JAVA LAMBDA EXPRESSİON

Hazırlayan

Zeyneb Eda YILMAZ

İçindekiler

1. Java Lambda Expression

Lambda ifadesi, işlevsel programlamayı kolaylaştıran, geliştirmeyi basitleştiren, koleksiyondan verilerin yinelenmesine, filtrelenmesine ve çıkarılmasına yardımcı olan anonim bir yöntemdir.

Lambda ifadesi, fonksiyonel arayüzün implementasyonunu sağlamak amacıyla kullanılır.

1.1. Functional Interface (Fonksiyonel Arayüz):

- Yalnızca bir abstract method içeren interface'e fonksiyonel arayüz denir.

```
public interface LambdaTest {
    public void getName(String name);
}
```

Şekil 1: LambdaTest isimli fonksiyonel sınıf

1.2. Lambda Expression Tanımlama:

- Lambda ifadesini şu şekilde tanımlarız:

Şekil 2: Lambda ifadesi tanımlama

- Şekilde 4'de gösterildiği gibi; eşitliğin sol tarafında, fonksiyonel sınıf adı ve değişken, eşitliğin sağ tarafında ise parametre, lambda operatörü ve metot gövdesi bulunmaktadır.
- Şekil 1'deki LambdaTest fonksiyonel sınıfının lambda ifadesi ile implementasyonu şu şekildedir:

```
public class Main {

    public static void main(String[] args) {
        LambdaTest lambdaTest = name -> System.out.println("Merhaba " + name);
        lambdaTest.getName("Zeyneb Eda");
    }
}
```

Şekil 3:LambdaTest fonksiyonel sınıfının implementasyonu

```
Merhaba Zeyneb Eda
```

Şekil 4: Şekil 3'deki kodun çıktısı

❖ Lambda ifadesi, birden çok parametresi olan metotları implement ederken de kullanılabilir.

```
public interface LambdaTest {
    public void getUser(String name, int age, String email);
}
```

Şekil 5: LambdaTest isimli fonksiyonel sınıf

```
public class Main {

    public static void main(String[] args) {
        LambdaTest lambdaTest = (name, age, email) -> System.out.println(
                "Kullanıcı Bilgileri: \n" + " Ad: " + name
                                          + "\n Yaş: " + age
                                          + "\n E-Posta Adresi: " + email);
        lambdaTest.getUser("Zeyneb Eda", 23, "zeynebedayilmaz@hotmail.com");
    }
}
```

Şekil 6: Birden çok parametre içeren metodun Lambda ifadesi ile implementasyonu

```
Kullanıcı Bilgileri:
 Ad: Zeyneb Eda
 Yaş: 23
 E-Posta Adresi: zeynebedayilmaz@hotmail.com
```

Şekil 7: Şekil 6'daki kodun çıktısı

❖ Lambda ifadesi, parametresiz olan metotları implement ederken de kullanılabilir.

```
public interface LambdaTest {
    public int getMinValue();
}
```

Şekil 8: LambdaTest isimli fonksiyonel sınıf

```
public class Main {

    public static void main(String[] args) {
        LambdaTest lambdaTest = () -> {
            return 8563;
            };
        System.out.println("Minimum Değer: "+ lambdaTest.getMinValue());
    }
}
```

Şekil 9: Parametresiz metodun Lambda ifadesi ile implementasyonu

```
Minimum Değer: 8563
```

Şekil 10: Şekil 9'daki kodun çıktısı

1.3. Lambda Gövdesi Çeşitleri:

1.3.1. Tek İfadeli Gövde (Expression Body):

- Tek ifadeli gövde, içerisinde tek bir ifadenin bulunduğu gövde çeşididir.
- "Expression Body" olarak da bilinir.
- Lambda operatöründen sonra direkt olarak ifade yazılır.

```
public static void main(String[] args) {
    LambdaTest lambdaTest = () -> System.out.println("Merhaba!");
}
```

Şekil 11: Tek ifadeli gövde örneği

1.3.2. Kod Bloğu İçeren Gövde (Block Body):

- Bu gövde çeşidinde, gövde içerisinde bir kod bloğu bulunur.
- "Block Body" olarak da bilinir.
- Lambda operatöründen sonra süslü parantez içerisine kod bloğu yazılır.

```
public static void main(String[] args) {
    LambdaTest lambdaTest = () -> {
        int price = 250;
        int doublePrice = price * 2;
        return doublePrice;
    };
}
```

Şekil 12: Kod bloğu içeren gövde örneği

1.4. Stream API

- Stream Api, koleksiyon içerisindeki veriler üzerinde işlemler yapılabilmesini sağlar.
- Filter, foreach, map, sorted, distinct gibi işlemler örnek olarak verilebilir.

1.4.1. Filter

- Koleksiyon içerisinde filtreleme işlemi yapmayı sağlar.

```
public static void main(String[] args) {
    List<String> places = new ArrayList<>();
    places.add("Tokat, Niksar");
    places.add("Tokat, Merkez");
    places.add("Ankara, Balgat");
    places.add("Ankara, Mamak");
    places.add("İstanbul, Beşiktaş");
    places.stream().filter((p) -> p.startsWith("Tokat")).forEach(System.out::println);
}
```

Şekil 13: filter() işlemi örneği

1.4.2. Foreach

- Koleksiyon içerisinde dolaşma özelliği sağlar.
- Genellikle koleksiyonu yazdırmak için kullanılır.
- Şekil 13'de kullanım örneği verilmiştir.

1.4.3. Map

- Koleksiyon içerisindeki verileri başka tiplere dönüştürme veya nesneler üzerinde işlem yapma özelliği sağlar.

```
public static void main(String[] args) {
    List<String> places = new ArrayList<>();
    places.add("Tokat, Niksar");
    places.add("Tokat, Merkez");
    places.add("Ankara, Balgat");
    places.add("İstanbul, Beşiktaş");
    places.stream().filter((p) -> p.startsWith("Tokat"))
                   .map((p) -> p.toUpperCase())
                   .forEach(System.out::println);
}
```

Şekil 14: map() işlemi örneği

1.4.4. Sorted

- Koleksiyon içerisindeki verileri sıralama özelliği sağlar.
- Yazı ise harf sıralaması, sayı ise küçükten büyüğe sıralama örnek olarak verilebilir.

```
public static void main(String[] args) {
    List<String> places = new ArrayList<>();
    places.add("Tokat, Niksar");
    places.add("Tokat, Merkez");
    places.add("Ankara, Balgat");
    places.add("İstanbul, Beşiktaş");
    places.stream().filter((p) -> p.startsWith("Tokat"))
                   .sorted()
                   .forEach(System.out::println);
}
```

Şekil 15: sorted() işlemi örneği

1.4.5. Distinct

- Koleksiyon içerisinde, tekrar eden verilerin tek veri olarak ele alınmasını sağlar.

```
public static void main(String[] args) {
    List<String> places = new ArrayList<>();
    places.add("Tokat");
    places.add("Ankara");
    places.add("İstanbul");
    places.add("Tokat");
    places.stream().distinct();
}
```

Şekil 16: distinct() işlemi örneği

```
Tokat, Ankara, İstanbul
```

Şekil 17: Şekil 16'daki kodun çıktısı

1.5. Lambda Expression ve Stream API

❖ Lambda ifadesi kullanılırken, Stream Api'nin özellikleri de kullanıldığında birçok işlem aynı anda gerçekleştirilebilir.

```
public interface StreamTest {
    public void getPlaces(String cityName);
}
```

Şekil 18: StreamTest isimli fonksiyonel sınıf

```
public static void main(String[] args) {
    List<String> places = new ArrayList<>();
    places.add("Tokat, Niksar");
    places.add("Tokat, Merkez");
    places.add("Ankara, Balgat");
    places.add("Ankara, Mamak");
    places.add("İstanbul, Beşiktaş");
    places.add("Tokat, Niksar");

    StreamTest streamTest = (cityName) -> {
        System.out.println(cityName + " ilinin ilçeleri: ");
        places.stream().filter((p) -> p.startsWith(cityName))
                       .map((p) -> p.toUpperCase())
                       .sorted()
                       .forEach((p) -> System.out.println(p));
    };
    StreamTest streamTest2 = (cityName) -> {
        System.out.println(cityName + " ilinin ilçeleri: ");
        places.stream().filter((p) -> p.startsWith(cityName))
                       .map((p) -> "İl-İlçe: "+ p)
                       .sorted()
                       .distinct()
                       .forEach((p) -> System.out.println(p));
    };
    streamTest.getPlaces("Tokat");
    System.out.println("-----------------------");
    streamTest2.getPlaces("Tokat");
}
```

Şekil 19: Stream API ve Lambda ifadelerinin aynı anda kullanımı

```
Tokat ilinin ilçeleri:
TOKAT, MERKEZ
TOKAT, NİKSAR
TOKAT, NİKSAR
-----------------------
Tokat ilinin ilçeleri:
İl-İlçe: Tokat, Merkez
İl-İlçe: Tokat, Niksar
```

Şekil 20: Şekil 19'deki kodun çıktısı

2. Kaynakça

- https://kerteriz.net/java-lambda-expressions-nedir/
- https://medium.com/@muhammetenesakcayir/java-8-stream-api-32dd3e9aa30f#:~:text=Java%20Stream%20API%2C%20Liste%20Dizi,u%C4%9Fundan%20dolay%C4%B1%20do%C4%9Frudan%20nesne%20almaz.
- https://www.mobilhanem.com/java-8-lambda-ifadeleri/