Merchant’s Guide to the Galaxy

Einführung

Willkommen zu Convo, dem intergalaktischen Umrechner für Nummern und Einheiten!

Wie gewünscht, wurde test-driven entwickelt. Dabei wurden zunächst die Tests geschrieben, die sich großteils direkt aus der Aufgabenbeschreibung ableiten ließen. Anschließend wurde dafür gesorgt, dass diese Tests erfolgreich durchlaufen. Später folgten diverse Refactorings.

Es gibt zwei Arten von Queries und insgesamt vier verschiedene Queries:

- Definitions-Queries:
    a. Definition von intergalaktischen Begriffen (`alienTerms`) für römische Zahlzeichen
    b. Definition von Wertes eines Guts in Form einer Anzahl Credits
        - Es können Wertentsprechungen für beliebige Anzahlen eines Guts angegeben werden. Intern wird die Anzahl Credits für 1 Einheit dieses Guts gespeichert.
- Frage-Queries:
    a. Nummer-Konvertierungs-Query. Eine durch Leerzeichen getrennten Sequenz von alienTerms wird als römische Zahl interpretiert. Diese wird anschließend in eine arabische Zahl umgerechnet.
    b. Abfrage zuvor definierter Werte von Gütern. Das Ergebnis wird in *Credits* angegeben.

Das Programm wurde so entwickelt, dass es komplett selbsterklärend sein sollte – sowohl hinsichtlich des Codes als auch während der Benutzung über den Chat. Im Chat werden per Eingabe von “usage” Informationen zur Bedienung ausgegeben. Diese sehen ungefähr wie folgt aus. Teile in runden Klammern `()` sind optional und können weggelassen werden. Bspw. können bei der Zuweisung von Credits zu einem Gut die alienTerms weggelassen werden. Dadurch werden die Credits für 1 Einheit dieses Guts festgelegt.
Ebenfalls weggelassen werden können Fragezeichen am Ende von Frage-Queries.

```
Define foreign terms for Roman numerals in the format
```

```
'[term] is [Roman numeral]'

Assign a number of credits to an amount of a good in the format
'([previously defined term1] [p. d. term2] [...]) [good] (is) [Arabic number
of] (credit(s))'

Convert a sequence of foreign numeral terms to an Arabic number by asking
'How much is [p. d. term1] [term2] [...] (?)'

Get the value in Credits of an amount of a good by asking
'How many Credits is [p. d. term1] [term2] ... [good] (?)'
```

Annahmen und Designentscheidungen

Ich hatte zunächst eine externe Library verwendet für die Konvertierung zwischen römischen und arabischen Zahlen, da in der Aufgabenstellung steht, dass man - wie bei echter Softwareentwicklung - Libraries nutzen darf. Gleichzeitig wird in der Aufgabe darum gebeten, nicht zu schummeln und eine eigene Lösung zu schreiben. Da ein Großteil der Aufgabenstellung Details zu römischen Zahlen und deren Regeln beinhaltet, ist mir später bewusst geworden, dass ich diese Funktionalität sicher selbst programmieren soll, was ich dann auch getan habe.

Da in der Aufgabenstellung die Rede von “common metals and dirt” ist, habe ich mich entschieden zu verallgemeinern, und so behandelt das Programm nun allgemein *goods* (Güter/Waren).

Queries mit Zuweisungen von Credits zu Gütern werden intern umgerechnet in die Anzahl Credits für 1 Einheit dieses Guts.

Ich habe 30 römische Zahlen definiert. Ich wäre auch mit nur 14 ausgekommen (I bis X; L, C, D, M), hätte dann aber immer die Basiszahlen 10 und 5 skalieren müssen (I → X → C → M; V → L → D). Bsp.: 90 = 9 x 10, also alle Zahlzeichen in 9 (I,X) um Faktor 10 hochskaliert: I x 10 = 10 = X; X x 10 = 100 = C. Somit ist IX x 10 = XC.
Stattdessen habe ich mich für die Definition 16 zusätzlicher Zahlen entschieden. So können alle möglichen Summanden einer römischen Zahl direkt nachgeschlagen werden.

Projektstruktur

Class RomanNumerals

Enthält Funktionalität für die Umrechnung zwischen arabischen und römischen Zahlen

Class Converter

- Enthält den Chatbot bzw. die Benutzerschnittstelle des Umrechners

- Einstiegspunkt ist die `main` function. Von dort werden im Normalfall Queries des Benutzers mit `Converter.submitQuery()` empfangen und verarbeitet, und anschließend wird ggf. eine Response zurückgegeben und ausgegeben.
- Falls der Query die Definition eines Begriffs für ein römisches Zahlzeichen oder eine Definition des Wertes (Credits) einer Ware war, so wird keine Response erzeugt.

Tests

- Aufteilung der Tests für RomanNumerals:
    - RomanNumeralsTest_Requirements testet die Regeln und Einschränkungen beim Format von und dem Rechnen mit römischen Zahlen, die in der Aufgabenstellung beschrieben sind.
    - RomanNumeralsTest_Methods testet die Methoden, die ich dafür geschrieben habe
        - Der umfangreiche Test `testConversionBackForth()` konvertiert alle ganzen Zahlen, die als römische Zahl ausgedrückt werden können (1 bis 3999) in eine römische Zahl und anschließend zurück, was wieder die ursprüngliche Zahl ergeben muss.

- ConverterTest enthält Tests für die 2 Frage-Queries.
- Für die zwei Definitions-Queries (Definition von intergalaktischen Begriffen (`alientTerms`) sowie von Credits pro Ware) gibt es keinen direkten Test. Die Korrektheit der Definitionen ist jedoch implizit durch die Tests der beiden Frage-Queries sichergestellt, deren erster Schritt die Definitions-Queries sind.

Verwendete Tools und wie man das Projekt baut

Das Projekt wurde mit folgenden Tools entwickelt:

- IDE: IntelliJ IDEA 2023.1 (Community Edition)
- Buildsystem: Gradle Version 8.0
- Java JDK: Corretto 19.0.2

Dependencies (s. Datei build.gradle):

- Guava für die BiMap / HashBiMap `RomanNumerals.Numbers`
- JUnit 5 / JUnit Jupiter für die Unit Tests

Build-Anleitung und Ausführung

Per

```
.\gradlew jar
```

kann mit dem Gradle wrapper eine JAR erzeugt werden.

Ausgeführt wird diese wie gewohnt per
`java -jar .\GalaxyGuide.jar`

Schwachstellen und Verbesserungspotenzial

- Der `Converter` rechnet immer entlang des Pfades
  Arabische Zahlen ↔ *römische Zahlen* ↔ *Alien-Labels für römische Zahlen*
  um
- Das ist umständlich / aufwendig. Wahrscheinlich lohnt es sich von einem geeigneten bestehenden Typ, z.B. `NumberFormat`, zu erben, um direkt mit römischen Zahlen rechnen zu können.
- Dadurch sollten sich die Umrechnungen *Arabische Zahlen* ↔ *römische Zahlen* vereinfachen und die Wartbarkeit und Leserlichkeit des Codes erhöhen.
- Für die Verarbeitung der eingegebenen Queries habe ich versucht Regexes und Matching groups einzusetzen, hatte jedoch Schwierigkeiten mit den gematchten Groups und habe sie dann wieder entfernt. Die Eingabe-Strings werden nun ohne Regexes sequenziell durchgearbeitet und die interessanten Komponenten extrahiert.
- Regexes hätten den Vorteil, dass sie sehr kompakt sind und zwei Probleme auf einmal lösen: Sie würden sicherstellen, dass Query-Strings dem gewünschten Format entsprechen, und m.H.v. Matching Groups sollte es möglich sein, gleich alle Komponenten zu erhalten.

Weitere Dinge, die ich einbauen bzw. ändern würde:
- Einen Query, der eine Liste der definierten Begriffe und der entsprechenden römischen Zahlzeichen ausgibt
- Einen Query, der alle Güter und ihren Wert in Credits ausgibt
- Verbesserung der Lesbarkeit und Wartbarkeit durch Extraktion von Methoden aus längeren Methoden wie `generateNumberConversionQueryResponse()`, `generateCreditsPerGoodsQueryResponse()`