Check und Prepare

Lösungsvorschläge

TECHNISCHE UNIVERSITÄT DARMSTADT

Autoren: Nhan Huynh und Darya Nikitina
Fachbereich: Informatik
Übungsblatt: 10
Version: 20. November 2021
Semesterübergreifend

V1 Theoriefragen

★★★

Welche der folgenden Aussagen ist wahr?

(A) Streams können aus Listen und Arrays erzeugt werden.

(B) Man kann nur mit Iteratoren über die einzelnen Elemente in einem Stream gehen.

(C) Ein Objekt der Klasse `Path` verwaltet den Pfadnamen einer Datei oder eines Verzeichnisses oder eines anderen Objektes, das im jeweiligen Dateisystem über einen Pfadnamen ansprechbar ist.

(D) Streams können in sich Daten speichern.

(E) Byteweiser Zugriff ist nur dann sinnvoll, wenn eine Datei nur Text und keine Bilder, Audio- oder Videodateien enthält, da bei einem Text die Bytes leichter gelesen werden können.

(F) `Runnable` ist ein Functional Interface. Die funktionale Methode heißt `run`, hat keine Parameter und ist `void`.

(G) Einzelne Threads können nicht terminiert werden, sie enden alle mit dem Ende des Gesamtprogramms.

(H) Eine Parallelisierung mit Threads verkürzt immer die Laufzeit eines Programms.

(I) Wann immer auf einen Button geklickt wird, wird Methode `actionPerformed` jedes bei diesem Button registrierten Listeners aufgerufen.

(J) Für jede Art von Listener gibt es eine eigene Registrierungsmethode.

(K) Die Klasse `Canvas` bietet eine unbegrenzte Zeichenfläche in einem Fenster.

Übungsblatt 10: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

(A) Wahr. Jede Collection besitzt die Methode `stream()`. Da das Interface `List` `Collection` erweitert, kann sie ebenfalls auf die Methode zugreifen. Um einen Stream aus einem Array zu erzeugen, gibt es folgende Möglichkeiten:

- `Arrays.stream(T[] array)`
- `Stream.of(T... values)`
- `Arrays.stream(...)` gibt es ebenfalls für primitive Datentypen, außer für `byte` und `char`.

(B) Wahr, da Streams nicht mit einem Index arbeiten und man dadurch nicht auf ein einzelnes Element im Stream zugreifen kann.

(C) Wahr. Außerdem muss es zum Pfadnamen in einem Path-Objekt keine Datei o.ä. im Dateisystem geben (sonst könnte man damit ja auch keine neue Datei erzeugen).

(D) Inkorrekt, Streams können zwar Daten in sich speichern, jedoch sind diese zur Bearbeitung der Daten gedacht und nicht zur permanenten Speicherung der Daten.

(E) Inkorrekt, da byteweiser Zugriff besser für Bild, Video oder Audiodaten geeignet ist. Generell sollte man für Texte **keine** Bytedaten verwenden.

(F) Wahr, es existiert ein Functional-Interface namens `Runnable` mit der erwähnten Methode (Package `java.lang`.

(G) Inkorrekt, man kann einzelne Threads zu jeder Zeit stoppen ohne auf das Ende des Gesamtprogramms zu warten.

(H) Inkorrekt, da Threads auch für mehr Laufzeit sorgen können, da sie selbst auch Laufzeit zu einem Programm hinzufügen. Also, wenn man zu viele Threads verwendet, kann das Programm langsamer als das ursprüngliche Programm ohne Threads werden.

(I) Wahr, da ein Button immer alle zu sich hinzugefügten Listener aufruft.

(J) Wahr.

(K) Wahr, jedoch wird das Canvas nur innerhalb des Fensters angezeigt.

Information: Die Abbildung 1 stellt eine beispielhafte Visualisierung von einem Stream dar und hilft Ihnen vielleicht Streams besser zu verstehen.

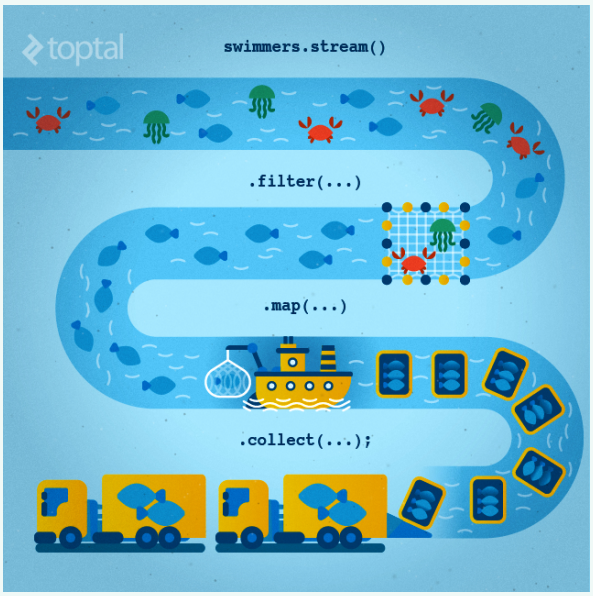


Abbildung 1: Quelle
https://www.exclamationlabs.com/blog/refactoring-for-java-8-streams/

Übungsblatt 10: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V2 Vereinfachung mittels Lambda-Ausdrücken ★☆☆

Gegeben sei nachstehender Java-Code. In diesem wird aus einer Liste von Zahlen der Durchschnittswert aller *positiven geraden Zahlen* berechnet, in einer Variablen gespeichert und zurückgegeben.

```java
/**
 * Computes the average of even numbers.
 *
 * @param numbers the array to compute the avearage of even numbers
 * @return the average of even numbers
 */
double averageOfEvenNumbers(int[] numbers) {
  int sum = 0, count = 0;
  for (int i = 0; i < numbers.length; i++) {
    // Only count even numbers and positive
    if (numbers[i] > 0 && numbers[i] % 2 == 0) {
      sum += numbers[i];
      count++;
    }
  }
  // int/int = int, one of the operand must be double to get a double as result
  return ((double) sum) / count;
}
```

1. Nutzen Sie Streams und Lambda-Ausdrücke um den Code zu verkürzen.
2. Beide Implementierungen – die „normale“ und die `Stream`-Variante – weisen eine konzeptionelle Unsauberkeit auf. Bei beiden kann es zum Auftreten einer `Exception` kommen.
   (a) An welcher Stelle kann es bei der „normale“ Implementierung zu einer `Exception` kommen? Um welche handelt es sich?
   (b) An welcher Stelle kann es bei der Implementierung mittels Streams zu einer `Exception` kommen? Um welche handelt es sich?
3. Modifizieren Sie den Code entsprechend, dass die Exceptions aus Teilaufgabe 2 gar nicht mehr auftreten können, sondern die Methode für alle übergebenen Arrays fehlerfrei durchläuft.

Übungsblatt 10: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

1.

```
1 double averageOfEvenNumbers(int[] numbers) {
2   return Arrays.stream(numbers).filter(x -> x > 0 && x % 2 == 0).average().getAsDouble();
3 }
```

2. (a) Eine `NullPointerException` wird geworfen, falls der übergebene Parameter gleich `null` ist. Ansonsten gibt es eine weitere konzeptionelle Unsauberkeit: Wenn der Wert von `count` 0 ist, würde die Methode zwar keine `Exception` werfen, jedoch wird `Double.NaN` zurückgegeben.

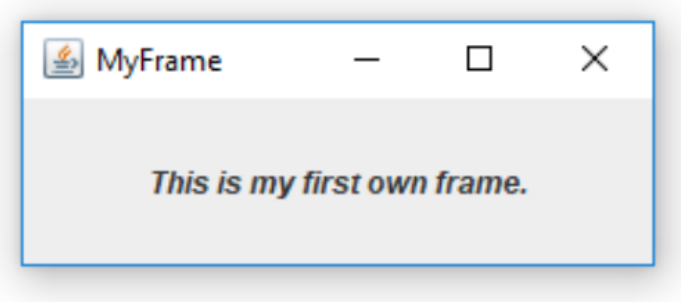
   (b) Wie bei der „normalen“ Implementierung wird hier ebenfalls eine `NullPointerException` geworfen. Des weiteren weist die Implementierung eine weitere konzeptionelle Unsauberkeit auf: Falls die nach dem Filter der Stream leer ist, gibt der Aufruf von `average()` einen leeren `OptionalDouble` zurück. Wenn wir nun `getAsDouble()` aufrufen, wird eine `NoSuchElementException` geworfen, da das `OptionalDouble` keine Werte enthält.

3. Man kann die Probleme bei der Stream-Variante beheben, indem man statt `getAsDouble()` `orElse(0)` aufruft, wodurch entweder der vorhandene Wert des `OptionalDouble` oder, wenn kein Wert vorhanden ist, 0 zurückgegeben wird. In der „normalen“ Variante kann man die Probleme beheben, indem man vor oder in Zeile 9 auf `count` > 0 testet und bei `count` ≤ 0 einen Standardwert (z.B: 0) zurückgibt. Um die `NullPointerException` in beiden Implementierung zu umgehen, reicht zu Beginn eine Abfrage, ob das Array gleich `null` ist.

Übungsblatt 10: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V3 Mein eigenes kleines Fenster ★☆☆

Implementieren Sie ein kleines Programm mit einer GUI, welche genauso aussieht wie in der nachfolgenden Abbildung:



Lösungsvorschlag:

```java
public class MyFrame extends JFrame {

  private static final int[] SIZE = {250, 100};
  private JLabel label;

  public MyFrame() {
    super("MyFrame");

    this.label = new JLabel("This is my first own frame.");
    this.label.setHorizontalAlignment(JLabel.CENTER);
    Font font = new Font(label.getFont().getName(), Font.BOLD + Font.ITALIC, 12);
    this.label.setFont(font);

    this.add(label);

    this.setSize(SIZE[0], SIZE[1]);

    // Center
    this.setLocationRelativeTo(null);
    this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    this.setVisible(true);
  }
}
```

Übungsblatt 10: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V4 Innere Klassen und Scope

Betrachten Sie den untenstehenden Java-Code, welcher eine innere Klasse `InnerClass` enthält. In dieser wiederum ist eine Methode definiert, die einen Lambda-Ausdruck enthält, dessen Operation mit der Methode `accept` des Interface `Consumer` aufgerufen wird.

```
1  public class LambdaScope {
2
3    public int x = 0;
4
5    class InnerClass {
6
7      public int x = 11;
8
9      void methodInInnerClass(int x) {
10       int z = 55;
11
12       Consumer<Integer> myConsumer = (y) -> {
13         System.out.println("x = " + x);
14         System.out.println("y = " + y);
15         System.out.println("this.x = " + this.x);
16         System.out.println("LambdaScope.this.x = " + LambdaScope.this.x);
17         System.out.println("z = " + z);
18       };
19       myConsumer.accept(x);
20     }
21   }
22
23   public static void main(String[] args) {
24     LambdaScope scope = new LambdaScope();
25     LambdaScope.InnerClass fl = scope.new InnerClass();
26     fl.methodInInnerClass(123);
27   }
28 }
```

Welche Ausgabe wird bei der Ausführung des Codes auf der Konsole ausgegeben? Überlegen Sie sich die Antworten ohne die Nutzung eines Compilers!

Lösungsvorschlag:

- `x = 123`
- `y = 123`
- `this.x = 11`
- `LambdaScope.this.x = 0`
- `z = 55`

Übungsblatt 10: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V5 Zeilen nummerieren

Implementieren Sie die Methode `void insertRowNumbers(String path)`, welche den Pfad einer Text-Datei als `String` übergeben bekommt. Die Methode soll den Text der Datei Zeile für Zeile einlesen. Beginnend ab der ersten Zeile soll dann in jeder zweiten Zeile nun die Zeilennummer eingefügt werden. War die alte Text-Datei folgendermaßen aufgebaut: `“Row1 \n Row2“` so soll die neue Text-Datei so aussehen: `“1 Row1 \n 2 Row2“`. Dabei steht `“\n“` für einen Zeilenumbruch.

Lösungsvorschlag:

```java
/**
 * Inserts the row number to each line to the given file.
 *
 * @param path path of the file, where we insert row numbers to each line
 * @throws IOException
 */
void insertRowNumbers(String path) throws IOException {
  BufferedReader reader = new BufferedReader(
  new InputStreamReader(new FileInputStream(path)));
  StringBuilder sb = new StringBuilder();
  String line;
  int i = 1;
  while ((line = reader.readLine()) != null) {
    sb.append(i).append("\n").append(line).append("\n");
    i++;
  }
  reader.close();
  BufferedWriter writer = new BufferedWriter(
      new OutputStreamWriter(new FileOutputStream(path), "UTF-8"));
  witer.write(sb.toString());
  writer.close();
}
```

Übungsblatt 10: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V6 Bäckerei gesucht ★★★

Sie planen eine Party und benötigen eine Menge Brötchen dafür. Sie suchen nun den Bäcker in Ihrer Umgebung, der Ihnen die günstigsten Brötchen verkaufen kann. Dafür gibt es Objekte des Typs `Bakery`, welche zum einen zwei `public double`-Attribute `distance` (gibt die Distanz zu Ihnen in km an) und `price` (gibt den Preis pro Brötchen an) besitzt. Außerdem gibt es Objekte des Typs `BakeryOffer`, welche ebenfalls zwei `public`-Attribute `bakery` vom Typ `Bakery` und `totalPrice` vom Typ `double` besitzt. Der Konstruktor der Klasse sieht folgendermaßen aus:

```
1 public BakeryOffer(Bakery b, double tp) {
2   this.bakery = b;
3   this.totalPrice = tp;
4 }
```

Implementieren Sie nun eine `public`-Methode vom Rückgabetyp `List<BakeryOffer>` mit dem Methodenkopf:

```
sortedBakeryOffers(Collection<Bakery> bakeries, double maxDistance)
```

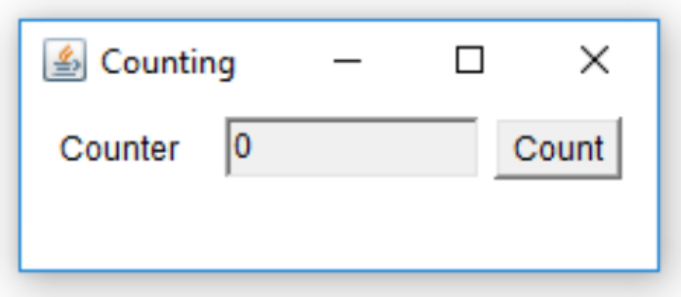
Die Methode erhält eine Sammlung von Bäckereien und die maximale Distanz zu einer solchen. Zurückgeliefert werden soll eine nach Gesamtpreis aufsteigend sortierte Liste mit Angeboten des Typs `BakeryOffer`. Bäckereien, die weiter als die übergebene maximale Distanz entfernt sind, sollen von der Betrachtung ausgeschlossen werden.

Lösungsvorschlag:

```
1  /**
2   * Returns a list of the bakeries sorted by total price within the specified distance.
3   *
4   * @param bakeries    the collection of bakeries which should be filtered
5   * @param maxDistance tbe maximum distance of a bakery to be considered
6   * @return a list of the bakeries sorted by total price within the specified distance
7   */
8  List<BakeryOffer> sortedBakeryOffers(Collection<Bakery> bakeries,
9      double maxDistance) {
10   return bakeries.stream().filter(bakery -> bakery.distance <= maxDistance)
11       .map(b -> new BakeryOffer(b, b.price))
12       .sorted(Comparator.comparingDouble(o -> o.totalPrice)).collect(Collectors.toList());
13 }
```

V7 Zähler ★★★

Implementieren Sie ein kleines Programm mit einer GUI, welche genauso aussieht wie in der nachfolgenden Abbildung:



Jedes Mal wenn der *Count*-Button gedrückt wird, soll sich der Wert im Feld um 1 erhöhen.

Hinweis: Ihr Programm besteht aus drei Komponenten:

1. `java.awt.Label` `"Counter"`
2. [a]non-editable `java.awt.TextField`
3. `java.awt.Button` `"Count"`

[a] http://programmedlessons.org/java5/Notes/chap60/ch60_13.html

Übungsblatt 10: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```
1  public class Counter extends Frame {
2
3    private static final int[] SIZE = {250, 100};
4    private Label counterLabel;
5    private TextField counterField;
6    private Button countButton;
7    private int counter = 0;
8
9    public Counter() {
10     super("Counting");
11     counterLabel = new Label("Counter");
12
13     counterField = new TextField(String.valueOf(counter), 10);
14     counterField.setEditable(false);
15
16     countButton = new Button("Count");
17     countButton.addActionListener(
18         e -> counterField.setText(String.valueOf(++counter)));
19
20     add(counterLabel);
21     add(counterField);
22     add(countButton);
23
24     setLayout(new FlowLayout());
25     setSize(SIZE[0], SIZE[1]);
26     setLocationRelativeTo(null);
27     addWindowListener(new WindowAdapter() {
28       @Override
29       public void windowClosing(WindowEvent e) {
30         dispose();
31       }
32     });
33     setVisible(true);
34   }
35 }
```