- **تمرین 13 سوال دوم: تفاوت بین stream و collection چیست؟**

پاسخ: collections و Stream دو مفهوم متفاوت هستند که برای دو هدف متفاوت استفاده می شوند. در ادامه تفاوت این دو مفهوم به صورت مبسوط توضیح داده شده است:

تفاوت اول: مفهومی

Collections برای ذخیره و گروه بندی داده ها در یک ساختار داده خاص مانند list، Map، Set و... استفاده می شود ولی stream برای انجام عملیات پیچیده پردازش داده، مانند filtering، Matching، Mapping و... روی داده های ذخیره شده مانند Arrays، Collections و I/O resources کاربرد دارد. این یعنی collections در اصل درباره داده هاست و stream در اصل درباره عملیات روی داده هاست.

مثال:

```
List<String> names = new ArrayList<>();

names.add("Charlie");

names.add("Douglas");

names.add("Sundaraman");

names.add("Charlie");

names.add("Yuki");

names.stream().distinct()
     .forEach(System.out::println);
```

تفاوت دوم: دستکاری داده ها

ما می‌توانیم عناصری را به collections اضافه یا از آن حذف کنیم. اما در stream نمی‌توانیم. stream یک منبع را مصرف می‌کند، عملیاتی روی آن انجام می‌دهد و نتیجه را برمی‌گرداند اما نمی‌تواند آن را دستکاری کند.

مثال:

```
List<String> names = Arrays.asList("Charlie", "Douglas", "Jacob");
names.add("Sundaraman");
names.add("Yuki");
names.remove(2);
Stream<String> uniqueNames = names.stream().distinct();
```

تفاوت سوم: Internal Iteration VS External Iteration

مهمترین ویژگی stream ها در java8 این است که نیازی نیست که کاربر هنگام استفاده از آنها به iteration فکر کند و این کار توسط خود stream انجام می‌شود و فقط کافی‌ست کاربر عملیاتی را که می‌خواهد روی stream انجام دهد را مشخص کند، اما در مورد collections این طور نیست و لازم است که خود کاربر با استفاده از loop ها روی collections، iterate کند.

مثال:

```
for (String name : names) {
    System.out.println(name);
}

names.stream().map(String::toUpperCase)
     .forEach(System.out::println);
```

تفاوت چهارم: Traversal

streams تنها یک بار قابل پیمایش و traverse هستند یعنی اگر یکبار Stream را traverse کردیم به معنی مصرف شدن آن است و برای انجام دوباره این کار لازم است که یک stream جدید از آن منبع ساخته شود. اما collections می‌توانند چندین بار پیمایش و traverse شوند.

```
List<Integer> numbers = Arrays.asList(4, 2, 8, 9, 5, 6, 7);
Stream<Integer> numbersGreaterThan5 = numbers.stream()
                                    .filter(I -> I > 5);
numbersGreaterThan5.forEach(System.out::println);
```

تفاوت پنجم: Eager construction Vs Lazy construction

Collections مشتاقانه (eagerly) ساخته شده اند ، یعنی تمام عناصر در ابتدا محاسبه می شوند؛ اما streams با تنبلی (lazy) ساخته می‌شوند؛ یعنی تا زمان فراخوانی عملیات ترمینال، عملیات واسطه‌ای ارزیابی نمی شوند.

مثال:

```
List<Integer> numbers = Arrays.asList(4, 2, 8, 9, 5, 6, 7);
numbers.stream()
      .filter(i -> i >= 5)
      .limit(3)
      .forEach(System.out::println);
```

در اینجا وقتی عملیات چاپ فراخوانی می شود، اعداد ارزیابی میشوند که 3 عدد بزرگتر از 5 برای چاپ کردن یافت شود.

- **تمرین 13- سوال 3: تفاوت بین map و flatMap چیست؟**

Map و FlatMap در امضا، تعریف و نحوه استفاده تفاوت‌هایی دارند که در ادامه توضیحات کامل آن داده شده است:

1) تفاوت در امضا:

```
Map: <R> Stream<R> map(Function<? super T, ? extends R>
mapper);

FlatMap: <R> Stream<R> flatMap(Function<? super T, ? extends
Stream<? extends R>> mapper);
```

2) تفاوت در تعریف:

Map و FlatMap هر دو یک تابع mapping می گیرند که به هر یک از اعضای Stream<T> اعمال شده است؛ و یک Stream<R> برمی‌گردانند. تنها تفاوت شان در این است که تابع mapping در مورد FlatMap یک stream از مقادیر جدید میسازد. درحالیکه Map برای هر یک از عناصر ورودی یک مقدار معین تولید می‌کند.

flatMap() برای mapping function معمولا از ...و List.stream()، Arrays.srtream() استفاده میکنند. چون mapping function برای flatMap() یک stream دیگر را برمی‌گرداند، ما باید stream ای از streamها بگیریم. با این وجود flatMap() اثر جایگزینی هر streamتولید شده با محتویات آن جریان را دارد. به بیان دیگر تمام stream های جدا گانه ایجاد شده توسط این Function در یک stream واحد یک دست می شوند.

3) تفاوت در استفاده و کاربرد:

چون map() یک stream شامل نتیجه اعمال function داده شده به عناصر stream ورودی، تولید می‌کند. معمولا از آن برای تبدیل stream ای از یک نوع به stream ای از نوع دیگر استفاده می‌شود. به عنوان مثال تبدیل لیستی از Character به لیستی از Integer :

```
Stream.of('1', '2', '3').map(String::valueOf).map(Integer::parseInt);
```

کد بالا با استفاده از عبارت لامبدا میتواند به صورت زیر خلاصه شود:

```
Stream.of('1', '2', '3').map(ch -> Integer.parseInt(ch.toString()));
```

ممکن است این سوال پیش بیاید که چرا به flatMap() نیاز داریم؟

در نظر بگیرید که یک List<List<Integer>> داریم و می‌خواهیم یک تک لیست از Integer داشته باشیم که شامل تمام اعضای هر یک از لیست‌های داخلی باشد، این کار را با کمک flatMap() به صورت زیر می‌توانیم انجام دهیم:

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
class StreamUtils {
	public static void main(String[] args) {
		List<Integer> a = Arrays.asList(1, 2, 3);
		List<Integer> b = Arrays.asList(4, 5);
		List<Integer> c = Arrays.asList(6, 7, 8);
		List<List<Integer>> listOfListOfInts = Arrays.asList(a, b, c);

		System.out.println("Before flattening : " + listOfListofInts);
		List<Integer> listOfInts = listOfListOfInts.stream()
					.flatMap(list -> list.stream())
					.collect(Collectors.toList());

		System.out.println("After flattening  : " + listofInts);
	}
}
```

Output:

```
Before flattening : [[1, 2, 3], [4, 5], [6, 7, 8]]
After flattening : [1, 2, 3, 4, 5, 6, 7, 8]
```