□□□ قدفت في افاج

لتعالجة 8 □□□□ في تقديمها تم التي □□□□□□ واجهة من الاستفادة يمكنك ،□□□□ في java.util.stream لاستخدام على والتقليل والتحويل، التصفية، مثل عمليات بإجراء لك السيلات تسمح ووضحية. وظيفية بطريقة البيانات مجموعات java.util.stream استخدام كيفية سأشرح أدناه، ووضوح. فعال بشكل المصفوفات□ أو القوائم□ مثل العناصر تسلسلات سيوضح هذا الصحيحة. الأعداد من قائمة من 10 من الأكبر الزوجية للأعداد المتوسط على العثور محدد: مثال مراجعة خلال من والمفاهيم. للسيلات الرئيسية العمليات

---

java.util.stream على عامة نظرة

الجبيانات. لمعالجة إلخ□ IntStream, DoubleStream □مثل صلة ذات وطبائع Stream واجهة java.util.stream حزمة يوفر **المتوسطة العمليات** □ هي: العمليات هذه أنبوب. في تنفيذها يتم التي العمليات يدعم العناصر من تسلسل هو السيل استدعاء عند إلا تنفيذها يتم ولا بطيئة العمليات هذه جديد. سيل وإرجاع، □filter, map□مثل السيل تصفية أو تحويل المعالجة أنبوب يثير مما ،□average, collect□مثل جانبي تأثير أو نتيجة إنتاج :**النهائية العمليات** □ نهائية. عملية للبيانات.

أو لتحويل المتوسطة العمليات تطبيق .2 قائمة□. □مثل البيانات مصدر من سيل إنشاء .1 ما: عادةً السيلات، لاستخدام الجبيانات. تصفية .3 استخدام نتيجة. لإنتاج نهائية عملية

المشكلة على مثال

أرقام أي هناك يكن لم إذا .10 من الأكبر الزوجية الأعداد جميع متوسط حساب ,List<Integer> قائمة مع المشكلة: هذه نحل دعونا java.util.stream. باستخدام ذلك القيام كيفية إليك .0.0 ارجع هذه، مثل

بخطوة خطوة حل

1. **سيل إنشاء**
   - □ أبدأ مع List<Integer> لمثل□ List.of(1, 2, 12, 15, 20, 25, 30)□.
   - □ استخدم طريقة stream() لإنشاء Stream<Integer>:

     ```
     list.stream()
     ```
2. **تصفية السيل**
   - □ استخدم طريقة filter لحفظ فقط الأعداد التي هي زوجية وأكبر من 10.
   - □ أخذ طريقة filter Predicate□ دالة□ ترجع قيمة□ منطقية□ كعبارة: لامدا

     ```
     .filter(number -> number % 2 == 0 && number > 10)
     ```
     - □ number % 2 == 0 يحدد إذا كانت الأرقام زوجية.

- `number > 10` من أكبر الرقم أن يضمن 10.
- بالنسبة للقائمة المثالية `[1, 2, 12, 15, 20, 25, 30]`, هذا يحتفظ بـ `[12, 20, 30]`.

3. **تحويل إلى IntStream**

- نظرًا لأن `average()` متاحة على السيلات الأولية مثل `IntStream` وليس `Stream<Integer>`، قم بتحويل `Stream<Integer>` إلى `IntStream` باستخدام `mapToInt`:

```
.mapToInt(i -> i)
```

- `i -> i` يفتح كل `Integer` إلى `int`. يمكنك أيضًا استخدام `Integer::intValue`.
- هذا يعطي `IntStream` من `[12, 20, 30]`.

4. **حساب المتوسط**

- استخدم طريقة `average()` على `IntStream`، والتي ترجع `OptionalDouble` لأن السيل قد يكون: فارغًا

```
.average()
```

- بالنسبة لـ `[12, 20, 30]`, هذا يحسب `(12 + 20 + 30) / 3 = 20.666...`
- إذا كان السيل فارغًا، فإنه يرجع `OptionalDouble` فارغًا.

5. **التعامل مع الحالة الفارغة**

- استخدم `orElse(0.0)` على `OptionalDouble` لإرجاع `0.0` إذا لم يكن هناك أرقام متفي: بالتصفية

```
.orElse(0.0)
```

- بالنسبة لـ `[12, 20, 30]`, هذا يرجع `20.666...`
- بالنسبة لمثل `[1, 3, 5]` الذي يوجد أرقام زوجية، يرجع `0.0`.

الكود الكامل هنا هو الحل الكامل كطريقة:

```
import java.util.List;

public class Main {
    public static double averageOfEvenGreaterThanTen(List<Integer> list) {
        return list.stream()
                .filter(number -> number % 2 == 0 && number > 10)
                .mapToInt(i -> i)
                .average()
                .orElse(0.0);
    }

    public static void main(String[] args) {
        //
        System.out.println(averageOfEvenGreaterThanTen(List.of(1, 2, 12, 15, 20, 25, 30)));  // ~20.666
        System.out.println(averageOfEvenGreaterThanTen(List.of(1, 3, 5)));                   // 0.0
        System.out.println(averageOfEvenGreaterThanTen(List.of(12)));                        // 12.0
```

```java
        System.out.println(averageOfEvenGreaterThanTen(List.of()));                    // 0.0
        System.out.println(averageOfEvenGreaterThanTen(List.of(10, 8, 6)));            // 0.0
        System.out.println(averageOfEvenGreaterThanTen(List.of(11, 13, 14, 16)));      // 15.0
    }
}
```

توضيحها تم التي java.util.stream لـ الرئيسية الميزات

- التسلسل: متي تسلسل العمليات في نمط مرير مثل .filter().mapToInt().average().
- البطء: مثل العمليات المتوسطة filter و mapToInt لا يتم تنفيذها إلا عند استدعاء العملية النهائية average.
- الأولية السيالات: IntStream يوفر طرقًا مثل average() مخصصة لعمليات العددية، مما يوفر تجنب تكلفة التعبئة.
- الاختيار مع الجة: OptionalDouble عالج الحالات التي لا يوجد فيها نتيجة، مع orElse يوفر قيمة افتراضية.

البديلة الطريقة

يمكنك أيضًا استخدام طائفة Collectors:

```java
import java.util.stream.Collectors;

double average = list.stream()
                     .filter(number -> number % 2 == 0 && number > 10)
                     .collect(Collectors.averagingInt(i -> i));
```

- البسيط المثال لهذا مباشرة أكثر mapToInt().average() ذلك، ومع الفارغ. للسيل 0.0 ويرجع مباشرة Double يرجع هذا الأولية. السيالات استخدام على ويوافق

السيالات استخدام متى

- استخدم java.util.stream للكود الواضح والمفهم عند معالجة المجموعات.
- الرغم على التوازي، من للاستفادة stream() من بدلاً parallelStream() استخدام اعتبر الكبيرة، للبيانات بالنسبة التكلفة. بسبب الصغيرة للقوائم اللازم من أكثر هو هذا أن من

تصفية، سيل، إنشاء الخطوات هذه تعديل يمكنك عملية. مشكلة لحل java.util.stream استخدام كيفية المثال هذا يوضح الحاجة! حسب أخرى حالات إلى وتقليل تحويل،