

Übung 06: Objektorientierter Entwurf

Abgabetermin: 8. 5. 2008, 8:15

Name: _____

Matrikelnummer: _____

Informatik: G1 (Prähofer) G2 (Prähofer) G3 (Wimmer) G4 (Wimmer)

WIN: G1 (Ibrahim) G2 (Ibrahim) G3 (Schwinger)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Pkte
Übung 6	24	Prosabeschreibung, Java-Programm, Ausgabe eines Testlaufs	Java-Programm	<input type="checkbox"/>	

Sudoku

Entwerfen und Implementieren Sie das Logikrätsel „Sudoku“. Ziel ist, ein 9x9-Gitter mit den Ziffern 1 bis 9 so zu füllen, dass jede Ziffer in einer Spalte, in einer Reihe und in einem Block (3x3-Unterquadrat) nur einmal vorkommt. Ausgangspunkt ist ein Gitter, in dem bereits mehrere Ziffern vorgegeben sind [<http://de.wikipedia.org/wiki/Sudoku>].

Modellieren Sie das Spielfeld: Unterscheiden Sie zwischen Zellen die beim Start fix vorgegeben wurden und nicht mehr geändert werden können, Zellen die noch leer sind, und Zellen die vom Spieler gesetzt wurden und daher noch geändert werden können. Finden Sie einen Algorithmus der erkennt, ob mit den gegenwärtig gesetzten Zellen das Sudoku noch gelöst werden kann oder nicht.

Es gibt verschiedene Arten von Spielern: Eine Person gibt den nächsten Zug auf der Konsole ein, während der Computer automatisch seinen nächsten Zug durchführt. Dabei kann der Computer nach unterschiedlichen Regeln setzen. Implementieren Sie zumindest einen Computer-Spieler, der das Sudoku durch probieren löst. Bei 9x9-Sudokus ist dieser Ansatz effizient genug (Rechenzeit < 1 Sekunde für praktisch alle lösbaren Vorgaben). Den genauen Algorithmus dafür finden Sie unten.

Die Ein- und Ausgabe soll textuell auf der Konsole erfolgen. Am Anfang des Spieles werden die fix vorgegeben Zellen eingegeben. Am einfachsten ist es, wenn Sie eine Folge von 81 Ziffern mit `In.readChar()` einlesen, wobei '0' oder '-' für uninitialisierte Zellen steht. Danach soll der Spieler-Typ ausgewählt werden. Der Computer-Spieler arbeitet dann automatisch, während beim menschlichen Spieler solange neue Werte abgefragt werden, bis das Sudoku korrekt gelöst ist. Lesen Sie Spaltennummer, Zeilennummer und neuen Wert der Zelle einzeln mit `In.readInt()` ein.

Hinweise:

- Das Hauptaugenmerk dieser Übung ist, eine gute Klassenhierarchie und Methodenschnittstelle zu erstellen. Sie müssen ihre Struktur und den Weg, wie sie dazu gekommen sind, in Ihrer Abgabe argumentieren.
- Achten Sie auf die Erweiterbarkeit Ihrer Lösung. Es soll z.B. einfach möglich sein, neue Arten von Spielern und neue Benutzeroberflächen (z.B. eine graphische Oberfläche) hinzuzufügen.

- Für die Speicherung der Daten, z.B. des Spielfeldes, steht Ihnen die Datenstruktur frei. Begründen Sie, warum Sie Ihre Lösung gewählt haben.

Algorithmus für den Computer-Spieler: Der Algorithmus arbeitet rekursiv nach dem Backtracking-Verfahren. Im folgenden Pseudocode verwaltet die Klasse `Feld` den Status der 81 Zellen. Gültige Zellen-Nummern sind also im Bereich von 0 bis 80. Fix vorgegebene Zellen werden übersprungen. Das Spielfeld kann 3 Zustände haben: *korrekt* (alle Zellen richtig ausgefüllt), *unvollständig* (noch nicht vollständig ausgefüllt, aber noch lösbar), und *ungültig* (nicht mehr lösbar, egal ob vollständig ausgefüllt oder nicht)

```
void SPIELLÖSEN(Feld feld)  
    ZELLESETZEN(feld, 0)
```

```
boolean ZELLESETZEN(Feld feld, int zellenNummer) {  
    if zellenNummer >= 81 then  
        return true  
    if feld.zelleFixiert(zellenNummer) then  
        return ZELLESETZEN(feld, zellenNummer + 1)  
    for int wert = 1 to 9 do  
        feld.zelleSetzen(zellenNummer, wert)  
        if feld nicht ungültig then  
            if ZELLESETZEN(feld, zellenNummer + 1) then  
                return true  
    feld.zelleLöschen(zellenNummer)  
    return false
```

Ausschnitt aus einem Spielablauf mit dem menschlichen Spieler: Ausgangspunkt ist ein Sudoku, bei dem nur mehr 3 Felder unbelegt sind:

Please enter the fixed cell values (use - or 0 for empty cells):

534678912
 ---195348
 198342567
 859761423
 426853791
 713924856
 961537284
 287419635
 345286179

Player (human, computer): h

Game state: Incomplete

	1	2	3	4	5	6	7	8	9
1	!5	!3	!4	!6	!7	!8	!9	!1	!2
2	-	-	-	!1	!9	!5	!3	!4	!8
3	!1	!9	!8	!3	!4	!2	!5	!6	!7
4	!8	!5	!9	!7	!6	!1	!4	!2	!3
5	!4	!2	!6	!8	!5	!3	!7	!9	!1
6	!7	!1	!3	!9	!2	!4	!8	!5	!6
7	!9	!6	!1	!5	!3	!7	!2	!8	!4
8	!2	!8	!7	!4	!1	!9	!6	!3	!5
9	!3	!4	!5	!2	!8	!6	!1	!7	!9

Column: 1
 Row: 2
 Value: 6

Game state: Incomplete

	1	2	3	4	5	6	7	8	9
1	!5	!3	!4	!6	!7	!8	!9	!1	!2
2	6	-	-	!1	!9	!5	!3	!4	!8
3	!1	!9	!8	!3	!4	!2	!5	!6	!7
4	!8	!5	!9	!7	!6	!1	!4	!2	!3
5	!4	!2	!6	!8	!5	!3	!7	!9	!1
6	!7	!1	!3	!9	!2	!4	!8	!5	!6
7	!9	!6	!1	!5	!3	!7	!2	!8	!4
8	!2	!8	!7	!4	!1	!9	!6	!3	!5
9	!3	!4	!5	!2	!8	!6	!1	!7	!9

Column: 1
 Row: 1
 Cell 1, 1 has fixed value

Game state: Incomplete

	1	2	3	4	5	6	7	8	9
1	!5	!3	!4	!6	!7	!8	!9	!1	!2
2	6	-	-	!1	!9	!5	!3	!4	!8
3	!1	!9	!8	!3	!4	!2	!5	!6	!7
4	!8	!5	!9	!7	!6	!1	!4	!2	!3
5	!4	!2	!6	!8	!5	!3	!7	!9	!1
6	!7	!1	!3	!9	!2	!4	!8	!5	!6
7	!9	!6	!1	!5	!3	!7	!2	!8	!4
8	!2	!8	!7	!4	!1	!9	!6	!3	!5
9	!3	!4	!5	!2	!8	!6	!1	!7	!9

Column: 2
 Row: 2
 Value: 6

Game state: Incorrect

	1	2	3	4	5	6	7	8	9
1	!5	!3	!4	!6	!7	!8	!9	!1	!2
2	6	6	-	!1	!9	!5	!3	!4	!8
3	!1	!9	!8	!3	!4	!2	!5	!6	!7
4	!8	!5	!9	!7	!6	!1	!4	!2	!3
5	!4	!2	!6	!8	!5	!3	!7	!9	!1
6	!7	!1	!3	!9	!2	!4	!8	!5	!6
7	!9	!6	!1	!5	!3	!7	!2	!8	!4
8	!2	!8	!7	!4	!1	!9	!6	!3	!5
9	!3	!4	!5	!2	!8	!6	!1	!7	!9

Column: 2
 Row: 2
 Value: 7

Game state: Incomplete

	1	2	3	4	5	6	7	8	9
1	!5	!3	!4	!6	!7	!8	!9	!1	!2
2	6	7	-	!1	!9	!5	!3	!4	!8
3	!1	!9	!8	!3	!4	!2	!5	!6	!7
4	!8	!5	!9	!7	!6	!1	!4	!2	!3
5	!4	!2	!6	!8	!5	!3	!7	!9	!1
6	!7	!1	!3	!9	!2	!4	!8	!5	!6
7	!9	!6	!1	!5	!3	!7	!2	!8	!4
8	!2	!8	!7	!4	!1	!9	!6	!3	!5
9	!3	!4	!5	!2	!8	!6	!1	!7	!9

Column: 3
 Row: 2
 Value: 2

Game state: Finished

	1	2	3	4	5	6	7	8	9
1	!5	!3	!4	!6	!7	!8	!9	!1	!2
2	6	7	2	!1	!9	!5	!3	!4	!8
3	!1	!9	!8	!3	!4	!2	!5	!6	!7
4	!8	!5	!9	!7	!6	!1	!4	!2	!3
5	!4	!2	!6	!8	!5	!3	!7	!9	!1
6	!7	!1	!3	!9	!2	!4	!8	!5	!6
7	!9	!6	!1	!5	!3	!7	!2	!8	!4
8	!2	!8	!7	!4	!1	!9	!6	!3	!5
9	!3	!4	!5	!2	!8	!6	!1	!7	!9