JAVA LAMBDA EXPRESSIONS

FONKSİYONEL PROGRAMLAMA TEMELLERİ

Lambda Syntax, Functional Interfaces, Method References

GİRİŞ: LAMBDA EXPRESSIONS NEDİR?

Lambda Nedir?

Lambda Expression = Anonim fonksiyon (isimsiz metod)

Neden Lambda?

- Daha kısa kod
- Daha okunabilir
- Fonksiyonel programlama
- Stream API ile birlikte güçlü

Eski Yol vs Lambda:

```
// ESKİ YOL - Anonim sınıf (Java 7 ve öncesi)
Runnable eski = new Runnable() {
    @Override
    public void run() {
        System.out.println("Merhaba Dünya");
    }
};

// YENİ YOL - Lambda (Java 8+)
Runnable yeni = () -> System.out.println("Merhaba Dünya");
```

Fark: 5 satır → 1 satır!

1. LAMBDA SYNTAX (SÖZDİZİMİ)

1.1 Temel Yapı

```
// Temel lambda yapısı:
(parametreler) -> { gövde }

// Örnekler:
() -> System.out.println("Merhaba")        // Parametresiz
(x) -> x * 2                                 // Tek parametre
(x, y) -> x + y                              // Çift parametre
(x, y) -> { return x + y; }                  // Blok gövde
```

1.2 Lambda Kuralları

```
import java.util.function.*;

public class LambdaSyntax {
    public static void main(String[] args) {

        // 1. PARAMETRESIZ LAMBDA
        Runnable r1 = () -> System.out.println("Parametresiz");
        r1.run();

        // 2. TEK PARAMETRE - Parantez opsiyonel
        Consumer<String> c1 = (mesaj) -> System.out.println(mesaj);
        Consumer<String> c2 = mesaj -> System.out.println(mesaj);
        c1.accept("Merhaba");

        // 3. ÇOK PARAMETRE - Parantez zorunlu
        BiFunction<Integer, Integer, Integer> toplam =
            (a, b) -> a + b;
        System.out.println("Toplam: " + toplam.apply(5, 3));

        // 4. TEK SATIR - Return opsiyonel
        Function<Integer, Integer> kare1 = x -> x * x;
        Function<Integer, Integer> kare2 = x -> { return x * x; };

        // 5. ÇOK SATIR - Süslü parantez ve return zorunlu
        BiFunction<Integer, Integer, String> karsilastir =
            (a, b) -> {
                if (a > b) return a + " büyük";
                else if (a < b) return b + " büyük";
                else return "Eşit";
            };
        System.out.println(karsilastir.apply(10, 5));
    }
}
```

2. FUNCTIONAL INTERFACES

2.1 Functional Interface Nedir?

Functional Interface = Sadece 1 abstract metod içeren interface

```
// Functional interface - sadece 1 abstract metod
@FunctionalInterface
interface Hesaplayici {
    int hesapla(int a, int b);

    // default ve static metodlar olabilir
    default void bilgi() {
        System.out.println("Hesaplayıcı");
    }
}

// Kullanım
public class FunctionalInterfaceOrnek {
    public static void main(String[] args) {

        // Lambda ile implementation
        Hesaplayici toplama = (a, b) -> a + b;
        Hesaplayici cikarma = (a, b) -> a - b;
        Hesaplayici carpma = (a, b) -> a * b;
        Hesaplayici bolme = (a, b) -> a / b;

        System.out.println("Toplama: " + toplama.hesapla(10, 5));
        System.out.println("Çıkarma: " + cikarma.hesapla(10, 5));
        System.out.println("Çarpma: " + carpma.hesapla(10, 5));
        System.out.println("Bölme: " + bolme.hesapla(10, 5));
    }
}
```

2.2 Hazır Functional Interfaces

Java 8 ile birlikte java.util.function paketinde hazır interface'ler:

A. Predicate<T> - Test eder, boolean döner

```
import java.util.function.Predicate;

public class PredicateOrnek {
    public static void main(String[] args) {

        // Tek parametre alır, boolean döner
        Predicate<Integer> ciftMi = sayi -> sayi % 2 == 0;
        Predicate<String> uzunMu = metin -> metin.length() > 5;
        Predicate<Integer> pozitifMi = sayi -> sayi > 0;

        System.out.println("10 çift mi? " + ciftMi.test(10));
        System.out.println("7 çift mi? " + ciftMi.test(7));
```

```
        System.out.println("'Merhaba' uzun mu? " +
            uzunMu.test("Merhaba"));

        // BİRLEŞTİRME - and, or, negate
        Predicate<Integer> ciftVePozitif = ciftMi.and(pozitifMi);
        System.out.println("10 çift VE pozitif mi? " +
            ciftVePozitif.test(10));

        Predicate<Integer> tekMi = ciftMi.negate();
        System.out.println("7 tek mi? " + tekMi.test(7));
    }
}
```

B. Function<T, R> - Dönüşüm yapar

```
import java.util.function.Function;

public class FunctionOrnek {
    public static void main(String[] args) {

        // T alır, R döner
        Function<String, Integer> uzunluk =
            metin -> metin.length();
        Function<Integer, Integer> kare = sayi -> sayi * sayi;
        Function<String, String> buyukHarf =
            metin -> metin.toUpperCase();

        System.out.println("'Merhaba' uzunluğu: " +
            uzunluk.apply("Merhaba"));
        System.out.println("5'in karesi: " + kare.apply(5));
        System.out.println("Büyük harf: " +
            buyukHarf.apply("java"));

        // ZİNCİRLEME - andThen, compose
        Function<Integer, Integer> ikiyeKat = sayi -> sayi * 2;
        Function<Integer, Integer> artiOn = sayi -> sayi + 10;

        // andThen: önce ikiyeKat, sonra artiOn
        Function<Integer, Integer> islem1 =
            ikiyeKat.andThen(artiOn);
        System.out.println("5 * 2 + 10 = " + islem1.apply(5));

        // compose: önce artiOn, sonra ikiyeKat
        Function<Integer, Integer> islem2 =
            ikiyeKat.compose(artiOn);
        System.out.println("(5 + 10) * 2 = " + islem2.apply(5));
    }
}
```

C. Consumer<T> - Tüketir, döndürmez

```
import java.util.function.Consumer;
import java.util.*;

public class ConsumerOrnek {
    public static void main(String[] args) {

        // T alır, void döner
        Consumer<String> yazdir =
            metin -> System.out.println(metin);
        Consumer<Integer> ikiyeKatYazdir = sayi ->
            System.out.println(sayi + " * 2 = " + (sayi * 2));

        yazdir.accept("Merhaba");
        ikiyeKatYazdir.accept(5);

        // ZİNCİRLEME - andThen
        Consumer<String> kucukYap = metin ->
            System.out.println("Küçük: " + metin.toLowerCase());
        Consumer<String> buyukYap = metin ->
            System.out.println("Büyük: " + metin.toUpperCase());

        Consumer<String> ikisiDe = kucukYap.andThen(buyukYap);
        ikisiDe.accept("Java");

        // LİSTE İLE KULLANIM
        List<String> isimler = Arrays.asList("Ali", "Ayşe");
        isimler.forEach(isim -> System.out.println("- " + isim));
    }
}
```

D. Supplier<T> - Üretir, parametre almaz

```
import java.util.function.Supplier;
import java.time.LocalDateTime;
import java.util.Random;

public class SupplierOrnek {
    public static void main(String[] args) {

        // Parametre almaz, T döner
        Supplier<String> merhabaDe = () -> "Merhaba Dünya!";
        Supplier<Integer> rastgeleSayi =
            () -> new Random().nextInt(100);
        Supplier<LocalDateTime> simdikiZaman =
            () -> LocalDateTime.now();
        Supplier<Double> piDegeri = () -> Math.PI;

        System.out.println(merhabaDe.get());
        System.out.println("Rastgele: " + rastgeleSayi.get());
        System.out.println("Şimdi: " + simdikiZaman.get());
        System.out.println("Pi: " + piDegeri.get());
```

```
        // 5 RASTGELE SAYI ÜRET
        System.out.println("\n5 Rastgele Sayı:");
        for (int i = 0; i < 5; i++) {
            System.out.println((i + 1) + ". " +
                rastgeleSayi.get());
        }
    }
}
```

E. BiFunction, BiPredicate, BiConsumer

```
import java.util.function.*;

public class BiOrnek {
    public static void main(String[] args) {

        // BiFunction<T, U, R> - İki parametre, R döner
        BiFunction<Integer, Integer, Integer> toplam =
            (a, b) -> a + b;
        BiFunction<String, String, String> birlestir =
            (s1, s2) -> s1 + " " + s2;

        System.out.println("Toplam: " + toplam.apply(5, 3));
        System.out.println("Birleştir: " +
            birlestir.apply("Merhaba", "Dünya"));

        // BiPredicate<T, U> - İki parametre, boolean döner
        BiPredicate<Integer, Integer> buyukMu = (a, b) -> a > b;
        BiPredicate<String, Integer> uzunlukKontrol =
            (metin, uzunluk) -> metin.length() > uzunluk;

        System.out.println("10 > 5 ? " + buyukMu.test(10, 5));
        System.out.println("'Merhaba' > 5 ? " +
            uzunlukKontrol.test("Merhaba", 5));

        // BiConsumer<T, U> - İki parametre, void döner
        BiConsumer<String, Integer> yazdirTekrar =
            (metin, sayi) -> {
                for (int i = 0; i < sayi; i++) {
                    System.out.println((i + 1) + ". " + metin);
                }
            };

        yazdirTekrar.accept("Java", 3);
    }
}
```

3. METHOD REFERENCE

3.1 Method Reference Nedir?

Method Reference = Mevcut bir metodu lambda gibi kullanma

4 Çeşit Method Reference:

```
// 1. Static metod referansı
ClassName::staticMethod

// 2. Instance metod referansı (nesne üzerinden)
object::instanceMethod

// 3. Instance metod referansı (tip üzerinden)
ClassName::instanceMethod

// 4. Constructor referansı
ClassName::new
```

3.2 Static Method Reference

```
import java.util.*;
import java.util.function.*;

public class StaticMethodReference {

    // Static metodlar
    public static int kareAl(int sayi) {
        return sayi * sayi;
    }

    public static boolean ciftMi(int sayi) {
        return sayi % 2 == 0;
    }

    public static void yazdir(String metin) {
        System.out.println("→ " + metin);
    }

    public static void main(String[] args) {

        // Lambda ile
        Function<Integer, Integer> kare1 = x -> kareAl(x);
        // Method reference ile (daha kısa!)
        Function<Integer, Integer> kare2 =
            StaticMethodReference::kareAl;

        System.out.println("Lambda: " + kare1.apply(5));
        System.out.println("Method Ref: " + kare2.apply(5));

        // Liste ile
```

```
        List<String> diller = Arrays.asList("Java", "Python");
        diller.forEach(StaticMethodReference::yazdir);
    }
}
```

3.3 Instance Method Reference

```
import java.util.*;

public class InstanceMethodReference {

    // Instance metod
    public void selamla(String isim) {
        System.out.println("Merhaba, " + isim + "!");
    }

    public static void main(String[] args) {

        InstanceMethodReference obj =
            new InstanceMethodReference();

        List<String> isimler = Arrays.asList("Ali", "Ayşe");

        // Lambda ile
        isimler.forEach(isim -> obj.selamla(isim));

        // Method reference ile
        isimler.forEach(obj::selamla);

        // String metodları
        List<String> kelimeler = Arrays.asList("java", "python");

        // Lambda: kelime -> kelime.toUpperCase()
        // Method reference: String::toUpperCase
        kelimeler.stream()
                 .map(String::toUpperCase)
                 .forEach(System.out::println);
    }
}
```

3.4 Constructor Reference

```
import java.util.function.*;
import java.util.*;

class Ogrenci {
    private String ad;
    private int numara;

    public Ogrenci() {
        this.ad = "Bilinmeyen";
        this.numara = 0;
```

```
    }

    public Ogrenci(String ad) {
        this.ad = ad;
        this.numara = 0;
    }

    public Ogrenci(String ad, int numara) {
        this.ad = ad;
        this.numara = numara;
    }

    @Override
    public String toString() {
        return "Ogrenci{ad='" + ad + "', numara=" +
               numara + "}";
    }
}

public class ConstructorReference {
    public static void main(String[] args) {

        // 1. PARAMETRESIZ CONSTRUCTOR
        Supplier<Ogrenci> olustur1 = Ogrenci::new;
        Ogrenci ogr1 = olustur1.get();
        System.out.println("Parametresiz: " + ogr1);

        // 2. TEK PARAMETRE
        Function<String, Ogrenci> olustur2 = Ogrenci::new;
        Ogrenci ogr2 = olustur2.apply("Ali");
        System.out.println("Tek parametre: " + ogr2);

        // 3. İKİ PARAMETRE
        BiFunction<String, Integer, Ogrenci> olustur3 =
            Ogrenci::new;
        Ogrenci ogr3 = olustur3.apply("Ayşe", 1001);
        System.out.println("İki parametre: " + ogr3);
    }
}
```

4. GERÇEK HAYAT ÖRNEKLERİ

4.1 Liste Filtreleme ve İşleme

```
import java.util.*;
import java.util.function.Predicate;

class Urun {
    private String ad;
    private double fiyat;
    private String kategori;

    public Urun(String ad, double fiyat, String kategori) {
        this.ad = ad;
        this.fiyat = fiyat;
        this.kategori = kategori;
    }

    public String getAd() { return ad; }
    public double getFiyat() { return fiyat; }
    public String getKategori() { return kategori; }

    @Override
    public String toString() {
        return String.format("%s (%.2f TL) - %s",
            ad, fiyat, kategori);
    }
}

public class UrunFiltrele {

    // Generic filtreleme metodu
    public static <T> List<T> filtrele(
        List<T> liste, Predicate<T> kosul) {

        List<T> sonuc = new ArrayList<>();
        for (T eleman : liste) {
            if (kosul.test(eleman)) {
                sonuc.add(eleman);
            }
        }
        return sonuc;
    }

    public static void main(String[] args) {

        List<Urun> urunler = Arrays.asList(
            new Urun("Laptop", 15000, "Elektronik"),
            new Urun("Mouse", 250, "Elektronik"),
            new Urun("Masa", 2000, "Mobilya"),
```

```
            new Urun("Klavye", 500, "Elektronik")
        );

        // Elektronik ürünler
        List<Urun> elektronik = filtrele(urunler,
            urun -> urun.getKategori().equals("Elektronik")
        );

        // 1000 TL'den pahalı
        List<Urun> pahali = filtrele(urunler,
            urun -> urun.getFiyat() > 1000
        );

        // Elektronik VE ucuz
        Predicate<Urun> elektronikMi =
            urun -> urun.getKategori().equals("Elektronik");
        Predicate<Urun> ucuzMu =
            urun -> urun.getFiyat() < 1000;

        List<Urun> elektronikVeUcuz = filtrele(urunler,
            elektronikMi.and(ucuzMu)
        );
    }
}
```

SONUÇ

Bu dokümanda Java Lambda Expressions'ın tüm temel konularını detaylı olarak inceledik:

- **Lambda Syntax:** Sözdizimi kuralları ve kullanım şekilleri
- **Functional Interfaces:** Predicate, Function, Consumer, Supplier
- **Method References:** Static, Instance, Constructor referansları
- **Gerçek Hayat Örnekleri:** Filtreleme ve liste işlemleri

Lambda Expressions, Java'da fonksiyonel programlamanın temelidir. Kodu daha kısa, okunabilir ve ifade edici hale getirir.

ÖNEMLİ HATIRLATMALAR

- Lambda'lar functional interface'ler ile çalışır
- Method reference daha temiz görünüm sağlar
- Tek satır lambda'da return ve süslü parantez opsiyonel
- Predicate ile and, or, negate kombinasyonları yapılabilir
- Function ile andThen ve compose zincirleme yapılabilir