Modellierung und Programmierung 1

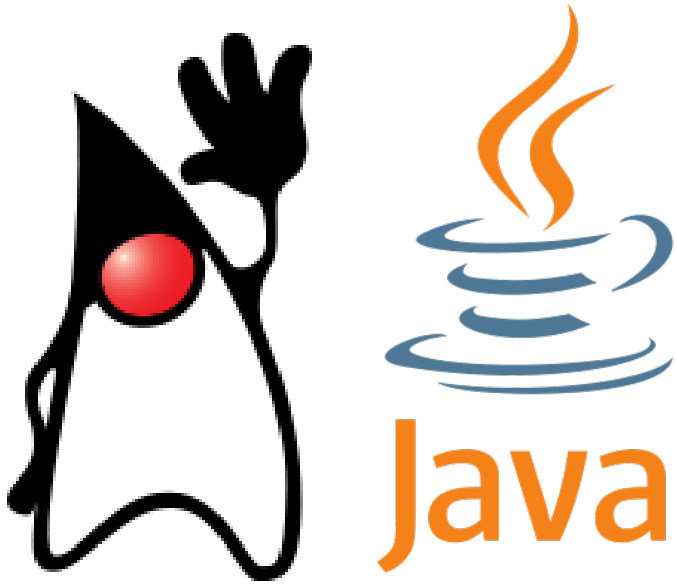
Prof. Dr. Sonja Prohaska

Computational EvoDevo Group
Institut für Informatik
Universität Leipzig

21. Oktober 2015

Entstehung von Java

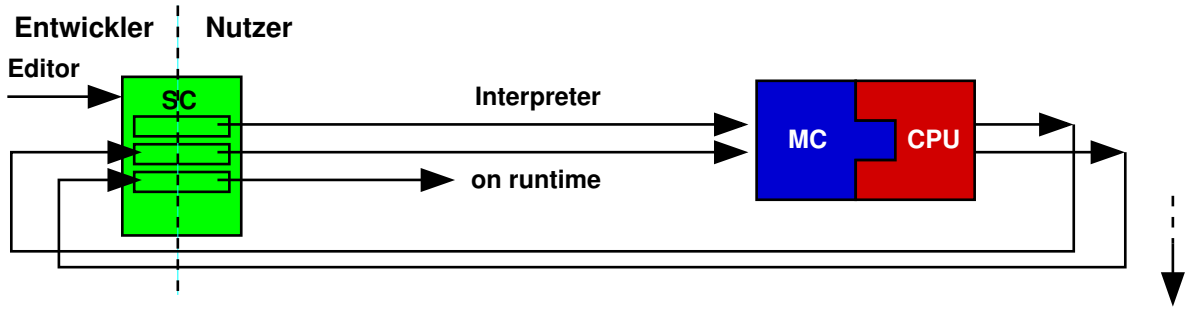
- 1991 entwickeln Mike Sheridan, James Gosling, Patrick Naughton u.a. bei *Sun Microsystems* die Programmiersprache **OAK (Object Application Kernel)**, ursprünglich zur Steuerung und Integration von Haushaltsgeräten.
- 1993 *OAK* ist klein, objektorientiert, platformunabhängig und robust und eignet sich für **Internet**-Anwendungen
- 1994 wird die Sprache in *Java* (**starker Kaffee**) umbenannt.
- 1995 wird *Java* in die führenden Web-Browser `Netscape Navigator` und `MicroSoft Internet Explorer` integriert.



Grundkonzept der Programmiersprache *Java*

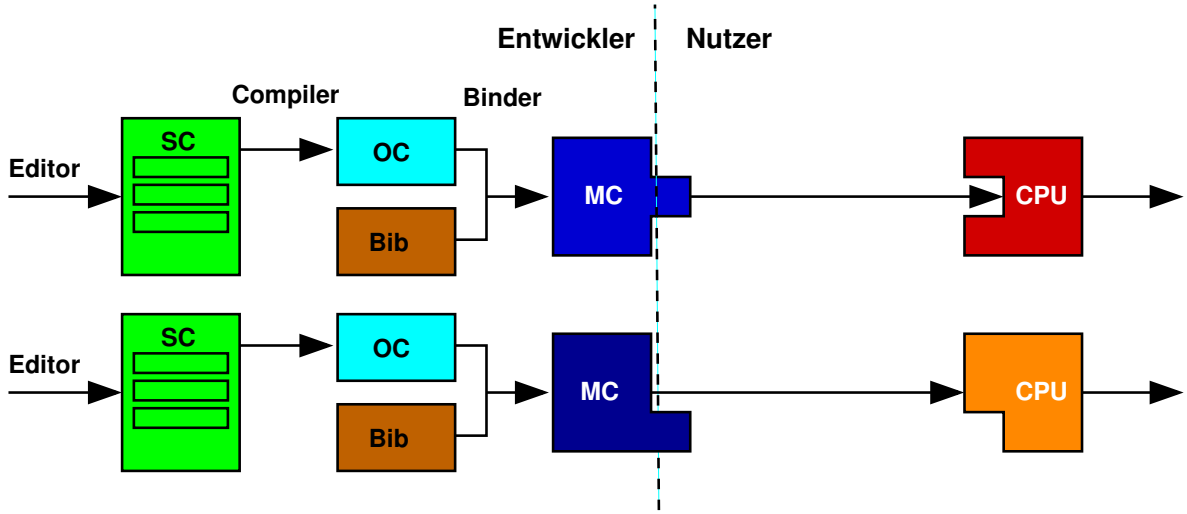
- **objektorientiert**, intuitiv, einfacher zu verstehen und zu erweitern
  syntaktische Ähnlichkeit zu `C++`, Nutzung von Klassenbibliotheken wie bei `Smalltalk`
- **robust und sicher**
  extensive Fehlerbehandlung und reduzierte Komplexität senkt die Wahrscheinlichkeit ungewollter Systemfehler
- **architekturneutral**
  unabhängig von der Rechnerarchitektur (Hardware) und Betriebssystem (Plattform)
- **portabel**
  verwendet Standards für Datenypen, Grössen und deren Verhalten
- **leistungsfähig**
  dank in-time-compilation und trotz der oben genannten Anforderungen

Interpreter



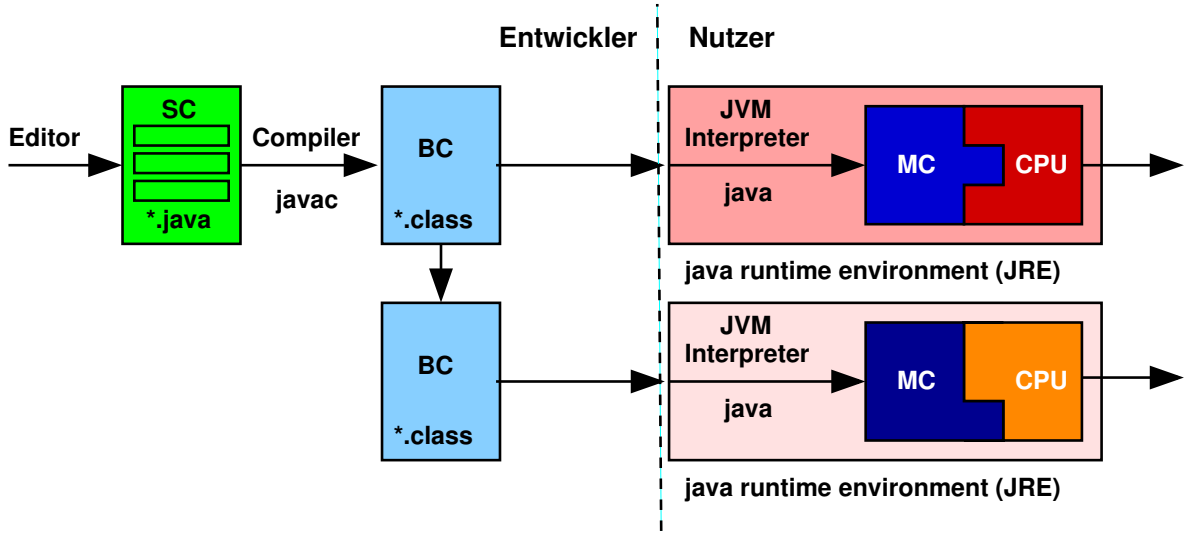
- Quellcode (engl. source code SC) bleibt bis zur Ausführung unbearbeitet
- die Übersetzung in Maschinecode (MC) liegt beim Nutzer
- pro Anweisung im Quellcode passiert folgendes
  - Interpreter wandelt QC in MC um
  - MC wird von der CPU ausgeführt
- interpretiert Programme sind langsamer
- Syntaxfehler werde erst zur Laufzeit bemerkt

Compiler



- Quellcode (SC) wird vom Compiler auf korrekte Syntax überprüft
- der Compiler erzeugt den Objektcode (OC)
- der Binder bindet die Bibliotheken ein und erzeugt MC
- die Übersetzung liegt beim Entwickler
- der Nutzer braucht MC specifisch für **seine** Plattform
- im Allgemeinen schnell
- schlecht geeignet im Zusammenschluss von verschiedenen Plattformen (Internet)

Java: Compiler and Interpreter



- der Quellcode (SC) wird seitens des Entwicklers vom Compiler in den platform**un**abhängigen **Bytecode** (BC) übersetzt
- BC wird von einem Interpreter, der **Java Virtual Mashine (JVM)**, seitens des Nutzers in plattformabhängigen MC umgewandelt
- Vorteil: Plattformunabhänigkeit des Bytecodes
- Nachteil: Geschwindigkeitsverlust durch Interpreter
- Option *Just-in-time Compiler* (JIT): beim ersten Aufruf wird das Programm in runtime compiliert und für weitere Durchläufe abgespeichert

Installieren und Starten von Java

- Java^{TM} SE Development Kit 8
  SE: **S**tandard **E**dition
  JDK: **J**ava **D**evelopment **K**it
  synonyms: Java SE 8, JDK 8, jdk-1.8, `java version "1.8.0_60"`
- Java^{TM} SE Runtime Environment 8
  SE: **S**tandard **E**dition
  JRE: **J**ava **R**untime **E**nvironment

- **Java application** – Programm (.java) für den Computer
  - `javac MyApplication.java  // compile source code`
    erzeugt `MyApplication.class`
  - `java MyApplication  // interpret bytecode`
- **Java applet** – kleines Programm (.java) für Web-Anwendung
  - `javac MyApplet.java  // compile source code`
  - entweder `appletviewer MyApplet.html`
    `MyApplet.html` mit Internet-Browser öffnen

Java Application

Es wird der Quellcode im Dateiformat `.java` benötigt.

```java
// MyApplication.java -- S.J. Prohaska, 2015

public class MyApplication {
    public static void main (String[] args) {
        System.out.println("Die Wissenschaft und ihre Lehre ist frei.");
    }
}
```

Quellcode von `MyApplication.java` mit Java syntax highlighting (emacs).

```
cachaca ~/MuP1/progs02$ javac MyApplication.java
cachaca ~/MuP1/progs02$ java MyApplication
Die Wissenschaft und ihre Lehre ist frei.
cachaca ~/MuP1/progs02$
```

Konsolenaufruf des Compilers und des Interpreters, gefolgt von der Programmausgabe.

Die Klasse `MyApplication` ist die oberste Struktureinheit des Programms. Der Name der Klasse **muss** mit dem Namen des Programms übereinstimmen.

Java Applet

Es werden benötigt

- der **Quellcode** im Dateiformat `.java` und
- eine **HTML-Dokument**, in welches der Aufruf des Applets eingebettet ist

Ausführung des Applets

- das HTML-Dokument **zusammen** mit dem Bytecode des Applets (.class) ablegen (für Zugang über das Internet auf einem Web-Server ablegen)
- Java-interpreter des Internet-Browser aktivieren
- das HTML-Dokument mit einem Browser öffnen

  alternativ: mit dem `appletviewer`

Java Applet – .java und .html

```java
// MyApplet.java -- S.J. Prohaska, 2015

import javax.swing.JApplet;
import java.awt.Graphics;

public class MyApplet extends JApplet {
    public void paint (Graphics g) {
        g.drawString("Die Wissenschaft und ihre Lehre ist frei.", 50, 100);
    }
}
```

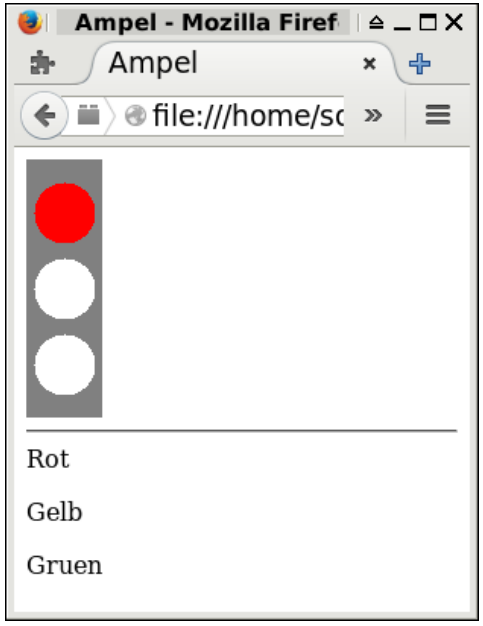
Quellcode von `MyApplet.java`.

```html
<html>
  <head>
    <title>Ein wichtiges Recht</title>
  </head>
  <body>
    <applet
       code =MyApplet.class width = 350 height = 200>
    </applet>
  </body>
</html>
```

HTML-Dokument welches den Bytcode `MyApplet.class` aufruft. (emacs mit HTML-Syntax highlighing)

JavaScript für Interaktion mit dem Applet

Das Applet der Klasse `AmpelApplet` soll ein Ampelschaltung Im Browser graphisch darstellen. Die Steuerung durch den Nutzer erfolgt über die drei Schaltflächen," Rot", "Gelb", "Gruen" unterhalb der Ampel.




```
<!--   Javascript   -->
    <a onMouseOver="document.A.setLampe( 1)">Rot
    </a><p>
    <a onMouseOver="document.A.setLampe( 2)">Gelb
    </a><p>
    <a onMouseOver="document.A.setLampe( 3)">Gruen
    </a>
```

Diese Applet ist (wegen Javascript) **nur** in Browser ausführbar, nicht aber mit dem `Appletviewer`.

JavaScript für Interaktion mit dem Applet

```java
// AmpelApplet.java    - Monika Meiler -        MM 2003

import java.applet.*;                          // Applet
import java.awt.*;                    // Graphics, Color

public class AmpelApplet extends Applet {

  int lampe = 0;

  public void paint(Graphics g)
  {
    setBackground(Color.gray);

    // Amplestellung:
    // 1 - rot, 2 - gelb, 3 - gruen, 0 - defekt
    switch(lampe)
    {
      case 1:  g.setColor(Color.red);
               g.fillOval(5, 15, 40, 40);
               g.setColor(Color.white);
               g.fillOval(5, 65, 40, 40);
               g.fillOval(5, 115, 40, 40);
               break;
      case 2:  g.setColor(Color.yellow);
               g.fillOval(5, 65, 40, 40);
               g.setColor(Color.white);
               g.fillOval(5, 15, 40, 40);
               g.fillOval(5, 115, 40, 40);
               break;
      case 3:  g.setColor(Color.green);
               g.fillOval(5, 115, 40, 40);
               g.setColor(Color.white);
               g.fillOval(5, 15, 40, 40);
               g.fillOval(5, 65, 40, 40);
               break;
      default: g.setColor(Color.red);
               g.fillOval(5, 15, 40, 40);
               g.setColor(Color.white);
               g.fillOval(5, 65, 40, 40);
               g.fillOval(5, 115, 40, 40);
    }
  }

  public void setLampe( int nr)
  {
    lampe = nr;
    repaint( 100L);
  }

}
```

```html
<html>
<!--  Monika Meiler                    MM 2003 -->
<!--  Diese Seite bindet das AmpelApplet ein.  -->

  <head>
    <title>Ampel</title>
  </head>

  <body>

<!--   Applet   -->
    <applet code=AmpelApplet.class name=A
            width=50 height=170>
    </applet>

    <hr>

<!--   Javascript   -->
    <a onMouseOver="document.A.setLampe( 1)">Rot
    </a><p>
    <a onMouseOver="document.A.setLampe( 2)">Gelb
    </a><p>
    <a onMouseOver="document.A.setLampe( 3)">Gruen
    </a>

  </body>
</html>
```

Applet
- Methode paint
- Methode setLampe

html-Dokument
- Javascript
  - ruft setLampe auf und uebergibt die "Farbe"

Output: Ausgabe

Methoden der **Java-Standardbibliothek**:
System.out.println() und System.out.print()
zur Ausgabe von Text und Zahlen auf der Konsole (stdout)
mit (println()) und ohne (print()) “new line & carrage return”

```
    int a = 3, b = 2;
    System.out.println( "a + b");      // '+' als Text
    System.out.println( a + b );       // '+' als Additions-Operator
    System.out.println( "a" + "b" );   // '+' Text-Verkn\"upfungs-Operator
```

```
a + b
5
ab
```

Analog: **System.err.println() und System.err.print()**
zur Ausgabe von Fehlermeldungen auf der Konsole (stderr)

Input: Eingabe über das `Tools` package

Tools.IO.IOTools ist eine Klasse für Tastatureingaberoutinen. Sie gehört **nicht** zu Java-Standardbibliothek.
Sie ermöglicht das Lesen von einer einzeiligen Tastatureingabe und gleichzeitige Zuweisung zu einem Datentyp.

Z.B. kann über die Methode `readInteger()` eine ganze Zahl eingelesen werden, die direkt, oder nach Aufforderung, über die Tastatur eigegeben wird.

```
int eingabe = IOTools.readInteger();

int eingabe = IOTools.readInteger("Nenne eine Zahl zwischen 1 und 100! ");
```

Aufgabe: Testen sie die funktionsweise der anderen Methoden der Klasse `IOTools`!

Das Tools package ist als zip-Archiv samt Dokumentation auf der MuP1-Webseite zu finden.