252-0027-00: Einführung in die Programmierung

Übungsblatt 10 (Bonus)

Abgabe: 05 Dezember 2023, 19:00

Checken Sie mit Eclipse wie bisher die neue Übungs-Vorlage aus. Importieren Sie das Eclipse-Projekt für die Bonusaufgabe.

Aufgabe 1: Datenbanken (Bonus!)

Achtung: Diese Aufgabe gibt Bonuspunkte (siehe "Leistungskontrolle" im `www.vvz.ethz.ch`). Die Aufgabe muss eigenhändig und alleine gelöst werden. Die Abgabe erfolgt wie gewohnt per Push in Ihr Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin. Auch wenn Sie vor der Deadline committen, aber nach der Deadline pushen, gilt dies als eine zu späte Abgabe. Bitte lesen Sie zusätzlich die allgemeinen Regeln.

In dieser Aufgabe implementieren Sie für eine Datenbank von Personengesundheitsdaten das Deklassifizieren von Einträgen (Task a) und das Verlinken von Einträgen (Task b). Alle Unteraufgaben können separat gelöst werden.
Die Datenbank selber ist bereits mit der Klasse `Database` implementiert. Die Datenbank hält eine Liste von Einträgen, welche durch die Klasse `Item` repräsentiert werden. Die folgenden 4 Paragraphen erklären alle in der Vorlage gegebenen Klassen im Detail.

`Item` Die Klasse `Item` repräsentiert einen Datenbankeintrag mit 4 Attributen: eine ID (int), ein Alter (int), einen Gesundheitswert (int), und ein Sicherheitslevel, welches durch die Klasse `Level` repräsentiert wird. Alter und Gesundheitswert sind immer ≥ 0. Die Methoden `Item.getID()`, `Item.getAge()`, `Item.getHealth()`, `Item.getLevel()` geben jeweils die ID, das Alter, den Gesundheitswert, und das Sicherheitslevel eines Eintrags zurück. Die Methode `Item.setHealth(int newHealth)` setzt den Gesundheitswert auf `newHealth`. Die anderen Attribute können nicht geändert werden.

`Level` Die Klasse `Level` repräsentiert ein Sicherheitslevel. Ein Sicherheitslevel wird über eine Liste von Integern definiert, welches in einem Attribut der Klasse `Level` gespeichert wird und von der Methode `Level.getPoints()` zurückgegeben wird. Ein Level `A` ist *verwandt* mit einem Level `B`, falls die Summe der Werte in `A.getPoints()` gleich der Summe der Werte in `B.getPoints()` ist. Zum Beispiel ist das Level $[1, 2, 3, 4]$ verwandt mit den Levels $[10]$ und $[4, 6]$ (die Summe ist überall 10), aber nicht mit dem Level $[4, 5]$.

`ItemFactory` Die Klasse `ItemFactory` wird verwendet, um Datenbankeinträge zu erstellen. Die Methode `ItemFactory.createItem(Level level, int id, int age, int health)` gibt ein Exemplar der Klasse `Item` zurück, deren Attribute mit den Argumenten initialisiert wurden.

`Database` Die Klasse `Database` repräsentiert eine Datenbank und hat folgende vorgegebene Methoden:

- `Database.getItemFactory()` gibt ein Exemplar von `ItemFactory` zurück. Die `ItemFactory` `I` ist assoziiert mit der Datenbank `D`, falls `I` von `D.getItemFactory()` zurückgegeben wird.
- `Database.add(Item item)` fügt der Datenbank den Eintrag `item` hinzu.
- `Database.getItems()` gibt die Liste aller Einträge zurück, welcher der Datenbank hinzugefügt wurden. Sie dürfen annehmen, dass für eine Datenbank `D` alle Einträge in `D.getItems()` eine einzigartige ID haben, über `D.add` hinzugefügt wurden, über `D.getItemFactory()` erstellt wurden, und keiner anderen Datenbank hinzugefügt werden. Ein hinzugefügter Eintrag wird nie wieder entfernt.

a) Implementieren Sie die Methode `ItemFactory.createDeclass(Level level, int id, int targetId)`, die einen *Deklassifikationseintrag* zurückgibt. Ein Deklassifikationseintrag ist selber ein Eintrag, also ein Exemplar der Klasse `Item`. Ein Deklassifikationseintrag hat damit auch eine ID, ein Sicherheitslevel, ein Alter, und einen Gesundheitswert, welche von den entsprechenden getter-Methoden zurückgegeben werden. ID und Sicherheitslevel eines Deklassifikationseintrags sind jeweils das `id` und `level` Argument des `createDeclass` Aufrufs, mit welchem der Eintrag erstellt wurde. Das Alter und der Gesundheitswert eines Deklassifikationseintrags sind jeweils das Alter und der Gesundheitswert des *Zieleintrags* vom Deklassifikationseintrag. Der Zieleintrag von einem Deklassifikationseintrag `D` ist der Eintrag `E`, so dass

   - `E.getID()` gleich dem Parameter `targetId` ist, mit welchem `D` erstellt wurde; *und*
   - `E` aus der Datenbank ist, mit welcher die `ItemFactory` assoziiert ist, mit welcher `D` erstellt wurde.

   Falls es keinen Zieleintrag gibt, wird eine `IllegalArgumentException` von der Methode `createDeclass` geworfen. Beachten Sie, dass Zieleinträge selber Deklassifikationseinträge sein können. Ein Aufruf der Methode `Item.setHealth(h)` auf einem Deklassifikationseintrag hat keinen Effekt; dies wird nicht in den Tests überprüft.

   Ein Deklassifikationseintrag `R` *erreicht* einen Eintrag `A`, falls entweder `A` der Zieleintrag von `R` ist oder falls der Zieleintrag von `R` ein Deklassifikationseintrag ist, welcher `A` erreicht. Die Methode `createDeclass` wirft eine `IllegalArgumentException`, falls der zurückzugebene Deklassifikationseintrag `R` einen Eintrag erreicht, dessen Level verwandt ist mit dem Level von `R`. Zur Erinnerung: Der Paragraph über die Klasse `Level` erklärt, wann zwei Level verwandt sind.

b) Implementieren Sie die Methode `Database.createLink(List<Integer> ids)`. Der Methodenaufruf `D.createLink(ids)` *verlinkt* alle Einträge der Datenbank `D` miteinander, welche eine ID haben, die im Argument `ids` enthalten ist. Wenn `E.setHealth(h)` auf einem Eintrag `E`

aufgerufen wird, dann wird der Gesundheitswert aller Einträge, welche mit `E` verlinkt sind, auf das Argument `h` gesetzt. Einträge können beliebig oft verlinkt werden und verlinken ist transitiv, das heisst, wenn ein Eintrag `A` mit einem Eintrag `B` verlinkt ist und `B` mit einem Eintrag `C` verlinkt ist, dann ist `A` auch mit `C` verlinkt. Verlinken ist auch immer symmetrisch, das heisst, wenn `A` mit `B` verlinkt ist, dann ist auch `B` mit `A` verlinkt. Zusätzlich ist verlinken reflexiv, das heisst, ein Eintrag ist immer mit sich selber verlinkt.

Der Aufruf `D.createLink(ids)` soll eine `IllegalArgumentException` werfen, falls es eine ID im Argument `ids` gibt, für welche es keinen Eintrag mit der gleichen ID in der Datenbank `D` gibt.

Wir geben zwei Testdateien zur Verfügung. “DatabaseTest.java” enthält Tests, welche wir an einer Prüfung geben würden. “GradingDatabaseTest.java” enthält Tests, welche wir zum Korrigieren einer Prüfung verwenden würden. Testen Sie Ihre Lösung zuerst ausgiebig mit “DatabaseTest.java” (am besten fügen Sie selber neue Tests hinzu) und dann können Sie “GradingDatabaseTest.java” verwenden, um zu sehen wie Ihre Lösung an einer Prüfung abgeschnitten hätte.