INNOVA-PATIKA

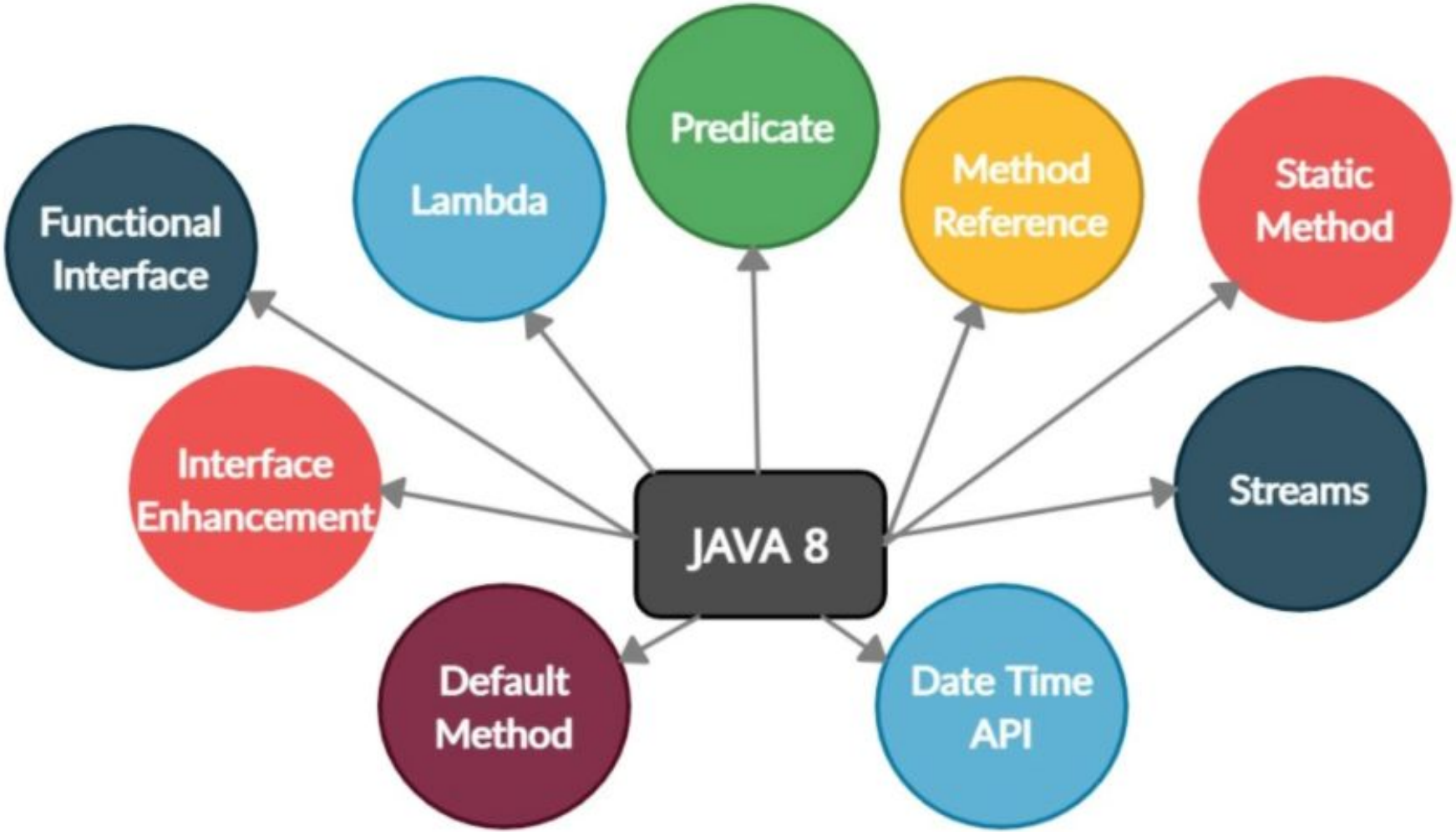
HOMEWORK 2

Hatice ÖZTÜRK
08/01/2022

Java 8 ve Java 9 ile gelen Yenilikler

- Lambda expressions
- Functional interfaces
- Default method ve Static method
- Method references
- Stream API
- Optional class
- Concurrency Enhancements
- JDBC Enhancements

- Moduler System
- Process API
- Variable Handles
- Compact String
- Concurrency
- Factory Method
- Stack Walking API
- jshell,jlink
- AOT Compiler
- G1 Garbage Collector

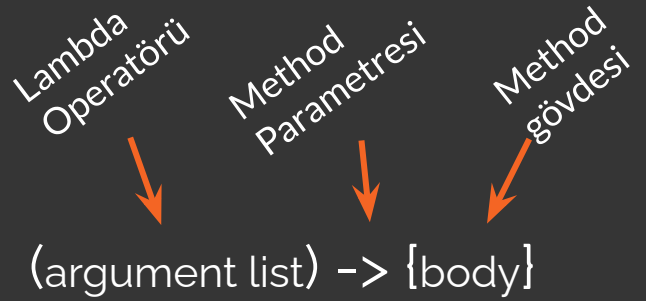

Functional Interface
Lambda
Predicate
Method Reference
Static Method
Interface Enhancement
JAVA 8
Streams
Default Method
Date Time API

1. Giriş

- 18 Mart 2014'te yayınlanan Java 8 bir çok yeni özelligi beraberinde getirdi.
- Java 7 den Java 8 e oldukça keskin bir geçiş oldu.
- Imperative programming den functional programming e geçiş oldu.

Lambda Expression

- Lambda ifadeleri kendi başlarına tanımlanabilen fonksiyonlardır. Yani bir sınıfa bağımlı olmalarına gerek kalmadan tanımlanabilirler.
- Lambda, Java 8 ile hayatımıza girmiş bulunmaktadır.
- Lambda ifadelerini tanımlamak için arrow operatörünü `->` kullanıyoruz. Arrow operatörünün sol tarafında lambda ifadesinin parametre listesi sağ tarafında ise implementasyonu yer alır.



İpucu

Lmbda ifadeleri bir fonksiyonel interface ile ilişkilendirilmelidir.

Neden Lambda Kullanmalıyız?

Lambda, kod miktarını azaltmak, daha sade, açık ve az kod yazmak için kullanılır.

Lambda ifadelerinin sonucu bir değişkene atanabilir veya bir fonksiyona parametre olarak geçebiliriz.

Ayrıca fonksiyonel programlama dünyasının kapısını açmamıza izin verir.

Java 8'den önce

```
public interface Animal {
    void name(String name);
}
```

```
public class Test {
    public static void main(String[] args) {
        Animal animal=new Animal(){
            @Override
            public void name(String name) {
                System.out.println("Hello "+name);
            }
        };
    }
}
```

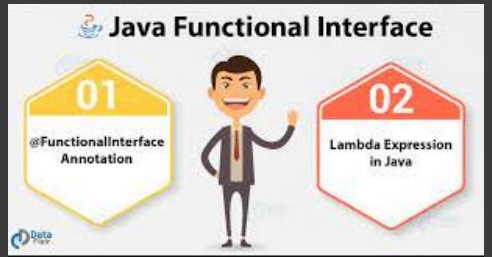
Java 8'den sonra

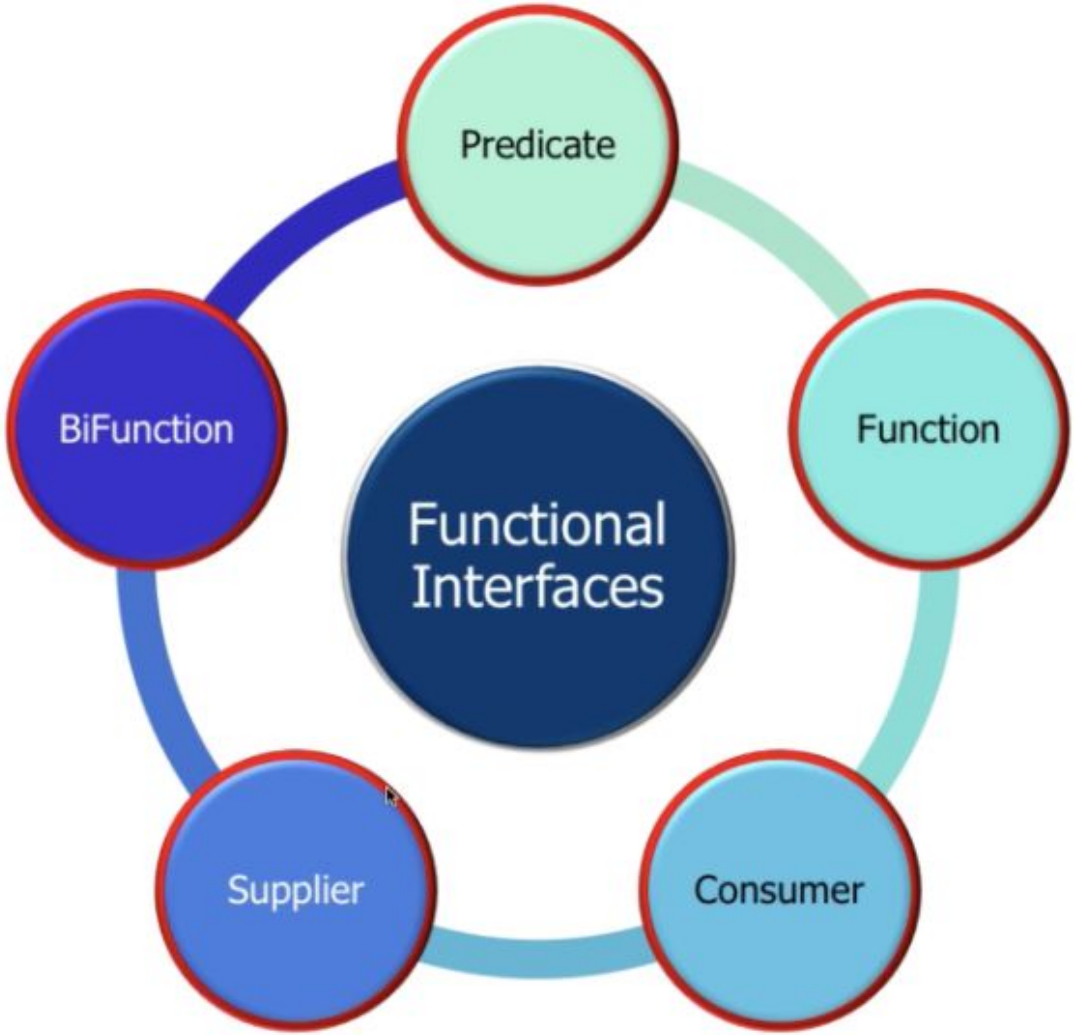
```
@FunctionalInterface
public interface Animal {
    void name(String name);
}
```

```
public class Test {
    public static void main(String[] args) {
        Animal animal=name -> {System.out.println(name);};
    }
}
```

Functional Interface

- Fonksiyonel interface özünde sadece bir abstract metod bulunduran interfacelere denir.
- Bu interfaceler sadece bir işe odaklanmıştır bu yüzden sadece bir absract method içerir. Bu methodun yanında private yada default yada static metodlarda içerebilirler fakat önemli olan sadece tek bir abstract methodun olmasıdır.
- Lambda ifadeleri ile sıkı bir ilişki içindedir. Hatta tek bir abstract method içermelerinin bir nedeni de lambda ifadeleri ile çalışabilmesini sağlamaktır.
- @FunctionalInterface anotasyonu ile interface'in bir functional intrface olduğunu belirtebiliriz. Bu anotasyonu kullanmak zorunlu değildir, fakat kullanımı; bu anotasyonu kullanan interface'e birden fazla soyut method eklendiğinde derleyici hatası alırız.





Java 8 functional paketinde bir çok functional interface tanımlanmıştır. Bu interfacelerden bazıları:

- Consumer
- Supplier
- Function
- Predicate

Functional Interface example

```java
@FunctionalInterface //optional
public interface FunctionalInterfaceExample {
    int operation(int num1, int num2);
    static String info (){
        return "this interface provides some operations";
    }
    default Integer findSquare(Integer number){
        return number*number;
    }
}
```

```java
public class Test {
    public static void main(String[] args) {
      FunctionalInterfaceExample addition =new FunctionalInterfaceExample()
      {
          @Override
          public int operation(int num1, int num2) {
              return num1+num2;
          }
      };
      FunctionalInterfaceExample multiplication=(x,y)->(x*y);

        System.out.println("5+5="+addition.operation(5,5));
        //çıktı: 5+5=10
        System.out.println("5*5="+multiplication.operation(5,5));
        //çıktı: 5*5=25
    }
}
```

Default method ve Static Method

- **Java 8 öncesinde herhangi bir Interface'de gövdeli method yazmak mümkün değilken, java 8 ile gelen default ve static keywordleri ile artık interface içerisinde gövdeli method yazmak mümkün.**

```
public interface Hayvan {
    default void beslenme(){
        System.out.println("Hayvanlar beslenir");
    }
    static void buyume(){
        System.out.println("Hayvanlar büyür ve gelişir");
    }
}
```

!!!!!

Pekii bir Class, aynı methoda sahip iki farklı interface'i implement ettiği zaman,çakışma durumu nasıl önlenir?

Eğer bir sınıf, aynı isimde varsayılan methoda sahip birden fazla arayüz uygularsa derleme zamanında hata oluşur.

Böyle çakışma durumunda, sınıf; default methodu override etmeli.

```java
public interface Canlı {
    default void beslenme(){
        System.out.println("Canlılar beslenir");
    }
}
```

```java
public interface Hayvan {
    default void beslenme(){
        System.out.println("Hayvanlar beslenir");
    }
    static void buyume(){
        System.out.println("Hayvanlar büyür ve gelişir");
    }
}
```

```java
public class Kedi implements Hayvan,Canlı{
    @Override
    public void beslenme() {
        System.out.println("Kedi beslenir");
        Hayvan.super.beslenme();
        Canlı.super.beslenme();
    }
}
```

Method Reference

- Method references da yine lamda ve functional interface domaini ile gelen ve bir arada kullanılabilen özelliklerinden biridir.
- Bazen lambda ifadeleri yerine kullanılabilir. Lambda ifadesinde objenin kendi methodlarından birini kullanıyorsak lambda ifadesi yerine direkt olarak method reference vererek yapabiliriz.

- **Static mehodlar için**

  <ClassName>::methodName;

- **non static methodlar için**

  <ObjectReference>::methodName;

Method Reference

```
public class Test {
    public static int staticMethod(int num1,int num2){
        return num1-num2;
    }
    public int nonStaticMethod(int num1,int num2){
        return num2-num1;
    }
    public static void main(String[] args) {
        List<Integer> numbers= Arrays.asList(2, 5, 13, 7, 43, 8);

        Test test=new Test();

        numbers.sort(Test::staticMethod);
        System.out.println(numbers.toString());//[2, 5, 7, 8, 13, 43]
        numbers.sort(test::nonStaticMethod);
        System.out.println(numbers.toString());//[43, 13, 8, 7, 5, 2]
    }
}
```

STREAM API

Stream API, Collectionlar üzerinde bazı işlemleri yapmayı kolaylaştıran bir yapıdır. Stream API, sayesinde sık kullanılan çeşitli operasyonları yapabilirsiniz. Bunlardan birkaçını şöyle sıralayabiliriz;

- filter(filtreleme)
- forEach(iterasyon)
- map(dönüştürme)
- reduce(indirgeme)
- distinct(tekilleştirme)
- limit(aralık)
- collect(türe dönüşüm)
- count(sayma)
- min/max

Stream türünden nesneler, çeşitli yollarla elde edilebilir. Bunlardan biri Collection API. Bu arayüzden türeyen tüm nesneler, stream() veya parallesStream() metodlarını çağırarak Stream türünden bir nesne ele edilmektedir.

stream() metodu ile elde edilen Strean nesnesi, yapacağı işlemleri ardışıl olarak yaparken, parallesStream() metoduyla elde edilen Stream nesnesi, bazı operasyonları paralel olarak yapmaktadır.

Stream API örnekleri

```java
List<String> names=new ArrayList();
names.add("Hatice");
names.add("Arzu");
names.add("Ahmet");
names.add("Ada");
// forEach
names.stream().forEach(name->{
    System.out.println(name);
});
names.stream().forEach(System.out::println);

//filter
System.out.println("filter example");
names.stream()
        .filter(name -> name.length()<5)
        .forEach(n->System.out.println(n));
```

```java
//distinct
//tekrarlı verileri çıkarır.
System.out.println("distinct example");
names.stream()
        .distinct()
        .forEach(System.out::println);

//sorted
System.out.println("sorted example");
names.stream()
        .sorted()
        .forEach(System.out::println);

//limit
//listenin ilk 2 elemanını alır
System.out.println("limit example");
names.stream()
        .limit(2)
        .forEach(System.out::println);
```

```java
//count
//eleman sayısını hesaplar.
System.out.println("count example");
System.out.println(names.stream().count());

//collect
//collect metodu, stream nesnelerini başka biçimdeki
//bir nesneye, veri yapısına dönüştürmek için kullanılır.
// to list
List list=names.stream()
        .collect(Collectors.toList());
names.stream().forEach(System.out::println);
// to set
Set set=names.stream()
        .collect(Collectors.toSet());
set.stream().forEach(System.out::println);
String collect=names.stream()
        .collect(Collectors.joining( delimiter: " - "));
// to map
Map<Integer,List<String>> mapList=names.stream()
       .collect(Collectors.groupingBy(name-> name.length()));
mapList.values().forEach(m->System.out.println(m));
```

```java
//map
//stream içerisindeki yığınsal olarak bulunan her
//bir veriyi dönüştürmeye yarar. function türünde bir
// parametre alır
System.out.println("MAP İŞLEMİ");
names.stream()
        .map(n->n.toUpperCase(Locale.ROOT))
        .forEach(System.out::println);

//reduce
//stream içerisindeki verilerin teker teker işlenmesidir.
names.stream()
        .reduce( identity: "1",(a,b)->{
            System.out.println(a+" "+b);
            return a+b;
        });
```

Optional Class

Java 8 ile birlikte gelen özelliklerden biri de bir objenin kullanılmadan önce yapılan null check'lerin daha okunabilir ve kontrol edilebilir olmasını sağlayan Optional yapısıdır.

```
//burada name null olursa exception fırlatır.
Optional.of(name);

//burada name objesi null da olabilir olmayadabilir.
// Herhangi bir exception fırlatmaz.
Optional.ofNullable(name);

//burada name objesi null değil ise isPresent methodu
//boolean true dönecektir.
Optional.ofNullable(name).isPresent();

//burada name objesi null ise exception fırlatır.
Optional.ofNullable(name).orElseThrow();
```

Concurrency Improvement

Java 8 ile birlikte yeni Concurrency API geliştirildi ve concurrent/multitasking işlemler anlaşılır hale geldi. Java 8 ile birlikte artık açık olarak Thread nesneleri oluşturma ve yönetme zorunluluğu kalmıyor.

Java 8 öncesi

```
Runnable task = () -> {
    String threadName = Thread.currentThread().getName();
    System.out.println("Hello " + threadName);
};

task.run();

Thread thread = new Thread(task);
thread.start();

System.out.println("Done!");
```

Java 8 sonrası

```
ExecutorService executor = Executors.newSingleThreadExecutor();
executor.submit(() -> {
    String threadName = Thread.currentThread().getName();
    System.out.println("Hello " + threadName);
});

// => Hello pool-1-thread-1
```

JDBC Enhancements

Java 8 ile birlikte artık JDBC-ODBC bridge desteklenmiyor. Oracle bu konuda database vendörün sağlayacağı JDBC-ODBC bridge'yi kullanmamızı öneriyor.

JDBCType, SQLType gibi birçok interface eklendi. Bazı güvenlik geliştirmeleri ile birlikte pure JDBC işlemleri Java 8 ile çalışacak hale getirildi.

Java 9

1. **Moduler Sistem :** 9 versiyonu ile gelen en önemli değişiklik moduler yapıdır. Önce java 7'e sonra java 8'e yetiştirmeye çalıştıkları değişiklik bu versiyon ile gelebildi. Peki neden böyle büyük bir değişikliğie gidildi.
    - Daha Güvenli Konfigürasyon: Program yazarken kimin neye bağımlı olduğunu açıkça belirtmek.
    - Daha Güçlü Enkapsülasyon: Sınıf içerisindeki enkapsülasyonu, sınıflarıda kapsayacak şekilde genişletmek. Böylece API içerisinden sadece istenilen sınıfların dışarı açılması.
    - Java'nın Ölçeklenebilir Olması: Özelleştirilmiş konfigürasyonlar ile istenilen modulleri içeren paketler çıkmak.
    - Performans

2- Process API : İşletim sistemi processlerinin yönetilmesine ve kontrol edilmesine yönelik iyileştirmeler.

3-Varibale Handles : Nesne alanları ve dizi öğeleri üzerinde java.util.concurrent.atomic ve sun.misc.Unsafe metotlarına karşılık gelen bir yenilik.

4-Compact String: String ifadelerin hafızada daha az yer kaplamasını hedefleyen yenilik.

5-Concurrreny: Flow API ile Reactive programlamanın temeli olan Publish-Subscribe yapısına yönelik yenilik.

5-Factory Method: Koleksiyon örneği oluştururken, koleksiyonun başlangıç elemanlarının verilmesini sağlayan bir yenilik.
(List<Integer> numbers = List.of(1,2,3,4);)

6-Stack-Walking API: Program'ın stack izini inceleyebilmeye yönelik bir yenilik.

7-jshell: REPL in java'ya eklenmesi. Komut satırından java kodu yazmamızı sağlayan yenilik.

8-AOT Compiler: JIT öncesi byte kodun native koda çevrilmesine dair yenilik.

9-jlink: Bir çalışma zamanı imajı oluşturmak için modül setlerini birbirine bağlayan bir komut satırı aracıdır.

10-G1 Garbage Collector: Default olarak kullanılır hale gelmesi

Kaynakça

https://devnot.com/2017/java-8-hakkinda-bilmeniz-gerekenler/

https://medium.com/@arif.erol16/java-8-ile-gelen-yenilikler-d9e896aca4cd

https://www.journaldev.com/2389/java-8-features-with-examples

https://kodedu.com/2014/09/java-8-method-reference/

https://bilisim.io/2017/12/10/java-9-yenilikleri-1-ne-oldu/