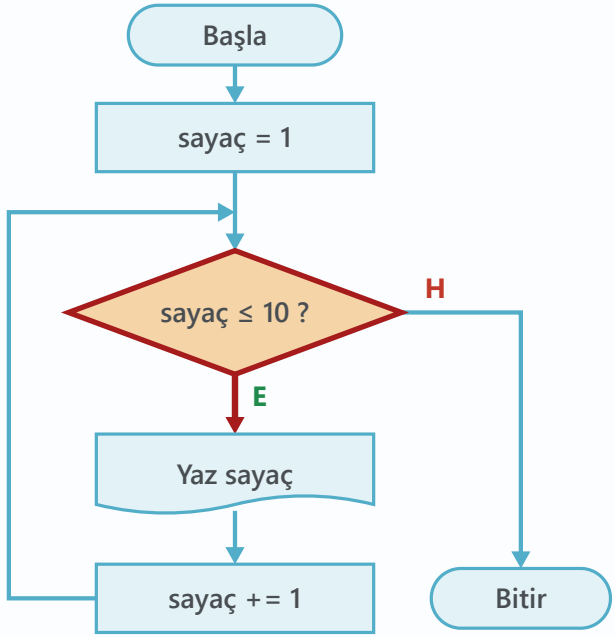
ÇIKTI

```
1 2 3 4 5 6 7 8
```

8 yazılır, `sayac` 9 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
          while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 9`

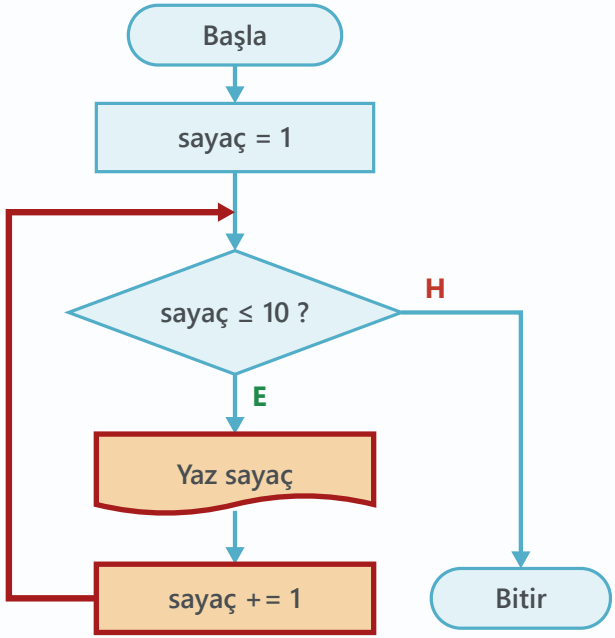
ÇIKTI

```
1 2 3 4 5 6 7 8
```

Koşul: `9 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
▶             System.out.print(sayac + " ");
▶             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 10`

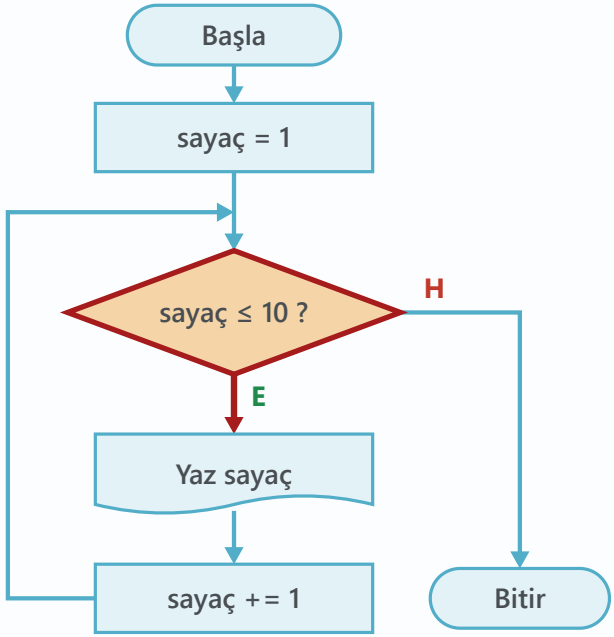
ÇIKTI

```
1 2 3 4 5 6 7 8 9
```

9 yazılır, `sayac` 10 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
          while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 10`

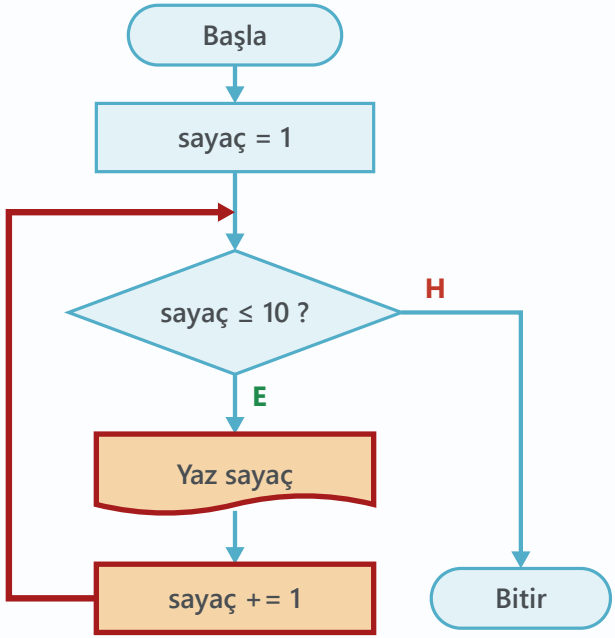
ÇIKTI

```
1 2 3 4 5 6 7 8 9
```

Koşul: `10 <= 10` **doğru** → Evet, döngü gövdesine girilir.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
▶             System.out.print(sayac + " ");
▶             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 11`

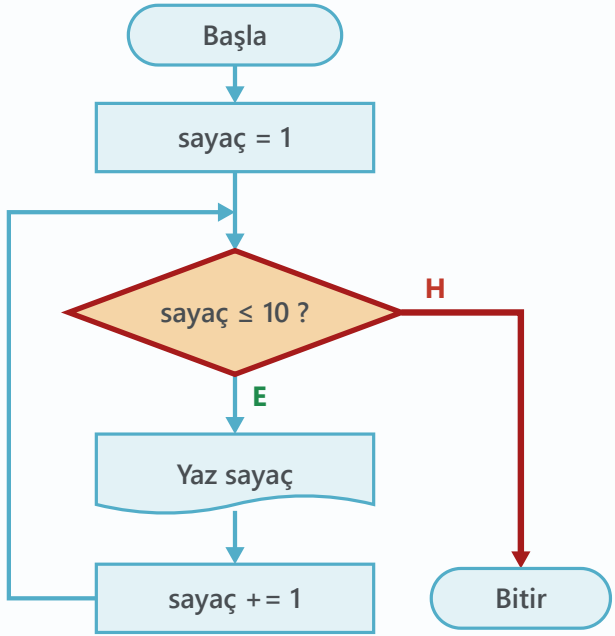
ÇIKTI

```
1 2 3 4 5 6 7 8 9
10
```

10 yazılır, `sayac` 11 olur; geri okla koşula dönülür.

Diyagramdan Java'ya: while Döngüsü

```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
▶         while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
8     }
9 }
```



DEĞİŞKENLER

`sayac = 11`

ÇIKTI

```
1 2 3 4 5 6 7 8 9
10
```

Koşul: `11 <= 10` **yanlış** → Hayır, döngüden çıkılır.

Diyagramdan Java'ya: while Döngüsü

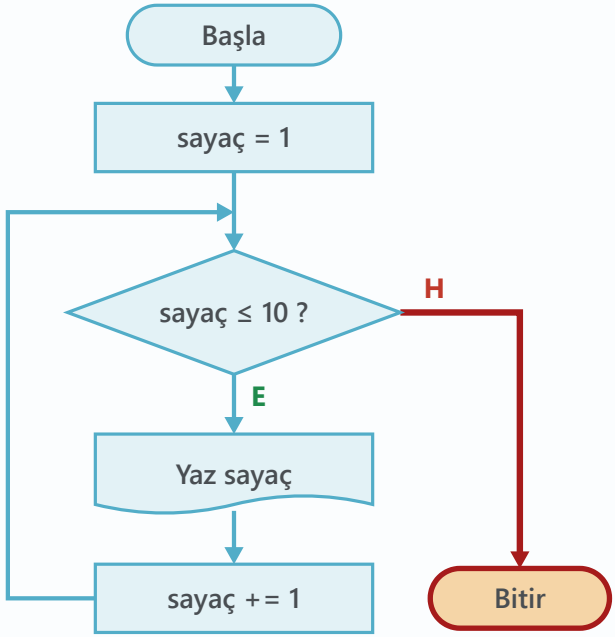
```
1 public class BirdenOna {
2     public static void main(String[] args)
  {
3         int sayac = 1;
4         while (sayac <= 10) {
5             System.out.print(sayac + " ");
6             sayac += 1;
7         }
▶     }
9 }
```

DEĞİŞKENLER

`sayac = 11`

ÇIKTI

```
1 2 3 4 5 6 7 8 9
10
```



Bitir. Gövde 10 kez, koşul 11 kez çalıştı.

TEKRARLI AKIŞ

Sınır Koşulu: < mı, ≤ mü?

Döngü koşulundaki tek bir karakter, tekrar sayısını değiştirir. Sayaç 1'den başlıyor ve her turda 1 artıyor:

Koşul	Gövde kaç kez?	Çıktı	1'den 10'a yazar mı?
`sayac < 10`	9	`1 2 3 4 5 6 7 8 9`	Hayır: 10 eksik
`sayac <= 10`	10	`1 2 3 4 5 6 7 8 9 10`	**Evet**
`sayac < 11`	10	`1 2 3 4 5 6 7 8 9 10`	**Evet** (aynı anlam)
`sayac <= 11`	11	`1 2 … 10 11`	Hayır: 11 fazla

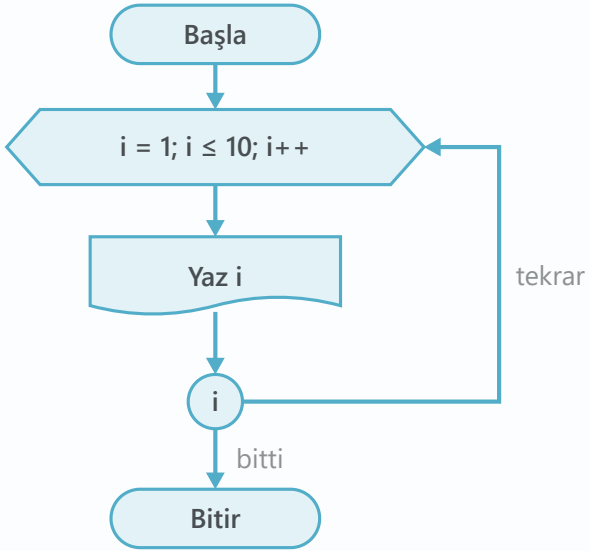
Bu hataya **bir eksik / bir fazla hatası** (off-by-one error) denir. Önlemek için koşul **son değerle** sınanır: sayaç = 10 iken koşul doğru mu? "1'den 10'a **kadar**" 10'u da kapsadığı için koşul `sayac ≤ 10` olmalıdır.

TEKRARLI AKIŞ

Döngüye Özel Akış (Döngü Sembolü)

Aynı problem: 1'den 10'a kadar sayıları yazdırma.

1. Başla
2. Döngü: 1'den 10'a kadar
3. Döngü değişkenini yaz
4. Döngü sonu
5. Bitir



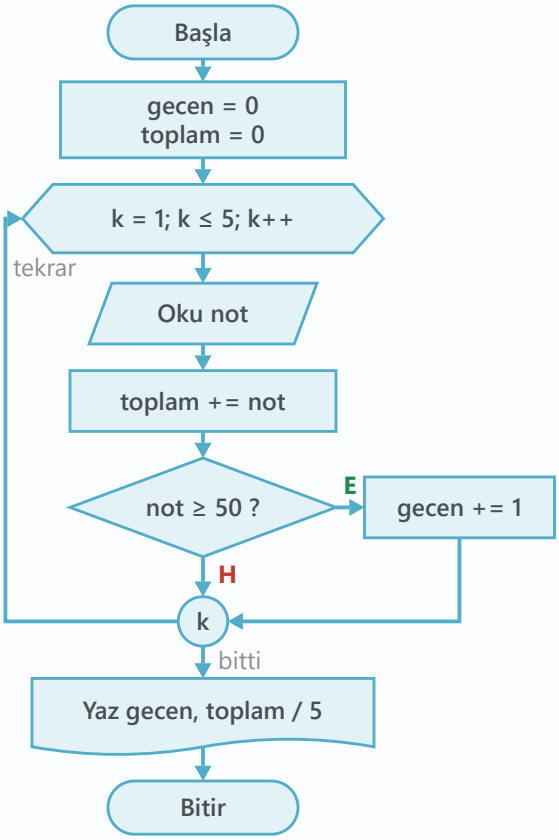
Altıgen sembol döngünün üç parçasını tek kutuda toplar:

- **başlangıç:** `i = 1` (bir kez)
- **koşul:** `i ≤ 10` (her turdan önce)
- **güncelleme:** `i++`, yani i'yi 1 artır (her turdan sonra)

"i" yazılı küçük çember **döngü sonudur**: akış oradan altıgene döner, döngü bitince Bitir'e geçer.

TEKRARLI AKIŞ

Döngü İçinde Karar: Sayaç ve Biriktirici



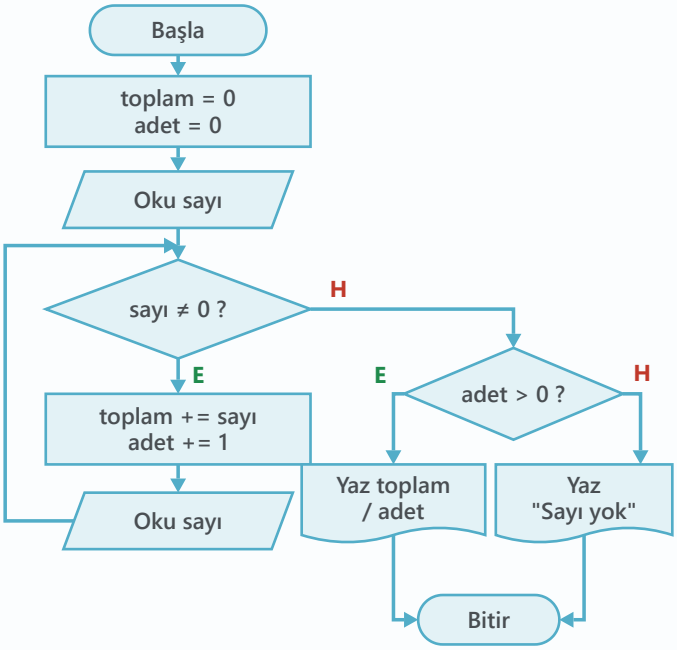

```
int gecen = 0, toplam = 0;
for (int k = 1; k <= 5; k++) {
  int not = giris.nextInt();
  toplam += not;
  if (not >= 50) {
    gecen++;
  }
}
System.out.println(gecen + " geçti, ortalama "
    + toplam / 5.0);
```

Girdi 45 70 50 30 88 → **Çıktı** `3 geçti, ortalama 56.6`

- **Sayaç** (`gecen`): koşul sağlandığında 1 artar. **Biriktirici** (`toplam`): her turda değer eklenir.
- Karar döngünün **içindedir**: her turda yeni okunan notla yeniden sınanır. Hayır kolu sayacı atlayıp döngü sonuna gider.
- 50 sınırda: `>=` olduğu için 50 alan geçer; `>` yazılsaydı sonuç 2 olurdu.
- `toplam / 5.0` : `5` yazılsaydı tam sayı bölmesi 56 verirdi.
- Üçüncü kalıp **bayrak**: `boolean` bir değişken, bir olayın olup olmadığını tutar (`bulundu = true`).

TEKRARLI AKIŞ

Bekçi Değerli Döngü



- **Bekçi değer** (sentinel value): girdinin bittiğini bildiren, veri olarak sayılmayan özel değer (0, −1, "son").
- Kaç değer geleceği önceden bilinmez: döngü koşula bağlıdır, Java'da `while`.
- **Önce okuma:** ilk değer döngüden önce okunur, sıradaki değer gövdenin **sonunda** okunur. Böylece bekçi gelince koşul hemen sınanır ve bekçi toplama katılmaz.
- Bekçi, gerçek veride olamayacak bir değer seçilir: notlar için −1 uygundur, 0 değildir (0 alan öğrenci olabilir).
- Döngüden sonra `adet = 0` durumu ayrıca sınanır: hiç sayı girilmezse `toplam / adet` sıfıra bölme olur (tam sayıda `ArithmeticException`, `double`'da `NaN`).

Bekçi Değer: Kod İzleme

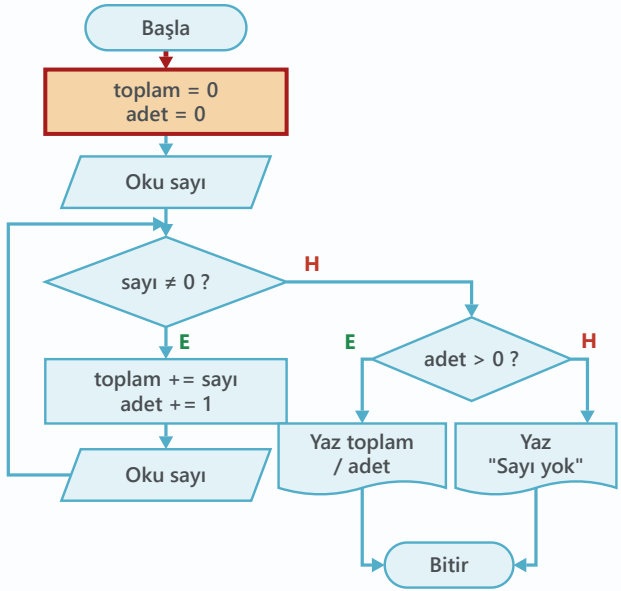
```
1  int toplam = 0, adet = 0;
2  int sayi = giris.nextInt();
3  while (sayi != 0) {
4    toplam += sayi;
5    adet++;
6    sayi = giris.nextInt();
7  }
8  if (adet > 0) {
9    System.out.println("Ortalama: "
10       + (double) toplam / adet);
11 } else {
12   System.out.println("Sayı yok");
13 }
```

DEĞİŞKENLER

`toplam = 0`

`adet = 0`

ÇIKTI



toplam = 0, adet = 0

Biriktirici ve sayaç sıfırlanır.

Bekçi Değer: Kod İzleme

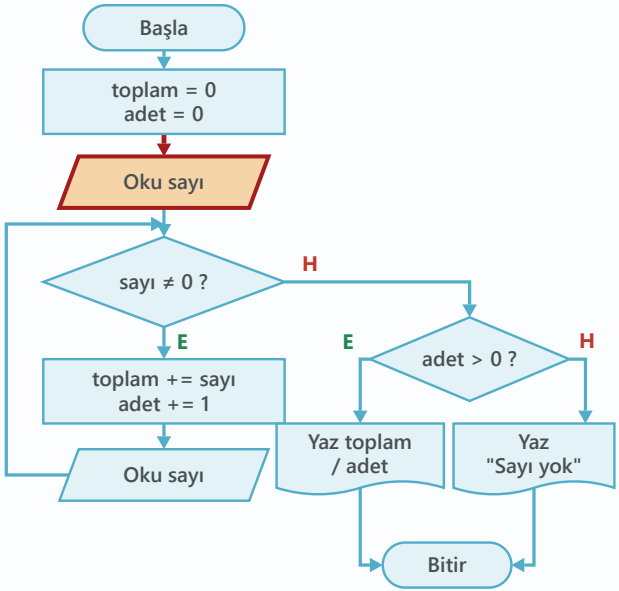
```
1  int toplam = 0, adet = 0;
▶ 2  int sayi = giris.nextInt();
3  while (sayi != 0) {
4    toplam += sayi;
5    adet++;
6    sayi = giris.nextInt();
7  }
8  if (adet > 0) {
9    System.out.println("Ortalama: "
10       + (double) toplam / adet);
11 } else {
12   System.out.println("Sayı yok");
13 }
```

DEĞİŞKENLER

- `toplam = 0`
- `adet = 0`
- `sayi = 12`

ÇIKTI

```
12
```



sayı = 12

Önce okuma: ilk sayı döngüden önce okunur. Girdi: 12 7 5 0.

Bekçi Değer: Kod İzleme

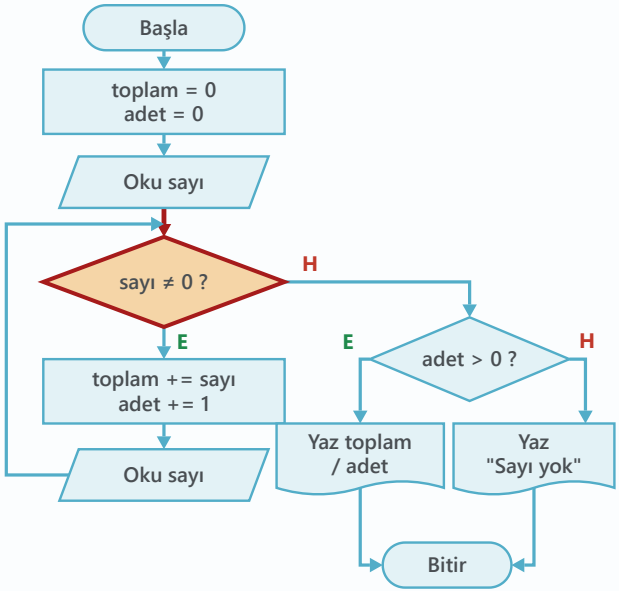
```
1  int toplam = 0, adet = 0;
2  int sayi = giris.nextInt();
▶  while (sayi != 0) {
4    toplam += sayi;
5    adet++;
6    sayi = giris.nextInt();
7  }
8  if (adet > 0) {
9    System.out.println("Ortalama: "
10       + (double) toplam / adet);
11 } else {
12   System.out.println("Sayı yok");
13 }
```

DEĞİŞKENLER

- `toplam = 0`
- `adet = 0`
- `sayi = 12`

ÇIKTI

```
12
```



12 ≠ 0 → E

Koşul: 12 ≠ 0 **doğru** → Evet, gövdeye girilir.

Bekçi Değer: Kod İzleme

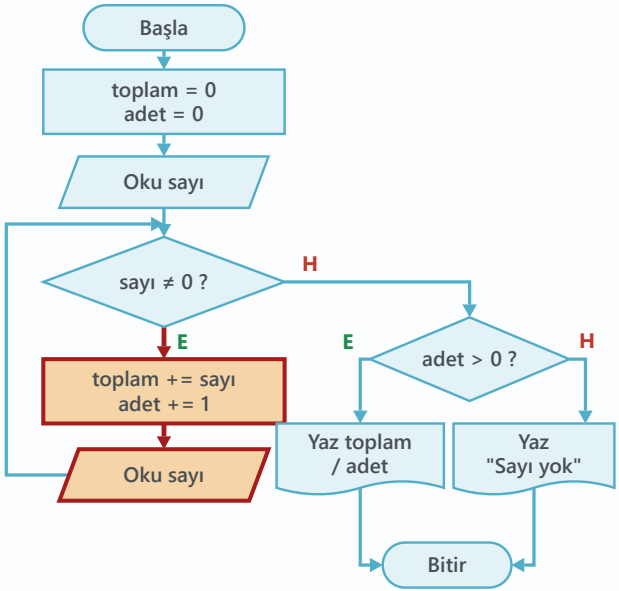
```
1 int toplam = 0, adet = 0;
2 int sayi = giris.nextInt();
3 while (sayi != 0) {
    toplam += sayi;
    adet++;
    sayi = giris.nextInt();
7 }
8 if (adet > 0) {
9   System.out.println("Ortalama: "
10      + (double) toplam / adet);
11 } else {
12   System.out.println("Sayı yok");
13 }
```

DEĞİŞKENLER

- `toplam = 12`
- `adet = 1`
- `sayi = 7`

ÇIKTI

```
12
7
```



toplam = 12, adet = 1, sayı = 7

`toplam` = 12, `adet` = 1; gövdenin sonunda sıradaki sayı okunur: 7.

Bekçi Değer: Kod İzleme

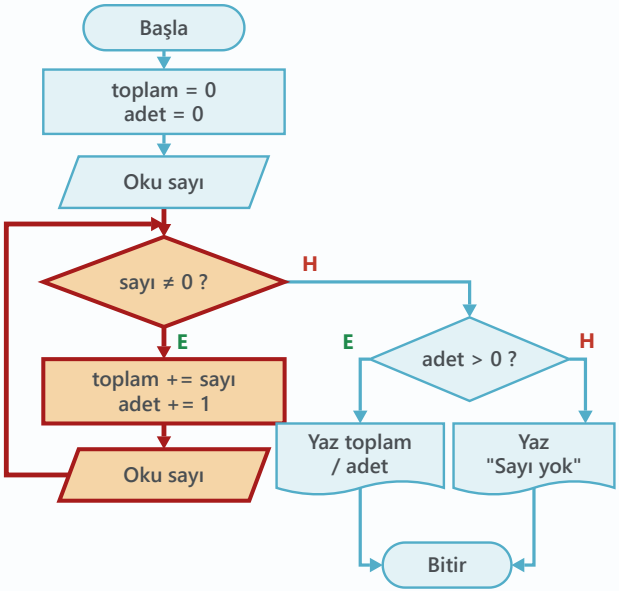
```
1  int toplam = 0, adet = 0;
2  int sayi = giris.nextInt();
▶  while (sayi != 0) {
▶    toplam += sayi;
▶    adet++;
▶    sayi = giris.nextInt();
7  }
8  if (adet > 0) {
9    System.out.println("Ortalama: "
10       + (double) toplam / adet);
11 } else {
12   System.out.println("Sayı yok");
13 }
```

DEĞİŞKENLER

- `toplam = 19`
- `adet = 2`
- `sayi = 5`

ÇIKTI

```
12
7
5
```



7 ≠ 0 → toplam = 19, adet = 2

Geri okla karara dönülür: 7 ≠ 0. `toplam` = 19, `adet` = 2; okunan 5.

Bekçi Değer: Kod İzleme

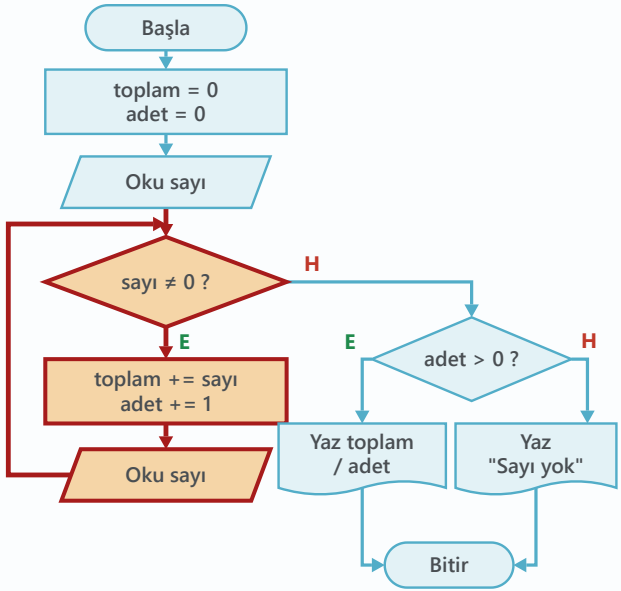
```
1  int toplam = 0, adet = 0;
2  int sayi = giris.nextInt();
   while (sayi != 0) {
     toplam += sayi;
     adet++;
     sayi = giris.nextInt();
7  }
8  if (adet > 0) {
9    System.out.println("Ortalama: "
10       + (double) toplam / adet);
11 } else {
12   System.out.println("Sayı yok");
13 }
```

DEĞİŞKENLER

- `toplam = 24`
- `adet = 3`
- `sayi = 0`

ÇIKTI

```
12
7
5
0
```



5 ≠ 0 → toplam = 24, adet = 3

5 ≠ 0: `toplam` = 24, `adet` = 3; okunan 0, yani bekçi.

Bekçi Değer: Kod İzleme

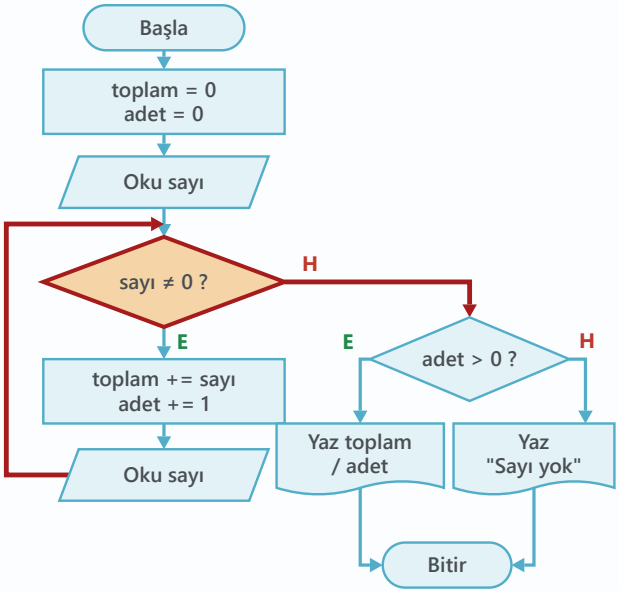
```
1  int toplam = 0, adet = 0;
2  int sayi = giris.nextInt();
▶ while (sayi != 0) {
4    toplam += sayi;
5    adet++;
6    sayi = giris.nextInt();
7  }
8  if (adet > 0) {
9    System.out.println("Ortalama: "
10       + (double) toplam / adet);
11 } else {
12   System.out.println("Sayı yok");
13 }
```

DEĞİŞKENLER

- `toplam = 24`
- `adet = 3`
- `sayi = 0`

ÇIKTI

```
12
7
5
0
```



0 ≠ 0 → H

0 ≠ 0 **yanlış** → Hayır: döngüden çıkılır. 0 toplama hiç katılmadı.

Bekçi Değer: Kod İzleme

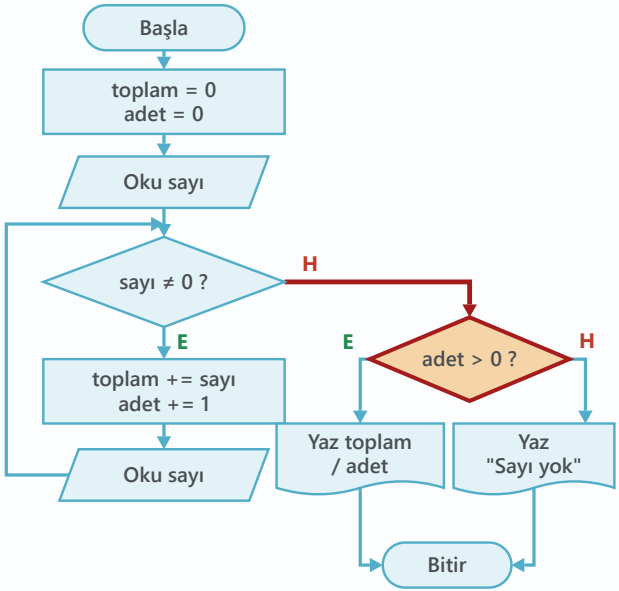
```
1  int toplam = 0, adet = 0;
2  int sayi = giris.nextInt();
3  while (sayi != 0) {
4    toplam += sayi;
5    adet++;
6    sayi = giris.nextInt();
7  }
▶ if (adet > 0) {
9    System.out.println("Ortalama: "
10       + (double) toplam / adet);
11 } else {
12   System.out.println("Sayı yok");
13 }
```

DEĞİŞKENLER

- `toplam = 24`
- `adet = 3`
- `sayi = 0`

ÇIKTI

```
12
7
5
0
```



3 > 0 → E

Döngü sonrası denetim: `adet` = 3 > 0 → Evet.

Bekçi Değer: Kod İzleme

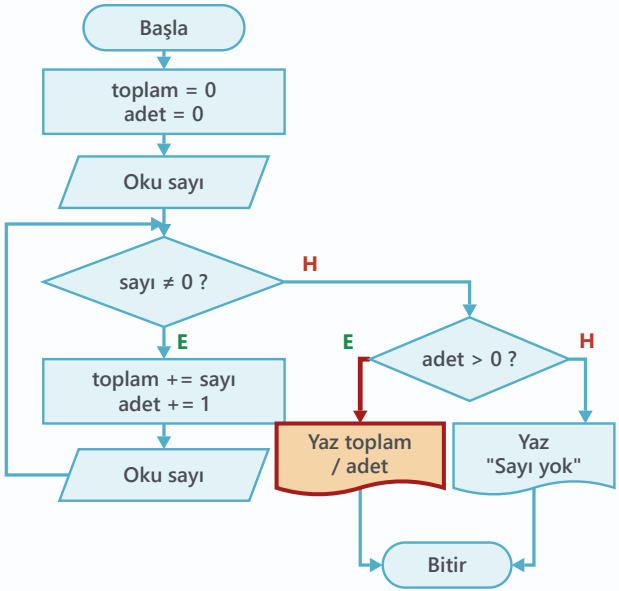
```
1  int toplam = 0, adet = 0;
2  int sayi = giris.nextInt();
3  while (sayi != 0) {
4    toplam += sayi;
5    adet++;
6    sayi = giris.nextInt();
7  }
8  if (adet > 0) {
▶    System.out.println("Ortalama: "
▶        + (double) toplam / adet);
11 } else {
12   System.out.println("Sayı yok");
13 }
```

DEĞİŞKENLER

- `toplam = 24`
- `adet = 3`
- `sayi = 0`

ÇIKTI

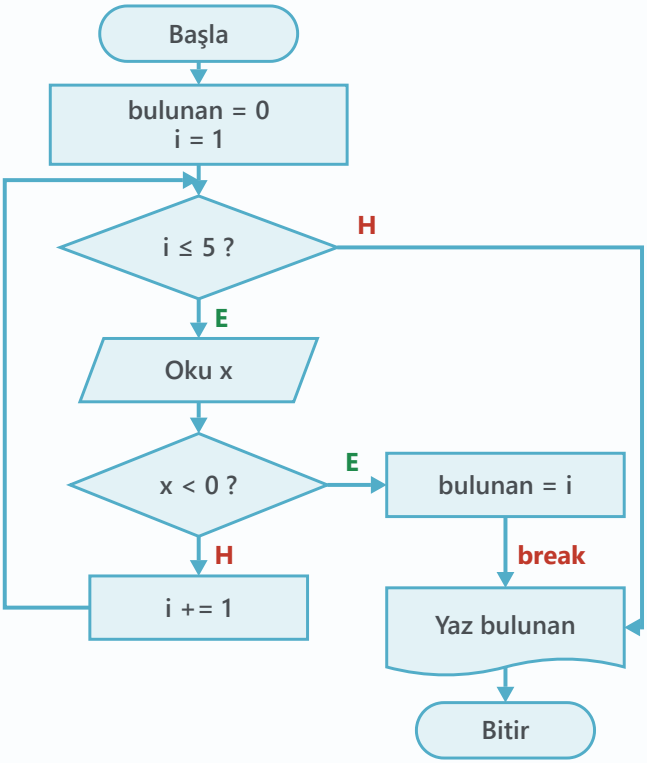
```
12
7
5
0
Ortalama: 8.0
```



24 / 3 = 8.0

24 / 3 = 8.0 yazılır. İlk sayı olarak 0 girilseydi Hayır kolundan `Sayı yok` yazılırdı.

Döngüden Erken Çıkış



- Aramada aranan bulununca kalan turlara gerek yoktur: döngünün içindeki **ikinci bir karar** akışı döngünün dışına çıkarır.
- Diyagramda döngünün iki çıkışı vardır: koşul yanlışlaşınca (bulunamadı) ve aranan bulununca. İki yol **Yaz bulunan** kutusunda birleşir.
- Java'da ikinci çıkış `break` 'tir; yalnızca içinde bulunduğu döngüyü bitirir.
- `bulunan = 0` "bulunamadı" demektir; sıra 1'den başladığı için gerçek bir sırayla karışmaz.
- Tek çıkışlı, yapısal karşılığı bir **bayrakla** yazılır:

```
int bulunan = 0;
boolean bulundu = false;
for (int i = 1; i <= 5 && !bulundu; i++) {
  int x = giris.nextInt();
  if (x < 0) {
    bulunan = i;
    bulundu = true;
  }
}
```

Erken Çıkış: Kod İzleme

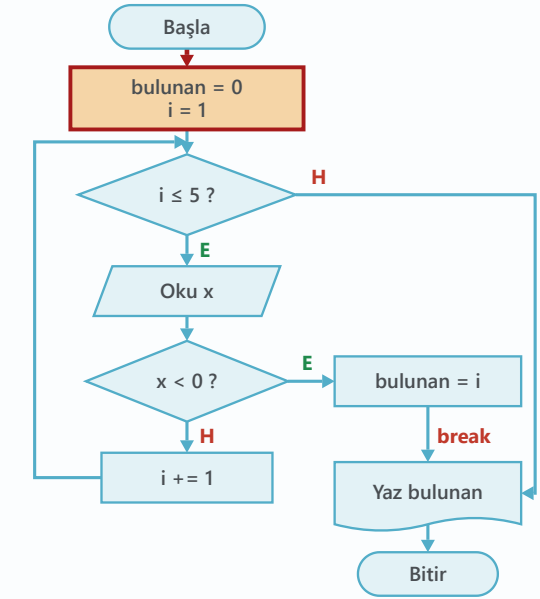
```
1  int bulunan = 0;
2  for (int i = 1; i <= 5; i++) {
3    int x = giris.nextInt();
4    if (x < 0) {
5      bulunan = i;
6      break;
7    }
8  }
9  System.out.println("Sıra: " + bulunan);
```

DEĞİŞKENLER

`bulunan = 0`

`i = 1`

ÇIKTI



bulunan = 0, i = 1

`bulunan` = 0 (henüz bulunamadı), `i` = 1. Girdi: 4 9 −2 6 1.

Erken Çıkış: Kod İzleme

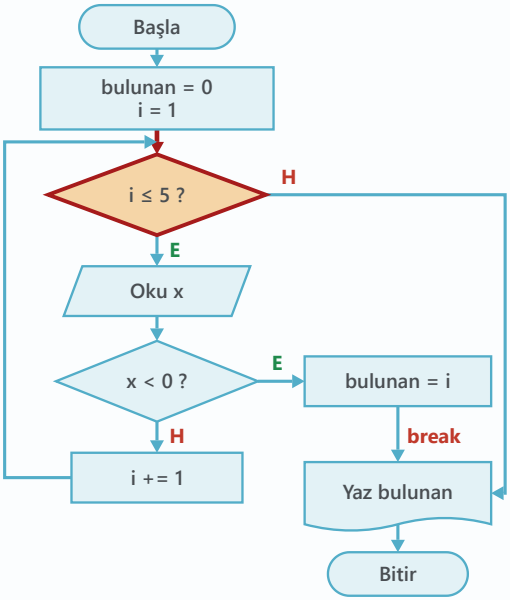
```
1 int bulunan = 0;
▶ for (int i = 1; i <= 5; i++) {
3   int x = giris.nextInt();
4   if (x < 0) {
5     bulunan = i;
6     break;
7   }
8 }
9 System.out.println("Sıra: " + bulunan);
```

DEĞİŞKENLER

`bulunan = 0`

`i = 1`

ÇIKTI

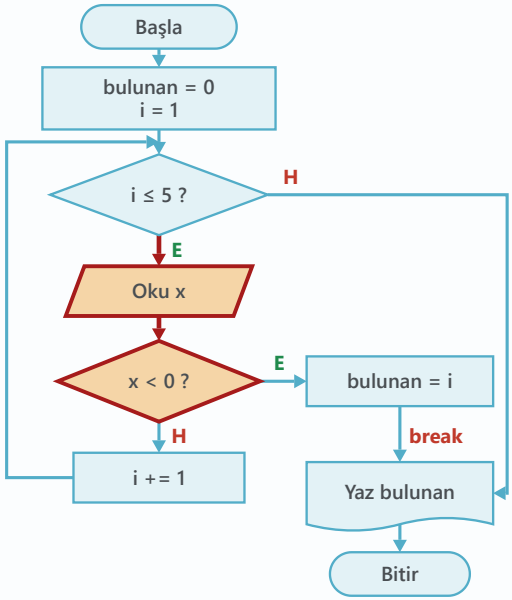


1 ≤ 5 → E

1 ≤ 5 → Evet, gövdeye girilir.

Erken Çıkış: Kod İzleme

```
1 int bulunan = 0;
2 for (int i = 1; i <= 5; i++) {
▶   int x = giris.nextInt();
▶   if (x < 0) {
5     bulunan = i;
6     break;
7   }
8 }
9 System.out.println("Sıra: " + bulunan);
```



DEĞİŞKENLER

`bulunan = 0`

`i = 1` `x = 4`

ÇIKTI

```
4
```

x = 4: 4 < 0 → H

`x` = 4; 4 < 0 yanlış → Hayır.

Erken Çıkış: Kod İzleme

```
1 int bulunan = 0;
▶ for (int i = 1; i <= 5; i++) {
3   int x = giris.nextInt();
4   if (x < 0) {
5     bulunan = i;
6     break;
7   }
8 }
9 System.out.println("Sıra: " + bulunan);
```

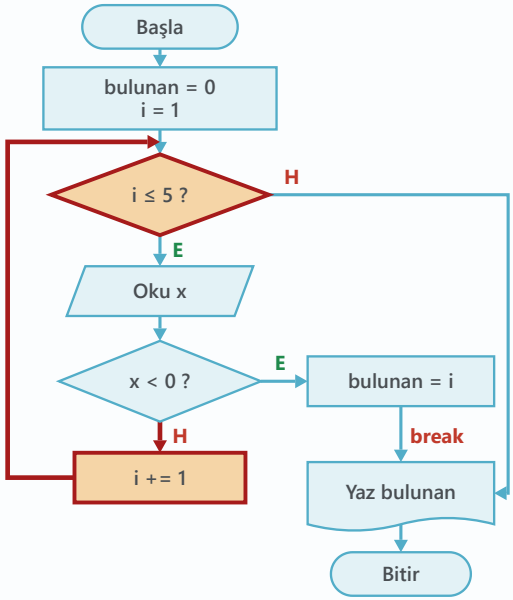
DEĞİŞKENLER

`bulunan = 0`

`i = 2`

ÇIKTI

```
4
```



i = 2: 2 ≤ 5 → E

`i` = 2; geri okla karara dönülür: 2 ≤ 5 → Evet.

Erken Çıkış: Kod İzleme

```
1 int bulunan = 0;
▶ for (int i = 1; i <= 5; i++) {
▶   int x = giris.nextInt();
▶   if (x < 0) {
5     bulunan = i;
6     break;
7   }
8 }
9 System.out.println("Sıra: " + bulunan);
```

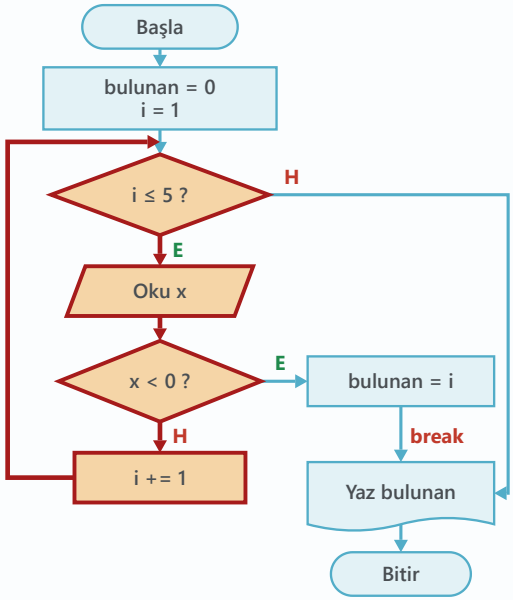
DEĞİŞKENLER

`bulunan = 0`

`i = 3`

ÇIKTI

```
4
9
```



x = 9: H; i = 3: E

`x` = 9 yine negatif değil; `i` = 3, 3 ≤ 5 → Evet.

Erken Çıkış: Kod İzleme

```
1 int bulunan = 0;
2 for (int i = 1; i <= 5; i++) {
▶   int x = giris.nextInt();
▶   if (x < 0) {
5     bulunan = i;
6     break;
7   }
8 }
9 System.out.println("Sıra: " + bulunan);
```

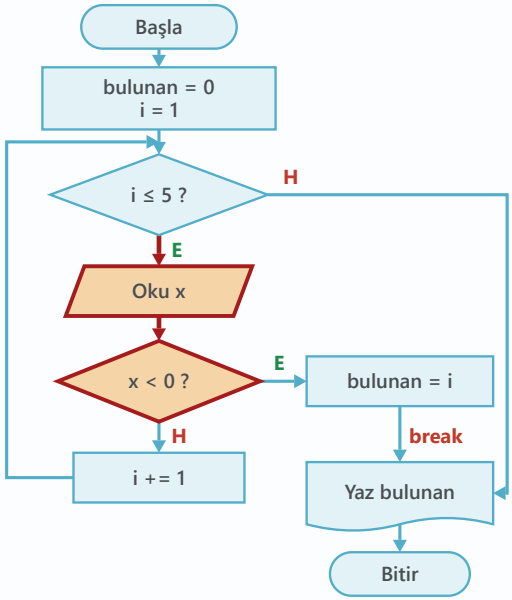
DEĞİŞKENLER

`bulunan = 0`

`i = 3` `x = −2`

ÇIKTI

```
4
9
-2
```



x = −2: −2 < 0 → E

`x` = −2; −2 < 0 **doğru** → Evet kolu.

Erken Çıkış: Kod İzleme

```
1 int bulunan = 0;
2 for (int i = 1; i <= 5; i++) {
3   int x = giris.nextInt();
4   if (x < 0) {
      bulunan = i;
      break;
7   }
8 }
9 System.out.println("Sıra: " + bulunan);
```

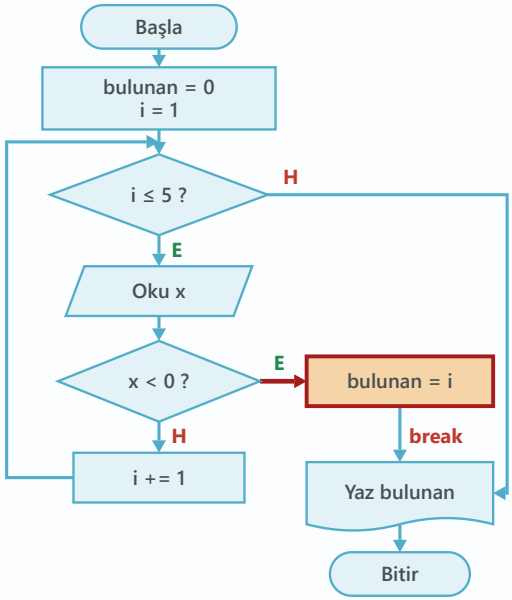
DEĞİŞKENLER

`bulunan = 3`

`i = 3`

ÇIKTI

```
4
9
-2
```



bulunan = 3

`bulunan` = 3. `break` : `i += 1` ve koşul atlanır, döngü hemen biter; 6 ve 1 hiç okunmaz.

Erken Çıkış: Kod İzleme

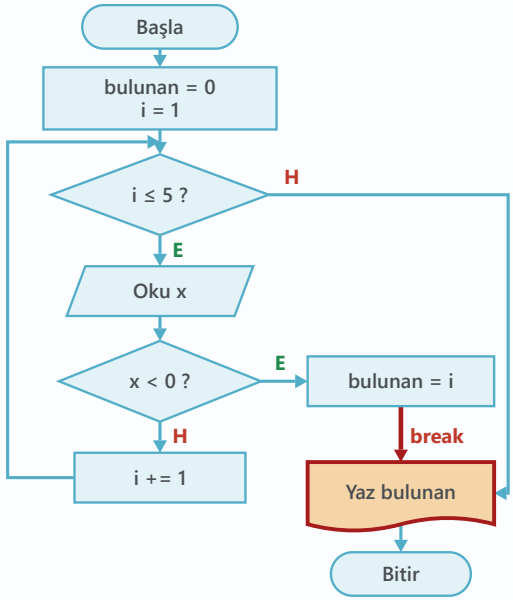
```
1 int bulunan = 0;
2 for (int i = 1; i <= 5; i++) {
3   int x = giris.nextInt();
4   if (x < 0) {
5     bulunan = i;
6     break;
7   }
8 }
▶ System.out.println("Sıra: " + bulunan);
```

DEĞİŞKENLER

`bulunan = 3`

ÇIKTI

```
4
9
-2
Sıra: 3
```



Sıra: 3

Sonuç yazılır. Negatif sayı hiç girilmeseydi döngü `i` = 6'da Hayır'dan çıkar, `Sıra: 0` yazılırdı.

İç İçe Döngü: Kod İzleme

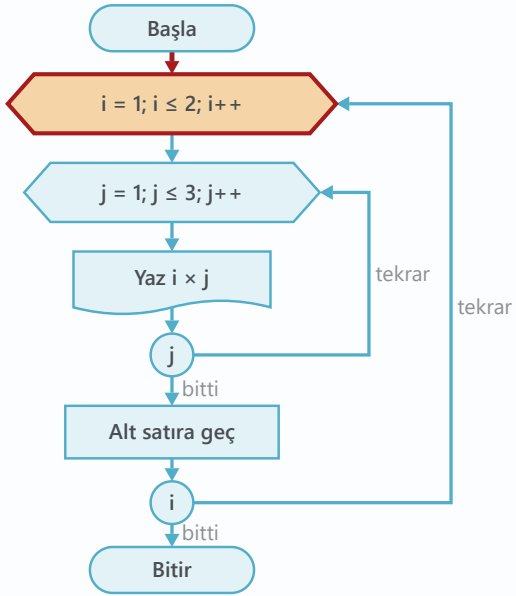
```
1 for (int i = 1; i <= 2; i++) {
2   for (int j = 1; j <= 3; j++) {
3     System.out.print(i * j + " ");
4   }
5   System.out.println();
6 }
```

DEĞİŞKENLER

`i = 1`

ÇIKTI





i = 1

Dış döngü başlar: `i` = 1; 1 ≤ 2 doğru.

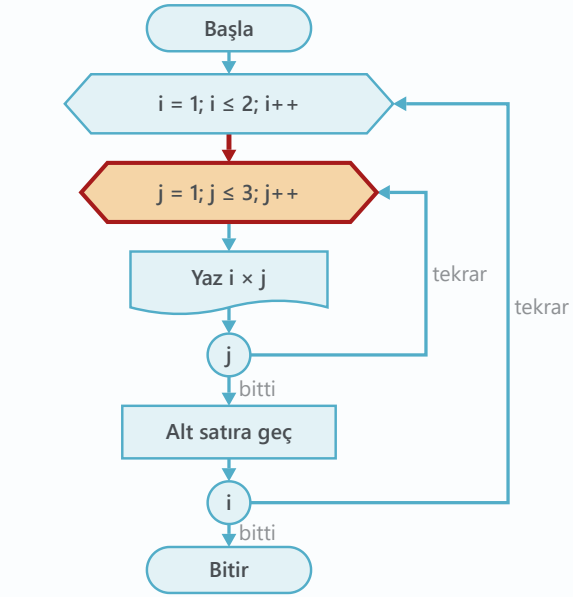
İç İçe Döngü: Kod İzleme

```
1 for (int i = 1; i <= 2; i++) {
2   for (int j = 1; j <= 3; j++) {
3     System.out.print(i * j + " ");
4   }
5   System.out.println();
6 }
```

DEĞİŞKENLER

`i = 1` `j = 1`

ÇIKTI



j = 1

İç döngü başlar: `j` = 1.

İç İçe Döngü: Kod İzleme

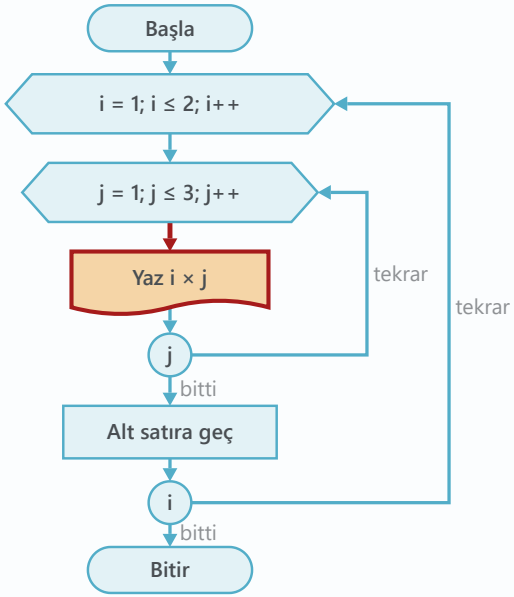
```
1 for (int i = 1; i <= 2; i++) {
2   for (int j = 1; j <= 3; j++) {
▶     System.out.print(i * j + " ");
4   }
5   System.out.println();
6 }
```

DEĞİŞKENLER

`i = 1` `j = 1`

ÇIKTI

```
1
```



1 × 1 = 1 yazılır.

İç İçe Döngü: Kod İzleme

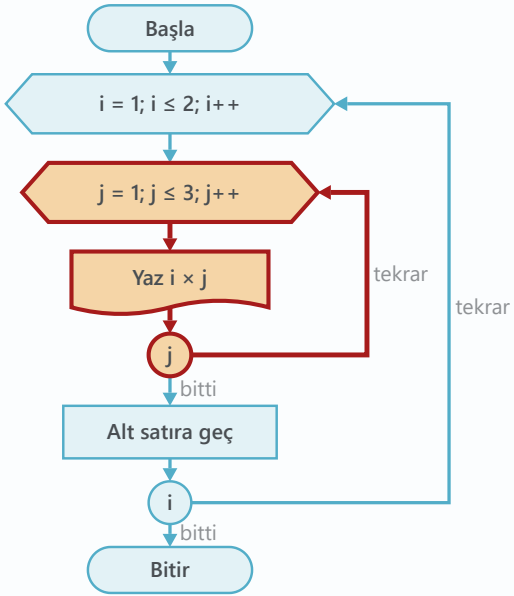
```
1 for (int i = 1; i <= 2; i++) {
▶   for (int j = 1; j <= 3; j++) {
▶     System.out.print(i * j + " ");
4   }
5   System.out.println();
6 }
```

DEĞİŞKENLER

`i = 1` `j = 2`

ÇIKTI

```
1 2
```

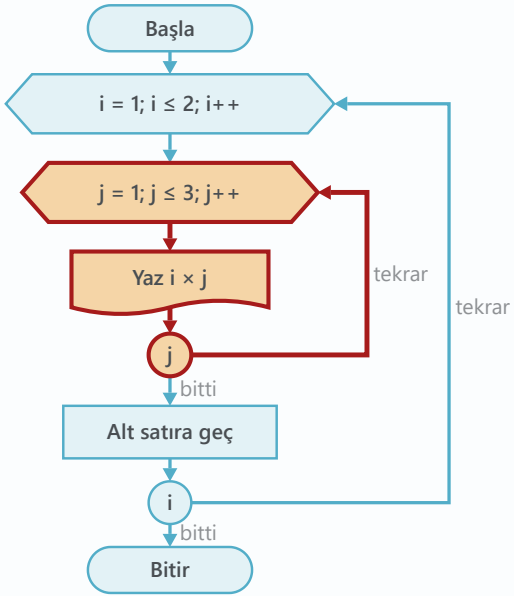


j = 2: 1 × 2 = 2

Döngü sonu **j**: tekrar okuyla iç döngüye dönülür. `j` = 2, 2 yazılır.

İç İçe Döngü: Kod İzleme

```
1 for (int i = 1; i <= 2; i++) {
▶   for (int j = 1; j <= 3; j++) {
▶     System.out.print(i * j + " ");
4   }
5   System.out.println();
6 }
```



DEĞİŞKENLER

`i = 1` `j = 3`

ÇIKTI

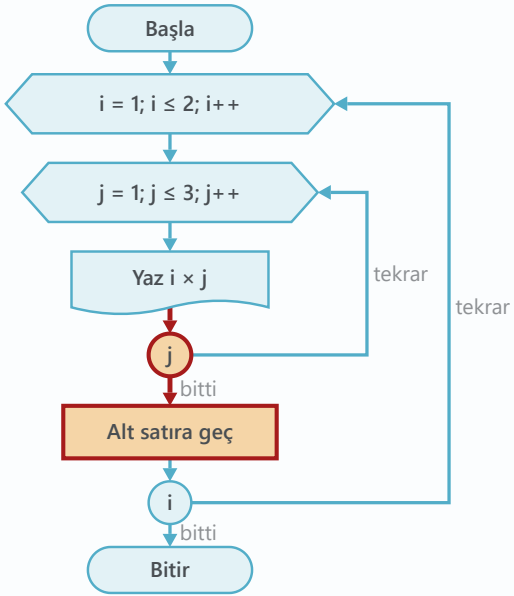
```
1 2 3
```

j = 3: 1 × 3 = 3

`j` = 3, 3 yazılır.

İç İçe Döngü: Kod İzleme

```
1 for (int i = 1; i <= 2; i++) {
▶   for (int j = 1; j <= 3; j++) {
3     System.out.print(i * j + " ");
4   }
▶   System.out.println();
6 }
```



DEĞİŞKENLER

`i = 1`

ÇIKTI

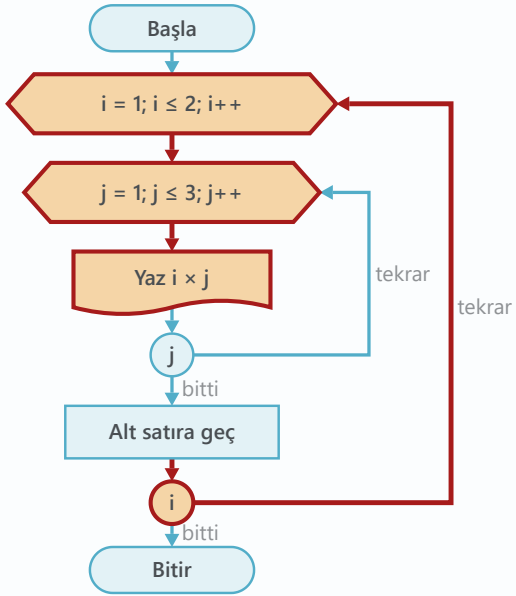
```
1 2 3
```

j = 4: 4 ≤ 3 → bitti

`j` = 4: 4 ≤ 3 yanlış, iç döngü biter (bitti oku).
`println` alt satıra geçer.

İç İçe Döngü: Kod İzleme

```
for (int i = 1; i <= 2; i++) {
  for (int j = 1; j <= 3; j++) {
    System.out.print(i * j + " ");
  }
  System.out.println();
}
```



DEĞİŞKENLER

`i = 2` `j = 1`

ÇIKTI

```
1 2 3
2
```

i = 2, j = 1: 2

Döngü sonu **i**: dış döngüye dönülür, `i` = 2. İç döngü **yeniden** `j` = 1'den başlar: 2 yazılır.

İç İçe Döngü: Kod İzleme

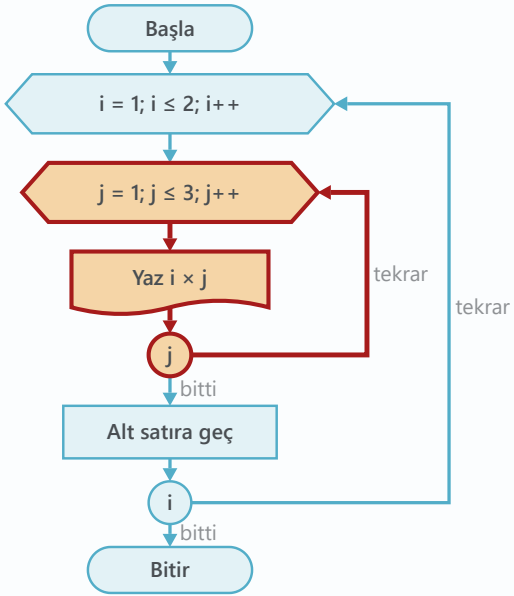
```
1 for (int i = 1; i <= 2; i++) {
▶   for (int j = 1; j <= 3; j++) {
▶     System.out.print(i * j + " ");
4   }
5   System.out.println();
6 }
```

DEĞİŞKENLER

`i = 2` `j = 3`

ÇIKTI

```
1 2 3
2 4 6
```



j = 2, 3: 4 ve 6

`j` = 2 ve 3: 4 ve 6 yazılır.

İç İçe Döngü: Kod İzleme

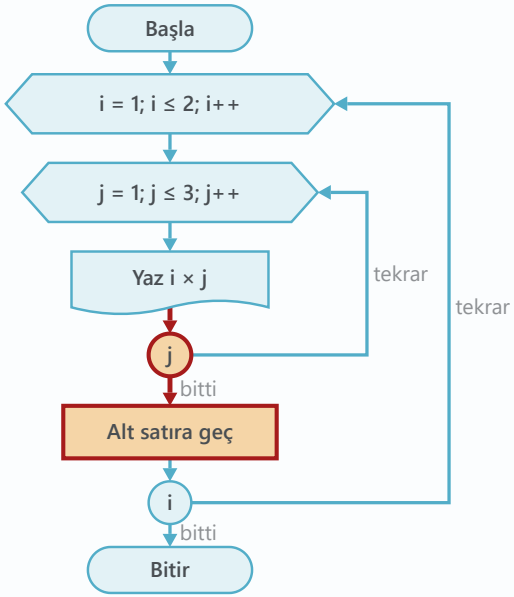
```
1 for (int i = 1; i <= 2; i++) {
▶   for (int j = 1; j <= 3; j++) {
3     System.out.print(i * j + " ");
4   }
▶   System.out.println();
6 }
```

DEĞİŞKENLER

`i = 2`

ÇIKTI

```
1 2 3
2 4 6
```

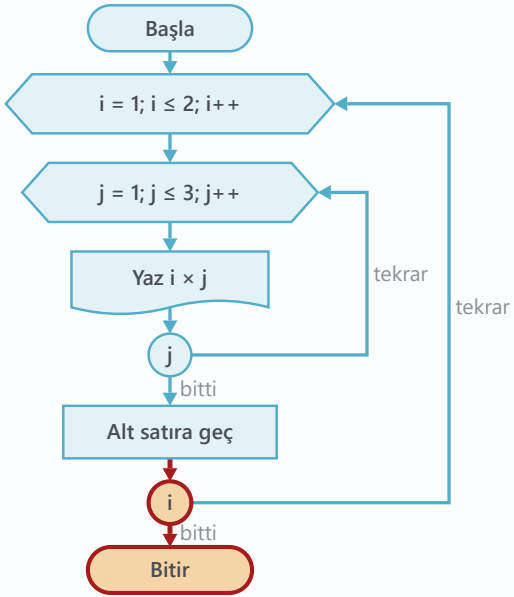


iç döngü bitti

İç döngü yine biter, alt satıra geçilir.

İç İçe Döngü: Kod İzleme

```
▶ for (int i = 1; i <= 2; i++) {
2   for (int j = 1; j <= 3; j++) {
3     System.out.print(i * j + " ");
4   }
5   System.out.println();
6 }
```



DEĞİŞKENLER

`i = 3`

ÇIKTI

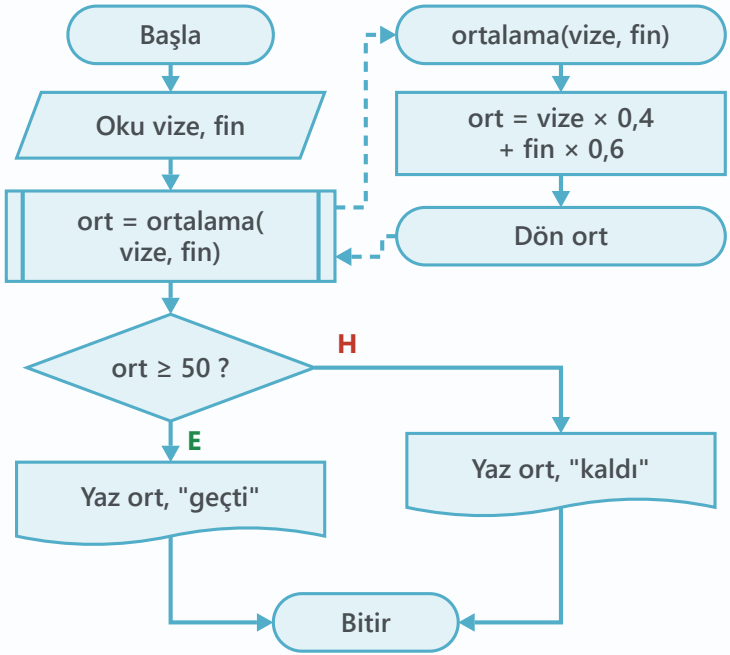
```
1 2 3
2 4 6
```

i = 3: 3 ≤ 2 → bitti

`i` = 3: dış döngü biter. İç gövde 2 × 3 = 6 kez, `println` 2 kez çalıştı.

ALT PROGRAM

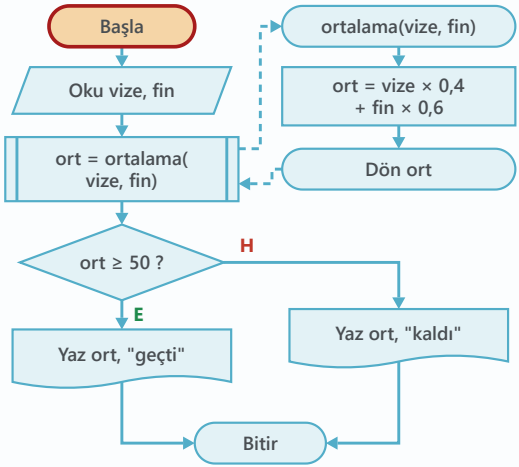
Alt Program Sembolü ve Metot



- Uzun bir diyagram anlamlı parçalara bölünür: bir parça **ayrı bir diyagram** olarak çizilir, ana diyagramda **alt program** sembolüyle çağrılır.
- Alt programın Başla kutusunda adı ve girdileri (**parametreleri**), sonunda **Dön** kutusu bulunur.
- Çağrıda akış alt programa geçer; Dön'e gelince sonuç çağırana verilir ve akış çağrının hemen sonrasından sürer (kesikli oklar).
- Java karşılığı **metottur**: `static double ortalama(int vize, int fin)`, çağrısı `ortalama(vize, fin)`. `final` Java'da anahtar sözcük olduğu için değişkenin adı `fin`.
- Kazanç: aynı adımlar yeniden çizilmez, her parça ayrı sınanır, ana diyagram kısa kalır.

Alt Program Çağrısı: Kod İzleme

```
1  static double ortalama(int vize, int fin) {
2    double ort = vize * 0.4 + fin * 0.6;
3    return ort;
4  }
▶ public static void main(String[] args) {
▶    Scanner giris = new Scanner(System.in);
7    int vize = giris.nextInt();
8    int fin = giris.nextInt();
9    double ort = ortalama(vize, fin);
10   if (ort >= 50) System.out.println(ort + " geçti");
11   else System.out.println(ort + " kaldı");
12 }
```



ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

—

ÇIKTI

main başlar

Program `main` 'den başlar (Başla).

Adım 1 / 8

Alt Program Çağrısı: Kod İzleme

```
1  static double ortalama(int vize, int fin) {
2    double ort = vize * 0.4 + fin * 0.6;
3    return ort;
4  }
5  public static void main(String[] args) {
6    Scanner giris = new Scanner(System.in);
▶    int vize = giris.nextInt();
▶    int fin = giris.nextInt();
9    double ort = ortalama(vize, fin);
10   if (ort >= 50) System.out.println(ort + " geçti");
11   else System.out.println(ort + " kaldı");
12 }
```

ÇAĞRI YIĞINI

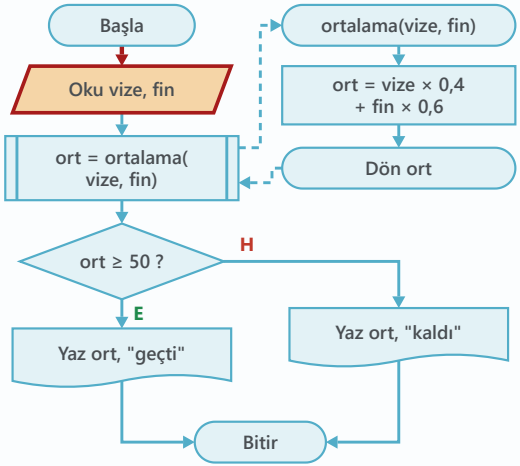
`main()`

DEĞİŞKENLER

`vize = 40` `fin = 70`

ÇIKTI

```
40 70
```



vize = 40, fin = 70

Kullanıcı aynı satıra 40 ve 70 girer.

Alt Program Çağrısı: Kod İzleme

```
▶ static double ortalama(int vize, int fin) {
2   double ort = vize * 0.4 + fin * 0.6;
3   return ort;
4 }
5 public static void main(String[] args) {
6   Scanner giris = new Scanner(System.in);
7   int vize = giris.nextInt();
8   int fin = giris.nextInt();
▶   double ort = ortalama(vize, fin);
10  if (ort >= 50) System.out.println(ort + " geçti");
11  else System.out.println(ort + " kaldı");
12 }
```

ÇAĞRI YIĞINI

`ortalama(40, 70)`

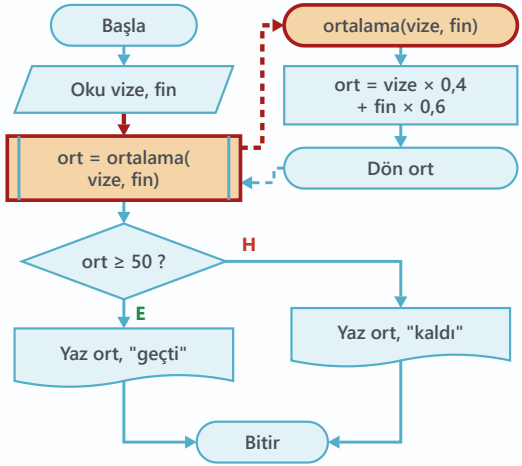
`main()`

DEĞİŞKENLER

`vize = 40` `fin = 70`

ÇIKTI

```
40 70
```



ortalama(40, 70) çağrılır

Alt program kutusu: `ortalama` çağrılır; 40 ve 70 parametrelere **kopyalanır**.

Alt Program Çağrısı: Kod İzleme

```
1  static double ortalama(int vize, int fin) {
▶    double ort = vize * 0.4 + fin * 0.6;
3    return ort;
4  }
5  public static void main(String[] args) {
6    Scanner giris = new Scanner(System.in);
7    int vize = giris.nextInt();
8    int fin = giris.nextInt();
9    double ort = ortalama(vize, fin);
10   if (ort >= 50) System.out.println(ort + " geçti");
11   else System.out.println(ort + " kaldı");
12 }
```

ÇAĞRI YIĞINI

`ortalama(40, 70)`

`main()`

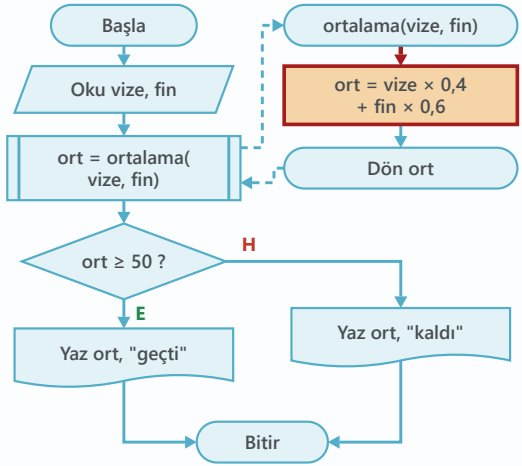
DEĞİŞKENLER

`vize = 40` `fin = 70`

`ort = 58.0`

ÇIKTI

```
40 70
```

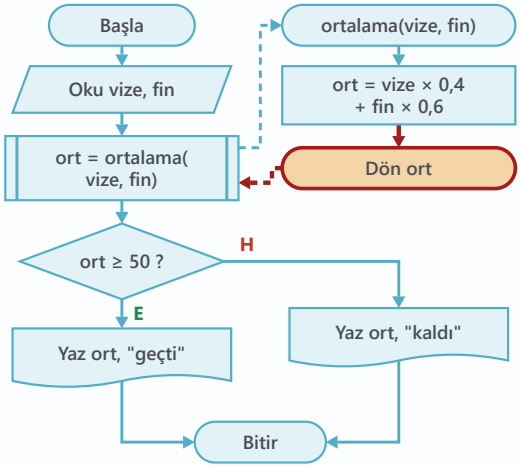


ort = 16 + 42 = 58.0

Alt programın kendi diyagramı çalışır: 16 + 42 = 58.0. Bu `ort`, main'deki `ort`'tan ayrı bir yerel değişkendir.

Alt Program Çağrısı: Kod İzleme

```
1  static double ortalama(int vize, int fin) {
2    double ort = vize * 0.4 + fin * 0.6;
▶    return ort;
4  }
5  public static void main(String[] args) {
6    Scanner giris = new Scanner(System.in);
7    int vize = giris.nextInt();
8    int fin = giris.nextInt();
▶    double ort = ortalama(vize, fin);
10   if (ort >= 50) System.out.println(ort + " geçti");
11   else System.out.println(ort + " kaldı");
12 }
```



Dön 58.0

Dön: değer çağırana verilir; `ortalama` 'nın çerçevesi yığından kalkar.

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`vize = 40` `fin = 70` `ort = 58.0`

ÇIKTI

```
40 70
```

Alt Program Çağrısı: Kod İzleme

```
1  static double ortalama(int vize, int fin) {
2    double ort = vize * 0.4 + fin * 0.6;
3    return ort;
4  }
5  public static void main(String[] args) {
6    Scanner giris = new Scanner(System.in);
7    int vize = giris.nextInt();
8    int fin = giris.nextInt();
9    double ort = ortalama(vize, fin);
▶    if (ort >= 50) System.out.println(ort + " geçti");
11   else System.out.println(ort + " kaldı");
12 }
```

ÇAĞRI YIĞINI

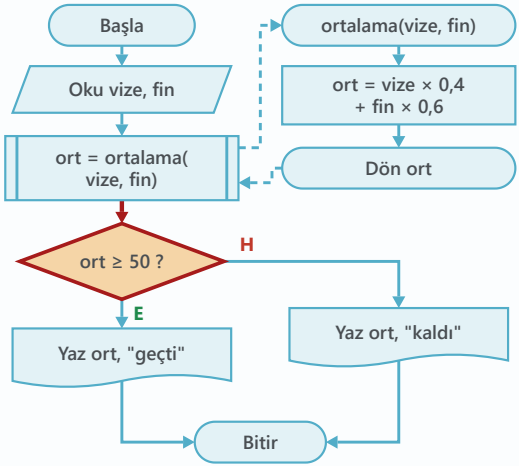
`main()`

DEĞİŞKENLER

`vize = 40` `fin = 70` `ort = 58.0`

ÇIKTI

```
40 70
```



58.0 ≥ 50 → E

Akış çağrının sonrasından sürer: 58.0 ≥ 50 **doğru** → Evet.

Alt Program Çağrısı: Kod İzleme

```
1  static double ortalama(int vize, int fin) {
2    double ort = vize * 0.4 + fin * 0.6;
3    return ort;
4  }
5  public static void main(String[] args) {
6    Scanner giris = new Scanner(System.in);
7    int vize = giris.nextInt();
8    int fin = giris.nextInt();
9    double ort = ortalama(vize, fin);
▶    if (ort >= 50) System.out.println(ort + " geçti");
11   else System.out.println(ort + " kaldı");
12 }
```

ÇAĞRI YIĞINI

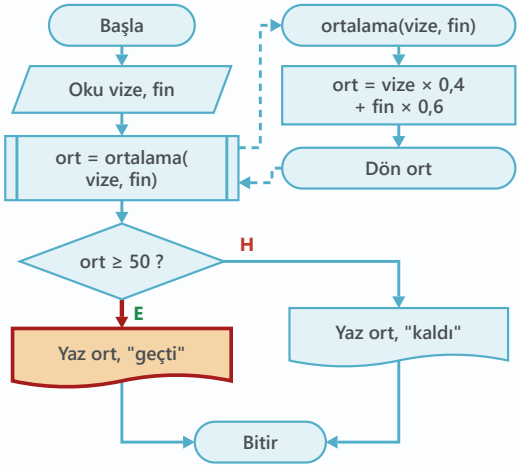
`main()`

DEĞİŞKENLER

`vize = 40` `fin = 70` `ort = 58.0`

ÇIKTI

```
40 70
58.0 geçti
```



58.0 geçti

Evet kolu yazdırır.

Alt Program Çağrısı: Kod İzleme

```
1  static double ortalama(int vize, int fin) {
2    double ort = vize * 0.4 + fin * 0.6;
3    return ort;
4  }
5  public static void main(String[] args) {
6    Scanner giris = new Scanner(System.in);
7    int vize = giris.nextInt();
8    int fin = giris.nextInt();
9    double ort = ortalama(vize, fin);
10   if (ort >= 50) System.out.println(ort + " geçti");
11   else System.out.println(ort + " kaldı");
▶  }
```

ÇAĞRI YIĞINI

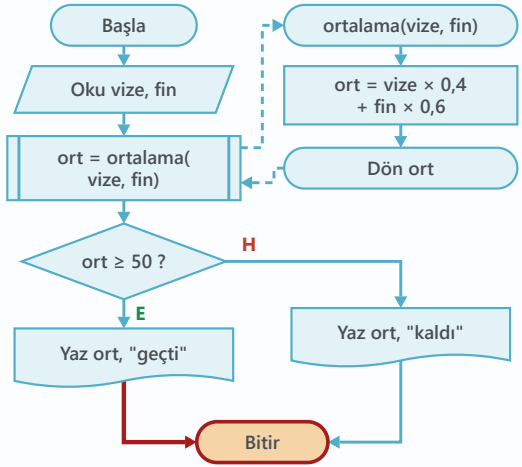
boş

DEĞİŞKENLER

`vize = 40` `fin = 70` `ort = 58.0`

ÇIKTI

```
40 70
58.0 geçti
```



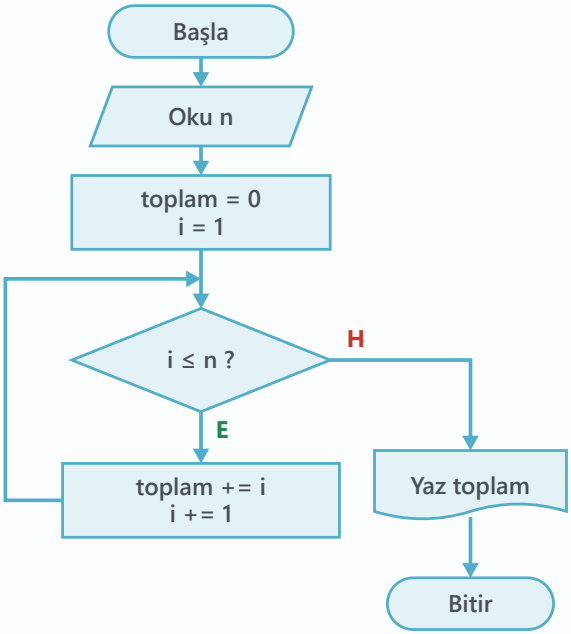
Bitir

Bitir. Aynı alt program başka girdilerle yeniden çağrılabilir: 45 ve 30 için `36.0 kaldı`.

Adım 8 / 8

UYGULAMA

Uygulama: 1'den n'e Kadar Toplam



Sıralı (okuma, başlangıç değerleri), tekrarlı (toplama) ve yine sıralı (yazdırma) akış bir arada. `toplam` değişkeni bir **biriktiricidir** (accumulator): 0'dan başlar, her turda büyür.

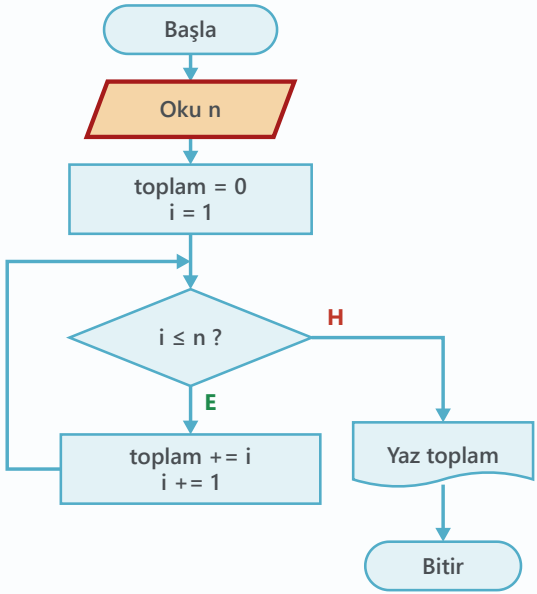
```
System.out.print("n: ");
int n = giris.nextInt();
int toplam = 0;
int i = 1;
while (i <= n) {
    toplam += i;
    i += 1;
}
System.out.println("Toplam: " + toplam);
```

i	1	2	3	4	5	6
toplam	1	3	6	10	15	15 (çıkış)

n = 5 için çıktı: `Toplam: 15` (1 + 2 + 3 + 4 + 5).

1'den n'e Toplam: Kod İzleme

```
System.out.print("n: ");
int n = giris.nextInt();
int toplam = 0;
int i = 1;
while (i <= n) {
    toplam += i;
    i += 1;
}
System.out.println("Toplam: " + toplam);
```



DEĞİŞKENLER

`n = 4`

ÇIKTI

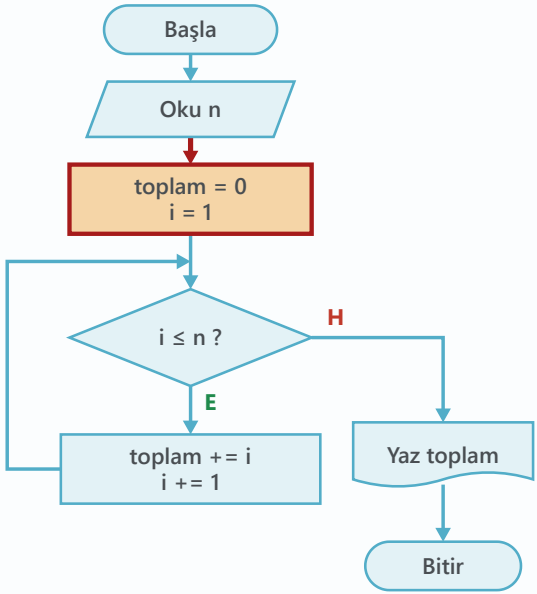
```
n: 4
```

n = 4

Mesaj yazılır, kullanıcı 4 girer: `n` = 4.

1'den n'e Toplam: Kod İzleme

```
1 System.out.print("n: ");
2 int n = giris.nextInt();
  int toplam = 0;
  int i = 1;
5 while (i <= n) {
6     toplam += i;
7     i += 1;
8 }
9 System.out.println("Toplam: " + toplam);
```



DEĞİŞKENLER

`n = 4` `toplam = 0` `i = 1`

ÇIKTI

```
n: 4
```

toplam = 0, i = 1

Başlangıç değerleri: biriktirici `toplam` = 0, sayaç `i` = 1.

1'den n'e Toplam: Kod İzleme

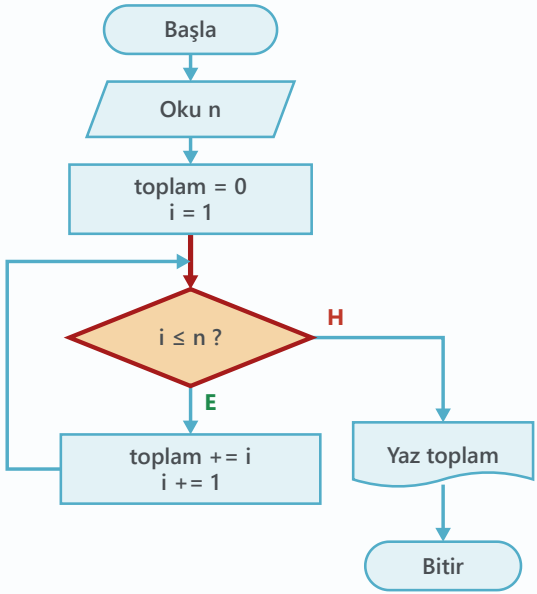
```
1 System.out.print("n: ");
2 int n = giris.nextInt();
3 int toplam = 0;
4 int i = 1;
▶ while (i <= n) {
6     toplam += i;
7     i += 1;
8 }
9 System.out.println("Toplam: " + toplam);
```

DEĞİŞKENLER

`n = 4` `toplam = 0` `i = 1`

ÇIKTI

```
n: 4
```



1 ≤ 4 → E

Koşul: 1 ≤ 4 **doğru** → Evet, gövdeye girilir.

1'den n'e Toplam: Kod İzleme

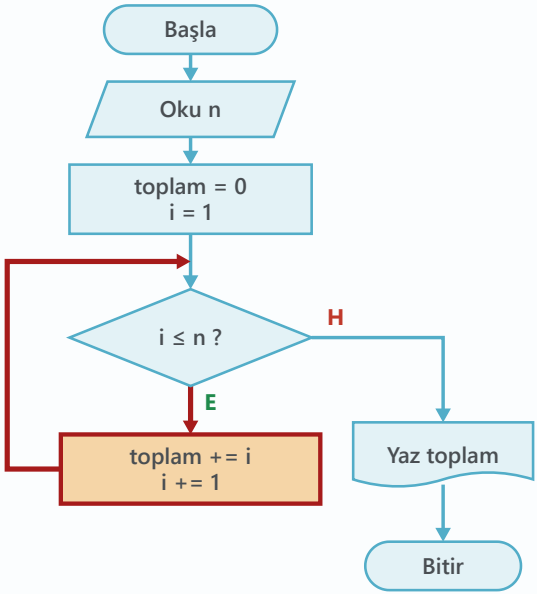
```
1 System.out.print("n: ");
2 int n = giris.nextInt();
3 int toplam = 0;
4 int i = 1;
5 while (i <= n) {
      toplam += i;
      i += 1;
8 }
9 System.out.println("Toplam: " + toplam);
```

DEĞİŞKENLER

`n = 4` `toplam = 1` `i = 2`

ÇIKTI

```
n: 4
```

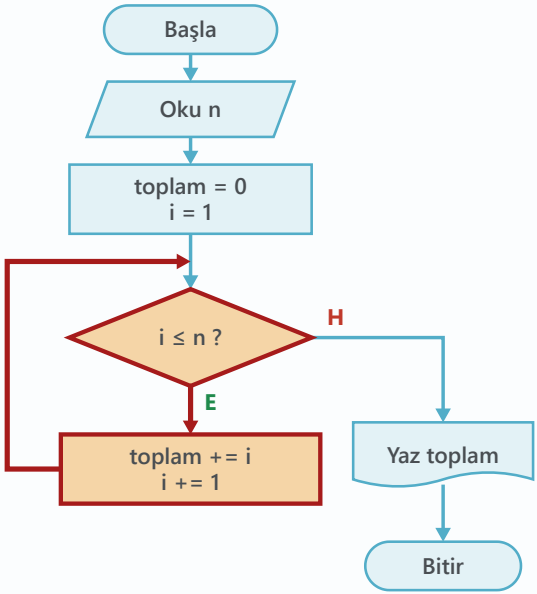


toplam = 0 + 1 = 1, i = 2

`toplam` = 0 + 1 = 1, `i` = 2; geri okla karara dönülür.

1'den n'e Toplam: Kod İzleme

```
1 System.out.print("n: ");
2 int n = giris.nextInt();
3 int toplam = 0;
4 int i = 1;
▶ while (i <= n) {
▶     toplam += i;
▶     i += 1;
8 }
9 System.out.println("Toplam: " + toplam);
```



DEĞİŞKENLER

`n = 4` `toplam = 3` `i = 3`

ÇIKTI

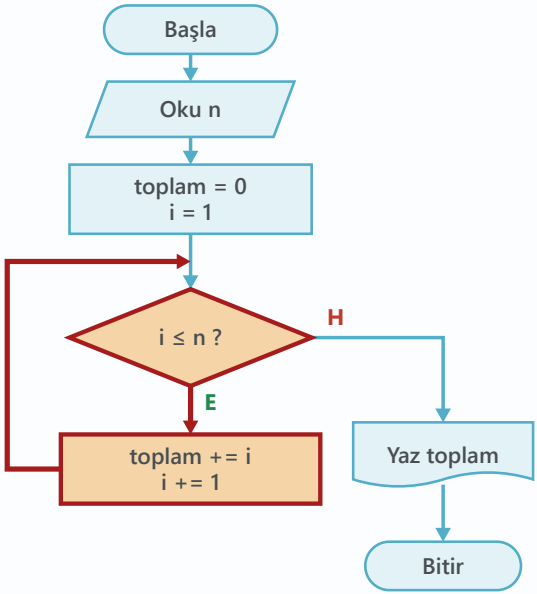
```
n: 4
```

2 ≤ 4 → toplam = 3, i = 3

2 ≤ 4 doğru: `toplam` = 1 + 2 = 3, `i` = 3.

1'den n'e Toplam: Kod İzleme

```
1 System.out.print("n: ");
2 int n = giris.nextInt();
3 int toplam = 0;
4 int i = 1;
▶ while (i <= n) {
▶     toplam += i;
▶     i += 1;
8 }
9 System.out.println("Toplam: " + toplam);
```



DEĞİŞKENLER

`n = 4` `toplam = 6` `i = 4`

ÇIKTI

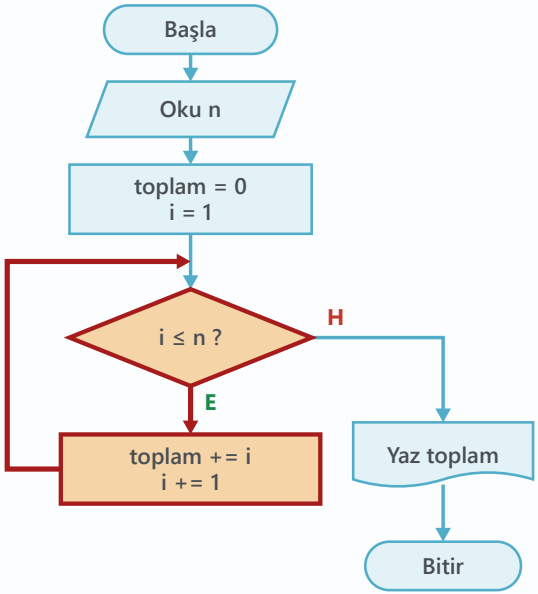
```
n: 4
```

3 ≤ 4 → toplam = 6, i = 4

3 ≤ 4 doğru: `toplam` = 3 + 3 = 6, `i` = 4.

1'den n'e Toplam: Kod İzleme

```
1 System.out.print("n: ");
2 int n = giris.nextInt();
3 int toplam = 0;
4 int i = 1;
▶ while (i <= n) {
▶     toplam += i;
▶     i += 1;
8 }
9 System.out.println("Toplam: " + toplam);
```



DEĞİŞKENLER

`n = 4` `toplam = 10` `i = 5`

ÇIKTI

```
n: 4
```

4 ≤ 4 → toplam = 10, i = 5

4 ≤ 4 doğru: `toplam` = 6 + 4 = 10, `i` = 5.

1'den n'e Toplam: Kod İzleme

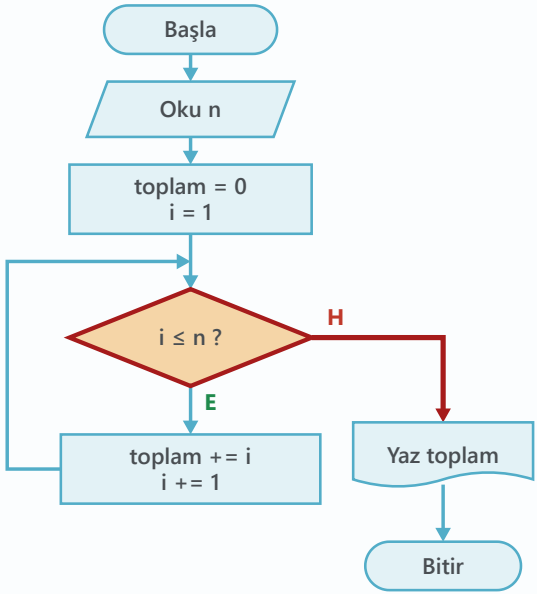
```
1 System.out.print("n: ");
2 int n = giris.nextInt();
3 int toplam = 0;
4 int i = 1;
▶ while (i <= n) {
6     toplam += i;
7     i += 1;
8 }
9 System.out.println("Toplam: " + toplam);
```

DEĞİŞKENLER

`n = 4` `toplam = 10` `i = 5`

ÇIKTI

```
n: 4
```



5 ≤ 4 → H

Koşul: 5 ≤ 4 **yanlış** → Hayır çıkışı, döngüden çıkılır.

1'den n'e Toplam: Kod İzleme

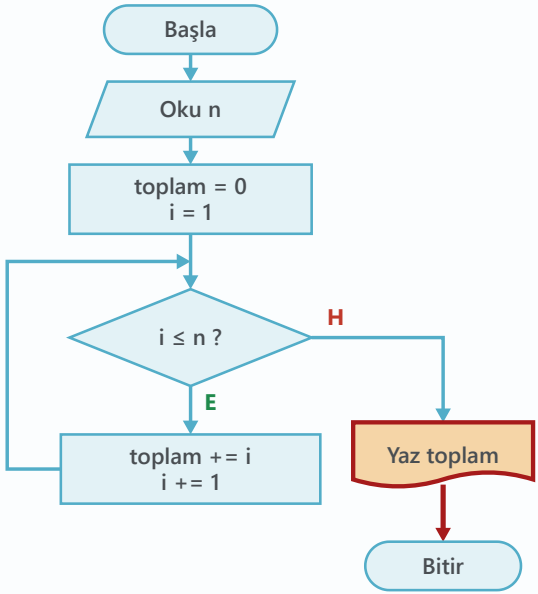
```
1 System.out.print("n: ");
2 int n = giris.nextInt();
3 int toplam = 0;
4 int i = 1;
5 while (i <= n) {
6     toplam += i;
7     i += 1;
8 }
▶ System.out.println("Toplam: " + toplam);
```

DEĞİŞKENLER

`n = 4` `toplam = 10` `i = 5`

ÇIKTI

```
n: 4
Toplam: 10
```

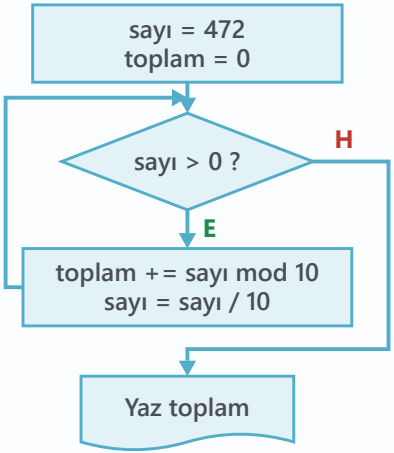


çıktı: Toplam: 10

Sonuç yazılır: 1 + 2 + 3 + 4 = 10. Gövde 4 kez, koşul 5 kez çalıştı.

UYGULAMA

Masa Başı Kontrol: İzleme Tablosu



Adım	sayı mod 10	toplam	sayı	sayı > 0 ?
başlangıç	–	0	472	E
1. tur	2	2	47	E
2. tur	7	9	4	E
3. tur	4	13	0	H → Yaz 13

- **Masa başı kontrol** (desk check): diyagram ya da kod bilgisayarsız, kâğıt üzerinde bir girdiyle izlenir.
- **İzleme tablosu**: her değişkene bir sütun, her tura bir satır. Koşul sütunu döngünün kaç kez döndüğünü gösterir: 3 tur, 4 sınama.
- `sayı / 10` tam sayı bölmesidir: 47 / 10 = 4. Basamaklar sağdan sola alınır.
- Tablo hataları erken gösterir: `sayı = sayı / 10` kutusu unutulsa sütun hep 472 kalır, yani sonsuz döngü.

```
int sayi = 472, toplam = 0;
while (sayi > 0) {
  toplam += sayi % 10;
  sayi /= 10;
}
System.out.println(toplam);
```

UYGULAMA

Diyagramı Sınamak: Test Durumları

Pozitif / negatif / sıfır diyagramı

Girdi	Neden seçildi?	Beklenen
5	1. kararın Evet yolu	pozitif
−2	2. kararın Evet yolu	negatif
0	iki kararın Hayır yolu	sıfır
1 ve −1	sınırın hemen iki yanı	pozitif, negatif

1'den n'e toplam diyagramı

n	Neden seçildi?	Beklenen
0	gövde hiç çalışmaz	Toplam: 0
1	tek tur	Toplam: 1
4	olağan durum	Toplam: 10
−3	geçersiz girdi	Toplam: 0 (istenen bu mu?)

- Tek girdiyle denemek yetmez: her karar kutusunun **her çıkışı** en az bir kez çalışmalıdır (**yol kapsama**).
- **Sınır değerleri**, koşulun sonucunun değiştiği noktanın hemen iki yanıdır: `sayı > 0` için 0 ve 1; `not ≥ 50` için 49 ve 50. Bir eksik / bir fazla hataları burada ortaya çıkar.
- Döngüler için üç durum: **hiç** dönmeyen, **bir kez** dönen, **çok kez** dönen girdi.
- Beklenen çıktı denemeden önce elle bulunur, sonra gerçek çıktıyla karşılaştırılır. Fark varsa hata ya diyagramda ya kodda ya da beklentidedir.

UYGULAMA

Algoritmayı Göstermenin Yolları

Gösterim	Tek/çift örneği	Güçlü yanı	Zayıf yanı
Doğal dil	"Sayıyı oku; 2'ye bölümünden kalan 0 ise çift, değilse tek yaz."	Herkes okuyabilir	Belirsiz olabilir
Akış diyagramı	Oku → İşlem → Karar → Yaz çift / Yaz tek	Akışı görsel gösterir, dallanma ve döngü kolay görülür	Büyük problemlerde hantal
Sözde kod	`OKU sayı; EĞER sayı mod 2 = 0 İSE YAZ "çift" DEĞİLSE YAZ "tek"`	Hızlı yazılır, koda yakın	Standart bir söz dizimi yok
Java	`if (sayi % 2 == 0) … else …`	Bilgisayar çalıştırabilir	Söz dizimi kurallarına uymak gerekir

Yazılım mühendisliğinde akış diyagramının modern karşılığı **UML etkinlik diyagramıdır** (activity diagram): aynı fikir; başlangıç dolu daire, karar eşkenar dörtgen, etkinlikler yuvarlatılmış kutular.

UYGULAMA

Akış Diyagramı Çizim Araçları

Araç	Ne yapar?	Not
Kâğıt ve kalem	En hızlı taslak; sınavda tek seçenek	Düzeltmesi zahmetli
draw.io (diagrams.net)	Tarayıcıda çalışan ücretsiz çizim aracı; hazır akış diyagramı sembolleri	Çizer, çalıştırmaz
Microsoft Visio	Ticari diyagram programı; standart sembol kütüphaneleri	Ücretli
Flowgorithm	Diyagramı **çalıştırır**, değişkenleri izler; Java dâhil birçok dile kod üretir	Ücretsiz, eğitim amaçlı
Mermaid	Metinden diyagram çizer; GitHub ve birçok Markdown aracı doğrudan gösterir	Kodla birlikte saklanır

Mermaid ile tek/çift diyagramı

```
flowchart TD
  A([Başla]) --> B[/Oku sayı/]
  B --> C{sayı mod 2 = 0 ?}
  C -- Evet --> D[Yaz çift]
  C -- Hayır --> E[Yaz tek]
  D --> F([Bitir])
  E --> F
```

- `([ ])` Başla/Bitir, `[/ /]` giriş, `{ }` karar, `[ ]` işlem; `-->` akış çizgisi, `-- Evet -->` etiketli çıkış.
- `TD` : yukarıdan aşağıya (top-down) yerleşim; kutuların yerini araç kendisi belirler.
- Çizim aracı ne olursa olsun kurallar aynıdır: tek Başla, etiketli karar çıkışları, kesişmeyen oklar.

ÖZET

Özet

Konu	Hatırlanacaklar
Akış diyagramı	algoritmanın sembol ve oklarla çizimi; dilden bağımsız plan
Semboller	oval Başla/Bitir · paralelkenar giriş · dikdörtgen işlem · eşkenar dörtgen karar · altıgen döngü · dalgalı kutu yazdır · çember bağlantı
Kurallar	tek Başla; yukarıdan aşağı; karar kutusunun iki etiketli çıkışı (E/H); kesişmeyen çizgiler
Sıralı akış	adımlar art arda, her biri bir kez; bağımlı adımların sırası önemli
Koşullu akış	K sorusu → Evet: P, Hayır: Q; Java `if/else`; iç içe kararlarla üç ve daha çok yol
Tekrarlı akış	geri ok + karar; başlangıç, koşul, güncelleme; Java `while`, `for`, `do-while`
Kalıplar	sayaç, biriktirici, bayrak; bekçi değerli döngü; erken çıkış (`break`); alt program = metot
Tuzaklar	`<` / `≤` sınır hatası; güncellemeyi unutmak → sonsuz döngü

ÖZET

Sık Yapılan Hatalar

Karar kutusunda tek çıkış
Karar sembolünden iki ok çıkmalı ve ikisi de E/H diye etiketlenmeli; aksi hâlde "hayır" durumunda ne olacağı belirsiz kalır.

Yanlış sembol
Klavyeden okuma için dikdörtgen, hesaplama için paralelkenar kullanmak. Girdi = paralelkenar, işlem = dikdörtgen.

Karşılaştırmada = kullanmak
`if (sonuc = 0)` derlenmez: `error: incompatible types: int cannot be converted to boolean`. Doğrusu `sonuc == 0`.

Başlangıç değerini unutmak
Diyagramda "sayaç = 1" kutusu yoksa Java da şikâyet eder: `error: variable sayac might not have been initialized`.

Güncellemeyi unutmak
"sayaç += 1" kutusu yoksa koşul hep doğru kalır: **sonsuz döngü**. Derleyici bu hatayı yakalamaz.

Sınır hatası
"1'den 10'a kadar" için `sayac < 10` yazmak 10'u atlar. Koşul son değerle sınanmalıdır.

Döngü başlığına noktalı virgül
`while (sayac <= 10);` derlenir ama gövdesi boş bir döngü olur; sayaç artmadığı için program takılır.

Geri oku yanlış yere bağlamak
Geri ok "sayaç = 1" kutusunun **üstüne** dönerse sayaç her turda 1'e sıfırlanır: yine sonsuz döngü. Ok karar kutusuna dönmelidir.

ÖZET

Örnek Problemler

1. "Kullanıcıdan yaşını al", "yaş ≥ 18 mi?", "vergi = fiyat × 0,2" ve "Sonucu ekrana yaz" adımları hangi sembollerle çizilir?
2. Diyagram: toplam = 0, i = 1; i ≤ 4 iken toplam += i ve i += 1; sonra Yaz toplam. Çıktı nedir? Döngüden çıkışta i kaçtır?
3. Tek/çift diyagramına **−3** girilirse ne yazılır? (Java'da `-3 % 2` kaçtır?)
4. 1'den 10'a diyagramında koşul `sayaç < 10` olsaydı kaç sayı yazılırdı? Karar kutusu kaç kez sınanırdı?
5. sayı = 1 iken "sayı < 100 ise sayı = sayı × 2" döngüsü kaç kez döner ve sayı kaç olur?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. Kenarları klavyeden okunan dikdörtgenin alanını ve çevresini yazan akış diyagramını çizin (sıralı).
2. Girilen üç sayıdan en büyüğünü bulan diyagramı çizin. 3, 9, 5 için izleyin.
3. 1'den n'e kadar **çift** sayıların toplamını bulan diyagramı çizin. n = 10 için sonuç?
4. n! (faktöriyel) hesaplayan diyagramı çizin ve Java'ya çevirin. n = 5 için sonuç?
5. Kullanıcı 0 girene kadar sayı okuyup toplamı yazan diyagramı çizin. Girdi: 4 7 2 0.
6. 0–100 arası notu harf notuna (≥ 85 A, ≥ 70 B, ≥ 50 C, değilse F) çeviren diyagramı çizin.

▶ **Cevaplar / ipuçları**

Kaynaklar

- ISO 5807:1985, *Information processing — Documentation symbols and conventions for data, program and system flowcharts*.
- C. Böhm ve G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules", *Communications of the ACM* 9(5), 1966.
- P. Deitel ve H. Deitel, *Java: How to Program*, 11. baskı, Pearson, 2017 (Bölüm 4–5: kontrol deyimleri ve etkinlik diyagramları).
- A. B. Downey ve C. Mayfield, *Think Java*, 2. baskı, O'Reilly, 2019.
- Oracle, *The Java Tutorials*: "Control Flow Statements" (docs.oracle.com/javase/tutorial).
- K. Sierra ve B. Bates, *Head First Java*, 3. baskı, O'Reilly, 2022.

Son

Bölüm 1.4: Akış Diyagramları

Sonraki bölüm: Bölüm 2.1: Programlama · Terim sözlüğü