Hamstern mit BlueJ

Dr. Dietrich Boles

Universität Oldenburg

01.02.2006

1. Einleitung

Ziel dieses Artikels ist die Beschreibung der Integration von ***BlueJ*** und dem ***Java-Hamster-Modell***, zwei existierenden Ansätzen zur Vermittlung objektorientierter Programmierkonzepte an Programmieranfänger.

BlueJ

BlueJ ist eine Entwicklungsumgebung für objektorientierte Java-Programme, die speziell für Programmieranfänger entworfen wurde. Im Einzelnen werden folgende Werkzeuge zur Verfügung gestellt:

- Ein Fenster, das die Programmstruktur graphisch visualisiert und es ermöglicht, interaktiv Objekte zu erzeugen und für diese Befehle aufzurufen,
- ein Editor zum Erstellen neuer Klassen,
- ein Compiler zum Compilieren von Programmen,
- ein Debugger zur Fehlersuche in Programmen
- und eine Direkteingabe zur interpretativen Ausführung einzelner Java-Anweisungen.

Mit der Entwicklungsumgebung einher geht eine didaktische Methode zur Einführung in die objektorientierte Programmierung. Ihr zugrunde liegt ein iteratives Vorgehen bei der Einführung der Konzepte der objektorientierten Programmierung, das unter dem Motto „Objekte zuerst“ steht.

Eine der großen Stärken von BlueJ ist die Möglichkeit des interaktiven Erzeugens von Objekten und des interaktiven Umgangs mit diesen. Eine weitere Stärke ist die Visualisierung der Programmstruktur durch Diagramme.

Weitere Informationen zu BlueJ finden sich auf der Website www.bluej.org sowie im Buch „**Objektorientierte Programmierung mit Java**“ (David J. Barnes und Michael Kölling, Peason Studium).

Java-Hamster-Modell

Genauso wie BlueJ richtet sich auch das Java-Hamster-Modell speziell an Programmieranfänger. Beim Java-Hamster-Modell handelt es sich um ein spezielles didaktisches Modell, das Programmieranfängern einen spielerischen Zugang zu der doch eher technischen Welt der Programmierung bietet. Programmieranfänger lernen die grundlegenden Programmierkonzepte und den Programmentwurf kennen, indem sie so genannte Hamster-Programme entwickeln, mit denen sie virtuelle

Hamster durch eine virtuelle Landschaft steuern und bestimmte Aufgaben bzw. Probleme lösen lassen. Dazu gibt es den so genannten Hamster-Simulator, der es ermöglicht, Hamster-Programme zu entwickeln und auszuführen, wobei man die Hamster bei der Erledigung ihrer Aufgaben mit Hilfe einer graphischen Umgebung beobachten kann. Aktuell besteht das Java-Hamster-Modell aus zwei Teilen. Im ersten Teil geht es um die imperative Programmierung, im zweiten Teil um die objektorientierte Programmierung. Relevant für diese Arbeit ist der zweite Teil.

Genauso wie bei BlueJ werden auch im Java-Hamster-Modell die wesentlichen Konzepte der Programmierung inkrementell und iterativ eingeführt. Da in Java die Konzepte der objektorientierten Programmierung jedoch auf den Konzepten der imperativen Programmierung aufbauen, werden letztere zuerst behandelt, und zwar in dem ersten Band des Java-Hamster-Buches „**Programmieren spielend gelernt mit dem Java-Hamster-Modell**“ (Dietrich Boles, Teubner-Verlag). Die objektorientierte Programmierung ist Thema des zweiten Bandes „**Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell**“ (Dietrich Boles und Cornelia Boles, Teubner-Verlag).

Die besondere Stärke des Java-Hamster-Modells insbesondere in Verbindung mit dem Hamster-Simulator liegt in der Visualisierung der Ausführung eines Programms. Der Programmierer sieht von Anfang an in einer ansprechenden Umgebung, was seine Programme bewirken.

Weitere Informationen zum Java-Hamster-Modell finden sich auf der Website www.java-hamster-modell.de sowie in den beiden genannten Büchern.

Ziel der Arbeit

Ziel des in diesem Artikel beschriebenen Projektes war die Kombination der beiden Ansätze, genauer die Integration des Hamster-Simulators in BlueJ. Mit den bereit gestellten Hilfsmitteln ist es nun möglich, Hamster-Programme mit den Werkzeugen und Möglichkeiten, die BlueJ bietet, zu entwickeln und im Hamster-Simulator ausführen zu lassen. Konkret bedeutet das an Vorteilen für Programmieranfänger, BlueJ visualisiert die Programmstruktur und erlaubt insbesondere die interaktive Erzeugung von Hamstern und den interaktiven Aufruf von Hamster-Befehlen und der Hamster-Simulator visualisiert die Programmausführung, d.h. der Programmierer sieht unmittelbar in einer graphischen Umgebung, was seine Anweisungen bzw. Programme bewirken.

2. Installation

Voraussetzung des „Hamsterns mit BlueJ“ ist die Installation von BlueJ (ab Version 2.1.0) und die Installation des Hamster-Simulators (ab Version 2.2). BlueJ können Sie sich von der Website www.bluej.org herunter laden. Die Installation wird im BlueJ-Tutorial beschrieben, das über den URL http://www.bluej.org/tutorial/blueJ-tutorial-deutsch.pdf zugänglich ist. Der Hamster-Simulator steht auf der Website www.java-hamster-modell.de zum Download zur Verfügung. Die Installation wird in Kapitel 2 des Benutzerhandbuchs beschrieben (siehe auch http://www-is.informatik.uni-oldenburg.de/~dibo/hamster/download/v22/handbuch.pdf).

Was ist anschließend noch zu tun? Im Folgenden ist mit `<BlueJ-Ordner>` der Ordner gemeint, in den Sie BlueJ installiert haben und `<Hamster-Simulator-Ordner>` ist der Ordner, in den Sie den Hamster-Simulator installiert haben. Kopieren (nicht verschieben!) Sie anschließend die beiden Dateien „`<Hamster-Simulator-Ordner>/hamstersimulator.jar`“ und „`<Hamster-Simulator-Ordner>/tools.jar`“ in den Ordner „`<BlueJ-Ordner>/lib/userlib`“. Das ist alles, was Sie zu tun haben.

3. Erste Schritte

Wenn Sie noch nie mit BlueJ gearbeitet haben, machen Sie sich zunächst mit BlueJ vertraut. Am besten, Sie lesen sich das komplette BlueJ-Tutorial (ca. 40 Seiten) durch. Außerdem stehen im Ordner „`<BlueJ-Ordner>/examples`“ einige Beispielprojekte zum Ausprobieren zur Verfügung.

Öffnen Sie anschließend einfach mal das Projekt `hamster` aus dem Ordner „`<Hamster-Simulator-Ordner>/HamsternMitBlueJ/beispiele`“. Abbildung 1 zeigt das entstehende BlueJ-Fenster. Erzeugen Sie dann ein Hamster-Objekt mit dem Konstruktor mit den vier `int`-Parametern. Geben Sie dabei für alle vier Parameter den Wert `1` ein. Es öffnet sich das Simulationsfenster des Hamster-Simulators (siehe auch Abbildung 1). Rufen Sie anschließend für das Hamster-Objekt die Methode `void vor()` auf. Der rote Hamster hüpft dadurch eine Kachel nach vorne. Probieren Sie einfach mal aus, was sie sonst noch alles mit dem Hamster machen können.

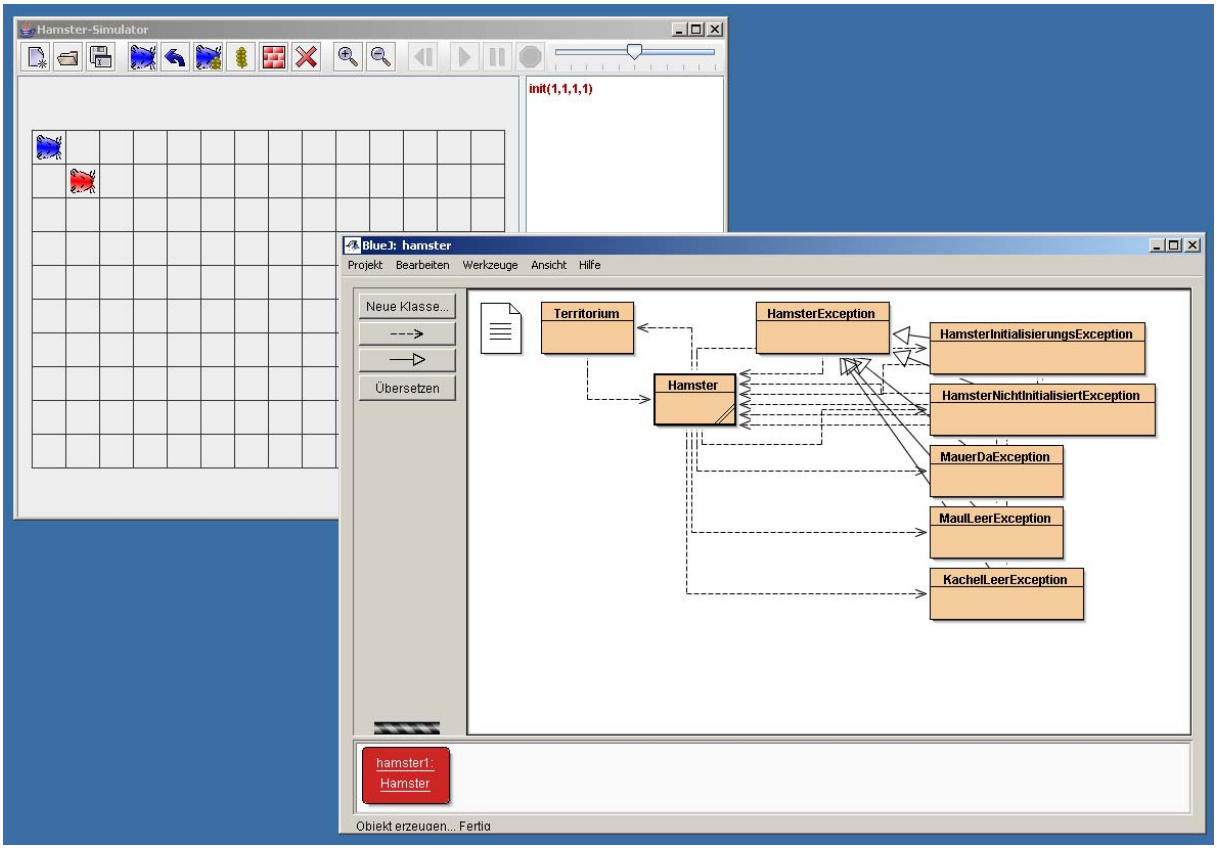

Abbildung 1: Graphische Oberfläche beim Hamstern mit BlueJ

4. Grundlagen des Java-Hamster-Modell

Die Grundidee des Java-Hamster-Modells ist ausgesprochen einfach: Sie als Programmierer müssen (virtuelle) Hamster durch eine (virtuelle) Landschaft steuern und sie gegebene Aufgaben lösen lassen.

Landschaft

Die Landschaft, in der die Hamster leben, wird durch eine gekachelte Ebene repräsentiert. Abbildung 2 zeigt eine typische Hamsterlandschaft - auch Hamster-Territorium genannt - inklusive Legende. Die Größe der Landschaft, d.h. die Anzahl der Kacheln, ist prinzipiell beliebig aber nie unendlich.

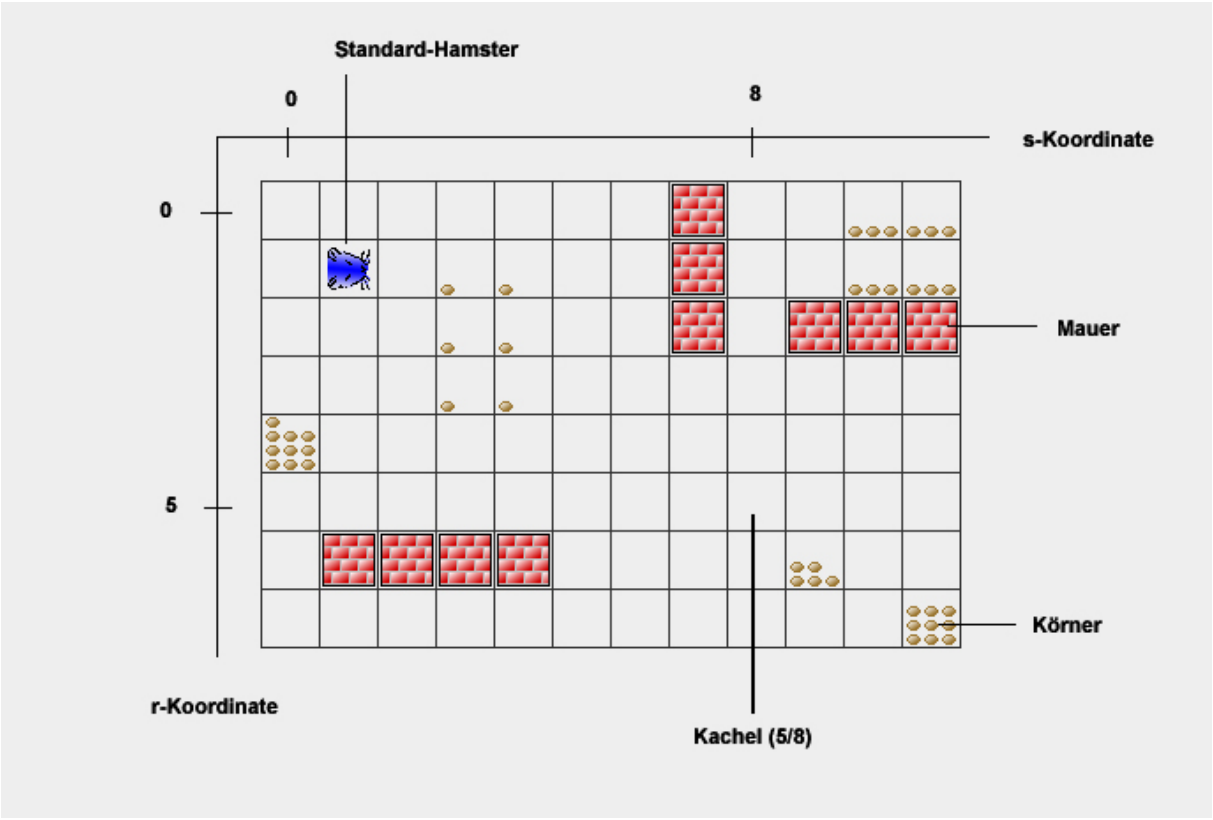

Abbildung 2: Hamster-Territorium mit Legende

Auf einzelnen Kacheln können kein, ein oder mehrere Körner liegen. Kacheln, auf denen sich Körner befinden, sind in den Landschaftsskizzen entsprechend gekennzeichnet.

Auf den Kacheln des Hamster-Territoriums können weiterhin auch Mauern stehen, was bedeutet, dass diese Kacheln blockiert sind. Die Hamster können sie nicht betreten. Es ist nicht möglich, dass sich auf einer Kachel sowohl eine Mauer als auch Körner befinden. Das Territorium ist immer vollständig von Mauern umgeben.

Damit jede Kachel des Hamster-Territoriums eindeutig identifiziert werden kann, ist dem Hamster-Territorium ein Koordinatensystem zugeordnet. Jede Kachel ist in einer bestimmten Reihe und Spalte platziert. Um Kacheln benennen zu können, bekommt jede Kachel eine r-Koordinate (r = Reihe) und eine s-Koordinate (s = Spalte). Die

Koordinaten sind natürliche Zahlen und beginnen bei 0. Die r-Koordinate wird nach unten pro Kachel um eine Zahl größer und die s-Koordinate wächst nach rechts hin. Die Kachel in der linken oberen Ecke hat damit die Position (r=0/s=0) -- kurz (0/0). In Abbildung 2 ist als Beispiel die Kachel (5/8) markiert.

Hamster

Jeder Hamster steht auf einer der Kacheln des Hamster-Territoriums. Es ist durchaus möglich, dass sich auf einer Kachel mehrere Hamster aufhalten. Diese Kachel darf nicht durch eine Mauer blockiert sein, sie kann jedoch Körner enthalten.

Die Hamster können in vier unterschiedlichen Blickrichtungen (Nord, Süd, West, Ost) auf den Kacheln stehen. Je nach Blickrichtung werden die Hamster durch unterschiedliche Symbole repräsentiert. Hamster können weiterhin kein, ein oder mehrere Körner im Maul haben.

Mit Hilfe bestimmter Befehle, die gleich noch genauer erläutert werden, kann ein Programmierer die Hamster durch ein gegebenes Hamster-Territorium steuern. Die Hamster können dabei von Kachel zu Kachel hüpfen, sie können sich drehen, Körner fressen und Körner wieder ablegen. Sie können sich die Hamster quasi als virtuelle Prozessoren vorstellen, die im Gegensatz zu realen Computer-Prozessoren (zunächst) keine arithmetischen und logischen Operationen ausführen können, sondern in der Lage sind, mit einem kleinen Grundvorrat an Befehlen ein Hamster-Territorium zu „erkunden“.

Hamster-Befehle

Die Aufgabe eines Hamster-Programmierers besteht darin, Hamster durch eine Landschaft zu steuern, um dadurch gegebene Hamster-Aufgaben zu lösen. Zur Steuerung der Hamster müssen diesen Anweisungen in Form von Befehlen gegeben werden. Hamster besitzen dabei die Fähigkeit, folgende Befehle (Methoden) zu verstehen und auszuführen:

- `void vor()`: Der Hamster hüpft eine Kachel in seiner aktuellen Blickrichtung nach vorne.
- `void linksUm()`: Der Hamster dreht sich auf der Kachel, auf der er gerade steht, um 90 Grad entgegen dem Uhrzeigersinn.
- `void nimm()`: Der Hamster frisst von der Kachel, auf der er sich gerade befindet, genau ein Korn, d.h. anschließend hat der Hamster ein Korn mehr im Maul und auf der Kachel liegt ein Korn weniger als vorher.
- `void gib()`: Der Hamster legt auf der Kachel, auf der er sich gerade befindet, genau ein Korn aus seinem Maul ab, d.h. er hat anschließend ein Korn weniger im Maul, und auf der Kachel liegt ein Korn mehr als vorher.
- `void schreib(String zeichenkette)`: Der Hamster gibt die als Parameter übergebene Zeichenkette auf den Bildschirm aus.
- `String liesZeichenkette(String aufforderung)`: Der Hamster gibt die als Parameter übergebene Zeichenkette auf den Bildschirm aus und fordert den Benutzer auf, eine Zeichenkette über die Tastatur einzugeben.

- `int liesZahl(String aufforderung)`: Der Hamster gibt die als Parameter übergebene Zeichenkette auf den Bildschirm aus und fordert den Benutzer auf, eine Zahl über die Tastatur einzugeben.

Bei den Befehlen `void vor()`, `void nimm()` und `void gib()` können Probleme auftreten, die sich in Laufzeitfehlern (Exceptions) äußern:

- Ein Hamster bekommt den Befehl `void vor()` und die Kachel in Blickrichtung vor ihm ist durch eine Mauer blockiert.
- Ein Hamster bekommt den Befehl `void nimm()` und auf der Kachel, auf der er sich gerade befindet, liegt kein einziges Korn.
- Ein Hamster bekommt den Befehl `void gib()` und er hat kein einziges Korn im Maul.

Um derartige Laufzeitfehler zu vermeiden, existieren drei spezielle Testbefehle:

- `boolean vornFrei()`: Liefert den Wert `true`, falls sich auf der Kachel in Blickrichtung vor dem Hamster keine Mauer befindet. Ist die Kachel durch eine Mauer blockiert, dann wird der Wert `false` geliefert.
- `boolean maulLeer()`: Liefert den Wert `false`, falls der Hamster ein oder mehrere Körner im Maul hat. Befinden sich keine Körner im Maul des Hamsters, dann wird der Wert `true` geliefert.
- `boolean 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 wird der Wert `false` geliefert.

Es existieren einige weitere Hamster-Befehle, die jedoch zunächst weniger wichtig sind. Bspw. gibt es Befehle, mit denen die aktuellen Zustandswerte (Reihe, Spalte, Blickrichtung, Anzahl an Körnern im Maul) eines Hamsters abgefragt werden können.

5. Besonderheiten und Details des Java-Hamster-Modells

Es gibt einen Hamster, der immer im Territorium sitzt, ohne ihn explizit erzeugen zu müssen: der so genannte Standard-Hamster (siehe auch Abbildung 2). Er wird immer blau dargestellt. Um ihm Befehle erteilen zu können, müssen Sie in BlueJ die Klassenmethode (static-Methode) `Hamster getStandardHamster()` der Klasse `Hamster` aufrufen und dann den Button „`hole`“ bzw. „`get`“ anklicken.

Die Klasse `Hamster` stellt drei Konstruktoren zur Verfügung. Am häufigsten werden Sie wahrscheinlich den Konstruktor mit den vier `int`-Parametern nutzen. Der erste Parameter gibt die Reihe an, in der der Hamster platziert werden soll, der zweite die Spalte, der dritte die Blickrichtung und der vierte die anfängliche Anzahl an Körnern im Maul des Hamsters. Nutzen Sie für die Angabe der Blickrichtung am besten die Konstanten `Hamster.OST`, `Hamster.NORD`, `Hamster.SUED` bzw. `Hamster.WEST`. Der Konstruktor mit dem einen `Hamster`-Parameter kann dazu genutzt werden, einen neuen Hamster mit den Zustandswerten eines bereits existierenden Hamsters zu erzeugen. Weiterhin existiert ein parameterloser Konstruktor. Wenn Sie diesen aufrufen, wird ein nicht-initialisierter Hamster erzeugt, der auch nicht im Territorium erscheint. Bevor Sie für einen nicht-initialisierten Hamster irgendwelche Befehle aufrufen, muss dieser mit dem Befehl `void init(int reihe, int spalte, int blickrichtung, int anzahlKoerner)` initialisiert werden. Ansonsten

kommt es zu einem Laufzeitfehler, d.h. es wird eine `HamsterNichtInitialisiertException` geworfen. Die Parameter des Befehls `void init(int reihe, int spalte, int blickrichtung, int anzahlKoerner)` entsprechen dabei den Parametern des Konstruktors mit den vier `int`-Parametern.

Ein Hamster-Befehl, der nicht im Original-Java-Hamster-Modell existiert, ist der Befehl `void loeschen()`. Er führt dazu, dass der Hamster in den Zustand „nicht-initialisiert“ zurückversetzt wird und aus dem Territorium verschwindet. Durch Aufruf des Befehls `void init(int reihe, int spalte, int blickrichtung, int anzahlKoerner)` kann der Hamster wieder reanimiert werden.

Neben der Klasse `Hamster` gibt es eine Klasse namens `Territorium`. Von dieser Klasse können Sie keine Objekte erzeugen. Vielmehr stellt die Klasse einige nützliche Klassenmethoden zur Verfügung, um die Beschaffenheit des aktuellen Territoriums abfragen zu können.

Eine Methode der Klasse `Territorium`, die im Original-Java-Hamster-Modell nicht existiert, ist die Methode `void ladeTerritorium(String dateiName)`. Hiermit können Sie ein Territorium, das Sie zuvor mit dem Hamster-Simulator erstellt und gespeichert haben, laden. Dabei wird jedoch automatisch für alle gerade existierenden Hamster außer dem Standard-Hamster die Methode `void loeschen()` aufgerufen, d.h. die Hamster verschwinden aus dem Territorium.

6. Bedienung des Hamster-Simulators in BlueJ

Wenn Sie den Original-Hamster-Simulator kennen, wird Ihnen der Hauptunterschied zum Hamster-Simulator in BlueJ bereits aufgefallen sein: Das Simulationsfenster ist da, es fehlt jedoch das Editor-Fenster, in dem Sie Hamster-Programme schreiben und debuggen können. Für diese Aufgaben können Sie jedoch nun die entsprechenden BlueJ-Tools nutzen.

Die Bedienung des Simulationsfensters ist eigentlich intuitiv. Probieren Sie es am besten einfach mal aus. Die genaue Funktionalität ist sehr ausführlich im Benutzungshandbuch des Hamster-Simulators beschrieben. Dort können Sie bei Bedarf nachschauen (Datei „`<Hamster-Simulator-Ordner>/handbuch.pdf`“).

7. Beispiele zum Hamstern mit BlueJ

Im Ordner „`<Hamster-Simulator-Ordner>/HamsternMitBlueJ/beispiele`“ finden Sie einige Beispiele für BlueJ-Projekte mit dem Java-Hamster, die im Folgenden beschrieben werden.

Die Beispiele wurden so entwickelt, dass sie sowohl unter Java 1.4 als auch unter Java 5 compilierfähig und lauffähig sind. Wenn Sie die Beispiele daher mit Java 5 compilieren, erscheint eine Warnung. Diese können Sie unterbinden, wenn Sie in BlueJ im Menü „Werkzeuge → Einstellungen → Diverses“ die Markierung aus dem Kästchen „show compiler warnings when unsafe collctions are used“ entfernen.

BlueJ-Projekt „hamster“

Dieses Projekt stellt die Grundlagen des „Hamsterns mit BlueJ“ zur Verfügung. Wenn Sie eigene Projekte erstellen wollen, müssen Sie dieses Projekt in ein neues Projekt kopieren und dann erweitern.

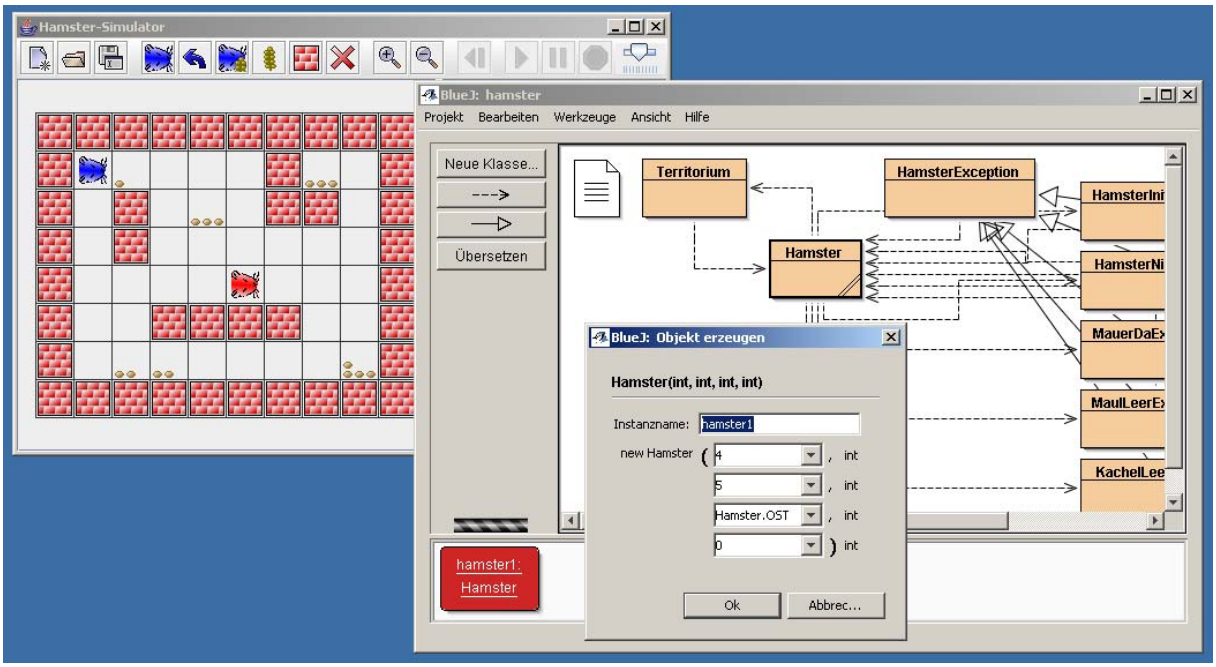

Abbildung 3: BlueJ-Projekt „hamster“

Wenn Sie zum ersten Mal mit BlueJ und dem Java-Hamster-Modell arbeiten, laden Sie zunächst dieses Projekt. Erzeugen Sie neue Hamster. Steuern Sie die Hamster durch Aufruf deren Methoden/Befehle durch ein Territorium. Probieren Sie mal die Klassenmethoden von der Klasse `Territorium` aus. Um die Exception-Klassen müssen Sie sich eigentlich nicht kümmern. Die sind was für fortgeschrittene Hamster-Programmierer.

Nutzungsvorschlag 1: Laden Sie zunächst das bereits vordefinierte Territorium „feld1“. Rufen Sie dazu die Klassenmethode `void ladeTerritorium(String dateiName)` der Klasse `Territorium` auf und geben Sie als Parameter die Zeichenkette `“Territorien/feld1“` ein. Es erscheint das Territorium wie in Abbildung 3 demonstriert. Erzeugen Sie dann einen Hamster auf der Kachel (4/5) mit Blickrichtung OST und 0 Körnern im Maul. Steuern Sie diesen Hamster mit Hilfe der Hamster-Methoden durch das Territorium und lassen Sie ihn alle Körner auffressen.

Nutzungsvorschlag 2: Laden Sie zunächst das vordefinierte Territorium „feld2“. Rufen Sie dazu die Klassenmethode der `void ladeTerritorium(String dateiName)` Klasse `Territorium` auf und geben Sie als Parameter die Zeichenkette `“Territorien/feld2“` ein. Holen Sie sich dann mittels der Klassenmethode `Hamster getStandardHamster()` der Klasse `Hamster` den Standard-Hamster als BlueJ-Objekt und lassen Sie diesen den Gipfel des Berges erklimmen, wobei der Hamster auf jeder Stufe ein Korn ablegen soll.

BlueJ-Projekt „bergsteigerhamster“

Dieses Projekt demonstriert, wie Sie neue erweiterte Hamster-Klassen definieren und nutzen können. Konkret geht es in diesem Beispiel-Projekt um Hamster, die einen Berg erklimmen sollen. Eingesetzt wird hierbei das Prinzip der Vererbung. Es wird eine Klasse `BergsteigerHamster` definiert, und zwar dadurch, dass sie von der Klasse `Hamster` abgeleitet wird. Hamstern, die von der Klasse `BergsteigerHamster` erzeugt werden, stehen damit neben den neu definierten Methoden auch weiterhin automatisch alle Methoden der Klasse `Hamster` zur Verfügung. Führen Sie einfach mal in BlueJ einen Doppelklick auf das Kästchen `BergsteigerHamster` aus und schauen Sie sich die Klassendefinition an.

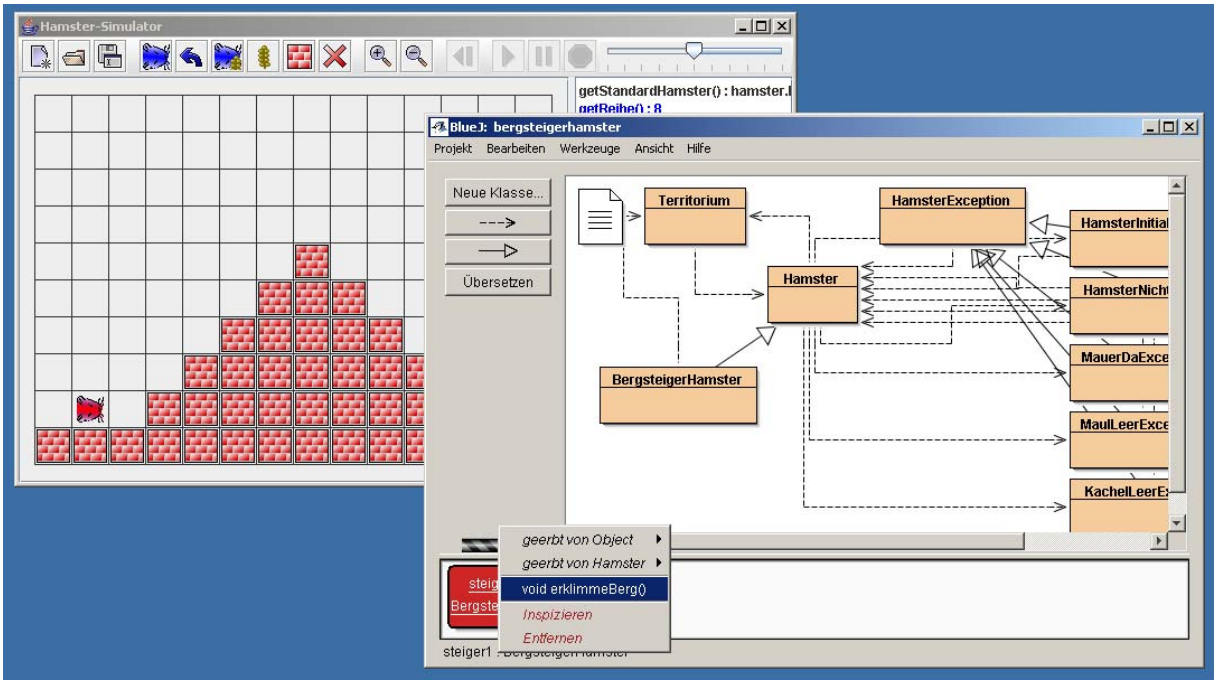

Abbildung 4: BlueJ-Projekt „bergsteigerhamster“

Nutzungsvorschlag 1: Erzeugen Sie mit dem Konstruktor `BergsteigerHamster(String dateiName)` einen Hamster der Klasse `BergsteigerHamster`. Geben Sie als String `“Territorien/berg1“` ein. Es wird das in dieser Datei vordefinierte Territorium geladen und ein BergsteigerHamster mit denselben Werten erzeugt, wie sie der Standard-Hamster aktuell besitzt. Rufen Sie dann für den Hamster die Methode `void erklimmeBerg()` auf (siehe Abbildung 4).

Nutzungsvorschlag 2: Erzeugen Sie mit dem Konstruktor `BergsteigerHamster(int reihe, int spalte, int blickrichtung, int anzahlKoerner, String dateiName)` einen Hamster der Klasse `BergsteigerHamster`. Geben Sie für die vier `int`-Parameter die Werte `1`, `1`, `Hamster.OST` und `0` ein und als String `“Territorien/berg2“`. Das in dieser Datei vordefinierte Territorium wird geladen und der Hamster im Territorium erzeugt. Steuern Sie dann den Hamster durch Nutzung der von der Klasse Hamster geerbten Methoden zum Fuß des Berges und rufen Sie dort die Methode `void erklimmeBerg()` auf.

BlueJ-Projekt „allroundhamster“

Nachdem Sie ein wenig mit den Hamstern experimentiert haben, werden Sie schnell feststellen, dass es bald nervig ist, jedes Mal erneut bspw. dreimal die Methode `void linksUm()` aufzurufen bzw. eine Methode `void rechtsUm()` zu definieren, um einen Hamster nach rechts umdrehen zu können. Aber dazu gibt es ja das Prinzip der Vererbung. Und wie man dieses nutzen kann, wird in diesem Projekt demonstriert. Was dieses Projekt nämlich definiert, ist eine Klasse `AllroundHamster`, die von der Klasse `Hamster` abgeleitet ist und viele nützliche weitere Befehle bereit stellt. Wenn Sie einen Hamster von der Klasse `AllroundHamster` erzeugen, können Sie für diesen Hamster weiterhin alle Hamster-Befehle aufrufen, aber zusätzlich die neuen Befehle. Erzeugen Sie einfach mal einen AllroundHamster und schauen Sie sich dessen Möglichkeiten, d.h. Methoden bzw. Befehle an (siehe auch Abbildung 5).

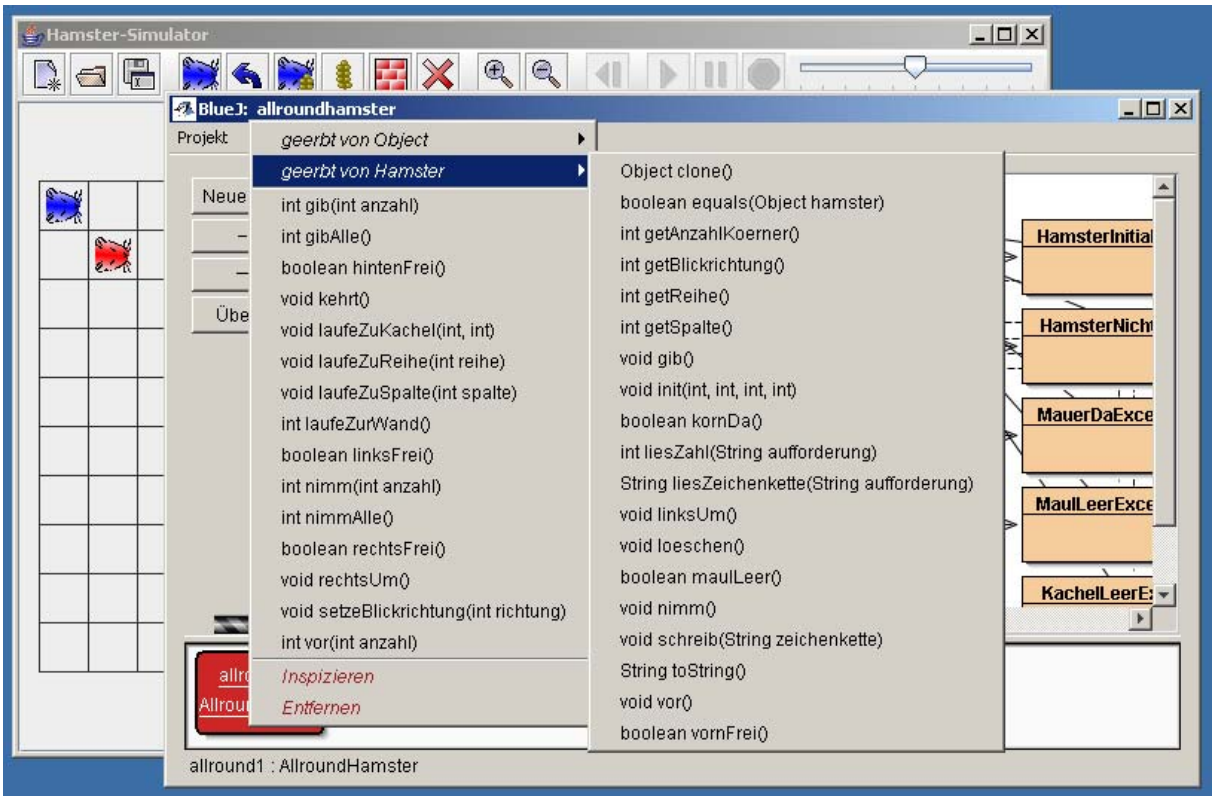

Abbildung 5: BlueJ-Projekt „allroundhamster“

Dieses BlueJ-Hamster-Projekt dient übrigens für alle im Folgenden noch aufgeführten BlueJ-Hamster-Projekte als Grundlage, d.h. sie wurden nicht durch Kopieren des BlueJ-Projektes „`hamster`“ sondern durch Kopieren des BlueJ-Projektes „`allroundhamster`“ erzeugt und können somit die Klasse `AllroundHamster` nutzen.

BlueJ-Projekt „nimmspiel“

Dieses BlueJ-Hamster-Projekt ist schon ein wenig komplexer. Es bietet den Hamstern die Möglichkeit, das so genannte Nimm-Spiel zu spielen. Es ist ein Spiel für zwei Hamster. Der Standard-Hamster fungiert quasi als Schiedsrichter. Er markiert den Ausgangspunkt eines Spiels. Vor ihm liegt eine Reihe mit Körnern, auf jeder Kachel jeweils ein Korn (vergleiche Abbildung 6). Bei Spielbeginn müssen sich die beiden spielenden Hamster zunächst zum Standard-Hamster begeben. Dann müssen sie die Körnerreihe ablaufen und abwechselnd jeweils entweder ein oder zwei Körner nehmen. Sie wissen dabei, wie lang die Körnerreihe ist. Wer das letzte Korn der Reihe nimmt, gewinnt das Spiel.

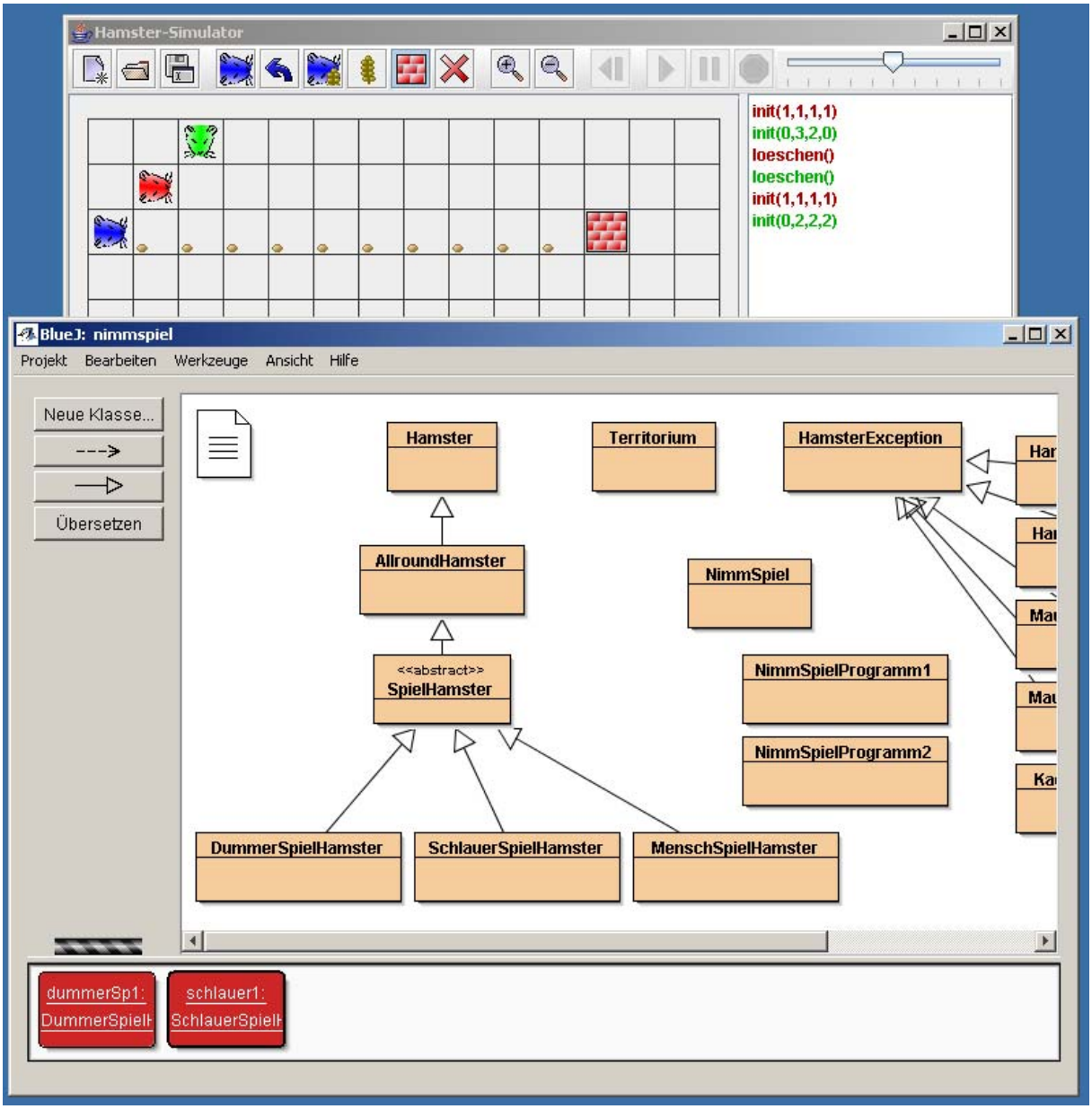

Abbildung 6: BlueJ-Projekt „nimmspiel“

Nutzungsvorschlag 1: Um einen Eindruck von dem Nimm-Spiel zu bekommen, rufen Sie einfach zunächst mal die Klassenmethode `void nimmSpielen()` der Klasse `NimmSpielProgramm1` auf und schauen Sie, was im Territorium passiert.

Nutzungsvorschlag 2: Erzeugen Sie zwei Objekte der Klasse `MenschSpielHamster`. Wo die beiden Hamster im Territorium platziert werden, ist egal. Erzeugen Sie dann ein Objekt der Klasse `NimmSpiel`. Rufen Sie für das Objekt die Methode `void spielen(String dateiName, SpielHamster spielerA, SpielHamster spielerB)` auf. Geben Sie für den ersten Parameter `"Territorien/nimmspielfeld1"` ein. In den beiden anderen Parametern übergeben Sie jeweils einen der Hamster. Im Hamster-Simulator wird ein vordefiniertes Territorium geladen und die beiden Hamster spielen gegeneinander das Nimm-Spiel. Hamster der Klasse `MenschSpielHamster` werden vom Benutzer gesteuert, d.h. Sie müssen jeweils eingeben, ob der entsprechende Hamster 1 oder 2 Körner aufnehmen soll.

Dieses BlueJ-Hamster-Projekt nutzt das objektorientierte Konzept der Polymorphie, um Programme erweiterbar zu gestalten. Die Klasse `NimmSpiel` arbeitet ausschließlich mit dem Interface `SpielHamster`. Sie weiß nicht, welche konkreten Hamster später gegeneinander spielen werden. Das Interface `SpielHamster` wird von drei Klassen implementiert. Die Klasse `DummerSpielHamster` implementiert SpielHamster, die abwechselnd jeweils 1 bzw. 2 Körner fressen. Die Klasse `MenschSpielHamster` implementiert SpielHamster, die den Benutzer fragen, wie viele Körner sie nehmen sollen. Die Klasse `SchlauerSpielHamster` kennt die Gewinnstrategie des Nimm-Spiels. Falls möglich müssen nämlich jeweils so viele Körner liegen bleiben, dass deren Anzahl ohne Rest durch 3 dividierbar ist. Wenn ein Hamster dies einmal erreicht, kann er nicht mehr verlieren.

Nutzungsvorschlag 3: Gehen Sie jetzt noch mal genauso vor, wie in Nutzungsvorschlag 2 beschrieben. Erzeugen Sie nun aber einfach mal einen Hamster der Klasse `DummerSpielHamster` und einen Hamster der Klasse `SchlauerSpielHamster` und lassen Sie diese beiden Hamster gegeneinander spielen.

Nutzungsvorschlag 4: Öffnen Sie die Klassendefinition der Klasse `NimmSpielProgramm1`. Schauen Sie sich die Methode `void nimmSpielen()` an. Sie verdeutlicht folgendes: Alles, was Sie interaktiv im BlueJ-Fenster machen können, insbesondere das Erzeugen von Objekten und der Aufruf von Methoden für diese Objekte, lässt sich auch in Form von Programmen repräsentieren. Ändern Sie die Methode einfach mal so, dass ein MenschSpielHamster gegen einen SchlauenSpielHamster spielt. Anschließend müssen Sie speichern und compilieren. Wenn es keine Fehlermeldungen gibt, können Sie danach die Klassenmethode `void nimmSpielen()` aufrufen und es spielen im Territorium ein MenschSpielHamster und ein SchlauerSpielHamster gegeneinander das Nimm-Spiel.

BlueJ-Projekt „kalah“

Was prinzipiell alles mit dem Java-Hamster-Modell möglich ist, wird in diesem BlueJ-Hamster-Projekt demonstriert. Es ist ein sehr komplexes Beispiel, das aus der Welt der Spiele-Programmierung stammt. Den Hamstern wird das Spielen beigebracht, so dass sie in der Lage sind, gegen uns Menschen oder gegen andere Hamster so genannte Zwei-Spieler-Strategiespiele wie Schach, Mühle, Dame, 4-Gewinnt oder Reversi zu spielen, und die Hamster werden so gut sein, dass wir als Menschen kaum noch eine Chance haben, gegen sie zu gewinnen.

Das Spiel, an dem das zugrunde liegende Prinzip demonstriert wird, nennt sich *Kalah*. Kalah ist ein afrikanisches Brettspiel für zwei Spieler. Es besteht aus einem Brett mit zwei Reihen von je sechs Mulden, in denen anfangs je sechs Steine liegen, und zwei weiteren Mulden, die *Kalah* heißen und anfangs leer sind. Spieler A gehören die oberen Löcher und die linke Kalah, Spieler B die unteren Löcher und die rechte Kalah (siehe auch Abbildung 7).

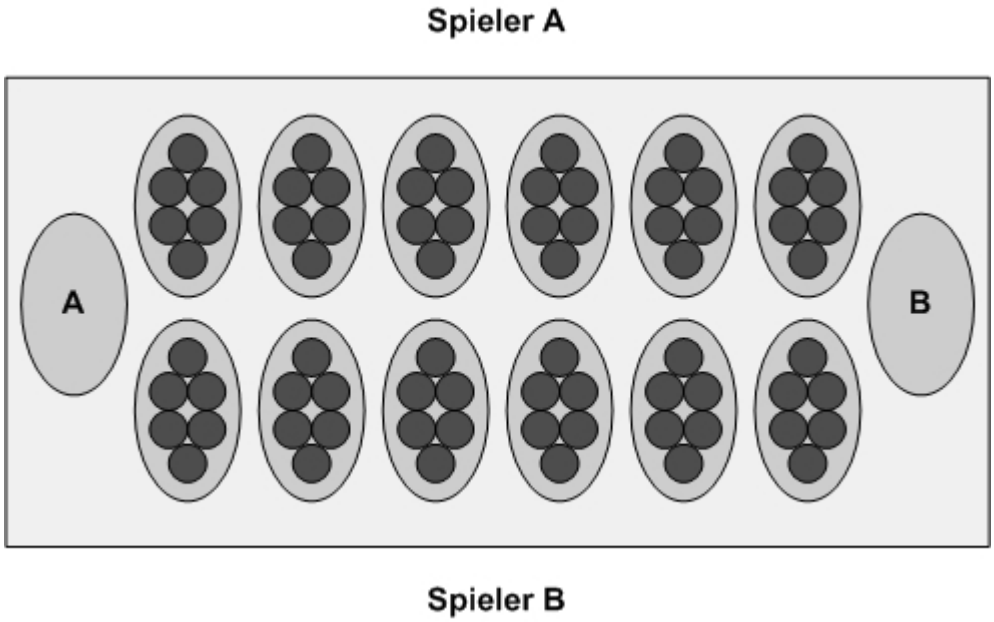

Abbildung 7: Kalah-Spielbrett

Ziel der beiden Spieler ist es, im Laufe eines Spiels möglichst viele Steine in die eigene Kalah zu bringen. Spieler A beginnt das Spiel. Ein Spielzug läuft dabei wie folgt ab:

- Der Spieler wählt eine seiner Mulden (nicht seine Kalah) aus, die nicht leer ist, entleert sie und verteilt auf die entgegen dem Uhrzeigersinn folgenden Mulden je einen Stein, wobei er die Kalah des Gegners auslässt.
- Landet der letzte Stein in einer eigenen leeren Mulde (keine Kalah) und ist die gegenüberliegende Mulde des Gegners nicht leer, so kommen der Stein sowie alle Steine der gegenüberliegenden Mulde in die eigene Kalah.
- Landet der letzte Stein in der eigenen Kalah, so muss der Spieler noch einmal ziehen. Ansonsten ist der Gegner an der Reihe.

Das Spiel ist beendet, wenn ein Spieler an der Reihe ist, aber keinen erlaubten Zug mehr ausführen kann, d.h. alle seine Mulden leer sind. Der andere Spieler darf dann alle Steine, die noch in seinen Mulden liegen, in die eigene Kalah legen. Gewonnen hat der Spieler, der anschließend mehr Steine in seiner Kalah liegen hat. Bei Gleichstand endet das Spiel mit einem Unentschieden.

Zunächst kümmern wir uns um den Fall, dass zwei menschliche Spieler mit Hilfe der Hamster gegeneinander das Kalah-Spiel spielen können. Gespielt wird dabei auf einem mauerlosen Hamster-Territorium mit mindestens 3 Reihen und 8 Spalten. Gespielt wird auch nicht mit Steinen, sondern natürlich mit „leckeren“ Körnern. Abbildung 8 zeigt das Hamster-Territorium, bevor das eigentliche Spielen beginnt. Wichtig ist die in der Abbildung skizzierte Nummerierung der Mulden.

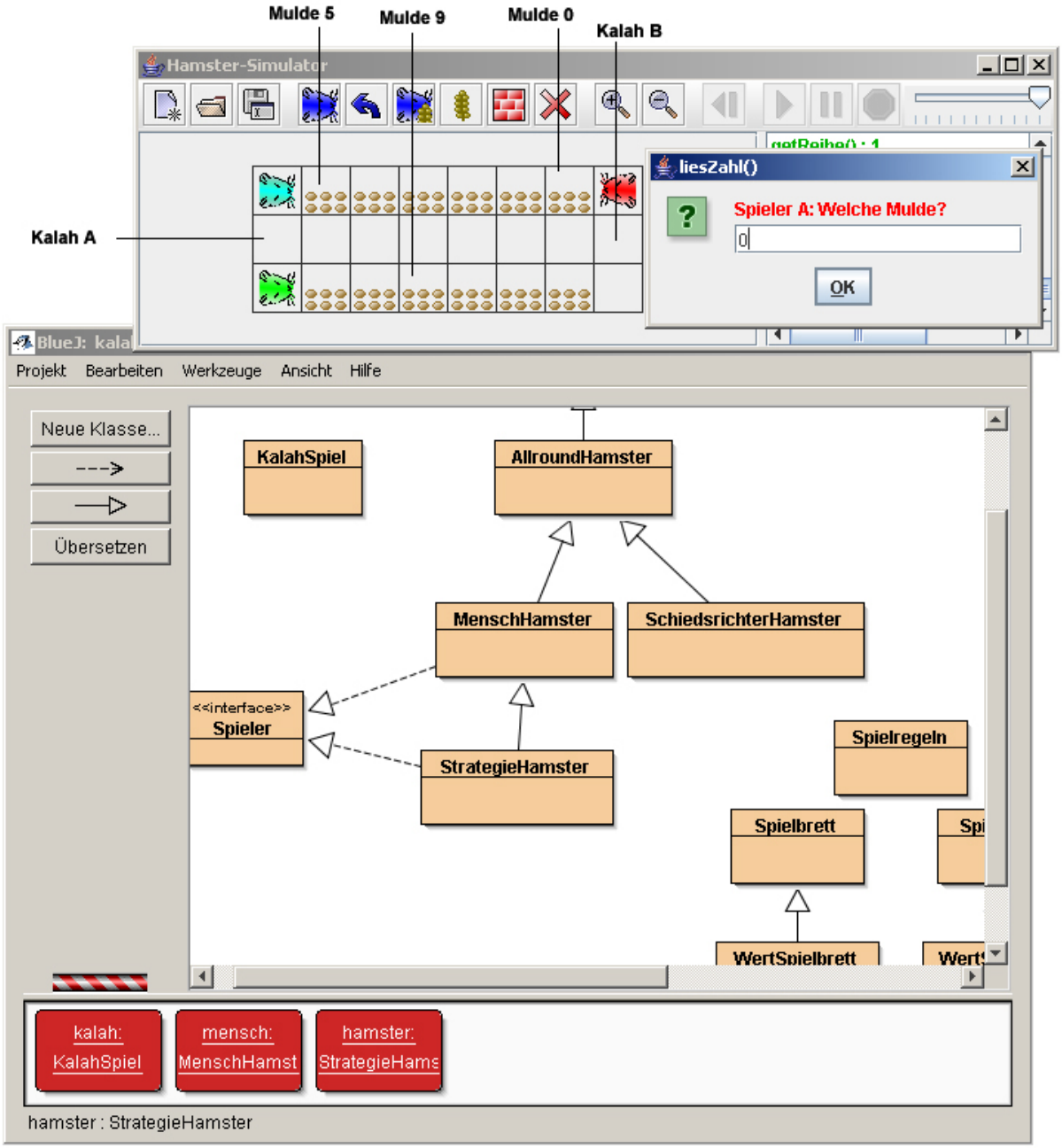

Abbildung 8: BlueJ-Projekt „kalah“

Nutzungsvorschlag 1: Erzeugen Sie in BlueJ zwei Objekte der Klasse `MenschHamster`. Erzeugen Sie weiterhin ein Objekt der Klasse `KalahSpiel`. Rufen Sie für das KalahSpiel-Objekt die Methode `void spielen(Spieler spielerA, Spieler spielerB)` auf und übergeben Sie jeweils einen der beiden Hamster als Parameter. Schauen Sie anschließend, was im Territorium passiert: Ein Schiedsrichter-Hamster baut das Spielfeld auf, die beiden MenschHamster laufen an ihre Ausgangsposition und dann fragen die beiden Hamster den Nutzer abwechselnd nach einem Spielzug, bis das Spiel beendet ist. Teilen Sie ihnen jeweils die Nummer der Mulde mit, die den nächsten Spielzug repräsentiert.

Nutzungsvorschlag 2: Nun lassen wir mal zwei „intelligente“ Hamster das Spiel gegeneinander spielen. Erzeugen Sie in BlueJ zwei Objekte der Klasse `StrategieHamster`. Der `int`-Parameter repräsentiert die Spielstärke des Hamsters. Je höher die Zahl ist, desto besser spielt der Hamster, desto länger überlegt er jedoch auch. Geben Sie beim ersten Hamster den Wert 2 ein und beim zweiten Hamster den Wert 6. Erzeugen Sie weiterhin ein Objekt der Klasse `KalahSpiel`. Rufen Sie für das KalahSpiel-Objekt die Methode `void spielen(Spieler spielerA, Spieler spielerB)` auf und übergeben Sie jeweils einen der beiden Hamster als Parameter. Anschließend spielen die beiden Hamster im Territorium gegeneinander das Kalah-Spiel und vermutlich (hier ist auch ein wenig Zufall im Spiel) wird der Hamster mit der Spielstärke 6 gegen den Hamster mit der Spielstärke 2 gewinnen. Rufen Sie nach Spielende erneut die Methode `void spielen(Spieler spielerA, Spieler spielerB)` auf und übergeben Sie die beiden Hamster in umgekehrter Reihenfolge, so dass beide jeweils einmal als Spieler A und einmal als Spieler B gegeneinander spielen.

Nutzungsvorschlag 3: So, die Stunde der Wahrheit steht bevor; wer spielt besser Kalah: Sie oder die Hamster? Um das zu erfahren, erzeugen Sie einen Hamster der Klasse `MenschHamster` und einen Hamster der Klasse `StrategieHamster` mit einer Spielstärke zwischen 5 und 9. Erzeugen Sie weiterhin ein Objekt der Klasse `KalahSpiel`. Rufen Sie für das KalahSpiel-Objekt die Methode `void spielen(Spieler spielerA, Spieler spielerB)` auf und übergeben Sie jeweils einen der beiden Hamster als Parameter. Und dann versuchen Sie mal, den StrategieHamster zu schlagen. Ich sage nur, das wird schwierig, sehr schwierig!

Wenn Sie bereits etwas Programmiererfahrung besitzen, können Sie sich auch ruhig mal die Klassen des Beispiels anschauen. Wie sie aufgebaut sind und wie man „intelligente“ Hamster bzw. generell „spielfähige Programme“ schreiben kann, wird ausführlich in Band 2 des Java-Hamster-Buches in Kapitel 15 erläutert.

8. Und nun…

So, jetzt sollten Sie eigentlich das Prinzip verstanden haben, das dem „Hamstern mit BlueJ“ zugrunde liegt. Spielen Sie einfach mal mit den Beispielprojekten herum, ändern Sie was oder erweitern Sie die Beispiele. Nehmen Sie sich weiterhin mal ein paar Aufgaben aus den beiden Hamster-Büchern vor und versuchen Sie diese, mit BlueJ zu lösen. Ich wünsche Ihnen viel Spaß und natürlich Programmier- und Lernerfolg beim „**Hamstern mit BlueJ**“!