Institut für Informatik
der Universität München
Prof. Dr. M. Hofmann
Dr. R. Matthes, M. Lange

WS 02/03

27. November 2002

Practical 2 zur Vorlesung

Einführung in die Informatik: Programmierung und Software-Technik

Abgabe: spätestens 11.12.2002, 14:00 Uhr

In diesem Practical werden Sie erstmals einen kleinen Algorithmus programmieren. Das Ziel ist, ein Programm zu schreiben, welches ein Labyrinth erzeugt, grafisch darstellt und von einem bestimmten Punkt in dem Labyrinth aus alle erreichbaren Punkte einfärbt.

Zur Bearbeitung brauchen Sie wieder die `GraphicsWindow`–Klasse. Beachten Sie jedoch, dass Sie die Version 2 benutzen müssen. Diese können Sie sich von der WWW-Seite der Vorlesung herunterladen.

Damit das elektronische Abgabesystem automatisch überprüfen kann, ob Ihr Programm fehlerfrei kompiliert, muss Ihre Klasse einen bestimmten Namen haben. Für dieses Practical ist dies

```
Practical2
```

Das bedeutet, dass in dem JAR-Archiv, welches Sie abgeben, eine Datei namens `Practical2.java` vorhanden sein muss. Diese Datei muss dann auch die `main`–Methode zum Ausführen Ihres Programmes enthalten. Im Übrigen ist der Name des Archivs, das Sie abgeben, egal.

Zur Wiederholung: Das JAR–Archiv erzeugen Sie mit

```
jar cvf ArchivName Datei1 Datei2 ...
```

Achten Sie darauf, dass *alle* Dateien außer den Standardbibliotheken, die von Ihrem Programm benutzt werden, in dem `jar`-Archiv vorhanden sind, also alle Klassen, die Sie für dieses Practical schreiben sollen sowie vorgegebene Klassen wie z.B. `GraphicsWindow.java`. Das elektronische Abgabesystem erwartet, dass die Dateien im Archiv in keinen Unterverzeichnissen liegen. Ist dies der Fall, so kann es die Dateien nicht finden, und Sie erhalten eine Fehlermeldung bei der Abgabe.

Beachten Sie auch, dass über `import` nur Java-Bibliotheksklassen eingebunden werden und nicht solche Klassen, die bereits im gleichen Verzeichnis wie `Practical2.java` liegen.

Geben Sie nur über das elektronische Abgabesystem ab. Schicken Sie Ihre JAR–Archive **nicht** per Mail an uns. Schicken Sie überhaupt nichts an uns, denn auch wir können nur JAR-Archive über die Eingabemaske des elektronischen Abgabesystems dort eintragen.

Bei Problemen wenden Sie sich an die studentischen Hilfskräfte, die die Rechnerraumbetreuung durchführen. Diese sind an den folgende Terminen im CIP-Pool Sibirien bzw. Gobi zu finden. Montag 10:00 – 20:00, Dienstag 9:00 – 13:00, 14:30 – 18:30, Mittwoch 9:00 – 13:00, Donnerstag 12:00 – 14:00, Freitag 13:00 – 17:00.

Auf der WWW-Seite der Vorlesung finden Sie das JAR–Archiv `Practical2Demo.jar`. In diesem befinden sich zwei Programme, die auf unterschiedliche Art und Weise die erreichbaren Teile eines Labyrinthes einfärben. Wenn Sie das Archiv heruntergeladen haben, können Sie diese folgendermassen ausführen.

```
java -classpath Practical2Demo.jar Breitensuche     bzw.
java -classpath Practical2Demo.jar Tiefensuche
```

Ziel dieses Practicals ist es, ein Programm zu erstellen, das sich so wie eines der Beispiele verhält.

Im folgenden wird beschrieben, wie Sie Schritt für Schritt zu solch einem Programm gelangen *k*önnen. Wenn Sie einen besseren Weg kennen, dann können Sie auch den beschreiten., Wichtig ist, dass Sie objekt-orientiert programmieren, d.h. Klassen sinnvoll verwenden. Ausserdem dürfen Sie auf keinen Fall den Abschnitt über die Kommentare ignorieren.

1 Aufgabenstellung

1.1 Die neue GraphicsWindow–Klasse

Die Version 2 der Klasse `GraphicsWindow`, wie Sie auf der WWW–Seite der Vorlesung zur Verfügung steht bietet nach aussen hin gegenüber der ersten Version eine wesentliche Änderung. Es steht ein weiterer Zeichenbefehl

```
public void drawBox(Point p, int width, int height, Color color)
```

zur Verfügung, welcher ein Rechteck malt, dessen linke, obere Ecke durch den Punkt `p`, Breite durch `width`, Höhe durch `height` und ausfüllende Farbe durch `color` gegeben ist.

1.2 Eine Klasse für Labyrinthe

Ein Labyrinth muss, wie alles andere auch, im Computer in geeigneter Weise repräsentiert werden. Eine Möglichkeit ist ein zweidimensionales Array, denn ein Labyrinth läßt sich in "Kästchen" zerlegen, die entweder "frei" oder "belegt" sind. Wählen Sie einen geeigneten Typ für die Array–Einträge, der mindestens drei verschiedene Werte hat, also z.B. `int`. Den dritten Wert ("besucht") brauchen Sie später, wenn es darum geht, alle erreichbaren "Kästchen" in dem Array zu finden.

Der Konstruktor Ihrer Labyrinth–Klasse soll ein Array der Größe nxn anlegen, wobei Sie als Programmierer das n selbst wählen dürfen. Benutzen Sie dafür einen in Ihrem Hauptprogramm als `final` deklarierten Wert, der an den Konstruktor des Labyrinths übergeben wird. Ihr Labyrinthobjekt kann diesen dann in einer Instanzvariable speichern. So können Sie den Wert später leicht ändern.

Als nächstes muss das Array geeignet gefüllt werden, d.h. es müssen einige Einträge als "belegt" und alle anderen als "frei" markiert werden.

Erzeugen Sie ein neues `Random`–Objekt, und füllen Sie dann alle Einträge des Arrays systematisch nach folgendem Muster. Falls der nächste `double`–Wert, den das `Random`–Objekt zurückliefert, kleiner als 0.36 ist, dann soll das Kästchen als "belegt" markiert werden, ansonsten als "frei". Mit anderen Worten: Ein Kästchen wird mit einer Wahrscheinlichkeit von 36% als belegt markiert.

Sie können Sich noch einigen späteren Ärger sparen, wenn Sie jetzt, wenn das gesamte Labyrinth aufgebaut wurde, alle Kästchen, die am Rand des Labyrinths liegen, auf "belegt" setzen.

Dann brauchen Sie in dieser Klasse noch

- eine Methode, der die Indizes eines Kästchens übergeben werden, und die den momentanen Eintrag, also "belegt", "frei" oder "besucht" zurückliefert.
- eine Methode, die ein bestimmtes Kästchen mit "besucht" markiert.

1.3 Eine Klasse für "Kästchen"

Es bietet sich an, ein Kästchen, d.h. ein Paar von Indizes eines Arrays, als Objekt anzusehen. Dies hilft Ihnen später, diese Kästchen als besucht zu markieren.

Eine Objekt "Kästchen" beinhaltet lediglich zwei Werte vom Typ `int` – die Indizes des Arrays, einen Konstruktor, der diese initialisiert, und zwei Methoden, die die jeweiligen Werte auslesen und zurückliefern.

Beachten Sie, dass Sie keine Umlaute in Bezeichnern verwenden sollten.

1.4 Datenstrukturen zum "Merken" von Objekten

Um im Hauptprogramm das Labyrinth zu "durchlaufen", brauchen Sie eine dynamische Datenstruktur, in der Sie sich Kästchen merken können. Diese stellen wir Ihnen auf der WWW–Seite der Vorlesung zur Verfügung. Dort finden Sie zwei Klassen `FIFO.java` und `LIFO.java`, die Sie sich herunterladen können.

FIFO implementiert das *first-in-first-out*–Prinzip, vergleichbar mit einer Supermarktschlange. Wer sich zuerst anstellt, wird auch zuerst bedient. Eine solche Datenstruktur wird auch *Queue* oder *Schlange* genannt. LIFO dagegen implementiert das *last-in-first-out*–Prinzip, vergleichbar mit einem Stapel Papier. Was zuletzt draufgelegt wurde, wird als erstes wieder heruntergenommen. Eine solche Datenstruktur wird auch *Stack*, *Keller* oder *Stapel* genannt.

Beide Klassen enthalten Konstruktoren

```
FIFO()      bzw.      LIFO()
```

mithilfe derer Sie eine neue, leere Datenstruktur anlegen können. Beide Arten von Datenstrukturen stellen folgende Methoden zur Verfügung:

```
public void in(Object x)
```

Damit können Sie das Object `x` (z.B. ein Kästchen) in die Datenstruktur einfügen.

```
public Object out()
```

liefert Ihnen das "nächste" Objekt aus der Datenstruktur zurück und entfernt es darin. Welches das nächste ist, hängt davon ab, ob Sie `LIFO` oder `FIFO` verwenden. Im Falle, dass die Datenstruktur leer ist, liefert der Aufruf von `out()` `null` zurück.

Beachten Sie, dass `out()` ein Objekt vom Typ `Object` zurückliefert. Das bedeutet, falls Sie in Ihrer Datenstruktur Kästchen speichern, die in einer Klasse `Kaestchen` z.B. definiert sind, dann müssen Sie `out()` wie folgt aufrufen.

```
naechstesKaestchen = (Kaestchen) datenStruktur.out();
```

Dann gibt es noch die Methode

```
public boolean empty()
```

die `true` zurückliefert, falls die Datenstruktur leer ist, und `false` sonst.

1.5 Das Hauptprogramm

Das Hauptprogramm, d.h. die `main`–Methode, muss sich in der Datei `Practical2.java` befinden. Es soll folgendes tun.

Zu Anfang wird ein Grafikfenster und ein neues Labyrinth erzeugt. Dann muss das Labyrinth in das Grafikfenster gemalt werden. Beachten Sie, dass die Indizes eines Kästchens im Labyrinth nicht den geometrischen Koordinaten des gemalten Kästchens im Fenster entsprechen. Jedoch können Sie

letztere aus ersteren gewinnen. Wenn Sie z.B. die Dimension 30x30 benutzt haben, dann ist eine Kästchenbreite und –höhe von 20 Pixeln ein guter Wert. Das bedeutet, dass Sie das Kästchen mit den Indizes i und j an die Stelle $(i * 20, j * 20)$ malen können.

Malen Sie das gesamte Labyrinth, indem Sie jedes Kästchen im Fenster abbilden. Für freie Kästchen schlagen wir die Farbe `Color.white`, für belegte Kästchen die Farbe `Color.black` vor.

Als nächstes soll Ihr Programm darauf warten, dass der Benutzer mit der Maus in ein freies Kästchen klickt. Dies erreichen Sie z.B. dadurch, dass Sie solange auf Mausklicks warten, bis deren Koordinaten ein freies Kästchen kodieren. Dazu müssen Sie die Mausklickkoordinaten natürlich entsprechend in Indizes für Ihr Labyrinth umrechnen.

Hat ein Mausklick dann einmal ein freies Kästchen getroffen, dann sollen alle von dort aus erreichbaren Kästchen mit einer anderen Farbe (z.B. `Color.red`) eingefärbt werden. Dies läßt sich folgendermaßen Schritt für Schritt bewerkstelligen. Zuerst einmal brauchen Sie eine der in Abschnitt 1.4 vorgestellten Datenstrukturen.

Markieren Sie das Kästchen, in das geklickt wurde, als "besucht" und färben Sie es mit der entsprechenden Farbe neu. Tragen Sie dann das Kästchen in Ihrer Datenstruktur ein.

Solange sich in der Datenstruktur noch Elemente befinden, müssen Sie folgendes machen. Nehmen Sie das nächste Element aus der Datenstruktur her. Dies ist ein bereits besuchtes und gefärbtes Kästchen, denn schließlich haben Sie bisher nur besuchte und gefärbte Kästchen in die Datenstruktur eingetragen. Betrachten Sie alle vier Nachbarn dieses Kästchens, d.h. das darüber–, darunter–, rechts davon und links davon liegende. Für jedes dieser Kästchen müssen Sie herausfinden, ob es noch frei ist. Falls dies der Fall ist, so müssen Sie es als besucht markieren, einfärben und in die Datenstruktur eintragen. Falls es belegt oder bereits besucht ist, so brauchen Sie damit nichts zu machen. Auf diese Art und Weise werden alle in dem Labyrinth vom Mausklick aus erreichbaren Kästchen eingefärbt.

Falls sich beim Ausführen das Einfärben immer weiter verlangsamt, bis es anscheinend zum Stillstand kommt, dann achten Sie darauf, dass Sie nur freie Kästchen als besucht markieren und dann in die Datenstruktur eintragen.

Falls Ihr Programm zu schnell abläuft, so dass alle erreichbaren Kästchen wie auf einen Schlag eingefärbt werden, so können Sie Ihr Programm künstlich verlangsamen. Auf der WWW-Seite der Vorlesung finden Sie eine Klasse `TimeWaste`, die nur eine Methode enthält. Diese rufen Sie mit

```
TimeWaste.waste(x);
```

auf, was dazu führt, dass x Millisekunden lang nichts getan wird. Dies können Sie z.B. jedes Mal dann machen, wenn Sie gerade wieder ein neues Kästchen eingefärbt haben. Sorgen Sie jedoch dafür, dass Ihr Programm nicht mehr als 15 Sekunden für die gesamte Ausführung braucht.

Wenn das Programm funktioniert, dann benutzen Sie einmal die andere Datenstruktur und vergleichen Sie die Ausführungen. Variieren Sie auch die Arraygröße. Geben Sie letztendlich das Programm ab, das die Datenstruktur benutzt, die Ihnen besser gefällt. Die Größe des Labyrinthes, welches von Ihrem abgegebenen Programm erzeugt wird, sollte 15x15 nicht unterschreiten.

1.6 Kommentare

Versehen Sie Ihr Programm und Ihre selbstgeschriebenen Klassen mit Kommentaren für Konstruktoren, Instanzvariablen und Methoden, so dass eine Dokumentation Ihrer Abgabe mit `javadoc` erzeugt werden kann. Zur Wiederholung: Ein JavaDoc–Kommentar beginnt mit `/**` und endet mit `*/` und steht vor dem dazugehörigen Block.