Turun yliopisto
University of Turku

Olio-ohjelmoinnin metodiikka
Viikkoharjoitustyö 1

Ryhmä:
Pasi Toivanen (517487)
Janina Kuosmanen (516580)
Santeri Loitomaa (516587)
Tommi Heikkinen

Viikkoharjoitustyö 1
22. syyskuuta 2018
?? sivua, 1 liites.

Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietotekniikka
Olio-ohjelmoinnin metodiikka

Sisältö

1 Tehtävä 1

Viittaaminen lukuun 3, toiseen lukuun 2, alilukuun 1.1, tätä alempaan lukuun 1.1.1, alimpaan lukuun 1.1.1, kuvaan 1.1 ja tauluun 1.

Kuva liitetään seuraavasti. ShareLaTeXin autocomplete rakentaa koko begin-end blockin yleensä puolestasi.

Taulukkoja tehdään seuraavasti.

Kirjallisuusviitteet lisätään bib-muodossa bibliografia tiedostoon ja niihin viitataan niiden ID:llä, joka on bib-muodon ensimmäinen kenttä **crawley2007write**.

1.1 Alaotsikko

Esimerkki viittaamisesta, jossa myös cite komennon tagi löytyy Bibliografia.bib tiedostosta **puasuareanu2009survey**.

1.1.1 Alempiotsikko

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eget tellus porttitor, tempus lacus non, pellentesque ligula. Donec sit amet erat condimentum, feugiat mi

Kuva 1.1: Kuvan otsikko

Taulukko 1.1: Taulukon otsikko tulee taulun yläpuolelle

Taulun	elementit	erotetaan
toisistaan	et-merkillä	
soluja voi myös		jättää tyhjäksi

accumsan, euismod quam.

Mauris laoreet maximus aliquet. Mauris at gravida elit. Ut nec lobortis elit. Sed lacinia nisi in ex sollicitudin, ac consequat lacus imperdiet. Etiam et velit eu lacus maximus faucibus.

Alinotsikko, joka ei näy sisällysluettelossa

```
1 // java-koodin voi kirjoittaa tallaiseen ymparistoon
2 // jolloin koodi syntaksivarjataan automaattisesti
3 // lyx ei tue tassa skandeja, mutta puhdas (xe)latex tukee
4 public class HelloWorld {
5   public static void main(String[] args) {
6     // Prints "Hello, World" to the terminal window.
7     System.out.println("Hello, World");
8   }
9 }
```

Otsikko tekstissä, joka ei näy sisällysluettelossa Mauris laoreet maximus aliquet. Mauris at gravida elit. Ut nec lobortis elit. Sed lacinia nisi in ex sollicitudin, ac consequat lacus imperdiet. Etiam et velit eu lacus maximus faucibus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec vulputate tellus ullamcorper odio sodales, non scelerisque neque eleifend.

2 Tehtävä 2

2.1 a-kohta

Loin data-abstraktioksi luokan Sudoku, jolla on ominaisuus kenttä. Kenttä on 9x9 matriisi, joka automaattisesti sisältää int arvoja välillä 0-9 (defaultilla 0). Alla Sudoku luokan constructori.

```
1  /**
2           * Luo sudokuruudukko olion, jolla on ominaisuutena 9x9 matriisi kentt .
3           * @.pre true
4           * @.post new Sudoku
5           */
6          public Sudoku(){
7                  int[][] kentt  = new int [9][9];
8          }
```

Luokka sisältää myös metodit lukujen katsomiseen tietyssä ruudussa ja lukujen lisäämiseen.

2.2 b-kohta

metodi onSudoku() ottaa vastaan objektin ja kertoo sitten oikeellisesti onko kyseessä sudoku olio, koska olion kenttä on jo valmiiksi määritelty siten ettei se ota vastaan muuta kuin inttejä, niin kauan kun olio on sudoku on sen data myös oikeellinen.

```
1  /**
2           * Tarkistaa ett  onko annettu objekti sudoku-data
3           * @.pre true
4           * @.post return true); || return false;
5           */
6          public static boolean onSudoku(Object o){
7                          if (o instanceof Sudoku) {
8                                  return true;
9                          }else {
10                                 return false;
11                         }
12         }
```

2.3 c-kohta

Metodi sudokuTila aloittaa tarkistamalla onko kyseesäs sudoku, hypäten suoraan loppuun jo se ei ole. Metodi käy sitten systemaattisesti läpi eri vaihtoehdot, tarkistaen onko jokaisessa ruudussa eri luku kuin 0. Tutkien sitten jokaisen pyst ja vaakarivin ja ruudukon, varmistaen ettei sama luku esiinny kahdesti. Kaikista näistä printataan teksti joka kertoo lopputuloksen.

```
1  /**
2           * Tarkistaa onko annettu objekti sudoku dataa, kertoo sen j lkeen onko
3           * @.pre true
4           * @.post (System.out.println("Objekti on Sudoku"); || System.out.println
5           */
6          public static void sudokuTila(Object o){
7                  boolean t ysi = true;
8                  boolean oikein = true;
9                  boolean brake = false;
10                 if (o instanceof Sudoku) {
11                         System.out.println("Objekti␣on␣Sudoku");
```

```
12                      // t st  alkaa tarkistus onko sudoku t ytetty vai kesk
13                      for (int x=0; x<9; x++) {
14                              for (int y=0; y<9; y++) {
15                                      if (((Sudoku) o).kentt [x][y]==0) {
16                                              t ysi = false;
17                                      }
18                              }
19                      }
20                      //T st  alkaa tarkistus onko sudoku oikein t ytetty (v
21                      for (int luku=1;luku<10;luku++) {
22                              for (int x=0; x<9; x++) {
23                                      int toistox = 0;
24                                      int toistoy = 0;
25                                      for (int y=0; y<9; y++) {
26                                              if (((Sudoku) o).kentt [x][y]==l
27                                                      toistox ++;
28                                              }
29                                              if (((Sudoku) o).kentt [y][x]==l
30                                                      toistoy ++;
31                                              }
32                                      }
33                                      if (brake){
34                                              break;
35                                      }
36                                      if (toistox>1 || toistoy>1) {
37                                              System.out.println("Sudoku␣on␣v
38                                              brake = true;
39                                      }
40                              }
41                              if (brake){
42                                      break;
43                              }
44                              for (int a=0;a<3;a++) {
```

```
45                                      for (int b=0;b<3;b++) {
46                                              int toistoz = 0;
47                                              for (int x=0;x<3;x++) {
48                                                      for (int y=0;y<3;y++) {
49                                                              if (((Sudoku) o).
50                                                                      toistoz +
51                                                              }
52                                                              if (toistoz>1) {
53                                                                      System.ou
54                                                                      brake = t
55                                                              }
56                                                              if (brake){
57                                                                      break;
58                                                              }
59                                                      }
60                                                      if (brake){
61                                                              break;
62                                                      }
63                                              }
64                                              if (brake){
65                                                      break;
66                                              }
67                                      }
68                                      if (brake){
69                                              break;
70                                      }
71                              }
72                              if (brake){
73                                      break;
74                              }
75                      }
76                      if (brake == false) {
77                              System.out.println("Sudoku on oikein t ytetty");
```

```
78                          }
79                          if (t ysi) {
80                                  System.out.println("Sudoku␣on␣t ysi");
81                          }else {
82                                  System.out.println("Sudoku␣ei␣ole␣t ysi");
83                          }
84                  }else {
85                          System.out.println("Objekti␣ei␣ole␣Sudoku");
86                  }
87          }
```

Vaihtoehtoisesti luokka voisi sisältää booleanit täysi, oikein ratkaistu yms. ja printtaukset voisi silloin korvata booleanien muokkauksella.

2.4 d-kohta

Metodi sudokuKaanto, ottaa vastaan objekti, varmistaen aluksi onko kyseessä sudoku, sitten kääntää muuttujan kenttä lukujen paikkoja. Tarkentaisin tehtävänantoa määrityksellä siitä kumpaan suuntaan sudokua tulee kääntää. (tämä versio kääntää myötäpäivään.)

```
1           /**
2            * Siirt   sudokun muuttujan kentt  lukujen paikkaa niin kuin oliota
3            * @.pre true
4            * @.post s.kentt [x][y] --> s.kentt [8-y][x] || System.out.println("Ob
5            */
6           public static void sudokuKaanto(Object s){
7                   if(onSudoku(s)) {
8                           Sudoku clone = new Sudoku();
9                           for (int x=0;x<9;x++) {
10                                  for (int y=0;y<9;y++) {
11                                          clone.t  yt  (8-y,x,((Sudoku)s).kentt [x
12                                  }
```

```
13                          }
14                          for (int x=0;x<9;x++) {
15                                  for (int y=0;y<9;y++) {
16                                          ((Sudoku)s).t  y t  (x,y,clone.k e n t t [x][
17                                  }
18                          }
19                  }else {
20                          System.out.println("Objekti␣ei␣ole␣sudoku!");
21                  }
22          }
```

2.5 e-kohta

Testin onnistuminen on varma, koska sudokun kääntö ei muuta sudoku oliota toiseksi olioksi, vaan muuttaa lukuja sen muuttujassa.

```
1  class FlipTest {
2
3          @RepeatedTest(value = 1000)
4          void test() {
5                  Sudoku sudoku = new Sudoku();
6                  T2.sudokuKaanto(sudoku);
7                  assertTrue(T2.onSudoku(sudoku));
8          }
9
10 }
```

Lopputulos 0 errors 0 failures

3 Tehtävä 4

Tehtävän tarkoituksena oli muodostaa oliorakenne, joka kuvaa Tetris-pelin palasia. Tetris-palasen kuvaus on jaettu kahteen luokkaan: Piece, joka kuvaa yhtä mielivaltaisen muotoista kappaletta – Shape, joka kuvaa Tetriksessä käytettävien kappaleiden muotoja. Luokkatoteutus tehtiin seuraavanlaisella java-koodilla:

```
1  /**
2   *
3   * Describes 4x4 piece in boolean values
4   * @author Pasi Toivanen
5   *
6   */
7  public class Piece {
8
9          private boolean[][] grid;
10         private int gridSize;
11
12 //--Constructor (Muodostimet)
13         /**
14          *
15          * @param shape
16          */
17         public Piece(Shape shape) {
18                 this.grid = shape.getGrid();
19                 this.gridSize = grid.length;
20         }
```

```
21
22          public void turnClockwise() {
23                  boolean[][] oldGrid = grid;
24                  grid = new boolean[gridSize][gridSize];
25                  for ( int i = 0; i < gridSize; i++) {
26                          for ( int j = 0; j < gridSize; j++) {
27                                  grid[i][j] = oldGrid[gridSize-j-1][i];
28                          }
29                  }
30          }
31
32  //--Getters (Havainnoijat)
33          public String toString() {
34                  String result = "";
35                  for ( int i = 0; i < grid.length; i++) {
36                          String printRow = "";
37                          for ( int j = 0; j < grid[i].length; j++) {
38                                  printRow += grid[i][j] ? "*" : "␣";
39                          }
40                          result += printRow + "\n";
41                  }
42                  return result;
43          }
44
45          public boolean[][] getGrid() {
46                  return grid;
47          }
48
49          public int getGridSize() {
50                  return gridSize;
51          }
52
53  //--Setters (Muutosoperaatiot)
```

```
54          /**
55           * TODO:gridSize changes grid and expands or shrinks it accordingly
56           * @.pre gridSize > 0
57           * @.post this.gridSize.equals(gridSize)
58           * @param gridSize
59           */
60          public void setGridSize(int gridSize) {
61                  this.gridSize = gridSize;
62          }
63
64          /**
65           * @.pre grid[i][j] = ( true || false ) for each i,j < gridSize
66           * @.post this.grid.equals(grid)
67           * @param grid TODO:this could be different size than this.grid
68           */
69          public void setGrid(boolean[][] grid) {
70                  this.grid = grid;
71          }
72
73  }
74
75  /**
76   *
77   * Describes a shape in 4x4 boolean grid
78   * @author Pasi Toivanen
79   *
80   */
81  enum Shape {
82          SQUARE,
83          LONG,
84          PYRAMID,
85          ZIGZAG;
86
```

```
87          public boolean[][] getGrid() {
88                  boolean[][] result = new boolean[4][4];
89                  switch(this) {
90                          case SQUARE:
91                                  result[1][1] = true; //
92                                  result[1][2] = true; // **
93                                  result[2][1] = true; // **
94                                  result[2][2] = true; //
95                                  break;
96                          case LONG:
97                                  result[0][1] = true; //
98                                  result[1][1] = true; //****
99                                  result[2][1] = true; //
100                                 result[3][1] = true; //
101                                 break;
102                         case PYRAMID:
103                                 result[0][2] = true; //
104                                 result[1][2] = true; // *
105                                 result[2][2] = true; //***
106                                 result[1][1] = true; //
107                                 break;
108                         case ZIGZAG:
109                                 result[0][1] = true; //
110                                 result[1][1] = true; //**
111                                 result[1][2] = true; // **
112                                 result[2][2] = true; //
113                                 break;
114                 }
115                 return result;
116         }
117
118         /**
119          * @return TODO: different gridSize for pieces smaller than 4x4
```

```
120             */
121             public int gridSize() {
122                     return 4;
123             }
124 }
```

Luokkarakenteella voidaan vastata kaikkiin kohtiin a-d. Vastaukset käydään läpi seuraavissa aliluvuissa.

3.1 A-tehtävä

Neliönmuotoinen Tetris-kappale voidaan luoda ja tulostaa seuraavasti:

```
1 Piece square = new Piece(Shape.SQUARE);
2 System.out.println(square.toString());
3 //Tulostaa konsoliin:
4 //
5 // **
6 // **
7 //
```

Itse kappaleen datarakenteen muodostaminen tapahtuu Shape-luokassa, joka tietää minkä muotoinen on neliö. Shape-luokasta saatu datarakenne tallennetaan Piece-luokan olion muodoksi ja se voidaan tulostaa.

3.2 B-tehtävä

Sattumanvarainen kappale voidaan luoda kirjoittamalla rivi:

```
1 Piece random = new Piece(Shape.values()[r.nextInt(Shape.values().length)]);
```

jossa r on tehtävään tarvittu Random-olio.

3.3 C-tehtävä

Kappale voidaan pyöräyttää käyttämällä Piece-olion turnCoclwise() -metodia:

```
1  //luodaan olio
2  Piece pyramid = new Piece(Shape.ZIGZAG);
3
4  //tulostetaan se ennen py r ytyst
5  System.out.println(pyramid.toString());
6
7  //py ritell  n nelj  kertaa kappaletta
8  for ( int i = 0; i < 4; i++) {
9          //Py r ytet  n 90  astetta my t p iv  n
10         pyramid.turnClockwise();
11
12         //tulostetaan jokaisen py r hdyksen j lkeen
13         System.out.println(pyramid.toString());
14 }
15 //Tuloste konsolissa:
16 //
17 //*
18 //**
19 // *
20 //
21 //
22 //
23 // **
24 //**
25 //
26 //
27 //
28 //*
29 //**
30 // *
```

```
31 //
32 //
33 //**
34 //**
35 //
36 //
37 //*
38 //**
39 // *
40 //
```

Pyöräytysmetodi käsittelee boolean arraytä tarkastelemalla jokaista indeksiä seuraavanlaisesti:

```
1         public void turnClockwise() {
2                 boolean[][] oldGrid = grid;
3                 grid = new boolean[gridSize][gridSize];
4                 for ( int i = 0; i < gridSize; i++) {
5                         for ( int j = 0; j < gridSize; j++) {
6                                 grid[i][j] = oldGrid[gridSize-j-1][i];
7                         }
8                 }
9         }
```

jossa jokainen kohta ruuduosta (grid) käydään ja kirjoitetaan soluun uusi oikea käännetty arvo.

3.4 D-tehtävä

Testataan mitä käy kappaleelle, jos sitä pyöräytetään neljäkertaa myötäpäivään. JUnit5-testimetodi on rakennettu seuraavanlaisesti:

```
1 /**
2  * Creates random shape, rotates it 4 times and checks if it creates the same str
```

```
 */
@RepeatedTest(value = 1000)
void test() {
        //d-kohta
        Random r = new Random();
        Shape random = Shape.values()[r.nextInt(4)];

        Piece original = new Piece(random);
        Piece rotated = new Piece(random);

        for( int i = 0; i < 4; i++) {
                rotated.turnClockwise();
        }

        assertEquals(rotated.toString(), original.toString());
}
```

Testin aikana muodostetaan sattumanvarainen muoto, josta rakennetaan kaksi identtistä kappaletta. Kappaleita pyöritetään neljäkertaa turnClockwise() -metodilla ja tämän jälkeen kappaleiden tulostumista verrataan.

Tulokseksi saadaan 1000 kertaa toistetulla toistokokeella:

```
// Runs: 1000/1000      Errors: 0       Failures:       0
```

joka osoittaa, että toString()-metodit palauttavat identtisen arvon, jolloin kappale tulostuu saman muotoisena 4 pyöräytyksen jälkeen. Voidaan olettaa, että silloin kappaleen grid-muuttujan data on myös identtinen pyöräytetyn kappaleen datan kanssa.

Liite A Liitedokumentti

Tässä esimerkki