

2. Compilation, Interpretation, virtuelle Rechner

Java-Beispiel:
Hello.java

Schwerpunkte

- Abarbeitung von Java-Programmen
- Virtuelle Rechner
- Getrennte Compilation

Abarbeitung eines Java-Programms

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

Hello.java

Schritte:

- Abspeichern des Java-Programms im File
(Name der Klasse = Name des Files)
`Hello.java`
- Aufruf des Java-Compilers:
`javac Hello.java`
→ Es entsteht das Objektfile ("Maschinenprogramm")
`Hello.class`
- Abarbeitung durch Aufruf des Java-Interpreters:
`java Hello`

In Eclipse ebenso

Vergleich

Java - Programm

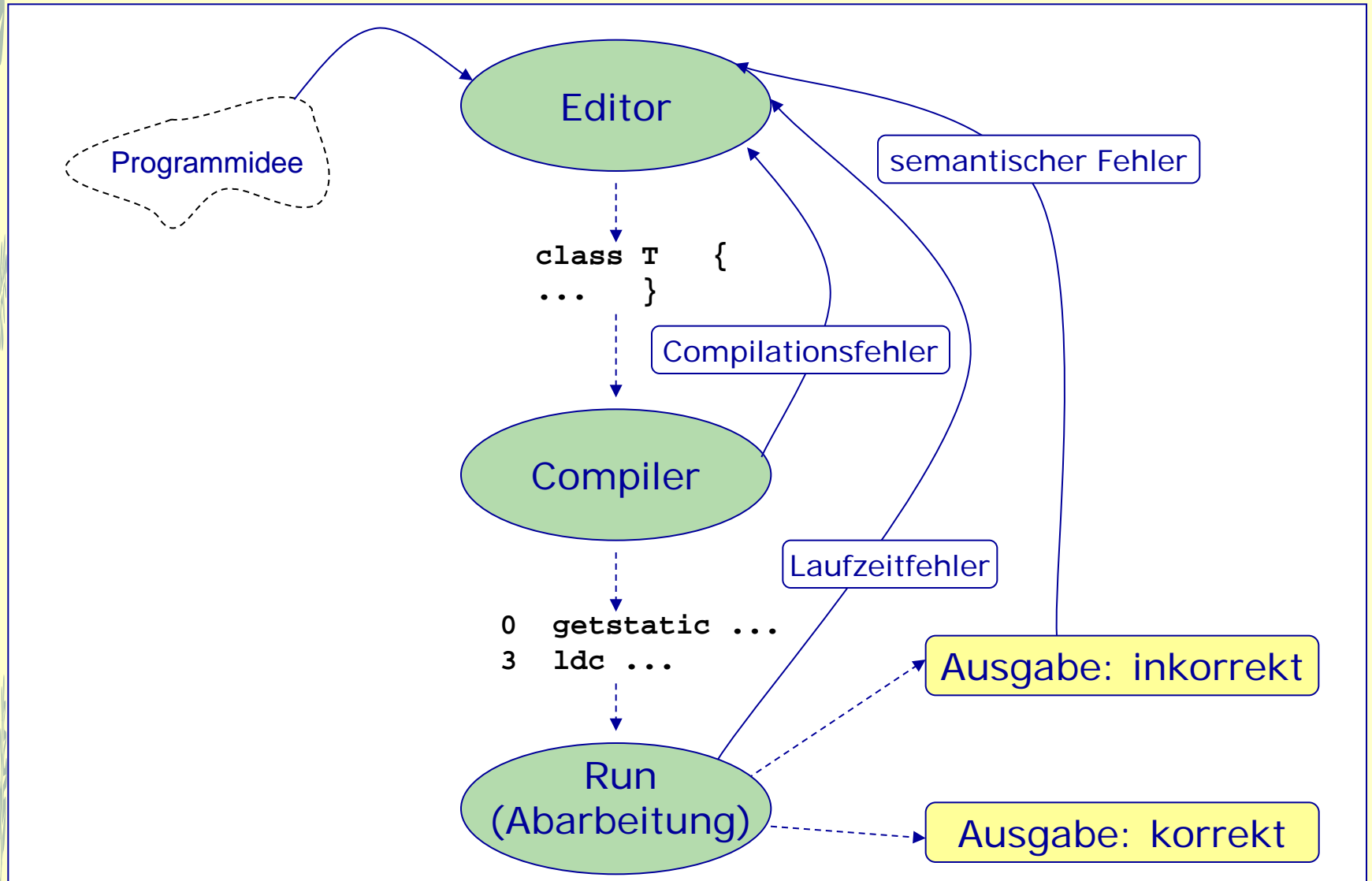
```
class Hello {  
  
    public static void main (String[] args) {  
  
        System.out.println("Hello!");  
    }  
}
```

Pascal - Programm

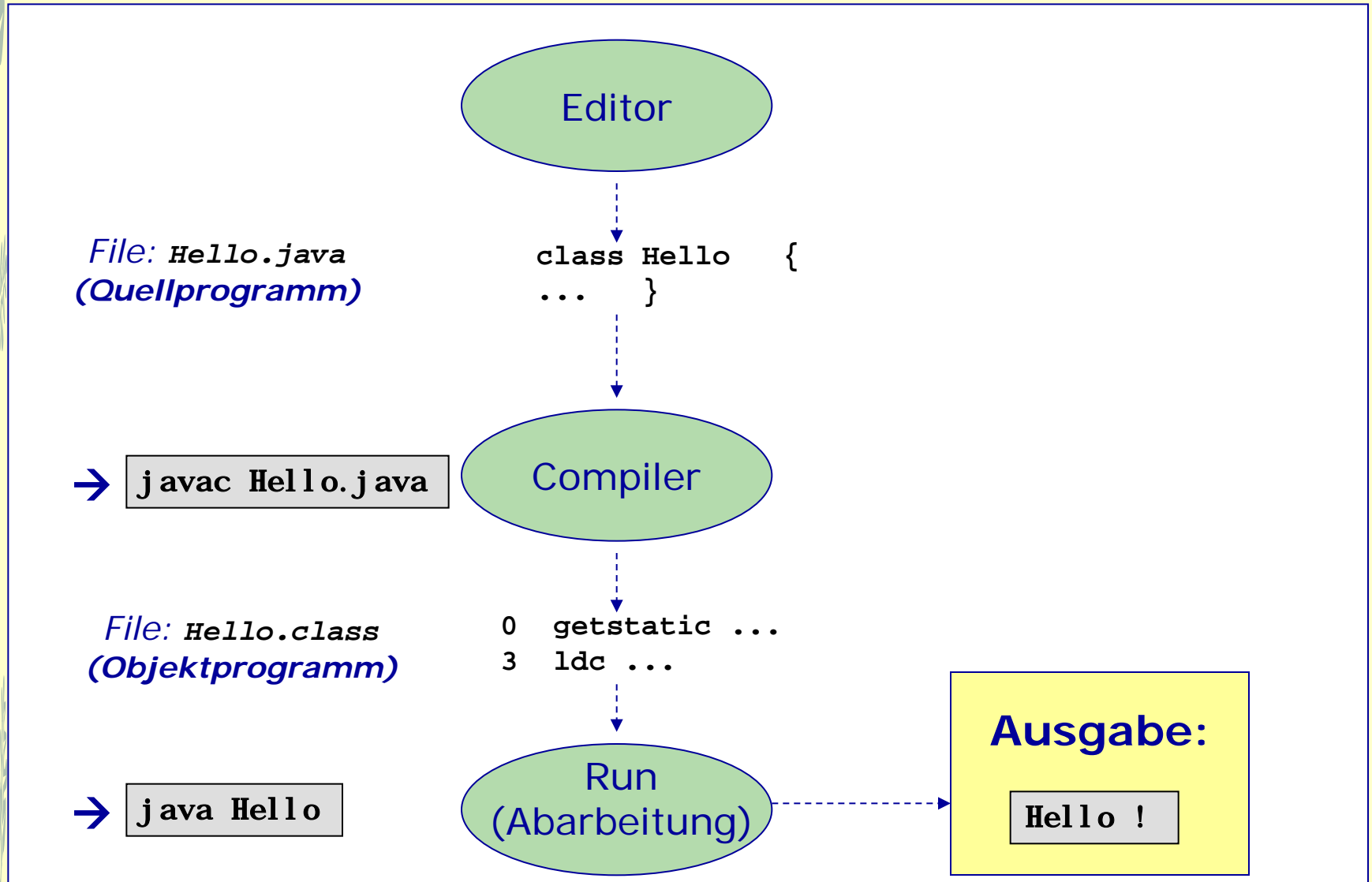
```
PROGRAM Hello;  
  
BEGIN  
  
    writeln("Hello!");  
  
END.
```

Pascal als
Einstiegssprache
besser geeignet

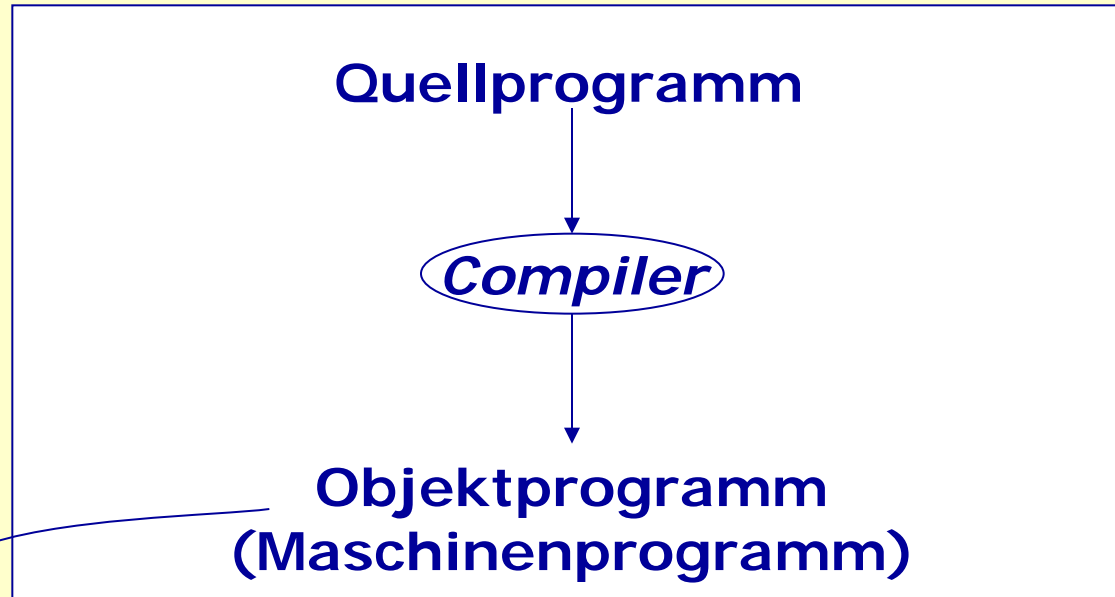
Programmentwicklung: Edit-, Compile-, Run / Debug-Zyklus



Edit, Compile, Run von Java-Programmen



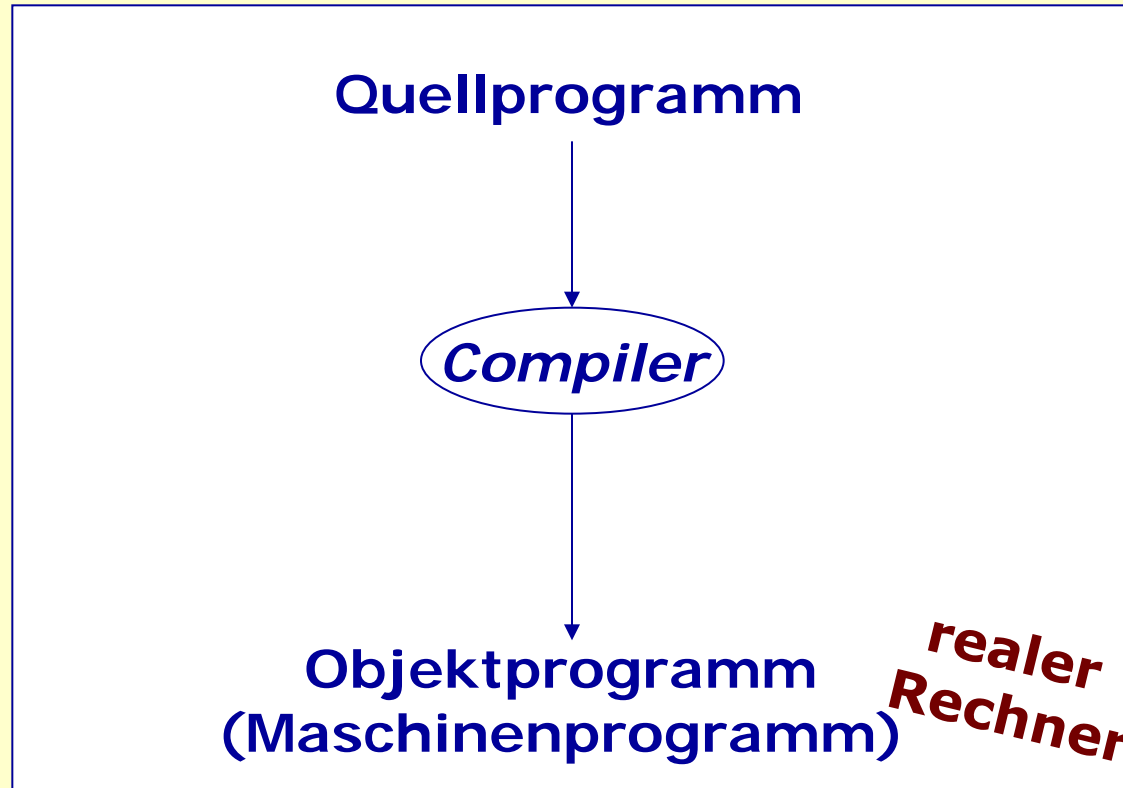
Compilation



- Maschinencode eines realen Rechners
- Maschinencode eines theoretischen Rechners (virtueller Code für virtuellen Rechner)

Virtueller Code - was soll das?

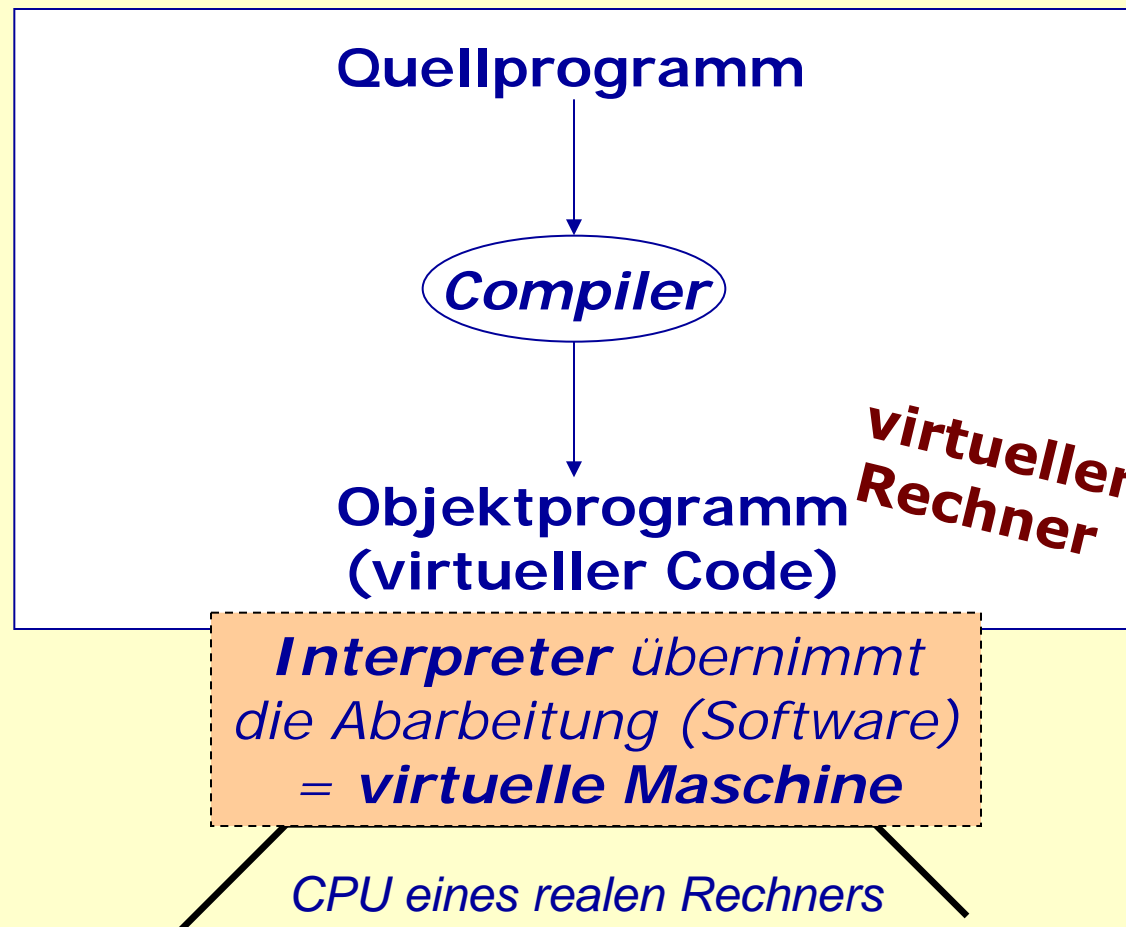
Compilation: realer Rechner



*CPU des Rechners übernimmt
die Abarbeitung (Hardware)*

z.B. C, C++, Pascal, Fortran ...
Vorteil: effizient (schnell)

Compilation: virtueller Rechner



z.B. Java (JVM: Java virtuelle Maschine), Pascal, Modula 2

Vorteil: Programme portabel, Compiler portabel

(ein virtueller Code für alle Rechner → ein Compiler, ein Interpreter)

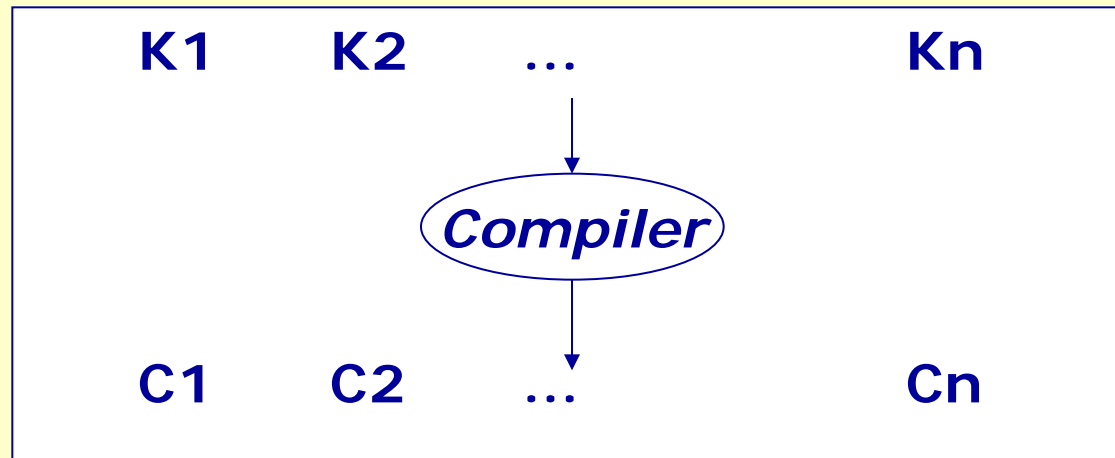
Getrennte Compilation

(unabhängige / separate Kompilation)

Quellprogramm:

Menge von Komponenten
(Übersetzungseinheiten)

**Wichtig:
bei komplexen
Programmen**



Objektprogramm:

Menge von Objektfiles

***Komponenten K_i : Klasse mit Namen 'name'
→ im File mit Namen 'name.java'***