Proseminar Programmiermethodik

Universität Innsbruck - Institut für Informatik

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

7.6.2018

2. Midterm Test - Programmieraufgaben (Gruppe 1)

Programmierteil

Verwenden Sie für diese Aufgaben Eclipse. Sie finden die Installation unter `/usr/site/eclipse`.

In diesem Teil des Tests sollen Sie selbst Java programmieren. Sehen Sie sich dazu die bereitgestellten Aufgaben an.

Hinweis

Sie können das Eclipse-Projekt importieren, indem Sie auf File > Import > General > Existing Projects Into Workspace klicken und für das Feld Select archive file die Datei `midterm2.zip` (OLAT) auswählen.

Hinweis

Exportieren Sie Ihre Lösung am Ende des Tests ebenfalls mit Eclipse. Wählen Sie dazu File > Export > General > Archive File und geben Sie Ihrem Archiv **midterm2_<Ihre C-Kennung>.zip** als Namen. Laden Sie diese Datei anschließend als Abgabe auf OLAT hoch.

Aufgabe 1 (MetaInfo-Collection) [6 Punkte]

Erstellen Sie eine generische Klasse `MetaInfoCollection`, die eine normale Collection des Typs `T` verwaltet und zusätzlich für jeden Collectioneintrag eine Metainfo des Typs `M` speichert. Beispielsweise könnten Sie eine Liste von Zahlen verwalten (`T = Integer`) und zu jeder Zahl zusätzlich einen String speichern, der Metainformationen dazu enthält (`M = String`).

Hinweis

Sie können davon ausgehen, dass sowohl alle Elemente in der Collection als auch alle MetaInfos unterschiedlich sind.

a) *2 Punkte* Erstellen Sie die Klasse `MetaInfoCollection` und übergeben Sie die Collection im Konstruktor. Beachten Sie, dass jede beliebige Collection wie z.B. `List` oder `Set` übergeben werden kann.

Hinweis ⚠

Sehen Sie sich die Klasse `MetaInfoCollectionTest` (OLAT) für ein Beispiel zur Verwendung an.

b) *1 Punkt* Erstellen Sie die Methode `add(elem, metaInfo)`, die das übergebene Element `elem` der Collection hinzufügt und die dazugehörige `metaInfo` speichert. Verwenden Sie zum Speichern der Metainfo eine `Map`.

c) *1 Punkt* Erstellen Sie die Methode `getMetaInfo(elem)`, welche die MetaInfo zum Element `elem` zurückgibt

d) *2 Punkte* Erstellen Sie die Methode `getElement(metaInfo)`, die das Element aus der Collection zurückgibt, für das die übergebene `metaInfo` gespeichert ist.

Testen Sie Ihre Implementierung mit der `main`-Methode der Klasse `MetaInfoCollectionTest` (OLAT). Sie dürfen diese Klasse dabei **nicht** verändern. Die Ausgabe sollte bei korrekter Implementierung 6x die Zeile `check: ok` ausgeben.

Aufgabe 2 (Arithmetische Reihe) [8 Punkte]

In dieser Aufgabe sollen Sie eine arithmetische Reihe implementieren und mit TestNG testen. Eine arithmetische Reihe startet bei einem Anfangswert a_1 und berechnet den nächsten Wert a_2, indem zum vorigen Wert ein Schritt d addiert wird, d.h. $a_2 = a_1 + d$, oder allgemeiner $a_i = a_{i-1} + d$. Beispielsweise würde eine Reihe mit Startwert 10 und Schritt 3 wie folgt aussehen:

a_1	a_2	a_3	a_4	...
10	13	16	19	...

Erstellen Sie die Klasse `ArithmeticSeries` und implementieren Sie das `NumberGenerator` (OLAT)-Interface, um innerhalb der Reihe vor- und zurückzunavigieren:

a) *0.5 Punkte* Initialisieren Sie die Klasse mit einem Startwert (`int start`) und einem Schritt (`int step`).

b) *1 Punkt* `nextNumber()` gibt den nächsten Wert der Reihe zurück:
- beim ersten Aufruf der Methode wird der Startwert zurückgegeben
- ansonsten wird der Wert a_{i+1} zurückgegeben, wenn sich die Reihe aktuell bei a_i befindet

c) *1 Punkt* `previousNumber()` gibt den vorherigen Wert der Reihe zurück:
- befindet sich die Reihe aktuell beim Wert a_i, so wird der Wert a_{i-1} zurückgegeben
- befindet sich die Reihe aktuell beim Startwert a_1, so soll eine Exception geworfen werden. Erstellen Sie dafür eine `NoPreviousElementException` und passen Sie das `NumberGenerator`-Interface entsprechend an.

Erstellen Sie nun eine NGTest-Klasse `ArithmeticSeriesTest` mit folgenden Tests:

a) *0.5 Punkte* In allen folgenden Tests soll eine arithmetische Reihe mit Startwert 5 und Schritt 3 verwendet werden. Verwalten Sie diese Werte als unveränderliche Konstanten, sodass sie ggf. an einer Stelle geändert werden können.

b) *2 Punkte* Der Test `sumTest()` überprüft, ob die Summe der ersten 100 Zahlen der Reihe dem Wert $100 \cdot (a_1 + a_{100})/2$ entspricht.

c) *2 Punkte* Der Test `backAndForthTest()` überrpüft, ob das Vor- und Zurücknavigieren in der Reihe korrekt ist:

- Gehen Sie zum 10. Wert der Reihe.
- Gehen Sie 100 Werte vor und anschließend wieder 100 Werte zurück und überprüfen Sie, ob der Wert derselbe ist.
- Fangen Sie Exceptions ab und lassen Sie im Falle einer Exception den Test mittels `org.testng.Assert.fail()` fehlschlagen.

d) *1 Punkt* Erstellen Sie einen negativen Test `exceptionTest`, der eine `NoPreviousElementException` erwartet, wenn direkt nach dem Anlegen der `ArithmeticSeries` die Methode `previousNumber()` aufgerufen wird.