Benutzungshandbuch

Frosch-Simulator

Version 1.0 (24.09.2009)

Dietrich Boles

Universität Oldenburg

Inhaltsverzeichnis

1 Einleitung

Programmieranfänger haben häufig Schwierigkeiten damit, dass sie beim Programmieren ihre normale Gedankenwelt verlassen und in eher technisch-orientierten Kategorien denken müssen, die ihnen von den Programmiersprachen vorgegeben werden. Gerade am Anfang strömen oft so viele inhaltliche und methodische Neuigkeiten auf sie ein, dass sie das Wesentliche der Programmierung, nämlich das Lösen von Problemen, aus den Augen verlieren.

1.1 Das Frosch-Modell

Das Frosch-Modell ist mit dem Ziel entwickelt worden, dieses Problem zu lösen. Mit dem Frosch-Modell wird Programmieranfängern ein einfaches, aber mächtiges Modell zur Verfügung gestellt, mit dessen Hilfe Grundkonzepte der (imperativen) Programmierung auf spielerische Art und Weise erlernt werden können. Programmierer entwickeln so genannte „Frosch-Programme“, mit denen sie einen virtuellen Frosch durch eine virtuelle Landschaft steuern und bestimmte Aufgaben lösen lassen. Die Anzahl der gleichzeitig zu berücksichtigenden Konzepte wird im Frosch-Modell stark eingeschränkt und nach und nach erweitert.

Prinzipiell ist das Frosch-Modell programmiersprachenunabhängig. Zum praktischen Umgang mit dem Modell wurde jedoch bewusst die Programmiersprache Java als Grundlage gewählt. Java – auch als „Sprache des Internet“ bezeichnet – ist eine moderne Programmiersprache, die sich in den letzten Jahren sowohl im Ausbildungsbereich als auch im industriellen Umfeld durchgesetzt hat.

Beim Frosch-Modell handelt es sich um eine so genannte Miniprogrammierwelt. Analoge Miniprogrammierwelten sind bspw. das Frosch-Modell (www.java-hamster-modell.de) oder Kara, der Mareinkäfer.

1.2 Der Frosch-Simulator

Beim Frosch-Modell steht nicht so sehr das "Learning-by-Listening" bzw. „Learning-by-Reading“ im Vordergrund, sondern vielmehr das „Learning-by-Doing“, also das praktische Üben. Genau das ist mit dem Frosch-Simulator möglich. Dieser stellt eine Reihe von Werkzeugen zum Erstellen und Ausführen von Frosch-Programmen zur Verfügung: einen Editor zum Eingeben und Verwalten von Frosch-Programmen, einen Compiler zum Übersetzen von Frosch-Programmen, einen Territoriumsgestalter zum Gestalten und Verwalten von Frosch-Territorien, einen Interpreter zum Ausführen von Frosch-Programmen und einen Debugger zum Testen von Frosch-Programmen. Der Frosch-Simulator ist einfach zu bedienen, wurde aber funktional und bedienungsmäßig bewusst an professionelle

Entwicklungsumgebungen für Java (z.B. Eclipse) angelehnt, um einen späteren Umstieg auf diese zu erleichtern.

1.3 Voraussetzungen

Zielgruppe des Frosch-Modells sind Schüler oder Studierende ohne Programmiererfahrung, die die Grundlagen der (imperativen) Programmierung erlernen und dabei ein wenig Spaß haben wollen. Kenntnisse im Umgang mit Computern sind wünschenswert. Der Frosch-Simulator ist dabei kein Lehrbuch sondern ein Werkzeug, das das Erlernen der Programmierung unterstützt. Auf gute begeleitende Lehrbücher zum Erlernen der Programmierung wird in Kapitel 7 (Literatur zum Programmieren lernen) hingewiesen.

1.4 Anmerkungen

Der Frosch-Simulator wurde mit dem Tool „Solist“ erstellt. Solist ist eine spezielle Entwicklungsumgebung für Miniprogrammierwelt-Simulatoren. Solist ist ein Werkzeug der Werkzeugfamilie des „Programmier-Theaters“, eine Menge von Werkzeugen zum Erlernen der Programmierung. Metapher aller dieser Werkzeuge ist die Theaterwelt. Schauspieler (im Falle von Solist „Solisten“) agieren auf einer Bühne, auf der zusätzlich Requisiten platziert werden können. Eine Aufführung entspricht der Ausführung eines Programms.

Im Falle des Frosch-Simulators ist die Bühne das Frosch-Territorium, auf dem der Frosch als Solist (es gibt nur einen Frosch) agiert. Requisiten sind Wasserkacheln und Mücken. Wenn Ihnen also beim Umgang mit dem Frosch-Simulator der Begriff „Solist“ begegnet, kennen Sie nun hiermit den Hintergrund dieser Namenswahl.

Mehr Informationen zu Solist finden Sie im WWW unter www.programmierkurs-java.de/solist.

1.5 Aufbau des Benutzerhandbuch

Das Benutzungshandbuch ist in 8 Kapitel gegliedert. Nach dieser Einleitung wird in Kapitel 2 die Installation des Frosch-Simulators beschrieben. Kapitel 3 stellt ausführlich das Frosch-Modell vor. Eine Übersicht über die Programmierkonzepte der imperativen Programmierung mit dem Frosch-Modell bzw. Java gibt Kapitel 4. Wie Sie Ihr erstes Frosch-Programm zum Laufen bringen, erfahren Sie kurz und knapp in Kapitel 5. Dieses enthält eine knappe Einführung in die Elemente und die Bedienung des Frosch-Simulators. Sehr viel ausführlicher geht dann Kapitel 6 auf die einzelnen Elemente und die Bedienung des Frosch-Simulators ein. Kapitel 7 demonstriert an einigen Beispielprogrammen die Programmierung mit dem Java-Frosch und gibt Ihnen einige Aufgaben an die Hand, die Sie bzw. der Frosch zu

lösen haben. Kapitel 8 enthält letztendlich Hinweise zu guter Begleitliteratur zum Erlernen der Programmierung mit Java.

2 Installation

2.1 Voraussetzungen

Voraussetzung zum Starten des Frosch-Simulators ist ein Java Development Kit SE (JDK) der Version 6 oder höher. Ein Java Runtime Environment SE (JRE) reicht nicht aus. Das jeweils aktuelle JDK kann über die Website http://java.sun.com/javase/downloads/index.jsp bezogen werden und muss anschließend installiert werden.

2.2 Download, Installation und Start

Der Frosch-Simulator kann von der Solist-Website http://www.programmierkurs-java.de/solist kostenlos herunter geladen werden. Er befindet sich in einer Datei namens *froschsimulator-1.0.zip*. Diese muss zunächst entpackt werden. Es entsteht ein Ordner namens *froschsimulator-1.0* (der so genannte *Simulator-Ordner*), in dem sich eine Datei *simulator.jar* befindet. Durch Doppelklick auf diese Datei wird der Frosch-Simulator gestartet. Alternativ lässt er sich auch durch Eingabe des Befehls `java –jar simulator.jar` in einer Console starten.

Beim ersten Start sucht der Frosch-Simulator nach der JDK-Installation auf Ihrem Computer. Sollte diese nicht gefunden werden, werden Sie aufgefordert, den entsprechenden Pfad einzugeben. Der Pfad wird anschließend in einer Datei namens *solist.properties* im Simulator-Ordner gespeichert, wo er später wieder geändert werden kann, wenn Sie bspw. eine aktuellere JDK-Version auf Ihrem Rechner installieren sollten. In der Property-Datei können Sie weiterhin die Sprache angeben, mit der der Frosch-Simulator arbeitet. In der Version 1.0 wird allerdings nur deutsch unterstützt.

3 Das Java-Frosch-Modell

Computer können heutzutage zum Lösen vielfältiger Aufgaben genutzt werden. Die Arbeitsanleitungen zum Bearbeiten der Aufgaben werden ihnen in Form von Programmen mitgeteilt. Diese Programme, die von Programmierern entwickelt werden, bestehen aus einer Menge von Befehlen bzw. Anweisungen, die der Computer ausführen kann. Die Entwicklung solcher Programme bezeichnet man als Programmierung.

Das Frosch-Modell ist ein spezielles didaktisches Modell zum Erlernen der Programmierung. Im Frosch-Modell nimmt ein virtueller Frosch die Rolle des Computers ein. Diesem Frosch können ähnlich wie einem Computer Befehle erteilt werden, die dieser ausführt.

Ihnen als Programmierer werden bestimmte Aufgaben gestellt, die sie durch die Steuerung des Froschs zu lösen haben. Derartige Aufgaben werden im Folgenden Frosch-Aufgaben genannt. Zu diesen Aufgaben müssen Sie in der Frosch-Sprache - eine Programmiersprache, die fast vollständig der Programmiersprache Java entspricht - Programme - Frosch-Programme genannt -- entwickeln, die die Aufgaben korrekt und vollständig lösen. Die Aufgaben werden dabei nach und nach komplexer. Zum Lösen der Aufgaben müssen bestimmte Programmierkonzepte eingesetzt werden, die im Frosch-Modell inkrementell eingeführt werden.

Die Grundidee des Frosch-Modells ist ausgesprochen einfach: Sie als Programmierer müssen einen (virtuellen) Frosch durch eine (virtuelle) Landschaft steuern und ihn gegebene Aufgaben lösen lassen.

3.1 Landschaft

Die Welt, in der der Frosch lebt, wird durch eine gekachelte Ebene repräsentiert. Abbildung 3.1 zeigt eine typische Froschlandschaft - auch Frosch-Territorium genannt. Die Größe der Landschaft, d.h. die Anzahl der Kacheln, ist dabei nicht explizit vorgegeben. Die Landschaft ist beliebig aber nie unendlich groß.

Die Froschlandschaft besteht aus Gras und Wasser. Konkret: Kacheln können entweder Graskacheln oder Wasserkacheln sein.

Auf den Kacheln des Frosch-Territoriums können sich weiterhin auch Mücken befinden, auf jeder Kachel allerdings maximal eine.

Das Frosch-Territorium ist übrigens torusförmig, d.h. wenn der Frosch es links verlässt, erscheint er rechts wieder (und umgekehrt) und wenn er es nach oben verlässt, erscheint er unten wieder.

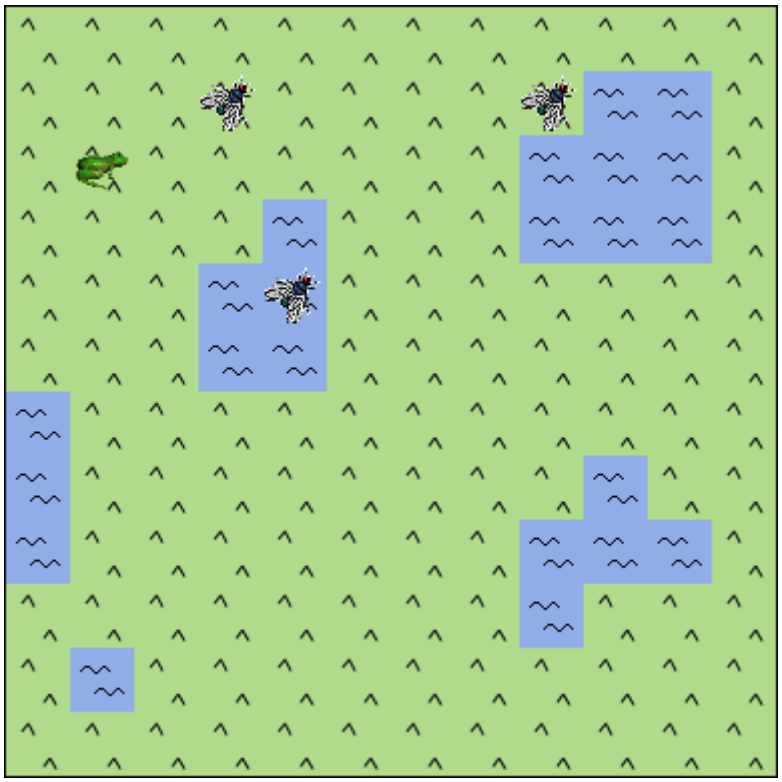


Abb. 3.1: Komponenten des Frosch-Modells

3.2 Frosch

Im Frosch-Modell existiert immer genau ein Frosch. Der Frosch steht dabei auf einer der Kacheln des Frosch-Territoriums. Der Frosch kann in vier unterschiedlichen Blickrichtungen (Nord, Süd, West, Ost) auf den Kacheln stehen. Je nach Blickrichtung wird der Frosch durch unterschiedliche Symbole repräsentiert. In Abbildung 3.1 schaut der Frosch nach Osten.

3.3 Befehle

Der Frosch kennt vier Befehle und drei Testbefehle, durch deren Aufruf ein Programmierer den Frosch durch ein gegebenes Frosch-Territorium steuern kann. Sie können sich den Frosch quasi als einen virtuellen Prozessor vorstellen, der im Gegensatz zu realen Computer-Prozessoren (zunächst) keine arithmetischen und logischen Operationen ausführen kann, sondern in der Lage ist, mit einem kleinen Grundvorrat an Befehlen ein Frosch-Territorium zu „erkunden“.

Die vier Befehle sind:

- `void huepfen();`: Der Frosch hüpft ein Feld in seiner aktuellen Blickrichtung nach vorne. Der Befehl darf nur aufgerufen werden, wenn sich der Frosch auf einer Graskachel befindet.
- `void schwimmen();`: Der Frosch schwimmt ein Feld in seiner aktuellen Blickrichtung nach vorne. Der Befehl darf nur aufgerufen werden, wenn sich der Frosch auf einer Wasserkachel befindet.
- `void rechtsUm();`: Der Frosch dreht sich um 90 Grad nach rechts.
- `void linksUm();`: Der Frosch dreht sich um 90 Grad nach links.

Die drei Testbefehle sind:

- `boolean imGras()`: Liefert genau dann true, wenn sich der Frosch auf einem Grasfeld befindet.
- `boolean vorneGras()`: Liefert genau dann true, wenn sich vor dem Frosch ein Grasfeld befindet.
- `boolean mueckeDa()`: Liefert genau dann true, wenn sich der Frosch auf einem Feld mit einer Mücke befindet.

3.4 Frosch-Programme

Ein Frosch-Programm setzt sich aus einer Prozedur mit dem Namen `main` sowie weiteren Prozeduren und Funktionen, die vor oder nach der main-Prozedur definiert werden dürfen, zusammen.

Der Aufruf bzw. Start eines Frosch-Programms bewirkt die automatische Ausführung der main-Prozedur.

3.5 Beispielprogramm

In folgendem Programm sitzt der Frosch, wie in Abbildung 3.2 skizziert, an einem Teich und bekommt die Aufgabe, den Teich zu umrunden, bis er auf eine Mücke stößt.

```
void main() {
    richtigPositionieren();
    while (!mueckeDa()) {
        einFeldBearbeiten();
    }
}

void richtigPositionieren() {
    while (!linksWasser()) {
        linksUm();
    }
}

void einFeldBearbeiten() {
```

```
    if (!linksWasser()) {
        linksUm();
        huepfen();
    } else if (!vorneWasser()) {
        huepfen();
    } else if (!rechtsWasser()) {
        rechtsUm();
        huepfen();
    } else if (!hintenWasser()) {
        kehrt();
        huepfen();
    }
}

boolean vorneWasser() {
    return !vorneGras();
}

boolean linksWasser() {
    linksUm();
    if (vorneWasser()) {
        rechtsUm();
        return true;
    } else {
        rechtsUm();
        return false;
    }
}

boolean rechtsWasser() {
    rechtsUm();
    if (vorneWasser()) {
        linksUm();
        return true;
    } else {
        linksUm();
        return false;
    }
}

boolean hintenWasser() {
    kehrt();
    if (vorneWasser()) {
        kehrt();
        return true;
    } else {
        kehrt();
        return false;
    }
}

void kehrt() {
    linksUm();
    linksUm();
}
```

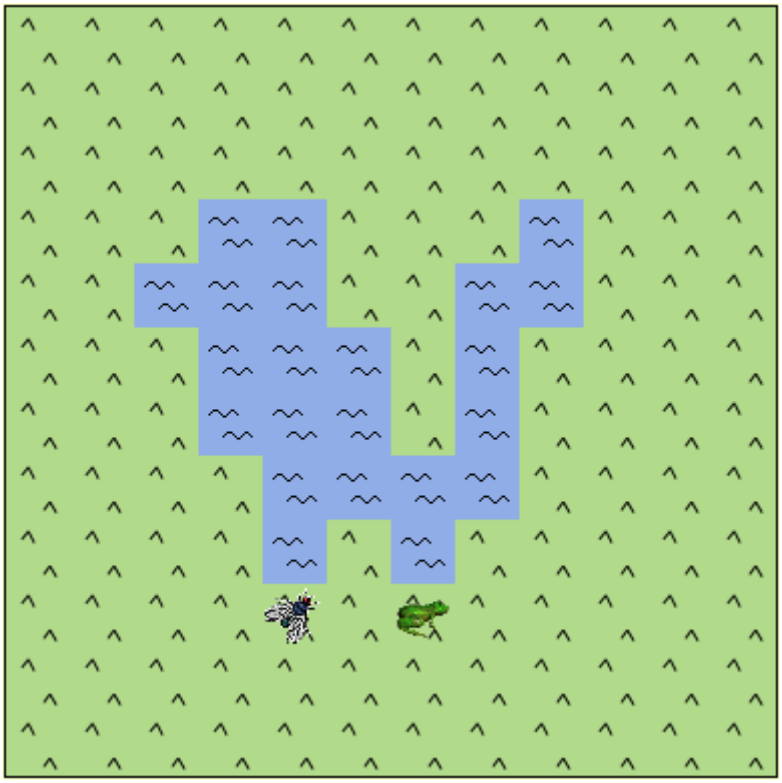


Abb. 3.2: Frosch-Territorium zum Beispielprogramm

4 Imperative Programmierkonzepte der Programmiersprache Java

Dieses Kapitel enthält eine Zusammenfassung der wichtigsten Programmierkonzepte der imperativen Programmierung mit Java, die die Grundlage von Frosch-Programmen bilden. Sie sollten sich jedoch begleitend zum Programmieren mit dem Frosch-Simulator noch ein Lehrbuch beschaffen, das die Sprachkonstrukte genauer beschreibt. Empfehlungen enthält Kapitel 8.

Wenn Sie noch überhaupt keine Kenntnisse der Programmierung haben, empfehle ich Ihnen, sich Informationen zu den allgemeinen Grundlagen der Programmierung auf der folgenden Website durchzulesen (Leseprobe des Java-Hamster-Buches): http://www-is.informatik.uni-oldenburg.de/~dibo/hamster/leseprobe/node3.html.

4.1 Grundlagen

Der Zeichenvorrat (die Lexikalik), den Sie beim Erstellen von Frosch-Programmen verwenden dürfen, entspricht dem 16-Bit-Zeichensatz Unicode.

Die Token einer Sprache, auch lexikalische Einheiten genannt, sind die Wörter, auf denen sie basiert. Wenn Sie Ihr Programm compilieren, teilt der Compiler Ihren Quellcode in Token auf und versucht herauszufinden, welche Anweisungen, Bezeichner und andere Elemente der Quellcode enthält. Token müssen in Java durch Wortzwischenräume voneinander getrennt werden. Zu den Wortzwischenräumen zählen Leerzeichen, Tabulatoren, Zeilenvorschub- und Seitenvorschubzeichen. Diese im Folgenden kurz als Trennzeichen bezeichneten Zeichen haben ansonsten keine Bedeutung.

Bezeichner, die zur Benennung von deklarierten Elementen (wie Prozeduren oder Variablen) verwendet werden, müssen in Java mit einem Buchstaben, einem Unterstrich (_) oder einem Dollarzeichen ($) beginnen, dem weitere Buchstaben, Unterstriche und Ziffern folgen können. Bezeichner dürfen beliebig lang sein.

In Java wird streng zwischen Groß- und Kleinbuchstaben unterschieden, d.h. dass bspw. die Bezeichner „rechts“ und „Rechts“ unterschiedliche Bezeichner sind.

Schlüsselwörter sind reservierte Wörter einer Programmiersprache. Sie dürfen nicht als Bezeichner verwendet werden.

4.2 Anweisungen und Programme

Programme setzen sich aus einer Menge von Befehle bzw. Anweisungen zusammen.

4.2.1 Frosch-Befehle

Die Aufgabe eines Frosch-Programmierers besteht darin, den Frosch durch eine Landschaft zu steuern, um dadurch gegebene Frosch-Aufgaben zu lösen. Zur Steuerung des Froschs müssen ihm Anweisungen in Form von Befehlen gegeben werden. Der Frosch besitzt dabei die Fähigkeit, vier verschiedene Befehle zu verstehen und auszuführen:

- `void huepfen();`: Der Frosch hüpft ein Feld in seiner aktuellen Blickrichtung nach vorne.
- `void schwimmen();`: Der Frosch schwimmt ein Feld in seiner aktuellen Blickrichtung nach vorne.
- `void rechtsUm();`: Der Frosch dreht sich um 90 Grad nach rechts.
- `void linksUm();`: Der Frosch dreht sich um 90 Grad nach links.

Bei den Befehlen `huepfen` und `schwimmen` Probleme auftreten:

- Der Frosch bekommt den Befehl `huepfen();` und er befindet sich nicht auf einer Graskachel.
- Der Frosch bekommt den Befehl `schwimmen();` und er befindet sich nicht auf einer Wasserkachel.

Bringen Sie den Frosch in diese für ihn unlösbaren Situationen, dann ist der Frsoch derart von Ihnen enttäuscht, dass er im Folgenden nicht mehr bereit ist, weitere Befehle auszuführen. Derartige Fehler werden Laufzeitfehler genannt. Laufzeitfehler können im Allgemeinen nicht schon durch den Compiler entdeckt werden, sondern treten erst während der Ausführung eines Programmes auf. Programme, die zu Laufzeitfehlern führen können, sind nicht korrekt!

4.2.2 Anweisungen

In imperativen Programmiersprachen werden Verarbeitungsvorschriften durch so genannte Anweisungen ausgedrückt. Anweisungen, die nicht weiter zerlegt werden können, werden elementare Anweisungen genannt. In der Frosch-Sprache sind die vier Grundbefehle elementare Anweisungen. Eine Folge von Anweisungen, die nacheinander ausgeführt werden, wird als Anweisungssequenz bezeichnet. Die einzelnen Anweisungen einer Anweisungssequenz werden in der angegebenen Reihenfolge hintereinander ausgeführt.

4.2.3 Programme

Ein Frosch-Programm besteht mindest aus einer main-Prozedur, die durch das Schlüsselwört `void`, gefolgt von `main`, einem runden Klammernpaar und einem geschweiften Klammernpaar, das eine Anweisungssequenz umschließt, gebildet wird. Beim Aufruf bzw. Start des Programms werden die Anweisungen der Anweisungssequenz innerhalb der geschweiften Klammern hintereinander ausgeführt.

4.2.4 Kommentare

Ziel der Programmierung ist es, Programme zu entwickeln, die gegebene Aufgaben lösen. Neben ihren Eigenschaften, korrekt und vollständig zu sein, sollten sich Programme durch eine weitere Eigenschaft auszeichnen; sie sollten gut verständlich sein. Das bedeutet, die Lösungsidee und die Realisierung sollte auch von anderen Programmierern mühelos verstanden und nachvollzogen werden können, um bspw. das Programm später noch zu erweitern oder in anderen Zusammenhängen wiederverwenden zu können.

Diesem Zweck der Dokumentation eines Programms dienen so genannte Kommentare. Sie haben auf die Steuerung des Froschs keinerlei Auswirkungen. Alles, was sie bewirken, ist eine bessere Lesbarkeit des Programms. In Java gibt es zwei Typen von Kommentaren: Zeilenkommentare und Bereichskommentare.

Zeilenkommentare beginnen mit zwei Schrägstrichen (`//`) und enden am nächsten Zeilenende. Den Schrägstrichen können beliebige Zeichen folgen.

Bereichskommentare beginnen mit der Zeichenkombination `/*` und enden mit der Zeichenkombination `*/`. Dazwischen können beliebige Zeichen stehen. Bereichskommentare können sich auch über mehrere Zeilen erstrecken.

4.3 Prozeduren

Prozeduren dienen zur Vereinbarung neuer Befehle. Diesbezüglich sind zwei Aspekte zu betrachten: die Definition von Prozeduren und deren Aufruf, d.h. Ausführung.

4.3.1 Prozedurdefinition

Durch eine Prozedurdefinition wird ein neuer Befehl vereinbart. In der Definition muss zum einen angegeben werden, wie der Befehl heißt (Prozedurname), und zum anderen muss festgelegt werden, was der Frosch tun soll, wenn er den neuen Befehl erhält. Ersteres erfolgt im so genannten Prozedurkopf, letzteres im so genannten Prozedurrumpf.

Im Prozedurkopf muss zunächst das Schlüsselwort `void` angegeben werden. Anschließend folgt ein Bezeichner, der Prozedurname bzw. der Name des neuen Befehls. Nach dem Prozedurnamen folgt ein rundes Klammernpaar, das den Prozedurkopf beendet. Der Prozedurrumpf beginnt mit einer öffnenden geschweiften Klammer, der eine Anweisungssequenz folgt. Der Prozedurrumpf und damit die Prozedurdefinition endet mit einer schließenden geschweiften Klammer.

4.3.2 Prozeduraufruf

Durch eine Prozedurdefinition wird ein neuer Befehl eingeführt. Ein Aufruf des neuen Befehls wird Prozeduraufruf genannt. Ein Prozeduraufruf entspricht syntaktisch dem Aufruf eines der vier Grundbefehle des Froschs. Er beginnt mit dem Prozedurnamen. Anschließend folgen eine öffnende und eine schließende runde Klammer und ein Semikolon.

Wird irgendwo in einem Programm eine Prozedur aufgerufen, so werden bei der Ausführung des Programms an dieser Stelle die Anweisung(en) des Prozedurrumpfes ausgeführt. Der Kontrollfluss des Programms verzweigt beim Prozeduraufruf in den Rumpf der Prozedur, führt die dortigen Anweisungen aus und kehrt nach der Abarbeitung der letzten Anweisung des Rumpfes an die Stelle des Prozeduraufrufs zurück. Durch Aufruf der return-Anweisung (`return;`) kann eine Prozedur vorzeitig verlassen werden.

4.4 Auswahlanweisungen

Auswahlanweisungen dienen dazu, bestimmte Anweisungen nur unter bestimmten Bedingungen ausführen zu lassen.

4.4.1 Testbefehle

Für aktuelle Zustandsabfragen und um Laufzeitfehler zu vermeiden, die ja bspw. dadurch entstehen können, dass Sie dem Frsoch den Befehl `huepfen` geben, obwohl er sich auf einer Wasserkachel befindet, existieren drei so genannte Testbefehle. Testbefehle liefern boolesche Werte, also *wahr* (`true`) oder *falsch* (`false`):

- `boolean imGras()`: Liefert genau dann true, wenn sich der Frosch auf einem Grasfeld befindet.
- `boolean vorneGras()`: Liefert genau dann true, wenn sich vor dem Frosch ein Grasfeld befindet.
- `boolean mueckeDa()`: Liefert genau dann true, wenn sich der Frosch auf einem Feld mit einer Mücke befindet.

4.4.2 Boolesche Operatoren und Ausdrücke

Die drei Testbefehle stellen einfache boolesche Ausdrücke dar. Ausdrücke sind Verarbeitungsvorschriften, die einen Wert berechnen und liefern. Boolesche Ausdrücke liefern einen booleschen Wert. Durch die Verknüpfung boolescher Ausdrücke mittels boolescher Operatoren lassen sich zusammengesetzte boolesche Ausdrücke bilden. Die Programmiersprache Java stellt drei boolesche Operatoren zur Verfügung:

- Der unäre Operator `!` (Negation) negiert den Wahrheitswert seines Operanden, d.h. er dreht ihn um. Liefert ein boolescher Ausdruck `bA` den Wert `true`, dann liefert der boolesche Ausdruck `!bA` den Wert `false`. Umgekehrt gilt, liefert `bA` den Wert false, dann liefert `!bA` den Wert `true`.
- Der binäre Operator `&&` (Konjunktion) konjugiert die Wahrheitswerte seiner beiden Operanden, d.h. er liefert genau dann den Wahrheitswert `true`, wenn beide Operanden den Wert `true` liefern.
- Der binäre Operator `||` (Disjunktion) disjungiert die Wahrheitswerte seiner beiden Operanden, d.h. er liefert genau dann den Wahrheitswert `true`, wenn einer seiner Operanden oder seine beiden Operanden den Wert `true` liefern.

Der Operator `!` hat eine höhere Priorität als der Operator `&&`, der wiederum eine höhere Priorität als der Operator `||` besitzt. Durch Einschließen von booleschen Ausdrücken in runde Klammern können Sie die Abarbeitungsfolge der Operatoren beeinflussen.

4.4.3 Blockanweisung

Mit Hilfe der Blockanweisung lassen sich mehrere Anweisungen zu einer Einheit zusammenfassen. Syntaktisch gesehen handelt es sich bei einer Blockanweisung um eine zusammengesetzte Anweisung. Innerhalb von geschweiften Klammern steht eine andere Anweisung – im Allgemeinen eine Anweisungssequenz.

Beim Ausführen einer Blockanweisung werden die innerhalb der geschweiften Klammern stehenden Anweisungen ausgeführt.

4.4.4 Bedingte Anweisung

Die bedingte Anweisung, die auch if-Anweisung genannt wird, ist eine zusammengesetzte Anweisung. Die bedingte Anweisung wird eingeleitet durch das Schlüsselwort `if`. Anschließend folgen innerhalb eines runden Klammernpaares ein boolescher Ausdruck und danach eine Anweisung. Bei dieser Anweisung, die im Folgenden true-Anweisung genannt wird, handelt es sich im Allgemeinen um eine Blockanweisung.

Beim Ausführen einer bedingten Anweisung wird zunächst der boolesche Ausdruck innerhalb der runden Klammern ausgewertet. Falls dieser Ausdruck den Wert `true` liefert, d.h. die Bedingung erfüllt ist, wird die true-Anweisung (daher der Name) ausgeführt. Liefert der boolesche Ausdruck den Wert `false`, dann wird die true-Anweisung nicht ausgeführt.

4.4.5 Alternativanweisung

Die Alternativanweisung ist eine bedingte Anweisung mit einem angehängten so genannten else-Teil. Dieser besteht aus dem Schlüsselwort `else` und einer Anweisung (false-Anweisung) – im Allgemeinen eine Blockanweisung. Die Alternativanweisung ist wie die bedingte Anweisung eine Auswahlanweisung.

Wird eine Alternativanweisung ausgeführt, dann wird zunächst der Wert der Bedingung (boolescher Ausdruck) ermittelt. Ist die Bedingung erfüllt, d.h. liefert der boolesche Ausdruck den Wert `true`, dann wird die true-Anweisung nicht aber die false-Anweisung ausgeführt. Liefert der boolesche Ausdruck den Wert `false`, dann wird die false-Anweisung nicht aber die true-Anweisung ausgeführt.

4.5 Wiederholungsanweisungen

Wiederholungsanweisungen – auch Schleifenanweisungen genannt – dienen dazu, bestimmte Anweisungen mehrmals ausführen zu lassen, solange eine bestimmte Bedingung erfüllt wird.

4.5.1 while-Anweisung

Die while-Anweisung ist eine zusammengesetzte Anweisung. Nach dem Schlüsselwort while steht in runden Klammern ein boolescher Ausdruck, die so genannte Schleifenbedingung. Anschließend folgt die Anweisung, die eventuell mehrfach ausgeführt werden soll. Sie wird auch Iterationsanweisung genannt. Hierbei handelt es sich im Allgemeinen um eine Blockanweisung.

Bei der Ausführung einer while-Anweisung wird zunächst überprüft, ob die Schleifenbedingung erfüllt ist, d.h. ob der boolesche Ausdruck den Wert `true` liefert. Falls dies nicht der Fall ist, ist die while-Anweisung unmittelbar beendet. Falls die Bedingung erfüllt ist, wird die Iterationsanweisung einmal ausgeführt. Anschließend wird die Schleifenbedingung erneut ausgewertet. Falls sie immer noch erfüllt ist, wird die Iterationsanweisung ein weiteres Mal ausgeführt. Dieser Prozess (Überprüfung der Schleifenbedingung und falls diese erfüllt ist, Ausführung der Iterationsanweisung) wiederholt sich so lange, bis (hoffentlich) irgendwann einmal die Bedingung nicht mehr erfüllt ist.

4.5.2 do-Anweisung

Bei Ausführung der while-Anweisung kann es vorkommen, dass die Iterationsanweisung kein einziges Mal ausgeführt wird; nämlich genau dann, wenn die Schleifenbedingung direkt beim ersten Test nicht erfüllt ist. Für solche Fälle, bei denen die Iterationsanweisung auf jeden Fall mindestens einmal ausgeführt werden soll, existiert die do-Anweisung – auch do-Schleife genannt.

Dem Schlüsselwort `do`, von dem die Anweisung ihren Namen hat, folgt die Iterationsanweisung. Hinter der Iterationsanweisung muss das Schlüsselwort `while` stehen. Anschließend folgt in runden Klammern ein boolescher Ausdruck – die Schleifenbedingung. Abgeschlossen wird die do-Anweisung durch ein Semikolon.

Bei der Ausführung einer do-Anweisung wird zunächst einmal die Iterationsanweisung ausgeführt. Anschließend wird die Schleifenbedingung überprüft. Ist sie nicht erfüllt, d.h. liefert der boolesche Ausdruck den Wert `false`, dann endet die do-Anweisung. Ist die Bedingung erfüllt, wird die Iterationsanweisung ein zweites Mal ausgeführt und danach erneut die Schleifenbedingung ausgewertet. Dieser Prozess wiederholt sich so lange, bis irgendwann einmal die Schleifenbedingung nicht mehr erfüllt ist.

4.6 Boolesche Funktionen

Während Prozeduren dazu dienen, den Befehlsvorrat des Froschs zu erweitern, dienen boolesche Funktionen dazu, neue Testbefehle einzuführen.

4.6.1 Boolesche return-Anweisung

Boolesche return-Anweisungen werden in booleschen Funktionen zum Liefern eines booleschen Wertes benötigt.

Die Syntax der booleschen return-Anweisung ist sehr einfach: Dem Schlüsselwort `return` folgt ein boolescher Ausdruck und ein abschließendes Semikolon. Boolesche return-Anweisungen sind spezielle Anweisungen, die ausschließlich im Funktionsrumpf boolescher Funktionen verwendet werden dürfen.

Die Ausführung einer booleschen return-Anweisung während der Ausführung einer booleschen Funktion führt zur unmittelbaren Beendigung der Funktionsausführung. Dabei wird der Wert des booleschen Ausdrucks als so genannter Funktionswert zurückgegeben.

4.6.2 Definition boolescher Funktionen

Die Syntax der Definition einer booleschen Funktion unterscheidet sich nur geringfügig von der Definition einer Prozedur. Statt Prozedurkopf, -name und -rumpf spricht man hier von Funktionskopf, Funktionsname und Funktionsrumpf.

Anstelle des Schlüsselwortes `void` bei der Definition einer Prozedur muss bei der Definition einer booleschen Funktion das Schlüsselwort `boolean` am Anfang des Funktionskopfes stehen. Ganz wichtig bei der Definition boolescher Funktionen ist jedoch folgende Zusatzbedingung: In jedem möglichen Weg durch die Funktion muss eine boolesche return-Anweisung auftreten!

Boolesche Funktionen können überall dort in einem Frosch-Programm definiert werden, wo auch Prozeduren definiert werden können.

4.6.3 Aufruf boolescher Funktionen

Eine boolesche Funktion darf überall dort aufgerufen werden, wo auch einer der drei vordefinierten Testbefehle aufgerufen werden darf. Der Aufruf einer booleschen Funktion gilt also als ein spezieller boolescher Ausdruck. Der Funktionsaufruf erfolgt syntaktisch durch die Angabe des Funktionsnamens gefolgt von einem runden Klammernpaar.

Wird bei der Berechnung eines booleschen Ausdrucks eine boolesche Funktion aufgerufen, so wird in deren Funktionsrumpf verzweigt und es werden die dortigen Anweisungen aufgerufen. Wird dabei eine boolesche return-Anweisung ausgeführt, so wird der Funktionsrumpf unmittelbar verlassen und an die Stelle des Funktionsaufrufs zurückgesprungen. Der von der booleschen return-Anweisung gelieferte Wert (also der Funktionswert) wird dabei zur Berechnung des booleschen Ausdrucks weiterverwendet.

4.7 Variablen und Ausdrücke

Durch die Einführung von Variablen und Ausdrücken bekommt der Frosch ein „Gedächtnis“ und lernt rechnen.

4.7.1 Datentypen

Ein Datentyp repräsentiert Werte eines bestimmten Typs. Im Frosch-Modell werden die Datentypen `boolean` und `int` genutzt. Der Datentyp `boolean` repräsentiert boolesche Werte, also `true` und `false`. Der Datentyp int repräsentiert ganze Zahlen zwischen -2^{31} und $2^{31} - 1$.

4.7.2 Variablen

Variablen sind Speicherbereiche („Behälter“), in denen Werte abgespeichert werden können. Vor ihrer Benutzung müssen sie definiert werden. Bei der Definition einer Variablen wird ihr ein Name – ein beliebiger Bezeichner – zugeordnet. Außerdem wird durch die Angabe eines Datentyps festgelegt, welche Werte die Variable speichern kann.

Die folgenden Anweisungen definieren eine Variable `frei` zum Speichern von booleschen Werten sowie eine Variable `anzahlMuecken` zum Speichern von ganzen Zahlen. Der Variablen frei wird der Initialwert `false` zugewiesen, der Variablen `anzahlMuecken` der Wert 13.

```
boolean frei = false;
int anzahlMuecken = 13;
```

4.7.3 Ausdrücke

Ausdrücke sind spezielle Programmierkonstrukte zum Berechnen und Liefern eines Wertes. Boolesche Ausdrücke haben Sie bereits kennen gelernt. Sie liefern boolesche Werte. Zur Berechnung boolescher Ausdrücke können auch boolean-Variablen hinzugezogen werden. Enthält ein boolescher Ausdruck den Namen einer booleschen Variablen, dann wird bei der Auswertung des booleschen Ausdrucks an der entsprechenden Stelle der Wert berücksichtigt, der aktuell in der Variablen gespeichert ist.

Einen weiteren Typ von Ausdrücken stellen arithmetische Ausdrücke dar. Sie berechnen und liefern Werte vom Typ int, also ganze Zahlen. Arithmetische Ausdrücke lassen sich auf folgende Art und Weise bilden:

- int-Literale: int-Literale werden durch Zeichenfolgen beschrieben, die aus dezimalen Ziffern (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) bestehen. Dabei gilt die Einschränkung, dass einer Zahl ungleich 0 keine „0“ vorangestellt werden darf. Gültige int-Literale sind also: 0, 2, 4711, 1234560789, …
- int-Variablenname: Der Name einer int-Variablen in einem arithmetischen Ausdruck repräsentiert den aktuell in der Variablen gespeicherten Wert.
- Unäre arithmetische Operatoren: Die Zeichen „+“ und „-“ kennzeichnen Vorzeichen von arithmetischen Ausdrücken. Die unären arithmetischen Operatoren sind rechtsassoziativ und besitzen die höchste Priorität aller arithmetischen Operatoren.
- Binäre arithmetische Operatoren: Es existieren insgesamt fünf binäre arithmetische Operatoren, mit denen jeweils zwei andere arithmetische Ausdrücke (die Operanden) verknüpft werden:

- „+“: liefert als Wert die Summe seiner beiden Operanden (Addition)
- „-“: liefert als Wert die Differenz seiner beiden Operanden (Subtraktion)
- „*“: liefert als Wert das Produkt seiner beiden Operanden (Produkt)
- „/“: liefert als Wert den Quotient seiner beiden Operanden; dabei werden entstehende Nachkommastellen ignoriert (ganzzahlige Division); z.B. 7/3 = 2
- „%“: liefert als Wert den Rest einer ganzzahligen Division (Modulo-Operator); z.B. 7%3 = 1. Zwischen der Ganzzahldivision und der Restbildung besteht folgende Beziehung: Seien x und y arithmetische Ausdrücke, dann gilt ((x/y) * y) + (x%y) = x.

Die binären arithmetischen Operatoren sind linksassoziativ. Die Operatoren „*“, „/“ und „%“ besitzen eine höhere Priorität als die Operatoren „+“ und „-“ („Punkt-vor-Strich-Rechnung“).

- Klammern: Zum Bilden von (komplexen) arithmetischen Ausdrücken können arithmetische Ausdrücke in Paare von runden Klammern gesetzt werden. Dadurch lässt sich die Priorität von arithmetischen Operatoren beeinflussen.

4.7.4 Zuweisung

Mit Hilfe einer Zuweisungsanweisung – kurz auch Zuweisung genannt – können Variablen neue Werte zugewiesen werden. Die alten Werte gehen dabei verloren.

Syntaktisch wird die Zuweisung so gebildet, dass dem Namen der betroffenen Variablen das Zeichen = – der Zuweisungsoperator – und anschließend ein Ausdruck folgt. Bei booleschen Variablen muss das ein boolescher Ausdruck sein, bei int-Variablen ein arithmetischer Ausdruck.

Bei der Ausführung einer Zuweisung wird zunächst der Ausdruck ausgewertet und anschließend der berechnete Wert der Variablen auf der linken Seite des Zuweisungsoperators zugewiesen. Die Zuweisungsanweisung `zahl = zahl + 1;` erhöht bspw. den Wert einer int-Variablen `zahl` um den Wert 1.

4.7.5 Vergleichsausdrücke

Vergleichsausdrücke sind boolesche Ausdrücke, die zum Vergleichen arithmetischer Ausdrücke dienen. Sie liefern boolesche Werte nach folgenden Gesetzmäßigkeiten: Seien x und y zwei arithmetische Ausdrücke, dann gilt:

- x == y liefert genau dann den Wert true, falls der Wert, den x liefert, gleich dem Wert ist, den y liefert (Gleichheitsoperator).
- x! = y liefert genau dann den Wert true, falls der Wert, den x liefert, ungleich dem Wert ist, den y liefert (Ungleichheitsoperator).

- x < y liefert genau dann den Wert true, falls der Wert, den x liefert, kleiner ist als der Wert, den y liefert (Kleineroperator).
- x <= y liefert genau dann den Wert true, falls der Wert, den x liefert, kleiner oder gleich dem Wert ist, den y liefert (Kleinergleichoperator).
- x > y liefert genau dann den Wert true, falls der Wert, den x liefert, größer ist als der Wert, den y liefert (Größeroperator).
- x >= y liefert genau dann den Wert true, falls der Wert, den x liefert, größer oder gleich dem Wert ist, den y liefert (Größergleichoperator).

Die Vergleichsoperatoren sind linksassoziativ. Die Operatoren <, <=, > und >= haben eine höhere Priorität als die Operatoren == und ! =. Weiterhin ist zu beachten, dass die Vergleichsoperatoren eine niedrigere Priorität besitzen als die arithmetischen Operatoren und eine höhere Priorität als der Zuweisungsoperator.

4.7.6 Gültigkeitsbereiche von Variablen

Variablen lassen sich innerhalb von Prozeduren und Funktionen definieren. In diesem Fall nennt man sie lokale Variablen. Variablen, die außerhalb von Prozeduren oder Funktionen definiert werden, heißen globale Variablen.

Als Gültigkeitsbereich einer Variablen wird der Teil eines Programmes bezeichnet, in dem eine Variable genutzt werden kann, d.h. in dem der Name einer Variablen verwendet werden darf. Der Gültigkeitsbereich einer lokalen Variablen erstreckt sich von der der Variablendefinition folgenden Anweisung bis zum Ende desselben Blockes und umschließt alle inneren Blöcke. Der Gültigkeitsbereich einer globalen Variablen umfasst das gesamte Frosch-Programm. Im Gültigkeitsbereich einer Variablen darf keine weitere Variable mit demselben Namen definiert werden.

4.7.7 Lebensdauer von Variablen

Während der Gültigkeitsbereich einer booleschen Variablen zur Compilierzeit von Bedeutung ist, ist die Lebensdauer einer booleschen Variablen eine Größe, die zur Laufzeit Relevanz besitzt. Sie ist definiert als die Zeitspanne, während der im Hauptspeicher Speicherplatz für eine Variable reserviert ist. Es gilt dabei: Die Lebensdauer einer globalen Variablen umfasst die gesamte Ausführungszeit eines Frosch-Programms. Die Lebensdauer einer lokalen Variablen beginnt bei ihrer Definition und endet nach der vollständigen Abarbeitung des Blocks, in dem sie definiert wurde.

4.8 int-Funktionen

Boolesche Funktionen liefern einen Wert vom Typ `boolean`. Dementsprechend sind int-Funktionen Funktionen, die einen Wert vom Typ `int`, also eine ganze Zahl, liefern.

4.8.1 Funktionsdefinition

Die Definition einer int-Funktion unterscheidet sich nur dadurch von der Definition einer booleschen Funktion, dass im Funktionskopf das Schlüsselwort `boolean` gegen das Schlüsselwort `int` ausgetauscht werden muss und die im Funktionsrumpf obligatorischen return-Anweisungen anstelle eines booleschen Wertes einen Wert vom Typ `int` liefern müssen.

4.8.2 Funktionsaufruf

Der Aufruf von int-Funktionen entspricht einem speziellen arithmetischen Ausdruck. int-Funktionen dürfen also überall dort aufgerufen werden, wo arithmetische Ausdrücke stehen dürfen. Der Aufruf einer int-Funktion erfolgt wie der Aufruf einer booleschen Funktion syntaktisch durch die Angabe des Funktionsnamens gefolgt von einem runden Klammernpaar. Beim Aufruf einer int-Funktion wird in den Funktionsrumpf verzweigt und die dortigen Anweisungen werden ausgeführt. Als Wert des arithmetischen Ausdrucks wird in diesem Fall der Wert genommen, den die Funktion berechnet und mit Hilfe einer return-Anweisung geliefert hat.

4.8.3 Verallgemeinerung des Funktionskonzeptes

Neben den Datentypen `boolean` und `int` gibt es in Java und in anderen Programmiersprachen weitere Datentypen, auf deren Einführung im Frosch-Modell zunächst verzichtet wird, für die das Funktionskonzept aber entsprechend gilt.

Funktionen können prinzipiell wie Prozeduren auch in Form von Anweisungen aufgerufen werden. Der gelieferte Funktionswert wird dann einfach ignoriert.

Prozeduren werden ab jetzt als Spezialform von Funktionen aufgefasst. Sie liefern keinen Wert, was durch das Schlüsselwort `void` bei ihrer Definition ausgedrückt wird.

Der Gültigkeitsbereich von Funktionen erstreckt sich über ein gesamtes Programm. Insbesondere dürfen Funktionen auch schon vor ihrer Definition aufgerufen werden.

Wenn sich nach der Ausführung einer Funktion der Programmzustand verändert hat (bspw. die Position oder Blickrichtung des Froschs), spricht man von einem Seiteneffekt, den die Funktion produziert hat.

4.9 Funktionsparameter

Das Parameterkonzept erhöht die Flexibilität von Prozeduren und Funktionen.

4.9.1 Formale Parameter

Parameter sind lokale Variablen von Funktionen, die dadurch initialisiert werden, dass der Funktion bei ihrem Aufruf ein entsprechender Initialisierungswert für die Variable übergeben wird.

Parameter werden im Funktionskopf definiert. Zwischen die beiden runden Klammern wird eine so genannte formale Parameterliste eingeschoben. Die Parameterliste besteht aus keiner, einer oder mehreren durch Kommata getrennten Parameterdefinitionen. Eine Parameterdefinition hat dabei eine ähnliche Gestalt wie die Variablendefinition. Was fehlt, ist ein expliziter Initialisierungsausdruck. In der Tat handelt es sich bei einem Parameter auch um eine ganz normale Variable. Sie ist lokal bezüglich des Funktionsrumpfes. Ihr können im Funktionsrumpf ihrem Typ entsprechend Werte zugewiesen werden und sie kann bei der Bildung von typkonformen Ausdrücken innerhalb des Funktionsrumpfes eingesetzt werden. Man nennt die Parameter innerhalb der Funktionsdefinition auch formale Parameter oder Parametervariablen.

4.9.2 Aktuelle Parameter

Der Funktionsaufruf wird durch die Angabe einer aktuellen Parameterliste zwischen den runden Klammern erweitert. Die durch Kommata getrennten Elemente dieser Liste werden als aktuelle Parameter bezeichnet. Hierbei handelt es sich um Ausdrücke.

4.9.3 Parameterübergabe

Bezüglich der Definition von Funktionen mit (formalen) Parametern und dem Aufruf von Funktionen mit (aktuellen) Parametern sind folgende zusätzliche Bedingungen zu beachten:

- Die Anzahl der aktuellen Parameter beim Aufruf einer Funktion muss gleich der Anzahl der formalen Parameter der Funktionsdefinition sein. (Ausnahme: so genannte varargs-Parameter)
- Für alle Parameter in der angegebenen Reihenfolge muss gelten: Der Typ eines aktuellen Parameters muss konform sein zum Typ des entsprechenden formalen Parameters (boolesche Ausdrücke sind konform zum Typ `boolean`, arithmetische Ausdrücke sind konform zum Typ `int`).

Wird eine Funktion mit Parametern aufgerufen, so passiert folgendes: Die aktuellen Parameter – hierbei handelt es sich ja um Ausdrücke – werden berechnet, und zwar immer von links nach rechts, falls es sich um mehr als einen Parameter handelt. Für jeden formalen Parameter der formalen Parameterliste wird im Funktionsrumpf eine lokale Variable angelegt. Diese Variablen werden anschließend – bei Beachtung der

Reihenfolge innerhalb der Parameterlisten – mit dem Wert des entsprechenden aktuellen Parameters initialisiert. Man spricht in diesem Zusammenhang auch von Parameterübergabe: Der Wert eines aktuellen Parameters wird beim Aufruf einer Funktion einem formalen Parameter der Funktion als Initialisierungswert übergeben.

4.9.4 Überladen von Funktionen

Eigentlich müssen Funktionsnamen in einem Programm eindeutig sein. Zwei oder mehrere Funktionen dürfen jedoch denselben Namen besitzen, falls sich ihre formalen Parameterlisten durch die Anzahl an Parametern oder die Typen der Parameter unterscheiden. Man nennt dieses Prinzip auch Überladen von Funktionen. Die tatsächlich aufgerufene Funktion wird dann beim Funktionsaufruf anhand der Anzahl bzw. Typen der aktuellen Parameterliste bestimmt.

5 Ihr erstes Frosch-Programm

Nachdem Sie den Frosch-Simulator gestartet haben, öffnet sich ein Fenster, das in etwa dem in Abbildung 5.1 dargestellten Fenster entspricht. Ganz oben enthält es eine Menüleiste mit 5 Menüs, darunter eine Toolbar mit einer Reihe von Buttons und ganz unten einen Meldungsbereich, in dem Meldungen ausgegeben werden. Im linken Teil befindet sich der Editor-Bereich, im rechten Teil der Simulationsbereich. Im Editor-Bereich geben Sie Frosch-Programme ein und im Simulationsbereich führen Sie Frosch-Programme aus.

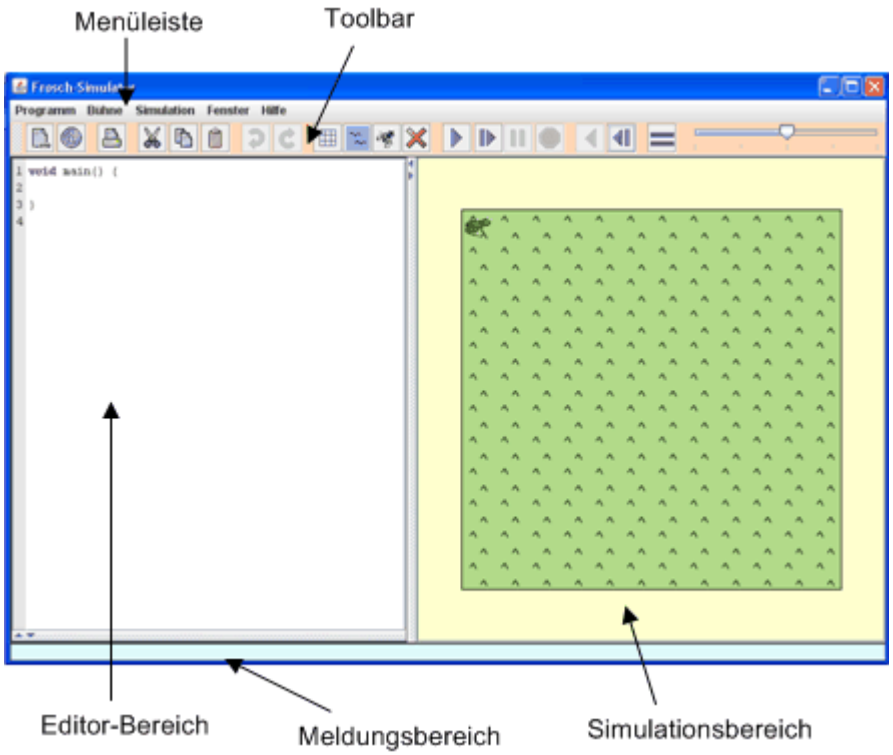


Abb. 5.1: Frosch-Simulator nach dem Öffnen

Um den Frosch ein wenig näher kennen zu lernen, epmpfehle ich Ihnen, als erstes zunächst die Frosch-Befehle einmal auszuprobieren. Wie das geht, wird in Abschnitt 5.1 erläutert. Anschließend wird in den darauf folgenden Abschnitten im Detail beschrieben, was Sie machen müssen, um Ihr erstes Frosch-Programm zu schreiben und auszuführen. Insgesamt müssen/können fünf Stationen durchlaufen werden:

- Gestaltung eines Frosch-Territoriums
- Eingeben eines Frosch-Programms
- Compilieren eines Frosch-Programms

- Ausführen eines Frosch-Programms
- Debuggen eines Frosch-Programms

5.1 Ausprobieren der Frosch-Befehle

Klicken Sie im Menü „Fenster“ das Menü-Item „Befehlsfenster“ an. Es öffnet sich ein Fenster mit dem Titel „Befehlsfenster“. In diesem Fenster erscheinen alle Befehle, die der Frosch kennt.

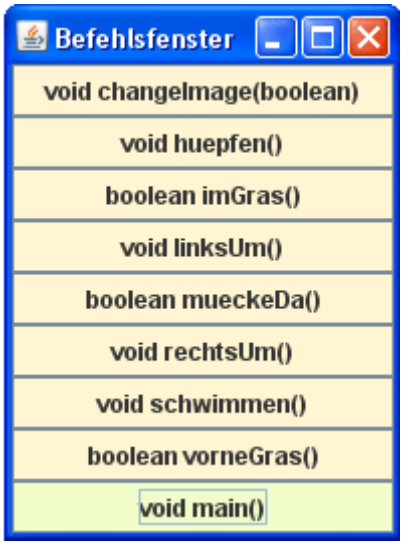


Abb. 5.2: Befehlsfenster

Sie können die jeweiligen Befehle ausführen, indem Sie den Maus-Cursor auf den entsprechenden Button verschieben und diesen anklicken. Klicken Sie bspw. auf den Button „void huepfen()“, dann hüpft der Frosch in seiner aktuellen Blickrichtung eine Kachel nach vorne (oder es erscheint eine Fehlermeldung, wenn der Frosch auf einer Wasserkachel steht).

Wenn Sie Programme mit Prozeduren oder Funktionen schreiben und erfolgreich kompilieren, werden die Prozeduren und Funktionen übrigens auch im Befehlsfenster angezeigt und können interaktiv ausgeführt werden. Dabei werden keine Zwischenzustände im Simulationsbeereich ausgegeben, sondern immer der jeweilige Endzustand nach Abarbeitung der Funktion.

5.2 Gestaltung eines Frosch-Territoriums

Nun wollen wir ein Frosch-Territorium aufbauen, in dem unser erstes Programm ablaufen soll. Das geschieht im Simulationsbereich. Hier wird das Frosch-Territorium dargestellt. Zur Gestaltung des Territoriums müssen Sie die Buttons 9 – 12 (von

links) in der so genannten *Toolbar* benutzen. Diese werden auch als *Gestaltungsbuttons* bezeichnet. Fahren Sie einfach mal mit der Maus über die einzelnen Buttons der Toolbar, dann erscheint jeweils ein Tooltipp, der beschreibt, wozu dieser Button dient.

Zunächst werden wir die Größe des Territoriums anpassen. Klicken Sie dazu auf den Button „Territoriumsgröße ändern“ (9. Button von links). Es erscheint eine Dialogbox, in der Sie die gewünschte Anzahl an Reihen und Spalten eingeben können. Um die dort erscheinenden Werte (jeweils 12) ändern zu können, klicken Sie mit der Maus auf das entsprechende Eingabefeld. Anschließend können Sie den Wert mit der Tastatur eingeben. Nach der Eingabe der Werte klicken Sie bitte auf den OK-Button. Die Dialogbox schliesst sich und das Territorium erscheint in der angegebenen Größe.

Nun werden wir den Frosch, der immer im Territorium sitzt, umplatzieren. Dazu klicken wir den Frosch an und ziehen (man spricht auch von „draggen“) ihn mit gedrückter Maustaste, auf die gewünschte Kachel. Dann lassen wir die Maustaste los.

Nun wollen wir aus einigen Graskacheln Wasserkacheln machen. Hierzu dient der Button „Neue Wasserkachel anlegen“ (10. Button von links). Wenn Sie ihn mit der Maus anklicken, erscheint am Maus-Cursor ein Wasserkachel-Symbol. Bewegen Sie die Maus nun über die Kachel, auf der eine Wasserkachel platziert werden soll, und klicken Sie dort die Maus. Auf der Kachel erscheint nun ein Wasserkachel-Symbol. Auf jeder Kachel kann übrigens höchstens eine Wasserkachel platziert werden.

Wenn Sie die Shift-Taste Ihrer Tastatur drücken und gedrückt halten und anschließend den Button „Neue Wasserkachel anlegen“ der Toolbar anklicken, haben Sie die Möglichkeit, mehrere Wasserkacheln im Territorium zu platzieren, ohne vorher jedes Mal erneut den Button anklicken zu müssen. Solange Sie die Shift-Taste gedrückt halten und eine Kachel anklicken, wird dort eine Wasserkachel platziert.

Mücken werden ähnlich wie Wasserkacheln auf Kacheln platziert. Nutzen Sie dazu den „Neue Mücke platzieren“-Button (11. Button von links). Auf jeder Kachel kann übrigens höchstens eine Mücke platziert werden.

Möchten Sie bestimmte Kacheln im Territorium wieder leeren, so dass weder eine Wasserkachel noch Mücken auf ihnen platziert sind, so aktivieren Sie den „Kachel löschen“-Button (12. Button von links). Klicken Sie anschließend auf die Kacheln, die geleert werden sollen. Der Button ist so lange aktiviert, bis er erneut oder ein anderer Gestaltungsbutton angeklickt wird.

Genauso wie der Frosch können auch Wasserkacheln und Mücken durch Anklicken und Bewegen der Maus bei gedrückter linker Maustaste im Territorium umplatziert werden.

Über das Menü „Bühne“ können Territorien auch in Datei gespeichert und später wieder geladen werden.

So, jetzt wissen Sie eigentlich alles, was notwendig ist, um das Frosch-Territorium nach Ihren Wünschen zu gestalten. Bevor Sie weiterlesen, erzeugen Sie als nächstes das in Abbildung 5.3 skizzierte Territorium.

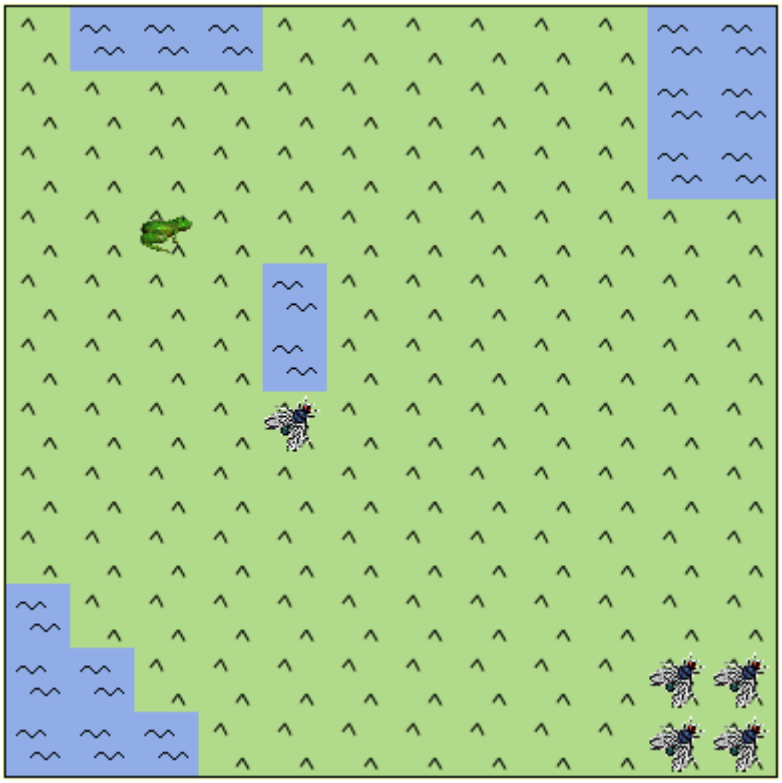


Abb. 5.3: Beispiel-Territorium

5.3 Eingeben eines Frosch-Programms

Nachdem wir unser erstes Frosch-Territorium im Simulationsbereich gestaltet haben, begeben wir uns nun in den Editor-Bereich. Dort werden wir unser erstes Frosch-Programm schreiben.

Unser erstes Programm soll bewirken, dass der Frosch in dem gerade von uns gestalteten Territorium eine Kachel mit einer Mücke aufsucht. Wir klicken in den Editor-Bereich und tippen dort wie in einem normalen Editor bzw.

Textverarbeitungsprogramm, wie Microsoft Word, die entsprechenden Frosch-Befehle ein, so dass letztlich folgendes im Eingabebereich steht:

```
void main() {
  huepfen();
  huepfen();
  rechtsUm();
  huepfen();
  schwimmen();
  schwimmen();
  huepfen();
}
```

Das ist unser erstes Frosch-Programm.

Ihnen sicher von anderen Editoren bzw. Textverarbeitungsprogrammen bekannte Funktionen, wie „Ausschneiden“, „Kopieren“, „Einfügen“, „Rückgängig“ und „Wiederherstellen“ können Sie über das Menü „Programm“ bzw. die entsprechenden Buttons in der Toolbar ausführen (vierter bis achter Button von links).

Weiterhin gibt es im Menü „Programm“ zwei Menü-Items zum Speichern von Programmen in Dateien und zum wieder Laden von abgespeicherten Programmen. Bei ihrer Aktivierung erscheint eine Dateiauswahl-Dialogbox, in der Sie die entsprechende Auswahl der jeweiligen Datei vornehmen müssen. Achtung: Wenn Sie eine abgespeicherte Datei laden, geht der Programmtext, der sich aktuell im Editorbereich befindet, verloren (wenn er nicht zuvor in einer Datei abgespeichert wurde).

Im Menü „Programm“ finden Sie darüber hinaus weitere nützliche Funktionen (Schriftgröße ändern, Drucken, …).

5.4 Compilieren eines Frosch-Programms

Nachdem wir unser Frosch-Programm geschrieben haben, müssen wir es compilieren. Der Compiler überprüft den Sourcecode auf syntaktische Korrektheit und transformiert ihn – wenn er korrekt ist – in ein ausführbares Programm. Zum Compilieren drücken Sie einfach auf den „Compilieren“-Button (erster Toolbar-Button von links oder „Compilieren“-Menü-Item im „Programm“-Menü). Compiliert wird dann das Programm, das gerade im Eingabebereich sichtbar ist. Es wird zuvor automatisch in einer (temporären) Datei (namens Solist.java) abgespeichert.

Wenn das Programm korrekt ist, erscheint eine Dialogbox mit der Nachricht „Compilierung erfolgreich“. Zur Bestätigung müssen Sie anschließend noch den OK-Button drücken. Das Programm kann nun ausgeführt werden. Merken Sie sich bitte:

Immer, wenn Sie Änderungen am Sourcecode Ihres Programms vorgenommen haben, müssen Sie es zunächst neu kompilieren.

Wenn das Programm syntaktische Fehler enthält – wenn Sie sich bspw. bei der Eingabe des obigen Programms vertippt haben –, werden unter dem Editor-Bereich die Fehlermeldungen des Compilers eingeblendet. Diese erscheinen in englischer Sprache. Weiterhin wird die Zeile angegeben, in der der Fehler entdeckt wurde. Wenn Sie mit der Maus auf die Fehlermeldung klicken, springt der Maus-Cursor im Editor-Bereich automatisch in die angegebene Zeile (siehe Abbildung 5.4).

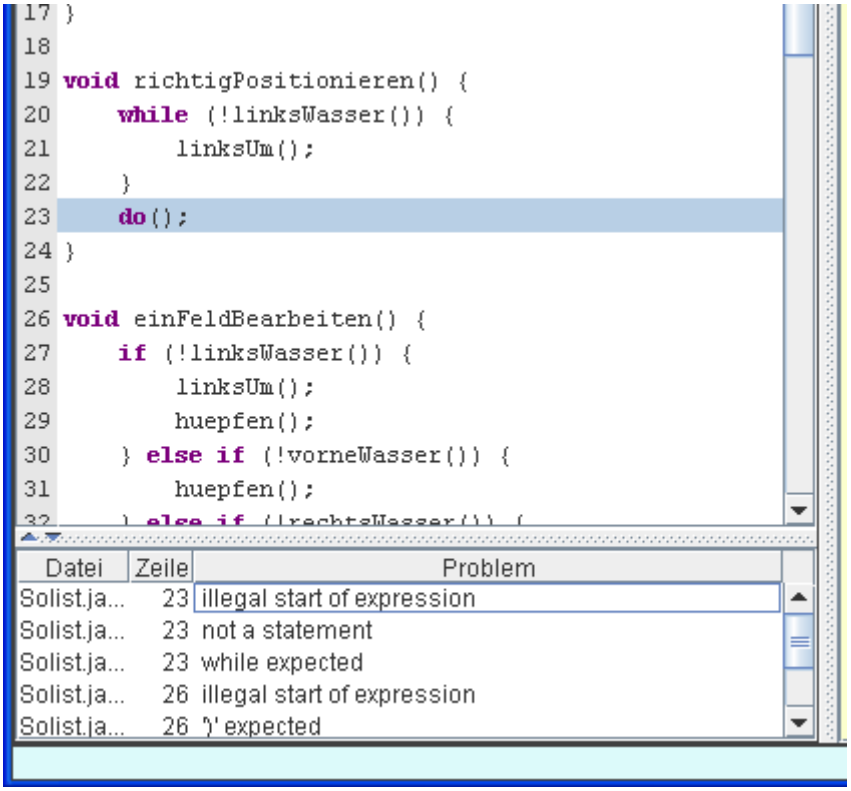


Abb. 5.4: Fehlermeldungen des Compilers

Vorsicht: Die Fehlermeldungen sowie die Zeilenangabe eines Compilers sind nicht immer wirklich exakt. Das Interpretieren der Meldungen ist für Programmieranfänger häufig nicht einfach und bedarf einiger Erfahrungen. Deshalb machen Sie ruhig am Anfang mal absichtlich Fehler und versuchen Sie, die Meldungen des Compilers zu verstehen.

Tipp: Arbeiten Sie die Fehler, die der Compiler entdeckt hat, immer von oben nach unten ab. Wenn Sie eine Meldung dann überhaupt nicht verstehen, compilieren Sie ruhig erst mal erneut. Häufig ist es (leider) so, dass der Compiler für einen einzelnen Fehler mehrere Fehlermeldungen ausgibt, was Anfänger leicht verwirren kann.

Nachdem Sie die Fehler korrigiert haben, müssen Sie das Programm erneut compilieren. Wiederholen Sie dies so lange, bis der Compiler die Meldung „Compilierung erfolgreich“ ausgibt. Erst dann können Sie das Programm ausführen!

5.5 Ausführen eines Frosch-Programms

Nach dem erfolgreichen Compilieren ist es endlich soweit: Wir können den Frosch bei der Arbeit beobachten. Macht er wirklich das, was wir ihm durch unser Programm beigebracht haben?

Zum Steuern der Programmausführung dienen die Buttons rechts in der Toolbar. Durch Anklicken des „Simulation starten“-Buttons (13. Button von links) starten wir das Programm. Wenn Sie bis hierhin alles richtig gemacht haben, sollte der Frosch loslaufen und wie im Programm beschrieben, die Mücke erreichen. Herzlichen Glückwunsch zu Ihrem ersten Frosch-Programm!

Wollen Sie die Programmausführung anhalten, können Sie dies durch Anklicken des „Simulation pausieren“-Buttons (15. Button von links) erreichen. Der Frosch pausiert so lange, bis Sie wieder den „Simulation starten/fortsetzen“-Button (13. Button von links) anklicken. Dann fährt der Frosch mit seiner Arbeit fort. Das Programm vorzeitig komplett abbrechen, können Sie mit Hilfe des „Simulation beenden“-Buttons (16. Button von links).

Wenn Sie ein Programm mehrmals hintereinander im gleichen Territorium ausführen wollen, können Sie mit dem „Rücksetzen“-Button (17. Button von links) den Zustand des Territoriums wieder herstellen, der vor dem letzten Ausführen des Programms Bestand hatte. Eine komplette Zurücksetzung des Territoriums auf den Zustand beim Öffnen des Frosch-Simulators ist mit dem „Komplett zurücksetzen“-Button möglich (18. Button von links).

Der Schieberegler ganz rechts in der Menüleiste dient zur Steuerung der Geschwindigkeit der Programmausführung. Je weiter Sie den Knopf nach links verschieben, umso langsamer erledigt der Frosch seine Arbeit. Je weiter Sie ihn nach rechts verschieben, umso schneller flitzt der Frosch durchs Territorium.

Die Bedienelemente zum Steuern der Programmausführung finden Sie übrigens auch im Menü „Simulation“.

5.6 Debuggen eines Frosch-Programms

„Debuggen eines Programms“ eines Programms bedeutet, dass Sie bei der Ausführung eines Programms zusätzliche Möglichkeiten zur Steuerung besitzen und sich den Zustand des Programms (welche Zeile des Sourcecodes wird gerade ausgeführt, welche Werte besitzen aktuell die Variablen) in bestimmten Situationen

anzeigen lassen können. Das ist ganz nützlich, wenn Ihr Programm nicht das tut, was es soll, und sie herausfinden wollen, woran der Fehler liegt.

Klicken Sie zum Debuggen zunächst den „Ablaufverfolgung aktivieren“-Button in der Toolbar an (19. Button von links). Es wird das so genannte Debugger-Fenster mit dem Titel „Debugger“ geöffnet. Im linken Bereich des Debugger-Fensters wird der Prozedur-Stack angezeigt, das ist die aktuelle Prozedur sowie die Prozeduren, aus denen die Prozedur heraus aufgerufen wurde. Im rechten Bereich werden – falls vorhanden – die gültigen Variablen und ihre aktuellen Werte dargestellt (siehe Abb. 5.5).

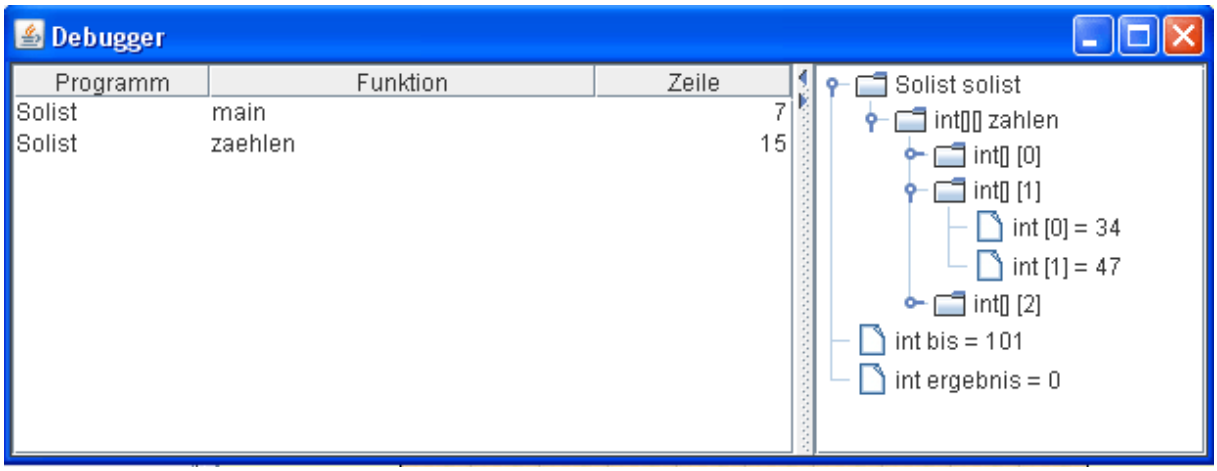


Abb. 5.5: Debugger-Fenster

Jetzt können Sie über den „Schritt“-Button (14. Toolbar-Button von links) jeden Befehl einzeln ausführen und bekommen im Editorbereich durch einen blauen Balken angezeigt, welcher Befehl bzw. welche Zeile als nächstes ausgeführt wird. Bei einem Prozeduraufruf wird dabei auch automatisch in die entsprechende Prozedur verzweigt. Im Debugger-Fenster wird zusätzlich der Prodzedur-Stack angezeigt, d.h. die aktuelle Prozedur sowie die Prozeduren, aus denen die Prozedur heraus aufgerufen wurde.

Sie können zunächst auch einfach das Programm durch Anklicken des „Starten“-Buttons starten und beobachten. Wenn Sie dann den „Pause“-Button drücken, haben Sie anschließend ebenfalls die Möglichkeit der schrittweisen Ausführung ab der aktuellen Position.

Die Ablaufverfolgung kann übrigens jederzeit durch erneutes Klicken des „Ablaufverfolgung“-Buttons deaktiviert bzw. reaktiviert werden.

Wenn Sie möchten, dass der Programmablauf beim Erreichen einer bestimmten Zeile automatisch in den Pausezustand gelangt, können Sie vor oder während des

Programmablaufs in der entsprechenden Zeile einen Breakpoint setzen. Führen Sie dazu im Editorbereich auf der entsprechenden Zeilennummer einen Doppelklick mit der Maus aus. Breakpoints werden durch eine violette Hinterlegung der Zeilennummer kenntlich gemacht. Durch erneuten Doppelklick oder über ein Popup-Menü, das oberhalb der Zeilennummern aktiviert werden kann, kann ein Breakpoint oder auch alle Breakpoints wieder gelöscht werden. Breakpoints nutzt man häufig dazu, dass man ein Programm bis zu einer bestimmten Stelle normal ablaufen lässt und ab dort dann die Möglichkeit der zeilenweisen Ausführung nutzt.

5.7 Zusammenfassung

Herzlichen Glückwunsch! Wenn Sie bis hierhin gekommen sind, haben Sie Ihr erstes Frosch-Programm erstellt und ausgeführt. Sie sehen, die Bedienung des Frosch-Simulators ist gar nicht so kompliziert.

Der Frosch-Simulator bietet jedoch noch weitere Möglichkeiten. Diese können Sie nun durch einfaches Ausprobieren selbst erkunden oder im nächsten Abschnitt im Detail nachlesen.

6 Bedienung des Frosch-Simulators

Im letzten Abschnitt haben Sie eine kurze Einführung in die Funktionalität des Frosch-Simulators erhalten. In diesem Abschnitt werden die einzelnen Funktionen des Simulators nun im Detail vorgestellt. Dabei wird sich natürlich einiges auch wiederholen.

Wenn Sie den Frosch-Simulator starten, öffnen sich ein Fenster mit dem Titel „Frosch-Simulator“. Abbildung 6.1 skizziert die einzelnen Komponenten des Fensters.

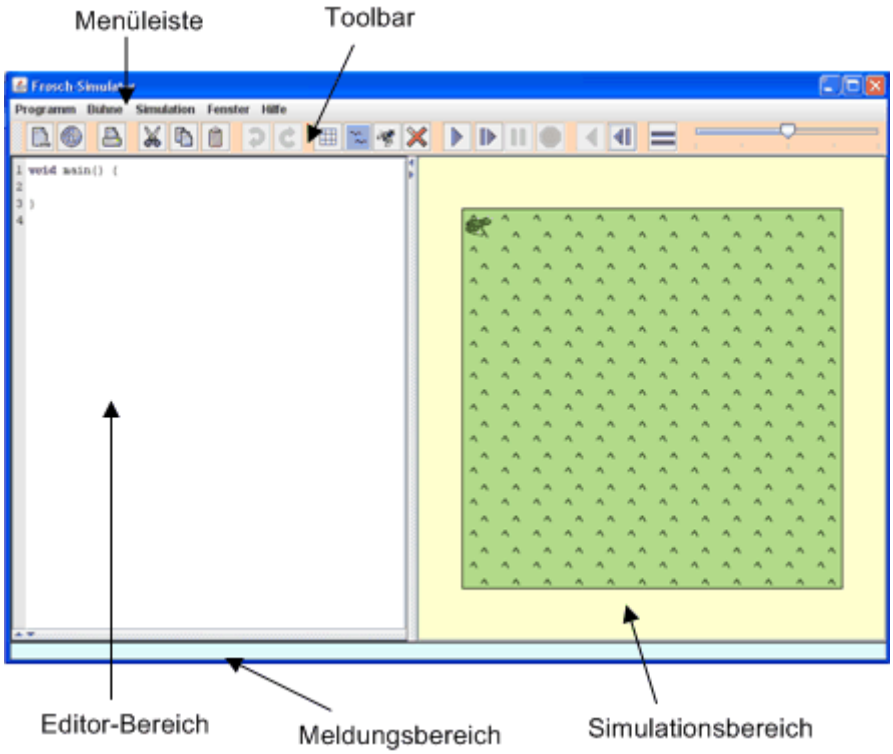


Abb. 6.1: Komponenten des Simulator-Fensters

Die Menüleiste oben im Fenster beinhaltet 5 Menüs. Darunter ist eine Toolbar mit Buttons platziert, über die die wichtigsten Funktionen der Menüs durch Anklicken eines Buttons schneller ausgeführt werden können. Ganz unten im Meldungsbereich werden wichtige Meldungen ausgegeben. Die Eingabe von Frosch-Programmen erfolgt im Editor-Bereich links und die Ausführung von Frosch-Programmen wird im Simulationsbereich (auch „Bühne“ genannt) visualisiert.

Als Hauptfunktionsbereiche des Frosch-Simulators lassen sich identifizieren:

- Verwalten und Editieren von Frosch-Programmen
- Compilieren von Frosch-Programmen
- Verwalten und Gestalten von Frosch-Territorien
- Interaktives Ausführen von Frosch-Befehlen
- Ausführen von Frosch-Programmen
- Debuggen von Frosch-Programmen

Bevor im Folgenden anhand dieser Funktionsbereiche der Simulator im Detail vorgestellt wird, werden zuvor noch einige Grundfunktionen graphischer Benutzungsoberflächen erläutert.

6.1 Grundfunktionen

In diesem Unterabschnitt werden einige wichtige Grundfunktionalitäten graphischer Benutzungsoberflächen beschrieben. Der Abschnitt ist für diejenigen von Ihnen gedacht, die bisher kaum Erfahrungen mit Computern haben. Diejenigen von Ihnen, die schon längere Zeit einen Computer haben und ihn regelmäßig benutzen, können diesen Abschnitt ruhig überspringen.

6.1.1 Anklicken

Wenn im Folgenden von „Anklicken eines Objektes“ oder „Anklicken eines Objektes mit der Maus“ gesprochen wird, bedeutet das, dass Sie den Maus-Cursor auf dem Bildschirm durch Verschieben der Maus auf dem Tisch über das Objekt platzieren und dann die – im Allgemeinen linke – Maustaste drücken.

6.1.2 Tooltipps

Als *Tooltipps* werden kleine Rechtecke bezeichnet, die automatisch auf dem Bildschirm erscheinen, wenn man den Maus-Cursor auf entsprechende Objekte platziert (siehe Abbildung 6.2). In den Tooltipps werden bestimmte Informationen ausgegeben.

Abb. 6.2: Tooltipp

6.1.3 Button

Buttons sind Objekte der Benutzungsoberfläche, die man anklicken kann und die daraufhin eine bestimmte Aktion auslösen (siehe Abbildung 6.3). Buttons besitzen

eine textuelle Beschreibung (z.B. „OK“) oder eine Graphik, die etwas über die Aktion aussagen. Sie erkennen Buttons an der etwas hervorgehobenen Darstellung. Graphik-Buttons sind in der Regel Tooltipps zugeordnet, die die zugeordnete Aktion beschreiben.

Abb. 6.3: Buttons

6.1.4 Menü

Menüs befinden sich ganz oben in einem Fenster in der so genannten *Menüleiste* (siehe Abbildung 6.4). Sie werden durch einen Text beschrieben (Programm, Bühne, Simulation, ...). Klickt man die Texte an, öffnet sich eine Box mit so genannten *Menü-Items*. Diese bestehen wiederum aus Texten, die man anklicken kann. Durch Anklicken von Menü-Items werden genauso wie bei Buttons Aktionen ausgelöst, die im Allgemeinen durch die Texte beschrieben werden (Speichern, Kopieren, ...). Nach dem Anklicken eines Menü-Items wird die Aktion gestartet und die Box schließt sich automatisch wieder. Klickt man irgendwo außerhalb der Box ins Fenster, schließt sich die Box ebenfalls und es wird keine Aktion ausgelöst.

Abb. 6.4: Menüleiste und Menü

Häufig steht hinter den Menü-Items ein weiterer Text, wie z.B. „Strg-O“ oder „Alt-N“. Diese Texte kennzeichnen Tastenkombinationen. Drückt man die entsprechenden Tasten auf der Tastatur, wird dieselbe Aktion ausgelöst, die man auch durch Anklicken des Menü-Items auslösen würde.

Manchmal erscheinen bestimmte Menü-Items etwas heller. Man sagt auch, sie sind ausgegraut. In diesem Fall kann man das Menü-Item nicht anklicken und die

zugeordnete Aktion nicht auslösen. Das Programm befindet sich in einem Zustand, in dem die Aktion keinen Sinn machen würde.

6.1.5 Toolbar

Direkt unterhalb der Menüleiste ist die so genannte *Toolbar* angeordnet (siehe Abbildung 6.5). Sie besteht aus einer Menge an Graphik-Buttons, die Alternativen zu den am häufigsten benutzten Menü-Items der Menüs darstellen.

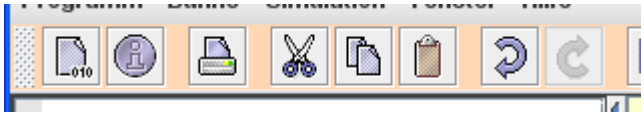


Abb. 6.5: Toolbar

6.1.6 Popup-Menü

Popup-Menüs sind spezielle Menüs, die bestimmten Elementen auf dem Bildschirm zugeordnet sind (siehe Abbildung 6.6). Man öffnet sie dadurch, dass man den Maus-Cursor auf das entsprechende Element verschiebt und danach die rechte Maustaste drückt. Genauso wie bei normalen Menüs erscheint dann eine Box mit Menü-Items.

Abb. 6.6: Popup-Menü

6.1.7 Eingabefeld

Eingabefelder dienen zur Eingabe von Zeichen (siehe Abbildung 6.7). Positionieren Sie dazu den Maus-Cursor auf das Eingabefeld und klicken Sie die Maus. Anschließend können Sie über die Tastatur Zeichen eingeben, die im Eingabefeld erscheinen.

6.1.8 Dialogbox

Beim Auslösen bestimmter Aktionen erscheinen so genannte *Dialogboxen* auf dem Bildschirm (siehe Abbildung 6.7). Sie enthalten in der Regel eine Menge von graphischen Objekten, wie textuelle Informationen, Eingabefelder und Buttons. Wenn

eine Dialogbox auf dem Bildschirm erscheint, sind alle anderen Fenster des Programms für Texteingaben oder Mausklicks gesperrt. Zum Schließen einer Dialogbox, d.h. um die Dialoxbox wieder vom Bildschirm verschwinden zu lassen, dienen in der Regel eine Menge an Buttons, die unten in der Dialogbox angeordnet sind. Durch Anklicken eines „OK-Buttons“ wird dabei die der Dialogbox zugeordnete Aktion ausgelöst. Durch Anklicken des „Abbrechen-Buttons“ wird eine Dialogbox geschlossen, ohne dass irgendwelche Aktionen ausgelöst werden.

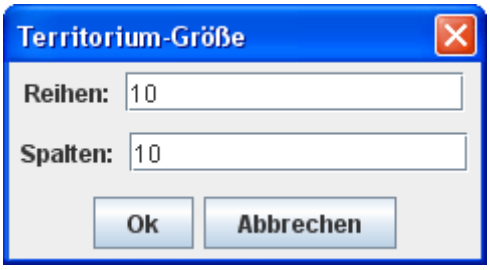


Abb. 6.7: Dialogbox mit Eingabefeldern

6.1.9 Dateiauswahl-Dialogbox

Dateiauswahl-Dialogboxen sind spezielle Dialogboxen, die zum Speichern und Öffnen von Dateien benutzt werden (siehe Abbildung 6.8). Sie spiegeln im Prinzip das Dateisystem wider und enthalten Funktionalitäten zum Verwalten von Dateien und Ordnern.

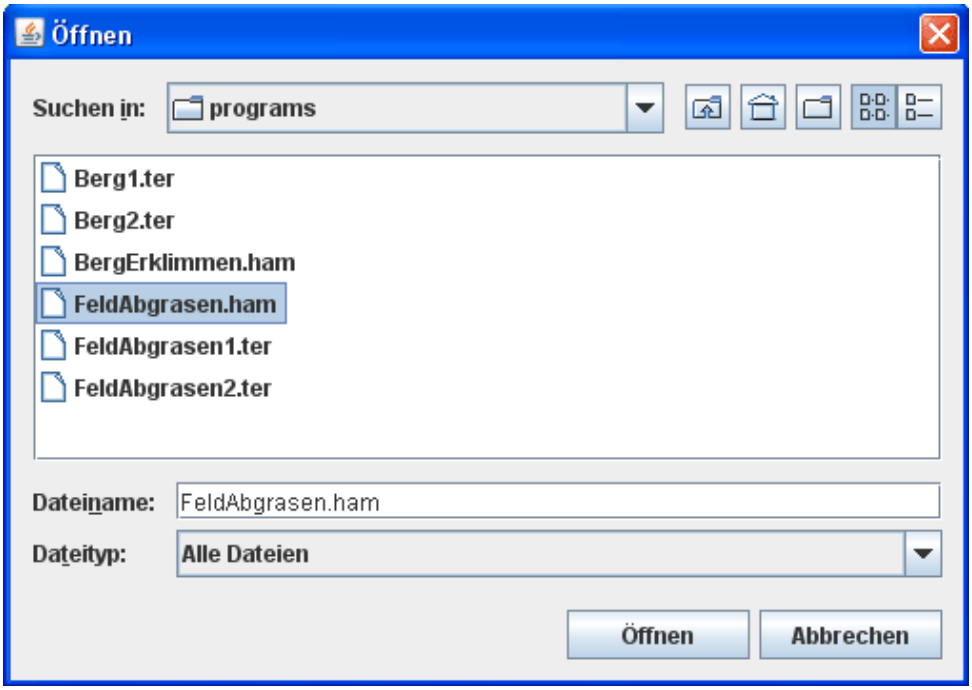


Abb. 6.8: Dateiauswahl-Dialogbox

Im mittleren Bereich einer Dateiauswahl-Dialogbox erscheinen alle Dateien und Unterordner des aktuellen Ordners. Sie sind durch unterschiedliche Symbole repräsentiert. Der eigentliche Zweck von Dateiauswahl-Dialogboxen ist – wie der Name schon sagt – die Auswahl einer Datei. Klickt man auf eine Datei, erscheint der Name automatisch im Eingabefeld „Dateiname“. Dort kann man auch über die Tastatur einen Dateinamen eingeben. Anschließend wird nach Drücken des OK-Buttons die entsprechende Datei geöffnet bzw. gespeichert.

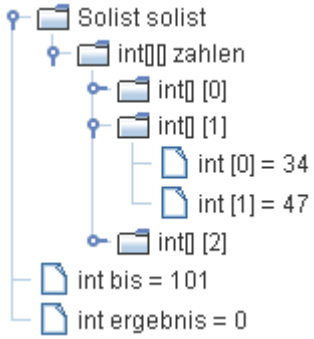
Dateiauswahl-Dialogboxen stellen jedoch noch zusätzliche Funktionalitäten bereit. Durch Doppelklick auf einen Ordner kann man in den entsprechenden Ordner wechseln. Es werden dann anschließend die Dateien und Unterordner dieses Ordners im mittleren Bereich angezeigt. Um zu einem übergeordneten Ordner zurück zu gelangen, bedient man sich des Menüs „Suchen in“, in dem man den entsprechenden Ordner auswählen kann.

Neben dem „Suchen in“-Menü sind noch fünf Graphik-Buttons angeordnet. Durch Anklicken des linken Buttons kommt man im Ordnerbaum eine Ebene höher. Durch Anklicken des zweiten Buttons von links gelangt man zur Wurzel des Ordnerbaumes. Mit dem mittleren Button kann man im aktuellen Ordner einen neuen Unterordner anlegen. Mit den beiden rechten Buttons kann man die Darstellung im mittleren Bereich verändern.

Möchte man einen Ordner oder eine Datei umbenennen, muss man im mittleren Bereich der Dateiauswahl-Dialogbox zweimal – mit Pause zwischendurch – auf den Namen des Ordners oder der Datei klicken. Die textuelle Darstellung des Namens wird dann zu einem Eingabefeld, in der man über die Tastatur den Namen verändern kann.

6.1.10 Elementbaum

Ein *Elementbaum* repräsentiert Elemente und strukturelle Beziehungen zwischen ihnen, bspw. die Ordner und Dateien des Dateisystems (siehe Abbildung 6.9).



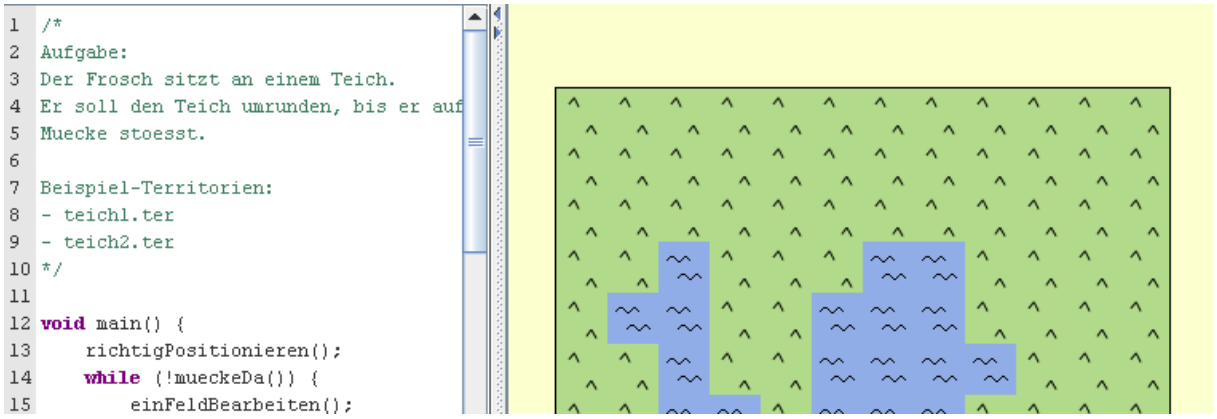
Abbi. 6.9: Elementbaum mit Verzeichnissen und Dateien

Unterschiedliche Elementtypen werden dabei durch unterschiedliche Symbole dargestellt, hinter denen entsprechende Bezeichnungen erscheinen. Durch Anklicken der Symbole auf der linken Seite kann man Strukturen öffnen und schließen, d.h. Unterstrukuren sichtbar bzw. unsichtbar machen.

Den Ordnern und Dateien sind Popup-Menüs zugeordnet. Um diese zu öffnen, muss man zunächst den Ordner bzw. die Datei mit der Maus anklicken. Der Name wird dann durch einen blauen Balken hinterlegt. Anschließend muss man die rechte Maustaste drücken. Dann öffnet sich das Popup-Menü. Die Popup-Menüs enthalten bspw. Menü-Items zum Löschen und Umbenennen des entsprechenden Ordners bzw. der entsprechenden Datei.

6.1.11 Split-Pane

Eine Split-Pane ist ein Element, das aus zwei Bereichen und einem Balken besteht. (siehe Abbildung 6.10). Die beiden Bereiche können dabei links und rechts oder oberhalb und unterhalb des Balkens liegen. Wenn Sie den Balken mit der Maus anklicken und bei gedrückter Maustaste nach links oder rechts (bzw. oben oder unten) verschieben, vergrößert sich einer der beiden Bereiche und der andere verkleinert sich. Durch Klicken auf einen der beiden Pfeile auf dem Balken können Sie einen der beiden Bereiche auch ganz verschwinden lassen.

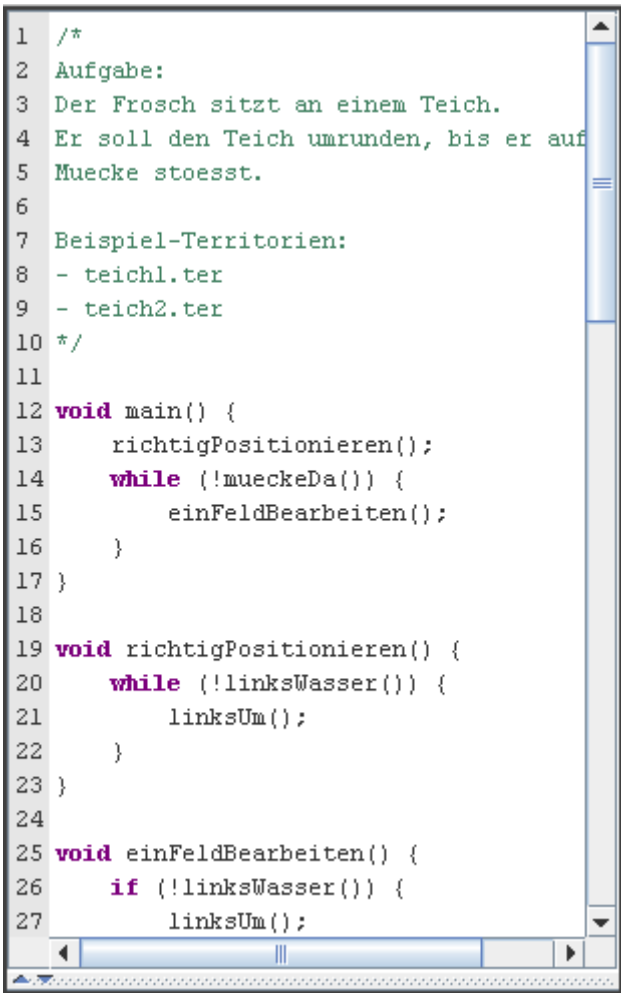


Abbi. 6.10: Split-Pane

6.2 Verwalten und Editieren von Frosch-Programmen

Das Schreiben von Programmen bzw. genauer gesagt das Schreiben des Sourcecodes von Programmen bezeichnet man als *Editieren*. Im Frosch-Simulator

dient der Editor-Bereich zum Editieren von Frosch-Programmen (siehe Abbildung 6.11)



```
1  /*
2  Aufgabe:
3  Der Frosch sitzt an einem Teich.
4  Er soll den Teich umrunden, bis er auf
5  Muecke stoesst.
6
7  Beispiel-Territorien:
8  - teich1.ter
9  - teich2.ter
10 */
11
12 void main() {
13     richtigPositionieren();
14     while (!mueckeDa()) {
15         einFeldBearbeiten();
16     }
17 }
18
19 void richtigPositionieren() {
20     while (!linksWasser()) {
21         linksUm();
22     }
23 }
24
25 void einFeldBearbeiten() {
26     if (!linksWasser()) {
27         linksUm();
```

Abb. 6.11: Editor-Bereich des Frosch-Simulators

Im Editor-Bereich können Sie Programme eintippen. Für das Verwalten und Editieren von Programmen ist das Menü „Programm“ wichtig. Unterhalb der Menüleiste ist eine spezielle Toolbar zu sehen, über die Sie die wichtigsten Funktionen der Menüs auch schneller erreichen und ausführen können. Schieben Sie einfach mal die Maus über die Buttons. Dann erscheint jeweils ein Tooltipp, der die Funktionalität des Buttons anzeigt. Die für das Editieren von Programmen wichtigen Buttons der Toolbar werden in Abbildung 6.12 skizziert.

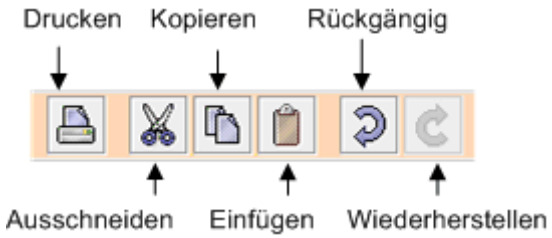


Abb. 6.12: Editor-Buttons der Toolbar

6.2.1 Schreiben eines neuen Frosch-Programms

Das Schreiben eines neuen Frosch-Programms ist durch das entsprechende Eintippen des Sourcecodes im Editor-Bereich möglich.

6.2.2 Ändern des aktuellen Frosch-Programms

Möchten Sie Teile des aktuellen Frosch-Programms ändern, klicken Sie im Editor-Bereich einfach an die entsprechende Stelle und fügen dort die entsprechenden Wörter ein oder löschen sie.

6.2.3 Löschen des aktuellen Frosch-Programm

Komplett löschen können Sie das aktuelle Frosch-Programm des Editor-Bereichs, indem Sie den kompletten Sourcecode mit der Maus selektieren und dann in der Toolbar den „Ausschneiden“-Button (4. Button von links) drücken.

6.2.4 Abspeichern des aktuellen Frosch-Programms

Normalerweise müssen Sie sich nicht um das Speichern des aktuellen Programms kümmern. Es wird automatisch vor dem Compilieren in einer internen Datei abgespeichert. Wenn Sie jedoch ein Programm explizit abspeichern möchten, können Sie im „Programm“-Menü das „Speichern unter…“-Menü-Item anklicken. Es öffnet sich eine Dateiauswahl-Dialogbox, über die Sie die gewünschte Datei auswählen können.

6.2.5 Öffnen eines einmal abgespeicherten Frosch-Programms

Möchten Sie ein einmal abgespeichertes Frosch-Programm wieder in den Editorbereich laden, können Sie dies über das Menü-Item „Laden…“ des „Programm“-Menüs tun. Nach dem Anklicken des Items erscheint eine Dateiauswahl-Dialogbox, über die Sie die gewünschte Datei auswählen können.

Achtung: Beim Laden einer abgespeicherten Datei geht der aktuelle Inhalt des Editor-Bereichs verloren! Sie müssen ihn also gegebenenfalls vorher in einer Datei abspeichern.

6.2.6 Drucken eines Frosch-Programms

Über den „Drucken“-Button (3. Toolbar-Button von links) können Sie das aktuelle Programm des Editor-Bereichs drucken. Es öffnet sich eine Dialogbox, in der Sie die entsprechenden Druckeinstellungen vornehmen und den Druck starten können.

6.2.7 Editier-Funktionen

Im Editor-Bereich können Sie – wie bei anderen Editoren auch – über die Tastatur Zeichen eingeben bzw. wieder löschen. Darüber hinaus stellt der Editor ein paar weitere Funktionalitäten zur Verfügung, die über das „Programm“-Menü bzw. die entsprechenden Editor-Buttons in der Toolbar aktiviert werden können.

- „Ausschneiden“-Button (4. Toolbar-Button von links): Hiermit können Sie komplette Passagen des Editor-Bereichs in einem Schritt löschen. Markieren Sie die zu löschende Passage mit der Maus und klicken Sie dann den Button an. Der markierte Text verschwindet.
- „Kopieren“-Button (5. Toolbar-Button von links): Hiermit können Sie komplette Passagen des Editor-Bereichs in einen Zwischenpuffer kopieren. Markieren Sie die zu kopierende Passage mit der Maus und klicken Sie dann den Button an.
- „Einfügen“-Button (6. Toolbar-Button von links): Hiermit können Sie den Inhalt des Zwischenpuffers an die aktuelle Cursorposition einfügen. Wählen Sie zunächst die entsprechende Position aus und klicken Sie dann den Button an. Der Text des Zwischenpuffers wird eingefügt.
- „Rückgängig“-Button (7. Toolbar-Button von links): Wenn Sie durchgeführte Änderungen des Sourcecode – aus welchem Grund auch immer – wieder rückgängig machen wollen, können Sie dies durch Anklicken des Buttons bewirken.
- „Wiederherstellen“-Button (8. Toolbar-Button von links): Rückgängig gemachte Änderungen können Sie mit Hilfe dieses Buttons wieder herstellen.

Die Funktionalitäten „Kopieren“ und „Einfügen“ funktionieren übrigens auch über einzelne Programme hinaus. Es ist sogar möglich, mit Hilfe der Betriebssystem-Kopieren-Funktion Text aus anderen Programmen (bspw. Microsoft Word) zu kopieren und hier einzufügen.

Die gerade aufgelisteten Funktionen finden Sie auch im „Programm“-Menü. Als zusätzliche Funktionalitäten werden dort angeboten:

- Einrückung: Durch Anklicken des Menü-Items wird der Einrück-Modus aktiviert bzw. deaktiviert. Ist der Einrück-Modus aktiviert, werden beim Eingeben von Text im Editor-Bereich automatisch Einrückungen an die entsprechende Spalte vorgenommen, wenn Sie die Enter-Taste drücken.
- Schriftgröße: Hierüber können Sie die Schriftgröße des Editor-Bereichs anpassen.

6.3 Compilieren von Frosch-Programmen

Beim Compilieren werden Programme – genauer gesagt der Sourcecode – auf ihre (syntaktische) Korrektheit überprüft und im Erfolgsfall ausführbare Programme erzeugt. Zum Compilieren von Programmen dient im Frosch-Simulator der „Compilieren“-Button (erster Button der Toolbar von links) bzw. das Menü-Item „Compilieren“ im „Programm“-Menü (siehe Abbildung 6.13).

An der Farbe des Compilier-Buttons können Sie erkennen, ob Compilieren aktuell notwendig ist oder nicht. Immer wenn Sie Änderungen im Editor-Bereich vorgenommen haben, erscheint der Button rot, und die Änderungen werden erst dann berücksichtigt, wenn Sie (erneut) compiliert haben. Erscheint der Button in einer neutralen Farbe, ist kein Compilieren notwendig.

6.3.1 Compilieren

Wenn Sie den „Compilieren“-Button anklicken, wird das Programm, das gerade im Editor-Bereich sichtbar ist, in einer internen Datei (mit dem Namen „Solist.java“) abgespeichert und kompiliert.

Wenn Ihr Programm syntaktisch korrekt ist, erscheint nach ein paar Sekunden eine Dialogbox mit einer entsprechenden Meldung. Es wurde ein (neues) ausführbares Programm erzeugt.

6.3.2 Beseitigen von Fehlern

Wenn Ihr Programm Fehler enthält, öffnet sich unterhalb des Editor-Bereichs in einer Scroll-Pane ein neuer Bereich, der die Fehlermeldungen des Compilers anzeigt (siehe Abbildung 6.13). Es wurde kein (neues) ausführbares Programm erzeugt! Jede Fehlermeldung erscheint in einer eigenen Zeile. Jede Zeile enthält eine Beschreibung des (wahrscheinlichen) Fehlers sowie die entsprechende Zeile der Anweisung im Programm. Wenn Sie eine Fehlermeldung anklicken, wird die entsprechende Anweisung im Eingabebereich blau markiert und der Maus-Cursor an die entsprechende Stelle gesetzt. Sie müssen nun die einzelnen Fehler beseitigen und dann erneut speichern und compilieren, bis Ihr Programm keine Fehler mehr enthält. Der Fehlermeldungsbereich schließt sich dann automatisch wieder.

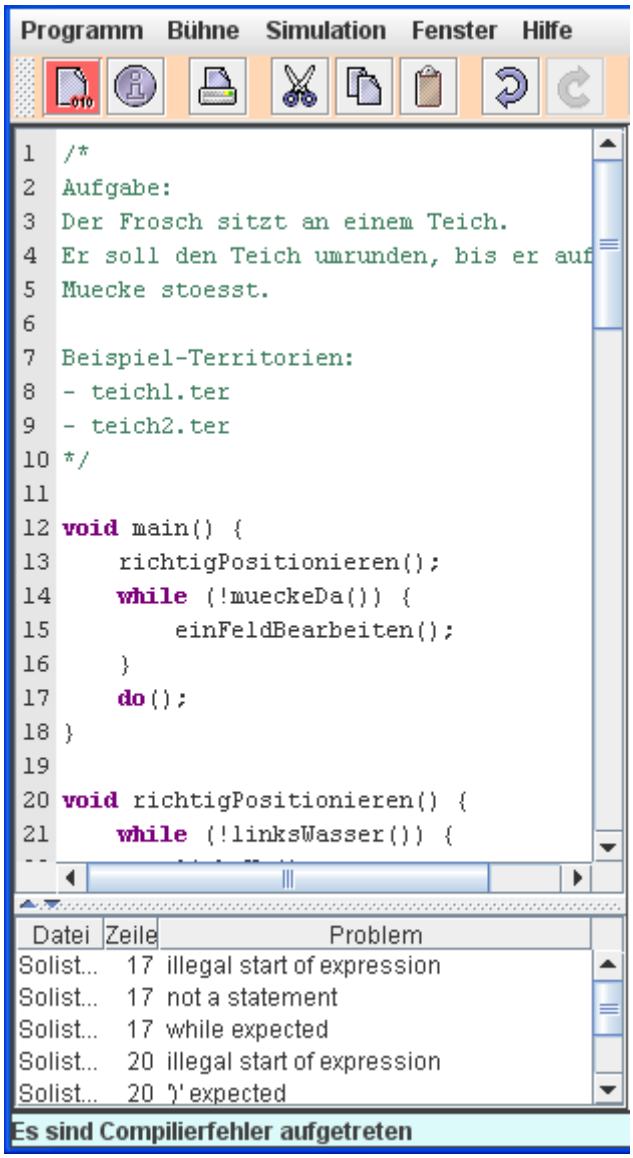


Abb. 6.13: Fehlermeldungen des Compilers

Achtung: Die Interpretation von Fehlermeldungen, die der Compiler ausgibt, ist nicht trivial. Die Meldungen sind nicht immer besonders präzise und oft auch irreführend. Häufig gibt der Compiler mehrere Fehlermeldungen aus, obwohl es sich nur um einen einzelnen Fehler handelt. Deshalb beherzigen Sie gerade am Anfang folgende Hinweise: Arbeiten Sie die Fehlermeldungen immer von oben nach unten ab. Wenn der Compiler eine große Menge von Fehlermeldungen liefert, korrigieren Sie zunächst nur eine Teilmenge und kompilieren Sie danach erneut. Bauen Sie – gerade als Programmieranfänger – auch mal absichtlich Fehler in Ihre Programme ein und schauen Sie sich dann die Fehlermeldungen des Compilers an.

6.4 Verwalten und Gestalten von Frosch-Territorien

Das Frosch-Territorium umfasst standardmäßig 12 Reihen und 12 Spalten. Der Frosch steht auf der Kachel ganz oben links, also der Kachel mit den Koordinaten (0/0). Er schaut nach Osten.

In der Toolbar dienen die Buttons 9 – 14 von links zum Gestalten des Frosch-Territoriums. Über das Menü „Bühne“ können Frosch-Territorien verwaltet werden (siehe auch Abbildung 6.14).

Abb. 6.14: Territorium-Buttons der Toolbar

Normalerweise sollten Sie ein Territorium vor der Ausführung eines Programms gestalten. Es ist jedoch auch möglich, während der Programmausführung noch Umgestaltungen vorzunehmen.

6.4.1 Verändern der Größe des Frosch-Territoriums

Durch Anklicken des „Größe ändern“-Buttons (9. Toolbar-Button von links) können Sie die Größe des Territoriums verändern. Es öffnet sich eine Dialogbox mit zwei Eingabefelder, in denen Sie die gewünschte Reihen- und Spaltenanzahl eingeben können. Nach Drücken des OK-Buttons schließt sich die Dialogbox und das Territorium erscheint in der angegebenen Größe.

Achtung: Steht der Frosch beim Verkleinern des Territoriums auf einer Kachel außerhalb der neuen Territoriumsgröße, wird er auf die Kachel (0/0) versetzt.

6.4.2 Umplatzieren des Froschs im Frosch-Territorium

Um den Frosch im Frosch-Territorium auf eine andere Kachel zu platzieren, klicken Sie ihn mit der Maus an und ziehen ihn bei gedrückter Maustaste auf die gewünschte Kachel.

6.4.3 Setzen der Blickrichtung des Froschs

Um die Blickrichtung des Froschs zu ändern, öffnen Sie bitte über das „Fenster“-Menü das Befehlsfenster und klicken dort den rechtsum- oder linksUm-Befehl an.

6.4.4 Platzieren von Wasserkacheln im Frosch-Territorium

Um auf einer Kachel des Frosch-Territoriums eine Wasserkachel zu platzieren, klicken Sie in der Toolbar den „Wasserkachel setzen“-Button (10. Toolbar-Button von links). Klicken Sie anschließend die entsprechende Kachel an. Insofern sich dort noch keine Wasserkachel befindet, wird eine Wasserkachel dort platziert.

Um nicht für jedes Platzieren einer Wasserkachel erneut den „Wasserkachel setzen“-Button anklicken zu müssen, können Sie auch folgendes tun: Drücken Sie die Shift-Taste der Tastatur und Klicken bei gedrückter Shift-Taste den „Wasserkachel setzen“-Button an. Solange Sie nun die Shift-Taste gedrückt halten, können Sie durch Anklicken einer Kachel des Frosch-Territoriums dort eine Wasserkachel platzieren.

Wasserkachel können auch im Territorium umplaziert werden. Klicken Sie dazu die entsprechende Kachel an und ziehen Sie die dortige Wasserkachel bei gedrückter Maustaste auf die gewünschte Kachel.

6.4.5 Platzieren von Mücken im Frosch-Territorium

Um auf einer Kachel des Frosch-Territoriums eine Mücke zu platzieren, klicken Sie in der Toolbar den „Mücke setzen“-Button (11. Toolbar-Button von links). Klicken Sie anschließend die entsprechende Kachel an. Insofern sich dort noch keine Mücke befindet, wird eine Mücke dort platziert.

Um nicht für jedes Platzieren einer Mücke erneut den „Mücke setzen“-Button anklicken zu müssen, können Sie auch folgendes tun: Drücken Sie die Shift-Taste der Tastatur und Klicken bei gedrückter Shift-Taste den „Mücke setzen“-Button an. Solange Sie nun die Shift-Taste gedrückt halten, können Sie durch Anklicken einer Kachel des Frosch-Territoriums dort eine Mücke platzieren.

Mücke können auch im Territorium umplaziert werden. Klicken Sie dazu die entsprechende Kachel an und ziehen Sie die dortige Mücke bei gedrückter Maustaste auf die gewünschte Kachel.

6.4.6 Löschen von Kacheln des Frosch-Territorium

Um einzelne Kacheln des Frosch-Territoriums zu löschen, d.h. gegebenenfalls vorhandene Wasserkacheln bzw. Mücken zu entfernen, müssen Sie zunächst in der

Toolbar den „Kachel löschen“-Button (12. Toolbar-Button von links) anklicken. Dadurch aktivieren Sie die Kachel-Löschen-Funktion. Sie erkennen dies daran, dass der Hintergrund des Buttons nun dunkler erscheint. Solange die Funktion aktiviert ist, können Sie nun durch Anklicken einer Kachel diese Kachel löschen.

Eine Deaktivierung der Kachel-Löschen-Funktion ist durch erneutes Anklicken des „Kachel löschen“-Buttons möglich. Eine Deaktivierung erfolgt automatisch, wenn ein anderer der Buttons zum Territorium-Gestalten gedrückt wird.

Eine weitere Möglichkeit, Wasserkacheln oder Mücken im Territorium wieder zu löschen, besteht darin, über den entsprechenden Elementen mit Hilfe der rechten Maustaste ein Popup-Menü zu aktivieren und das darin erscheinende Item „Löschen“ anzuklicken.

6.4.7 Abspeichern eines Frosch-Territoriums

Sie können einmal gestaltete Frosch-Territorien in einer Datei abspeichern und später wieder laden. Zum Abspeichern des aktuellen Territoriums aktivieren Sie im Menü „Bühne“ das Menü-Item „Speichern unter…“. Es öffnet sich eine Dateiauswahl-Dialogbox. Hierin können Sie den Ordner auswählen und den Namen einer Datei eingeben, in die das aktuelle Territorium gespeichert werden soll.

Weiterhin ist es möglich, das aktuell Territorium als Bild (gif- oder png-Datei) abzuspeichern. Eine entsprechende Funktion findet sich im „Bühne“-Menü.

6.4.8 Wiederherstellen eines abgespeicherten Frosch-Territoriums

Abgespeicherte Frosch-Territorien können mit dem „Laden“-Menü-Item des „Bühne“-Menüs wieder geladen werden. Es erscheint eine Dateiauswahl-Dialogbox, in der Sie die zu ladende Datei auswählen können. Nach dem Anklicken des OK-Buttons schließt sich die Dialogbox und das entsprechende Frosch-Territorium ist wiederhergestellt.

Achtung: Der Zustand des Frosch-Territoriums, der vor dem Ausführen der Territorium-Laden-Funktion Gültigkeit hatte, geht dabei verloren. Speichern Sie ihn daher gegebenenfalls vorher ab.

6.5 Interaktives Ausführen von Frosch-Befehlen

Sie können einzelne Frosch-Befehle oder selbst definierte Funktionen und Prozeduren bspw. zu Testzwecken auch interaktiv ausführen. Aktivieren Sie dazu im Menü „Fenster“ den Eintrag „Befehlsfenster“. Es öffnet sich das so genannte Befehlsfenster (siehe Abbildung 6.15).

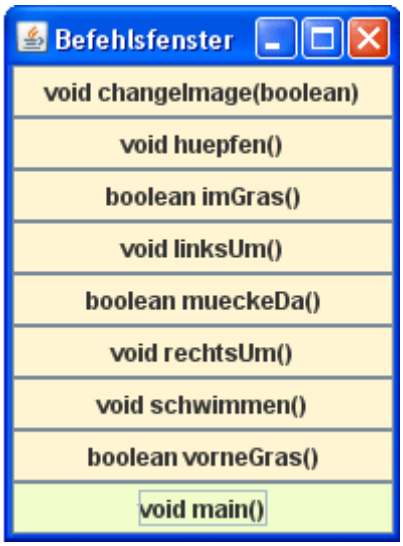


Abb. 6.15: Befehlsfenster

6.5.1 Befehlsfenster

Im Befehlsfenster werden die 7 Frosch-Befehle (mit grünlichem Hintergrund) sowie die Prozeduren und Funktionen dargestellt, die im aktuellen Frosch-Programm beim letztmaligen erfolgreichen Compilieren definiert waren (mit orangenem Hintergrund).

Beim Anklicken mit der Maus werden die entsprechenden Befehle jeweils ausgeführt. Die Ausführung erfolgt dabei ohne Anzeige von Zwischenzuständen. D.h. wird bspw. eine Prozedur `kehrt` dadurch definiert, dass zweimal der Befehl `linksUm` aufgerufen wird, und wird diese Prozedur `kehrt` im Befehlsfenster aktiviert, dreht sich der Frosch tatsächlich nur einmal um 180 Grad um.

Dauert die Ausführung eines Befehls mehr als 5 Sekunden (vermutlich enthält dann die Funktion eine Endlosschleife), wird der Befehl abgebrochen und der Frosch-Simulator wird komplett auf seinen Startzustand zurückgesetzt.

6.5.2 Parameter

Besitzt eine Prozedur oder Funktion Parameter, erscheint nach ihrer Aktivierung im Befehlsfenster eine Dialogbox, in der die entsprechenden aktuellen Parameterwerte eingegeben werden müssen. Hinweis: Aktuell wird nur die Eingabe von Werten der Java-Standard-Datentypen (`int`, `boolean`, `float`, ...) sowie die Eingabe von Zeichenketten (Strings) unterstützt.

6.5.3 Rückgabewerte von Funktionen

Bei der Ausführung von Funktionen wird der jeweils gelieferte Wert in einem Dialogfenster dargestellt.

6.5.4 Befehls-Popup-Menü

Alternativ zur Benutzung des Befehlsfensters ist es auch möglich, über ein Popup-Menü die Frosch-Befehle interaktiv auszuführen. Sie können dieses Popup-Menü durch Drücken der rechten Maustaste oberhalb des Froschs im Territorium aktivieren (siehe Abbildung 6.16).

Abb. 6.16: Befehls-Popup-Menü

6.6 Ausführen von Frosch-Programmen

Ausgeführt werden können (erfolgreich compilierte) Frosch-Programme mit Hilfe der in Abbildung 6.17 skizzierten Buttons der Toolbar. Alle Funktionen sind darüber hinaus auch über das Menü „Simulation“ aufrufbar.

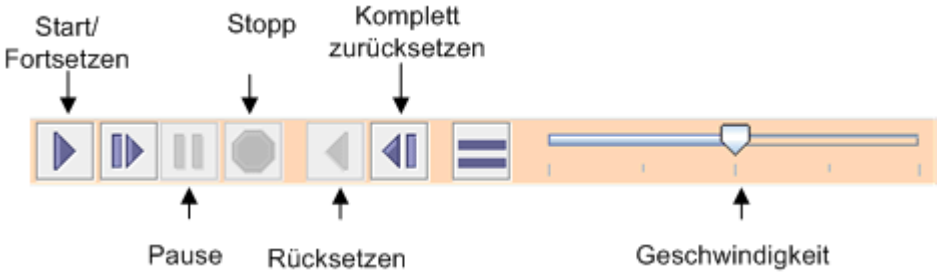


Abb. 6.17: Simulationsbuttons der Toolbar

6.6.1 Starten eines Frosch-Programms

Bevor ein Frosch-Programm ausgeführt werden kann, muss es erfolgreich compiliert worden sein. Gestartet werden kann das aktuelle Frosch-Programm dann durch Anklicken des „Start/Fortsetzen“-Buttons (13. Toolbar-Button von links).

Nach dem Starten eines Frosch-Programms wird der Frosch im Frosch-Territorium aktiv und tut das, was das Programm ihm vorgibt. Während des Ausführens eines Frosch-Programms wird der Editor-Bereich ausgegraut, d.h. es können während der Ausführung eines Programms keine Änderungen am Sourcecode durchgeführt werden.

6.6.2 Stoppen eines Frosch-Programms

Die Ausführung eines Frosch-Programms kann durch Anklicken des „Stopp“-Buttons (16. Button der Toolbar von links) jederzeit abgebrochen werden.

6.6.3 Pausieren eines Frosch-Programms

Möchten Sie ein in Ausführung befindliches Programm (kurzfristig) anhalten, können Sie dies durch Anklicken des „Pause“-Buttons (15. Button der Toolbar von links) tun. Wenn Sie anschließend auf den „Start/Fortsetzen“-Button klicken, wird die Programmausführung fortgesetzt.

6.6.4 Während der Ausführung eines Frosch-Programms

Treten bei der Ausführung eines Programms Laufzeitfehler auf, z.B. wenn ein Frosch versucht auf einer Wasserkachel anstatt zu schwimmen zu hüpfen, wird eine Dialogbox geöffnet, die eine entsprechende Fehlermeldung enthält. Nach dem Anklicken des OK-Buttons in der Dialogbox wird das Frosch-Programm beendet. Weiterhin öffnet sich das Konsolen-Fenster, in dem ebenfalls die Fehlermeldung ausgegeben wird.

Während der Ausführung eines Frosch-Programms ist es durchaus noch möglich, das Territorium umzugestalten, also bspw. neue Wasserkacheln und Mücken zu platzieren.

6.6.5 Einstellen der Geschwindigkeit

Mit dem Schieberegler ganz rechts in der Toolbar können Sie die Geschwindigkeit der Programmausführung beeinflussen. Je weiter links der Regler steht, desto langsamer wird das Programm ausgeführt. Je weiter Sie den Regler nach rechts verschieben, umso schneller flitzt der Frosch durchs Territorium.

6.6.6 Wiederherstellen eines Frosch-Territoriums

Beim Testen eines Programms recht hilfreich ist der „Rücksetzen“-Button (17. Button der Toolbar von links). Sein Anklicken bewirkt, dass das Frosch-Territorium in den Zustand zurückversetzt wird, den es vor dem letztmaligen Start eines Programms inne hatte.

Über den „Komplett Zurücksetzen“-Button (18. Button der Toolbar von links) ist eine Rücksetzung des Territoriums in den Zustand möglich, der beim Öffnen des Frosch-Simulators gültig war. Sollte es irgendwann einmal bei der Benutzung des Frosch-Simulators zu unerklärlichen Fehlern können, ist es mit Hilfe dieses Buttons möglich, den Frosch-Simulator zu reinitialisieren.

6.7 Debuggen von Frosch-Programmen

Debugger sind Hilfsmittel zum Testen von Programmen. Sie erlauben es, während der Programmausführung den Zustand des Programms zu beobachten und gegebenenfalls sogar interaktiv zu ändern. Damit sind Debugger sehr hilfreich, wenn es um das Entdecken von Laufzeitfehlern und logischen Programmfehlern geht.

Der Debugger des Frosch-Simulators ermöglicht während der Ausführung eines Frosch-Programms das Beobachten des Programmzustands. Sie können sich während der Ausführung eines Frosch-Programms anzeigen lassen, welche Anweisung des Sourcecodes gerade ausgeführt wird und welche Werte die Variablen aktuell speichern. Die interaktive Änderung von Variablenwerten wird aktuell nicht unterstützt.

Die Funktionen des Debuggers sind eng mit den Funktionen zur Programmausführung verknüpft. Sie finden die Funktionen im Menü „Simulation“. Es bietet sich jedoch an, die entsprechenden Buttons der Toolbar zu verwenden. Neben dem „Start/Fortsetzen“-, dem „Pause“- und dem „Stopp“-Button gehören die zwei Buttons „Schrittweise Ausführung“ und „Ablaufverfolgung“ zu den Debugger-Funktionen (siehe auch Abbildung 6.18).

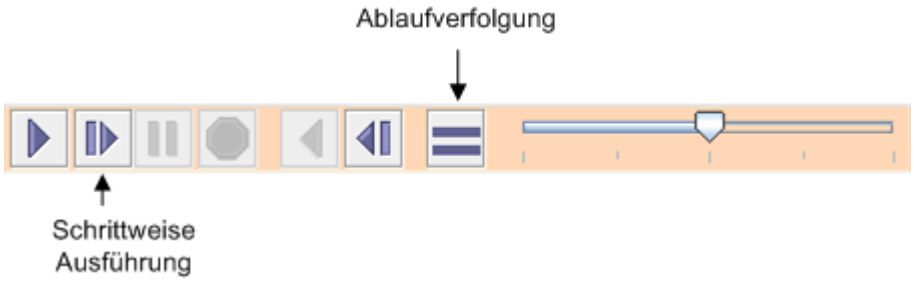


Abb. 6.18: Debugging-Buttons der Toolbar

6.7.1 Beobachten der Programmausführung

Durch Anklicken des Buttons „Ablaufverfolgung“ (19. Button der Toolbar von links) aktivieren bzw. (bei erneuten Anklicken) deaktivieren Sie die Ablaufverfolgung des Debuggers. Bei der Aktivierung öffnet sich dazu das Debugger-Fenster (siehe Abbildung 6.19). Dieses können Sie auch über das Menü „Fenster“ sichtbar bzw. unsichtbar machen.

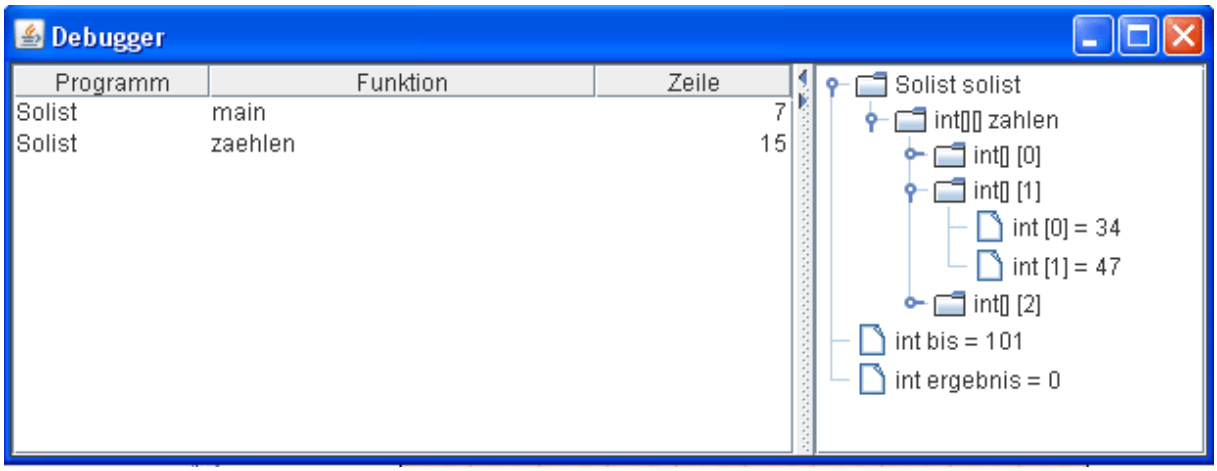


Abb. 6.19: Debugger-Fenster

Ist die Ablaufverfolgung aktiviert, wird bei der Ausführung des Programms im Editor-Bereich der Befehl (bzw. die entsprechende Zeile), der als nächstes ausgeführt wird, blau markiert. Bei einem Prozedur- bzw. Funktionsaufruf wird in die entsprechende Funktion gesprungen. Weiterhin werden im Debugger-Fenster der aktuelle Stack der Funktionsaufrufe (Name der Funktion und aktuelle Position der Ausführung der Funktion) sowie die aktuelle Belegung der Variablen dargestellt.

Im linken Bereich des Debugger-Fensters werden Informationen zu den aktiven Funktionen angezeigt, und zwar jeweils der Name der Funktion und die aktuelle Position der Ausführung der Funktion. Ganz unten erscheint die aktuell aktive Funktion, darüber gegebenenfalls die Funktion, die diese Funktion aufgerufen hat, usw. Ganz oben steht also immer die `main`-Funktion.

Im rechten Bereich des Debugger-Fensters werden die aktiven Variablen und ihre aktuellen Werte angezeigt. Die Darstellung erfolgt dabei in einem Elementbaum, d.h. bei komplexen Variablen, wie Arrays, können Sie durch Anklicken des Symbols vor dem Variablennamen die Komponenten einsehen.

Auch während die Ablaufverfolgung aktiviert ist, können Sie die Programmausführung durch Anklicken des „Pause“-Buttons anhalten und durch anschließendes Anklicken des „Start/Fortsetzen“-Buttons wieder fortfahren lassen. Auch die Geschwindigkeit der Programmausführung lässt sich mit dem Schieberegler anpassen.

6.7.2 Schrittweise Programmausführung

Mit dem Toolbar-Button „Schrittweise Ausführung“ (14. Button der Toolbar von links) ist es möglich, ein Programm schrittweise, d.h. Anweisung für Anweisung

auszuführen. Immer, wenn Sie den Button anklicken, wird die nächste Anweisung (bzw. Zeile) ausgeführt.

Sie können das Programm mit der schrittweisen Ausführung starten. Sie können jedoch auch zunächst das Programm normal starten, dann pausieren und ab der aktuellen Anweisung schrittweise ausführen. Eine normale Programmweiterführung ist jederzeit durch Klicken des „Start“-Buttons möglich.

6.7.3 Breakpoints

Wenn Sie das Programm an einer bestimmten Stelle anhalten möchten, können Sie in der entsprechenden Zeile einen so genannten Breakpoint setzen. Führen Sie dazu vor oder während der Programmausführung im Editor-Bereich mit der Maus einen Doppelklick auf die entsprechende Zeilennummer aus. Breakpoints werden durch violett hinterlegte Zeilennummern dargestellt (siehe Abbildung 6.20). Ein Doppelklick auf einen Breakpoint löscht den Breakpoint wieder. Über der Spalte mit den Zeilennummern lässt sich auch durch Klicken der rechten Maustaste ein Popup-Menü aktivieren, das als Funktionen das Setzen bzw. Entfernen von Breakpoints bzw. das gleichzeitige Löschen aller Breakpoints bietet.

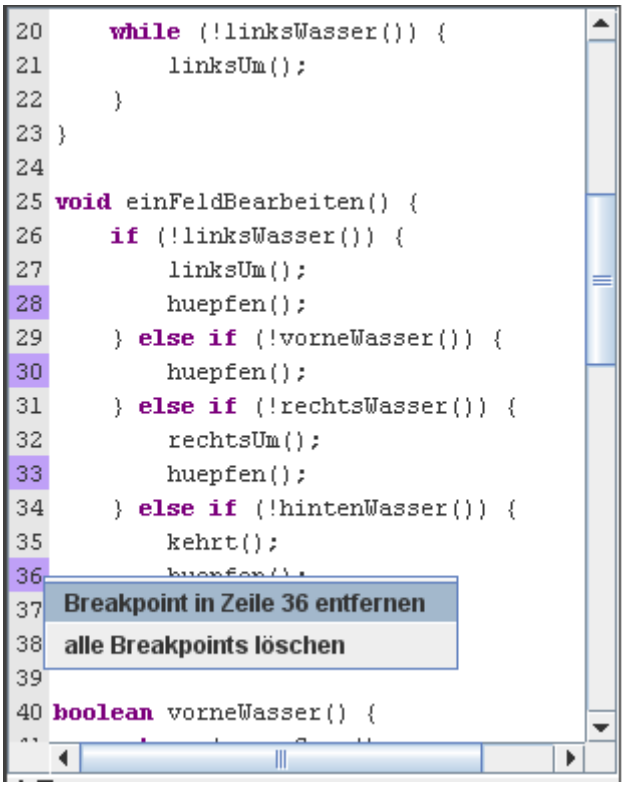


Abb. 6.20: Breakpoints

Wenn Sie ein Programm mit Breakpoints ausführen, wird die Ausführung jedesmal pausiert, wenn eine Zeile mit einem Breakpoint erreicht wird (unabhängig davon, ob die Ablaufverfolgung eingeschaltet ist). Durch Drücken des „Start/Fortsetzen“-Buttons kann die Ausführung des Programms fortgesetzt werden.

6.7.4 Debugger-Fenster

Das Debugger-Fenster wird automatisch bei Aktivierung der Ablaufverfolgung sichtbar gemacht. Über das Menü „Fenster“ lässt es sich jedoch auch explizit sichtbar bzw. unsichtbar machen. Das Fenster besteht aus einem rechten und einem linken Teil innerhalb einer Split-Pane. Der Inhalt des Debugger-Fensters wird nur dann aktualisiert, wenn die Ablaufverfolgung aktiviert ist.

Im linken Bereich des Fensters werden die aktuell aufgerufenen Funktionen und die jeweilige Zeilennummer dargestellt, in der sich die Programmausführung innerhalb der Funktion gerade befindet. Ganz oben steht dabei immer die main-Funktion, ganz unten die zuletzt aufgerufene Funktion.

Im rechten Bereich werden Variablen und deren aktuelle Werte dargestellt. Im Normalfall sind das genau die Variablen, die in der zuletzt aufgerufenen Funktion (also der im linken Bereich ganz unten stehenden Funktion) gültig sind. Die Darstellung erfolgt dabei in einem Elementbaum. Bei strukturierten Variablen kann durch Mausklick auf das Symbol vor dem Variablennamen die Struktur weiter geöffnet (bzw. wieder geschlossen) werden.

Im pausierten Zustand ist es möglich, sich auch die Werte lokaler Variablen anderer Funktionsinkarnationen anzuschauen. Das ist möglich, indem man im linken Bereich auf den entsprechenden Funktionsnamen klickt.

6.8 Konsole

Die Konsole ist ein zusätzliches Fenster, das bei bestimmten Aktionen automatisch geöffnet wird, sich aber über das Menü „Fenster“ auch explizit öffnen bzw. schließen lässt (siehe Abbildung 6.20).

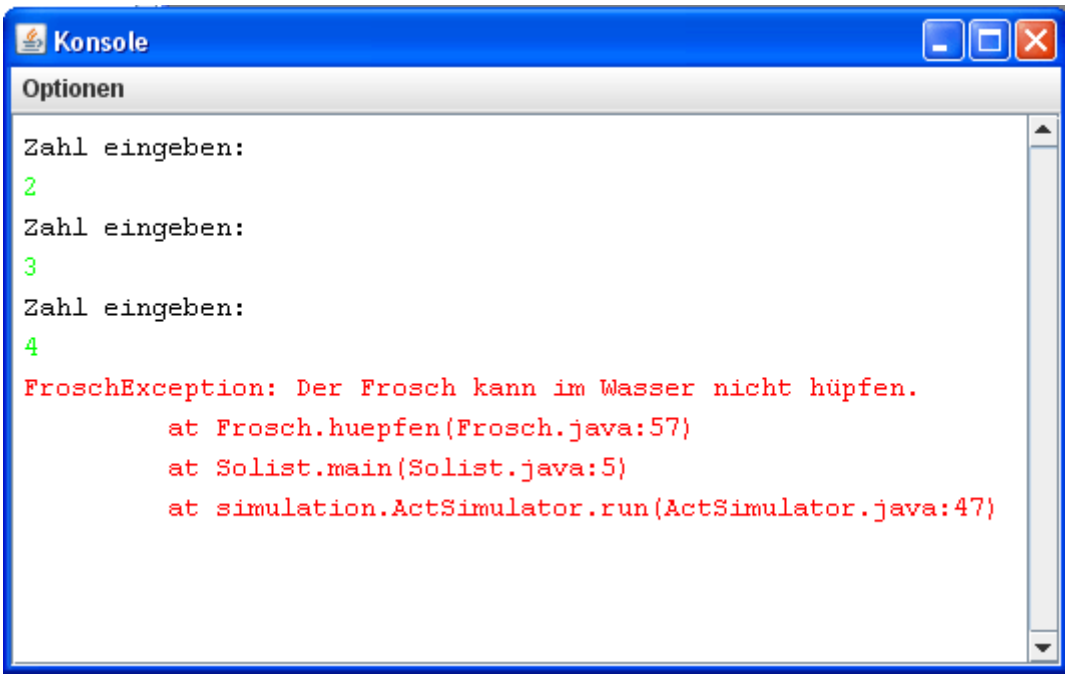


Abb. 6.20: Konsole

Die Konsole ist für die Java-Eingabe mittels `System.in` und die Ausgabe mittels `System.out` und `System.err` zuständig. Wird in Ihrem Programm bspw. der Befehl `System.out.println(„hallo“);` ausgeführt, wird die Zeichenkette `hallo` in der Konsole ausgegeben. `System.out` und `System.err` unterscheiden sich durch eine schwarze (out) bzw. eine rote (err) Ausgabe des entsprechenden Textes. Auch Fehlermeldungen des Frosch-Simulators erscheinen in der Konsole (bspw. Laufzeitfehler, wenn der Frosch versucht auf einer Graskachel zu schwimmen).

Ein Eingabebefehl via `System.in` blockiert das Frosch-Programm so lange, bis der Nutzer eine entsprechende Eingabe in der Konsole getätigt und im Allgemeinen durch Drücken der Enter-Taste abgeschlossen hat. Die folgende Funktion `readInt` erwartet bspw. vom Nutzer die Eingabe einer Zahl in der Konsole und liefert den entsprechenden Wert an das Frosch-Programm. Wenn der Nutzer in der Konsole den Wert 4 eingibt, hüpft der Frosch vier Kacheln nach vorne (insofern sich dort keine Wasserkacheln befinden).

```
void main() {
    System.out.println("Zahl eingeben: ");
    int zahl = readInt();
    while (zahl > 0) {
        huepfen();
        zahl = zahl - 1;
    }
}
```

```
// Einlesen eines int-Wertes
int readInt() {
    try {
        java.io.BufferedReader input =
            new java.io.BufferedReader(
                new java.io.InputStreamReader(System.in));
            String eingabe = input.readLine();
       return new Integer(eingabe);
      } catch (Throwable exc) {
       return 0;
      }
}
```

Die Konsole enthält in der Menüleiste ein Menü namens „Optionen“. Hierin finden sich drei Menü-Items, über die es möglich ist, den aktuellen Inhalt der Konsole zu löschen, den aktuellen Inhalt der Konsole in einer Datei abzuspeichern bzw. die Konsole zu schließen.

7 Beispielprogramme und Aufgaben

Dem Frosch-Simulator sind zwei Beispielprogramme beigefügt, die Sie über das Menü „Programm“ laden können (Menü-Item „Laden…“). Diese werden in den beiden folgenden Abschnitten erläutert. Der dritte Abschnitt enthält einige Anregungen, sich selbstständig Aufgaben zu überlegen, die der Frosch erledigen könnte.

7.1 Mücke suchen

Die Aufgabe lautet: Der Frosch soll sich solange nach vorne bewegen, bis er auf eine Mücke trifft. Das Territorium kann beliebig gestaltet sein.

Dateiname: einereihe.frosch

Beispiel-Territorien: einereihe1.ter und einereihe2.ter

```
void main() {
    while (!mueckeDa()) {
        bisAnsWasserHuepfen();
        if (!mueckeDa()) {
            bisAnsUferSchwimmen();
        }
    }
}

void bisAnsWasserHuepfen() {
    while (imGras() && !mueckeDa()) {
        huepfen();
    }
}

void bisAnsUferSchwimmen() {
    while (imWasser() && !mueckeDa()) {
        schwimmen();
    }
}

boolean imWasser() {
    return !imGras();
}
```

7.2 Teich umrunden

Die Aufgabe lautet: Der Frosch sitzt an einem Teich. Er soll den Teich umrunden, bis er auf eine Muecke stoesst.

Dateiname: teichumrunden.frosch

Beispiel-Territorien: teich1.ter und teich2.ter

```
void main() {
    richtigPositionieren();
    while (!mueckeDa()) {
        einFeldBearbeiten();
    }
}

void richtigPositionieren() {
    while (!linksWasser()) {
        linksUm();
    }
}

void einFeldBearbeiten() {
    if (!linksWasser()) {
        linksUm();
        huepfen();
    } else if (!vorneWasser()) {
        huepfen();
    } else if (!rechtsWasser()) {
        rechtsUm();
        huepfen();
    } else if (!hintenWasser()) {
        kehrt();
        huepfen();
    }
}

boolean vorneWasser() {
    return !vorneGras();
}

boolean linksWasser() {
    linksUm();
    if (vorneWasser()) {
        rechtsUm();
        return true;
    } else {
        rechtsUm();
        return false;
    }
}

boolean rechtsWasser() {
    rechtsUm();
    if (vorneWasser()) {
        linksUm();
        return true;
    } else {
        linksUm();
        return false;
    }
}

boolean hintenWasser() {
    kehrt();
    if (vorneWasser()) {
        kehrt();
        return true;
```

```
    } else {
        kehrt();
        return false;
    }
}

void kehrt() {
    linksUm();
    linksUm();
}
```

7.3 Teich umrunden (objektorientierte Variante)

Der Frosch hat übrigens auch einen Namen, nämlich `kermit`. Darüber ist es möglich, Befehle an den Frosch auch in einer objektorientierten Notation zu verwenden:

```
kermit.huepfen();
kermit.schwimmen();
kermit.linksUm();
kermit.rechtsUm();
if (kermit.imGras()) ...
while (kermit.vorneGras()) ...
while (!kermit.mueckeDa()) ...
```

Die objektorientierte Notation kann auch bei selbst definierten Prozeduren bzw. Funktionen verwendet werden:

```
void kehrt() {
    kermit.links();
    kermit.linksUm();
}

void main() {
    kermit.linksUm();
    kermit.kehrt();
}
```

Lösen wir die zweite Beispielaufgabe nun einmal in dieser objektorientierten Variante. Aufgabe: Der Frosch sitzt an einem Teich. Er soll den Teich umrunden, bis er auf eine Muecke stoesst.

Dateiname: OOteichumrunden.frosch

Beispiel-Territorien: teich1.ter und teich2.ter

```
void main() {
    willi.laufeZumBerg();
    willi.erklimmeGipfel();
}
```

```
void laufeZumBerg() {
    while (willi.vornFrei()) {
        willi.vor();
    }
}

void erklimmeGipfel() {
    do {
        willi.erklimmeEineStufe();
    } while (!willi.vornFrei());
}

void erklimmeEineStufe() {
    willi.linksUm();
    willi.vor();
    willi.rechtsUm();
    willi.vor();
}

void rechtsUm() {
    willi.kehrt();
    willi.linksUm();
}

void kehrt() {
    willi.linksUm();
    willi.linksUm();
}
```

7.4 Frosch-Aufgaben

Am besten ist, sie überlegen sich einfach selber mal Aufgaben, die Sie den Frosch lösen lassen wollen. Oder Sie orientieren sich an Aufgaben aus dem Buch „Programmieren spielend gelernt mit dem Java-Hamster-Modell“ (siehe Kapitel 8). Vielleicht helfen Ihnen folgende Anregungen ein wenig weiter:

- Der Frosch soll in einem Teich eine Mücke finden.
- Der Frosch soll eine Mücke finden, darf aber keine Wasserkacheln betreten.
- Der Frosch soll zwischen zwei Wasserkacheln hin und her laufen.
- Der Frosch soll im Slalom um vorhandene Wasserkacheln hüpfen.
- Der Frosch soll in einem Wasser-Labyrinth eine Mücke finden.
- Der Frosch soll Mücken zählen.

8 Literatur zum Erlernen der Programmierung

Der Frosch-Simulator ist ein Werkzeug, das Programmierern beim praktischen Erlernen der (imperativen) Programmierung hilft. Er ist kein Lehrbuch. Um mit dem Frosch-Simulator programmieren zu lernen, sollten Sie sich ein begleitendes Lehrbuch anschaffen.

Beim Java-Hamster-Modell handelt es sich um ein ähnliches Modell wie beim hier vorgestellten Java-Frosch-Modell. Zu dem Hamster-Modell gibt es ein Lehrbuch: „**Programmieren spielend gelernt mit dem Java-Hamster-Modell**“ von Dietrich Boles, erschienen im Vieweg+Teubner-Verlag. In diesem Buch werden die grundlegenden Konzepte der Programmierung anhand des Java-Hamster-Modells vorgestellt. Das Buch geht langsam und schrittweise vor. Es enthält viele Beispiele und auch eine Reihe von Aufgaben. Dieses Buch kann uneingeschränkt auch als Begleitbuch für Frosch-Programmierer empfohlen werden. Das Buch ist dabei insbesondere für solche Programmieranfänger zu empfehlen, die sich beim Erlernen der Programmierung schwer tun. Mehr Informationen und auch Links zu Leseproben gibt es auf der Java-Hamster-Website www.java-hamster-modell.de.

Es gibt heutzutage unzählige Lehrbücher zu Java und es ist schwer zu sagen, für wen welches Buch das geeignetste ist. Viele Bücher stehen bei Amazon oder Google-Books zumindest auszugsweise online zur Verfügung und ich kann nur empfehlen, über diese Online-Angebote selbst einmal in die Bücher hineinzuschnuppern. Neben dem Java-Hamster-Buch kann ich die beiden weiteren Bücher insbesondere für Programmieranfänger empfehlen:

- Ratz, D., Scheffler, J., Seese, D. und Wiesenberger, J.: „Grundkurs Programmieren in Java, Band 1“, Hanser-Verlag.
- Heinisch, C., Müller, F. und Goll, J..: „Java als erste Programmiersprache“, Vieweg+Teubner.

Wenn Sie noch überhaupt keine Kenntnisse der Programmierung haben, empfehle ich Ihnen, sich Informationen zu den allgemeinen Grundlagen der Programmierung auf der folgenden Website durchzulesen (Leseprobe des Java-Hamster-Buches): http://www-is.informatik.uni-oldenburg.de/~dibo/hamster/leseprobe/node3.html.