Unterschiede zwischen Java und C / C++

In der grundlegenden Beschreibung der Programmiersprache Java von *James Gosling* werden die Gründe für das Beibehalten bzw. Abändern von Eigenschaften der Vorgängersprachen C / C++ erläutert. Es handelt sich um eine Veröffentlichung aus dem Jahre 1996 und viele spätere Spracherweiterungen, die Java zu der Allround-Programmiersprache gemacht haben, die sie heute ist, sind darin noch nicht erwähnt. Das Dokument ist dennoch wertvoll und lesenswert. Es hat den Status eines Klassikers erlangt:

The **Java Language Environment** (White Paper)

http://www.oracle.com/technetwork/java/langenv-140151.html

Gleich auf den ersten Seiten begründet Gosling die syntaktische Ähnlichkeit von Java und C/C++:

> "Another major design goal is that Java looks ***familiar*** to a majority of programmers in the personal computer and workstation arenas, where a large fraction of system programmers and application programmers are familiar with C and C++.
> Thus, Java "looks like" C++. Programmers familiar with C, Objective C, C++, Eiffel, Ada, and related languages should find their Java language learning curve quite short—on the order of a couple of weeks."

Eine weitere (allerdings recht abstrakte) Beschreibung der Programmiersprache Java wurde von SUN in dem Dokument **Java Language Specification** veröffentlicht (*James Gosling, Bill Joy, Guy Steele, Gilad Bracha)*.

http://docs.oracle.com/javase/specs/index.html

Java ist für einen in C und C++ geübten Programmierer leicht zu erlernen, da die Sprach-Syntax und -Grammatik sehr ähnlich ist. Dennoch gibt es einige grundsätzliche Unterschiede, die zu beachten sind.

- Strikte Objektorientiertheit (Klassenmodularität)
- Keine ungebundenen Klassen (strikte Klassenhierarchie)
- Keine Mehrfachvererbung
- Keine statischen (Stack-) Objekte
- Keine globalen Deklarationen und Definitionen
- Keine Einbindung externer Dateien
- Strikte Unterscheidung zwischen Wert- und Objektdatentypen
- Keine Operatoren für Adressen (* und &)
- Keine Wahl zwischen call-by-value / call-by-reference
- Keine implizite Konvertierung (Typenkontrolle)
- Keine Variablenüberdeckung in Methoden

Einige wichtige Unterschiede zwischen Java und C / C++

Eine Klasse bildet die minimale Grundeinheit, aus der ein Java-Programm bestehen muss. Jede Klasse ist automatischer Bestandteil der Java-Klassenhierachie und erbt grundlegende Attribute und Methoden von der obersten Klasse <`Object`>.

Objekte müssen grundsätzlich mit `new` erzeugt werden. Der Objektname ist eine Referenz (Zeiger) auf den Speicherort der Objektdaten. Es gibt keine statischen (Stack-)Objekte, sondern nur dynamische (Heap-) Objekte. Objekte können nicht explizit zerstört werden (es gibt kein delete), sondern sie werden durch die automatische Speicherbereinigung (garbage collection) entsorgt, wenn es keine gültige Referenz mehr gibt. Es gibt nur einen einzigen Referenzierungoperator, den Punkt.

Methoden müssen prinzipiell innerhalb der Klasse implementiert werden. Es lassen sich Objekt- und Klassenmethoden unterscheiden. Eine Objektmethode wird für ein Objekt aufgerufen und arbeitet mit dessen Daten, während eine Klassenmethoden nicht mit Objektdaten arbeiten kann, sondern als Ersatz für die nicht vorhandenen (globalen) Funktionen dient.

Attribute einer Klasse können sofort initialisiert werden.

Es gibt nur wenige Arten von Anweisungen, die außerhalb einer Klasse stehen dürfen, wie z.B. `import` (Bekanntmachung externer Klassen) oder `package` (Einbeziehung in ein Klassen-Modul). Die Import-Anweisung fügt (im Unterschied zu C/C++ <include>) keinen Code ein, sondern macht nur den genauen Ort einer vorhandenen Klassendeklaration bekannt. Andere Verzeichnisse können nicht explizit angegeben werden. Externe Dateien können nicht eingebunden werden.

Es sind keine globalen Typdefinitionen oder Makros erlaubt, die für mehrere Klassen gelten.

Nur elementare Datentypen (Zahlen / Buchstaben) sind Wertdatentypen. Ihr Name ist identisch mit ihrem Wert, der ein einzelner sein muss. Alle Datentypen, die aus mehreren Werten bestehen, werden grundsätzlich als Objekt behandelt. Alle Arrays, auch wenn ihre Elemente aus Wertdatentypen bestehen, sind Objekte und müssen mit `new` erzeugt werden. Bei Array-Deklarationen dürfen die [ ]-Klammern auch beim Datentyp stehen.

Es gibt keine Symbole für Pointer < * > und keine Operatoren für Adressmanipulationen <&>. Damit sind viele der Fehlerquellen ausgeräumt, die in C / C++ oft für Probleme sorgten. Intern arbeitet die Virtual Machine (Laufzeitumgebung) weiterhin mit Stack, Heap und Adressen, es sind allerdings keine direkten Speicherzugriffe erlaubt.

Bei der Argumentübergabe von Methoden gibt es keine Wahlmöglichkeiten zwischen call-by-value und call-by-reference. Alle Datentypen werden prinzipiell call-by-value übergeben. Bei Objekten haben alle Veränderungen, die sich nur auf den Objektnamen beziehen, nach Verlassen der Methode keine dauerhaften Auswirkungen - der Zugriff auf Objektdaten ist hingegen permanent.

Es gibt keine automatische (implizite) Typkonvertierung bei Zuweisungen, sondern es muss ausdrücklich (typ-) gecastet werden, auch beim Rückgabewert von Methoden.

Lokale Variablennamen dürfen in einem inneren Block einer Methode nicht neu deklariert (überschrieben) werden.

Main übernimmt beim Aufruf ein Array von Strings (ohne Programmname) und kann keinen Rückgabewert an die Betriebssystem-Ebene zurückgeben. Main muss als static-Methode Bestandteil einer (beliebigen) Klasse sein.

Deklarationen von Datentypen (Variablen oder Konstanten) müssen nicht mehr ganz am Anfang eines Gültigkeitsblocks stehen (wie in C / C++), sondern können an beliebiger Stelle erfolgen, an der sie benötigt werden. Zählervariablen können innerhalb der `for`-Anweisung deklariert werden und gelten dann nur innerhalb der Schleife.