

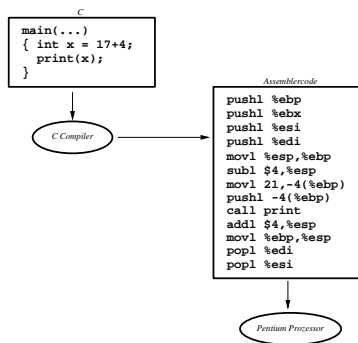
# 13 Java

- Objekt-orientierte Sprache  
Objekte, Methoden, Klassen, Vererbung
- Kern: Imperative Programmierung
- Beeinflusst von C++
- Traditionelles Programmiermodell:
  - Erstellen eines Programms mit Editor
  - Übersetzung des Programms (Compiler)
  - Ausführung des Programms

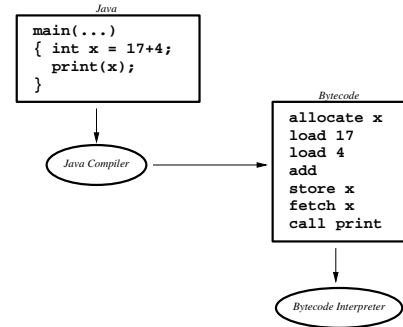
## Im Vergleich zu Scheme

- Keine selbständigen Funktionen (ausser vordefinierten: +, -, ...), alles muss durch Methoden geschehen
- Keine Funktionen als Parameter oder Ergebnisse, aber es können Objekte übergeben werden
- Vorgegebenes Typsystem
- Typdeklarationen sind Teil des Programms und werden vom Compiler überprüft

## Das Compiler Modell



## Das Java Modell



Java-Bytecode läuft auf *virtueller Maschine*, der JVM

- Unverändert lauffähig auf jeder Maschine, die die JVM implementiert (*Bytecode-Interpreter*)
- Bytecode-Compiler übersetzt Bytecode → Maschinencode
- JIT-Compiler (*just-in-time*) übersetzt verzahnt mit der Ausführung

## Konventionen

- Anwendung (*application*): eigenständiges Java-Programm
- Start der Anwendung:
  - Erzeuge Anwendungsobjekt
  - Rufe Methode `main` auf

## 13.1 Aufgabe: Anwendung Hello

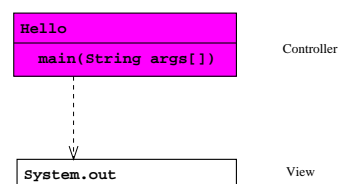
Schreibe auf den Bildschirm:

Hello world!

49

Welche Objekte sind daran beteiligt?

## Klassendiagramm für Hello



## Java Programm für Hello

```
/** Hello prints two lines on the console */
public class Hello // Neue Klasse
{ public static void main (String[] args) // Klassenmethode
  { System.out.println ("Hello world!");
    System.out.println (49);
  }
}
```

- Inhalt der Datei Hello.java; kompiliert nach Hello.class
- `void` Ergebnistyp und `String[]` Argumenttyp von `main`
- `System.out` vordefiniertes Ausgabeobjekt
- `println` Methode des Ausgabeobjekts

```
> java Hello
Hello, world!
49
```

## 13.2 Aufgabe: Erzeugen eines Objekts

```
import java.util.*;
/** NameAndDate prints my name and the exact date and time. */
public class NameAndDate
{ public static void main(String[] args)
  { System.out.print("Fred Mertz --- "); // a partial line
    System.out.println( new GregorianCalendar().getTime() );
    System.out.println(); // a blank line
    System.out.println("Finished.");
  }
}
```

```
Fred Mertz --- Wed Oct 20 11:56:07 CEST 1999

Finished.
```

### 13.2.1 Packages

Java besitzt umfangreiche Klassenbibliothek(en)

Java API (*Application Programming Interface*)

organisiert in *Packages* (Mengen von Klassen)

Package wird eingebunden durch

```
import <package name>.<class name>;
```

<class name> = \* : alle Klassen aus <package name>

Packagedokumentation: Webseiten

### Beispiele

- `java.lang` (enthält z.B. `System.out`, immer verfügbar)
- `java.util` (enthält z.B. `GregorianCalendar`, muß explizit importiert werden)
- `java.awt` Bibliothek für Benutzerschnittstellen

### 13.2.2 Erzeugung von Objekten

```
new <class name> (<expr-list>);
```

- Erzeugt ein neues Objekt der Klasse <class name>
- Ruft dessen Konstrukturfunktion mit den Werten der aktuellen Parameter <expr-list> auf
- Ergebnis: (Verweis auf) das neue Objekt

## 13.3 Syntax

- Bezeichner (*identifier*): Folge von Buchstaben, Ziffern und `_`, die nicht mit einer Zahl beginnt (im Sinne von Unicode)  
`BezeichnerFürEineKlasse`, `EINE_KONSTANTE`,  
`rufeDieseMethode2mal`, `das_geht_auch`  
Konvention:
  - Bezeichner für Klassen beginnen mit Grossbuchstaben,
  - Bezeichner für Konstanten: nur Grossbuchstaben,
  - Bezeichner für Methoden beginnen mit Kleinbuchstaben (möglichst mit einem Verb)
  - Bezeichner für Variablen beginnen mit Kleinbuchstaben
  - mehrere Worte durch Kapitalisierung

- Schlüsselworte: reservierte Bezeichner

`public`, `class`, `static`, `void`, ...

- Zahlen: Folge von Ziffern, ggf. mit `+` oder `-` davor
- Spezielle Zeichen: `(`, `)`, `{`, `}` usw.

- *white space*: Folge von Leerzeichen, Tabulatoren, Zeilenvorschüben, usw.

Trennt Bezeichner, wird ignoriert.

Java ist *formatfrei*, d.h. die grafische Anordnung eines Programms spielt keine Rolle.

- Kommentare

- `//` bis Zeilenende
- Text zwischen `/*` und `*/`

### Kontextfreie Syntax, vorläufiger Ausschnitt

```
<class> ::= public class <class name> { <components> }
<components> ::= <method> <components> | ε
<method> ::= public static void
           main (String[] args) { <statements> }
<statements> ::= <statement> ; <statements> | ε
<statement> ::= <expr>
<expr> ::= <identifier>
          | <expr> . <method name> ( <expr-list> )
          | new <class name> ( <expr-list> )
<expr-list> ::= <expr-list-1> | ε
<expr-list-1> ::= <expr>
                | <expr> , <expr-list-1>
```

## 13.4 Ausdrücke, Variable, Zuweisungen

### 13.4.1 Aufgabe: Temperaturkonversion Celsius → Fahrenheit

Umrechnungsformel:  $t_{\text{Fahrenheit}} = \frac{9}{5}t_{\text{Celsius}} + 32$

Eingabe: durch Konstante im Programm (zunächst)

```
/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 * output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit
{ public static void main(String[] args)
  { int c;
    c = 20;
    double f = ((9.0/5.0) * c) + 32;
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}
```

- `int c` ist (*statement*), das eine Variable `c` vom Typ `int` deklariert.
- Der Datentyp `int` besteht aus dem Intervall  $[-2^{31}, 2^{31} - 1] \subseteq \mathbb{Z}$ , sowie den arithmetischen Operationen  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$ ,  $\dots$ . Jedes Element von `int` wird durch ein 32-Bit-Wort dargestellt.
- Die arithmetischen Operationen werden in Infix-Schreibweise benutzt, dabei gelten die üblichen Prioritätsregeln. Klammern sind erlaubt und erwünscht.

Vorläufige Version

50

© 2001 Peter Thiemann

Vorläufige Version

51

© 2001 Peter Thiemann

```
/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 * output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit
{ public static void main(String[] args)
  { int c;
    c = 20;
    double f = ((9.0/5.0) * c) + 32;
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}
```

- `c = 20` ist *Zuweisungsoperation* (ein (*statement*)).
- Überschreibt den alten Inhalt der Variable `c` mit 20.
- (*statement*) ::= (*identifizier*) = (*expr*).
- Der Typ der rechten Seite ((*expr*)) muss ein Subtyp des Typs von (*identifizier*) sein.
- Kombination mit Deklaration möglich: `int c = 20`.

Vorläufige Version

52

© 2001 Peter Thiemann

```
/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 * output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit
{ public static void main(String[] args)
  { int c = 20;
    double f = ((9.0/5.0) * c) + 32;
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}
```

- Der Datentyp `double` besteht aus Gleitkommazahlen der Form  $s \cdot m \cdot 2^e$  mit  $s \in \{+1, -1\}$  (*Vorzeichen*),  $m \in \{1 + i/2^{52} \mid 0 \leq i < 2^{52}\}$  (*Mantisse*) und  $e \in [-1022, 1023]$  (*Exponent*), sowie den üblichen arithmetischen Operationen  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\dots$  (*überladen!*). Jedes Element wird durch ein 64-Bit-Wort dargestellt.
- `int` ist Subtyp von `double`, d.h. wo immer ein `double` erwartet wird, kann auch ein `int` übergeben werden. Dabei wird automatisch konvertiert.
- 9.0 ist Konstante vom Typ `double`, 32 ist Konstante vom Typ `int`.

Vorläufige Version

53

© 2001 Peter Thiemann

```
/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 * output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit
{ public static void main(String[] args)
  { int c = 20;
    double f = ((9.0/5.0) * c) + 32;
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}
```

- `"For Celsius degrees "` ist Konstante vom Typ `String` (Zeichenketten).
- `String` ist vordefinierte Klasse `java.lang.String`.
- Der Operator `+` ist *überladen*. Falls eins seiner Argumente ein `String` ist, so berechnet er die String-Verkettung seiner beiden Argumente. Argumente von anderem Typ werden automatisch in `Strings` umgewandelt.
- Jeder `String s` ist ein Objekt und versteht Methodenaufrufe, zum Beispiel `s.length()`, `s.equals(s1)`, `s.substring(i, j)`. So ist `"degrees".length() = 7`.

Vorläufige Version

54

© 2001 Peter Thiemann

### 13.4.2 Aufg.: Temperaturkonversion mit Parameter

```
/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 * output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit
{ public static void main(String[] args)
  { int c = new Integer (args[0]).intValue ();
    double f = ((9.0/5.0) * c) + 32;
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}
```

- `String[] args` deklariert den formalen Parameter `args` der Methode `main` als Variable vom Typ `String []`.
- Der Typ `String []` ist der Typ der Reihenungen (Arrays, Vektoren) mit Elementen vom Typ `String`.
- Beim Start der Applikation erhält `args` als Wert eine Reihung mit den Argumenten des Programmaufrufs (als `Strings`).

Vorläufige Version

55

© 2001 Peter Thiemann

```
/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 * output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit
{ public static void main(String[] args)
  { int c = new Integer (args[0]).intValue ();
    double f = ((9.0/5.0) * c) + 32;
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}
```

- Die Reihung `args` ist ein Objekt. Die Länge der Reihung (Anzahl der Elemente) ist `args.length`. Die Elemente der Reihung sind `args[0], ..., args[args.length-1]`.
- Vor Zugriff auf `args[i]` wird getestet, ob  $i \geq 0$  und  $i < args.length$ . Falls dieser Test fehlschlägt wird ein Fehler signalisiert.
- Falls `i` den Typ `int` hat, so hat `args[i]` den Typ `String`.

Vorläufige Version

56

© 2001 Peter Thiemann

```
/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 * output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit
{ public static void main(String[] args)
  { int c = new Integer (args[0]).intValue ();
    double f = ((9.0/5.0) * c) + 32;
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}
```

- Die Typen `int` und `double` (wie auch `byte`, `short`, `long`, `char`, `float` und `boolean`) sind *primitive Datentypen*. Ihre Werte sind keine Objekte.
- `byte < short < int < long < float < double`, sowie `char < int`.
- Der Typ (Klasse) `Integer` ist vordefinierter Objekttyp. Objekte der Klasse `Integer` haben genau ein Attribut vom Typ `int`.
- Wenn der Konstruktor `new Integer (args[0])` mit einem Argument vom Typ `String` aufgerufen wird, so wird das Argument in ein `int` umgewandelt und als Attributwert gespeichert.

Vorläufige Version

57

© 2001 Peter Thiemann

```

/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 * output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit
{ public static void main(String[] args)
  { int c = new Integer (args[0]).intValue ();
    double f = ((9.0/5.0) * c) + 32;
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}

```

- Die Methode `intValue` der Klasse `Integer` nimmt keine Parameter und liefert den Wert des Attributs.
- Analog zu `Integer` gibt es Klassen `Boolean`, `Byte`, `Character`, `Double`, `Float`, `Long` und `Short` mit ähnlicher Funktionalität.

```

/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 * output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit
{ public static void main(String[] args)
  { int c = new Integer (args[0]).intValue ();
    double f = ((9.0/5.0) * c) + 32;
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}

```

```

> java CelsiusToFahrenheit 20
For Celsius degrees 20,
Degrees Fahrenheit = 68.0

```

```

> java CelsiusToFahrenheit 100
For Celsius degrees 100,
Degrees Fahrenheit = 212.0

```

### 13.4.3 Variablen für Objekte

Wann liest `GregorianCalendar` die Uhr?

```

public class Mystery
{ public static void main (String[] args)
  { GregorianCalendar now = new GregorianCalendar ();
    System.out.println ("Now is " + now.getTime ());
    GregorianCalendar later = new GregorianCalendar ();
    System.out.println ("Later is " + later.getTime ());
    System.out.println ("later than " + now.getTime ());
  }
}

```

```

> java Mystery
Now is Thu Jan 11 19:22:00 CET 2001
Later is Thu Jan 11 19:22:01 CET 2001
later than Thu Jan 11 19:22:00 CET 2001

```

### 13.4.4 Gültigkeitsbereich einer Variable

```

{
  ...
  int i ← ... ;
  ... i ...
  { ... }
  ... i ...
}

```

Statement *nach* Deklaration/Initialisierung bis zur nächsten schliessenden Klammer

```

{
  int i = ... ;
  ... i ...
  { ... }
  int i = ...
}

```

Verbotene Redeklaration: Compiler meldet Fehler

### Löcher im Gültigkeitsbereich

```

{
  int i ← ... ;
  ... i ...
  { int i ← ...
    ... i ...
  }
  ... i ...
}

```

### 13.4.5 Ergänzungen zur Syntax

```

<statement> ::= ...
| <type> <identifier>
| <type> <identifier> = <expr>
| <identifier> = <expr>

<expr> ::= ...
| <constant>
| <unary operator> <expr>
| <expr> <binary operator> <expr>
| ( <expr> )

```