Objektorientierte Softwareentwicklung

- Softwaretechnik -

SoSe 2014

Jähnichen
Duske

Ausgabe: Mittwoch, 28.05.2014
Abgabe: Freitag, 13.06.2014

2. Übungsblatt: Weiterentwicklung von LOS

Als weitere Ausbaustufe von LOS wollen wir Vererbung ermöglichen. Weiterhin wollen wir eine einfache Form von Aufzählungstypen integrieren. Die LOS-Erweiterung soll auf der Lösung zum 1. Übungsblatt basieren - aufgedeckte Fehler sollten zuerst korrigiert werden.

Vererbung

Unser Hauptziel ist die Nutzung von Features einer Oberklasse. Wir möchten geerbte Methoden aber auch in der Subklasse überschreiben können. Zudem erlauben wir Polymorphie bzw. Subtyping.

Klassen

Wir erweitern die Syntax zur Klassenerzeugung um ein optionales zweites Argument:

```
1 Class{'MySubClass', MyClass,
2   attribute3 = Number,
3   attribute4 = MyClass
4 }
```

Wenn hier eine bereits definierte Klasse angegeben wird, dann soll diese bei der neuen Klasse als Oberklasse (`_super`) eingetragen werden. Wird ein Wert übergeben, der keine gültige Oberklasse darstellt, soll ein Fehler ausgegeben werden.

Attribute

Attributsdeklarationen dürfen nicht überschrieben werden. Eine Klasse darf also kein Attribut gleichen Namens wie eine ihrer Oberklassen deklarieren. Dies soll durch eine Fehlermeldung aufgezeigt werden.

Bei der Zuweisung an ein Attribut kann nun auch gelten, dass die Klasse das Attribut zwar nicht selbst deklariert hat, jedoch eine ihrer Oberklassen. Falls der Typ des Attributs eine Klasse ist, sind neben Instanzen dieser Klasse auch Instanzen der Subklassen als Wert erlaubt. Ungültige Zuweisungen resultieren weiterhin in Fehlermeldungen.

Methoden

Die Deklaration von Methoden ist wie bisher möglich. Hierbei wird nun entweder eine geerbte Methode überschrieben oder eine neue erstellt. Gleiches gilt für Konstruktoren. Beim Zugriff auf eine Methode, wird zunächst in der Klasse selbst gesucht. Falls sie dort nicht gefunden wird, wird die Liste der Oberklassen durchsucht. Ist die Methode auch dort nicht vorhanden, wird eine Fehlermeldung ausgegeben.

`super`-Aufrufe

Innerhalb von Methoden sollen gezielt Aufrufe von Methoden der Oberklasse erlaubt sein. Für solche Aufrufe soll statt `self` das Schlüsselwort `super` vorangestellt werden:

```
1 function MySubClass:hello(x)
2   self.attribute3 = self.attribute3 + x
3   super:hello()
4 end
```

Zudem sollen mittels `super` innerhalb von `create`-Methoden (und nur dort) auch überschriebene Konstruktoren aufgerufen werden können:

```
1 function MySubClass:create(param1, param4)
2   super:create(param1)
3   self.attribute4 = param4
4 end
```

Bei falscher Verwendung von `super`-Aufrufen, z.B. außerhalb jeglicher Methode, in Klassen ohne Oberklasse, oder mit Oberklasse ohne entsprechende Methode, soll eine Fehlermeldung ausgegeben werden.

Aufzählungstypen

Neben den Basistypen der ersten Übungsaufgabe sowie den Klassentypen erweitern wir LOS um eine weitere Art von Typen: Aufzählungstypen, die eine festgelegte Menge an zulässigen Werten haben.

Aufzählungstypen sind keine Erfindung der Objektorientierung, wir wollen sie jedoch in Hinblick auf die dritte Übungsaufgabe in unsere objektorientierte Sprache und deren Typsystem integrieren.

Definition

Aufzählungstypen werden mit folgender Syntax erzeugt:

```
Enum{'MyEnum',
  {'value1', 'value2', default = 'value3'}
}
```

MyEnum ist der Name des zu definierenden Aufzählungstyps, unter welchem der Typ global verfügbar sein soll. Daraufhin folgt eine nicht-leere Tabelle, welche die zulässigen Werte des Typs enthält. Sowohl für den Namen als auch für die Werte sind nur nicht-leere Strings ohne Leerzeichen erlaubt. Mehrere Werte des gleichen Namens sind nicht erlaubt.

Die Benennung eines der Werte als `default` ist optional. Wird kein Wert explizit als `default` benannt, so ist der erste angegebene Wert der `default`-Wert. Die Werte eines Aufzählungstypes werden einmalig festgelegt und dürfen danach nicht ergänzt, geändert oder gelöscht werden.

Verwendung

Auf die Werte des Aufzählungstyps kann mittels der Punktnotation `MyEnum.value1` zugegriffen werden. Vergleiche zwischen zwei Werten sollen genau dann `true` liefern, wenn es sich um den gleichen Wert des gleichen Aufzählungstyps handelt:

```
print( MyEnum.value1 == MyEnum.value1    )    --> true
print( MyEnum.value1 == MyEnum.value2    )    --> false
print( MyEnum.value1 == OtherEnum.value1 )    --> false
print( MyEnum.value1 == "value1"         )    --> false
```

Sowohl die Werte als auch der Aufzählungstyp selbst haben eine `tostring`-Repräsentation, die den Wert als String bzw. den Namen des Aufzählungstyps liefert:

```
print( tostring(MyEnum) )            --> MyEnum
print( tostring(MyEnum.value1) )     --> value1
```

Wird ein Aufzählungstyp für ein Attribut in einer Klasse verwendet, so sind seine Werte die zulässigen Belegungen. Insbesondere ist `nil` kein zulässiger Wert. Bei der Instanziierung eines Objekts, wird ein solches Attribut mit dem `default`-Wert des Aufzählungstyps initialisiert.

Auch für Aufzählungstypen resultiert eine fehlerhafte Verwendung natürlich in einer entsprechenden Fehlermeldung.

Hinweise

- Da `super` ein allgemeines 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 vernachlässigt werden darf.

- Auch in dieser Aufgabe sollte zunächst die Hauptfunktionalität implementiert werden. Erst dann folgen Fehlerfälle und aufwändigere Features wie `super`-Aufrufe.

Bearbeitung und Abgabe

Die Aufgabe soll in Gruppen von maximal 3 Personen bearbeitet werden. Mit der Aufgabe wird ein kleines Testprogramm (`uebung2_test.lua`) bereitgestellt, das natürlich mit der zu erstellenden LOS-Implementierung ausführbar sein muss. *Wir empfehlen, die LOS-Implementierung darüber hinaus mit eigenen Tests zu prüfen. Wir werden das tun!*
Die Abgabe erfolgt im Gruppenforum auf ISIS. Dazu soll ein neues Thema mit dem Betreff *„Abgabe Übung 2“* angelegt werden. Die Dateien werden gepackt in einem einzelnen `zip`-Archiv als Anhang eingefügt. Im Kopf jeder Datei müssen (als Kommentar) Namen und Matrikelnummern aller aktiv mitarbeitenden Gruppenmitglieder stehen.