

Objektorientierte Hundedressur

*Ein anschauliches Beispiel zur Einführung in die objektorientierte Modellierung und die objektorientierte Programmierung
von Andreas Rittershofer*

Inhaltsverzeichnis

Einführung, Voraussetzungen und Ziele.....	1
Aufgabenstellung.....	2
Identifikation der Klassen.....	3
Identifikation der Attribute.....	4
Identifikation der Methoden.....	5
Modellierung und Programmierung der Methoden.....	6
Methode „dressiereHund()“.....	7
Methode „sageKommando()“.....	7
Methode „hoereKommandoundreagiere()“.....	9
Methode „main()“.....	10
Programmlauf.....	11
Zusammenfassung und Schluss.....	12

Einführung, Voraussetzungen und Ziele

Im modernen Informatikunterricht des Gymnasiums sind neben vielen anderen aktuellen Themenbereichen auch die objektorientierte Modellierung und die objektorientierte Programmierung nicht mehr wegzudenken. Dieser Text stellt ein sehr anschauliches Beispiel vor, mit dem auch und gerade für Einsteiger in die objektorientierte Modellierung und die objektorientierte Programmierung viele wesentliche Prinzipien erläutert werden können:

- Klassen
- Klassen und deren Attribute und Methoden
- Vererbungsbeziehungen zwischen Klassen
- Assoziationen zwischen Klassen
- Abstrakte Klassen
- Erzeugung von Objekten
- Konstruktoren
- Links zwischen Objekten
- Fluss der Botschaften zwischen Objekten
- Klassendiagramme

- Aktivitätsdiagramme
- Storydiagramme

Wir setzen dazu als Programmiersprache die objektorientierte Sprache Java ein und als Werkzeug für die objektorientierte Modellierung Fujaba¹. Im Prinzip kann die objektorientierte Modellierung auch ohne Softwarewerkzeug erfolgen, was aber einen wesentlichen Verzicht auf moderne Möglichkeiten der Softwareentwicklung bedeuten würde. Aus diesem Grund ist auch ein Werkzeug, das ausschließlich Klassendiagramme erzeugen kann, nicht zu empfehlen.

Voraussetzungen für eine erfolgreiche Bearbeitung dieses Beispiels sind ein grundlegendes und sicheres Computerhandling im Allgemeinen, eine bereits erfolgte erste Einführung in die Begriffswelt der Objektorientierung (Begriffe wie Klasse und Objekt, Attribut, Botschaft und Methode, usw. sollten bekannt sein) und die entsprechende Softwareausstattung:

- Ein aktuelles Java-Software-Development-Kit SDK von <http://java.sun.com>
- Eine aktuelle Fujaba-Version von <http://www.fujaba.de>

Ein halbwegs sicherer Umgang mit grundlegenden Funktionen dieser Software sollte durch eine entsprechende Vorbereitung gegeben sein, dazu gehören:

- Übersetzung von Java-Quellcode in Java-Bytecode mit dem Java SDK
- Ausführung von Java-Bytecode durch die Virtual Machine des Java SDK²
- Erzeugung von Projekten, Diagrammen, Klassen
- Generierung von Java-Quellcode

Viele Materialien – auch kleinschrittige Anleitungen mit Angabe der Fujaba-Menüpunkte – rund um diesen Themenkomplex „UML / Java“ stehen auch in Form einer E-Learning-Umgebung für selbstorganisiertes Lernen auf dem LmTM-Server <http://www.LmTM.de> unter Informatik bereit. Fragen jeder Art – sei es die Modellierung, Programmierung oder auch Installation und Konfiguration der Software – können im dortigen Forum unter http://www.LmTM.de/w-agera/index.php?bn=lmtm_umljava diskutiert werden.

Aufgabenstellung

Herr Maier hat einen Hund namens Bello. Diesen will er dressieren, dazu geht er mit ihm auf einen Hundedressurplatz. Der Hund erhält verschiedene Kommandos und reagiert darauf – oder auch nicht. Je nach Reaktion des Hundes verhält sich Herr Maier unterschiedlich: er lobt seinen Hund oder ist unzufrieden mit ihm.

1 Fujaba ist ein UML-Tool. UML steht für Unified Modeling Language und ist die Standard-Notation für die objektorientierte Modellierung, mehr dazu unter <http://www.uml.org>
 2 Das Java SDK (Software Development Kit) bringt auch ein JRE (Java Runtime Environment) mit, das ist im wesentlichen der Java-Interpreter für den Bytecode.

Diese Situation soll mit Hilfe der Objektorientierung modelliert und programmiert werden. Zur Modellierung verwenden wir Fujaba, zur Programmierung Java. Im UML-Tool Fujaba sollen Diagramme erstellt werden, die das System beschreiben; daraus generiert Fujaba Java-Quellcode. Details werden manuell direkt als Java-Code programmiert.

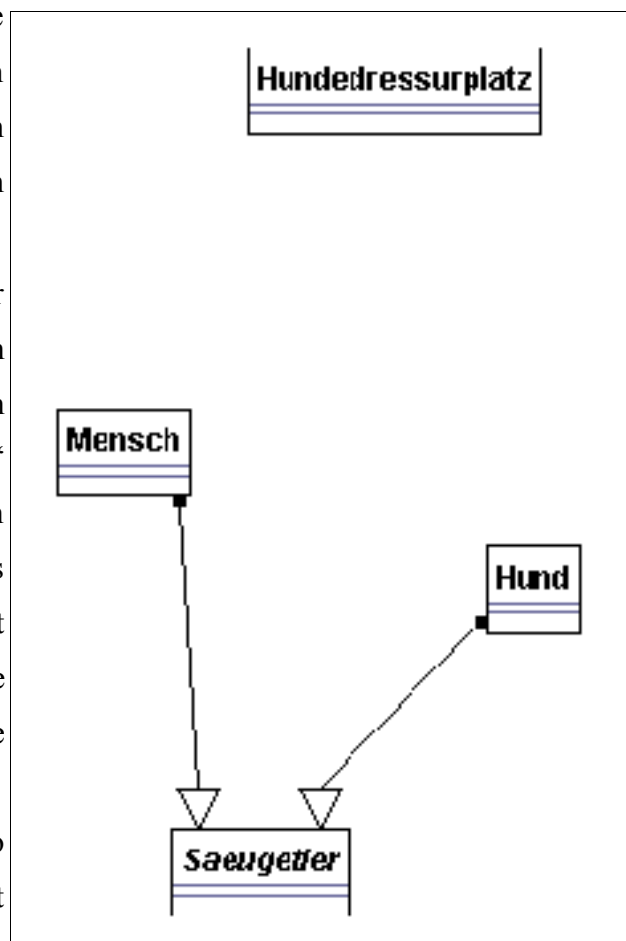
Identifikation der Klassen

Der erste Schritt bei der Modellierung ist die Identifikation der beteiligten Klassen. Herr Maier ist ein Mensch, Bello ein Hund – also benötigen wir die Klassen `Mensch` und `Hund`, damit wir später die Objekte Maier und Bello erzeugen können. (Zur Notation: Klassennamen werden nicht unterstrichen, Objektnamen werden unterstrichen.)

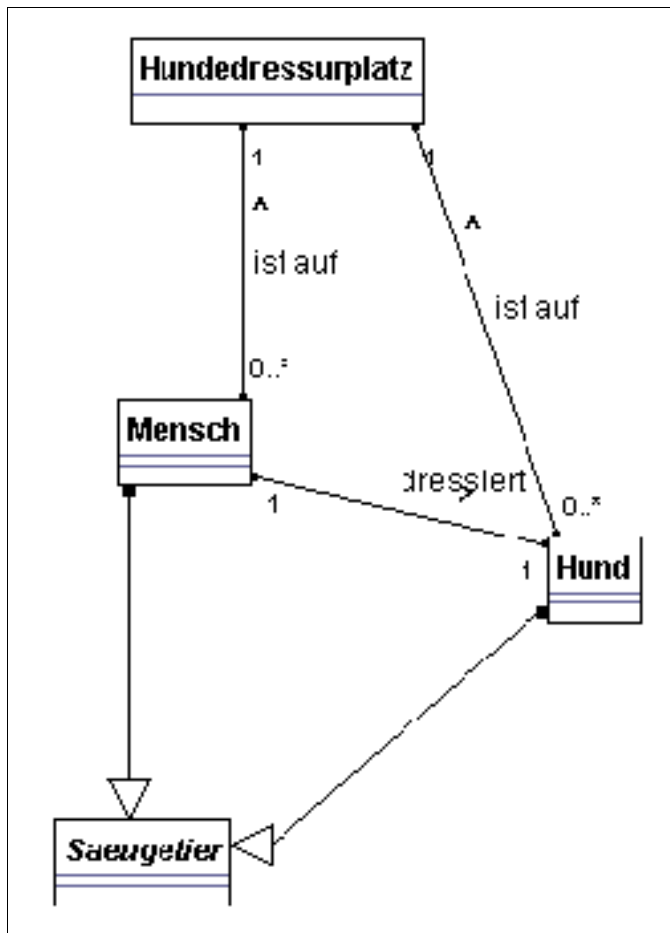
Die ganze Aktion findet auf einem Hundedressurplatz statt, also ist eine weitere Klasse der Hundedressurplatz. Wir können noch eine Verallgemeinerung machen: Herr Maier und sein Hund Bello sind beide Säugetiere, die Klassen `Mensch` und `Hund` können also Unterklassen der Oberklasse `Saeugetier` sein.

Damit können wir einen ersten Anlauf zur Modellierung machen: Wir legen in Fujaba ein neues Projekt „Hunedressur“ an und dort ein Klassendiagramm, das auch „Hunedressur“ heißen kann. In diesem Klassendiagramm erzeugen wir die genannten Klassen. Das Klassendiagramm gibt uns eine statische Sicht auf unser System, nicht dargestellt werden die später zur Programmlaufzeit erzeugten Objekte gemäß den Vorgaben der Klassen.

Klassen in einem Klassendiagramm werden also als Rechtecke dargestellt, der Klassenname steht dabei in der obersten von drei Rubriken und ist



fett gedruckt. Beim `Saeugetier` ist der Name zusätzlich kursiv – wir haben diese Klasse als abstrakte Klasse erstellt. Von abstrakten Klassen können keine Objekte erzeugt werden, sie können aber als Oberklasse für Unterklassen dienen und hier sollen ja auch die Klassen `Mensch` und `Hund` Unterklassen der Oberklasse `Saeugetier` sein. Diese Vererbungsbeziehung wird



über „Derived from“ bei Mensch und Hund eingetragen. Vererbungsbeziehungen werden im Klassendiagramm durch Pfeile dargestellt. Dabei zeigt der Pfeil von der Unterklasse zur Oberklasse, die Pfeilspitze ist nicht ausgefüllt. Zwischen den Klassen Mensch, Hund und Hundedressurplatz gibt es auch noch Beziehungen, diese sind aber keine Vererbungsbeziehungen sondern Assoziationen. Mensch und Hund sind assoziiert über die Assoziation „dressiert“ – natürlich in der Leserichtung, dass der Mensch den Hund dressiert – so sollte es zumindest sein. Die Leserichtung wird durch das >-Zeichen dargestellt. An den beiden Enden der Assoziation stehen die jeweiligen Kardinalitäten. Bei der

Assoziation „dressiert“ ist es jeweils eine „1“, was aussagt, dass 1 Mensch-Objekt 1 Hund-Objekt dressiert, wenn später zur Programmlaufzeit diese Objekte dann existieren. Mensch-Objekt und Hund-Objekt befinden sich während der Dressur auf dem Hundedressurplatz, also sind diese Klassen über die Assoziation „ist auf“ verknüpft. Die Kardinalitäten geben an, dass 0..* (beliebig viele) Mensch-Objekte und 0..* Hund-Objekte auf 1 Hundedressurplatz-Objekt sind, wenn später zur Programmlaufzeit diese ganzen Objekte dann existieren.

Damit ist das Klassendiagramm in seiner Grundstruktur bereits fertiggestellt: Es enthält alle beteiligten Klassen und deren Verbindungen untereinander, seien es Vererbungsbeziehungen oder Assoziationen.

Identifikation der Attribute

Wir müssen uns nun überlegen, welche Klassen welche Attribute benötigen. Das ist bei diesem Beispiel besonders einfach, es genügt uns das Attribut „Name“ bei der Klasse *Saeugetier*. Attribute werden im Klassendiagramm in der zweiten Rubrik einer Klasse eingetragen.

Durch Vererbung steht dann dieses Attribut auch den Unterklassen Mensch und Hund zur Verfügung.

Identifikation der Methoden

Zur Vervollständigung des Klassendiagrammes fehlen noch die Methoden. Aus den Überlegungen, was unser System einmal können soll ergeben sich zwanglos die erforderlichen Methoden:

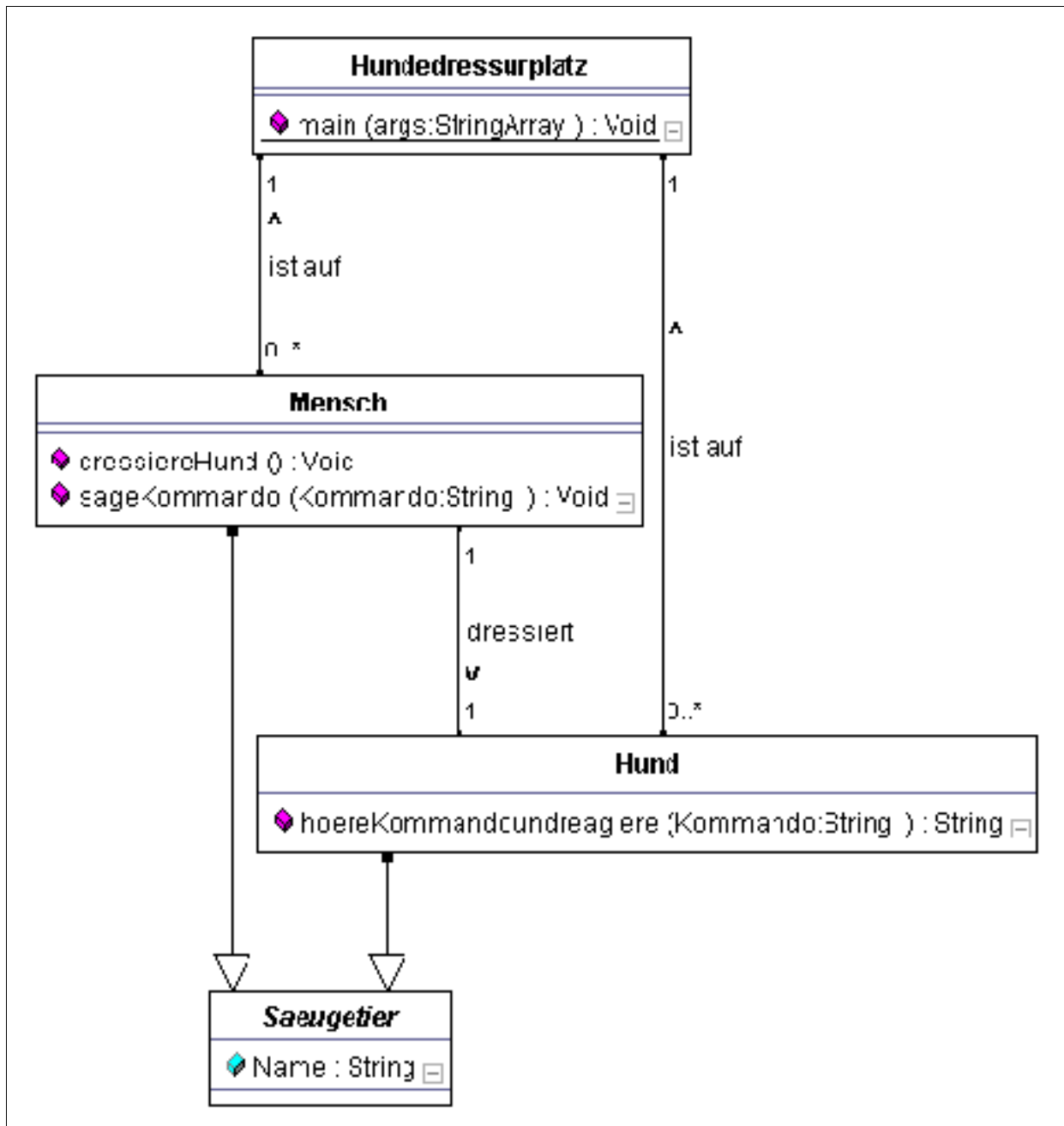
- Mensch-Objekt soll Hund-Objekt dressieren, also benötigen wir eine Methode „dressiereHund()“.
- Zur Dressur des Hundes werden Kommandos gesagt, also benötigen wir eine Methode „sageKommando()“.
- Der Hund soll die Kommandos hören und darauf reagieren, also benötigen wir eine Methode „hoereKommandoundreagiere()“.
- Zu guter Letzt noch die Methode „main()“ um das Programm starten zu können.

Natürlich müssen wir uns bei den Methoden auch Gedanken über ihren Typ und eventuelle Parameter machen:

- Die Methode „main()“ ist vom Typ „Void“ und erhält die üblichen Parameter.
- Die Methode „dressiereHund()“ ist ebenfalls vom Typ „Void“ und benötigt keine Parameter, sie steuert die Dressur.
- Die Methode „sageKommando()“ ist auch „Void“, hat aber als Parameter das „Kommando“, das dem Hund-Objekt gesagt wird, dieses ist natürlich vom Typ „String“.
- „hoereKommandoundreagiere()“ hat als Parameter das „Kommando“ und gibt einen String zurück, nämlich die Reaktion des Hund-Objektes.

Damit ist das Klassendiagramm vollständig modelliert. Methoden werden im Klassendiagramm in der dritten Rubrik bei den Klassen eingetragen.

Die weitere Arbeit steckt in der detaillierten Modellierung bzw. auch Programmierung der einzelnen Methoden.

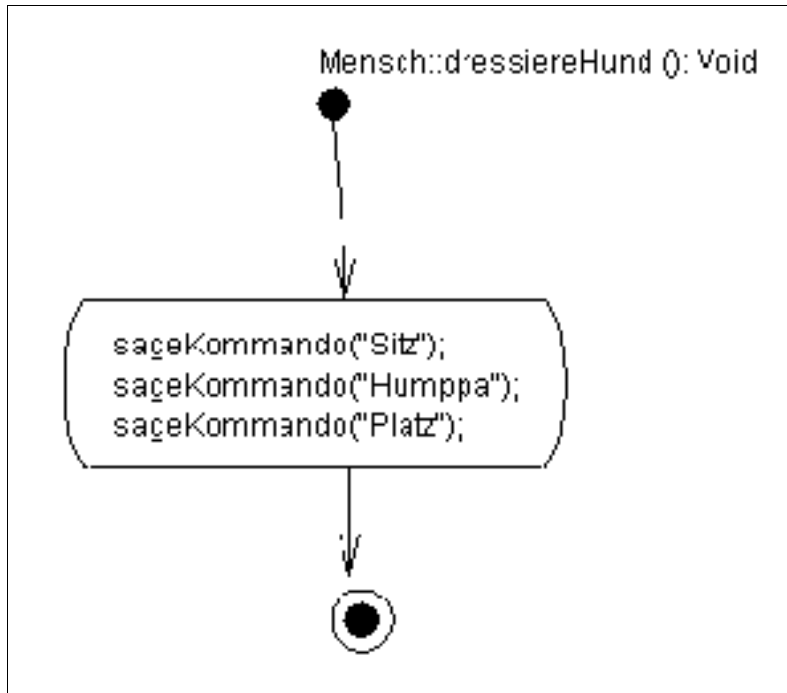


Modellierung und Programmierung der Methoden

Wir modellieren die Methoden in der Reihenfolge ihrer Komplexität. Die Modellierung einer Methode beginnt immer damit, dass im Klassendiagramm mit der rechten Maustaste genau auf einen Methodennamen geklickt wird. Im sich dann öffnenden Kontextmenü kann ein Aktivitätsdiagramm für diese Methode erstellt werden, in dem dann weitergearbeitet wird.

Methode „dressiereHund()“

Jedes Aktivitätsdiagramm hat einen Startpunkt und einen Stoppunkt. Dazwischen erfolgt die Modellierung der jeweiligen Aktivitäten. Die Methode „dressiereHund()“ ist sehr einfach

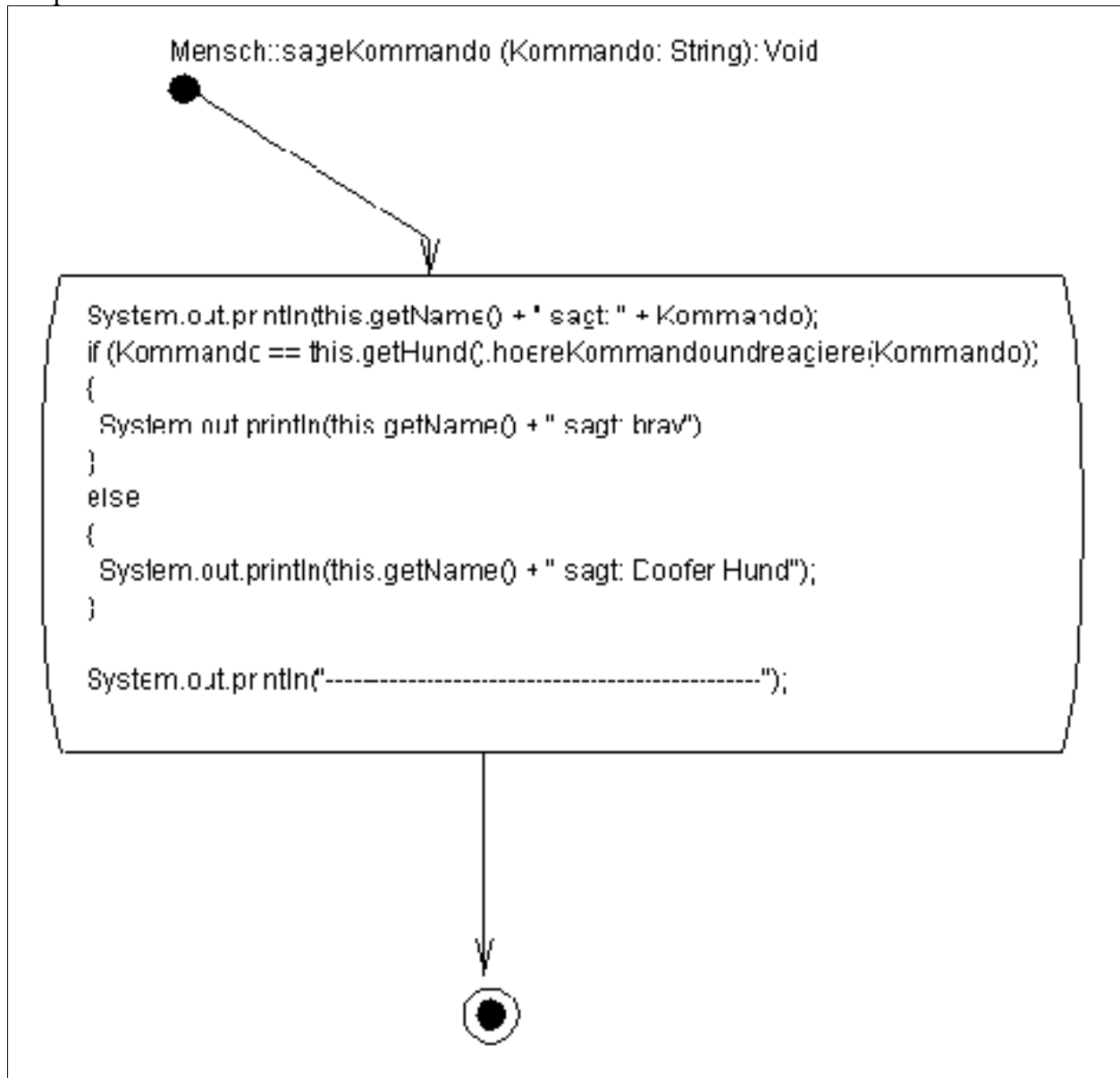


gehalten, weil die ganze Hundedressur nur aus drei verschiedenen, nacheinander gegebenen Kommandos besteht, wie das Aktivitätsdiagramm zeigt.

Methode „sageKommando()“

Die Methode „sageKommando()“ richtet die einzelnen Kommandos an die Hund-Objekte aus, die dann später zur Programmlaufzeit existieren. Dies geschieht, indem die Methode „hoereKommandoundreagiere()“ aufgerufen wird. Spannend ist hier aber die Frage, welches Hund-Objekt denn angesprochen wird? Sobald mehrere Mensch-Objekte und mehrere Hund-Objekte auf dem Dressurplatz unterwegs sind ist es ja wichtig, dass jedes Hund-Objekt nur auf die Kommandos seines eigenen Mensch-Objektes hört und nicht auf jedes Kommando, das irgendjemand gibt. Die Lösung ist „this.getHund()“. „getHund()“ ist eine Methode der Klasse Mensch, denn sie wird ja in der Methode „sageKommando()“ dieser Klasse aufgerufen. Das „this“ zeigt, dass das jeweilige Mensch-Objekt selbst gemeint ist, das während der Programmlaufzeit existiert. Wenn also das Mensch-Objekt Maier ein Hund-Objekt Bello hat, dann muss die Methode „this.getHund()“ des Objektes Maier den Wert Bello zurückliefern. Wenn das Mensch-Objekt Mueller ein Hund-Objekt Hasso hat, dann muss die Methode „this.getHund()“ des Objektes Mueller den Wert Hasso zurückliefern und genau das tut sie auch. „this.getHund()“ liefert das jeweils zugehörige Hund-Objekt zurück und diesem

Hund-Objekt wird die Botschaft „hoereKommandoundreagiere()“ gesendet. Antwort auf diese Botschaft ist die Reaktion des jeweiligen Hund-Objektes, worauf sich das Mensch-Objekt entsprechend verhält.

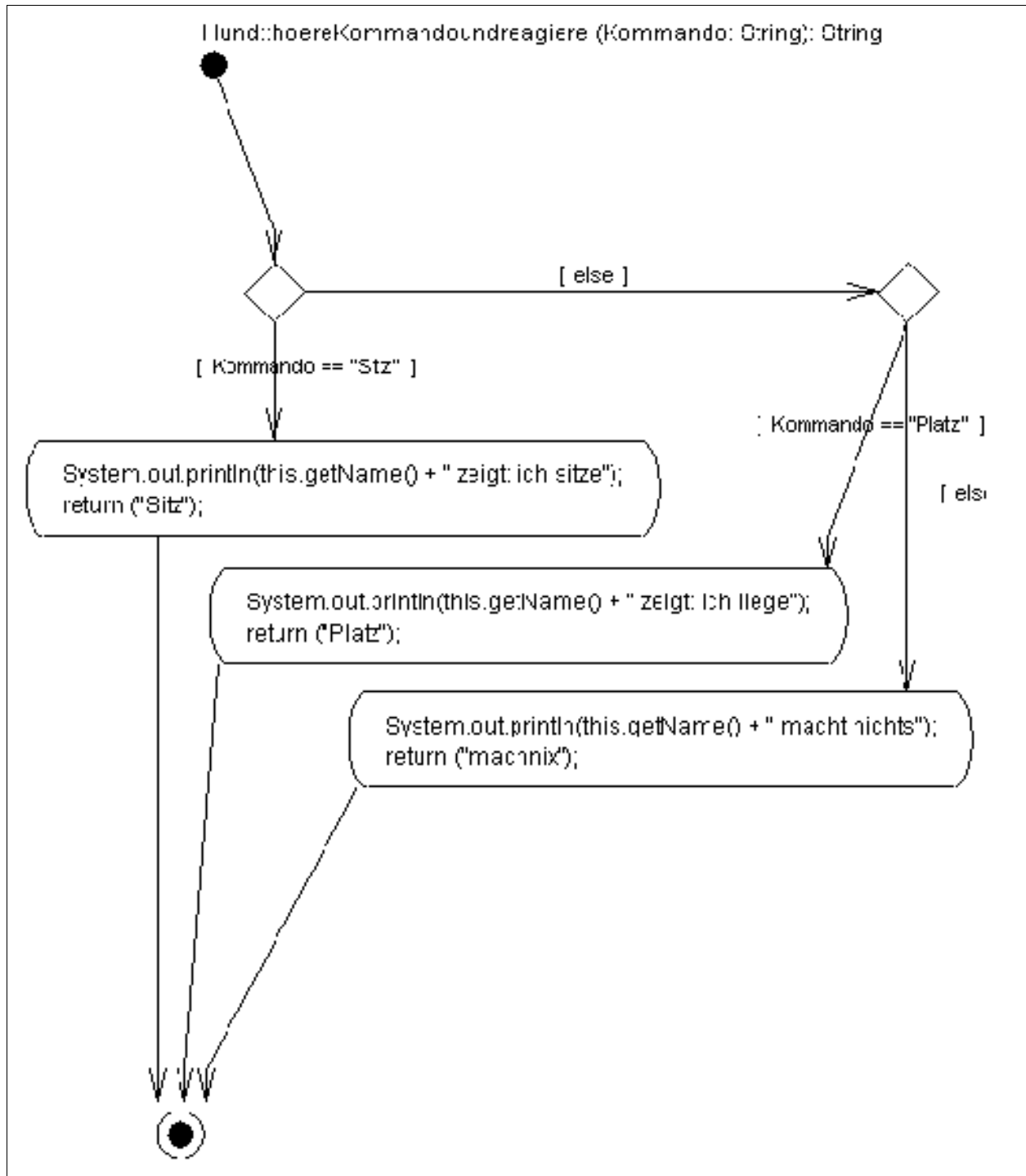


Es bleibt noch zu klären, woher die Methode „getHund()“ überhaupt stammt, denn wir haben sie nirgends manuell direkt als Methode eingegeben – folglich muss Fujaba sie selbstständig erzeugt haben. So ist es auch – und zwar auf Grund der Assoziation „dressiert“ zwischen den Klassen Mensch und Hund. Assoziationen im Klassendiagramm stellen also nicht nur optisch im Diagramm einen Zusammenhang zwischen Klassen her, sondern sorgen auch für den entsprechenden Niederschlag im Java-Quellcode, damit später zur Laufzeit die entsprechenden Objekte voneinander wissen können. Hier zeigt sich eindrucksvoll die Mächtigkeit von

modernen UML-Tools wie Fujaba, die dem Anwender mit ihrer leistungsfähigen Codegenerierung viel lästige Programmierarbeit abnehmen können.

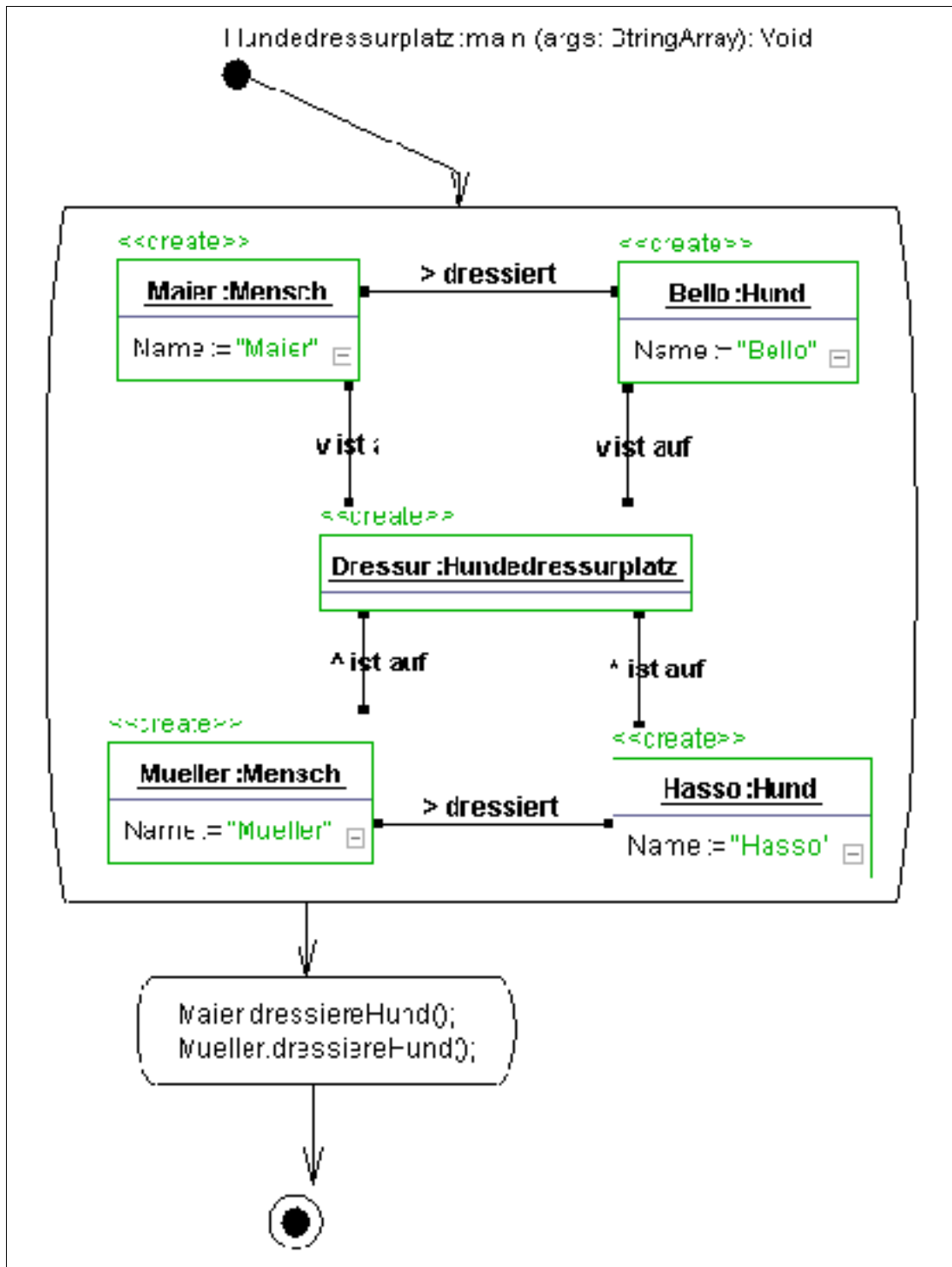
„Methode „hoereKommandoundreagiere()“

Die Methode „hoereKommandoundreagiere()“ haben wir einmal mehr grafisch orientiert modelliert, um auch die Möglichkeiten in diesem Bereich demonstrieren zu können. Hund-



Objekte verstehen also zwei verschiedene Kommandos, nämlich „Sitz“ und „Platz“. Diese Kommandos führen sie auch aus und teilen dies durch den Rückgabewert dem zugehörigen Mensch-Objekt mit.

Methode „main()“



Zum Schluss modellieren wir die zentrale „main()“-Methode, ohne die bekanntlich keine Java-Applikation auskommt. Die Verantwortlichkeiten dieser Methode sind schnell genannt: sie muss alle erforderlichen Objekte erzeugen und dann dafür sorgen, dass die Hund-Objekte von den Mensch-Objekten dressiert werden. Die Erzeugung der Objekte geschieht in einem Storydiagramm, das ist ein Diagramm innerhalb eines Aktivitätsdiagrammes. Storydiagramme sind eine Fujaba-Spezialität, die es im reinen UML nicht gibt. Wir erzeugen zwei Mensch-Objekte, Maier und Mueller, und zwei Hund-Objekte, Bello und Hasso. Zusätzlich erzeugen wir ein Hundedressurplatz-Objekt Dressur. Über sogenannte Assertions können wir Attribute der Objekte mit Werten belegen, was wir bei den ganzen Lebewesen auch machen: Wir belegen das Attribut „Name“ mit dem jeweiligen Namen des Lebewesens. Die Assoziationen zwischen den einzelnen Klassen, die wir ja im Klassendiagramm modelliert haben, erlauben uns im Storydiagramm, Links zwischen den einzelnen Objekten zu erzeugen. Der Name und die Richtung der Links ist natürlich genau so wie bei den jeweiligen Assoziationen.

Zu guter Letzt lassen wir die Dressur beginnen, indem wir den beiden Mensch-Objekten die entsprechende Botschaft schicken.

Programmlauf

Damit ist die Modellierung und Programmierung abgeschlossen. Wir lassen Fujaba den Java-Quellcode generieren, übersetzen ihn in Bytecode und lassen ihn ausführen. Die Ausgabe ist dann:

```
Maier sagt: Sitz
```

```
Bello zeigt: ich sitze
```

```
Maier sagt: brav
```

```
-----
```

```
Maier sagt: Humppa
```

```
Bello macht nichts
```

```
Maier sagt: Doofer Hund
```

```
-----
```

```
Maier sagt: Platz
```

```
Bello zeigt: ich liege
```

```
Maier sagt: brav
```

```
-----
```

```
Mueller sagt: Sitz
```

Hasso zeigt: ich sitze

Mueller sagt: brav

Mueller sagt: Humppa

Hasso macht nichts

Mueller sagt: Doofer Hund

Mueller sagt: Platz

Hasso zeigt: ich liege

Mueller sagt: brav

Das Verhalten der Mensch-Objekte und der Hund-Objekte ist wie gewünscht. Zu beachten ist auch, dass wie weiter oben ausführlich erläutert, tatsächlich jeder Hund auf sein eigenes Herrchen hört.

Zusammenfassung und Schluss

Damit ist dieses anschauliche Beispiel für eine eingängige und auch von Einsteigern nachvollziehbaren Einführung in die objektorientierte Modellierung und objektorientierte Programmierung an seinem Ende angelangt. Selbstverständlich ist dieses Beispiel nicht von besonderer Tiefgründigkeit und es hat im Grunde auch keinen praktischen Nutzen. Seine Aufgabe, wesentliche Charakteristika der Objektorientierung deutlich herauszustellen, ohne durch unnötige Komplexität oder Abstraktheit unnötig zu belasten, kann es aber hervorragend erfüllen. Damit sind die Grundlagen geschaffen, auf denen aufbauend weitere Projekte erfolgreich in Angriff genommen werden können.

Es ist auch problemlos möglich, dieses Beispiel noch etwas auszubauen, um z.B. im Rahmen einer Binnendifferenzierung besonders leistungsfähigen Schülern Material für weitere Arbeiten zu liefern:

- Der Hund könnte ein Kommando nur mit einer gewissen Wahrscheinlichkeit korrekt ausführen.
- Der Hund könnte als Belohnung ein Belohnung-Objekt (Hundekuchen) erhalten und darauf reagieren.
- Nach einer erhaltenen Belohnung könnte die Wahrscheinlichkeit, mit der das nächste Kommando korrekt ausgeführt wird, zunehmen.

Weitere Informationen und Materialien speziell zu diesem Beispiel finden sich online unter <http://www.LmTM.de/InformatiXTM/umljava/texte/hundedressur1.html>

Dort liegen auch die Fujaba-Projektdatei, die Java-Quellcode- und class-Dateien, usw..

Eine Online-Lernumgebung rund um UML und Java findet sich unter <http://www.LmTM.de> und dort unter „InformatiXTM“ und dort unter „UML/Java“.

Für alle Interessierten steht auch ein Diskussionsforum zur Verfügung, wo alles zu diesem Themenbereich online diskutiert werden kann:

http://www.LmTM.de/w-agera/index.php?bn=lmtm_umljava