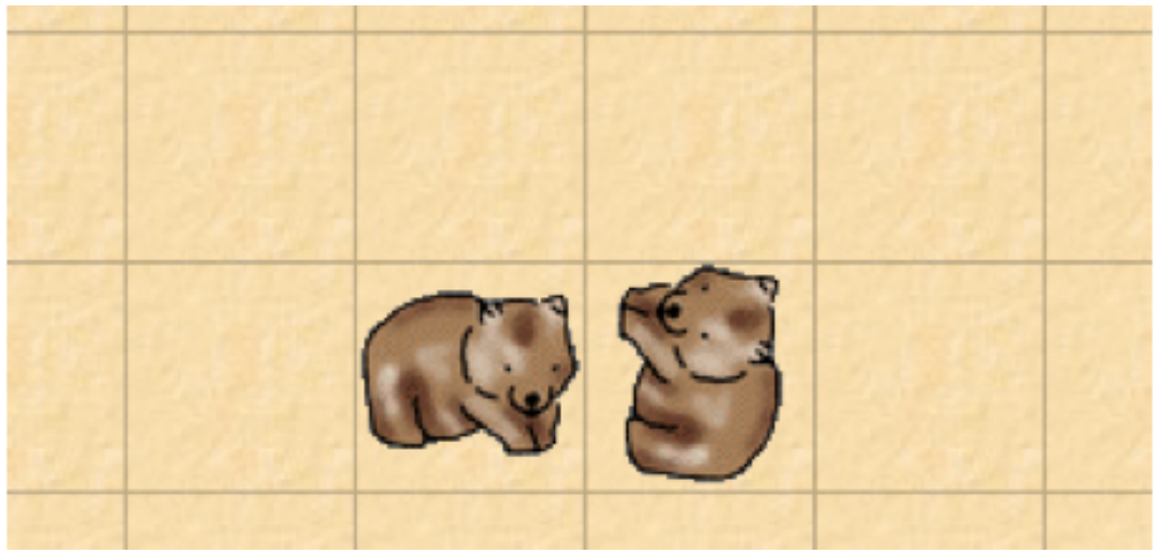
Aufgaben

Objektorientierte Programmierung

1. Baue in die Klasse Wombat eine Methode turnRight() ein, damit sich der Wombat auch im Uhrzeigersinn drehen kann!



2. Notiere alle Befehle (Stift und Blatt Papier oder Methode act() in Klasse Wombat), die bewirken, dass der Wombat alle Blätter nacheinander isst!

UML - Unified Modelling Language

einfach

wie bei Greenfoot

gut für Übersichten

UML - Unified Modelling Language

mittel

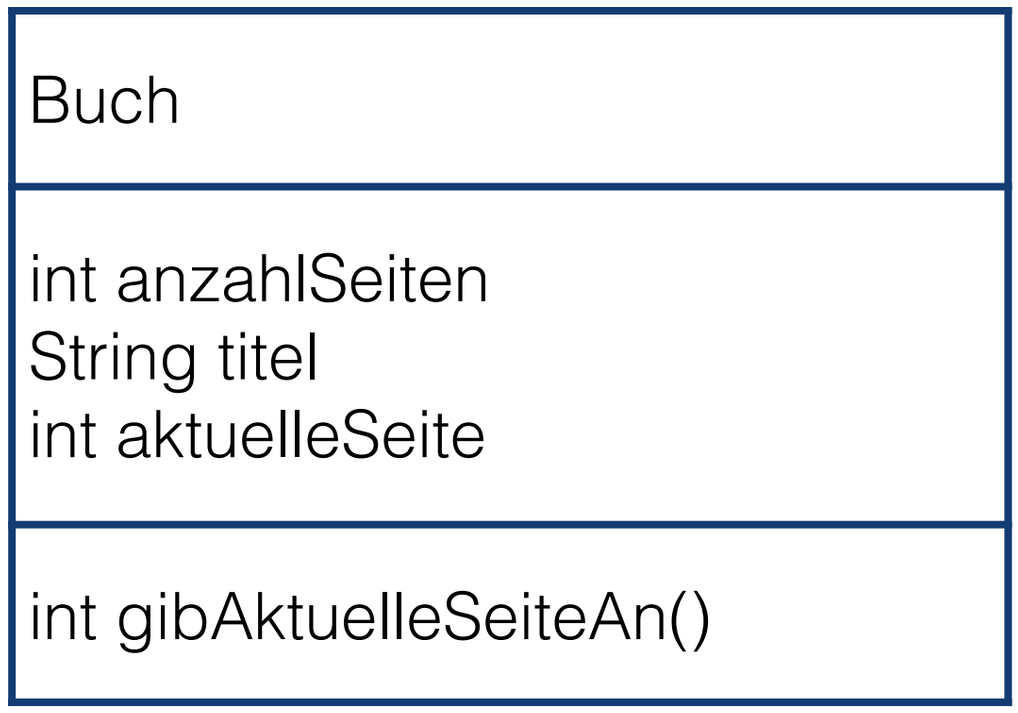
Variablen

Methoden

UML - Unified Modelling Language

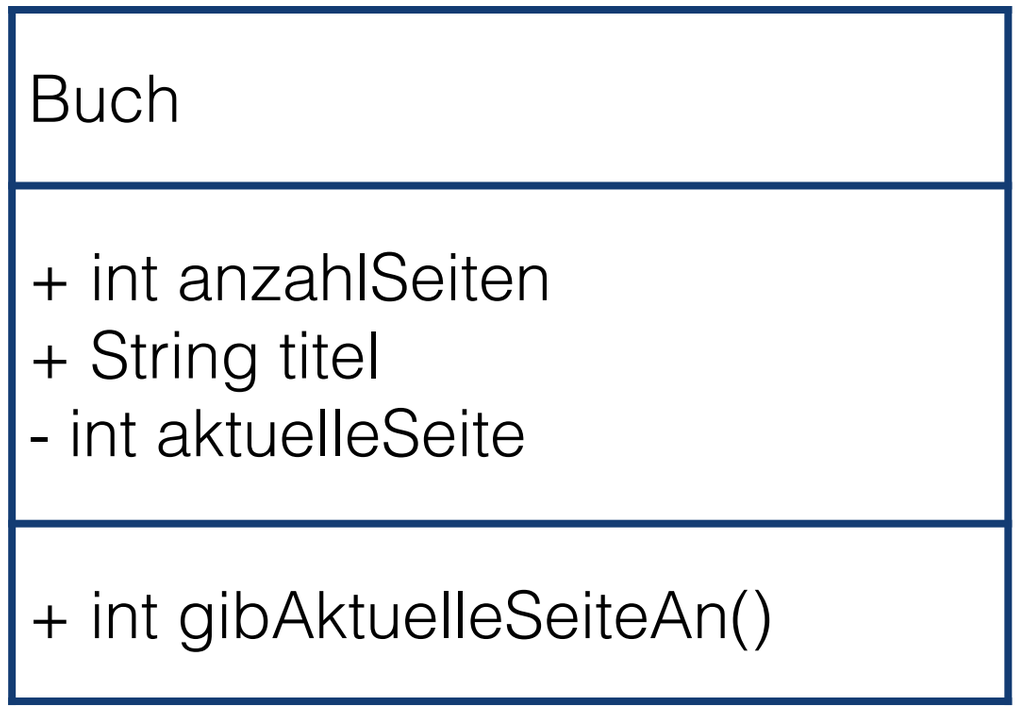
komplex

Typen



UML - Unified Modelling Language

Sichtbarkeit



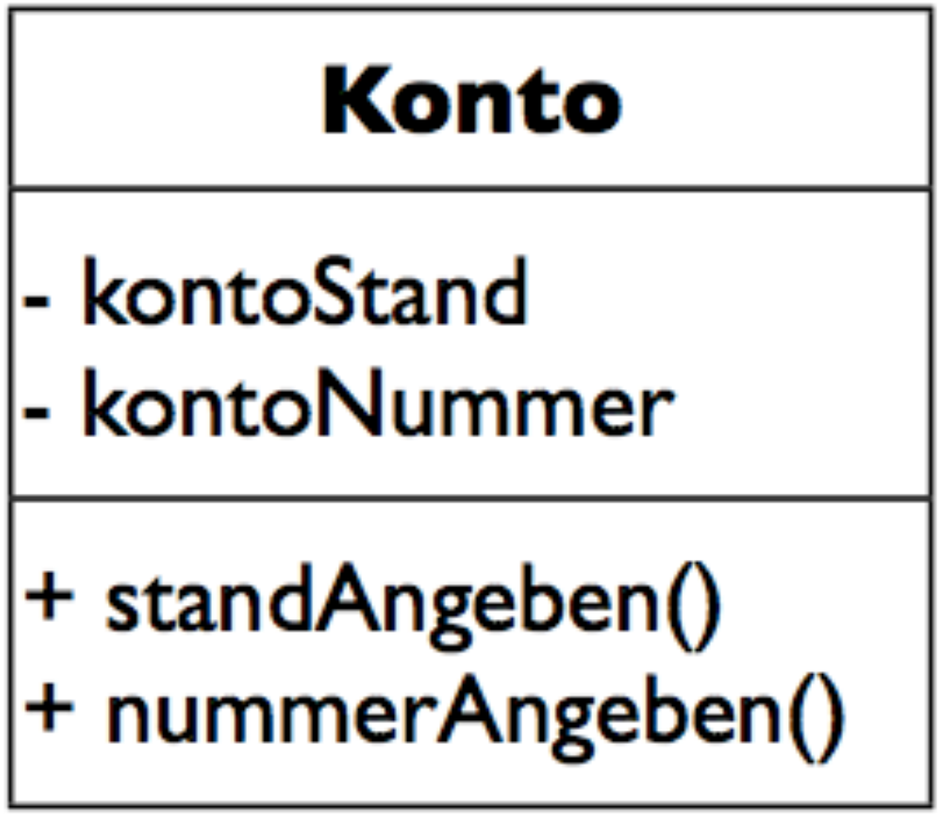
Sichtbarkeit

	Klasse	Paket-klasse	Unter-klasse	andere Klassen
public	+	+	+	+
protected	#	#	#	
package	~	~		
private	-			

3 In einer Obstgärtnerei werden auf Karteikarten zu den Apfelbäumen die Maximalhöhe, die Blütedauer und der Wasserbedarf pro Woche notiert.
Des weiteren interessiert den Betrieb, dass der Baum zuerst blühen und dann Obst tragen kann.

Erstelle ein Klassendiagramm der Klasse Apfelbaum!

4 Gegeben ist das nebenstehende UML-Diagramm. Erkläre anhand dieses Diagramms die Begriffe Klasse, Objekt,Attribut und Methode mit eigenen Worten! Grenze insbesondere die Begriffe Klasse und Objekt voneinander ab!



5 Gegeben sind die Klassen Personenfahrzeug (PKW), Lastfahrzeug (LKW), Landfahrzeug, Kraftfahrzeug und Motorrad.

a) Entwickle ein geeignetes Klassendiagramm, das diese Struktur unter Zuhilfenahme der Vererbung abbildet!

b) Erweitern Sie die Vererbungshierarchie um Wasser- und Luftfahrzeuge mit diversen Unterklassen!

c) Erläutern Sie anhand des Beispiels aus Teilaufgabe a) und b) die Spezialisierung und deren Vorteile!

6. Zeichne das Klassendiagramm der Klasse Wombat! (mittlere Detailstufe)

7. Beschreibe umgangssprachlich, was der Wombat kann (Methoden) und welche Informationen seine Attribute beinhalten!

8. Gib an welche Attribute sich durch Anwendung welcher Methoden ändern!

9. Erstelle das Klassendiagramm der Klasse Leaf!

11. Beschreibe, was dir im Vergleich zur Wombatklasse auffällt!

12. Ermittle die Anzahl der Zustände, in denen sich ein Wombat in der gegebenen Wombatwelt befinden kann!
Die Zustandsermittlung soll sich dabei auf die x- und y-Koordinate sowie auf die Drehung beziehen.

13. Erläutere die Beziehung zwischen Klasse, Objekt und Identität!

14. Nimm Stellung zur These, dass Objekte unterschiedlicher Klassen nicht den gleichen Zustand besitzen können!

Vorteile des objektorientierten Paradigmas

Wiederverwertbarkeit

Eine Klasse Wombats ermöglicht **beliebig viele Objekte** dieses Typs.

Vorteile des objektorientierten Paradigmas

Aufteilung in überschaubare Einzelteile

Das Szenario (die **Software**) Wombat besteht aus den **Klassen** World, WombatWorld, Actor, Wombat und Leaf.

Vorteile des objektorientierten Paradigmas

Erweiterungsmöglichkeit

Wombat **erbt** von Actor = Wombat erweitert Actor

Actor Leaf **erbt** von Actor = Leaf erweitert Actor

WombatWorld **erbt** von bzw. erweitert World

einfache Möglichkeit für eigene Erweiterungen

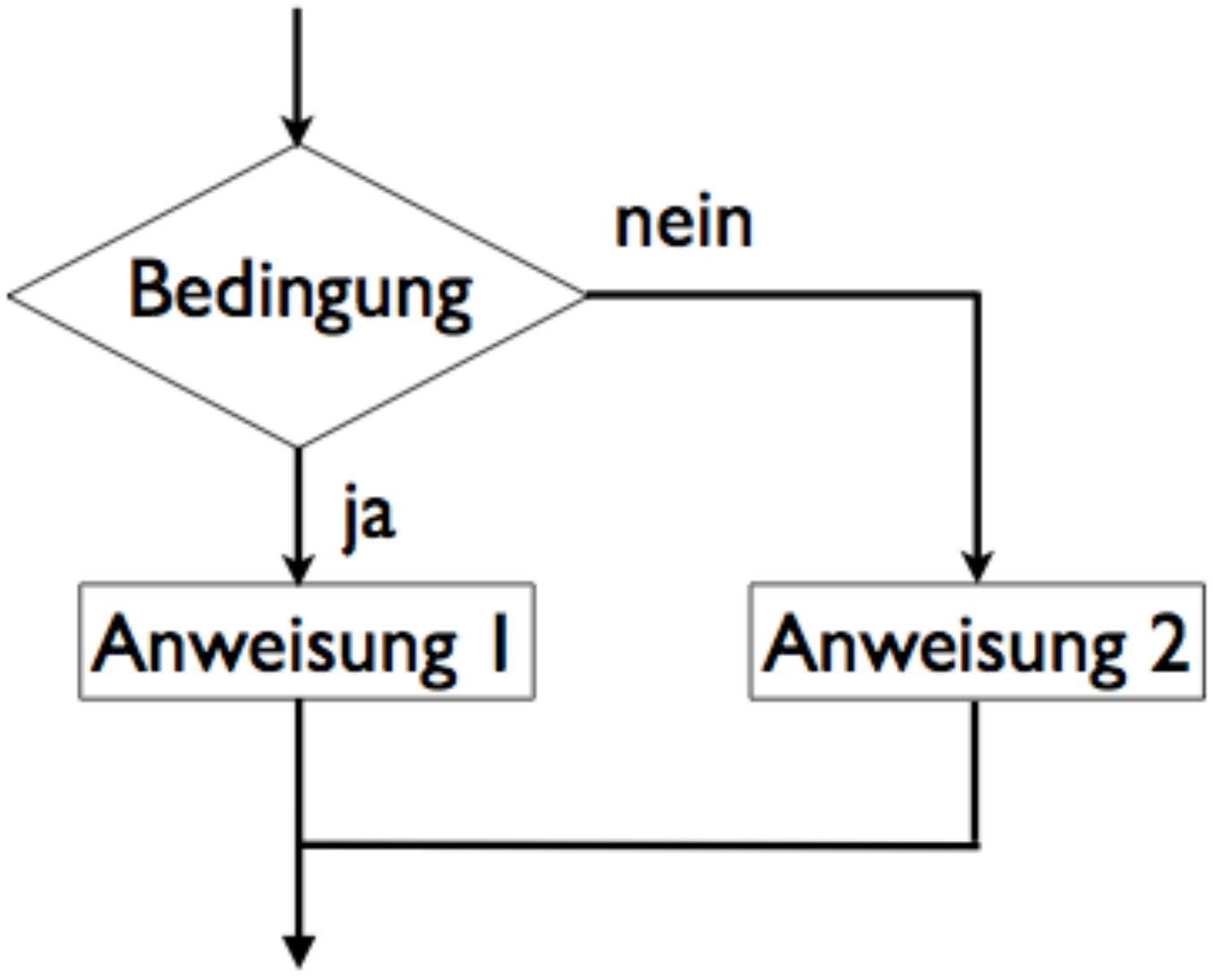
Vorteile des objektorientierten Paradigmas

Sicherheit durch Geheimnisprinzip

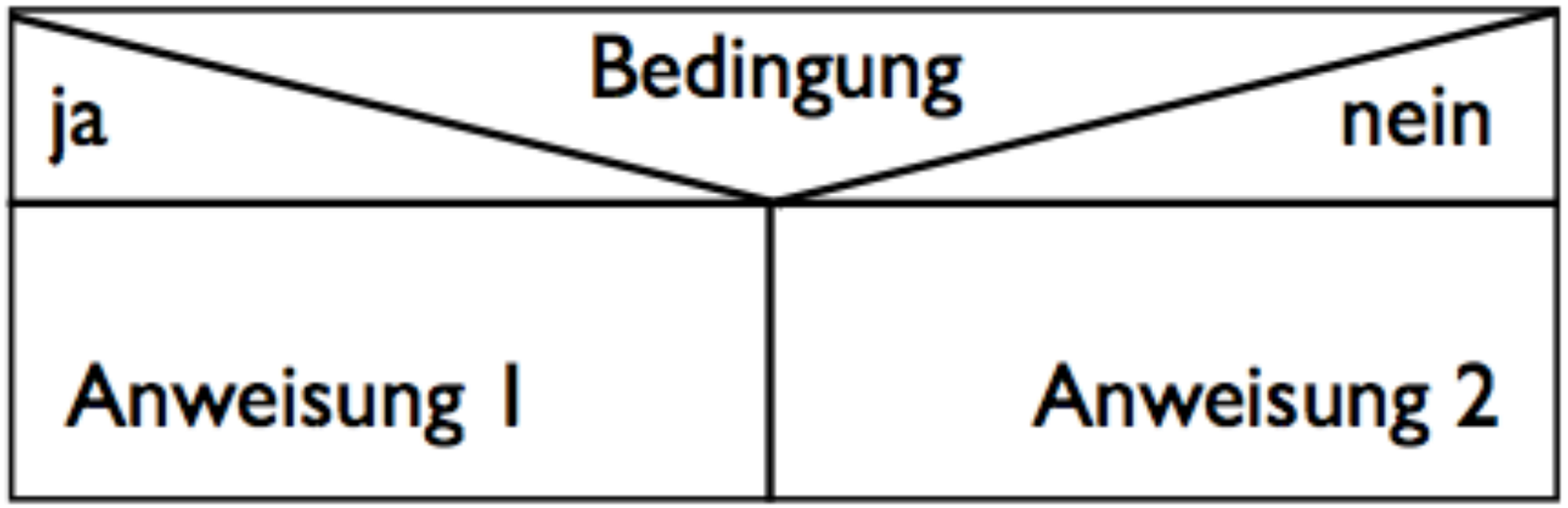
Erweiterbarkeit ist auch möglich, wenn nur die übersetzte Datei **(*.class), der Bytecode,**

weiter gegeben wird.

Bedingte Anweisung - zweiseitig



Bedingte Anweisung - zweiseitig



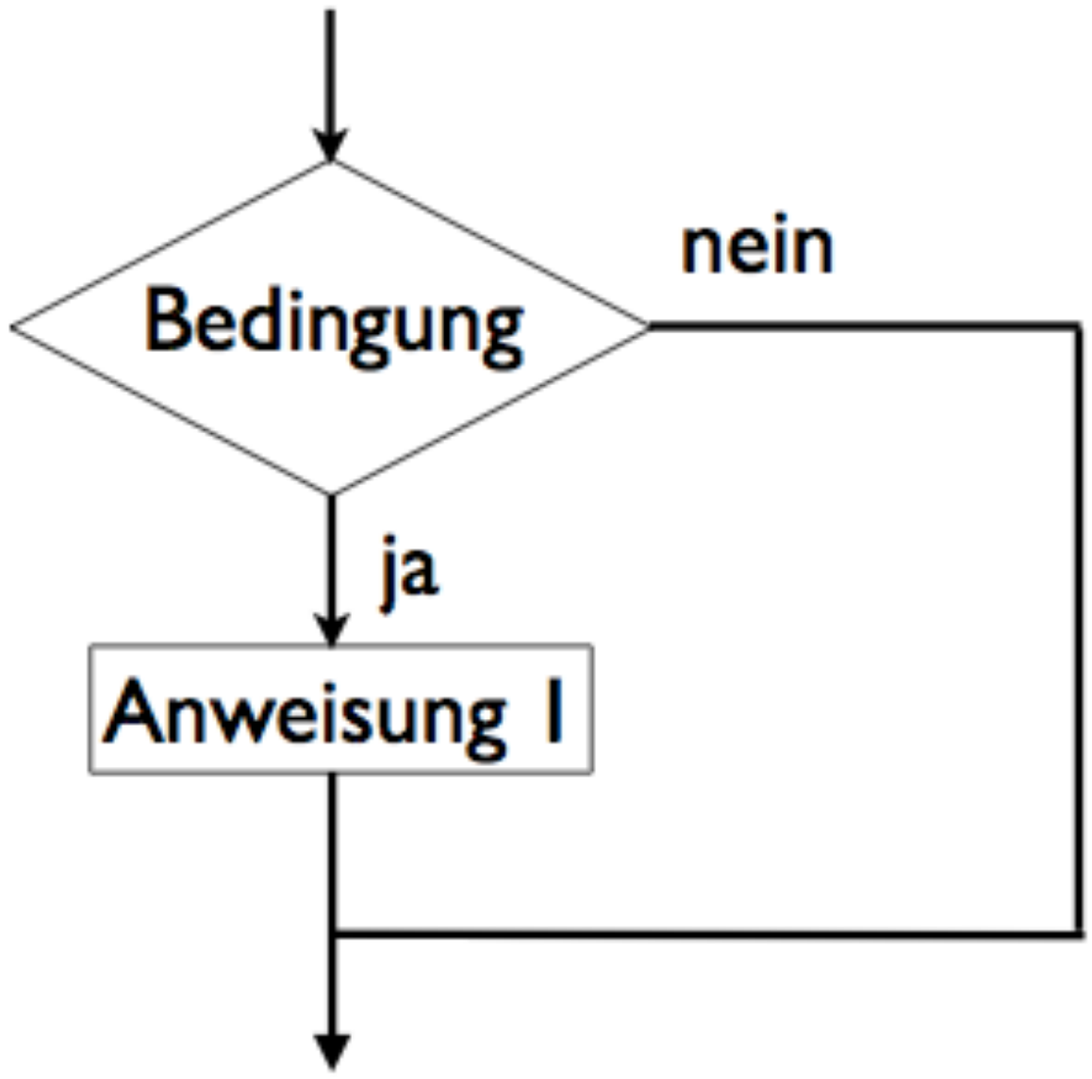
Bedingte Anweisung - zweiseitig

```
if (Bedingung)
  {
   Anweisungsblock1
  }
else {
   Anweisungsblock2
  }
```

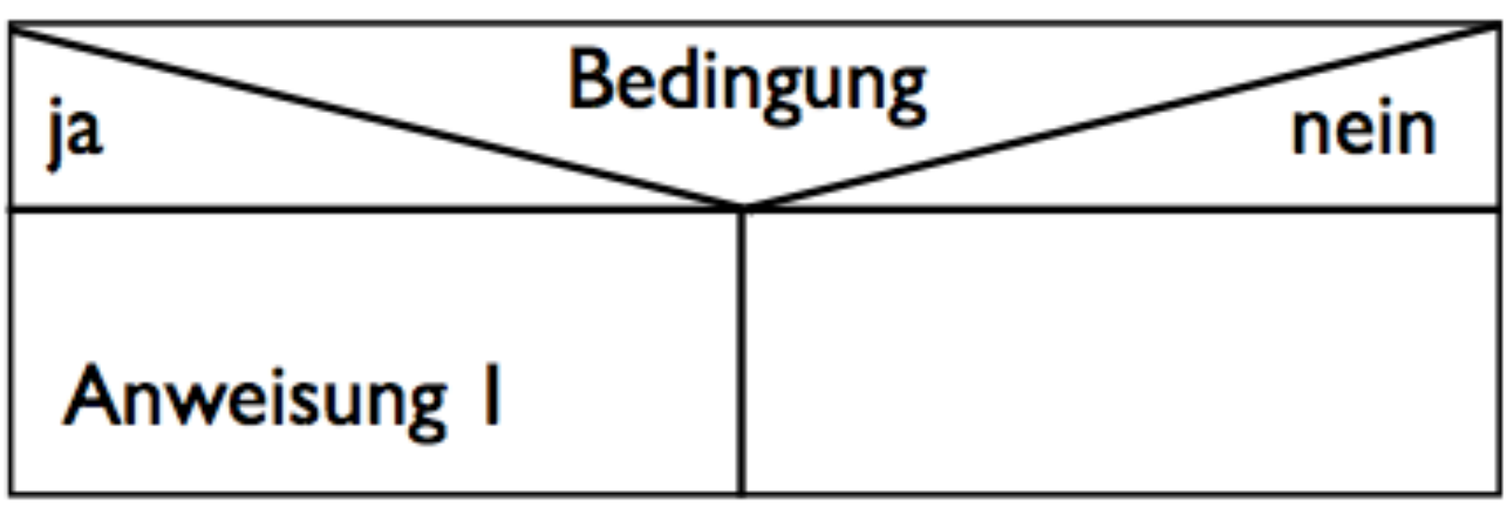
Bedingte Anweisung - zweiseitig

```
if (baumVorne())
  {
   turnLeft();
   move();
   turnRight();
    }
else
{
   move();
}
```

Bedingte Anweisung - einseitig



Bedingte Anweisung - einseitig



Bedingte Anweisung - einseitig

```
if (Bedingung)
  {
   Anweisungsblock
  }
```

Bedingte Anweisung - einseitig

```
if (baumRechts())
  {
   turnLeft();
   move();
   turnRight();
}
```

Bedingte Anweisung - mehrseitig

```
switch (ausdruck)
{
   case wert1:anweisung1;
   case wert2:anweisung2;
   case wert3:anweisung3;
...
   case wertN:anweisungN;
   default:anweisungSonst;
}
```

Bedingte Anweisung - Java, Greenfoot

15. Programmiere folgende Vorgaben:

a. Der Wombat soll sich bewegen, wenn rechts von ihm ein Baum ist. Sonst soll er nichts tun.

b. b. Wenn LinksVonWombatBaum
dann WombatLinksDrehen
sonst WombatRechtsDrehen

UND-Verknüpfung

Die UND-Verknüpfung ist nur dann wahr, wenn alle Teilaussagen wahr sind.

Aussage a	Aussage b	a & b
WAHR	WAHR	WAHR
WAHR	FALSCH	FALSCH
FALSCH	WAHR	FALSCH
FALSCH	FALSCH	FALSCH

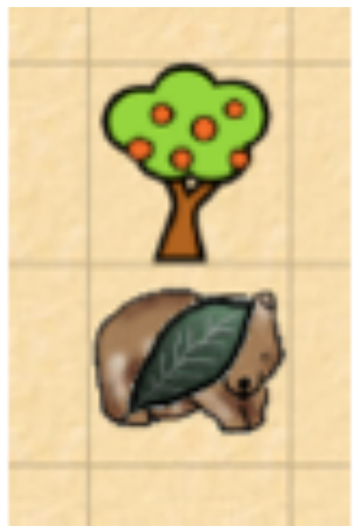
ODER-Verknüpfung

Die ODER-Verknüpfung ist nur dann falsch, wenn alle Teilaussagen falsch sind.

Aussage a	Aussage b	a | b
WAHR	WAHR	WAHR
WAHR	FALSCH	WAHR
FALSCH	WAHR	WAHR
FALSCH	FALSCH	FALSCH

16. Was passiert?

```
if (foundLeaf() && baumLinks())
   {
    eatLeaf();
    move();
    haufenAblegen();
  }
 else
 {
    move();
 }
```



17. Was passiert?

```
if (foundLeaf() && baumLinks())
  {
   eatLeaf();
   move();
   haufenAblegen();
  }
else
{
   move();
}
```

18. Was passiert?

```
if (foundLeaf()  &&
baumLinks())
  {
   eatLeaf();
   move();
   haufenAblegen();
  }
else
{
   haufenAblegen();
}
```

UND/ODER-Verknüpfung

19. Programmiere folgende Vorgabe

Ein Wombat bewegt sich eine gerade Reihe von Feldern entlang.
Dort liegende Blätter soll er essen, wenn

a. links und rechts von ihm ein Baum ist
b. links oder rechts von ihm ein Baum ist
c. links und rechts von ihm ein Baum oder vor ihm ein Stein ist!

Bedingte Anweisung - Java

20. Welche Ausgabe erhält man, wenn der Wert von x zu Beginn x=-1 (x=3; x=5) ist?

```
if (x>0) if (x>4)
    x=0;
else
    x=10;
System.out.println(x);
```

Bedingte Anweisung - Java

21. Welche Ausgabe erhält man, wenn der Wert von x zu Beginn x=-1 (x=3; x=5) ist?

```
if (x>0)
{
    if (x>4)
     x=0;
}
else
   x=10;
System.out.println(x);
```

NOT-Verknüpfung

Die NOT-Verknüpfung ist dann wahr, wenn die (eigentliche) Aussage falsch ist.

Aussage a	! a
WAHR	FALSCH
FALSCH	WAHR

22. Gib an, bei welchen Situationen sich der Wombat dreht!

```
if (foundLeaf()   ||
    !baumLinks()  ||
    baumVorne()   ||
    baumRechts())
{
  turnRight();
}
```

23. Gib an, bei welchen Situationen sich der Wombat dreht!

```
if (foundLeaf() &&
    baumLinks() &&
    baumVorne() &&
    baumRechts())
{
  turnRight();
}
```

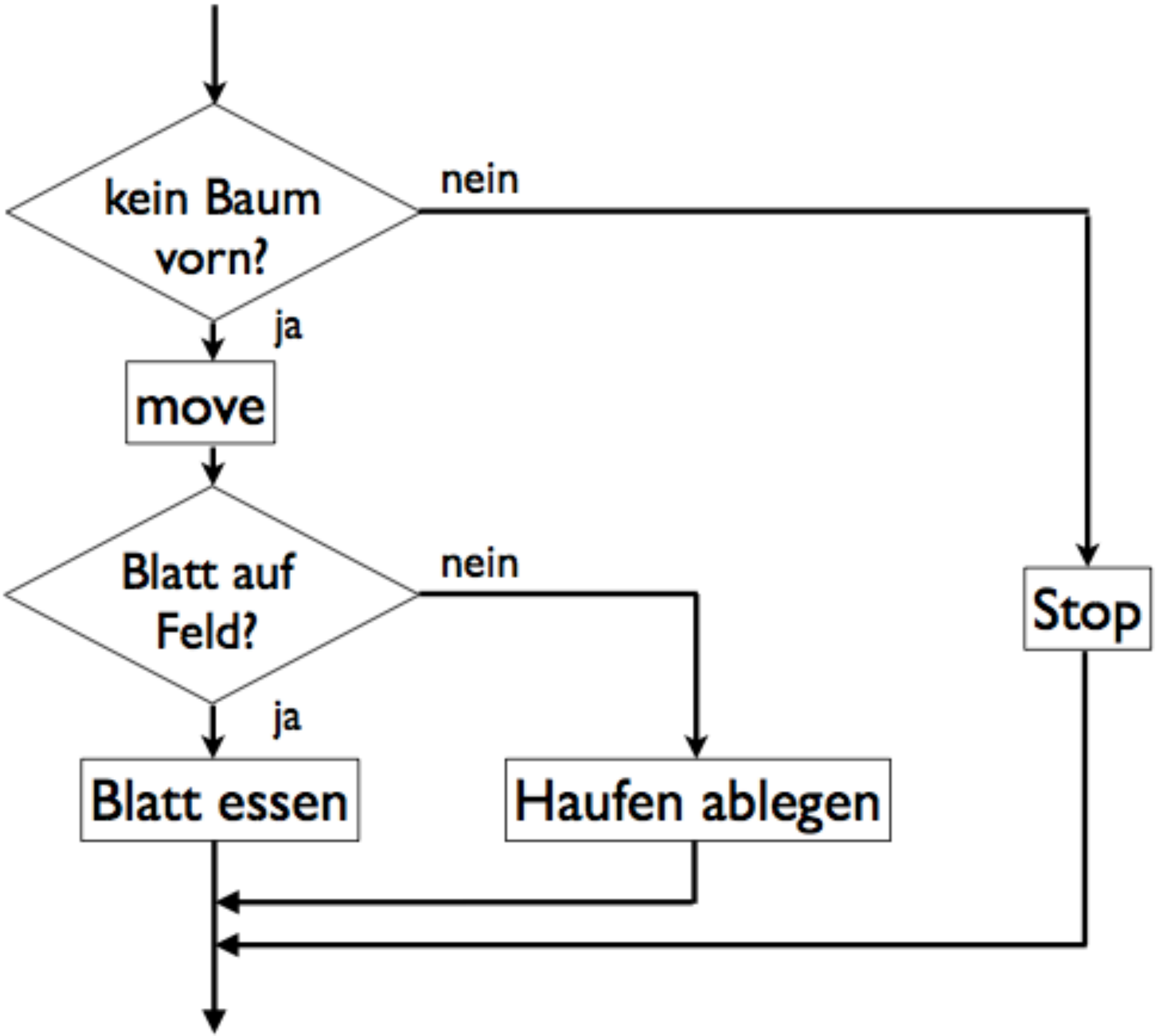
24. Gib an, bei welchen Situationen sich der Wombat dreht!

```
if (foundLeaf() &&
    baumLinks() ||
    baumVorne() ||
    baumRechts())
{
  turnRight();
}
```

25. Programmiere folgende Vorgabe!

Ein Wombat soll gerade bis zu einem Baum laufen. Bis dorthin soll er alle Blätter, die sich auf Feldern befinden, die er betritt, essen. Befindet sich kein Blatt auf einem Feld, soll er dort einen Haufen hinterlassen.

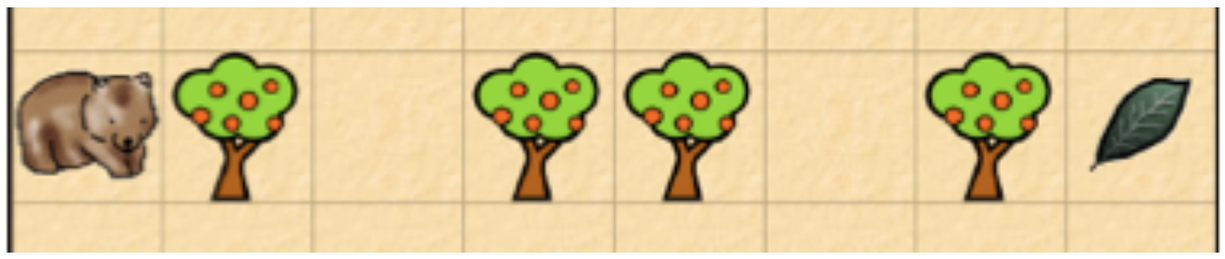
Stelle den Algorithmus in einem PAP oder Struktogramm dar!
Implementiere ihn!


kein Baum vorn?
nein
ja
move
Blatt auf Feld?
nein
ja
Stop
Blatt essen
Haufen ablegen

26. Programmiere folgende Vorgabe!

Ein Wombat sucht ein Blatt. Es befindet sich in derselben Zeile wie der Wombat. Er muss nur die Bäume umlaufen.
Grundaufgabe: Bäume stehen immer einzeln.

Erweiterung: Bäume können auch nebeneinander stehen.



27. Programmiere folgende Vorgabe!

In der Wombatwelt gibt es Rundgänge folgender Art:
Jedes Feld hat genau zwei freie Nachbarfelder. Eines davon liegt vor, links oder rechts vom Wombat.

Ein Blatt ist im Labyrinth, dieses soll gesucht werden.