Objektorientierte Softwareentwicklung

- Softwaretechnik -

SS 2010

Jähnichen
Mertgen

Ausgabe: Mittwoch, 12.05.10
Abgabe: Freitag, 28.05.10

2. Übungsblatt: Weiterentwicklung von LOS

Als weitere Ausbaustufe von LOS wollen wir zunächst `super`-Aufrufe ermöglichen. Weiterhin wollen wir eine einfache Form von Aspektorientierung integrieren. Dafür führen wir ein neues Modul *Aspect* ein. Die LOS-Erweiterung soll auf eurer Lösung zum 1. Aufgabenblatt basieren - dazu solltet ihr natürlich als erstes die durch Test und Feedback aufgezeigten Defizite eurer Version beheben.

`super`-Aufrufe

Innerhalb von Methoden sollen nun gezielt Aufrufe von Methoden der Oberklasse erlaubt sein. Für solche Aufrufe soll statt dem aufrufenden Objekt das Schlüsselwort `super` vorangestellt werden. Beispiel:

```
1 function A:foo(x) {
2   self:bar()
3   super:foo(x)
4 }
```

Da `super` nur ein Schlüsselwort ist, müssen wir dafür sorgen, dass unsere LOS-Umgebung an seiner Stelle das aufrufende Objekt übergibt und dazu die richtige Super-Methode findet.

Eine Möglichkeit zur Realisierung ist, für jeden Methodenaufruf eine Wrapper-Methode zu bilden. Vor Ausführung der originalen Methode wird darin das aufrufende Objekt und die Klasse, aus welcher die aufgerufene Methode stammt, temporär gespeichert. Wird während der Methodenausführung `super` verwendet, können wir auf diese Informationen zurückgreifen. Dies muss natürlich auch für verschachtelte (`super`-)Aufrufe funktionieren. Die Implementierung mittels Wrapper-Methoden ist natürlich nicht sonderlich effizient, was an dieser Stelle aber nebensächlich ist.

Deklaration von Aspekten

In ähnlicher Weise zur Deklaration von Klassen wollen wir die Deklaration von Aspekten umsetzen.

Ein Aspekt wird mit folgender Syntax erzeugt:

```
1 Aspect{'MyAspect';
2   adapts = {MyClassA, MyClassB},
3   attributes = {somevar = Number},
4   before = {aspectmethod = 'basemethod', foo = 'bar'},
5   after = {afterset = 'set%w*'}
6 }
```

1. `Aspect` ist ein Schlüsselwort von LOS, analog zu `Class`. *MyAspect* ist der Name des Aspekts. Die erzeugte Struktur soll anschließend unter diesem Namen global verfügbar sein. Im Gegensatz zu Klassen können unsere Aspekte nicht von anderen Aspekten erben.

2. Unter dem Schlüssel `adapts` wird eine Liste von (bereits existierenden) Klassen angegeben, welche durch diesen Aspekt adaptiert werden. Wir nennen sie Basisklassen.

3. Unter dem Schlüssel `attributes` wird eine Liste von Attributdeklarationen angegeben. Die Basisklassen sollen sich so verhalten, als hätten sie selbst das jeweilige Attribut deklariert. Hat die Basisklasse bereits ein Attribut gleichen Namens, so müssen die deklarierten Typen gleich sein.

4. Unter den Schlüsseln `before` und `after` kann je eine Liste von Methodenbindungen angegeben werden. Der Schlüssel-Teil bezeichnet den Namen einer Methode dieses Aspekts. Die hier genannten Methoden sollten vom Programmierer natürlich später für den Aspekt implementiert werden. Der Wert-Teil definiert ein Namensmuster von Methoden der Basisklasse(n). Ein Muster ist jeder gültige String. Es wird **nicht** verlangt, dass für ein genanntes Basismethoden-Muster auch tatsächlich eine Methode in den Basisklassen existiert.

5. Sollte die Aspektdeklaration Elemente enhalten, welche nicht der Spezifikation entsprechen, so soll ein Fehler ausgegeben werden. Wird eine Basisklasse oder ein Attributstyp mit einer Variablen angegeben, deren Wert undefiniert bzw. *nil* ist, so darf hier ein Fehler ("nicht-definierter Typ") ausgegeben werden (dies gibt Bonuspunkte), andernfalls ist eine solche Anweisung einfach still zu ignorieren.

Methoden der Aspekte werden nachfolgend wie bei Klassen deklariert:

```
function MyAspect:aspectmethod(arg)
  print(arg)
  return true
end
```

Effekte von Aspekten

Der Effekt von Aspekt-Attributen und Methoden ist einfach: adaptierte Basisklassen sollen sich so verhalten, als hätten sie selbst das Attribut deklariert bzw. die Methode definiert. Die `super`-Hierarchie der Basisklassen wird durch Aspekte nicht beeinflusst. Die Reihenfolge für die Suche nach Features, welche nicht im Objekt gefunden werden, ändert sich wie folgt:

1. in den Aspekten der Klasse, in der Reihenfolge ihrer Deklaration (last aspect wins)
2. in der Klasse des Objekts
3. in den Aspekte der Oberklasse (usw.)

Die Methoden eines Aspekts sollen also auch in den adaptierten Klassen vorhanden sein. Sollte eine Aspekt-Methode den gleichen Namen haben, wie eine Methode der adaptierten Klasse, so wird die Klassen-Methode durch die Aspekt-Methode überschrieben (ebenfalls *last aspect wins*). Die Original-Methode ist nicht mehr erreichbar. `Super`-Aufrufe beziehen sich nachwievor auf die übergeordnete Klasse (inkl. ihrer Aspekte).

Für `before`- und `after`-Methodenbindungen gilt: Sollte eine Methode einer Basisklasse ausgeführt werden, deren Name einem Muster entspricht, welches in der `before` bzw. `after`-Liste aufgeführt ist, so soll die korrespondierende Aspektmethode davor bzw. danach ausgeführt werden. Zur Beschreibung von Namensmustern verwenden wir die Standard Lua-Syntax (*siehe "Lua Manual 5.4.1 - Patterns"*), zur Auswertung die Methode `string.match`. Führt der Name einer Basismethode zu einem vollständingen Match (d.h. `name == string.match(name, pattern)`), so soll die Aspektmethode ausgeführt werden. Die Aspektmethoden bekommen dabei die selben Argumente, die auch der ursprünglichen Methode beim Aufruf mitgegeben wurden. Der Einfachheit halber gilt die Bindung nur für Methoden der angegebenen Basisklasse, nicht für überschriebene Varianten der Methoden in Subklassen. Bei der Suche nach `before` bzw. `after`-Methoden wird also ausschliesslich innerhalb des Aspekts gesucht (im Gegensatz zum dynamischen Binden bei normalen Methodenaufrufen). Sollte eine solche Methode im Ausführungsfall nicht gefunden werden, so wird ein Fehler ausgegeben - analog zu fehlenden Features an einer Klasse.

Die Ausführung einer Aspektmethode kann die Ausführung von weiteren Aspekt-Methoden auslösen. Vorsicht: manche Bindungen können zu einer Rekursion von Aufrufen führen (z.B. `after = {foo = 'foo'}`). Wer möchte, darf dies verbieten und bei Deklarationen oder Aufrufen, die wahrscheinlich zu einer Rekursion führen, einen Fehler ausgeben.

`before`-Methoden haben eine Besonderheit: sollte der Aspekt eine seiner `before`-Methode ausführen, so bestimmt die Rückgabe dieser Methode, wie es weitergeht: bei *true* soll der ursprüngliche Methodenaufruf normal ausgeführt werden, bei *false* soll die Ausführung unmittelbar nach der `before`-Methode abgebrochen werden, d.h. die ursprüngliche Methode kommt nicht zur Ausführung (und damit auch keine weiteren `before-` oder `after`-Methoden).

Es ist erlaubt, dass mehrere Aspekte die gleiche Klasse adaptieren. Dabei kann es vorkommen, dass sie die selbe Basismethode binden. In diesem Fall soll die Ausführungsreihenfolge der Aspektmethoden der Deklarationsreihenfolge der Aspekte entsprechen: der Aspekt, der zuerst deklariert wurde, wird bei `before`-Bindungen zuletzt, bei `after`-Bindungen zuerst ausgeführt. Sollte innerhalb desselben Aspekts die selbe Basismethode mehrfach gebunden werden, so ist die Ausführungsreihenfolge dieser Aspektmethoden beliebig.

Aktivieren von Aspekten

Aspekte sollen dynamisch ein- und ausschaltbar sein. Dazu bietet ein Aspekt standardmäßig die Methoden `enable` und `disable` an. Ein Aspekt ist zu Beginn stets inaktiv.

```
1 Aspect{'MyAspect'; ...}
2 ...                    -- Aspekt ist inaktiv
3 MyAspect:enable()      -- Aspekt wird aktiviert
4 ...                    -- Aspekt ist aktiv
5 MyAspect:disable()     -- Aspekt wird deaktiviert
6 ...                    -- Aspekt ist inaktiv
```

Sämtliche beschriebenen Effekte eines Aspekts gelten nur, wenn der Aspekt aktiv ist. Ein inaktiver Aspekt hat keinerlei Auswirkung: Seine Attribute und Methoden sind nicht in den Klassen vorhanden; `before` und `after`-Methoden werden niemals ausgeführt. Bei mehrmaliger Aktivierung und Deaktivierung eines Aspekts sollen Objekte die ihnen zugewiesenen Werte für Aspektattribute behalten, auch wenn auf diese in der deaktivierten Phase natürlich nicht zugegriffen werden kann.

Bearbeitung und Abgabe

Die Aufgabe soll in Gruppen von 3 Personen bearbeitet werden. Die Abgabe erfolgt in eurem Gruppenforum. Legt dazu einen neuen Beitrag mit dem Betreff *Abgabe Übung 2* an und fügt die Dateien gepackt in einem einzelnen `zip`-Archiv als Anhang ein. Im Kopf jeder Datei müssen (als Kommentar) Namen und Matrikelnummern aller aktiv mitarbeitenden Gruppenmitglieder stehen. Mit der Aufgabe wird ein kleines Testprogramm bereit gestellt, das natürlich mit eurer LOS-Version ausführbar sein muss.