Kapitel 4 Einfache Methoden der Klasse MAMPFI

Lernziel:
Eine Klasse in Java erstellen: Methoden

4.1 Vervollständigen des Klassendiagramms

Nachdem du im letzten Kapitel in dem Bauplan für mampfi seine Eigenschaften (Attribute festgelegt hast, sollst du ihm jetzt Fähigkeiten beibringen. Dies bedeutet du musst in der Klasse MAMPFI Methoden ergänzen.

Hinweis:
Methoden der Klasse KREISFORM hast du ausführlich in Kapitel 2 kennen gelernt und genutzt. Sie haben dir geholfen, beispielsweise die Werte der Attribute zu ändern.

Aufgabe 4.1
Welche Methoden sind für das Objekt mampfi wichtig?
Welche Möglichkeiten gibt es über Methoden die Blickrichtung zu ändern? Überlege dir verschiedene Möglichkeiten!

Wichtige Fähigkeiten für Mampfi sind seine Blickrichtung zu wechseln und auch seine Verwundbarkeit zu verändern. (Er muss ja später auf das Schlucken der Zauberpille reagieren.) Im Klassendiagramm in Abbildung 1 sind dazu entsprechende Methoden ergänzt.

Hinweis:
Wie schon in Kapitel 3 wird alles, was die Position betrifft zurückgestellt und erst später behandelt, wenn es ein Spielfeld gibt.

Aufgabe 4.2
Welchen prinzipiellen Unterschied gibt es zwischen den vier Methoden die die Blickrichtung ändern und der Methode *verwundbarSetzen*? Vergleiche auch mit den Methoden der Klasse KREISFORM aus Kapitel 2!

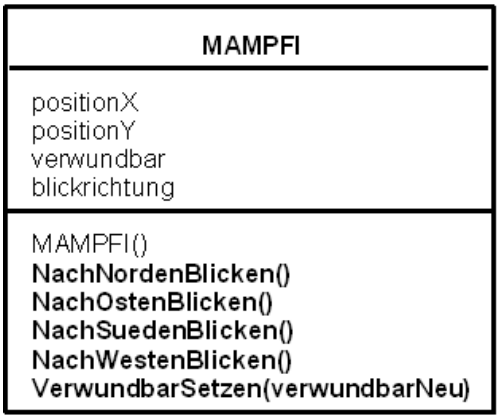


Abbildung 1: Klassendiagramm der Klasse MAMPFI durch Methoden ergänzt.

Die Methode *NachNordenBlicken* benötigt keine Zusatzinformationen um ausgeführt zu werden. Im Gegensatz dazu ist bei der Methode V*erwundbarSetzen* die Information „auf was gesetzt werden soll“ erforderlich. Soll das Attribut verwundbar true oder false werden? Die Methode benötigt einen **Eingabewert.** Entsprechend dem Datentyp vom Attribut verwundbar muss auch der Datentyp des Eingabewertes der Methode V*erwundbarSetzen* vom Typ `boolean` sein.[1]

1 Mit Eingabewerten bei Methodenaufrufen hast du schon in Kapitel 2 gearbeitet. So benötigte dort beispielsweise die Methode *RadiusSetzen* der Klasse KREISFORM einen Eingabewert vom Datentyp `int`. Der Eingabewert enthält die Information, welchen Wert der Radius annehmen soll.

Somit benötigt man zur Planung der Methoden (wie auch bei der Planung von Attributen) zwei Schritte:

1) Welche Methoden sind (für die Aufgabenstellung) wichtig? Welche dieser Methoden benötigt beim Aufruf Eingabewerte?
   Das Ergebnis dieses Planungsschrittes ist das Klassendiagramm (Abbildung 1).
2) Welchen Datentypen haben die Eingabewerte?
   Das Ergebnis dieses Planungsschrittes ist das erweiterte Klassendiagramm (Abbildung 2)

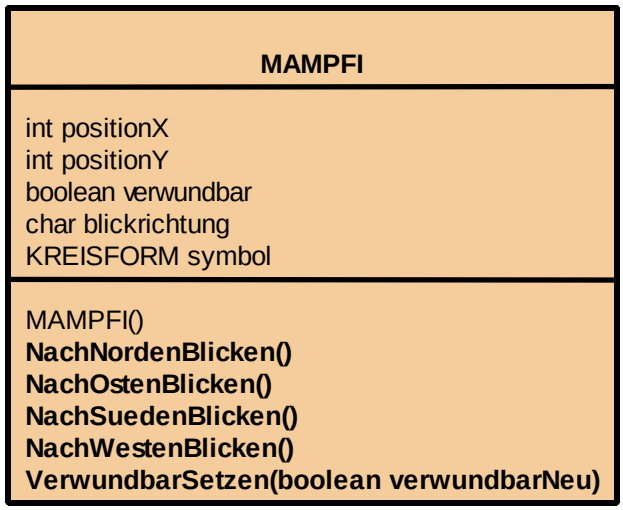


Abbildung 2: Erweitertes Klassendiagramm der Klasse MAMPFI – Im Vergleich zur Abbildung 1 wurde bei den Eingangsparametern der Methoden die Datentypen ergänzt.

Hinweise:

- Beachte, dass dieses schrittweise Vorgehen bei der Planung – erst ohne Datentypen, dann mit – identisch zu dem Vorgehen bei Attributen ist.
- Unterscheide die Begriffe **Eingabewert** und **(Eingangs-)parameter**. Für die allgemeine Formulierung einer Methode wird ein Platzhalter – (Eingangs-)parameter genannt – verwendet. Im Beispiel der Methode V*erwundbarSetzen* ist dies verwundbarNeu: V*erwundbarSetzen(**verwundbarNeu**)*
  Wird die Methode dann aufgerufen, wird der Parameter (Platzhalter) durch einen konkreten Wert ersetzt, z.B.
  V*erwundbarSetzen(**true**)*

Aufgabe 4.3
Erkläre deinem Nachbar den Unterschied der Begriffe Eingabewert und Eingangsparameter am Beispiel der Methode *StartWinkelSetzen* der Klasse KREISFORM. Falls nötig kannst du das Klassendiagramm in Kapitel 2 Abbildung 5 finden.

Nach den Überlegungen welche Methoden nötig sind und ob sie Eingabewerte benötigen wird im nächsten Kapitel besprochen, wie man sie in Java umsetzen kann. Dazu muss man jedoch festlegen, was das Objekt tun soll, wenn die Methode aufgerufen wird.

Aufgabe 4.4
Was soll mampfi tun, wenn seine Methode *NachOstenBlicken* aufgerufen wird? Beschreibe zunächst in Worten, welche Aktionen auszuführen sind. Versuche dann diese Beschreibung in Form von Zuweisungen und Methodenaufrufen zu formulieren.

4.2 Einfache Methoden in Java

Wird die Methode *NachOstenBlicken* aufgerufen, dann muss mampfi einerseits den Wert seines Attributs blickrichtung auf 'O' setzen, andererseits muss er dafür sorgen, dass auch der Wert des Startwinkels seiner symbols angepasst wird, so dass auch die Darstellung nach rechts blickt.

Der Quelltext in Java zur Umsetzung dieser Aufträge ist im rechten Teil der Abbildung 3 zu sehen:

```
void Methodenname()
{
    //Beschreibung der
    //Anweisungsfolge
}
```

```
void NachOstenBlicken()
   {
       blickrichtung = 'O';
       symbol.StartWinkelSetzen(30);
   }
```

Abbildung 3: Aufbau einer Methode ohne Rück- und Eingabewerte in Java: Allgemein und am Beispiel der Methode NachOstenBlicken

Den konkreten Anweisungen der Methode *NachOstenBlicken* ist in Abbildung 3 auch ihr allgemeiner Aufbau gegenübergestellt.

Hinweise:
Hier werden Methoden betrachtet, die keine Eingabewerte, d.h. keine zusätzlichen Informationen zum Ausführen benötigen.
Weiterhin können Methoden auch Informationen ausgeben. So haben alle Handys eine Methode der Art *LetzteGewählteRufnummerGeben*, die dem Aufrufer die letzte gewählte Rufnummer zurück gibt. als Rückgabewert eben di könnte eine der allgemeine Aufbau einer Methode ohne Rückgabewert und ohne Eingabewerte Methoden mit Rückgabewerten wurden bisher noch nicht angesprochen, sie werden an späterer Stelle behandelt.

In der folgenden Tabelle werden die einzelnen Bestandteile einer Methode erläutert:

Bestandteil einer Methode ohne Ein- und Ausgabewert	Erläuterung
`void`	Kennzeichnet, dass die Methode keinen Rückgabewert hat
`NachOstenBlicken`	Bezeichner der Methode
`()`	Ein Klammerpaar nach einem Bezeichner ist ein eindeutiges Kennzeichen für eine Methode. Hat eine Methode keine Eingabewerte, dann steht innerhalb des Klammerpaares nichts.
`{ ... }`	Die geschweiften Klammern legen den Beginn und das Ende der Anweisungsfolge fest.
`blickrichtung = 'O';` `symbol.StartWinkelSetzen(30);`	Eine Folge von Anweisungen, die festlegt, was in dieser Methode getan wird. In diesem Fall wird in der ersten Zeile dem Attribut blickrichtung der Wert 'O' zugewiesen. In der zweiten Zeile wird in Punktnotation die Methode *StartWinkelSetzen* des Objekts symbol mit dem Eingabewert 30 aufgerufen.

Abbildung 4: Bestandteile einer Methode am Beispiel der Methode *NachOstenBlicken*

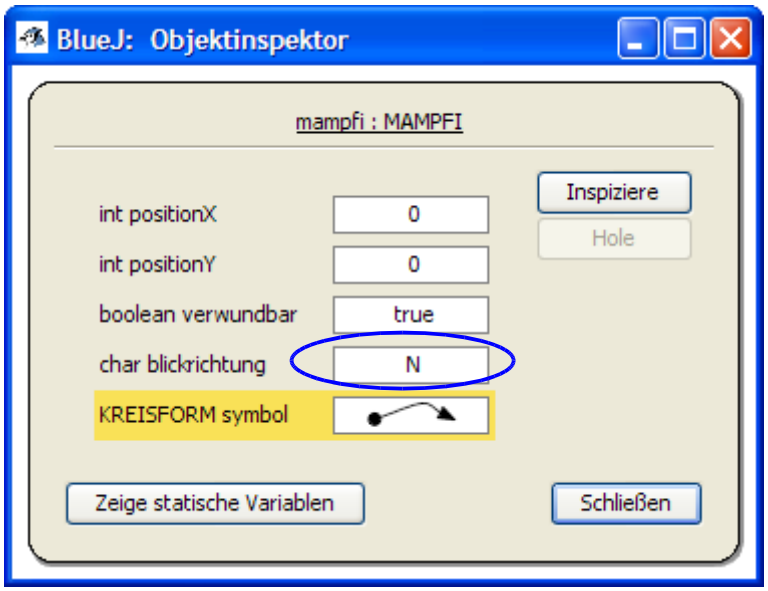
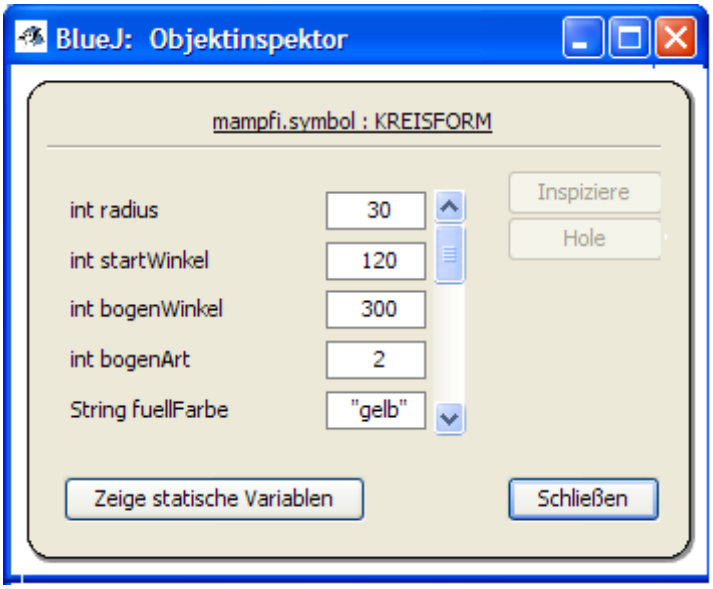
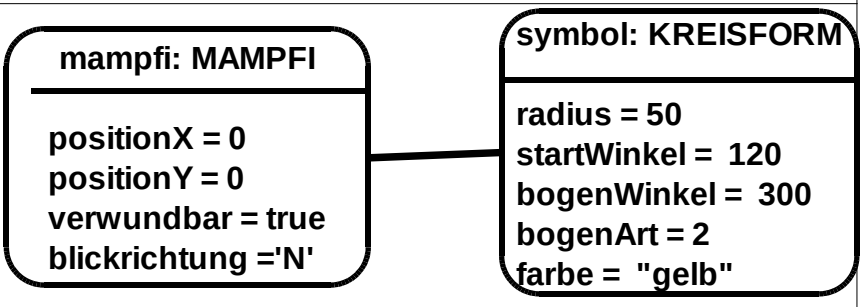
Hinweise:
- Man nennt die erste Zeile einer Methode **Methodenkopf** und die durch geschweifte Klammern eingerahmte Anweisungsfolge **Methodenrumpf**.
- Der Konstruktor ist eine besondere Methode. Im Gegensatz zu den hier beschriebenen beginnt sein Methodenkopf nicht mit dem Wort void.
- Die Methoden werden im Java Quelltext direkt nach dem Konstruktor geschrieben. Dies gilt sowohl für die Klassendiagramme als auch für den Quelltext.

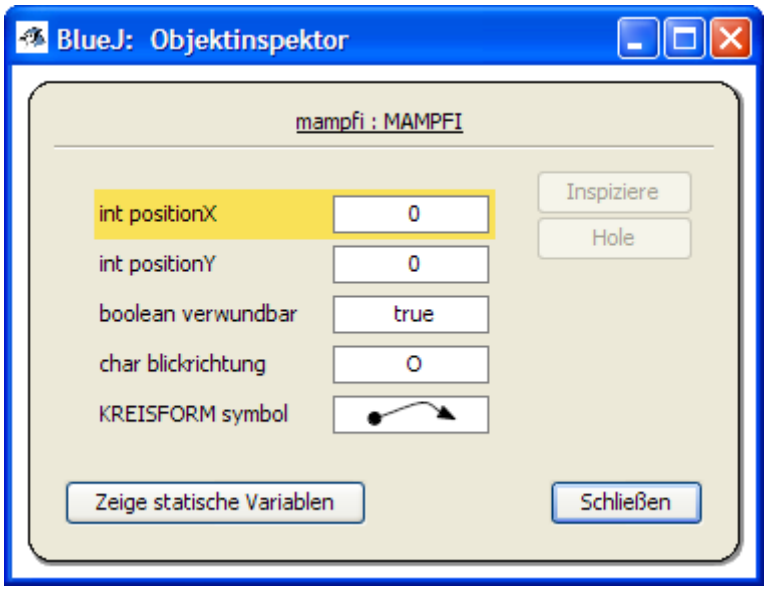
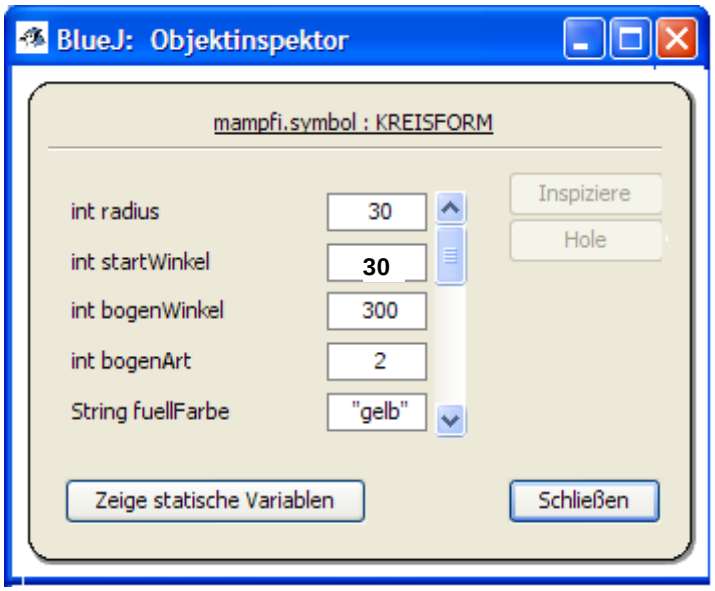
Aufgabe 4.5
Setze die Methoden *NachOstenBlicken, NachSuedenBlicken, NachWestenBlicken* und *NachNordenBlicken* in deinem BlueJ Projekt um.

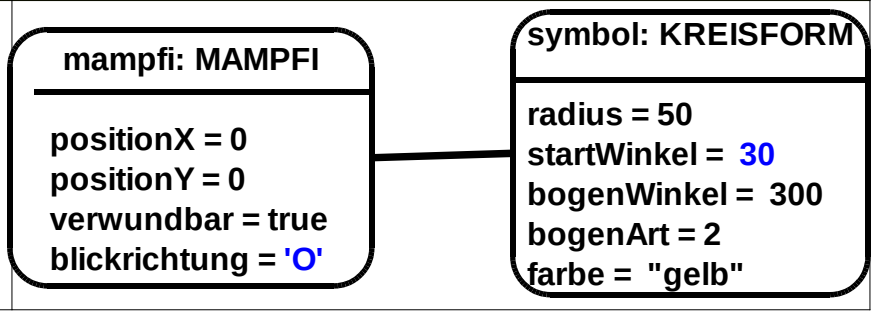
4.3 Funktionieren die Methoden wirklich? - Testen ist wichtig

Nach jeder Programmänderung muss man überprüfen, ob das Ziel erreicht wurde. Ein wichtiges Hilfsmittel dazu ist der Objektinspektor. Ein typisches Vorgehen findest du in der folgenden Tabelle:

Schritt beim Testvorgang	Ergebnis am Beispiel der Methode *NachOstenBlicken*
Compilieren der veränderten Klasse und Erzeugen eines Objekts	

Schritt beim Testvorgang	Ergebnis am Beispiel der Methode *NachOstenBlicken*
Aufruf des Objektinspektors [Falls vorhanden, sollte die zugehörige Zeichenfläche auch betrachtet werden]  	 
Beachte: Der Objektinspektor in BlueJ ist äquivalent zum Objektdiagramm auf Modellierungs-ebene. Aus Platzgründen wird diese Variante ab Kapitel 5 gewählt	

Schritt beim Testvorgang	Ergebnis am Beispiel der Methode *NachOstenBlicken*
Aufruf der zu testenden Methode	
Beobachtung im Objektinspektor, ob die gewünschten Attributwertänderungen durchgeführt wurden [Falls vorhanden, sollte die zugehörige Zeichenfläche auch betrachtet werden]	  

Schritt beim Testvorgang	Ergebnis am Beispiel der Methode *NachOstenBlicken*
Alternative Darstellung im Objektmodell	

Aufgabe 4.6
Prüfe auch die Methoden *NachSuedenBlicken, NachWestenBlicken* und *NachNordenBlicken auf diese Art und Weise*.

4.4 Darstellung von Objektkommunikation durch Sequenzdiagramme

In der Tabelle am Ende von Kapitel 4.3 sind zwei Objektdiagramme enthalten. Es wird dadurch deutlich, dass beim Methodenaufruf *NachOstenBlicken* zwei eigenständige Objekte beteiligt sind.

Aufgabe 4.7
Wie erfährt, das Objekt der Klasse KREISFORM von der „Arbeit“ die es zu erledigen hat?

Betrachtet man den Ablauf bei einem Aufruf der Methode *NachOstenBlicken* Schritt für Schritt, dann ergeben sich folgende Bestandteile:

- Du als externer Aufrufer rufst per Mausklick die Methode *NachOstenBlicken* des Objekts mampfi auf.
- Mampfi selbst benötigt die Hilfe des Objekts symbol für eine korrekte Darstellung auf der Zeichenfläche. Da Mampfi in seinem Referenzattribut eine Referenz auf das Objekt symbol gespeichert hat, kann er ihm Nachrichten schicken. So ruft mampfi innerhalb der Methode *NachOstenBlicken* die Methode *StartWinkelSetzen* des Objekts symbol mit dem Eingabewert 30 auf.

Im Prinzip findet hier im Vergleich zum letzten Kapitel nichts Neues statt. Die Objekte kommunizieren über Methodenaufrufe miteinander. Da die Objektkommunikation zentral für das Funktionieren von Programmen ist, gibt es eine eigene Modellierungstechnik diese zu veranschaulichen: **Sequenzdiagramme** (Abbildung 5).

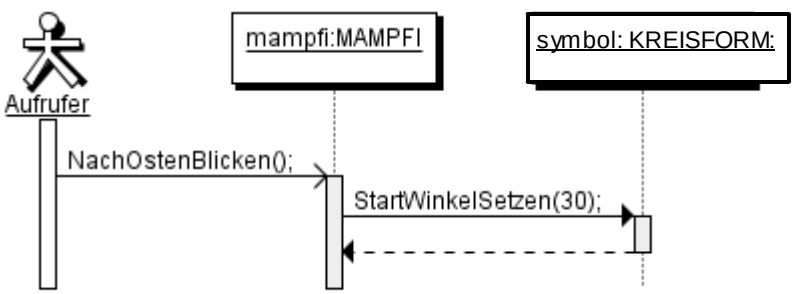


Abbildung 5: Sequenzdiagramm für den Aufruf der Methode NachOstenBlicken

Objekte sind als Rechtecke dargestellt, Methodenaufrufe durch waagrechte Pfeile. Erhält ein Objekt einen Methodenaufruf wird es aktiv. Dies ist erkennbar dadurch, dass die Lebenenslinie (senkrecht gestrichelte Linie) in einen Balken übergeht. Ist ein Objekt aktiv, kann es seinerseits Botschaften mit Methodenaufrufen versenden, sollte es Hilfe von anderen Objekten benötigen.

Hinweis:
Voraussetzung für eine funktionierende Objektkommunikation ist immer, dass das Sendeobjekt das Empfängerobjekt über eine Referenz kennt.

Aufgabe 4.8
Führe die Objektkommunikation in Abbildung 5 wieder als Rollenspiel durch. Beachte, dass im Unterschied zu Aufgabe 3.7 nun pro Runde 3 Objekte beteiligt sind und dass die Methodenaufrufe im Sequenzdiagramm nicht in Punktnotation notiert sind (Den Adressaten des Methodenaufrufs sieht man im Sequenzdiagramm einfach durch den Bezug zu den Objekten)

4.5 Methoden mit Parametern in Java

Bei der Erstellung des Klassendiagramms (Abbildungen 1 und 2) wurde festgelegt, dass die Methode *verwundbarSetzen* einen Eingabewert benötigt. Die Umsetzung dieser Methode in Java unterscheidet sich strukturell nicht im Vergleich zu Methoden ohne Eingabewerten (Kapitel 4.3) Einzig muss im Klammerpaar nach dem Methodennamen Datentyp und Bezeichner der Eingangsparameter angegeben werden (genauso wie es bereits im erweiterten Klassendiagramm notiert ist. Einen Überblick über den Java-Quelltext einer Methode mit Eingabeparametern im Allgemeinen und die Methode *VerwundbarSetzen* im Speziellen gibt Abbildung 6:

```
void Methodenname(Parameterliste)
{
    //Beschreibung der
    //Anweisungsfolge
}
```

```
void VerwundbarSetzen(boolean verwundbarNeu)
{
    verwundbar= verwundbarNeu;
}
```

Abbildung 6: Aufbau einer Methode mit Eingangsparametern in Java - Allgemein und am Beispiel der Methode VerwundbarSetzen

Hinweis:
Mehrere Eingangsparameter sind bei Methoden auch möglich (Parameterliste). Alle werden innerhalb des Klammerpaars, jeweils durch ein Komma getrennt, notiert, z. B. AdresseSetzen(String strasseNeu, int PLZNeu, String ortNeu)

4.6 Exkurs: Methoden arbeiten wie Maschinen

Eine Methode kann man sich wie eine Maschine vorstellen, die irgendetwas arbeitet (in unserem Fall dem Attribut verwundbar den Wert verwundbarNeu zuweisen). Benötigt die Maschine für ihre Arbeit Informationen von außen, so hat die Maschine für jede benötigte Informationseinheit einen Trichter für

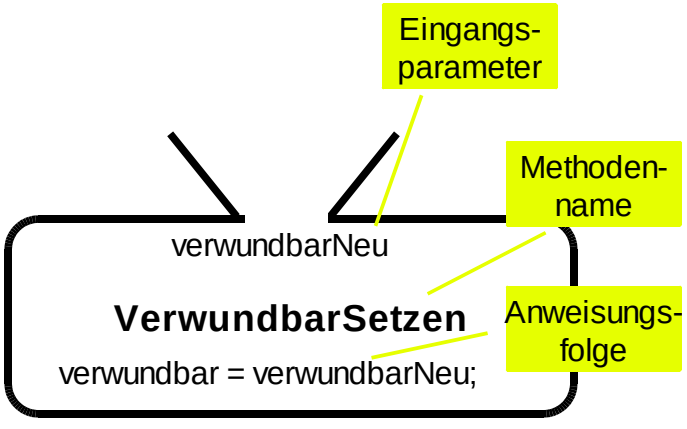


Abbildung 7: Die “Maschine” verwundbarSetzen” mit dem Eingangsparameter verwundbarNeu

eine Eingabe. Jeder Trichter hat einen Namen. Er symbolisiert einen **Eingangsparameter** (in unserem Fall verwundbarNeu).
Wird die Methode nun aufgerufen, muss man einen konkreten Eingabewert eingeben. Bildlich gesprochen wird der Eingabewert in den Trichter geworfen. Dieser Wert ersetzt nun im Methodenrumpf an allen Stellen den zugehörigen Eingangsparameter. Die Methode kann ihre Arbeit ausführen (Abbildung 8).

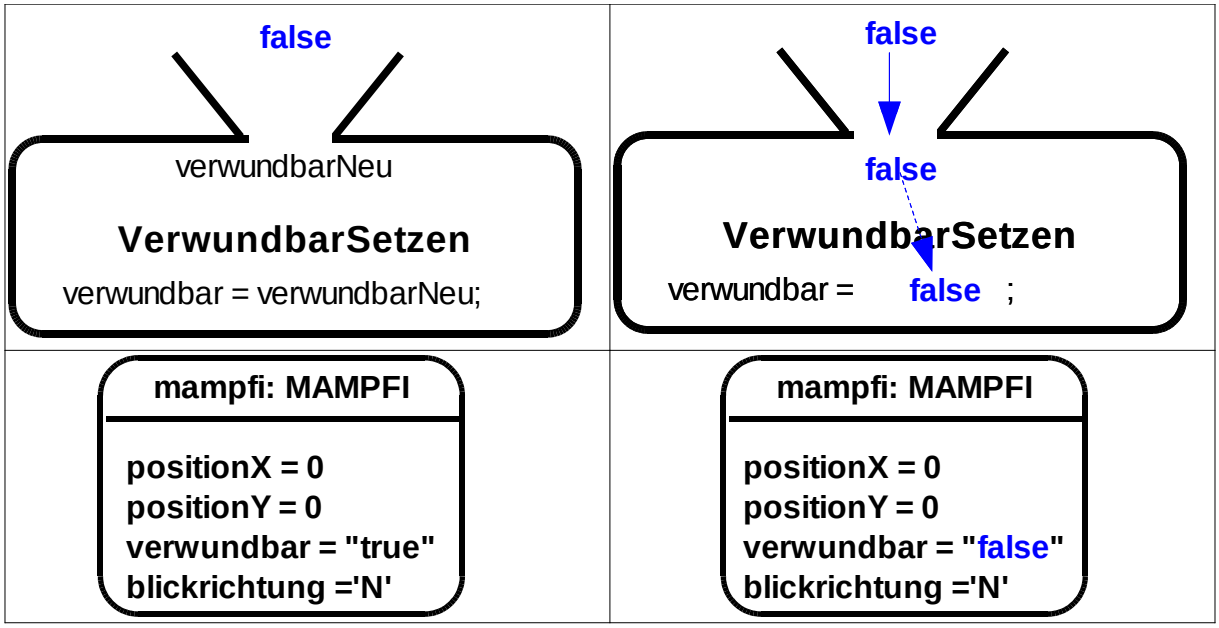


Abbildung 8:
links: Der Wert false wird in die Methode VerwundbarSetzen eingegeben;
rechts: Die Zuweisung innerhalb der Methode VerwundbarSetzen wurde ausgeführt. Das Attribut verwundbar hat jetzt den Wert false.

Aufgabe 4.9
Realisiere die Methode *VerwundbarSetzen* in deinem Projekt und teste sie.

Aufgabe 4.10

Realisiere die Methoden *VerwundbarMachen* und *UnverwundbarMachen*. Die beiden Methoden sollen zusammen eine Alternative zur Methode *VerwundbarSetzen* sein und ohne Eingangsparameter arbeiten.

4.7 Zusammenfassung

Aufgabe 4.11
Wiederum wurden in diesem Kapitel viele neue Fachbegriffe vorgestellt. Da im weiteren Unterrichtsverlauf immer wieder darauf zurückgegriffen wird, ist ein sicherer Umgang damit wichtig.
Erstelle eine Zusammenfassung in dein Heft, bei der folgende Punkte enthalten sein sollten:

Methode, Methodenkopf, Methodenrumpf, Eingabewert, Eingangsparameter, Sequenzdiagramm