Objektorientierte Softwareentwicklung

- Softwaretechnik -

SoSe 2014

Jähnichen
Duske

Ausgabe: Mittwoch, 07.05.2014
Abgabe: Freitag, 23.05.2014

1. Übungsblatt: LOS

Ziel dieser und der zweiten Aufgabe ist es, einen Interpreter für eine objektorientierte Programmiersprache zu implementieren: **LOS** (*Lua Objekt Sprache*). Die Spezifikation der Sprache wird in mehreren Schritten vorgestellt. Den ersten Schritt, welcher in dieser Aufgabe umzusetzen ist, machen wir mit der Einführung von Objekten und Klassen inklusive Type Checking. LOS soll auf Basis der aktuellen Lua-Version 5.2 entwickelt werden.

Objekte, Klassen

Objekte und Klassen in LOS sollen durch Lua-Tables implementiert werden. Objekte speichern nur ihren Zustand, ihre Methoden (Funktionen mit implizitem erstem Argument `self`) sollen in Klassen ausgelagert werden.

Klassen

Klassen werden mit folgender Syntax erzeugt:

```
1 Class{'MyClass',
2   attribute1 = String,
3   attribute2 = MyClass
4 }
```

`Class` ist kein Schlüsselwort von Lua, sondern eines der Erweiterung LOS. Die obige Syntax ist ein Funktionsaufruf mit einer drei-elementigen Liste als Argument - `f{}` ist in Lua lediglich eine Abkürzung für `f({})`.

1. *MyClass* ist der Name der zu definierenden Klasse. Die erzeugte Struktur soll anschließend unter diesem Namen global verfügbar sein. Als Namen für Klassen sind nur nicht-leere Strings ohne Leerzeichen erlaubt. Ist der angegebene Name bereits vergeben, kann keine Klasse erstellt werden und eine Fehlermeldung wird ausgegeben.

2. Im Anschluss kann optional eine beliebige Anzahl von Attributen deklariert werden. Eine Attributdeklaration besteht aus einem Schlüssel (Name des Attributs als String) und einem Wert (Typ des Attributs). Als Typ für ein Attribut kommen die Basistypen *String*, *Number* und *Boolean* in Frage, sowie alle selbstdefinierten Klassen. Stellt ein Schlüssel-Wert-Paar keine gültige Attributsdeklaration dar, soll ein Fehler ausgegeben werden.

Im Anschluss an die Klassendefinition können der Klasse Methoden hinzugefügt werden. Dies geschieht mit folgender Syntax (Beispiel):

```
1 function MyClass:hello()
2   print('hello')
3 end
```

Es ist nicht erlaubt, einer Klasse mehrere Methoden gleichen Namens hinzuzufügen bzw. bereits vorhandene Methoden zu überschreiben oder zu löschen. Auch Namen, die für Attribute der Klasse verwendet werden, stehen nicht für Methoden zur Verfügung. Diese Fälle soll durch entsprechende Fehlermeldungen abgefangen werden.

Methoden können nur an Objekten aufgerufen werden, ein Aufruf an der Klasse selbst führt zu einer Fehlermeldung.

Objekterzeugung

Zur Instanziierung von Objekten soll an Klassen eine Methode `create` als Konstruktor zur Verfügung stehen:

```
1 obj = MyClass:create()
```

Nach der Initialisierung sollen sämtliche Attribut-Felder eines Objekts mit Default-Werten für den jeweiligen Typ belegt sein:

- 0 für *Number*
- Leerer String für *String*
- `false` für *Boolean*
- `nil` für Objekt-Referenzen.

Der Konstruktor wird automatisch von LOS bereitgestellt und muss nicht vom Programmierer eines LOS-Programms definiert werden. Der Standard-Konstruktor darf allerdings in der Klasse durch einen spezielleren Konstruktor, z.B. mit Parametern, überschrieben werden:

```
function MyClass:create(param1)
  self.attribute1 = param1
end
```

Es gibt in jeder Klasse genau eine Version des Konstruktors. Weitere Konstruktoren werden in LOS nicht unterstützt. Mögliche Rückgaben einer `create`-Methode müssen nicht berücksichtigt werden. Anders als die oben erwähnten Methoden, kann die `create`-Methode an der Klasse selbst, aber nicht an Objekten aufgerufen werden.

Ein spezielles Feld `_class` im Objekt soll die Referenz auf seine Klasse halten. (Für `_class` wie auch für alle anderen „*_versteckten*“ Felder gilt die Konvention, dass sie nur zu internen Zwecken dienen. Für den Programmierer eines LOS-Programms soll es nicht notwendig sein, diese Variablen zu kennen oder zu nutzen.)

Attribute und Methoden

Attribute und Methoden werden anhand ihrer Namen eindeutig identifiziert. Als Namen für Attribute und Methoden sind nur nicht-leere Strings ohne Leerzeichen erlaubt. Für jede Zuweisung an ein Attribut soll gelten, dass die Klasse des Objekts das Attribut deklariert haben muss und dass der Typ des zuzuweisenden Wertes dem deklarierten Typ entspricht. Der Wert `nil` darf genau dann zugewiesen werden, wenn der Typ des Attributs eine Klasse ist. Bei einer unerlaubten Zuweisung soll ein Fehler ausgegeben werden.
Auf Attribute kann nur an Objekten zugegriffen werden, ein Zugriff an einer Klasse führt zu einer Fehlermeldung.

Features (Attribute und Methoden) werden in Klassen deklariert und über Objekte aufgerufen. Beim Aufruf eines Features an einem Objekt muss zunächst geprüft werden, ob das Feature in der Klasse des Objekts deklariert ist. Wird keine passende Deklaration des Features gefunden, soll eine Fehlermeldung ausgegeben werden.
Die Werte von Attributen werden in den Objekten selbst gespeichert, die Implementierungen von Methoden finden sich dagegen in der entsprechenden Klasse.

Fehler und Ausgabe

Bei korrekten Programmen soll LOS keine Ausgaben produzieren, welche nicht vom Anwenderprogramm ausgelöst werden (d.h. insbesondere, dass keine Debugging-Informationen ausgegeben werden). Bei Programmen mit Verletzungen der LOS-Spezifikation soll mit der Funktion `error` eine aussagekräftige Fehlermeldung ausgegeben werden, welche über Problem und Ursache informiert. Darüber hinaus ist keine Fehlerbehandlung gefordert.

Hinweise

- Die genannten Basistypen existieren in Lua nicht als „Typ“. Sie sollen lediglich durch einen Textstring `'string'`, `'number'` bzw. `'boolean'` kodiert werden. Was stellt `String` im vorangehenden Beispiel in Lua eigentlich dar?

- Der Typ `MyClass` im ersten Beispiel existiert ebenfalls noch nicht, denn wir legen die Klasse in diesem Moment erst an. An Stelle des bekannten `nil`-Wertes müssen wir zwischen einer tatsächlich nicht vorhandenen Variable und einem Platzhalter unterscheiden.
  Wie genau wertet Lua einen Ausdruck der Form `f({k = v})` aus? Wie kann in diesen Vorgang eingegriffen werden?

- Das Überschreiben eines Konstruktors ist eigentlich nur eine Ergänzung des Default-Konstruktors. Die Instanziierung eines neues Objekts sowie die Initialbelegung der Attribute erfolgen immer, danach wird auf dem neu erstellten Objekt gegebenenfalls die spezielle `create`-Methode aufgerufen.

- Die „normale“ Nutzung von Lua soll so wenig wie möglich eingeschränkt werden, insbesondere soll der Variablenzugriff wie bisher möglich sein:

```
1    three = 3
2    print( three == 3 )       --> true
3    print( unknown == nil )   --> true
```

- Wir empfehlen, zunächst die Hauptfunktionalität für korrekte Programme zu implementieren und erst dann Fehlerfälle sowie Features, die eine Differenzierung zwischen `nil` und nicht vorhandenen Variablen erfordern, zu betrachten.

Bearbeitung und Abgabe

Die Aufgabe soll in Gruppen von maximal 3 Personen bearbeitet werden. Mit der Aufgabe wird ein kleines Testprogramm (`uebung1_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 1“* 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.