Grundlagen der Informatik 1
WS 2008/09

Prof. Mühlhäuser, Dr. Rößling, Melanie Hartmann, Daniel Schreiber
http://proffs.tk.informatik.tu-darmstadt.de/gdi1

Übung 8 - Lösungsvorschlag Version: 1.2 8.12.2008

---

1 Mini Quiz

1. ☐ Scheme ist näher an der Funktionsweise eines Rechners orientiert als Java.
2. ☒ '−' kann als unäre Operation verwendet werden.
3. ☒ $a >> b$ verschiebt die Bitrepräsentation von a um b Stellen nach rechts.
4. ☐ $<$ hat eine niedrigere Piorität als &&.

2 Fragen

1. Was ist der Hauptunterschied zwischen Java und Scheme?
2. Erklären Sie, was Operator-Priorität ist und bringen Sie folgende Operatoren in die richtige Reihenfolge: $*$, $=$, &&, $<=$, $++$, $-$.
3. Warum spielte in Scheme die Operator-Priorität keine Rolle?
4. Erklären Sie, was Operator-Assoziativität ist. Welche Assoziativität haben die oben in 2.2 genannten Operatoren?

Lösungsvorschlag:

1. *Java ist eine objektorientierte imperative Sprache und Scheme ist funktional. Scheme löst Aufgaben durch Dekomposition und Komposition, während Java Probleme durch Delegation an andere Objekte löst. Scheme "denkt" in Funktionen und Java in Rechenschritten, daher ist Java näher am Rechner orientiert.*
2. *Die Priorität gibt an, welche Operatoren zuerst angewendet werden. Die richtige Reihenfolge (von hoher zu niedriger Priorität) lautet: $++$, $*$, $-$, $<=$, &&, $=$.*
3. *Durch die Präfixnotation und Klammerung in Scheme war die Priorisierung immer explizit gegeben, z.B. wird $3 + 4 * 2$ in Scheme als* `(+ 3 (* 4 2))` *repräsentiert, wodurch klar ist, dass $*$ zuerst angewendet wird.*
4. *In einem Ausdruck mit mehr als einem Operator gleicher Priorität wird entweder der Operator ganz links (links-assoziativ) oder ganz rechts (rechts-assoziativ) zuerst angewandt. Links assoziativ sind $*$, &&, $<=$, und $-$, rechts-assoziativ sind $=$ und $++$.*

3 Schleifen

Gegeben sei folgende `while` Schleife in Java:

```
1 int x = 2;
2 int number = 10;
3 int count = 1;
4 while (Math.pow(x, count) <= number){
5   count++;
6 }
```

Hinweis: `Math.pow(a,b)` berechnet a^b.

1. Was macht die Funktion? Kommt sie Ihnen bekannt vor?

   Lösungsvorschlag:
   Die Funktion berechnet die Länge der Repräsentation der Zahl `number` *zur Basis* `x`*, z.B. 10 zur Basis 2 ist 1010 und hat somit die Länge 4. Dies wurde bereits in Aufgabe 5.1 berechnet.*

2. Ersetzen Sie die `while` Schleife durch eine `for` Schleife.

   Lösungsvorschlag:

```
1 for (count = 0; Math.pow(x, count) < number; count++);
```

3. Ersetzen Sie die `while` Schleife durch eine `do while` Schleife.

   Lösungsvorschlag:

```
1 count = -1;
2 do {
3   count++;
4 } while (Math.pow(x, count) < number);
```

4 Scheme in Java

1. **RGB Werte** In Übung 3 Aufgabe 3 haben Sie eine Scheme Prozedur geschrieben, die aus einem gegebenen Farbwert in RGB den zugehörigen Farbwert (Hue-Wert) ermittelt. Schreiben Sie nun die entsprechende Methode **float** getHue(**int** red, **int** green, **int** blue) in Java. Zur Erinnerung, die Berechnung erfolgt wie folgt:

$$r = red/255, g = green/255, b = blue/255$$

$$M = max(r, g, b), m = min(r, g, b)$$

$$H = \begin{cases} 0 & \text{wenn } M = m \\ 60(g-b)/(M-m) & \text{wenn } r = M \\ 120 + 60(b-r)/(M-m) & \text{wenn } g = M \\ 240 + 60(r-g)/(M-m) & \text{wenn } b = M \end{cases}$$

Lösungsvorschlag:

```
1  /**
2   * Computes the hue for a given RBG triple
3   *
4   * @param red the color value for red
5   * @param green the color value for green
6   * @param blue the color value for blue
7   *
8   * @return the hue value in degrees
9   */
10 public static float getHue(int red, int green, int blue) {
11   float sum = 255;
12   float redNormalized = red / sum;
13   float greenNormalized = green / sum;
14   float blueNormalized = blue / sum;
15   float max = Math.max(Math.max(redNormalized, greenNormalized),
16                        blueNormalized);
17   float min = Math.min(Math.min(redNormalized, greenNormalized),
18                        blueNormalized);
19
20   float result = 0;
21
22   if (max == min)
23       return 0;
24   else if (redNormalized == max)
25           result = 60 * (greenNormalized - blueNormalized) / (max -
               min);
26   else if (greenNormalized == max)
27           result = 120 + 60 * (blueNormalized - redNormalized) / (
               max - min);
28   else if (blueNormalized == max)
29           result = 240 + 60 * (redNormalized - greenNormalized) / (
               max - min);
30   else
31           return -1;
32   if (result < 0)
33       result += 360;
34   return result;
35 }
```

2. **Skalarprodukt** In Übung 2 Aufgabe 6.1 haben Sie das Skalarprodukt zweier Vektoren berechnet. Schreiben Sie diese Methode in Java (**int** computeScalar(**int**[] a, **int**[] b)). Vektoren sollen dabei als ein Array von Zahlen repräsentiert werden. Zur Erinnerung: Das Skalarprodukt zweier Vektoren ist die Summe der Produkte der Komponenten der Vektoren:
`scalarProduct(new int[]{3, 1, 0}, new int[]{1, 0, 2})` = `3*1 + 1*0 + 0*2`.

Lösungsvorschlag:

```java
/**
 * Computes the scalar product of two arrays
 * Note: does not perform any error handling!
 *
 * @param a the first array (treated as a mathematical vector)
 * @param b the second array (treated as a mathematical vector)
 * @return the scalar product of a and b
 */
public static int computeScalar(int[] a, int[] b) {
  int result = 0;
  for (int i = 0; i < a.length; i++) {
    result += a[i] * b[i];
  }
  return result;
}
```

3. **Bubblesort** In Übung 5 Aufgabe 6 haben Sie Bubblesort implementiert. Zur Erinnerung: Beim Durchlaufen des Arrays (in Scheme: der Liste) vertauscht man zwei benachbarte Elemente genau dann, wenn sie nicht in der richtigen Reihenfolge stehen. Dadurch "bubblet" das größte Element automatisch an die höchste Position des Arrays. Daher verringert sich die Länge der zu sortierenden Arrayteilfolge mit jeder Iteration um eins, womit sichergestellt ist, dass die Prozedur terminiert.

   Wenn Sie am Rechner arbeiten, schreiben Sie zur Kontrolle der Sortierung eine weitere Methode **void** printArray(**int** [] array), die ein Array auf der Konsole ausgibt. Verwenden Sie dazu die Methode System.out.println(String output) und den Operator "+", um zwei Strings zu verketten, z.B. System.out.println("Der Wert der Variable x ist "+x); (x muss dabei kein String sein, es wird dann automatisch die Stringrepräsentation von x gewählt).

   Implementieren Sie nun Bubblesort in Java (**void** bubblesort(**int** [] array)). Verwenden Sie `printArray` wenn Sie am Rechner arbeiten, um sich die Schritte der Sortierung anzeigen zu lassen.

   Lösungsvorschlag:

```java
/**
 * Sorts the given array using bubblesort
 * @param the list to be sorted
 */
public static void bubblesort(int[] array) {
  printArray(array);
  for (int i = array.length; i > 0; i--) {
    for (int j = 1; j < i; j++)
      if (array[j - 1] > array[j]) {
        int temp = array[j - 1];
        array[j - 1] = array[j];
        array[j] = temp;
      }
    printArray(array);
  }
}

/**
```

```
19   * Prints the given array on the console in the form "[1, 2, 3, 6]"
20   *
21   * @param array the array to be printed
22   */
23  public static void printArray(int[] array) {
24    StringBuffer buffer = new StringBuffer(256);
25    buffer.append("[");
26    for (int i = 0; i < array.length - 1; i++)
27      buffer.append(array[i]).append(", ");
28    buffer.append(array[array.length - 1]).append(']');
29    System.out.println(buffer.toString());
30  }
```

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren.

Abgabe: Bis spätestens Fr, 09.01.2009, 16:00

5 Hello Java (0P)

Bevor Sie mit der eigentlichen Hausübung beginnen, sollten Sie sich zuerst mit den Sprachelementen von Java und Ihrer Entwicklungsumgebung vertraut machen. Wir empfehlen Ihnen *Eclipse* als Entwicklungsumgebung. Dies kann mit `eclipse` auf den Poolrechnern aufgerufen werden. Auf der GdI 1 Veranstaltungsseite finden Sie unter *Java-Add-Ons* neben dem Downloadlink von *Java* und *Eclipse* auch einen Link zum 'Java-Tutorial von Sun'.
Wenn Sie Eclipse benutzen möchten, empfehlen wir Ihnen, das *„Create a Hello World application"* Tutorial durchzuarbeiten, das Sie in Eclipse über `Help> Welcome> Tutorials` finden können.
Programme werden in Eclipse über `Run> Run As> Java Application` ausgeführt; eventuelle Ausgaben sehen Sie in der *Console*. Sie können zum Testen den folgenden Programmcode nutzen:

```
1  class HelloJava {
2    public static void main(String[] args) {
3      System.out.println("Hello Java!");
4    }
5  }
```

6 Code Chaos (4P)

Die aus Übungsblatt 3 bekannte Firma "Schematics" hat beschlossen, einen Teil ihrer Softwareentwicklung fortan in Java zu machen. Da die Mitarbeiter aber über keinerlei Erfahrung in Java verfügen, kommt dabei folgender Code zustande (den Sie auch auf der Vorlesungsseite herunterladen können):

```
1  public class Nonsense {
2    static int[] a;
```

```
3
4  public static double getD(){
5    double result = 0.0; for (int i=0; i<a.length; i++)     result+=a[i];
6    return result/a.length;
7  }
8
9  public static int getE(){
10   if (a.length<1) return -1;
11   double b = getD();       double d = Math.absolut(b-a[0]);int c = a[0];
       for (int i=1; i<a.length; i++){
12   if (Math.absolut(b-a[i])< d){d = Math.absolut(b-a[i]);c = a[i]; }
13   }return c;
14 }
15
16 public static void main(String[] args){
17   a = new int[]{2,5,1,4,9,7};
18   getE();
19 }
20 }
```

Damit wir erst mal erahnen können, was hier passiert, sollen einige Codeverschönerungs und -bereinigungsoperationen vorgenommen werden. Wir empfehlen hierzu Eclipse zu verwenden, damit Sie Eclipse genauer kennen lernen (die in Eclipse benötigten Operationen werden im Folgenden angegeben), aber Sie können das auch mit jedem anderen Tool machen.

1. Formatieren Sie den Code, so dass nur noch ein Befehl in einer Zeile steht. In Eclipse können Sie dazu `Source->Format` verwenden.

2. Der Code lässt sich so wie er ist nicht kompilieren, da eine Methode `Math.absolut` verwendet wird, die nicht existiert. Der Entwickler wollte hier wohl den absoluten Wert einer Zahl berechnen. Wie heißt die Methode richtig? Wenn Sie Eclipse verwenden, gehen Sie dazu mit dem Cursor hinter `Math.` und drücken Strg+Leertaste. Eclipse zeigt Ihnen dann an, welche Funktionen in der Klasse `Math` zur Verfügung stehen. Wählen Sie die passende aus.

3. Als nächstes benennen wir einige Variablen um, da `a`, `b`, `c` und `d` nicht besonders aussagekräftige Namen sind. `a` soll `array` heißen, `b` `average`, `c` `bestElement` und `d` `minDistance`. Dies kann mit Eclipse wie folgt gemacht werden: wählen Sie die entsprechende Variable an (Cursor auf ihren Namen setzen), wählen Sie `Refactor->Rename` und geben Sie den neuen Variablennamen ein.

4. Geben Sie nun auch den Methoden (`getF`, `getE`) und der Klasse `Nonsense` aussagekräftigere Namen. Gehen Sie dabei mit Eclipse wie bei der Umbenennung der Variablen vor.

5. Fügen Sie in die Methode, die bis eben `getE` hieß, vor der Rückgabe eine Ausgabe ein, die angibt, was die Methode gerade berechnet hat. Geben Sie dazu einen Satz auf der Konsole aus, in dem `average`, `bestElement` und `minDistance` vorkommen.

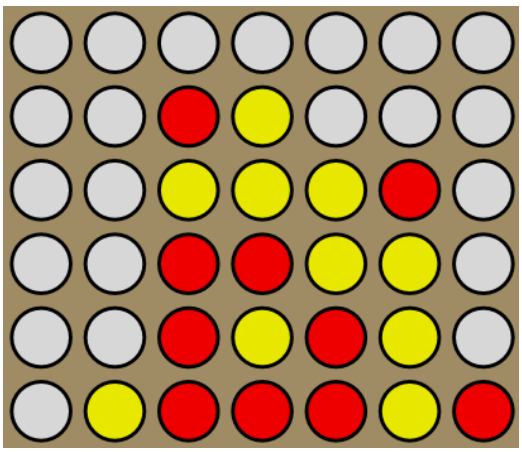
6. Die Firmenpolitik untersagt eigentlich, dass globale Variablen (hier `array`) in der main Methode direkt gesetzt werden dürfen. Schreiben Sie daher eine Methode `getArray`, die `array` zurückgibt und eine Methode `setArray`, die ein Array aus ints entgegennimmt und in der Variablen `array` speichert. In Eclipse können Sie dazu die entsprechende Variable anwählen und über `Source->Generate Getters and Setters...` die zu erzeugenden Methoden auswählen.

Lösungsvorschlag:

```
1  public class SomeCalculation {
2
3     static int[] array;
4
5     public static int[] getArray() {
6        return array;
7     }
8
9     public static void setArray(int[] a) {
10       SomeCalculation.array = a;
11    }
12
13    public static double getAverage() {
14       double result = 0.0;
15       for (int i = 0; i < array.length; i++) {
16          result += array[i];
17       }
18       return result / array.length;
19    }
20
21    public static int getElementClosestToAverage() {
22       if (array.length < 1)
23          return -1;
24       double average = getAverage();
25       double minDistance = Math.abs(average - array[0]);
26       int bestElement = array[0];
27       for (int i = 1; i < array.length; i++) {
28          if (Math.abs(average - array[i]) < minDistance) {
29             minDistance = Math.abs(average - array[i]);
30             bestElement = array[i];
31          }
32       }
33       System.out.println("Best value is " + bestElement
34             + " with a distance of " + minDistance + " to " + average);
35       return bestElement;
36    }
37
38    public static void main(String[] args) {
39       setArray(new int[] { 2, 5, 1, 4, 9, 7 });
40       getElementClosestToAverage();
41    }
42
43 }
```

7 4 gewinnt (9P)

In dieser Hausübung sollen Sie das Spiel “4 gewinnt” implementieren. Die Spielregeln sind wie folgt: Das aufrecht stehende Spielbrett besteht aus sieben Spalten und sechs Reihen (s. Abbildung) in das zwei Spieler abwechselnd ihre Spielsteine fallen lassen. Jeder Spieler besitzt 21 gleichfarbige Spielsteine. Gewinner ist der Spieler, der es als erster schafft, vier seiner Spielsteine waagerecht, senkrecht oder diagonal in eine Linie zu plazieren.



Im GDI-Portal köonnen Sie ein Gerüust für die zu implementierende Klasse `ConnectFour` herunterladen. `ConnectFour` erweitert die aus der Vorlesung bekannte Klasse `DialogProgram`, damit stehen alle Funktionen aus `DialogProgram` (wie z.B. `println` oder `readInt`) in ihrer Klasse zur Verfügung. Dazu müssen Sie die zugehörige Bibliothek (acm.jar) herunterladen und einbinden, wie im Portal unter *Java Add-Ons* beschrieben. Sie können statt `DialogProgram` auch `ConsoleProgram` verwenden, dann hat das Spiel eine Konsolenausgabe. Der Hauptteil der Programms befindet sich in der Methode `run`, die das Spielfeld darstellt und abwechselnd Züge der zwei Spielern einliest.
Das Spielbrett wird als ein Array aus `chars` repräsentiert, das die Breite `width` und Höhe `height` hat. Jeder Spielstein eines Benutzers wird durch einen Character dargestellt (entweder 'x' oder 'o'), der über `getPlayerChar(int playerNo)` abgefragt werden kann. Die Nummer des Spielers ist entweder 1 oder 2.
Hinweis: Vergessen Sie nicht, für jede Methode mit Hilfe von JavaDoc Vertrag (`@param`, `@return`) und Beschreibung anzugeben.

1. Implementieren Sie die Methoden promptPlayerMessage, columnFullMessage, winMessage, drawMessage und errorMessage so, dass sinnvolle Meldungen an den Spieler ausgegeben werden.

2. Implementieren Sie die Methode **void** init (), die das gesamte Spielfeld „leer“ (mit '_') initiiert.

3. Implementieren Sie die Methode **int** putInColumn(**int** colNo, **int** playerNo), die eine Spaltennummer und die Spielernummer entgegennimmt und einen Spielstein des entsprechenden Spielers in die entsprechende Spalte einfügt. `putInColumn` soll die y-Koordinate zurück geben, an die der Spielstein eingefügt wurde. Ist die Spalte bereits voll, soll die Methode -1 zurückgeben. Die `run` Methode ruft die Funktion nach einer Benutzereingabe auf. Gibt die Methode -1 zurück, wird ein Fehler ausgegeben und der aktuelle Spieler wird um eine neue Eingabe gebeten.

   Vorsicht: der Benutzer gibt eine Spaltennummer zwischen 1 und `width` ein. An die Methode wird aber die Spaltennummer - 1 übergeben, da die Nummerierung eines Arrays bei 0 und nicht bei 1 beginnt.

4. Die Methode **boolean** hasWon(**int** x, **int** y) gibt zurück, ob ein neu hinzugefügter Spielstein in der Zeile x auf der Höhe y eine Viererreihe bildet und der aktuelle Spieler somit gewonnen hat. Sie greift auf die Methoden **boolean** hasFourVertically(), **boolean** hasFourHorizontally() und **boolean** hasFourDiagonally() zurück, die als nächstes implementiert werden sollen.

   - Implementieren Sie die Methode **boolean** hasFourVertically(**int** x, **int** y). Diese Methode nehmen die Koordinaten des eingesetzten Spielsteins entgegen (x, y) und geben zurück, ob sich eine vertikale Viererkette gebildet hat.

- Implementieren Sie die Methode **boolean** hasFourHorizontally(**int** x, **int** y). Diese Methode überprüft wie oben auf horizontale Viererketten.
- Implementieren Sie die Methode **boolean** hasFourDiagonally(**int** x, **int** y), die überprüft, ob sich eine diagonale Viererkette durch den neuen Spielstein bei (x, y) gebildet hat.

Lösungsvorschlag:

```
1  import acm.program.ConsoleProgram;
2
3  /**
4   * ConnectFour is an implementation of 4-Gewinnt
5   *
6   * @author Melanie Hartmann, Daniel Schreiber
7   * You may use ConsoleProgram or DialogProgram as a base class,
8   *          whatever you prefer.
9   *
10  */
11 public class ConnectFourSolution extends ConsoleProgram /* DialogProgram
      */{
12    /**
13     *
14     */
15    private static final long serialVersionUID = -2746484328169503540L;
16    // state variables of this object
17    int width = 7;
18    int height = 6;
19    char[][] board = new char[width][height];
20
21    /* output messages */
22    private void promptPlayerMessage(int p) {
23       println("Player " + p + "...:");
24    }
25
26    private void rowFullMessage(int x) {
27       println("Die Reihe " + x + "ist bereits voll!");
28    }
29
30    private void winMessage(int p) {
31       println("Player " + p + " has won!");
32    }
33
34    private void drawMessage() {
35       println("It's a Draw!");
36    }
37
38    private void errorMessage() {
39       println("Fehlerhafte Eingabe, bitte gib eine Zahl zwischen 1 und 7
           ein");
40    }
41
42    /**
43     * Main loop of the program. Let players take turns entering moves
          until one
44     * player wins or the board is full, in which case the game is a draw.
45     */
46    public void run() {
```

```
47      int player = 1;
48      println("----- 4 GEWINNT -----");
49
50      // no winner yet.
51      int winner = 0;
52
53      while (winner == 0 && !isFull()) {
54        // print the current board
55        printBoard();
56        // prompt the current player to enter a move
57        promptPlayerMessage(player);
58        // read the move from the player
59        int x = readInt();
60
61        // terminate when -1 is entered
62        if (x == -1)
63          System.exit(0);
64
65        // players count from 1, Java from 0
66        x--;
67
68        // is input in correct range?
69        if (x >= 0 && x < width) {
70          // valid move, perform it
71          int y = putInColumn(x, player);
72          if (y == -1)
73            // output row is full message
74            rowFullMessage(x);
75          else if (hasWon(x, y))
76            winner = player;
77          else
78            // next player
79            player = 3 - player;
80        } else
81          // invalid move
82          errorMessage();
83
84      }
85      // game has ended, either we have a winner or a draw
86      if (winner == 0)
87        drawMessage();
88      else
89        winMessage(winner);
90    }
91
92    /**
93     * Initializes the Connect Four Game
94     *
95     */
96    public ConnectFourSolution() {
97      init();
98    }
99
100   /**
101    * Initialize the board with blanks
102    *
103    */
104   public void init() {
```

```
105     for (int i = 0; i < height; i++) {
106       for (int j = 0; j < width; j++) {
107         board[j][i] = '_';
108       }
109     }
110   }
111
112   /**
113    * Checks whether another game piece can be added
114    *
115    * @return true if the board is full
116    */
117   public boolean isFull() {
118     for (int x = 0; x < width; x++)
119       if (board[x][0] == '_')
120         return false;
121     return true;
122   }
123
124   /**
125    * Returns the char that represents a player (i.e. x or o)
126    *
127    * @param playerNo
128    *          number of the player, i.e. 1 or 2
129    * @return the char for the player
130    */
131   public char getPlayerChar(int playerNo) {
132     char[] playerChars = { 'x', 'o' };
133     return playerChars[playerNo - 1];
134   }
135
136   /**
137    * Put a gaming piece on top of the gaming pieces in the given column
       by the
138    * given player
139    *
140    * @param column
141    *          columnNo where to enter the gaming piece, number between 0
       and
142    *          width-1
143    * @param playerNo
144    *          number of the player, i.e. 1 or 2
145    * @return the y-coordiante where the gaming piece was added, -1 if the
        gaming
146    *         piece could not be added
147    */
148   public int putInColumn(int column, int playerNo) {
149     for (int i = height - 1; i >= 0; i--) {
150       if (board[column][i] == '_') {
151         board[column][i] = getPlayerChar(playerNo);
152         return i;
153       }
154     }
155     return -1;
156   }
157
158   /**
159    * Print the current board For example: | | | | o x | o x x |o x x o x
```

```
160     * ---------------  1 2 3 4 5 6 7
161     *
162     */
163    public void printBoard() {
164      StringBuffer sb = new StringBuffer();
165      for (int i = 0; i < height; i++) {
166        // start of a line
167        sb.append("|");
168        for (int j = 0; j < width; j++) {
169          sb.append(board[j][i]).append(' ');
170        }
171        // end of a line
172        sb.append('\n');
173      }
174      sb.append(" ---------------\n");
175      sb.append(" 1 2 3 4 5 6 7");
176      println(sb.toString());
177    }
178
179    /**
180     * Test whether there are four gaming pieces with the given char above
           each
181     * other. Only checking column x needed as this is the only column a
           vertical
182     * four-in-a-row could have been created in this turn.
183     *
184     * @param x
185     *          column of the last added piece
186     * @param y
187     *          row of the last added piece
188     * @return whether their are four gaming pieces above each other
189     */
190    public boolean hasFourVertically(int x, int y) {
191      char sign = board[x][y];
192      int count = 0;
193      for (int i = y; i <height; i++) {
194        if (sign == board[x][i]) {
195          count++;
196          if (count == 4)
197            return true;
198        } else
199          count = 0;
200      }
201      return false;
202    }
203
204    /**
205     * Test whether there are four gaming pieces with the given char next
           to each
206     * other. Only checking column x needed as this is the only column a
           horizontal
207     * four-in-a-row could have been created in this turn.
208     *
209     * @param x
210     *          column of the last added piece
211     * @param y
212     *          row of the last added piece
213     * @return whether their are four gaming pieces next to each other
```

```
214     */
215   public boolean hasFourHorizontally(int x, int y) {
216     char sign = board[x][y];
217     int count = 0;
218     for (int i = 0; i < width; i++) {
219       if (sign == board[i][y]) {
220         count++;
221         if (count == 4)
222           return true;
223       } else
224         count = 0;
225     }
226     return false;
227   }
228
229
230   /**
231    * Test whether there are 4 gaming pieces next to each other on a
         diagonal.
232    * Only checking column x needed as this is the only column a
         horizontal
233    * four-in-a-row could have been created in this turn.
234    *
235    * @param x
236    *          column of the last added piece
237    * @param y
238    *          row of the last added piece
239    * @return whether there are 4 gaming pieces next to each other on a
         diagonal
240    */
241   public boolean hasFourDiagonally(int x, int y) {
242     char sign = board[x][y];
243
244     // Check the whole diagonal starting from the bottom (start_x,
         start_y)
245     // going up to the right, i.e. /
246     int delta = Math.min(height - 1 - y, x);
247     int start_x = x - delta;
248     int start_y = y + delta;
249     int y_temp = start_y;
250     int count = 0;
251     for (int x_temp = start_x; x_temp < width && y_temp >= 0; x_temp++) {
252       if (sign == board[x_temp][y_temp]) {
253         count++;
254         if (count == 4)
255           return true;
256       } else {
257         count = 0;
258       }
259       y_temp--;
260     }
261
262     // Check the whole diagonal starting from the bottom (start_x,
         start_y)
263     // going up to the left, i.e. \
264     delta = Math.min(y, x);
265     start_x = x - delta;
266     start_y = y - delta;
```

```
267       y_temp = start_y;
268       count = 0;
269       for (int x_temp = start_x; x_temp < width && y_temp < height; x_temp
            ++) {
270         if (sign == board[x_temp][y_temp]) {
271           count++;
272           if (count == 4)
273             return true;
274         } else {
275           count = 0;
276         }
277         y_temp++;
278       }
279       return false;
280     }
281
282     /**
283      * Checks whether the game is won and who won the game
284      *
285      * @return the number of the winner (i.e. 1 or 2) or 0 if no one has
           yet won
286      *         the game
287      */
288     public boolean hasWon(int x, int y) {
289       return (hasFourHorizontally(x, y) || hasFourVertically(x, y)
290           || hasFourDiagonally(x, y));
291     }
292
293     /**
294      * Main method. This is automatically called by the environment when
           your
295      * program starts.
296      *
297      * @param args
298      */
299     public static void main(String[] args) {
300       new ConnectFourSolution().start();
301     }
302
303 }
```