Turun yliopisto
University of Turku

Olio-ohjelmoinnin metodiikka
Viikkoharjoitustyö 2

Ryhmä:
Pasi Toivanen (517487)
Janina Kuosmanen (516580)
Santeri Loitomaa (516587)
Tommi Heikkinen (517749)

Viikkoharjoitustyö 2
27. syyskuuta 2018
?? sivua

Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietotekniikka
Olio-ohjelmoinnin metodiikka

Sisältö

1 Tehtävä 1 **1**

1.1 fi.utu.oomkit -luokat 1

1.2 OOMApp -luokka sovelluksen ikkunan luomiseksi 2

1.3 Ohjelman ikkunan sisällön hallitseminen 2

1.4 Ohjelmanaikaiset tapahtumat 4

2 Tehtävä 2 **5**

2.1 a-kohta 5

2.2 b-kohta 5

2.3 c-kohta 6

2.4 d-kohta 9

2.5 e-kohta 9

3 Tehtävä 4 **11**

3.1 A-tehtävä 14

3.2 B-tehtävä 15

3.3 C-tehtävä 15

3.4 D-tehtävä 16

1 Tehtävä 1

Tehtävässä tutustutaan oom-kit -kehykseen ja siitä löytyviin luokkiin

1.1 fi.utu.oomkit -luokat

fi.utu.oomkit.gui -paketista löytyy luokat:

```
1  public interface Console {}
2  class DefaultDialogFactory implements DialogFactory {}
3  public interface DialogFactory {}
4  class FXConsole extends OutputStream implements Console {}
5  public interface MainWindow extends WindowContent {}
6  public abstract class MainWindowBase implements MainWindow {}
7  class MergedStream extends OutputStream {}
8  public abstract class OOMApp extends Application {}
9  public class ReactiveCanvas<X> extends SimpleCanvas implements Observer<X> {}
10 public class ReactiveLabel<T> extends Label implements Observer<T> {
11         public interface LabelHandler<T> {}
12 }
13 public class SimpleCanvas extends Canvas {}
14 class StreamWrapper extends OutputStream {}
15 public interface WindowContent {}
```

Joista kaikki ovat julkisia paketin sisällä, mutta paketin ulkopuolella näkyvistä on jätetty luokat DefaultDialogFactory, FXConsole, MergedStream ja StreamWrapper. fi.utu.oomkit -paketista löytyy luokat:

```
1  public final class AppConfiguration {}
2  public interface AppLogic extends KeyHandler, Scheduled {}
```

Joista kaikki ovat julkisia luokkia niin paketin sisällä kuin ulkopuolella.

1.2 OOMApp -luokka sovelluksen ikkunan luomiseksi

OOMApp -luokka peritään ja sille kirjoitetaan generateMainWindow -metodi.

GenerateMainWindow -metodilla määritellään mikä on ohjelman (=app) nimi ja kuinka ison alueen ohjelma käyttää. Metodin tulee palauttaa MainWindow -alaluokka. OOMApp -luokan käyttöä on esitetty DemoApp1 -tutoriaalissa seuraavasti:

```
1  public class DemoApp1 extends OOMApp {
2      // alustaa pelilogiikan
3      final static LaatikkoGameLogic gameLogic = new LaatikkoGameLogic();
4
5      // kytkee piirtopinnan käyttöliittymään
6      @Override
7      protected MainWindow generateMainWindow(String appName, double width,
8      double height) { return new SimpleMainWindow(appName, width, height) {
9              @Override public SimpleCanvas mainContent() {
10                 return gameLogic.piirtoPinta;
11             }
12         };
13     }
14 }
```

EmptyApp -luokassa generateMainWindow -metodille on tehty mielekäs esimerkki-implementaatio, jolloin sitä voi käyttää sellaisenaan muodostaessa ohjelman ikkunaa. Siitä löytyy valmiiksi ajettava main-metodi.

1.3 Ohjelman ikkunan sisällön hallitseminen

Ikkuna luodaan käyttämällä luokkaa MainWindow. Huomionarvoista on, että MainWindow -olion sisältö koostuu ylä- ja alapalkista, sekä Canvas-piirtoalueesta. Oletuksena nämä ovat tyhjiä, jolloin oman sisällön voi luoda ylikirjoittamalla metodit topBarContent(), mainContent(), bottomBarContent().

```
1  public class EmptyApp extends OOMApp {
2          @Override
3          protected MainWindow generateMainWindow(){
```

```
4                 return new SimpleMainWindow(appName, width, height) {
5
6                         @Override
7                         public List<Node> topBarContent() {
8                                 //return yläpalkin sisältö
9                         }
10
11                        @Override
12                        public Canvas mainContent() {
13                                //return pääsisältö
14                        }
15
16                        @Override
17                        public List<Node> bottomBarContent() {
18                                //return alapalkin sisältö
19                        }
20                }
21        }
22 }
```

Ylä- ja alapalkkiin voidaan lisätä mitä tahansa graafisia solmuja, jotka voivat olla erilaisia graafisia käyttöliittymäkomponentteja. Ylikirjoittamalla topBarContent() -metodin voidaan sijoittaa käyttöliittymään lista mitä tahansa elementtejä kuten esimerkiksi luokkien Canvas, ImageView, Shape tai Button luokkien olio. Yläpalkin lisäämisessä voitaisiin käyttää esimerkiksi seuraavaa metodia:

```
1 @Override
2 public List<Node> topBarContent() {
3     return Arrays.asList(
4         new Button("File"),
5         new Button("Edit"),
6         new Button("Window"),
7         new Button("Help")
8     );
9 }
```

1.4 Ohjelmanaikaiset tapahtumat

OOMApp -luokan olion konstruktorilla pystytään luomaan useanlaisia logiikoita ohjelman suoritukseksi. Oma logiikka voidaan lisätä käyttämällä OOMApp konstruktoria, johon syötetään ohjelmalogiikka. Ohjelmalogiikkaluokka peritään AppLogic -luokalta. Seuraavassa esimerkissä on esitetty, kuinka MyLogic -luokalla voidaan hallita ohjelmalogiikkaa niin, että asiakkaan koodi suoritetaan joka 25ms.

```
1  public class MyApp extends OOMApp {
2          public MyApp() {
3                  super(new MyLogic());
4          }
5  }
6
7  public class MyLogic extends AppLogic {
8          @override
9          AppConfiguration configuration() {
10                 return new AppConfiguration(25, "Asiakkaan ohjelma", true);
11         }
12
13         @override
14         void tick() {
15                 //asiakkaan koodi
16         }
17 }
```

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 kenttä.
3     * @.pre true
4     * @.post new Sudoku
5     */
6    public Sudoku(){
7      int[][] kenttä = 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 että 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
3       jälkeen onko ruudukko oikein täytetty ja täysi.
4     * @.pre true
5     * @.post (System.out.println("Objekti on Sudoku"); ||
6       System.out.println("Objekti ei ole Sudoku");) &&
7       ( System.out.println("Sudoku on väärin täytetty"); ||
8       System.out.println("Sudoku on oikein täytetty");) &&
9       (System.out.println("Sudoku on täysi"); ||
10      System.out.println("Sudoku ei ole täysi"); )
11    */
12   public static void sudokuTila(Object o){
13     boolean täysi = true;
14     boolean oikein = true;
15     boolean brake = false;
16     if (o instanceof Sudoku) {
17         System.out.println("Objekti␣on␣Sudoku");
18         // tästä alkaa tarkistus onko sudoku
19          täytetty vai kesken
20         for (int x=0; x<9; x++) {
```

```
21            for (int y=0; y<9; y++) {
22              if (((Sudoku) o).kenttä[x][y]==0) {
23                täysi = false;
24              }
25            }
26          }
27          //Tästä alkaa tarkistus onko sudoku oikein
28          täytetty (vasta vaakarivit)
29          for (int luku=1;luku<10;luku++) {
30            for (int x=0; x<9; x++) {
31              int toistox = 0;
32              int toistoy = 0;
33              for (int y=0; y<9; y++) {
34                if (((Sudoku) o).kenttä[x][y]==luku) {
35                  toistox ++;
36                }
37                if (((Sudoku) o).kenttä[y][x]==luku) {
38                  toistoy ++;
39                }
40              }
41              if (brake){
42                break;
43              }
44              if (toistox>1 || toistoy>1) {
45                System.out.println("Sudoku on väärin
46 täytetty");
47                brake = true;
48              }
49            }
50            if (brake){
51              break;
52            }
53            for (int a=0;a<3;a++) {
54              for (int b=0;b<3;b++) {
55                int toistoz = 0;
56                for (int x=0;x<3;x++) {
57                  for (int y=0;y<3;y++) {
58                    if (((Sudoku) o).kenttä[x+a*3]
59                    [y+b*3]==luku) {
60                      toistoz ++;
61                    }
```

```
62                  if (toistoz>1) {
63                    System.out.println("Sudoku
64 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣on␣väärin␣täytetty");
65                    brake = true;
66                  }
67                  if (brake){
68                    break;
69                  }
70                }
71                if (brake){
72                  break;
73                }
74               }
75              if (brake){
76                break;
77              }
78            }
79            if (brake){
80              break;
81            }
82          }
83          if (brake){
84            break;
85          }
86        }
87        if (brake == false) {
88          System.out.println("Sudoku␣on␣oikein
89 ␣␣␣␣␣␣␣␣␣täytetty");
90        }
91        if (täysi) {
92          System.out.println("Sudoku␣on␣täysi");
93        }else {
94          System.out.println("Sudoku␣ei␣ole␣täysi");
95        }
96      }else {
97        System.out.println("Objekti␣ei␣ole␣Sudoku");
98      }
99    }
```

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   * Siirtää sudokun muuttujan kenttä lukujen paikkaa niin kuin
3     oliota käännettäisiin 90 astetta.
4   * @.pre true
5   * @.post s.kenttä[x][y] --> s.kenttä[8-y][x] ||
6     System.out.println("Objekti ei ole sudoku!");
7   */
8  public static void sudokuKaanto(Object s){
9    if(onSudoku(s)) {
10     Sudoku clone = new Sudoku();
11     for (int x=0;x<9;x++) {
12       for (int y=0;y<9;y++) {
13         clone.täytä(8-y,x,((Sudoku)s).kenttä[x][y]);
14       }
15     }
16     for (int x=0;x<9;x++) {
17       for (int y=0;y<9;y++) {
18         ((Sudoku)s).täytä(x,y,clone.kenttä[x][y]);
19       }
20     }
21   }else {
22     System.out.println("Objekti ei ole sudoku!");
23   }
24 }
```

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

Tekijänä Tommi Heikkinen (517749)

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 pyöräytystä
5  System.out.println(pyramid.toString());
6
7  //pyöritellään neljä kertaa kappaletta
8  for ( int i = 0; i < 4; i++) {
9          //Pyöräytetään 90   astetta myötäpäivään
10         pyramid.turnClockwise();
11
12         //tulostetaan jokaisen pyörähdyksen jälkeen
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 string
3  */
4 @RepeatedTest(value = 1000)
5 void test() {
6         //d-kohta
```

```
7         Random r = new Random();
8         Shape random = Shape.values()[r.nextInt(4)];
9
10        Piece original = new Piece(random);
11        Piece rotated = new Piece(random);
12
13        for( int i = 0; i < 4; i++) {
14                rotated.turnClockwise();
15        }
16
17        assertEquals(rotated.toString(), original.toString());
18  }
```

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:

```
1  // 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.

Tekijänä Pasi Toivanen (517487)