



## Die Syntax von Java

## Ursprünge

- Borland Software Corp
- 1995
- Syntax: Pascal
- Objektorientierte Programmierung optional
- Plattformen: Windows (Linux, Mac OS X)
- Sun Microsystems
- 1995
- Syntax: C/C++
- Objektorientiert von Grund auf!
- Plattformunabhängig (vom Mainframe bis zum Handy)

## Das Wichtigste in Kürze

- Objektorientiert von Anfang an
- Open Source: Compiler und Werkzeuge unter: <http://java.sun.com>
- Hybrid: Wird zwar kompiliert, das Binärformat (.class-Dateien) wird jedoch von einer virtuellen Maschine interpretiert.

## Konsequenzen

- Komfortables Programmieren:
  - ✓ Garbage Collection
  - ✓ Speicherschutz
  - ✓ KEINE Pointer-Arithmetik.

## Rund um Java

- Mächtige Bibliotheken verfügbar.
- Mächtige IDEs verfügbar:
  - Java-Editor
  - Eclipse
  - NetBeans
- Einfaches Arbeiten auf der Konsole möglich!

## Weiteres Vorgehen

- Vergleich der einzelnen Sprachkonstrukte zwischen Delphi und Java.
- Java und Dateien
- Benutzen der Java-Werkzeuge auf der Konsole

## Gleichheitszeichen



- Zuweisung:

`:=`

`=`

- Arithmetischer Vergleich:

`=`

`==`

7

## Blöcke und Kommentare



- Anweisungsblöcke:

**begin ... end**

`{ ... }`

- Kommentare:

// bis Zeilenende

// bis Zeilenende

`{ ... }`

`/* ... */`

`{* ... *}`

8

## Variablendeklaration



- Programmabschnitt: **var**
- Syntax: `index:integer`
- Überall im Programm möglich.
- Syntax: `int index;`
- Initialisierung mit Wert bei der Deklaration möglich:  
`int index = 0;`

9

## Gültigkeitsbereich

- **Beginn:** Zeile nach der Deklaration
- **Ende:** Ende des Blocks, in dem sie deklariert wurde

```
...  
if (a > 0) {  
  int b = 0;  
  while (a > 0) {  
    int c = 4;  
    b = b + c;  
    a -= 1; //oder a--  
  }  
}
```

...

## Bedingungen



- Konjunktion:

**and**

`&&`

- Disjunktion:

**or**

`||`

- Negation:

**not**

`!`

- „ungleich“:

`<>`

`!=`

11

## Bedingungen (Beispiel)



`(a<b) and (b<=c)`

`(a<b) && (b<=c)`

`(a=b) or not (b<>c)`

`(a==b) || !(b != c)`

12

## Entscheidungen



```
if (a=b) or not (b<>c) then begin
  a := c;
  ok := true;
end
else
  ok := false;

if ( (a==b) || !(b!=c) ) {
  a=c;
  ok=true;
} else
  ok=false;
```

13

## Schleifen (1)



```
• while:

while a < 100 do begin
  ...
end

while (a < 100) {
  ...
}
```

14

## Schleifen (2)



• while (fußgesteuert):

```
repeat
  ...
until a >= 100;

do {
  ...
} while (a < 100);
```

(es gibt keine echte repeat-until-Schleife in Java)

15

## Schleifen (3)



• for

```
for index:=low to high do
  ...
```

```
for (index = low; index <= high; index=index+1)
```

16

## Schleifen (4)



• Arrays:

```
var liste: array [1..5] of real; float liste = new float[5];
```

• Schleifen über Arrays:

```
for i in liste do ...; for (float i: liste) ...
```

17

## Verzweigungen



```
case OrdWert of
```

```
1: ...;
2: ...;
3: ...;
else ...;
end;
```

```
switch (OrdWert) {
```

```
case 1: ...; break;
case 2: ...; break;
case 3: ...; break;
default: ...;
}
```

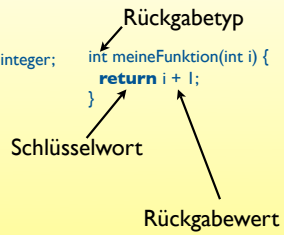
18

# Funktionen



## • Funktionen:

```
function meineFunktion(i: integer): integer;
begin
  result := i + 1;
end;
```



# Prozeduren



```
procedure tueEtwas(a: real);
begin
  ...
end;
```

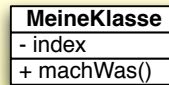
```
void tueEtwas(float a) {
  ...
}
```

Prozeduren und Funktionen werden in Java nur über den Rückgabotyp unterschieden

# Funktionen/Prozeduren

- Keine Aufteilung in Interface und Implementierung
- Können an beliebiger Stelle innerhalb einer Klassendefinition stehen.
- heißen in der Sprache der Objektorientierten Modellierung einheitlich ‚Methoden‘

# Klassen (1)



```
class MeineKlasse extends Oberklasse {
  private int index;

  public MyClass() {
    index = 0;
  }

  public void machWas() {
    index++;
  }
}
```

# Klassen (2)



```
type
  TMeineKlasse = class(Oberklasse)
  private
    Index: Integer;
  public
    constructor Create;
    procedure MachWas;
  end;

  constructor TMyClass.Create;
  begin
    Index:=0;
  end;

  procedure TMyClass.MachWas;
  begin
    Index:=Index+1;
  end;
```

```
class MeineKlasse extends Oberklasse {
  private int index;

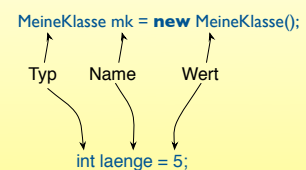
  public MyClass() {
    index = 0;
  }

  public void machWas() {
    index++;
  }
}
```

# Klassen (3)



Instanziierung einer Klasse:  
(Erzeugen einer neuen Objektvariablen)



## Konstruktoren



```
type ...  
private ...
```

```
public
```

```
constructor Create;  
end;
```

```
constructor TMyClass.Create;
```

```
begin  
  Index:=0;  
end;
```

```
class MyClass extends Oberklasse {
```

```
...
```

```
public MyClass() {  
  index = 0;  
}
```

```
...
```

```
}
```

- Name der Klasse
- Kein Rückgabetyt

25

## Hauptprogramm

- Es gibt kein separates Hauptprogramm
- Spezielle Methode einer (beliebigen) Klasse dient als Hauptprogramm
- **public static** void main(String argv[])

## Hauptprogramm



```
program name;
```

```
const ...
```

```
type ...
```

```
var ...
```

```
begin
```

```
...  
end.
```

```
class Name {
```

```
...
```

```
public static void main(String[] argv) {  
  ...  
}
```

Das „Hauptprogramm“ ist nichts weiter als eine spezielle Methode einer Klasse!

27

## Module



```
unit name;
```

```
interface
```

```
uses ...
```

```
type ...
```

```
var ...
```

```
implementation
```

```
uses ...
```

```
type ...
```

```
var ...
```

```
initialization ...
```

```
finalization ...
```

```
end
```

```
package name;
```

```
import ...
```

```
class ...
```

```
...
```

28

## Package-Namen (2)

- Konvention: Umgedrehte URLs
- Beispiel:
  - org.mysql.drivers
  - com.sap.bapi.connector
  - de.gymherm.bankautomat

## Fragen?