

# Praktische Informatik 1

Imperative Programmierung und Objektorientierung

Karsten Hölscher und Jan Peleska

Wintersemester 2011/2012  
Session 2



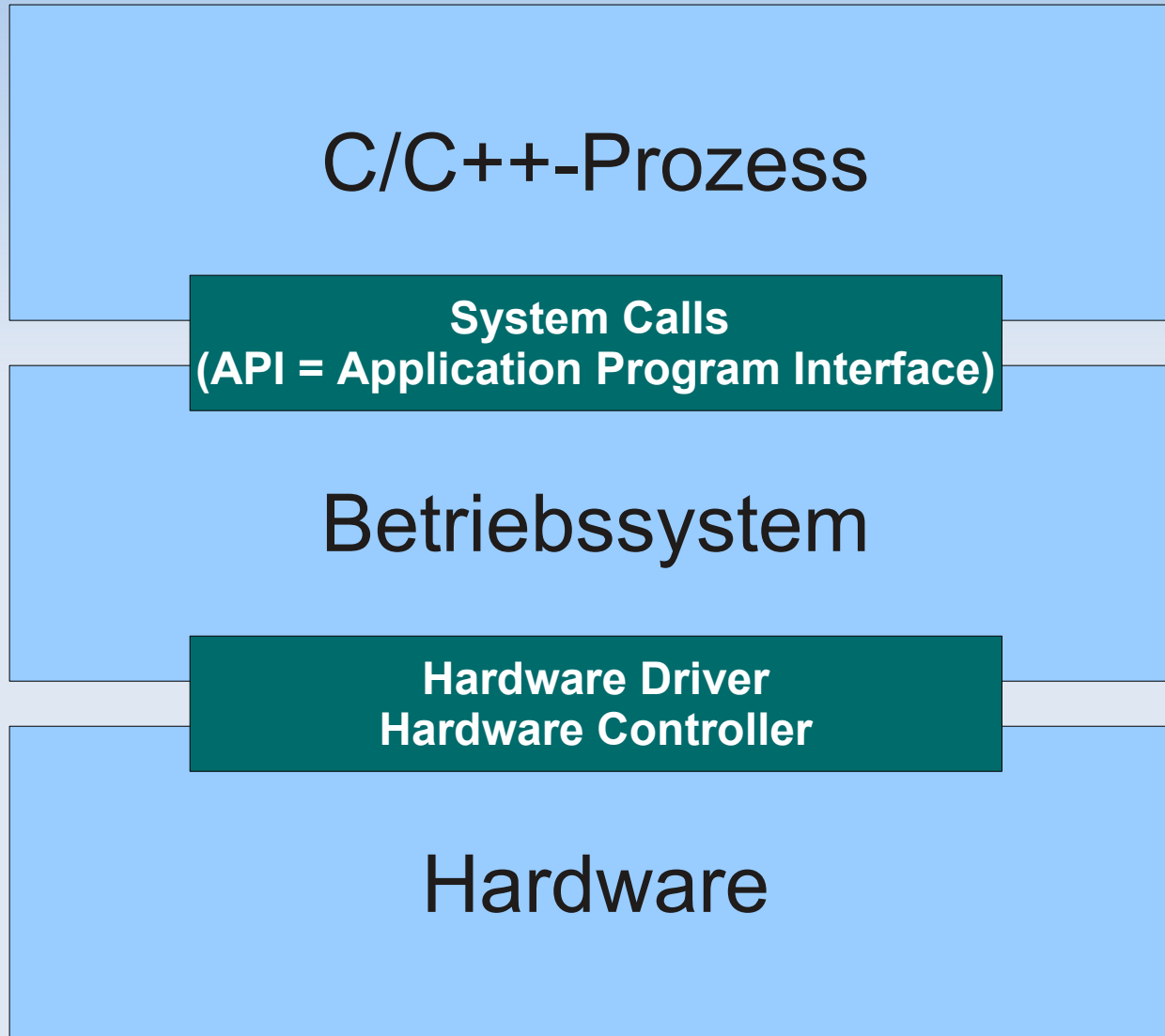
# Programmierung – Begriffe

- C/C++ Compiler: übersetzt Quellcode in Assemblercode
- Assembler: übersetzt das Assemblercode in Maschinencode
- Linker/Loader: bindet Maschinencode-Einheiten zu einem Programm zusammen
- Betriebssystem: ermöglicht die Ausführung des Programms als Prozess, und stellt diesem Betriebsmittel zur Verfügung (CPU, Speicher, Zugriff auf Schnittstellen)
- Weitere Details hierzu: TI 2, Betriebssysteme I

# Programmierung – Begriffe

- Java Compiler: übersetzt Quellcode in Java Byte Code
- Java Laufzeitumgebung:
  - interpretiert den Byte Code und transformiert ihn in Maschinencode
  - führt den Maschinencode mit Hilfe des Betriebssystems auf der Hardware aus
  - → Java Programme sind unabhängig vom Betriebssystem
  - → Java Programme sind unabhängig von der darunter liegenden HW-Plattform

# Programmierung – Begriffe



# Programmierung – Begriffe

Java Programm (Byte Code)

**Java Instruktionen**

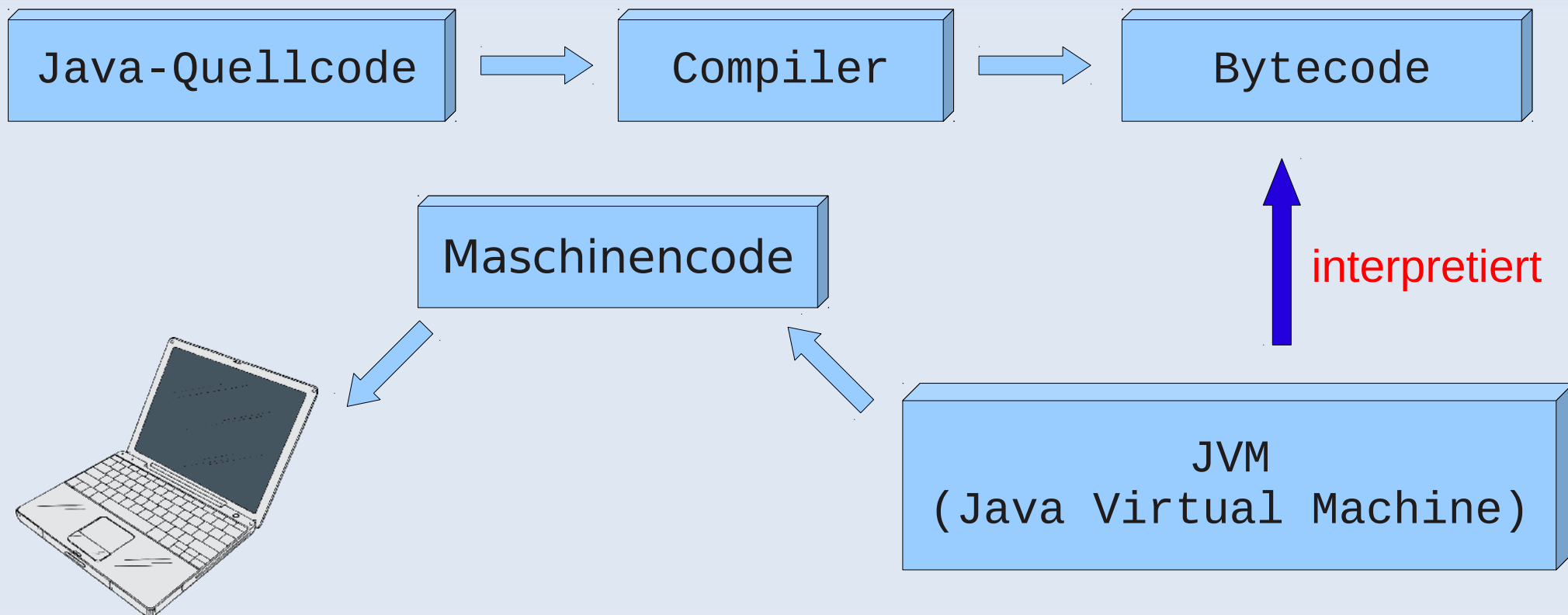
Java Laufzeitsystem

**System Calls**

Betriebssystem

# Übersetzung

- Computer verstehen kein Java!
- genauer:



# Programmierung in JAVA

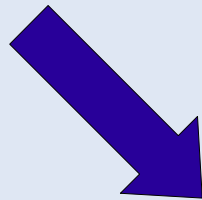
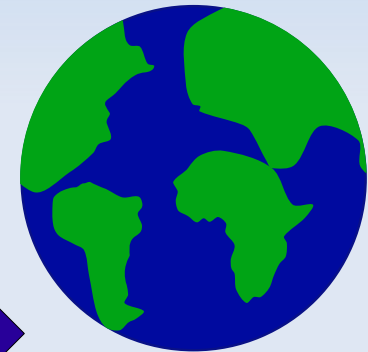
- Im ersten Semester gilt: keine Ablenkung durch *Fancy Tools*, daher
- Editor – z. B. Emacs, Vi, Xcode Editor
  - Editieren des Java Quellcodes
- Unix Shell – Cygwin Shell unter Windows
  - Compilieren und Ausführen des Codes
- Anbindung an SVN-Repository
  - Zusammenarbeit in Gruppen
  - Abgabe des Programmcodes
- Zu Beginn des 2. Semesters: Einführung in die Eclipse IDE

# Klassen und Objekte

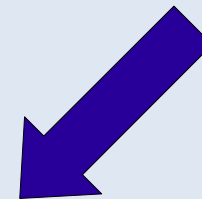


Computerprogramm:

Modelliert einen  
Ausschnitt der realen Welt



```
01011101010010
10001010110101
01010010101111
01010010010100
01101010010101
```



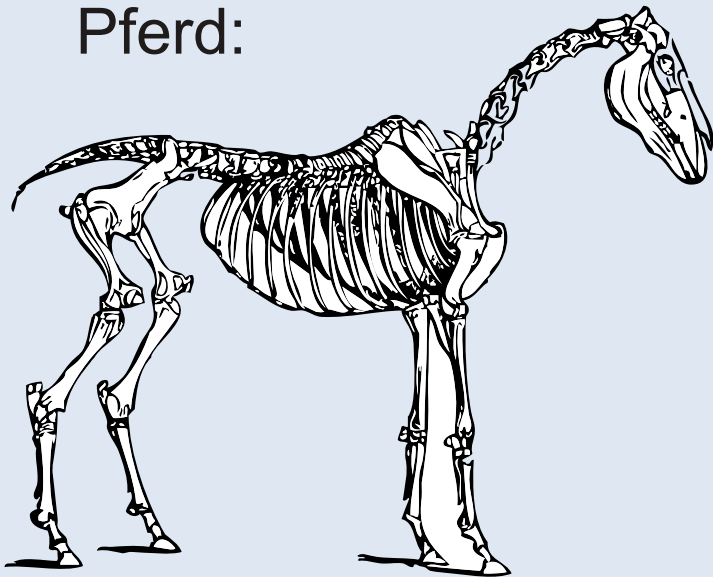


# Klassen und Objekte

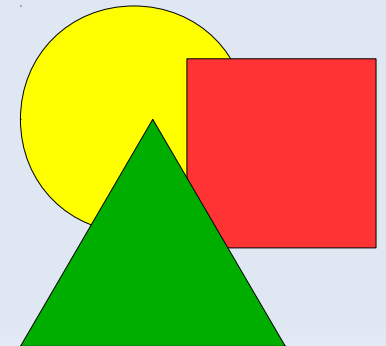
- **Objekte:** “Dinge” – sog. **Instanzen** – der realen Welt

bzw. eines Problembereichs  
...oder dieser luftgekühlte Käfer:

z.B. dieses  
Pferd:



...oder diese geometrischen Figuren:



# Klassen und Objekte

- **Klassen:** alle Objekte einer Art bzw. Kategorie

**Objekt :** *konkretes* Pferd



Landina

Zwei Instanzen  
der Klasse "Pferd"



Pitu

**Klasse Pferd:**



**Eine Klasse:**  
**mehrere** Instanzen

# Attribute

- ein Objekt hat **Attribute**  
d.h. *Werte*, die in *Datenfeldern* gespeichert sind



Attributname	Wert
Name	Landina
Rasse	Hannoveraner
Alter	10
Stockmaß	1.78



Attributname	Wert
Name	Pitu
Rasse	Oldenburger
Alter	12
Stockmaß	1.74

# Attribute

- die **Klasse** definiert, welche Attribute die Objekte haben:

Pferd
Name
Rasse
Alter
Stockmaß

# Zustand

- jedes Objekt speichert seine *eigenen* Werte (seinen **Zustand**):

Attributname	Wert
Name	Pony
Rasse	Shettlandpony
Alter	15
Stockmaß	0.96



# Attribute

- Attribute haben
  - einen *Namen*
  - einen *Typ*
  - einen *Zugriffsmodifikator*

private String farbe

```
graph TD; A[einen Namen] --- B(farbe); C[einen Typ] --- D(String); E[einen Zugriffsmodifikator] --- F(private);
```

# Datentypen

- wichtige sogenannte **primitive** Datentypen

- **int**

- ganze Zahlen

- `11`

- `-2913`

- `42`

- **String**

- beliebige Zeichenketten

- `"hello world"`

- `"PI-1 ist erhellend"`

- **boolean**

- Wahrheitswerte

- `true`

- `false`

# Datentypen

- wichtige sogenannte **primitive** Datentypen

- **float**

Gleitkommazahlen

11.1

-2913E-12

- Weitere primitive Datentypen

- byte
- short
- long
- double
- char



# Klassen und Objekte – etwas formaler

- Klassen sind Typen
- Typen sind Mengen, repräsentieren also die Menge aller Elemente einer “bestimmten Sorte”
- Klassen sind Mengen, die Objekte repräsentieren
- Alle Objekte, die zu einer Klasse gehören, besitzen die selben Attribute (dieses Konzept wird später im Zusammenhang mit *Vererbung* noch verfeinert ...)

# Klassen und Objekte – etwas formaler

- Ein **Objektzustand** wird durch den aktuellen Wert aller seiner Attribute identifiziert.
- Besitzt eine Klasse nur Attribute aus primitiven Datentypen,

$t_1 a_1; t_2 a_2; \dots ; t_n a_n$

dann ist der aktuelle Zustand eines Objektes dieser Klasse ein Element der Menge

$t_1 \times t_2 \times \dots \times t_n$

# Klassen und Objekte – etwas formaler

- Zwei Objekte können den selben Zustand haben – kann man sie dann noch unterscheiden ?
- Ja → die Java Laufzeitumgebung identifiziert jedes Objekt über eine **Referenz**, d.h., eine virtuelle Speicheradresse
- Die Identifikation ist eine injektive Abbildung

$$r : \text{Objects} \rightarrow \text{References}$$
$$\forall o_1, o_2 \in \text{Objects} : r(o_1) = r(o_2) \Rightarrow o_1 = o_2$$

# Erzeugung von Objekten

- Objekte werden aus ihrer Klasse durch Anwendung eines **Konstruktors** erzeugt
- Der Konstruktor übergibt der Java Laufzeitumgebung Befehle zum Anlegen eines neuen Objektes gemäß Klassentyp
- Die Laufzeitumgebung allokiert den hierzu notwendigen Speicher und speichert dort das Objekt
- Jede Klasse besitzt einen **Default Constructor**, der alle Attribute mit ihren Default-Werten belegt

# Erzeugung von Objekten

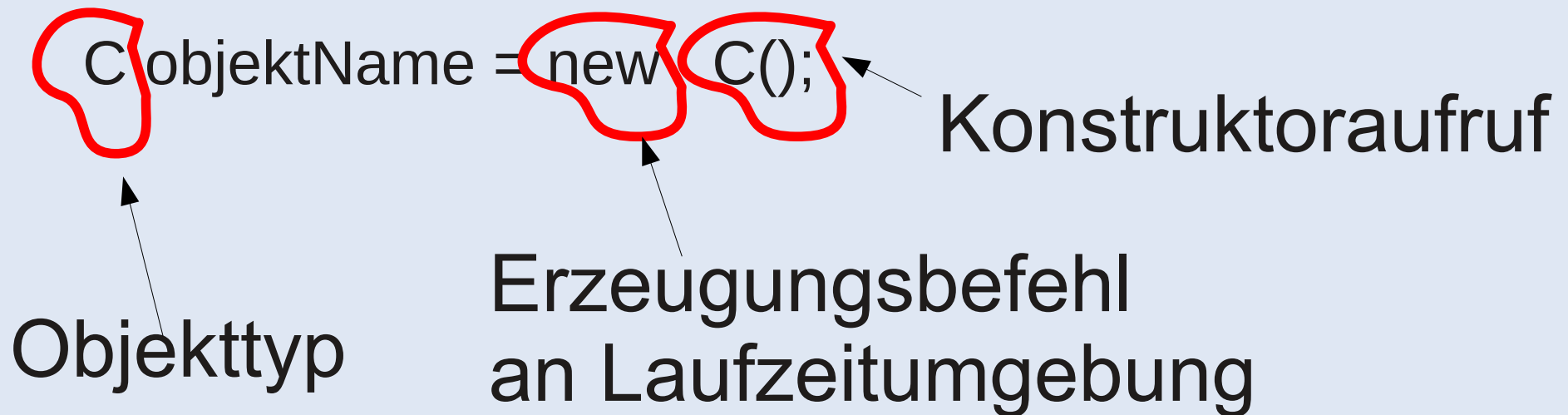
- Konstruktoren haben immer den selben Namen wie ihre Klasse
- Die Objekterzeugung unter Nutzung des Default Constructors zu einer Klasse C erfolgt durch den Befehl

`C` `objektName` = `new` `C()`;

Objektyp

Erzeugungsbefehl  
an Laufzeitumgebung

Konstruktoraufruf

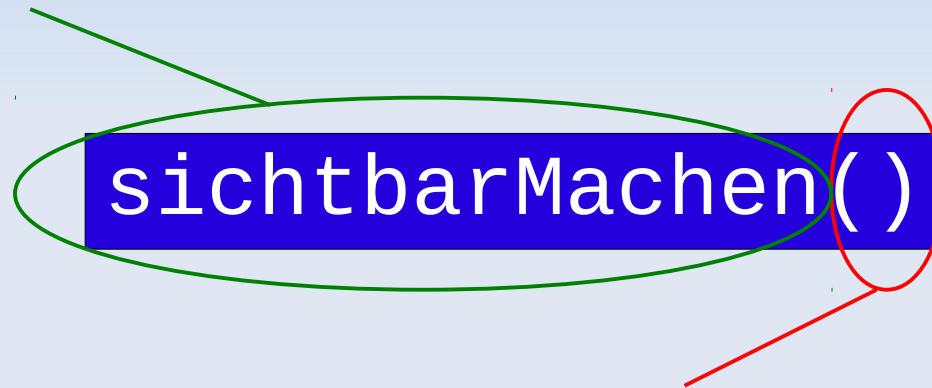
The diagram shows the code snippet 'C objektName = new C();' with three red hand-drawn circles highlighting the parts 'C', 'new', and 'C()'. An arrow points from the text 'Objektyp' to the first 'C'. Another arrow points from the text 'Erzeugungsbefehl an Laufzeitumgebung' to the 'new' keyword. A third arrow points from the text 'Konstruktoraufruf' to the 'C()' part of the code.

# Kommunikation mit Objekten

- über *Methoden* kann mit Objekten kommuniziert werden
- Methoden sind *Operationen*, die aufgerufen werden können

# Methoden

- Methoden haben
  - einen *Namen*



- gefolgt von optionalen *Parametern*

# Parameter

- Parameter haben
  - einen *Typ*

farbeAendern(**String** **neueFarbe**)



- einen *Namen*



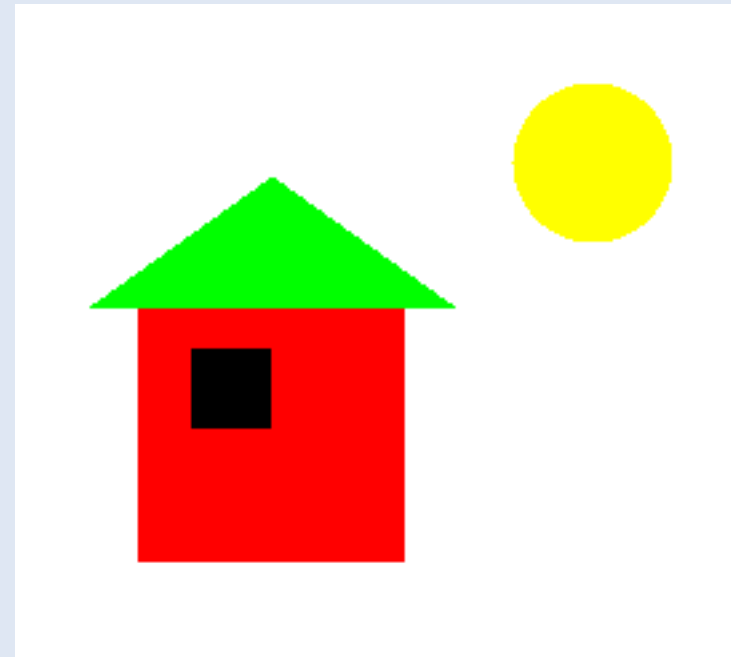
# Parameterliste

- mehrere Parameter möglich:

```
moveTo(int xkoor, int ykoor, int zkoor)
```

# Objektinteraktion

- **Objekte** können
  - andere Objekte *erzeugen*
  - *Methoden* dieser anderen Objekte *aufrufen*



# Ergebnis

- Methoden können ein Ergebnis zurückgeben!
- **Typ** des *Ergebnisses / Rückgabewertes*



```
String gibName()
```

# Signatur

- *Signatur* einer Methode:
  - Kopf der Methode
  - nötige Informationen für den Aufruf
  - allgemein:

```
Rückgabotyp Methodename(Parameter)  
mit Parameter = (P1, P2, P3, ...)  
und Pn = Parametertyp Parametername
```

Ergebnislose Methoden haben den Rückgabotyp: **void!**

Parameter sind optional! Klammern aber *zwingend* nötig!

# Objekte als Parameter

- Name der entsprechenden **Klasse** als Typ:

```
void trageStudentEin(Student neuerStudent)
```

