



Bölüm 3: Kontrol Yapıları

JAVA ile Nesne Yönelimli Programlama



Kontrol Yapıları

- Programın akışını yönlendirmek için kullanılırlar.
- Java'da üç temel kontrol yapısı vardır:
 - Sıralı İşlemler
 - Karar Yapıları
 - Döngüler



Sıralı İşlemler

- Kod, yukarıdan aşağıya doğru satır satır sırayla çalışır.
- Bir işlem bitmeden diğeri başlamaz.

```
int x = 5;
```

```
int y = 10;
```

```
int sonuc = x + y;
```

```
System.out.println(sonuc);
```



Karar Yapıları

- Koşullara dayalı işlemleri kontrol eder.
- İki temel tür: if-else ve switch-case.

```
int sayi = 5;  
if (sayi > 0) {  
    System.out.println("Sayı pozitif.");  
} else {  
    System.out.println("Sayı negatif veya sıfır.");  
}
```



Döngüler

- Belirli işlemleri tekrar etmek için kullanılır.
- İki temel tür: for ve while döngüleri.

```
for (int i = 0; i < 5; i++) {  
    System.out.println("Merhaba, Dünya!");  
}
```



Kontrol Yapılarının Önemi

- Programları daha esnek ve güçlü hale getirirler.
- Kararlar alarak farklı işlemler yaptırılabilir.
- Döngülerle tekrar eden işlemler otomatikleştirilebilir.



Karar Verme İşlemi

- Karar verme, bir programın hangi yolun izleneceğine karar vermesidir.
- Belirli koşullara bağlı olarak farklı işlemleri yürütme yeteneği sağlar.
- Karar verme yapısı, if-else kullanılarak gerçekleştirilir.



if-else Yapısı

```
if (koşul) {  
    // Koşul doğruysa burası çalışır  
} else {  
    // Koşul yanlışsa burası çalışır  
}
```

- "Koşul" doğru (true) ise if bloğu çalışır.
- "Koşul" yanlış (false) ise else bloğu çalışır (opsiyonel).



Örnek: Yaş Kontrolü

```
int yas = 18;
```

```
if (yas >= 18) {  
    System.out.println("Ehliyet alabilirsiniz.");  
} else {  
    System.out.println("Ehliyet alamazsınız.");  
}
```



Mutlak Değer

```
Scanner scanner = new Scanner(System.in);
System.out.print("bir sayı girin: ");
double sayi = scanner.nextDouble();
double mutlakDeger;
if (sayi < 0) {
    mutlakDeger = -sayi;
} else {
    mutlakDeger = sayi;
}
System.out.println("sayının mutlak değeri: " + mutlakDeger);
```



Karar Yapısı İşleyişi

- Koşul değerlendirilir.
 - Eğer koşul doğruysa,
 - if bloğu çalışır,
 - else bloğu atlanır.
 - Eğer koşul yanlışsa,
 - if bloğu atlanır,
 - else bloğu çalışır (opsiyonel).



İç İçe Kararlar (Nesting)

- Birden fazla karar yapısı iç içe kullanılabilir.
- Daha karmaşık koşullar ve işlemler için kullanışlıdır.
- Kodun okunabilirliğini olumsuz etkileyebilir.

```
if (koşul1) {  
    if (koşul2) {  
        // işlemler  
    }  
} else {  
    // işlemler  
}
```



İç İçe Kararlar (Nesting)

```
int not = 85;
```

```
if (not >= 60) {  
    if (not >= 90) {  
        System.out.println("Harf notu: A");  
    } else {  
        System.out.println("Harf notu: B");  
    }  
} else {  
    System.out.println("Harf notu: F");  
}
```



Karar Verme İşleminin Önemi

- Programın farklı şartlara ve kullanıcı girdilerine tepki vermesini sağlar.
- Kodun kontrolünü ve akışını yönlendirir.
- İşlemleri belirli koşullara göre seçerek programın esnekliğini artırır.



switch-case Yapısı

- Bir dizi koşulu karşılaştırmak ve duruma göre farklı işlemleri gerçekleştirmek için kullanılır.
- Birden çok şarta dayalı bir yol seçmeye yarar.
- Çok sayıda koşulun olduğu durumlarda kullanışlıdır.



switch-case Yapısı

```
switch (değişken) { // "Değişken," kontrol edilen değer
    case değer1: // Her "case" bir değeri temsil eder.
        // İşlemler 1
        break;
    case değer2:
        // İşlemler 2
        break;
    // ...
    default: // "Default", hiçbir "case" eşleşmezse çalışır.
        // Varsayılan işlemler (opsiyonel)
}
```




Örnek: Mevsim Belirleme

```
int ay = 4;
String mevsim;

switch (ay) {
    case 3:
    case 4:
    case 5:
        mevsim = "İlkbahar";
        break;
    // ...
    default:
        mevsim = "Bilinmiyor";
}
```



switch-case Yapısının İşleyişi

- Değişken, **case** ifadeleri ile karşılaştırılır.
- Eşleşen **case** bulunursa,
 - o **case** içindeki işlemler çalıştırılır ve
 - ardından **break** ifadesi ile çıkılır.
- Eşleşen **case** bulunmazsa,
 - **default** bölümü (varsa) çalışır.
 - **default** bölümü yoksa, **switch-case** yapısından çıkılır.



switch-case ve if-else Karşılaştırması

- **switch-case** yapısı, birden çok koşulu kontrol etmek için kullanılır.
- **if-else** yapısı daha esnek, çok fazla koşul olduğunda karmaşık olabilir.
- **switch-case** sadece sabit ifadeleri kullanabilir.
- **if-else** her türlü koşulu kontrol edebilir.



Sıcaklık Dönüşüm Tablosu

Celsius (°C)	Fahrenheit (°F)	Kelvin (K)
-40	-40	233.15
0	32	273.15
25	77	298.15
100	212	373.15



Sıcaklık Dönüşüm Formülleri

- Celsius to Fahrenheit:
 - $(^{\circ}\text{C} * 1.8) + 32 = ^{\circ}\text{F}$
- Fahrenheit to Celsius:
 - $(^{\circ}\text{F} - 32) / 1.8 = ^{\circ}\text{C}$
- Celsius to Kelvin:
 - $^{\circ}\text{C} + 273.15 = \text{K}$



Sıcaklık Dönüşümü

```
public class Program {  
  
    public static void main(String[] args)  
    {  
        double celsius = 25.0;  
        double fahrenheit = (celsius * 1.8) + 32;  
        double kelvin = celsius + 273.15;  
        System.out.println(celsius + "°C=" + fahrenheit + "°F");  
    }  
}
```



Sıcaklık Dönüşümü

```
public static void main(String[] args)
{
    double[] celsiuses = {-40, 0, 25, 100};
    for (double celsius : celsiuses) {
        double fahrenheit = (celsius * 1.8) + 32;
        double kelvin = celsius + 273.15;
        System.out.println(celsius + "°C=" + fahrenheit + "°F");
    }
}
```



Döngü ve Türleri

- Aynı işlemi birden çok kez tekrarlamak için kullanılır.
- **for** : Belirli bir aralıktaki değerler ile çalışır.
- **while** : Belirli bir koşul sağlandığı sürece çalışır.
- **do-while** : Koşul sona eklenir, yani döngü en az bir kez çalışır.



for Döngüsü

```
for (int i = 0; i < 5; i++) {  
    // Döngü içinde yapılacak işlemler  
}
```

- Başlangıç, koşul, ve artırma adımlarını belirler.
- Her döngü dönüşünde koşul kontrol edilir.
- Artırma adımı, döngü değişkenini günceller.



Sayı Toplama

```
int toplam = 0;
for (int sayac = 1; sayac <= 10; sayac++) {
    toplam += sayac;
}
System.out.println("1 ile 10 arası sayı toplamı: " + toplam);
```



Aralık Toplama

```
Scanner klavye = new Scanner(System.in);
System.out.print("başlangıç değeri (a): ");
int a = klavye.nextInt();
System.out.print("bitiş değeri (b): ");
int b = klavye.nextInt();
int toplam = 0;
for (int i = a; i <= b; i++) {
    toplam += i;
}
System.out.println("sayıların toplamı: " + toplam);
```



while Döngüsü

```
int i = 0;  
while (i < 5) {  
    // Döngü içinde yapılacak işlemler  
    i++;  
}
```

- Sadece bir koşulu kontrol eder.
- Koşul sağlandığı sürece çalışır.
- Sonsuz döngülere dikkat!



Sayı Toplama

```
int toplam = 0;
```

```
int sayac = 1;
```

```
while (sayac <= 10) {
```

```
    toplam += sayac;
```

```
    sayac++;
```

```
}
```

```
System.out.println("1 ile 10 arası sayı toplamı: " + toplam);
```



Faktoriyel Hesaplama

```
int n = 5;
int faktoriyel = 1;
int sayac = 1;

while (sayac <= n) {
    faktoriyel *= sayac;
    sayac++;
}

System.out.println(n + " faktoriyeli: " + faktoriyel);
```



do-while Döngüsü

```
int i = 0;  
do {  
    // Döngü içinde yapılacak işlemler  
    i++;  
} while (i < 5);
```

- İlk olarak işlem yapar, ardından koşulu kontrol eder.
- Koşul sağlandığı sürece tekrar eder.



Döngülerin İşleyişi

- Başlangıç adımıyla başlar.
- Koşul kontrol edilir.
 - Koşul sağlanıyorsa, döngü içindeki işlemler yapılır.
 - Arttırma adımı uygulanır (for döngüsü için).
 - Koşul hala sağlanıyorsa, tekrar döngü içine girilir.
 - Koşul sağlanmazsa, döngüden çıkılır.



Döngülerin Önemi

- Veri kümesinin sırayla işlenmesi ve işlem tekrarları için kullanışlıdır.
- Kodun daha düzenli ve okunabilir olmasını sağlar.
- Özellikle listeler ve dizilerle çalışırken önemlidir.



İç İçe Döngüler

- Bir döngü içinde başka bir döngü kullanmaktır.
- Daha karmaşık işlemleri gerçekleştirmek için kullanılır.
- Matris, çok boyutlu dizi ve örüntülerin kullanıldığı senaryoda kullanışlıdır.
- İç içe döngülerin sayısı ve koşulları, kodun karmaşıklığını etkileyebilir.
- Fazla iç içe döngüler karmaşık ve verimsiz kodlara yol açabilir!



İç İçe Döngü Yapısı

```
for (int i = 1; i <= 3; i++) {  
    for (int j = 1; j <= 2; j++) {  
        // İç içe döngü gövdesi  
    }  
}
```

- Dıştaki döngü (i) dışarıdaki işlemi yönlendirir.
- İçteki döngü (j) içteki işlemi yönlendirir.
- Dıştaki döngünün her bir adımında, içteki döngü bir kez tamamlanır.



İç İçe Döngülerin İşleyişi

- Dıştaki döngü başlar ve i değeri belirlenir.
 - İçteki döngü başlar ve j değeri belirlenir.
 - İçteki döngü gövdesi çalışır.
 - İçteki döngü tamamlanır ve j değeri güncellenir.
 - İçteki döngü tamamlandığında dıştaki döngüye geri dönülür.
- Dıştaki döngü tamamlanır ve i değeri güncellenir.
- İşlem tamamlanana kadar bu süreç tekrar eder.



Örnek: İç İçe Döngülerle Desen Oluşturma

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

- Bu örnek, iç içe döngülerle yıldız deseni oluşturur.



Basamak Değeri

```
for (int onlar = 0; onlar <= 4; onlar++) {  
    for (int birler = 1; birler <= 9; birler++) {  
        int sayi = onlar * 10 + birler;  
        System.out.println("Onlar basamağı: " + onlar + ",  
                            Birler basamağı: " + birler + " --> " + sayi);  
    }  
}
```

- Çıktı ne olur?



Çarpım Tablosu

```
for (int i = 1; i <= 10; i++) {  
    for (int j = 1; j <= 10; j++) {  
        int carpim = i * j;  
        System.out.print(i + "x" + j + "=" + carpim + "\t");  
    }  
    System.out.println(); // Yeni satır  
}
```



Döngülerde "break" İfadesi

- **break**, döngüyü sonlandırmak için kullanılır.
- Döngü içinde bir koşul sağlandığında, döngüden çıkılır, döngü sona erer.

```
for (int i = 1; i <= 10; i++) {  
    if (i == 5) {  
        break;  
    }  
    System.out.println("Döngü adımı: " + i);  
}
```




"break" İfadesinin İşleyişi

- Döngü, her bir adımda koşulu kontrol eder.
- Eğer **break** ifadesi çalışırsa, döngü sona erer.
- Döngü dışındaki işlemler devam eder.



Döngülerde "continue" İfadesi

- **continue**, döngü içinde koşulu sağlayan adımları atlamak için kullanılır.
- **continue**, o anki adımın işlenmesini durdurur ve bir sonraki adıma geçer.

```
for (int i = 1; i <= 10; i++) {  
    if (i % 2 == 0) {  
        continue;  
    }  
    System.out.println("Tek sayı: " + i);  
}
```



"continue" İfadesinin İşleyişi

- Döngü her bir adımda koşulu kontrol eder.
- **continue** çalışırsa, o adımın işlenmesi durur ve bir sonraki adıma geçilir.
- Döngü devam eder ve diğer adımlar işlenir.



SON