Prof. Dr. A. Weber, Dr. G. Kniesel
J. Tautges, P. Bihler
oose.cs.uni-bonn.de

Universität Bonn
Institut für Informatik II
Institut für Informatik III
03.06.2008

Sommersemester 2008

Übungen zur Vorlesung
Objektorientierte Softwareentwicklung (BA-INF-024)
Aufgabenblatt 8

Zu bearbeiten bis: 13.06.2008

Aufgabe 1 *(Null-Objekt, Ausnahmebehandlung, Nebenläufige Programmfäden - 2 Punkte)*

Überlegen Sie sich für jede Aussage ein Argument, mit dem Sie eindeutig bestimmen können, ob die Aussage wahr oder falsch ist. Für jede richtig angekreuzte Antwort werden 0,5 Punkte angerechnet, für jede falsch angekreuzte Antwort 0,5 Punkte abgezogen. Nicht angekreuzte Antworten geben keinen Punktabzug.

wahr	falsch	
		Jeder Java-Variable kann man das spezielle „Objekt“ `null` zuweisen. Damit wird eine vorher vorhandene Referenz auf ein Objekt im Speicher gelöscht.
		Einem einzelnen `try`-Block können mehrere `catch`-Blöcke folgen.
		Einem `try-catch`-Block muss immer ein `finally`-Block folgen.
		Programmcode, der in Java nebenläufig ausgeführt werden soll, muss in Unterklassen der Klasse `Thread` implementiert werden.

Aufgabe 2 *(Begriffe - 5 Punkte)*

a) Erklären Sie den Unterschied zwischen *Klassenvariablen* und *Instanzvariablen*!

b) Erklären Sie den Unterschied zwischen *Klassenfunktionen* und *Instanzfunktionen*! Wann sollte man was verwenden?

Aufgabe 3 *(Threads - 6 Punkte)*

Schreiben Sie eine Klasse `DateiBeobachter` mit Konstruktoren `DateiBeobachter(String)` und `DateiBeobachter(java.io.File)`, die das Interface `Runnable` implementiert. Die Klasse soll beobachten, ob sich eine Datei im Dateisystem ändert, und gegebenenfalls eine entsprechende Meldung ausgeben.[1] Lassen Sie sich während der Programmausführung im Abstand von einer halben Sekunde den Namen der Datei anzeigen. Legen Sie zum Testen eine Textdatei an und modifizieren Sie diese schließlich zur Laufzeit des Threads!

[1] Benutzen Sie die Methode `java.io.File.lastModified()` um auf die Zeit der letzten Modifikation zuzugreifen.

Aufgabe 4 *(Damenproblem - 12 Punkte)*

Schon im Jahre 1850 wurde u. a. von C. F. GAUSS die folgende Aufgabe betrachtet:

Man finde eine Stellung für acht Damen auf einem Schachbrett, so dass keine zwei Damen sich gegenseitig schlagen können.

Eine Dame kann im Schachspiel beliebig viele Spielfelder horizontal, vertikal oder diagonal laufen. Die Damen sind also so zu platzieren, dass jede Zeile, jede Spalte und jede Diagonale des Schachbretts höchstens eine Dame enthält.

In dieser Aufgabe soll nun der Rahmen für eine mögliche Lösung des n–Damenproblems geschaffen werden, d. h. auf einem $n \times n$ Felder großen Schachbrett sollen n Damen so platziert werden, dass sie sich nicht gegenseitig schlagen können. Vorgegeben seien die folgenden Teile der JAVA–Klasse `Damenproblem`:

```
public class Damenproblem {
    private int n;
    private int[][] feld;

    public Damenproblem (int d) {
       ...
    }
    public void ausgabe() {
       ...
    }
    public boolean korrektPlatziert() {
       ...
    }
    static public void main (String[] arg) {
        Damenproblem dOk = new Damenproblem (4);
        dOk.feld[1][0] = 1;
        dOk.feld[3][1] = 1;
        dOk.feld[0][2] = 1;
        dOk.feld[2][3] = 1;
        if (dOk.korrektPlatziert()) {
            System.out.println ("dOk ist eine moegliche Loesung!:");
        }
        else {
            System.out.println ("dOk ist keine erlaubte Loesung:");
        }
        dOk.ausgabe();
    }
}
```

Das 2–dimensionale Array `feld` speichert eine mögliche Platzierung der Damen. Der Wert 0 soll bedeuten, dass das Feld leer ist, der Wert 1 soll bedeuten, dass das Feld mit einer Dame besetzt ist.

a) Implementieren Sie den Konstruktor, der für eine übergebene Größe d das Spielfeld initialisiert (d. h. jedes Feld mit dem Wert 0 belegt und die Dimension in der Variablen `n` speichert).

b) Implementieren Sie die Methode `ausgabe`, die mit Hilfe der Befehle `System.out.print` und `System.out.println` das Spielfeld auf der Konsole ausgibt. Die in `main` aufgebaute Stellung soll z. B. wie folgt ausgegeben werden:

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

c) Implementieren Sie die Methode `korrektPlatziert`, die `true` zurückliefert, wenn auf dem $(n \times n)$–Spielfeld n Damen so platziert sind, dass sie sich nicht gegenseitig schlagen können, sonst `false`.

d) Schreiben Sie eine Methode `platziere`, welche auf dem $n \times n$–Felder grossen Spielfeld n Damen so aufstellt, dass sie sich nicht gegenseitig schlagen können, und die alle möglichen Lösungen mit Hilfe der Ausgabemethode auf der Konsole ausgibt.

Prinzipiell können Sie die Methode `platziere` beliebig implementieren. Hier aber noch ein paar Tipps, wie eine solche Lösung aussehen könnte:

- Die übliche Lösung benutzt die Signatur `void platziere(int i)`. Mit dieser Methode wird in der i–ten Spalte des Spielfelds eine Dame platziert. Anschließend wird *rekursiv* `platziere(i+1)` aufgerufen.
- Innerhalb einer Spalte stehen n Positionen (bzw. Zeilen) für die Platzierung einer Dame zur Verfügung. Trivialerweise kann man hier stets in der ersten Zeile anfangen und dann der Reihe nach die anderen Positionen besetzen.
- Die geschickte Kombination der beiden Ansätze führt zu einer Lösung des Problems.
- Geben Sie während der Programmentwicklung nach jeder Platzierung einer Dame das Spielfeld aus, damit Sie die Funktionsweise Ihres Algorithmus am Bildschirm überprüfen können.
- Wann kann der Algorithmus testen, ob die Damen auf dem Spielfeld korrekt platziert sind?
- Wann soll der Algorithmus eine Lösung ausgeben?