JAVA8 FONKSİYONEL PROGRAMLAMA MÜLAKAT HAZIRLIĞI (KAVRAM SORULARI)

Soru1:Fonksiyonel Programlama nedir? Java FP'yi destekler mi?

CEVAP: Problem çözümünü matematiksel fonksiyonların diğer bir ifade ile hazır metotlar üzerinden gerçekleştiren programlama stili. Java8 ile beraber Fonks. Programlama desteklenmeye başlanmıştır. FP ile kod yazmaktan ziyade probleme odaklanmak daha kolay olur. Daha kısa ve daha anlaşılabilir kodlar yazmak mümkündür.

Soru2: Javada FP hangi koşullarda kullanılabilir?

Cevap: Java dili, FP'yi kullanabilmek için lambda ifadelerinden faydalanmaktadır. Ancak Lambda fonksiyonlarını kullanmak için **Fonksiyonel arayüzler** kullanılmalıdır.

Soru3: Lambda ifadeleri nedir? Explicit ve İmplicit tanımlama ne demektir.

CEVAP: Fonksiyonel Programlamanın temeli Lambda Hesaplamasına dayanmaktadır. Lambda İfadeleri ise isimsiz fonksiyonlardır.

- Lambda fonksiyonları tanımlanırken değişken türleri tanımlanmaz ise → **implicit**
- Değişken türleri belirtiliyorsa → **explicit**

(parameterler) -> {expressions }

Soru4: Lambda fonksiyonları içerisinde new ve this operatörleri neden kullanılamaz?

CEVAP: Çünkü Lambda bir nesne değildir. Yani **Object** sınıfından türetilmemiştir. Bu yüzden içerisinde Nesne tabanlı metotlar kullanılmaz.

Soru5: Lambda ifadelerinde Erasure nedir?

CEVAP: Erasure derleyicinin bir lambda ifadesini aslında orijinal Java koduna çevirmesidir.

x -> System.out.println(x)

Derleyici bu **Lambda ifadesini** bir **static fonksiyona** çevirir.

```
public static void genName(Integer x) {
   System.out.println(x);
}
```

Soru6: Fonksiyonel Arayüzler nelerdir?

CEVAP: SADECE bir tek **abstract** (soyut) metodu olan arayüzlerdir. Default metotlara sahip olabilir. Sadece tek metot şartı, Lambda ifadelerine uygun olmasını sağlamak içindir. İstenirse @FunctionalInterface anotasyonu kullanılarak tanımlanabilir ama zorunlu değildir.

Soru7: Predicate Nedir? Predicate ile Function arayüzleri arasındaki farkları belirtiniz?

CEVAP: Java8'de fonksiyonel programlama kullanabilmek için 4 ana kategoride 40 dan fazla fonksiyonel interface tanımlanmıştır. Bu interfaceler **java.util.function**: kütüphanesinde tanımlanmıştır ve isimleri: **Function**, **Supplier**, **Consumer**, **Predicate**

Predicate<T> : Bir şartın değerlendirilmesini sağlayan fonksiyonel arayüzdür. Şarta göre boolean değer döndürür. Çalıştırmak için **test()** metodunu kullanır.

Function<T,R> : T tipinde parametre alan, **R tipinde sonuç** döndüren arayüzdür. Çalıştırmak için **apply()** metodu vardır.

Soru8: Consumer arayüzü Nedir?

CEVAP: Consumer<T> : **void** tipinde bir fonksiyonel arayüzdür. **accept()** metodu ile çalıştırılabilir.

Soru9: Supplier arayüzü Nedir?

CEVAP: Supplier<T>: Parametre almayan fonksiyonel arayüzdür. **get()** metodu ile çalıştırılır.

Soru10: High Order Function Nedir? Örnek verir misiniz?

CEVAP: Bir fonksiyonu (metodu) parametre olarak alabilen ve / veya sonuçları bir fonksiyon olarak döndürebilen fonksiyonlardır.

Sream API içerisinde bir çok HOF tanımlanmıştır. HOF'lar Fonksiyonel Interface'ler ile çalışabildiği için parametre olarak bir lambda fonksiyonu alabilmektedir.

Fonksiyonel Arayüz	Kullanım Örneği (HOF)
Consumer	forEach(Consumer),peek(Consumer)
Predicate	filter(Predicate)
Function	map(Function)
Supplier	reduce(Supplier), collect(Supplier)

Soru11: Metot Referansı nedir? Örnek ile açıklayınız.

Metot referansı Java 8 ile gelen bir özelliktir ve bir class'a ait bir metodun çağrılmasının değişik yöntemi gibi düşünülebilir. Genelde anlaşılabilirliği arttırdığı için Lambda fonksiyonları yerine Method referansı kullanımı tercih edilir.

Syntax:
Class adı :: Metot adı

Soru12: Stream API nedir?.

CEVAP: Büyük veri içeren nesnelerini (Collection v.b) fonksiyonel programlama ile işlememize imkan sağlayan bir API'dir. Verilerin sıralı işlemlerden (**pipeline**) geçirilerek işlenmesini ve istenilen sonuçların elde edilmesini sağlar. Java8 versiyonu ile gelmiştir ve **java.util.stream** paketinde yer almaktadır.

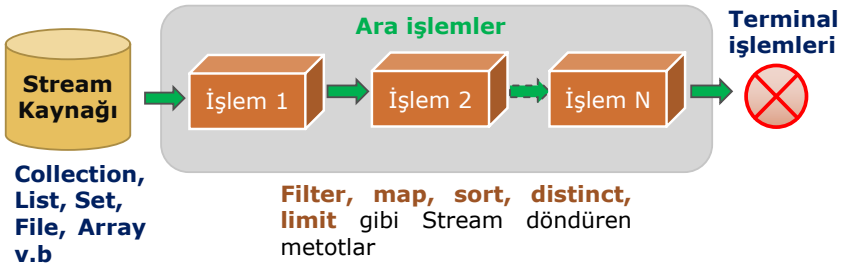
Stream API içerisindeki metotlar **Lambda ifadelerini** desteklemektedir. Metotlar içerisinde fonksiyonel arayüzler kullandığı için Lambda ifadelerini kullanmak mümkündür.

Soru13: Collection API ile Stream API arasındaki en temel fark nedir?

CEVAP: Collection API'de asıl amaç verileri saklamak iken Stream API'de asıl amaç veriler saklamak değil onları işlemek ve sonuç döndürmektir. Stream API verileri saklamaz inceler ve hesaplama yapar.

Soru14: Stream Pipeline (hattı) hangi işlemler içerebilir?

CEVAP: Bir Stream hattı; bir kaynak, O'nu takip eden 0 veya daha fazla **ara işlem** ve bir terminal (sonlandırıcı) işlem içerir.



Soru15: Bir dizi Stream'e döndürülebilir mi ?Örnekle açıklayınız.

CEVAP: evet. Stream.of() metodu ile bir dizi bir Stream haline çevrilebilir.

```
Integer[] dizi = { 3, 1, 4, 1, 5, 9 };
Stream<Integer> streamDizi = Stream.of(dizi);
```

Soru16: Stream API'deki Lazy ve Eager kavramlarını açıklayınız?

CEVAP: Stream'lerde ara işlemler, tembel (**LAZY**) olarak yürütülür. Yani terminal işlemi çağrılana kadar koşturulmazlar. Sadece, yeni bir stream nesnesi hazırlarlar. Terminal işlemi çağrıldığında ise bu stream'ler alınarak tek tek ara işlemler gerçekleştirilir ve sonuç oluşturulur.

Terminal işlemleri ise **EAGER'dır** yani Stream hattını kapatır ve stream nesnelerini tüketirler. Stream'lerin büyük veri gruplarında çalışacağı düşünüldüğünde terminal işlemini çağrılmadan tüm işlemleri baştan yapmak zaman kaybına yol açabilir. Belki de bazı işlemlerdeki veriler hiç kullanılmayabilir. Bu yüzden tembel davranmak daha efektif bir yöntemdir.

Soru17: filter() metodu ne işe yarar kısaca açıklayınız?

CEVAP: Filter() metodu bir ara işlemdir ve bir stream nesnesi içerisindeki elemanları filtreleyerek döndürür. Filtreleme işlemini ise içerisinde çağırdığımız bir lambda fonskiyonu veya metot referansına göre yapar. Çünkü filter() metodu aslında **bir Predicate <T>** interface'i ile çalışmaktadır.

Soru18: map() metodu ne maçla kullanılır?

CEVAP: Stream'deki verileri, verilen metoda göre **değiştiren** (transformasyon) ara işlem metodudur. Yani Stream'de bulunan bir verinin daha değişik bir halini almak istiyorsanız map() kullanmak gerekmektedir. Örneğin, isimler listesindeki tüm isimleri büyük harfe çevirmek için map(String::toLowerCase) kullanılabilir.

Soru19: Map ve flatMap metotlarının farkı nedir?

CEVAP: Map(), stream'leri transformasyona uğratmak için kullandığımız metottur. flatMap() ise iç içe (nested) stream nesnelerini tek bir stream nesnesine dönüştürmek için kullanılır.

İçiçe liste örneği

```
List<List<String>> list = Arrays.asList( Arrays.asList("a"),
Arrays.asList("b"));
```

Soru20: Stream pipeline'da reduction nedir? Örnek verebilir misiniz?

CEVAP: İndirgeme (***reduction***) *b*ir stream'in **bir türe veya bir primitive'e** dönüştürülmesini sağlayan bir terminal işlemdir. Reduction'da metotlar, ilgili işlemleri gerçekleştirip **tek bir değer** döndürmektedir.

Java 8 Stream API'de **average(), sum(), min(), max()** ve **count()** gibi tanımlanmış bir çok indirgeme metodu bulunmaktadır.

reduce() : Kendi indirgeme işlemlerimizi tanımlayabileceğimiz genel amaçlı bir metottur.

Soru21: Optional Class nedir?

CEVAP: Optional Class Java8 ile Javaya eklenmiştir. Optional Clas'ın asıl amacı NullException'ler ile uğraşmayı kolaylaştırmaktır. Bu class'ın içerisinde isPresent() orElse() v.b bir çok hazır metot bulunmaktadır.

Soru22: collect() metodu ne amaçla kullanılır?

CEVAP: Stream'de işlenen verilerin bir collection olarak saklanması isterirse collect() metodu kullanılabilir.

Soru23: Stream API ile paralel işleme kullanılabilir mi? Avantaj ve Dezavantajları nelerdir?

CEVAP: Stream API ile parallelStream() metodu ile paralel işleme kullanmak mümkündür. Avantajı kullanımının normal multi-threading programlamaya göre çok kolay olmasıdır. Ancak, paralel işlemenin ilk işlem fazlalığı (overhead) çok fazla olduğu için küçük veriler ile kullanmak faydadan çok zarar verebilir. Yani paralel işleme her zaman daha iyi sonuç vermeyebilir. Yüklü miktarda veri varsa ve karmaşık işlemler yapıyorsak kullanmak daha mantıklıdır.