4.1 Testbefehle (Boolesche Ausdrücke)

Wenn Sie den Hamster in eine unlösbare Situation bringen (z.B. **`vor();`** obwohl vor ihm eine Mauer steht), dann ist der Hamster derart von Ihnen enttäuscht (wg. Kopfweh), dass er nicht mehr mitspielt.
Es tritt ein Laufzeitfehler auf: z.B. MauerDaException(1,2) in beispiel.main (Zeile 2).

Um dies zu vermeiden, werden nun drei so genannte Testbefehle eingeführt:

- **`vornFrei()`** Liefert den Wert **true** wenn keine Mauer vor dem Hamster ist. Ist die Kachel vor dem Hamster mit einer Mauer blockiert, dann wird der Wert **false**.
- **`maulLeer()`** Befinden sich keine Körner im Maul ist der Wert **true**. Liefert den Wert **false** falls der Hamster ein oder mehrere Körner im Hamstermaul hat.
- **`kornDa()`** Liefert den Wert **true**, falls auf der Kachel auf der der Hamster gerade steht, ein oder mehrere Körner liegen. Befindet sich kein Korn auf der Kachel, dann ist der Wert **false**.

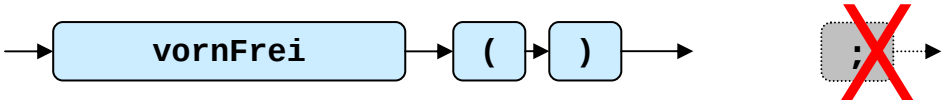
Testbefehle liefern boolesche Werte, also **true** (wahr) oder **false** (falsch).
Die drei Testbefehle sind **boolesche Ausdrücke.**

Mit Hilfe der drei Testbefehle lassen sich die drei gefährlichen Situationen nun vorherbestimmen, und entsprechende Fehler können vermieden werden.
Die drei gefährlichen Situationen sind:

- Wenn der Hamster vor einer Mauer steht und Sie den Befehl **`vor();`** geben.
- Wenn der Hamster keine Körner im Maul hat, Sie aber den Befehl **`gib();`** geben.
- Wenn der Hamster mittels des Befehls **`nimm();`** ein Korn von einer leeren Kachel aufnehmen soll.

4.2 Syntax der Testbefehle

Hinter dem Namen des Testbefehls folgen eine öffnende und eine schließende runde Klammer. Das abschließende Semikolon fehlt.



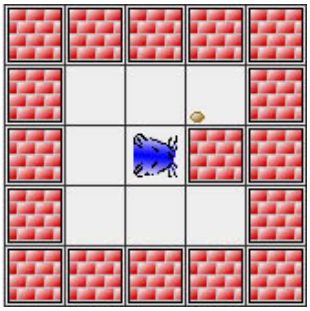
Syntaxdiagramm: Testbefehl vornFrei()

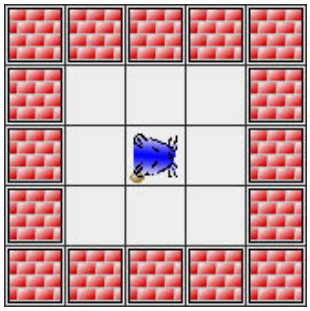
4.3 Beispiele

Schauen Sie sich die beiden Territorien an.

Wird dem Hamster im linken Territorium der Testbefehl: **`vornFrei()`** gegeben, so liefert der Testbefehl den booleschen Wert: **false**.

Wird dem Hamster im rechten Territorium der Testbefehl: **`kornDa()`** gegeben, so liefert der Testbefehl den booleschen Wert: **true.**





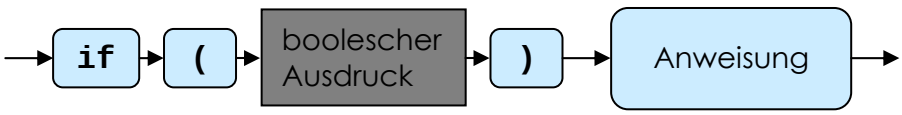
4.4 IF-Anweisung (Bedingte Anweisung)

Wo und wie lassen sich Testbefehle (boolesche Ausdrücke) im Hamstermodell benutzen? In Abschnitt 4.1 haben wir gelesen, dass sich mit Hilfe der Testbefehle gefährliche Situationen vorherbestimmen lassen. Dies nutzen wir jetzt bei der IF-Anweisung.

Beispielsweise soll der Befehl **`vor();`** nur dann ausgeführt werden, wenn der Testbefehl **`vornFrei()`** den Wert **true** liefert. Dies nennt man eine bedingte Anweisung oder auch Verzweigung. Nur wenn . . . , dann . . .

4.5 Syntax der IF-Anweisung

Die bedingte Anweisung, ist eine zusammengesetzte Anweisung!
Sie wird eingeleitet durch das Schlüsselwort **`if`** . Anschließend folgen innerhalb eines runden Klammerpaares ein boolescher Ausdruck und danach eine Anweisung oder eine Anweisungsliste. Was ist eine Anweisungsliste? Siehe 4.6



Syntaxdiagramm: `if` -Anweisung

4.6 Anweisungsliste → { }

Eine Anweisungsliste fasst eine oder mehrere Anweisungen zusammen. Um die Anweisung(en) zusammenzufassen, werden die geschweiften Klammern **{ }** verwendet (man spricht auch von einer Blockanweisung). Warum dies wichtig ist, wird in 4.7 erklärt.

Und so sieht das aus:

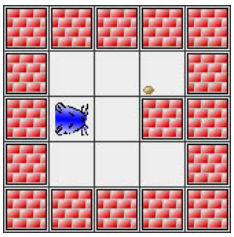
```
{
vor();
linksUm();
nimm();
vor();
}
```

Die vier Anweisungen sind durch die geschweiften Klammern zu einer Anweisungsliste "verheiratet".

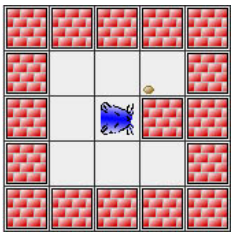
4.7 Beispiele IF-Anweisung

Beispiel 1

```
if(vornFrei())
vor();
```



Territorium 1



Territorium 2

Ein sehr einfaches Beispiel, da **`vor();`** der einzige Befehl ist. Der Hamster geht vorwärts, wenn vor ihm ein Feld frei ist.
Im Territorium 1 wird der Hamster den Befehl **`vor();`** ausführen.
Bei dem Territorium 2 ist dies anders. Hier liefert der Testbefehl den Wert: **false**. Der Hamster wird also nichts tun.

Beispiel 2

```
if(vornFrei())
{
    vor();
    linksUm();
}
```

Diesmal werden zwei Anweisungen ausgeführt, wenn die Bedingung erfüllt ist. Daher sind hier die geschweiften Klammern notwendig (Anweisungsliste). Durch die Einrückung der Befehle um drei Leerzeichen wird deutlich gemacht, dass die beiden Befehle **`vor();`** und **`linksUm();`** Bestandteil einer IF-Anweisung sind. Sie werden also nicht in jedem Fall ausgeführt, sondern nur dann, wenn eine bestimmte Bedingung (Testbefehl) erfüllt ist.

BENUTZEN Sie die **geschwungenen Klammern** (Anweisungsliste) auch wenn Sie nur eine Anweisung haben. Dies hilft Fehler zu vermeiden.

BENUTZEN Sie **Einrückungen**. Die Einrückungen erleichtert das Lesen und Verstehen eines Programms!! Bei Schachtelungen geht ihnen sonst der Zusammenhang verloren!!

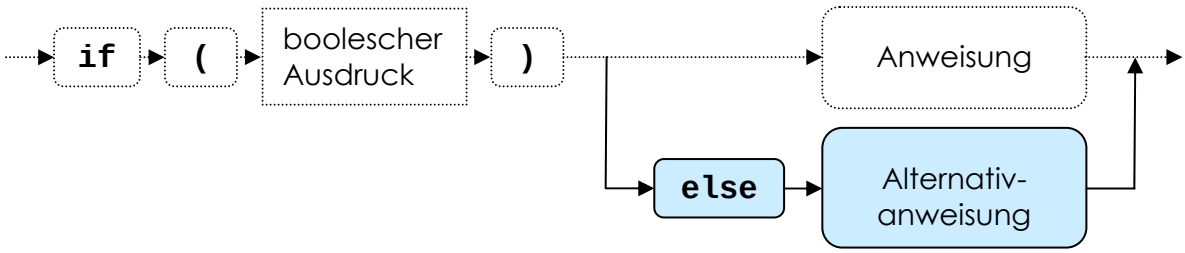
4.8 IF-ELSE-Anweisung

Zur **`if`-Anweisung** gibt es auch noch ein **`else`.** **`Else`** bedeutet soviel wie Alternative. Wir erweitern die **`if`-Anweisung** also um die Alternativanweisung **`else`**. Dies bedeutet:

Wird eine bedingte Anweisung ausgeführt, so wird zunächst der Wert der Bedingung (boolescher Ausdruck: Testbefehl) ermittelt. Ist die Bedingung erfüllt: **true**, dann wird die Anweisung/Anweisungsliste ausgeführt. Ist die Bedingung nicht erfüllt: **false**, dann wird die Alternativanweisung **`else`** ausführt.

4.9 Syntax der ELSE-Anweisung

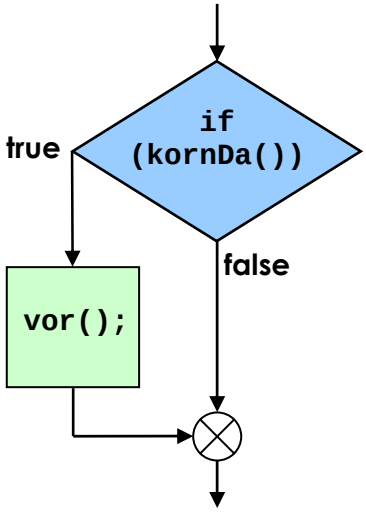
Sie wird eingeleitet durch das Schlüsselwort **`else`** . Anschließend folgt eine Anweisung / Anweisungsliste. Die Alternativanweisung **`else`** hat keine runden Klammern.



Syntaxdiagramm: `else` -Anweisung

4.11 Beispiele und Ablaufdiagramme für IF- bzw. IF/ELSE-Anweisung

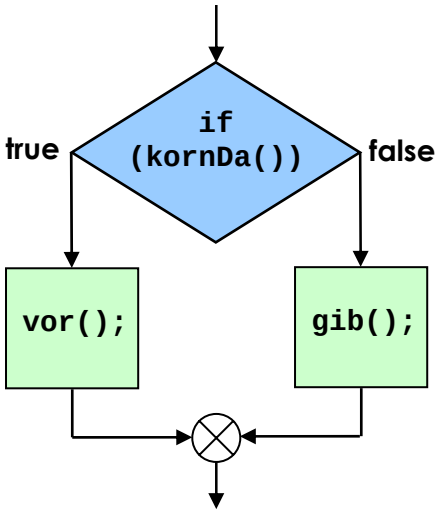
`if`-Anweisung



Weiter im
Programm

```
if (kornDa())
{
vor();
}
```

`if-else`-Anweisung

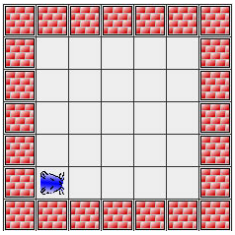


Weiter im
Programm

```
if (kornDa())
{
vor();
}
else
{
gib();
}
```

4.12 Beispiele

Beispiel 1



Der Hamster ist heute verwirrt und weiß nicht wie viele Körner er im Maul hat. Falls möglich soll er in jeder Ecke des Territoriums ein Korn ablegen und dann stehen bleiben.

Lösung 1a

Der Hamster besucht jede der vier Ecken im Territorium und legt, falls er noch ein Korn im Maul hat, jeweils eins ab.

Programmieren und testen Sie das Beispiel. Geben Sie dem Hamster unterschiedlich viele Körner mit und beobachten Sie was passiert.

Im Simulationsfenster werden die abgearbeiteten Befehle angezeigt!

```
01 void main()
02 {
03 if (!maulLeer())
04 gib();
05 laufeEcke();
06 if (!maulLeer())
07 gib();
08 laufeEcke();
09 if (!maulLeer())
10 gib();
11 laufeEcke();
12 if (!maulLeer())
13 gib();
14 laufeEcke();
15 }
16
17 void laufeEcke()
18 {
19 vor(); vor(); vor(); vor(); linksUm();
20 }
```

Lösung 1b

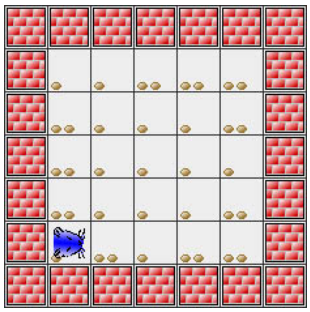
Bei dieser Lösung ist der Hamster schlauer. Wenn er keine Körner mehr im Maul hat, braucht er ja auch gar nicht mehr weiterzulaufen. Er überprüft also nach jedem Ablegen eines Korns, ob es sich lohnt weiterzulaufen.

Programmieren und testen Sie das Beispiel. Geben Sie dem Hamster unterschiedlich viele Körner mit und beobachten Sie was passiert.

Im Simulationsfenster werden die abgearbeiteten Befehle angezeigt!

```
01 void main()
02 {
03   if (!maulLeer())
04   {
05     gib();
06     if (!maulLeer())
07     {
08       laufeEcke();
09       gib();
10       if (!maulLeer())
11       {
12         laufeEcke();
13         gib();
14         if (!maulLeer())
15         {
16           laufeEcke();
17           gib();
18         }
19       }
20     }
21   }
22 }
void laufeEcke() →  wie bei Lösung 1a
```

Beispiel 2



Auf jedem Feld liegen ein oder zwei Körner. Der Hamster soll für Ordnung sorgen. Auf jedem Feld soll genau ein Korn liegen.

Lösung

```
01 void main()
02 {
03   ueberpruefeEineReihe();
04   linksUm(); vor(); linksUm();
05   ueberpruefeEineReihe();
06   rechtsUm(); vor(); rechtsUm();
07   ueberpruefeEineReihe();
08   linksUm(); vor(); linksUm();
09   ueberpruefeEineReihe();
10   rechtsUm(); vor(); rechtsUm();
11   ueberpruefeEineReihe();
12 }
13
14 void ueberpruefeEineReihe()
15 {
16   evtlFressen(); vor();
17   evtlFressen(); vor();
18   evtlFressen(); vor();
19   evtlFressen(); vor();
20   evtlFressen();
21 }
22
23 void evtlFressen()
24 {
25    // erstmal ein Korn fressen
26    nimm();
27    /* falls es das einzige Korn war, muss es
28    wieder abgelegt werden*/
29    if (!kornDa())
30    {
31      gib();
32    }
33 }
34
35 void rechtsUm()
36 {
37    linksUm();
38    linksUm();
39    linksUm();
40 }
```

Befehlsübersicht (Hamsterumgebung)

Funktion	Beschreibung	Typ
`vor();`	Der Hamster geht genau 1 Feld weiter	void
`linksUm();`	Der Hamster dreht sich um 90° nach links	void
`nimm();`	Der Hamster nimmt ein Korn auf	void
`gib();`	Der Hamster legt ein Korn ab	void
`vornFrei()`	Liefert **true**, falls der Hamster nicht vor einer Wand steht	boolean
`kornDa()`	Liefert **true**, falls das Feld, auf dem der Hamster gerade steht, mindestens ein Korn enthält.	boolean
`maulLeer()`	Liefert **true**, falls der Hamster kein Korn im Maul hat.	boolean

Typ "**void**" heißt, dass die Funktion kein Ergebnis zurückliefert.
Eine Funktion vom Typ "**boolean**" liefert einen von zwei Wahrheitswerten zurück: **true** oder **false**. Den Wahrheitswert **maybe** gibt es nicht ☺.

Hausaufgabe:

Recherchieren Sie nach:

Boolesche Logik, IF ELSE Anweisung

www.beeck-com.de/E-Informatik

Nutzen Sie hierfür das Schlagwortregister von "JAVA ist auch eine Insel" und Wikipedia.