Bölüm 3: Fonksiyonlar

JAVA ile Nesne Yönelimli Programlama

Fonksiyonlar

- Tekrar kullanılabilir kod parçalarını temsil eder.
- Aldığı parametreler ve dönüş değer tipi ile tanımlanırlar.
- Kodun okunabilirliğini artırır ve kod tekrarını azaltır.
- Modüler ve düzenli kod yazımını teşvik eder.
- Kodu küçük parçalara ayırarak daha yönetilebilir hale getirir.

Fonksiyonların İşleyişi

- **Girdi (Input):**
  - Fonksiyona veri sağlanır.
  - Fonksiyonun ne üzerinde çalışacağını belirler.
- **İşlem (Process):**
  - Fonksiyon, verilen girdiyi işler.
  - Belirli bir görevi yerine getirir.
- **Çıktı (Output):**
  - Fonksiyon işlem sonucunda bir sonuç üretir.
  - Sonuç değerini geri döndürür.

Fonksiyon Tanımı

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

- **public**: Erişim düzenleyici
- **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 girdi parametresi alır.
- Bu parametreleri toplar ve sonucu geri döner.

Fonksiyon Kullanımı

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

- toplama(5, 3); çağrı sonucu 8 değerini döner.
- sonuc değişkenine 8 değeri atanır.

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

- Sınıfların özel metodudur.
- Bir sınıftan bir nesne oluşturulacağında çağrılır.
- Sınıf ile aynı adı paylaşır.
- Geriye değer döndürmez.
- Nesnenin ilk durum değerlerini atamak için kullanılır.
- Parametre alarak nesneyi özelleştirmek için kullanılabilir.

Yapıcı (Constructor) Fonksiyon

- **Varsayılan (Default):**
  - Sınıfa yapıcı metot tanımlanmaz ise, otomatik olarak parametresi olmayan ve hiçbir işlem yapmayan bir tane tanımlanır.
- **Aşırı Yüklenmiş (Overloaded):**
  - Bir sınıf birden fazla yapıcı metoda sahip olabilir.
  - Farklı parametre listeleri ile birbirlerinden ayrılırlar.
  - Her birine aşırı yüklenmiş yapıcı metot denir.
- **Zincirleme (Chaining):**
  - Bir yapıcı metot, bir diğerini 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;
  }
}
```

Nokta Operatörü (Dot Operator)

- Nokta işleci, nesnelerin niteliklerine ve metotlarına erişim için kullanılır.

```
Kitap kitap = new Kitap();
//Kitap nesnesinin özelliklerine erişim
kitap.baslik = "Java Programlama";
kitap.yazar = "John Doe";
//Kitap nesnesinin metotlarına erişim
kitap.kitapBilgisiGoster();
```

En Sık Kullanılan Fonksiyonlar

- 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 konsola 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");
liste.remove("Ayva");
```

Metin Karşılaştırma Fonksiyonları

- Metinleri karşılaştırmak için kullanılır.
- equals(), startsWith(), ve contains() gibi karşılaştırma metotları 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");
HttpURLConnection baglanti = (HttpURLConnection)
                                        url.openConnection();
```

Grafik ve Görsel İşlemleri Fonksiyonları

- Grafikler, resimler ve görsel efektler oluşturmak için kullanılır.
- 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.
- Karmaşık görevleri parçalar, kodu 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

- Fonksiyon çalıştırılırken var olan değişkenlerdir.
- Fonksiyon dışından erişilemezler.

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

- sonuc, sadece carpma fonksiyonu yürütümü sırasında var olur.

Soyutlama

- Karmaşıklığı azaltmak için 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

- Bağlam (context),
  - Bir sayının, bir denklem içindeki diğer unsurlarla ilişkisini,
  - Bir kelimenin anlamının, cümle içindeki yeri ve durumu ile ilişkisini,
  - Bir kod bloğu içindeki durum ve etkileşimleri, ifade eder.
- Bir ifadenin anlamı, cümle içindeki diğer kelimeler tarafından etkilenebilir.
- Genelleme,
  - Bir bağlama ait bir nesneyi, başka bağlamlara uygun hale getirme.
  - Bir ürünü daha genel ve çok amaçlı kullanılabilir hale getirme.

Hesaplamalar Üzerinde Soyutlama

- İşlevsel soyutlama veya işlemci soyutlama olarak da bilinir.
- Bir fonksiyonun, nasıl yerine ne hesapladığını bilmek önemlidir.

Fonksiyonel Soyutlama

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

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

- Fonksiyon, verilen metni konsola yazar, nasıl yaptığını bilmek gerekmez.

Her Değişken Sadece Bir Şeyi Temsil Etmelidir

- Her değişken, sadece bir nesneyi veya bir veriyi temsil etmelidir.
- İyi isimlendirme, kodun anlaşılır ve bakımı kolay olmasını sağlar.
- 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 isimlerinden 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 küçük 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 kullanılmalıdır.
- **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.
- Değişkenin erişilebilirlik ve yaşam süresini belirler.
- Yerel ve genel (global) kapsamlar farklı erişim kurallarına sahiptir.
- İç içe kapsam,
  - İç kapsamdan bir üst kapsamda bulunan değişkenlere erişilebilir,
  - Ancak, isim çakışması sorunu yaşanabilir.
- Doğru kullanılmadığında hatalara yol açabilir.

Yerel Kapsam (Local Scope)

- Yerel değişkenler sadece belirli bir kod bloğunda erişilebilirler.
- Fonksiyonların içinde tanımlanırlar.
- Değişkenin yaşam süresi fonksiyonun yaşam süresi kadardır.

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

Global Kapsam (Global Scope)

- Global değişkenlere programın her yerinden erişilebilir.
- Değişkenin yaşam süresi programın yaşam süresi kadardır.

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

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

İç İçe Kapsam (Nested Scope)

- Bir kapsamın içinde başka bir kapsam tanımlanabilir.
- İç kapsam, dış kapsama ait değişkenlere erişebilir.

```
public void fonksiyon() {
    int disDegisken = 10;
    {
        int icDegisken = 5;
        // disDegisken ve icDegisken 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):** fonksiyonun parametre listesi.
- **->:** fonksiyonun gövde ve parametre listesini ayırır.
- **İfade:** fonksiyonunun işlevini tanımlayan kod parçası.

Toplama Lambda İfadesi

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

- Parametre olarak iki tamsayı alır ve toplar.

Kullanım Alanları

- **Koleksiyon işlemleri**: Liste filtreleme, sıralama, eşleştirme.
- **İş parçacığı yönetimi**: Paralel yürütülen işlemler için.
- **Daha okunabilir kod**: Küçük işlemler için fonksiyon tanımlamak yerine.

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());
```

- Lambda ifadesiyle çift sayıları filtreler.

SON