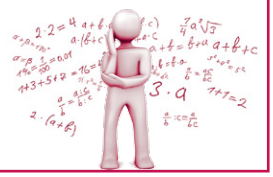


Auch Ceylon ist eine Insel

Ceylon 1.0

Klaus Rohe

Ceylon ist eine neue Programmiersprache für die JVM und JavaScript VMs, deren Entwicklung 2011 von Gavin King bei Red Hat angestoßen wurde. Die Sprache wird mit dem Ziel entwickelt, neueste Erkenntnisse über den Entwurf von Programmiersprachen zu berücksichtigen sowie die praktischen Erfahrungen mit den guten und schlechten Eigenschaften von Java in Ceylon einfließen zu lassen. In diesem Artikel wird ein kurzer Überblick gegeben und ein Blick in die Glaskugel bezüglich zukünftiger Akzeptanz von Ceylon versucht. Interessante Besonderheiten von Ceylon werden dabei anhand von Beispielen und im Vergleich mit Java vorgestellt.

Steckbrief von Ceylon

Ceylon ist eine Programmiersprache, deren Syntax Ähnlichkeiten mit C# und Java aufweist. Sie unterstützt sowohl imperative, objektorientierte als auch funktionale Programmierparadigmen. Ceylon ist statisch typisiert und streng blockorientiert, das heißt, die Sprache hat lexikalische Gültigkeitsbereiche für Variablen.

Ein Ceylon-Programm kann mit einer JVM oder JavaScript VM ausgeführt werden, abhängig davon, wie es kompiliert wurde. Aktuell sind allerdings nicht alle Ceylon-Module auch für JavaScript verfügbar, wie `ceylon.math`.

In Ceylon gibt es ein einheitliches Typsystem, das heißt, einfache (primitive) Datentypen, wie `char`, `int`, `float` und `double` in Java, bietet es nicht. In Ceylon gibt es Klassen und Schnittstellen. Mehrfachvererbung ist in Ceylon (wie in Java) nur über die Implementierung von unterschiedlichen Interfaces möglich. Überladung von Operatoren ist in Ceylon im Gegensatz zu Java möglich, aber nur mit einem sehr eingeschränkten Mechanismus, welcher als *Operator Polymorphism* bezeichnet wird, der im nächsten Abschnitt näher diskutiert wird.

Als Nächstes wird gezeigt, wie man einfache Ceylon-Programme von der Kommandozeile aus kompilieren kann, und in dem Abschnitt danach werden einige Unterschiede von Ceylon und Java behandelt. Eine ausführliche Diskussion ist im Rahmen dieses Artikels leider nicht möglich. Die Auswahl beruht auf einer subjektiven Sicht des Verfassers.

```
1 import ceylon.math.float { cos, toRadians }
2
3 /***
4   Higher order function that returns the polar equation
5   (function) of a conic depending on the parameters 'e' and 'p'.
6 ***/
7 shared Callable<Float,[Float]> conic(Float e, Float p) {
8   if (e == 0.0) {
9     print("Returning the polar equation (function) of a circle.\n");
10  }
11  else if (e == 1.0) {
12    print(
        "Returning the polar equation (function) of a parabola.\n");
13  }
14  else if (e > 0.0 && e < 1.0) {
15    print(
        "Returning the polar equation (function) of an ellipse.\n");
16  }
17  else {
18    print(
        "Returning the polar equation (function) of a hyperbola.\n");
19  }
20  Float r(Float theta) => (p / (1.0 + e * cos(toRadians(theta))));
21  return r;
22 }
23
24 "The runnable method of the module."
25 void run(){..
26   Callable<Float,[Float]> r = conic(0.6, 3.0);
27   variable Float angle = -1.0;
28   Float[] angles = [for (i in 0..360) angle = angle + 1.0];
29   for (theta in angles) {
30     print("r(" + theta.string + ") = " + r(theta).string);
31   }
32 }
```

Listing 1: Ein einfaches Ceylon-Programm

Kompilieren und ausführen eines einfachen Ceylon-Programms

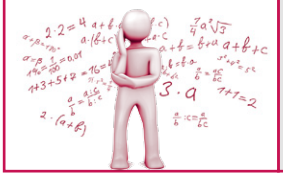
In Listing 1 ist der Quellcode eines ablauffähigen Ceylon-Programms dargestellt, welcher in der Datei `test01.ceylon` gespeichert ist. Diese Datei befindet sich in einem Verzeichnis `/source/com/test01`. In dem gleichen Verzeichnis befindet sich eine weitere Datei `module.ceylon`, die den folgenden Inhalt hat `module com.test01 "1.0.0" { import ceylon.math "1.0.0"; }`.

Das Programm wird von dem Verzeichnis oberhalb von `source` mit dem Befehl `ceylon compile com.test01` übersetzt. Nach der Übersetzung entsteht der Verzeichnisbaum `/modules/com/test01/1.0.0`, der den ausführbaren Ceylon-Code enthält. Ausgeführt wird das Programm dann mit dem Kommando `ceylon run com.test01`.

Unterschiede von Ceylon und Java

Das Ceylon-Programm in Listing 1 weist deutliche Unterschiede zu einem einfachen, ablauffähigen Java-Programm auf. Es enthält keine Klassendeklaration und keine `main()`-Methode. Deren Aufgabe wird in Ceylon von der `run()`-Methode erledigt. Weiterhin sieht man, dass man in Ceylon rein prozedural programmieren kann. Ceylon ist also eine hybride Sprache im Gegensatz zu Java, die rein objektorientiert ist.

Ceylon unterstützt höherwertige Funktionen. Dies sind Funktionen beziehungsweise Methoden, die als Parameter oder Rückgabewert Funktionen haben. Eine solche Funktion namens `conic(…)` ist in Listing 1 ab Zeile 7 implementiert. Sie



gibt in Abhängigkeit von den Parametern `e` und `p`, welche Fließkommazahlen sind, eine reellwertige Funktion der Form:

r(Θ)=p/(1+e*cos(Θ))

zurück (je nach Wert von `e` und `p` stellt sie einen der Kegelschnitte Kreis, Ellipse, Hyperbel oder Parabel in Polarkoordinaten dar). In den Zeilen 8 – 19 wird zusätzlich auf `stdout` geschrieben, um welche Art von Kegelschnitt es sich handelt. Die zurückgegebene Funktion wird in Zeile 20 deklariert. In der Schleife in den Zeilen 29 – 31 in Listing 1 werden die Werte der Funktion r(Θ) für Θ= 0°..360° ausgegeben.

In der Schleife wird zu diesem Zweck durch das Array `angels` iteriert. Das Array wurde in Zeile 28 mit Hilfe eines Programmkonstrukts erzeugt, welches als *Comprehension* bezeichnet wird. Es ist auch in anderen Programmiersprachen wie Python oder Ruby verfügbar. Comprehensions erlauben die effektive Erzeugung von neuen Listen (allgemein Behältertypen) aus gegebenen Listen durch Anwendung einer Funktion auf jedes Element der Liste. Dieses Beispiel zeigt schon, dass Ceylon wesentlich ausdrucksstärker ist als Java und den Entwurf gut lesbarer Programme unterstützt, was für die Wartbarkeit von Software ein großer Vorteil ist.

Bezüglich der Kommentierung im Programmcode unterstützt Ceylon sowohl die von Java bekannte Art als auch die in Zeile 24 in Listing 1 gezeigte Methode. Für die Zugriffsrechte auf Methoden und Attribute kennt Ceylon, im Gegensatz zu Java, nur `shared`. Ein weiterer Unterschied zu Java ist, dass `shared` kein Schlüsselwort der Sprache ist, sondern eine Annotation. Wenn ein Attribut oder eine Methode mit `shared` annotiert ist, bedeutet dies, dass dieses Attribut beziehungsweise diese Methode überall sichtbar ist, wo auch die Schnittstelle beziehungsweise Klasse sichtbar ist, in der sie enthalten sind.

Weitere Eigenschaften von Ceylon und Unterschiede zu Java werden anhand des Ceylon-Programms in Listing 2 herausgearbeitet. In diesem wird eine Klasse `Vector2D` deklariert, die einen zweidimensionalen kartesischen Vektor repräsentiert. Die Klasse enthält die öffentlichen (`shared`) Attribute `xValue` und `yValue`, auf die über „Getter“ (Zeile 6 und 11) zugegriffen werden kann und die über entsprechende „Setter“ (Zeile 9 und 13) verändert werden können. Im Gegensatz zu Java wird kein expliziter Konstruktor deklariert, sondern es wird bei der Deklaration der Klasse hinter deren Namen (Zeile 3) eine Liste mit Parametern angegeben, die zum Initialisieren der entsprechenden Attribute der Klasse dienen.

In Listing 2 sind die Parameter mit `variable` annotiert, was bedeutet, dass sie nach Instanziierung der Klasse über die entsprechenden „Setter“ verändert werden können. Außerdem werden hier Voreinstellungen vorgenommen, nämlich `x` und `y` auf `0.0` gesetzt, falls eine Instanz der Klasse, wie in Zeile 26, ohne explizite Parameter erzeugt wird. Diese Parameter sind in Ceylon automatisch Instanzvariablen. Sie sind in der Parameterliste der Klassendeklaration in Zeile 3 nicht mit `shared` annotiert, daher sind sie außerhalb der Klasse nicht sichtbar und nur über die „Getter“ und „Setter“ `xValue` und `yValue` les- und veränderbar. Der „Getter“ für `xValue` ist in Zeile 6 deklariert, der „Setter“ in Zeile 9. In Zeile 11 und 13 entsprechend für `yValue`. In Zeile 15 ist ein „Getter“ `length` deklariert, der die Länge des Vektors ausgibt. In Zeile 17 wird der „Getter“ `string` überschrieben, den die Klasse `Vector2D` von der Klasse `Object` erbt. Dies ist an der Annotation `actual` sichtbar. Überschreiben wird in Ceylon auch als *refinement* bezeichnet.

```
 1 import ceylon.math.float { sqrt }
 2
 3 class Vector2D(variable Float x = 0.0, variable Float y = 0.0)
 4   satisfies Summable<Vector2D> {
 5   " Getter for x."
 6     shared Float xValue { return x; }
 7
 8   " Setter for x."
 9   assign xValue {..x = xValue;..}
10
11   shared Float yValue { return y; }
12
13   assign yValue { y = yValue; }
14
15   shared Float length { return sqrt(x * x + y * y); }
16
17   shared actual String string { return "(``x``, ``y``)"; }
18
19   shared actual Vector2D plus(Vector2D v) {
20     return Vector2D(x + v.xValue, y + v.yValue);
21   }
22 }
23
24 void run(){
25   Vector2D v1 = Vector2D(1.0, 3.0);
26   Vector2D v2 = Vector2D();
27   v2.xValue = 2.0;
28   v2.yValue = 1.0;
29   Vector2D v3 = v1 + v2;
30   print(v3);
31   print("length=" + v3.length.string);
32 }
```

Listing 2: Objektorientiertes Ceylon-Programm

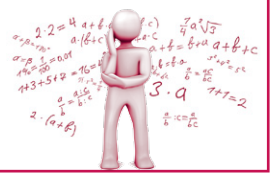
In Zeile 4 von Listing 2 wird durch das Schlüsselwort `satisfies` angezeigt, dass die Klasse `Vector2D` die generische Schnittstelle `Summable<T>` implementiert, in der die Methode `plus(…)` definiert ist. Diese Methode erwartet einen Parameter vom Typ `T` und hat einen Rückgabewert vom gleichen Typ. In diesem Fall ist der Typ `Vector2D`. Die Methode `plus(…)` ist für `Vector2D` in den Zeilen 19 – 21 so deklariert, dass sie die Addition von zweidimensionalen kartesischen Vektoren ausführt. Alle Typen, welche die generische Schnittstelle `Summable<T>` implementieren, können über den `+`-Operator verknüpft werden. Dies wird als *Operator Polymorphism* bezeichnet.

Vererbungsbeziehungen zwischen Klassen werden in Ceylon, genau wie in Java, über das Schlüsselwort `extends` deklariert. Wie in Java kann eine Klasse in Ceylon nur von einer

SDK und Entwicklungsumgebung

Ende 2013 wurde die Version 1.0.0 von Ceylon freigegeben. Aktuell gibt es eine Entwicklungsumgebung, die kommandozeilen-orientiert ist, sowie ein Plug-in für die Eclipse-Versionen Juno und Kepler. Die kommandozeilen-orientierte Entwicklungsumgebung ist für alle gängigen Betriebssysteme verfügbar (z. B. für Windows als zip-Datei). Die Eclipse Ceylon IDE erhält man mit dem üblichen Mechanismus zur Installation von Updates und Erweiterungen. Für Eclipse kann man dann auch noch ein Projekt von Git importieren, welches alle Beispiele des „Walkthroughs“ in [Ceylon] enthält.

Bevor man die Arbeit mit der Kommandozeilen-Version beginnt, sollte man unbedingt die Datei „README.md“ lesen, die sich in der Wurzel des Ceylon-Installationsverzeichnisses befindet. Die Umgebungsvariable `PATH` muss den vollständigen Pfad zu `ceylon-1.0.0/bin` enthalten und die Umgebungsvariable `JAVA_HOME` muss entsprechend gesetzt werden. Ceylon benötigt eine zu Java 7 kompatible JVM. Wer Ceylon ohne großen Aufwand ausprobieren möchte, kann den Ceylon-Web-Runner [CeylonWR] nutzen.



Merkmal	Ceylon	Java
Name der Quellcodedatei	Beliebig, muss aber die Erweiterung `.ceylon` enthalten	Muss den Namen der implementierten Klasse mit Erweiterung `.java` haben
Namenskonvention für Klassen, Methoden usw.	Klassennamen müssen mit einem Großbuchstaben beginnen. Namen von Methoden usw. müssen mit Kleinbuchstaben anfangen. Dies wird vom Ceylon-Compiler erzwungen. In Interoperabilitätsszenarien, in denen diese Namenkonvention nicht möglich ist, kann sie durch Voranstellen von `\I` bzw. `\i` ausgeschaltet werden	keine
Schnittstellen (interfaces)	Schnittstellen können implementierte Methoden enthalten	Schnittstellen können nur Methodensignaturen enthalten
Klammerung in `if-` ..., `else if-` ..., `else`-Blöcken	Auch wenn nach `if`, `else if` oder `else` nur eine Anweisung folgt, muss diese in geschweifte Klammern gesetzt werden	Man kann in Java in diesem Fall auf geschweifte Klammern verzichten
Verhalten der `switch`-Anweisung	Kein „fall through"-Verhalten wie in C	„fall through"-Verhalten wie in C
`for () { … } else { … }`-Konstrukt	Der `else { … }`-Block wird nicht ausgeführt, wenn die `for`-Schleife mit einer `break`-Anweisung verlassen wurde	Gibt es in dieser Form nicht
Implizite Konvertierung von numerischen Typen bei Zuweisung	Nein, es müssen die entsprechenden Attribute der Schnittstelle `Number` benutzt werden	Ja, Zuweisungen in Pfeilrichtung sind möglich: `byte->char, char->int->long->float->double`

Tabelle 1: Einige weitere Unterschiede zwischen Ceylon und Java

anderen Klasse erben. Mehrfachvererbung kann man nur über Schnittstellen realisieren.

Weitere Merkmale von Ceylon, in denen es sich von Java unterscheidet, sind in Tabelle 1 aufgeführt, die aber keinen Anspruch auf Vollständigkeit erhebt.

Interoperabilität mit Java

Ceylon und Java sind interoperabel. Es können sowohl Java-Klassen in Ceylon-Programmen genutzt werden als auch Ceylon-Module in Java. Wobei die erste Möglichkeit – bei der heute existierenden Menge von Java-Code – aktuell die wichtigere ist.

Um Java-Klassen in Ceylon-Programmen nutzen zu können, muss in der Datei `module.ceylon` die Zeile `import Java.Base „7";` eingefügt werden. In dem Ceylon-Programm werden dann die Java-Klassen, die man benutzen möchte, über eine entsprechende `import`-Anweisung bekannt gemacht. Es gibt im Ceylon SDK ein Modul `ceylon.interop.java`, das Klassen und Methoden enthält, welche eine sichere Umwandlung von komplexeren Java-Datentypen, wie Iteratoren, in die entsprechenden Ceylon-Datentypen ermöglichen.

Ein einfaches Ceylon-Beispielprogramm, welches Java-Klassen benutzt, ist in Listing 3 zu sehen. In Zeile 2 werden die Java-Klassen `File` und `PrintWriter` importiert. Der zweite Parameter der Ceylon-Methode `test4Prime` ist vom Typ `PrintWriter` und wird dazu benutzt, falls der erste Parameter eine Primzahl ist, diese in eine Textdatei zu schreiben. Diese wird in Zeile 15 durch Aufrufe des Konstruktors der Klasse `File` angelegt, danach wird eine Instanz von `PrintWriter` erzeugt und als zweiter Parameter der Methode `test4Prime` in Zeile 18 übergeben.

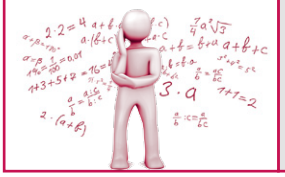
```
 1 import ceylon.math.float { sqrt }
 2 import java.io { File, PrintWriter }
 3
 4 void test4Prime(Integer n, PrintWriter pw) {
 5   variable Integer m = sqrt(n.float).integer + 1;
 6   for (i in 2..m) {
 7     if (n % i == 0) { break; }
 8   }
 9   else {
10     pw.println(n.string);
11   }
12 }
13
14 void run () {
15   File f = File("primelist.txt");
16   PrintWriter output = PrintWriter(f);
17   for (i in 2..10000) {
18     test4Prime(i, output);
19   }
20   output.close();
21 }
```

Listing 3: Einfaches Beispiel für Ceylon-Java-Interoperabilität

Hiermit wird die Vorstellung der Sprache Ceylon beendet und ich werde versuchen, einen Blick in die Zukunft zu werfen.

Die Zukunft von Ceylon

Das Ziel von Ceylon ist es, die Implementierung von robuster Software zu unterstützen und dem Programmierer Mittel zu geben, die ihn produktiver machen und es gestatten, gut verständliche und möglichst fehlerfreie Programme zu schreiben. Dieses Ziel hat Ceylon meiner Meinung nach erreicht, vor allem dadurch, dass es die unterschiedlichen Programmierparadigmen gut unterstützt und man dadurch in der Lage ist, das



für eine Implementierungsaufgabe am besten geeignete Paradigma zu wählen.

Wichtig für die Zukunft von Ceylon wird sein, wie gut sich die Interoperabilität mit existierendem Java-Code in der Praxis bewährt. Die Erfahrung mit neuen Programmiersprachen zeigt aber, dass sie es sehr schwer haben – auch wenn sie modernste Konzepte umsetzen und einfach und intuitiv zu handhaben sind –, sich gegen etablierte Sprachen durchzusetzen. Wie Tim Jones [Jon11] schreibt, ist der Rand des Weges in die Zukunft der Informatik mit Programmiersprachen übersät, die irgendwann als „the next big thing" gepriesen wurden, die sich aber trotz überlegener Eigenschaften nicht gegen das Beharrungsvermögen der etablierten Sprachen wie C, C++ und Java durchsetzen konnten.

Interessant ist auch, dass JavaScript, das unter dem Aspekt, welche Eigenschaften eine gute Programmiersprache heute haben sollte, wirklich nicht gut abschneidet, sich in den letzten Jahren dennoch sehr stark verbreitet hat. Dies letztlich, weil es eine sehr große Anzahl an Leuten gibt, die mit JavaScript programmieren können. Damit Ceylon eine Chance hat, sich bei der Entwicklung von Software einen gewissen Marktanteil zu verschaffen, muss es gelingen, sie in möglichst vielen Projekten erfolgreich einzusetzen und nachzuweisen, dass man damit besser und schneller Software entwickeln kann als mit den etablierten Programmiersprachen. Wichtig für die Verbreitung einer Programmiersprache ist auch ihr intensiver Einsatz in der Forschung und Lehre an Hochschulen. Ceylon hat dafür technisch gesehen sehr gute Voraussetzungen.

Links

[Ceylon100] Ceylon 1.0.0 is now available, 12.11.2013, `http://ceylon-lang.org/blog/tags/100/`

[Ceylon] Ceylon-Web-Seite, `http://ceylon-lang.org/`

[CeylonWR] Ceylon-Web-Runner, `http://try-ceylon.rhcloud.com/`

[Jon11] M. T. Jones, Ceylon: True advance, or just another language? Object-oriented and functional programming for the masses, 7.7.2011, `http://www.ibm.com/developerworks/library/l-ceylon/`

[King13] G. King, The Ceylon Language, Say more, more clearly, Version: Final release draft (1.0), 11.11.2013 `http://ceylon-lang.org/documentation/1.0/spec/html/`

[MALW13] S. Maple, A. Arhipov, E. Lindpere, O. White, Kapitel über „Ceylon" (ab Seite 11), in: The Adventurous Developer's Guide to JVM Languages, 29.4.2013, `http://zeroturnaround.com/rebellabs/the-adventurous-developers-guide-to-jvm-languages-java-scala-groovy-fantom-clojure-ceylon-kotlin-xtend/`

Klaus Rohe beschäftigt sich mit Softwaretechnik, Integrationstechnologien und freier Software für technisch wissenschaftliche Berechnungen und Visualisierungen. E-Mail: klaus-rohe@t-online.de.