M411 LB03

Beer Webservice

O'Connor Christopher
Christopher.oconnor@edu.tbz.ch

1 INHALTSVERZEICHNIS

2 EINLEITUNG

3 PROJEKTWAHL

Ich wollte zu Beginn ein Projekt wählen mit dem höchsten Schwierigkeitsgrad. Einerseits für eine gute Note andererseits auch um mich selbst zu fördern und um einer spannender Arbeit nach zu gehen.

Da ich leider den Unterricht durch Krankheit verpasste, bei dem XML und JSON behandelt wurden, konnte ich mich sofort für das Bier-Webservice Projekt begeistern.

Ich habe bereits viel über JSON gehört und bin sehr interessiert was das eigentlich genau ist aber ich hatte noch keine Möglichkeit damit etwas zu tun. Damit ich den verpassten Unterrichtsstoff nachholen kann war dies die Perfekte Gelegenheit mich darüber zu Informieren und etwas Produktives zu entwickeln.

4 AUSGANGSLAGE

Der Chef Ihrer Stammbeiz möchte in seiner Beiz-App Wissenswertes und Kurioses über verschiedene Biere anbieten. Er hat bereits vom Webservice BreweryDb.com gehört. Dort sind verschiedenste Bierarten, Brauereien und Biere gespeichert und können über einen JSON-basierten Webservice abgefragt werden.

Implementieren Sie alle notwendigen Klassen und Methoden für die Klasse “BeerAdmin”. Nutzen Sie soweit möglich Datenstrukturen der java.util Bibliothek.

5 AUFGABE

Erstellen Sie die Klasse BeerAdmin mit allen beschriebenen Funktionen. Implementieren Sie in der main() Funktion einen geeigneten Testrahmen. Zunächst werden die Bierarten vom Webservice geladen. ein kleines Benutzermenü soll die oben genannten Funktionen komfortabel testen. Zusätzlich enthält das Benutzermenü eine Option zum Beenden.

Erwartete Ergebnisse

- Struktogramm für printBeerStyles(String search)
- Saubere Implementierung des Programms inklusive inline Dokumentation
- Dokumentation des Projektablaufs inklusive Verantwortlichkeiten
- Kurze Präsentation des Programms und der Struktogramme

6 INFORMIEREN

Damit das Projekt erfolgreich abgeschlossen werden kann habe ich eine Auflistung der Teilaufgaben in der korrekten Reihenfolge erstellt, nach der ich mich halten werde:

- JSON
    - Wie / Was ist JSON überhaupt?
    - Wie wird JSON implementiert?
- API
    - Wie kann man sich verbinden?
    - Wie sieht die Antwort einer Anfrage RAW aus?
- Programm
    - Welche Aufgaben müssen erfüllt werden
    - Passende Datenstruktur in Form eines Klassendiagramms entwickeln

In dieser Dokumentation werde ich nicht genauer über das Informieren von JSON eingehen, sondern direkt zu den Erkenntnissen aus den Antworten von der API kommen.

7 ERKENNTNISSE

Bei der RAW JSON Server Antwort konnte ich feststellen, dass die Klasse Beer dieselben Attribute wie Styles hat welche benötigt werden (String id & String name) für das Erstellen der Objekte. Durch diese Entdeckung habe ich mir eine zusätzliche Challenge geschaffen:

Die Applikation soll so kompakt wie möglich werden aber dennoch alle Funktionen erfüllen.

Zum Erstellen der Style Liste kann ich also die Beer Klasse “Missbrauchen”. Dadurch finde ich den Namen der Klasse “Beer” unpassend und ändere diesen zu “Item”.

Die IDs der Styles sind ohne Ausnahme Integers diesbezüglich werde ich keine HashMap verwenden, sondern eine TreeMap. Diese sortiert Keys nach ihrem Wert Automatisch und beim Ausgeben aller Styles sieht dies dann schöner aus zudem ist die Zugriffszeit kürzer wenn man einen Key sucht.

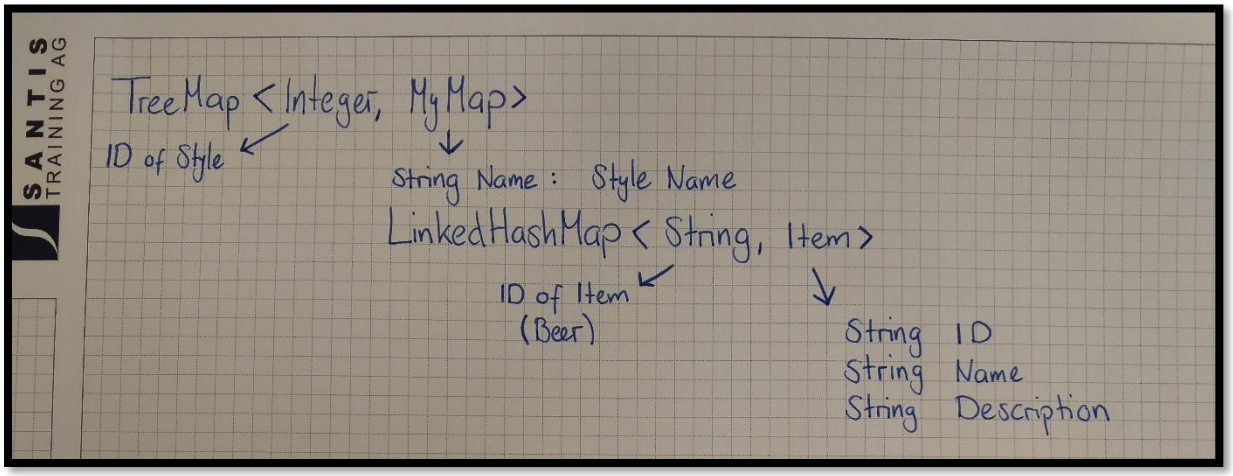
Damit auch bei den Beers eine Sortierung möglich ist, verwende ich dort eine LinkedHashMap, diese behält ihre Population Reihenfolge beim Iterieren. Dementsprechend kann ich die Liste der Beers vorab alphabetisch nach dem Namen Sortieren und dann die Map sortiert befüllen. Diese ist dann bei der Ausgabe immer noch Sortiert (nicht wie bei einer HashMap).

8 DATENSTRUKTUR

Bei der Datenstruktur, welche vom Arbeitsauftrag gewünscht wird sollten zwei HashMaps erstellt werden. Eine mit Styles <ID, Name> sowie eine für die Beers <ID, BeerObject (ITEM)>.

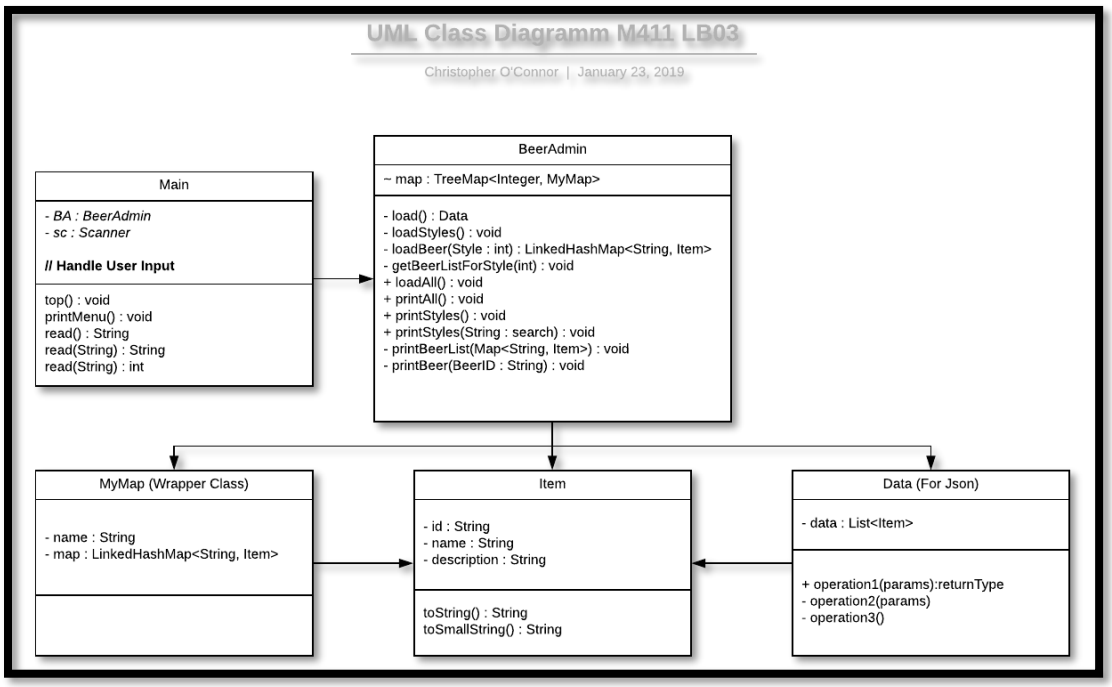
Um meine Challenge zu erfüllen kam mir die Idee: Wieso Nicht eine Map in einer Map? Dadurch wird die Komplexität zusätzlich erhöht und bei der Item Klasse kann ich das Attribut mit der StyleId weglassen. Denn wenn ein Beer in der Untergeordneten Map der Style Map gespeichert ist, weiss ich zu welchem Style das entsprechende Beer gehört.

Zur Veranschaulichung meiner Idee habe ich folgende Skizze kreiert:



Wie erkenntlich ist muss ich eine eigene Map Klasse erstellen, weil ich einen Ort brauche um den Namen des Styles zu Speichern. Es ist lediglich eine Wrapper Klasse mit einem String Name und der untergeordneten LinkedHashMap.

Um mir nun eine Übersicht zu verschaffen über die notwendigen Klassen habe ich ein Klassendiagramm erstellt:



9 REALISIERUNG

9.1 DISCLAIMER

Der in der Dokumentation ersichtlicher SourceCode wurde für die Leserlichkeit angepasst und kann unter Umständen nicht der Realität entsprechen bspw. System.out.println wurde zu Print gekürzt oder auch Formatierungen wie "\n", "\t" etc. sind entnommen.

9.2 JSON

Um die Objekte von JSON erstellen zu können müssen diese Klassen nachgebildet werden. Dazu habe ich eine Wrapper Klasse Data erstellt. Als einziges Attribut hat sie eine Liste mit Items. Weil mir aufgefallen ist, dass ich nur eine Klasse um Styles und Beers zu erstellen mit JSON muss ich lediglich die URL anpassen.

Von JSON lass ich mir die Objekte erstellen und in eine Liste abfüllen. Dies kann mit folgendem Code erreicht werden:

```
new Gson().fromJson(
            new InputStreamReader(new URL(url + APIKey + filter).openStream()), Data.class);
```

9.3 MAP POPULATION

9.3.1 TreeMap Styles

Mit der erstellten Liste von JSON kann ich nun die Maps füllen mit einer n Iteration:

```
for (Item i : data.getData()) {

  map.put(Integer.valueOf(i.getID()), new MyMap(i.getName()));

}
```

Weil ich innerhalb der TreeMap als Value ein eigenes Objekt brauche muss dies auch noch erstellt werden. Dieser Vorgang geschieht beim Start der Applikation bevor das Benutzermenü ausgegeben wird.

9.3.2 LinkedHashMap Beers

Die Beers werden, erst falls diese gebraucht werden vom Benutzer, jeweils geladen für den entsprechenden Style. Ausserdem wird in Betracht gezogen ob diese bereits existieren. Wenn dies der Fall ist müssen sie nämlich nicht noch einmal heruntergeladen, erstellt und überschrieben werden.

Die Liste wird durch JSON wie oben bereits beschrieben erstellt. Bei der API kann ein StyleId Filter in der URL angeben werden damit nur die entsprechenden Beers zurückgeliefert werden.

Vor dem „populaten“ der LinkedHashMap wird die Liste nach Namen alphabetisch sortiert:

```
data.getData().sort(Comparator.comparing(Item::getName));
```

Die erstellte LinkedHashMap wird dann dem entsprechenden korrektem Style der MyMap Klasse zugewiesen:

```
map.get(idStyle).setMap(loadBeer(idStyle));
```

Damit der Benutzer auch die Möglichkeit hat alle Beers mittels eines Knopfdrucks herunterzuladen ist dies natürlich auch implementiert. Dabei wird die TreeMap mit allen Styles n iteriert und für jeden Style die oben beschriebene Methode aufgerufen.

Dabei wird ausserdem ein Timer gestartet um anzuzeigen wie lange es ging alle zu erstellen, ca. eine Minute beim Testen auf dem Entwicklersystem. Durch den Timer kann man auch nachweisen, dass wenn die Beers eins Style bereits heruntergeladen wurden, diese nicht erneut ersetzt werden: Denn wenn der Timer massiv weniger Zeit braucht (~1 Sekunde) bei einem wiederholten Vorgang und alle Beers nach wie vor existieren wurde nichts neu heruntergeladen, erstellt oder sortiert.

```
long startTime = System.currentTimeMillis();

map.forEach((key, value) -> {

  if(value.getMap() == null) {

    Print(String.format(" – Loading: %d -> %s", key, value.getName()));

    value.setMap(loadBeer(key));

  } else {

    System.out.println(String.format(" -- Already  %d -> %s", key, value.getName()));

    }

});

Print(" -- Finished The Download\t-/> Elapsed Time : %1$,.2f sec..",
((double) (System.currentTimeMillis()-startTime) / 1000F));
```

9.4 AUSGABEN

Eine Map kann mittels eins Lambdas sehr einfach iteriert werden. Dabei werden dann jeweils die gewünschten werte ausgegeben.

```
map.forEach((key, value) -> {
  ...
});
```

Durch meine Datenstruktur muss kein Filter angewendet werden, wenn alle Beers eines gewünschten Styles ausgegeben werden sollen. Dies ist der entscheidende Vorteil von der Verschachtelung der Maps und bringt enorm viel, wenn grössere Datenmengen vorhanden sind.

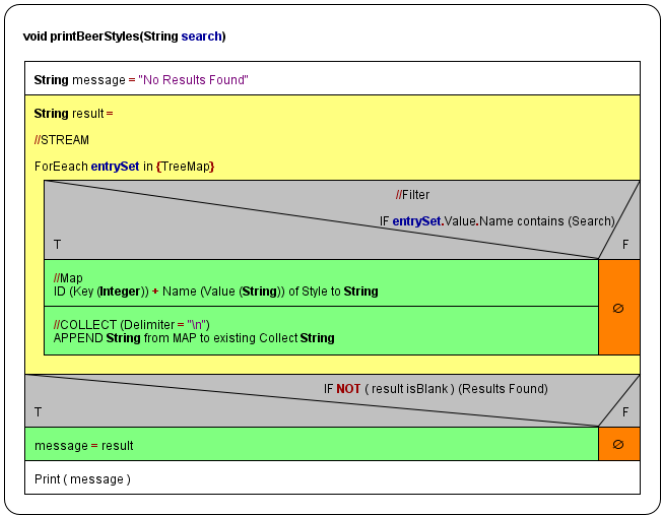
Möchte man aber alles Printen, Styles und die Entsprechenden Beers dann ist dies mit einer verschachtelten ForEach schleife möglich.

```
map.forEach((key, value) -> {

  Println(String.format("%d -> %s:", key, value.getName()));

  if(value.getMap() != null) {

    if (!(value.getMap().isEmpty())) {

      value.getMap().forEach((subKey, subValue) ->

          Println(subValue.toSmallString())

      );

    }

  } else Println(":> Empty...");

});
```

9.5 SEARCH

9.5.1 Style Name

Um Nach einem Style Namen zu suchen habe ich folgendes Struktogramm angefertigt:



Aus diesem sollte die Logik abzuleiten sein. Dazu steht hier noch der SourceCode zur Verfügung:

```
String message = "No results Found for " + search + " ... :(";


String result  = map.entrySet().stream()

          .filter(entry -> entry.getValue().getName().contains(search))

            .map(entry -> String.format("/> %d :: %s", entry.getKey(),
entry.getValue().getName()))

              .collect(Collectors.joining("\n"));



if (!result.isBlank()) message = "Your Search Matches :/> " + result;

Println(message);
```

Die komplette TreeMap mit den Styles wird gestreamt. Dies kann man mit einem ForEach vergleichen. Bei jedem Entry wird dabei der Name überprüft ob dieser den gesuchten String beinhaltet. Wenn Ja dann wird der Key und der Value von einer Map umgewandelt und vom Collector zum bereits bestehenden temporären String hinzugefügt. Danach hängt dieser ein „\n“ am Ende hinzu. Wenn alle Elemente verglichen wurden wird der Ergebnis String auf Blank geprüft. Ist dieser nämlich leer, dann wurden keine Übereinstimmungen gefunden.

9.5.2 Beer ID

Um nach einer Beer ID zu suchen muss durch alle Styles der TreeMap durch Iteriert werden und die entsprechende Beer Map nach der ID durchsucht werden. Dies ist der größte Nachteil meiner verschachtelten Maps.

9.6 Menu & User Input Handling

Um dem Benutzer die Benutzung dieser Applikation zu erleichtern gibt es ein Menü mit den verfügbaren Möglichkeiten als Übersicht. Mit einem Switch Case kann der Benutzer auswählen was er tun möchte (auch die erwünschte Beendigung des Programms ist umgesetzt). Bei einzelnen Funktionalitäten benötigt die Applikation ausschliesslich Integers, auch dies wird getestet mit Regex ("^[0-9]+") und somit gewährleistet.

Die ganze Logik dazu wurde im Main#top() implementiert.

Sehr viel Zeit wurde in das Styling der Konsolen Aus- und Eingabe investiert:

9.7 TESTUNG

Die Applikation wurde vom Entwickler auf dem Entwicklersystem auf alle möglichen Fehler überprüft. Dies Betrifft insbesondere, Falsche Benutzer eingaben wie etwa Strings, wenn ein Integer erwartet wird oder auch Allgemeine Falsche eingaben wie nicht aufgeführte Keys.

Des Weiteren wurden alle Methoden angepasst auch auf leere respektive auch null zu überprüfen und damit wie gewünscht umzugehen.

Da dies eine nur von mir erbrachten Arbeit ist, müssen die Verantwortlichkeiten aller beteiligten Personen nicht wie ausführlich verlangt ersichtlich spezifisch gemacht werden.