JAVA

objektorientiert :

- beim Programmieren werden insbesondere Daten und diese Daten verändernde Methoden beachtet;
- außer primitiven Datentyen sind alle Dinge in JAVA Klassen und Objekte;
- eine Klasse ist die kleinste ablauffähige Einheit in JAVA;
- alle JAVA-Programme sind Klassen.

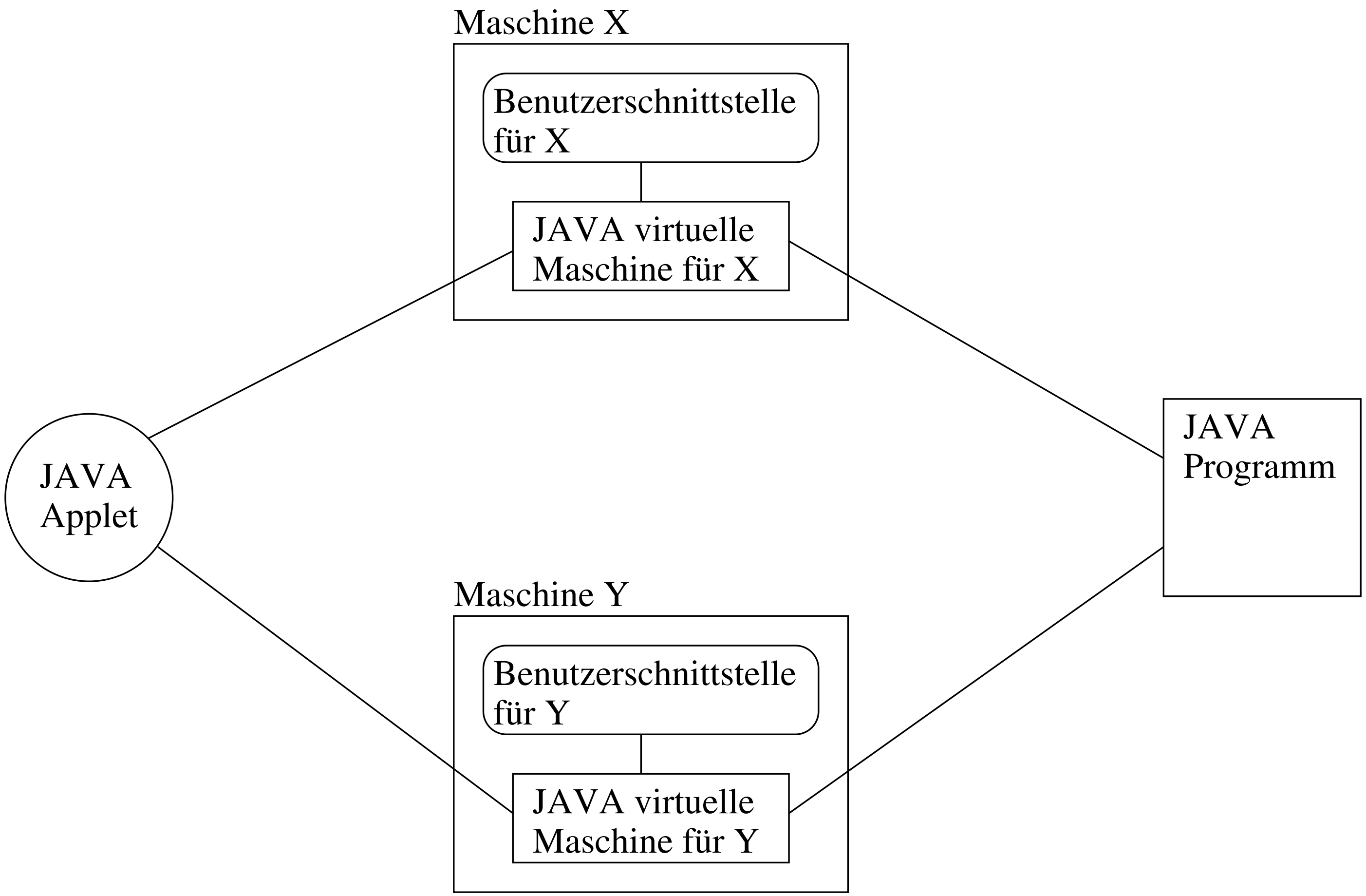
plattformunabhängig : die Sprache wird in eine virtuelle Maschine übersetzt, für die die Plattformen eine Schnittstelle bereitstellen;

klares Typ-Konzept :der Wertebereich einer Variablen ist festgelegt;

verteilt : die integrierte Netzwerkunterstützung und das Laden und Ausführen von Programmen über das Internet

nebenläufig : gleichzeitige Bearbeitung mehrerer Aufgaben

Architektur


Maschine X
Benutzerschnittstelle für X
JAVA virtuelle Maschine für X
JAVA Applet
JAVA Programm
Maschine Y
Benutzerschnittstelle für Y
JAVA virtuelle Maschine für Y

Vorgehen

- JAVA installieren – im Pizza-Pool geschehen
- Texteditor aufrufen

  `emacs &`

- JAVA-Programm schreiben; das Programm heißt `name.java`
- JAVA-Übersetzer aufrufen

  `javac name.java`

- Sie rufen Ihr Programm auf mit

  `java name`

Syntax

Lexikalische Ebene: wie aus einzelnen Zeichen Wörter zusammengesetzt werden dürfen.

Syntaktische Ebene: wie aus den Wörtern Sätze zusammengesetzt werden dürfen.

Grammatik Eine Grammatik besteht aus

- einem Startsymbol S,
- einem Alphabet A, das aus einer Menge von terminalen Symbolen T und einer Menge von nicht-terminalen Symbolen N besteht,
- einer Menge von Produktionen.

Produktionen

$\Phi, \Psi \in A^+$ Ketten von Symbolen des Alphabets

$n_i \in N$

Typ 0, unbeschränkte Grammatik: $\Phi \mapsto \Psi$

Typ 1, kontextsensitive Grammatik: $\Phi \mapsto \Psi$, wobei $S \notin \Psi$ und $\mid \Phi \mid \leq \mid \Psi \mid$

Typ 2, kontextfreie Grammatik: $n \mapsto \Psi$

Typ 3, reguläre Grammatik: $n_j \mapsto n_k \Psi$

Generierung mit einer Grammatik

Grammatik:

$S \mapsto Kopf\ Rumpf$
$Kopf \mapsto \texttt{Uta}$
$Rumpf \mapsto \texttt{spielt Ball}$
$Rumpf \mapsto \texttt{tritt den Ball}$

Generierung:

$$S \mapsto Kopf\ Rumpf,$$

$$Kopf\ Rumpf \mapsto \texttt{Uta}\ Rumpf,$$

$$\texttt{Uta}\ Rumpf \mapsto \texttt{Uta spielt Ball}$$

$$\text{oder} \mapsto \texttt{Uta tritt den Ball}.$$

Sprache:

`Uta spielt Ball`, `Uta tritt den Ball`.

Analyse

Satz: `Uta tritt den Ball`

Grammatik:

$S \mapsto Kopf\ Rumpf$
$Kopf \mapsto$ `Uta`
$Rumpf \mapsto$ `spielt Ball`
$Rumpf \mapsto$ `tritt den Ball`

Analyse:

$Kopf$ `tritt den Ball`

$Kopf\ Rumpf$

S

wohlgeformt!

Sprachspezifikation von JAVA

$S \mapsto A\ B\ C$
$S \mapsto \quad\ B\ C$
$S \mapsto D\ E\ F$

wird geschrieben:

$$\begin{array}{lll} S: & A_{opt} & B\ C \\ & D & E\ F \end{array}$$

Syntax einer Klassendeklaration in JAVA

$ClassDeclaration$:

$\quad Modifiers_{opt}$ `class` $Identifier\ Super_{opt}$

$\qquad\qquad Interfaces_{opt}\ ClassBody$

$Super$: `extends` $ClassType$

$Interfaces$: `implements` $InterfaceTypeList$

$InterfaceTypeList$:

$\quad InterfaceType$

$\quad InterfaceTypeList\ ,\ InterfaceType$

$ClassBody$:

$\quad \{\ ClassBodyDeclarations_{opt}\ \}$

$ClassBodyDeclarations$:

$\quad ClassBodyDeclaration$

$\quad ClassBodyDeclarations\ ClassBodyDeclaration$

$ClassBodyDeclaration$:

$\quad ClassMemberDeclaration$

$\quad StaticInitializer$

$\quad ConstructorDeclaration$

$ClassMemberDeclaration$:

$\quad FieldDeclaration$

$\quad MethodDeclaration$

$StaticInitializer$: `static` $Block$

Semantik einer Programmiersprache

Alle Folgen von terminalen Symbolen, die einem nicht-terminalen Symbol entsprechen, haben eine gemeinsame Bedeutung.

Nicht-terminale Symbole und Schlüsselwörter haben eine eindeutige Bedeutung.

Die *Semantik* einer Programmiersprache setzt aus der Bedeutung der einzelnen Teile die Bedeutung des Programms zusammen.

Die Bedeutung eines Programms ist *operational*, d.h. sie entspricht Operationen, die auch tatsächlich ausgeführt werden.

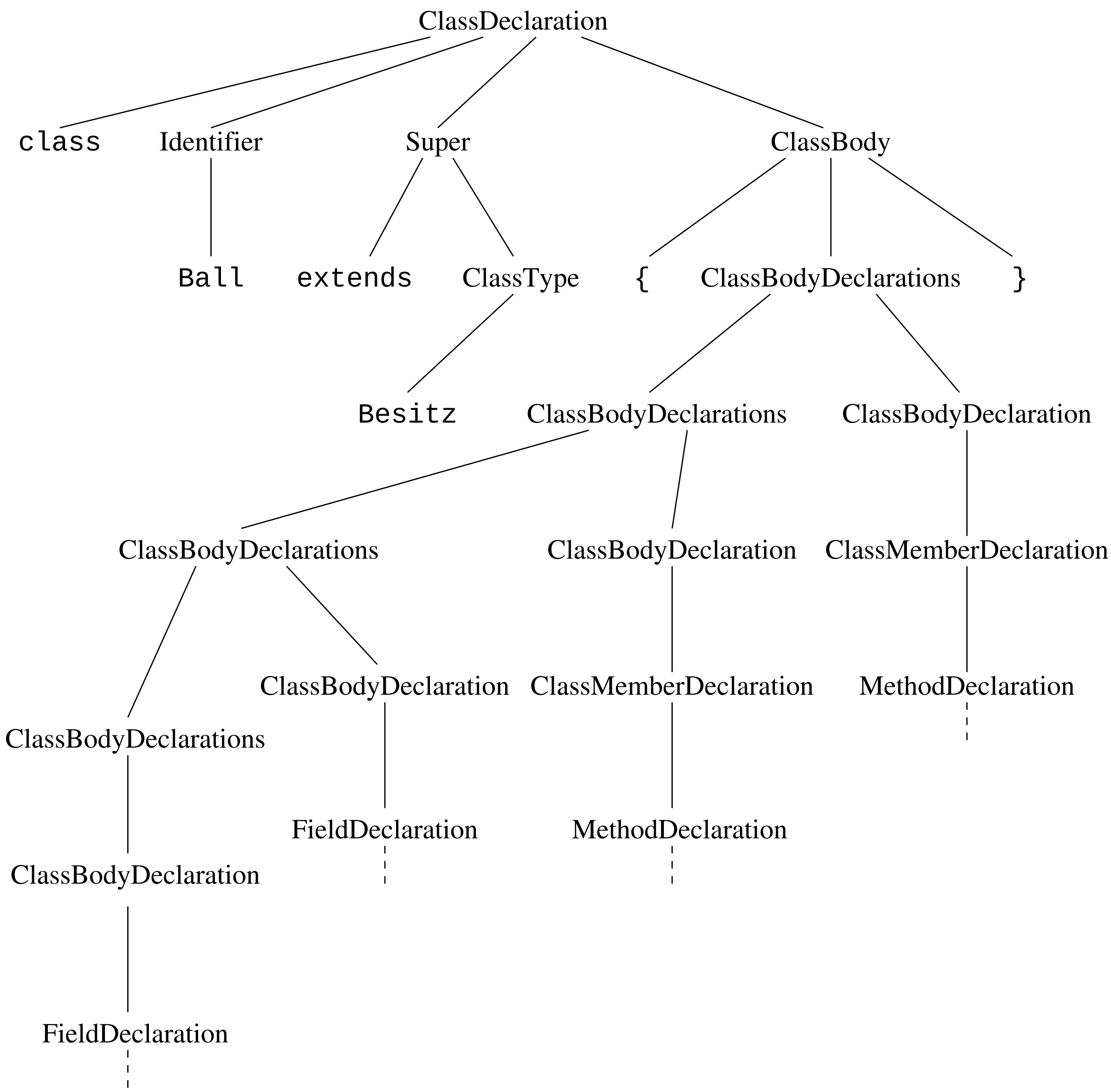
Klasse Ball

```
class Ball extends Besitz {   float x,y; String farbe;
    publicBall (String _name,String _farbe, float _x, float _y)
{   super (_name);
        farbe = _farbe;
        x = _x;
        y = _y;  }

    public void rolle (float dx,float dy) {
        x += dx;
        y += dy;
    }

}
```

Analyse des Übersetzers



Unsere 1. JAVA-Klasse

- die Vererbungshierarchie – $extends$,
- ihre Ausnutzung durch $super$,
- eine Eigenschaft, die jedes Objekt der Klasse hat – $farbe$,
- eine Methode, wie Objekte einer Klasse erzeugt werden (Konstruktor) – **Ball(String _name,String _farbe, float _x, float _y)**,
- eine Methode, die eine Botschaft von außen bearbeitet – **rolle(float dx, float dy)**

Konstruktor

$ConstructorDeclaration$:

$$Modifiers_{opt}\ ConstructorDeclarator\ Throws_{opt}\ ConstructorBody$$

$ConstructorDeclarator$:

$$SimpleName\ (\ FormalParameterList_{opt}\ )$$

$ConstructorBody$:

$$\{\ ExplicitConstructorInvocation_{opt}\ BlockStatements_{opt}\ \}$$

$ExplicitConstructorInvocation$:

$$\texttt{this}\ (\ ArgumentList_{opt}\ )\ ;$$

$$\texttt{super}\ (\ ArgumentList_{opt}\ )\ ;$$

Klasse Besitz

```
class Besitz {
    Mensch besitzer;
    String name;

    public Besitz (String _name) {
        name = _name;
    }
    public void gehoere (Mensch _besitzer) {
        besitzer = _besitzer; }
}
```

Unser 1. JAVA-Programm

```
class BallBeispiel {
    public static void main (String argv[]) {
        Ball ball;
        ball = new Ball ("ball1","blau",1,1);
    }
}
```