Von Compilern, Interpretern und virtuellen Maschinen

Dozent:	Prof. Dr. Michael Eichberg
Kontakt:	michael.eichberg@dhbw.de, Raum 149B
Version:	1.0.1

Folien:	https://delors.github.io/prog-interpreter-vms-und-compiler/folien.de.rst.html https://delors.github.io/prog-interpreter-vms-und-compiler/folien.de.rst.html.pdf
Fehler melden:	https://github.com/Delors/delors.github.io/issues
Kontrollfragen:	https://delors.github.io/prog-interpreter-vms-und-compiler/kontrollfragen.de.rst.html

1. Einführung

Interpreter

Definition

Ein Interpreter ist ein Programm, das ein Programm einer bestimmten Programmiersprache direkt ausführt.

Arbeitsweise eines Interpreters:

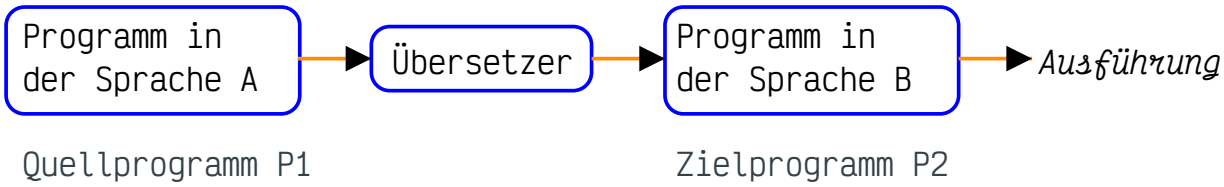
(`inpExpr` sei der aktueller Ausdruck des auszuführenden Programms)

1. Syntaktische Analyse von `inpExpr`
2. Überführung von `inpExpr` in eine Befehlsfolge der Maschinensprache, oder der Sprache, in der das Interpreterprogramm selbst geschrieben ist (`outExpr`)
3. Ausführung von `outExpr`
4. Wiederholung der Schritte (1) bis (3) für die nächste Anweisung.

Compiler

Definition

Ein Übersetzer (Compiler) ist ein Programm, das Programme aus einer Programmiersprache A in eine Programmiersprache B übersetzt



Legende

A ist die Ausgangssprache und B die Zielsprache.

Wesentlicher Aspekt ist die semantische Korrektheit: Jedem Quellprogramm P1 in A wird genau ein Zielprogramm P2 in B zugeordnet. Das dem Quellprogramm P1 zugeordnete Zielprogramm P2 muss die gleiche Bedeutung (Semantik) wie P1 besitzen.

Mutmaßlich erfunden von Konteradmiral Grace Murray Hopper (1906–1992)

Nach eigener Aussage hat sie den Compiler aus Faulheit erfunden, und weil sie hoffte, dass "Programmierer wieder Mathematiker werden" könnten.

Compiler - Übersetzungsphasen

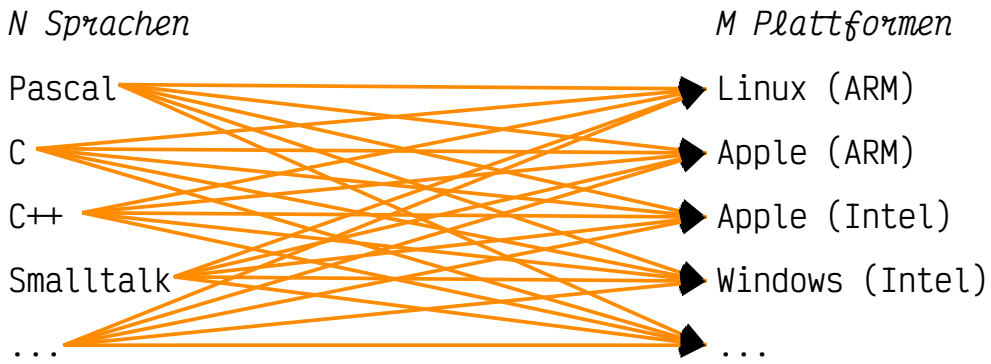
Lexikalische Analyse:
Quellprogramm wird in eine Folge von Worten zerlegt

Syntaktische Analyse:
Testet, ob das Quellprogramm den Syntaxregeln der Quellsprache entspricht. Strukturiert Worte in gültige Sätze.

Semantische Analyse:
Testet, ob alle im Quellprogramm benutzten Namen deklariert wurden und ihrem Typ entsprechend verwendet werden, usw.

Code-Generierung: Zielprogramm wird erzeugt.

Compiler - Übersetzung (Traditionell)



D. h. der Quelltext in Sprache A wird meinem spezialisierten Compiler für Sprache A und Zielplattform X in ein ausführbares Programm für X übersetzt.

✓ Performance/Effizienz: Optimale Ausnutzung der jeweiligen Prozessoreigenschaften und hohe Abarbeitungsgeschwindigkeit der übersetzten Programme.

! Plattformabhängigkeit: Ein Programm, das in einer höheren Programmiersprache geschrieben ist, kann - bei Verfügbarkeit eines Compilers - auf jeder Maschine laufen.

Interpreter - Vor- und Nachteile

✓es lassen sich relativ schnell lauffähige Programmversionen erstellen (Prototyping)

✓Schnelle Änderbarkeit: geänderte Anweisungen / Deklarationen des Quellprogramms sind sofort ausführbar

✓Neuübersetzung nicht notwendig

❗ Längere Ausführungszeit

❗ Werden Anweisungen des Quellprogramms k-mal verwendet (z.B. bei Schleifen), werden sie k-mal analysiert und überführt

❗ Bei Zugriffen auf Variablen müssen die zugeordneten Adressen immer wieder bestimmt werden.

Bemerkung

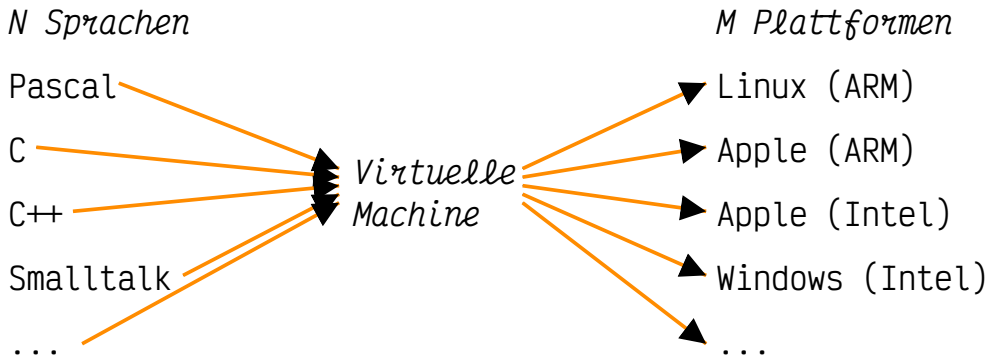
Auch heute gibt es noch *reine* Interpreter, aber Programmiersprachen wie Python und Java etc. setzen schon lange auf hybride Ansätze, die versuchen das Beste aus allen Welten vereinen.

Virtuelle Maschinen

Definition

Eine virtuelle Maschine ist ein Programm, das die Arbeit eines Prozessors in Software simuliert

- Programme einer höheren Sprache werden in eine Assembler-ähnliche Zwischensprache übersetzt.
- Der simulierte Hardware-Prozessor nutzt diese Zwischensprache und besitzt einige Software-Register
- Die Anweisungen der Zwischensprache nennt man auch Byte-Code.
- Die Zwischensprache wird von der Virtuellen Maschine interpretiert.
- Eine virtuelle Maschine versteckt die spezifischen Eigenschaften eines konkreten Prozessors. Wir haben somit eine neue Abstraktionsschicht auf der Hardware-Ebene!



Eine VM verdeckt die speziellen Eigenschaften des jeweiligen Prozessortyps und dient somit als Abstraktionsschicht!

✓Übersetzte Programme einer Sprache laufen auf allen Prozessortypen, für die es einen Byte-Code Interpreter(VM) gibt.

✓Es wird nur ein Compiler benötigt und die Sprache wird plattformunabhängig.

Natürlich braucht man eine VM pro Prozessortyp und Platform. Aber das ist ein geringerer Aufwand als für jede Sprache einen eigenen Compiler zu schreiben.

! Byte-Code Programme sind langsamer als Maschinenprogramme

Just-in-time-compiler (JIT) versuchen diesen Nachteil aufzulösen. Sie Übersetzen den Byte-Code in ein Objekt-Programm für einen speziellen Prozessortyp sobald es geladen wird, oder nach einer gewissen Anzahl an Ausführungen.

Struktur eines Java-Programms

Ein Java-Programm kann aus beliebig vielen Klassen bestehen, von denen mindestens eine die `main`-Operation besitzen muss (Hauptprogrammklasse).

Aufgaben von `main`

- Objekterzeugung; d. h. der Aufbau einer anfangs minimalen Welt
- Aufruf der ersten Operation
- Sollte in der Regel keinen weitergehenden Kontrollfluss des Java-Programms enthalten
  Der Kontrollfluss wird innerhalb der Objektoperationen realisiert.
- `main` wird mit Hilfe des Java-Interpreters gestartet und ausgeführt

Java-Laufzeitumgebung

Java Interpreter:
Programm zur Ausführung des Java-Bytecodes auf dem konkreten Rechner.

Just-In-Time-Compiler (JIT-Compiler):
Klassen bzw. Methoden werden bei Bedarf in Code der jeweiligen Maschine übersetzt. Ggf. auf verschiedenen Optimierungsstufen.

Runtime-System:
Stellt einem Java-Programm wichtige Ressourcen zur Verfügung.

Bytecode Verifier:
Überprüft, ob die geladenen Bytecodes der JVM-Spezifikation entsprechen.
Klassen können über das Netz oder aus dem lokalen Dateisystem zur Laufzeit einer Java-Anwendung in das Laufzeitsystem nachgeladen werden
Ein Teil der Sicherheitsmaßnahmen wird durch den Bytecode Verifier realisiert.

Achtung!

Die Sicherheit, die der Bytecode-Verifier und Java als solches bietet - ist nicht ausreichend, um die Sicherheit des Systems zu gewährleisten. Führen Sie Java Programme, denen Sie nicht vertrauen, niemals aus.

Java-Übersetzung

- Die Eingabe für `javac` sind ein oder mehrere Java-Dateien, die jeweils die eine oder mehrere Klassendefinitionen enthalten.
- Eine derartige Datei nennt man eine Übersetzungseinheit
- Die Ausgabe ist pro Klasse X genau eine Datei X.class, die den Bytecode der Klasse enthält.

Beispiel

HelloWorld.java (im Default Package)

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Kompilieren mit javac und Ausführen mit java

```
$ javac HelloWorld.java
$ ls
HelloWorld.class HelloWorld.java
$ java HelloWorld                                    # ACHTUNG kein ".java"!
Hello, World!
```

HelloWorld.java (im Package de.dhbw)

Die Datei HelloWorld.java muss im Verzeichnis de/dhbw liegen!

```java
package de.dhbw;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Kompilieren mit javac und Ausführen mit java

```
$ javac de/dhbw/HelloWorld.java
$ ls de/dhbw
HelloWorld.class HelloWorld.java
$ java de.dhbw.HelloWorld                # ACHTUNG vollständiger Name der Klasse!
Hello, World!
```

Übung

1.1. RPN Taschenrechner kompilieren/Java Code übersetzen

Stellen Sie sicher, dass Ihr Programm für den RPN Taschenrechner durch die Klasse RPN implementiert wird und diese Klasse in der entsprechenden Java Datei gespeichert ist. Die Klasse RPN soll im Package `rpn` sein! Die Klassen für den Stack und die Liste sollen im Package `ds` liegen.

Compilieren Sie nur Ihr Program und die benötigten Hilfsklassen in einem Schritt mit Hilfe von `javac`. Starten Sie danach Ihr Programm mit Hilfe von `java`. Vergessen Sie nicht den vollqualifizierten Namen der RPN Klasse zu verwenden. Stellen Sie auch sicher, dass Sie sich im passen Root-Verzeichnis befinden.

Falls Sie den Code nicht haben, dann können Sie den Code von hier verwenden. Achten Sie darauf die Dateien in den entsprechenden Verzeichnissen zu speichern!

Datei: rpn/RPN.java

```
1  package rpn;
2
3  import ds.Stack;
4
5  public class RPN {
6
7      static void printAll(Stack<String> stack) {
8          for (int i = 0; i < stack.size(); i++) {
9              System.out.print(stack.get(i) + " ");
10         }
11         System.out.println();
12     }
13
14     public static void main(String[] args) {
15         if (args.length == 0) {
16             System.out.println("Usage: java RPN <expr>");
17             return;
18         }
19         // Main logic
20         Stack<String> infix = new Stack<>();
21         Stack<Double> ops = new Stack<>();
22         for (String arg : args) {
23             switch (arg) {
24                 case "+":
25                     ops.push(ops.pop() + ops.pop());
26                     infix.push("(" + infix.pop() + " + " + infix.pop() + ")");
27                     break;
28                 case "*":
29                     ops.push(ops.pop() * ops.pop());
30                     infix.push("(" + infix.pop() + " * " + infix.pop() + ")");
31                     break;
32                 default:
```

```
                    infix.push(arg);
                    ops.push(Double.parseDouble(arg));
                }
            }
            System.out.println(infix.peek() + " = " + ops.peek());

            printAll(infix);
        }
}
```

Datei: ds/Stack.java

```
package ds;

import java.util.NoSuchElementException;

public class Stack<T> extends List<T> {

    public void push(T element) {
        add(element);
    }

    public T pop() {
        if (size() == 0) {
            throw new NoSuchElementException();
        }
        T element = get(size() - 1);
        remove(size() - 1);
        return element;
    }

    public T peek() {
        if (size() == 0) {
            throw new NoSuchElementException();
        }
        return get(size() - 1);
    }
}
```

Datei: ds/List.java

```
package ds;

import static java.lang.System.arraycopy;

public class List<T> {

    private T[] elements;
    private int count;

    public List(int size) {
        @SuppressWarnings("unchecked")
        var tElements = (T[]) new Object[Math.max(size, 16)];
        elements = tElements;
        count = 0;
```

```
    }

    public List() {
        this(16);
    }

    public int size() {
        return count;
    }

    public T get(int index) {
        if (index < 0 || index >= count) {
            throw new IndexOutOfBoundsException();
        }
        return elements[index];
    }

    public void add(T element) {
        if (count == elements.length) {
            Object[] newElements = new Object[Math.min(elements.length * 2, 1000)];
            System.arraycopy(elements, 0, newElements, 0, elements.length);
            @SuppressWarnings("unchecked")
            var tElements = (T[]) newElements;
            elements = tElements;
        }
        elements[count++] = element;
    }

    @SafeVarargs
    final public void addAll(T... elements) {
        for (T element : elements) {
            add(element);
        }
    }

    public void set(int index, T element) {
        if (index < 0 || index >= count) {
            throw new IndexOutOfBoundsException();
        }
        elements[index] = element;
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("[");
        for (int i = 0; i < count; i++) {
            if (i > 0) {
                sb.append(", ");
            }
            sb.append(elements[i]);
        }
        sb.append("]");
        return sb.toString();
    }
```

```
69
70      public void remove(int index) {
71          if (index < 0 || index >= count) {
72              throw new IndexOutOfBoundsException();
73          }
74
75          if (count < elements.length / 4 && elements.length > 16) {
76              @SuppressWarnings("unchecked")
77              T[] newElements = (T[]) new Object[Math.max(elements.length / 2, 16)];
78              arraycopy(elements, 0, newElements, 0, index);
79              arraycopy(elements, index + 1, newElements, index, count - index - 1);
80              elements = newElements;
81              count--;
82          } else {
83              arraycopy(elements, index + 1, elements, index, count - index - 1);
84              count--;
85          }
86      }
87
88  }
```