Proseminar Programmiermethodik

Universität Innsbruck - Institut für Informatik

A. Blaas, M. Mayerl, B. Murauer, M. Tschuggnall

Blatt 7: Modern Java

Diskussionsteil

a) ★ Sehen Sie sich folgende Methode an:

```
1   public static void readAndPrintFile() throws IOException {
2       FileReader reader = new FileReader("MyFile.txt");
3       BufferedReader bufReader = new BufferedReader(reader);
4
5       String currentLine;
6       while ((currentLine = bufReader.readLine()) != null) {
7           System.out.println(currentLine);
8       }
9   }
```

In diese Methode hat sich ein Fehler eingeschlichen. Wie können Sie diesen beheben? Denken Sie dabei auch an die Möglichkeit, dass während der Ausführung der Methode Exceptions geworfen werden können.

b) ★★ Sehen Sie sich folgende Methode an:

```
1   public static double calculateAvgWeight(List<Cat> cats, int minimumAge) {
2       int catsCount = 0;
3       double weightSum = 0.0;
4
5       for (Cat c : cats) {
6           if (c.getAge() >= minimumAge) {
7               catsCount++;
8               weightSum += c.getWeight();
9           }
10      }
11
12      return (catsCount > 0) ? weightSum / catsCount : 0.0;
13  }
```

Die Methode berechnet das durchschnittliche Gewicht für alle Katzen in einer Liste, die ein gewisses Mindestalter haben. In ihrer jetzigen Form verwendet diese Methode imperativen Code, um ihr Ergebnis zu berechnen. Schreiben Sie die Methode so um, dass sie Java 8 Streams und einen deklarativen Programmierstil verwendet:

```
1       public static double calculateAvgWeight(List<Cat> cats, int minimumAge) {
2           return cats.stream()./*your code here*/
3       }
```

c) ★★ Nehmen Sie an Sie entwickeln eine Bibliothek, die von vielen Projekten genutzt werden kann. In Ihrer Bibliothek haben Sie unter anderem folgendes Interface:

```
1       public interface Summable {
2           double getSum();
3       }
```

Ihr Chef gibt Ihnen jetzt folgende Aufgabe: Erweitern Sie das Interface um eine `getSumRounded()`-Methode, welche die Summe gerundet auf zwei Kommastellen zurückgibt. Damit stehen Sie vor einem Problem. `Summable` wird von vielen Klassen in Projekten, auf die Sie gar keinen Zugriff haben, implementiert. Wenn Sie die Methode einfach zum Interface hinzufügen, also

```
1       public interface Summable {
2           double getSum();
3           double getSumRounded();
4       }
```

dann würden diese Projekte nicht mehr kompilieren, da ihre Klassen, die `Summable` implementieren, diese Methode nicht enthalten.

Wie können Sie dieses Problem mithilfe der mit Java 8 eingeführten neuen Sprachfeatures lösen?

d) ★ Diskutieren Sie mit Ihren Kolleginnen und Kollegen folgende Fragen:

- Was ist ein Functional Interface? Welche vordefinierten Functional Interfaces stellt Java im Package `java.util.function` zur Verfügung?
- Was sind Method References?
- Warum lässt sich der folgende Code kompilieren:

```
1       public static void doSomething() {
2           int i = 0;
3           Supplier<Integer> getInt = () -> i;
4
5           System.out.println("Value is " + getInt.get());
6       }
```

Dieser hier aber nicht:

```
1       public static void doSomething() {
2           int i = 0;
3           Supplier<Integer> getInt = () -> i;
4           i = 1;
5
6           System.out.println("Value is " + getInt.get());
7       }
```

Übungsteil (selbstständig zu lösen)

Aufgabe 1 (Refactoring mit Java 8) [4 Punkte]

In dieser Aufgabe werden Sie bestehendes Projekt modernisieren, indem Sie einige der in dem Projekt enthaltenen Methoden so umschreiben, dass sie mit Java 8 neu eingeführte Features (Java 8 Streams, Lambda Expressions, ...) verwenden. Importieren Sie dafür zuerst in Eclipse das Projekt `CitizenStatistics.zip` (OLAT). Lesen Sie sich den Code des Projekts genau durch und versuchen Sie zu verstehen, was das Programm macht und machen soll.
Sehen Sie sich vor allem die Datei `Statictics.java` an. Darin sind einige Methoden enthalten, die statistische Informationen über Listen von Bürgern berechnen. Ebenfalls interessant ist die Datei `StatisticsTest.java`. Diese enthält Unit Tests für die `Statistics`-Klasse und wird Ihnen bei Ihrer Arbeit behilflich sein – mit den Tests können Sie zu jeder Zeit während Ihrer Arbeit überprüfen, ob Ihre Änderungen korrekt waren. Die Tests sollten, nachdem Sie mit dieser Aufgabe fertig sind, alle erfolgreich ausgeführt werden.
Schreiben Sie nun folgende Methoden in der Klasse `Statistics` so um, dass sie statt einem imperativen Programmierstil einen deklarativen Stil unter Verwendung von Java 8 Streams nutzen:

a) *0.5 Punkte* `getAverageAge(List<Citizen> citizens)`

b) *0.5 Punkte* `getAverageYearlyIncome(List<Citizen> citizens)`

c) *0.75 Punkte* `getCitizensPerOccupation(List<Citizen> citizens)`

d) *0.75 Punkte* `getAvgIncomePerOccupation(List<Citizen> citizens)`

e) *0.75 Punkte* `getAgeHistogram(List<Citizen> citizens)`

f) *0.75 Punkte* `getTopEarners(List<Citizen> citizens, int n)`

Ändern Sie im Zuge Ihrer Umbauten nichts an den Signaturen der Methoden und ebenfalls nichts an den zur Verfügung gestellten Unit Tests.

Aufgabe 2 (CSV-Reader) [3 Punkte]

In dieser Aufgabe werden Sie das Projekt aus Aufgabe 1 um die Möglichkeit, Citizens aus einer CSV-Datei zu laden, erweitern. Sehen Sie sich dazu die Datei `citizens.csv` an, die schon im Projekt enthalten ist.

Hinweis ⚠

Aufgabe 1 und Aufgabe 2 verwenden zwar das gleiche Projekt, sind aber ansonsten voneinander unabhängig. Sie können Aufgabe 2 also auch bearbeiten, falls Sie Aufgabe 1 ausgelassen haben.

a) *0.5 Punkte* Erstellen Sie eine Klasse `CsvCitizenRepository`, welche `CitizenRepositoy` implementiert. Erstellen Sie in der Klasse eine Variable vom Typ `java.nio.file.Path` – für die CSV-Datei, die gelesen werden soll – und einen Konstruktor, der diese Variable entsprechend setzt.

b) *1.5 Punkte* Implementieren Sie nun die Methode `loadCitizens()`. Verwenden Sie dabei einen `BufferedReader`, um die CSV-Datei zeilenweise einzulesen, und einen `Scanner`, um die einzelnen Felder einer Zeile zu parsen. Achten Sie dabei genau darauf, dass Ressourcen wieder korrekt freigegeben werden.

Achten Sie weiters auf eine sinnvolle Fehlerbehandlung. Wenn die angegebene Datei etwa nicht existiert oder nicht dem geforderten Format entspricht, soll die Methode ihre Ausführung abbrechen und eine sinnvolle Fehlermeldung auf der Konsole ausgeben.

c) *1 Punkt* Erweitern Sie zu guter Letzt noch die `main`-Methode des Projekts. Fragen Sie den Benutzer des Programmes zu Beginn, ob er Bürger aus einer CSV-Datei einlesen will. Bejaht er dies, fragen Sie ihn nach dem Pfad zur Datei und verwenden Sie Ihr gerade erstelltes `CsvCitizenRepository`, um sie einzulesen. Verneint er, sollen weiterhin zufällige Bürger generiert werden.

Zum Einlesen der Benutzereingaben auf der Konsole können Sie einen `Scanner` benutzen.

Hinweis ⚠

Implementierne Sie die Logik zum Lesen der CSV-Datei selbst. Die Verwendung externer Bibliotheken ist nicht gestattet.

Aufgabe 3 (Lazy Initialization) [3 Punkte]

In dieser Aufgabe werden Sie ein wichtiges und praktisches Konzept der funktionalen Programmierung in Java umsetzen: Lazy Initialization. Bei Lazy Initialization geht es darum, ein Objekt erst dann zu erstellen bzw. eine Berechnung erst dann durchzuführen, wenn das Objekt bzw. der Ergebnis der Berechnung tatsächlich gebraucht wird. Dies kann in vielen Fällen Zeit sparen (nämlich dann, wenn das Objekt/das Ergebnis der Berechnung im Endeffekt gar nie benötigt werden).

a) *1.5 Punkte* Erstellen Sie ein neues Java-Projekt. Erstellen Sie dann in diesem Projekt eine generische Klasse `Lazy<T>`, welche einen lazy initialisierten Wert vom Typ `T` darstellt. Die Klasse enthält folgende Felder:

- `value` (Typ `T`): Der fertig initialisierte Wert.
- `initialized` (Typ `boolean`): Flag, das angibt, ob der Wert bereits initialisiert wurde.
- `supplier` (Type `java.util.function.Supplier<T>`): Der Supplier, der den Wert initialisiert, sobald er benötigt wird.

Erstellen Sie einen Konstruktor, der das Feld `supplier` setzt. Erstellen Sie weiters eine Methode `get()`. Diese Methode prüft zuerst, ob der Wert bereits initialisiert wurde (mittels `initialized`). Ist er es nicht, initialisiert sie ihn mithilfe des Suppliers und schreibt das Ergebnis in `value`. Anschließend gibt sie `value` zurück.

Da ein `Lazy<T>` in unserem Fall *immutable* sein soll, erstellen Sie bitte *keinen* Setter.

b) *0.5 Punkte* Kopieren Sie die Datei `LazyMain.java` (OLAT) in Ihr Projekt und führen Sie das Projekt mit der darin enthaltenen `main`-Methode aus. Was wird ausgegeben? Erklären Sie, warum es zu den unterschiedlichen Ausgaben kommt.

c) *1 Punkt* Überlegen Sie sich (oder recherchieren Sie), für welche Anwendungsfälle die Verwendung von Lazy Initialization sinnvoll sein kann. Suchen Sie sich einen Anwendungsfall aus, den Sie besonders wichtig finden, und beschreiben Sie diesen in einigen Sätzen. Implementieren Sie außerdem eine minimale Version dieses Anwendungsfalles in Java (im selben Projekt, in dem Sie `Lazy<T>` implementiert haben).

Wichtig: Laden Sie bitte Ihre Lösung (.txt, .java oder .pdf) in OLAT hoch und geben Sie mittels der Ankreuzliste auch unbedingt an, welche Aufgaben Sie gelöst haben. Die Deadline dafür läuft am Vortag des Proseminars um 23:59 (Mitternacht) ab.