



Bölüm 3: Fonksiyonlar

JAVA ile Nesne Yönelimli Programlama



Fonksiyonlar

- Fonksiyonlar, tekrar kullanılabilir kod parçalarını temsil eder.
- Parametreler ve dönüş değeri ile tanımlanırlar.
- Kodun okunabilirliğini artırır ve kod tekrarını azaltır.
- Modüler ve düzenli kod yazımını teşvik eder.



Niçin Kullanırız?

- Kodun daha okunabilir ve düzenli olmasını sağlar.
- Kod tekrarını azaltır ve hata olasılığını düşürür.
- Modülerlik: Kodu küçük parçalara ayırarak daha yönetilebilir hale getirir.



Fonksiyonların İşleyişi

- **Girdi (Input):** Fonksiyona veri veya parametre sağlanır. Bu, fonksiyonun ne üzerinde çalışacağını belirler.
- **İşlem (Processing):** Fonksiyon, verilen girdiyi işler ve belirli bir görevi yerine getirir.
- **Çıktı (Output):** Fonksiyon işlem sonucunda bir sonuç üretir ve bu sonucu döner.



Fonksiyon Tanımı

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

- **public**: Erişim belirleyici
- **int**: Dönüş değeri türü
- **toplama**: Fonksiyon adı
- **(int sayi1, int sayi2)**: Parametreler
- **{...}**: Kod bloğu



Fonksiyon Tanımı

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

- toplama fonksiyonu, sayi1 ve sayi2 adlı iki giriş parametresi alır.
- Bu parametreleri toplar ve sonucu geri döner.



Fonksiyon Kullanımı

```
int sonuc = toplama(5, 3);  
System.out.println("Toplam: " + sonuc);
```

- toplama(5, 3) çağrısı sonucu 8 dönecektir.



Parametreler ve Dönüş Değeri

- **Parametreler:** Fonksiyona giriş verilerini temsil eder.
- **Dönüş Değeri:** Fonksiyonun çıktı değerini temsil eder.



Örnek Fonksiyonlar

```
//Toplama fonksiyonu
public int toplama(int sayi1, int sayi2) {
    return sayi1 + sayi2;
}

//Kare hesaplama fonksiyonu
public double kareHesapla(double sayi) {
    return sayi * sayi;
}

//Merhaba dünya fonksiyonu
public void merhabaDunya() {
    System.out.println("Merhaba, Dünya!");
}
```



Yapıcı (Constructor) Fonksiyon

- Bir Java sınıfının özel bir metodudur.
- Bir sınıfın her nesnesi oluşturulduğunda çağrılır.
- Sınıf ile aynı ada sahip olur ve geriye değer döndürmez.
- Nesnenin ilk durumunu atamak için kullanılır.
- Parametre olarak nesneyi özelleştirmek için kullanılabilir.



Yapıcı (Constructor) Fonksiyon

- **Varsayılan (Default) Yapıcı Fonksiyon:**
 - Eğer bir sınıfa constructor tanımlanmazsa, Java varsayılan olarak parametresiz ve hiçbir işlem yapmayan bir constructor tanımlar.
- **Aşırı Yüklenmiş (Overloaded) Constructor:**
 - Bir sınıf birden fazla constructora sahip olabilir. Farklı parametre listeleri ile aşırı yüklenmiş constructorlar oluşturulabilir.
- **Zincirleme Yapıcı Fonksiyon (Constructor Chaining):**
 - Bir constructor, başka bir constructorı çağırabilir.



Yapıcı (Constructor) Fonksiyon

```
public class Kitap {  
    String ad;  
    String yazar;  
    int sayfaSayisi;  
  
    public Kitap(String ad) {  
        this.ad = ad;  
    }  
    public Kitap(String ad, String yazar) {  
        this(ad); // Constructor chaining  
        this.yazar = yazar;  
    }  
}
```



En Çok Kullanılan Fonksiyonlar

- Programlamada kullanılan fonksiyonlar, çok çeşitli görevleri yerine getirmek için kullanılır.
- İşlemleri kolaylaştırır, kodun okunabilirliğini artırır ve daha verimli programlar yazılmasına yardımcı olur.



print() Fonksiyonu

- System.out.print() veya System.out.println() ile kullanılır.
- Metin ve değerleri ekrana yazdırmak için kullanılır.

```
System.out.println("Merhaba, Dünya!");
```



Math Sınıfı Fonksiyonları

- Math sınıfı matematiksel işlemler için kullanılır.
- Örnekler: `Math.abs()`, `Math.sqrt()`, `Math.max()`.

```
double kareKok = Math.sqrt(16);
```

```
double mutlakDeger = Math.abs(-17.2);
```

```
double maksimum = Math.max(3, 5);
```



String Fonksiyonları

- Metin işlemleri için kullanılır.
- Örnekler: `length()`, `charAt()`, `substring()`.

```
String orijinal = "Merhaba, Dünya!";
```

```
int uzunluk = orijinal.length();
```

```
String altDize = orijinal.substring(7, 12); // "Dünya" çıkar
```

```
int virgulIndex = orijinal.indexOf(",");
```




Dizi (Array) Fonksiyonları

- Dizilerde işlem yapmak için kullanılır.
- Örnekler: length, sort(), indexOf().

```
int[] dizi = {5, 2, 9, 1, 5, 4, 8, 7, 3};  
int[] siraliDizi = Arrays.copyOf(dizi, dizi.Length);  
Arrays.sort(siraliDizi); // Sırala  
int arananEleman = 5;  
int sonuc = Arrays.binarySearch(siraliDizi, arananEleman);  
boolean esitMi = Arrays.equals(dizi, siraliDizi);
```



Kullanıcı Girişi Fonksiyonları

- Kullanıcıdan veri almak için kullanılır.
- Scanner sınıfı ile girdi alınır.
- `Scanner scanner = new Scanner(System.in);`
- `int yas = scanner.nextInt();`



Dosya İşlemleri Fonksiyonları

- Dosyaları okuma, yazma ve işleme işlemleri için kullanılır.
- File ve FileReader gibi sınıflarla dosya işlemleri yapılır.

```
File dosya = new File("ornek.txt");  
dosya.createNewFile();  
FileWriter yazici = new FileWriter(dosya);  
yazici.write("Bu bir örnek metin dosyasıdır.");  
yazici.close();  
okuyucu.hasNextLine();  
String satir = okuyucu.nextLine();
```



Veritabanı Bağlantısı Fonksiyonları

- Veritabanı ile etkileşim sağlamak için kullanılır.
- Veri tabanı işlemleri, SQL sorguları ile gerçekleştirilir.

```
String url = "jdbc:mysql://localhost:3306/veritabani_adı";  
baglanti = DriverManager.getConnection(url,kullanici,sifre);  
Statement statement = baglanti.createStatement();  
String sorgu = "SELECT * FROM tablo_adi";  
ResultSet sonuclar = statement.executeQuery(sorgu);  
statement.executeUpdate(eklemeSorgusu);  
statement.executeUpdate(guncellemeSorgusu);
```



Veri Yapıları Fonksiyonları

- Veri yapıları, verileri düzenlemek ve işlemek için kullanılır.
- Örnekler: Diziler, listeler, yığınlar (stacks), kuyruklar (queues).

```
ArrayList<String> liste = new ArrayList<String>();  
liste.add("Elma");  
liste.add("Armut");
```



Metin Karşılaştırma Fonksiyonları

- Metinleri karşılaştırmak için kullanılır.
- equals(), startsWith(), ve contains() gibi metin karşılaştırma yöntemleri vardır.
- `String kelime1 = "Merhaba";`
- `String kelime2 = "merhaba";`
- `boolean esitMi = kelime1.equalsIgnoreCase(kelime2);`



Zaman İşlemleri Fonksiyonları

- Tarih ve zaman işlemleri yapmak için kullanılır.
- Date, Calendar, ve SimpleDateFormat gibi sınıflarla çalışılır.

```
Date simdikiZaman = new Date();  
SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy  
HH:mm:ss");  
String formatliZaman = sdf.format(simdikiZaman);
```



Web İşlemleri Fonksiyonları

- Web servislerine veya API'lere erişmek için kullanılır.
- Veri alışverişi ve güncellemelerde kullanılır.

```
URL url = new URL("https://api.example.com/data");  
URLConnection baglanti = (URLConnection)  
url.openConnection();
```




Grafik ve Görsel İşlemleri Fonksiyonları

- Grafikler, resimler ve görsel efektler oluşturmak için kullanılır.
- Java'da Swing ve JavaFX gibi araçlarla grafiksel kullanıcı arayüzleri oluşturulabilir.

```
JFrame pencere = new JFrame("Grafik Uygulama");  
JPanel panel = new JPanel();
```



Fonksiyon İçinde Fonksiyon

- Bir fonksiyon, başka bir fonksiyonu çağırabilir.
- Bu, karmaşık görevleri parçalayarak kodunuzu daha okunabilir ve yönetilebilir hale getirir.

```
public int kareAl(int sayi) {  
    return sayi * sayi;  
}  
  
public int topla(int sayi1, int sayi2) {  
    return sayi1 + sayi2;  
}  
  
public int ikiKareToplami(int sayi1, int sayi2) {  
    return topla(kareAl(sayi1), kareAl(sayi2));  
}
```



Yerel Değişken

- Yerel değişkenler, bir fonksiyon çalıştırılırken var olan değişkenlerdir.
- Diğer fonksiyonlardan veya dışarıdan erişilemezler.

```
public int carpma(int sayi1, int sayi2) {  
    int sonuc = sayi1 * sayi2; // 'sonuc' bir yerel değişkendir  
    return sonuc;  
}
```

- sonuc, sadece carpma fonksiyonu çalıştırıldığında var olur.



Soyutlama

- Karmaşıklığı azaltmak için bazı detayları yok saymayı ifade eder.
- Kodun okunabilirliğini artırır ve işlemi basitleştirir.
- Büyük ve karmaşık programları daha anlaşılır hale getirir.
- Kodun parçalara ayrılmasını ve her parçanın ayrıca düşünülmesini sağlar.



Genelleme

- Belirli bir bağlamda çalışan bir şeyi daha fazla bağlamda çalışmaya uygun hale getirmeyi ifade eder.
- Daha genel ve çok amaçlı kullanılabilir hale getirme.



Hesaplamalar Üzerinde Soyutlama

- Hesaplamaları soyutlamak, işlevsel soyutlama veya işlemci soyutlama olarak da bilinir.
- Bir fonksiyonun, işlemin nasıl hesaplandığını bilmek yerine sadece sonucunu bilmek önemlidir.



Fonksiyonel Soyutlama

- Bir fonksiyonun anlamı/görevi bilindiği sürece, işlemi nasıl hesapladığı umursanmaz.
- Fonksiyonun gerçekleştirimi (gövdesi) hakkında endişelenilmez.

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

- Bu fonksiyon, konsola yazı yazar, ancak nasıl yaptığını bilmek gerekmez.



Her Değişken Sadece Bir Şeyi Temsil Etmelidir

- Her değişken, sadece bir şeyi veya bir veriyi temsil etmelidir.
- İyi isimlendirme, kodun anlaşılır ve bakımı kolay olmasına yardımcı olur.
- Kodun okunabilirliğini artırır.
- Mantıksal hataları azaltır.
- Kodun yeniden kullanılabilirliğini artırır.



Değişken İsimlendirme

- Değişkenlerin açıklayıcı ve anlamlı isimlere sahip olması önemlidir.
- Temsil ettikleri veri veya işlemin doğasını yansıtan isimlere sahip olmalıdır.
- İsimler, değişkenin içeriğini ve kullanımını anlatmalıdır.
- Anlamsız veya tek harfli değişken isimleri kaçınılmalıdır.

```
int ogrenciSayisi; // İyi bir isimlendirme örneği
```

```
int a; // Anlamsız bir isim
```

```
int yas = 25;
```

```
String isim = "Ahmet";
```

```
double ortalama = 85.5;
```

```
String kitapAdi;
```



Problemi Parçalama

- Problemi daha küçük ve yönetilebilir parçalara ayırma, programlama sürecinin temel adımıdır.
- Fonksiyonlar, kodun daha düzenli ve okunabilir olmasını sağlar.
- Kodun yeniden kullanılabilir ve sürdürülebilir olmasını sağlar.



Fonksiyon Kullanmaya Ne Zaman Karar Verilir

- Bir kural: DRY (**Don't Repeat Yourself**).
- Kod kopyalama ve yapıştırma gereksinimi olduğunda, fonksiyon kullanın!
- Kodun tekrar kullanılabilirliği ve bakımı için önemlidir.
- İyi bir fonksiyon, tek bir işlevi yerine getirir ve okunaklıdır.



İyi Bir Fonksiyon Nasıl Tasarlanır

- **Tek Sorumluluk:** Her fonksiyon, sadece bir işlevi yerine getirmelidir.
- **Açıklayıcı İsim:** Fonksiyonun ne yaptığını anlatan isimler kullanın.
- **Parametreler:** Gerekli verileri parametreler aracılığıyla almalıdır.
- **Dönüş Değeri:** Fonksiyon sonucunu açıkça dönmelidir.
- Sadece kullanılacak yerlerde çağrılmalıdır.



Kapsam (Scope)

- Bir değişkenin tanımlandığı ve kullanılabildiği kod bölgesini ifade eder.
- Her değişkenin bir kapsamı vardır.
- Kapsam, bir değişkenin erişilebilirlik ve yaşam süresini belirler.
- Yerel ve global kapsamlar farklı erişim kurallarına sahiptir.
- İç içe kapsam, iç kapsamdaki değişkenlere erişebilir, ancak isim çakışması sorunu olabilir.
- Doğru kullanılmadığında hatalara yol açabilir.



Yerel Kapsam (Local Scope)

- Yerel değişkenler sadece belirli bir kod bloğunda erişilebilirler.
- Genellikle fonksiyonların içinde tanımlanırlar.

```
public void fonksiyon() {  
    int yerelDegisken = 5; // yerel kapsam  
}
```



Global Kapsam (Global Scope)

- Global değişkenler, programın her yerinden erişilebilirler.
- Programın başından sonuna kadar yaşarlar.

```
int globalDegisken = 10; // global kapsam
```

```
public void fonksiyon() {  
    int yerelDegisken = 5; // yerel kapsam  
}
```



İç İçe Kapsam (Nested Scope)

- Bir kapsamın içinde başka bir kapsam olabilir.
- İç içe kapsam, içteki kapsamdaki değişkenlere erişebilir.

```
public void fonksiyon() {  
    int disDegisken = 10;  
  
    {  
        int icerdekiDegisken = 5;  
        // disDegisken ve icerdekiDegisken burada erişilebilir  
    }  
}
```




Anonim (Lambda) Fonksiyonlar

- Anonim fonksiyonlar, isimsiz fonksiyonlar olarak da bilinir.
- İsim yerine doğrudan kod parçasını temsil ederler.
- Java'da Lambda ifadeleri olarak bilinirler.

(ParametreListesi) -> İfade

- **(ParametreListesi):** Lambda fonksiyonunun parametre listesi.
- **->:** Gövdeyi parametre listesinden ayırır.
- **İfade:** Lambda fonksiyonunun işlevini tanımlayan kod parçası.



Örnek: Toplama Lambda İfadesi

`(int x, int y) -> x + y`

- İki tamsayı parametre alır ve bunları toplar.



Kullanım Alanları

- **Koleksiyon işlemleri:** Liste filtreleme, sıralama, eşleştirme.
- **Thread yönetimi:** Paralel işlemler için kullanılır.
- **Daha okunabilir kod:** Küçük işlemler için karmaşık fonksiyonlar yerine lambda ifadeleri kullanılır.



Lambda İfadesi Örneği: Liste Filtreleme

- `List<Integer> sayilar = Arrays.asList(1, 2, 3, 4, 5, 6, 7);`
- `List<Integer> ciftSayilar = sayilar.stream().
filter(s -> s % 2 == 0).
collect(Collectors.toList());`
- Bu örnek, lambda ifadesiyle çift sayıları filtreler.

Ödev



- Kullanıcıdan girdi olarak satır sayısını alan ve şekildeki benzer çıktı üreten java kodunu derste bahsedilen kurallara uygun olarak yazınız. *
 - sercan.kulcu@giresun.edu.tr **
 - Son tarih: 9 Kasım 2023 saat 23:59'a kadar ***
 - Konu: «EEM-103 Ödev 2» ****
- Merhaba Hocam, *****
- Ben xxxx numaralı 'yım. Ekte ödevimi gönderiyorum. *****
- Saygılarımla, *****
- İyi çalışmalar dilerim.
- Ekler: odev2.java

Ödev



- Ödevler isteğe bağlı değil, **zorunlu**.
- ChatGPT gibi araçlara yazdırabilirsiniz, ancak **ne gönderdiğinizden** sorumlusunuz.
- Ödevi **bireysel** olarak yapmalısınız.
- Mail **konu** ve metin kısmına dikkat ediniz.
- Ödevinizi **tam** olarak teslim ediniz. (koda müdahale gerekmemeli)



SON