Turun yliopisto
University of Turku

Olio-ohjelmoinnin metodiikka
Viikkoharjoitustyö 1

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

Viikkoharjoitustyö 1
22. syyskuuta 2018
25 sivua
Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietotekniikka
Olio-ohjelmoinnin metodiikka

Sisältö

1 Tehtävä 1

Tässä tehtävässä pyydettiin kehittämään graafisen ohjelman värienhallinta metodeja.

1.1 Kohta (1.a)

Kohdassa (1.a) pyydettiin luomaan data-abstraktio väreille. Toteutin tämän luomalla Colour-luokan, jossa värit voidaan erottaa toisistaan String arvolla colour. Kohdassa pyydettiin myös luomaan 10 perusväriä kyseisellä menetelmällä. Itse päädyin väreihin Red, Blue, Yellow, Purple, Green, Orange, Brown, Black, Gray ja White.

Tässä Colour-olion konstruktori

```
1  /**
2   * Creates a Colour object.
3   * @.post new Colour created.
4   * @param String colour
5   */
6  public Colour(String colour) throws InputMismatchException{
7          if(colour == null) {
8                  throw new InputMismatchException(
9                  "This colour is not supported.");
10         }
11         this.colour = colour;
12 }
```

Tässä Colour-olioiden luonti ArrayList-olioon

```
1 ArrayList<Colour> colours = new ArrayList<Colour>(Arrays.asList(
2         new Colour("Red"), new Colour("Blue"), new Colour("Yellow"),
3         new Colour("Purple"), new Colour("Green"),
4         new Colour("Orange"), new Colour("Brown"),
5         new Colour("Black"), new Colour("Gray"),
6         new Colour("White")));
```

1.2 Kohta (1.b)

Kohdassa (1.b) pyydettiin luomaan jokin metodi, jonka avulla voitaisiin tunnistaa, ovatko 2 väriä samat. Tekemäni Colour-oliot voidaan erotella toisistaan niiden String colour-arvon perusteella.

Tässä yksinkertainen equals()-metodi

```
1  /**
2   * Returns true if the colours are the same.
3   * @param Colour colour
4   * @return true if the same Colour, false if not.
5   */
6  @Test
7  public boolean equals(Colour colour) {
8          if(colour == null) {
9                  return false;
10         }
11         else if(this.colour.equals(colour.colour))
12                 return true;
13         return false;
14 }
```

1.3 Kohta (1.c)

Kohdassa (1.c) kysytään määrittelyn teknisistä rajoitteista ja sopivuudesta ulkopuoliseen käyttöön. Tällä hetkellä Colour-olio ei ole oikein käyttökelpoinen, sillä sillä ei ole kohtaa värin hex-arvolle, jonka koen hyvinkin tärkeäksi. Se tosin on helposti lisättävissä pienellä päivityksellä. Tekemäni Colour olio tosin on hyvinkin helppo ottaa käyttöön ulkopuoliseen sovellukseen, kuten itse todistin tekemällä juuri niin. Lopullinen ohjelma, josta suoritukseni näkee, on saatavilla GitHubistani. Sen tärkeänä osana toimii Colours-luokka.

Colours-luokka

```
1  public class Colours {
2          static ArrayList<Colour> colours = new ArrayList<Colour>();
3          static ArrayList<Colour> randomColours =
4                   new ArrayList<Colour>();
5          /**
6           * Creates the Colours.txt with the default 10 colours if it
7           * doesn't exsist and creates an ArrayList out of it.
8           * @.post !Colours.equals(new ArrayList<Colour>())
9           */
10         @SuppressWarnings("unchecked")
11         public static void init() {
12                 try {
13                         FileInputStream file =
14                                 new FileInputStream("Colours.txt");
15                         ObjectInputStream load =
16                                 new ObjectInputStream(file);
17                         colours = (ArrayList<Colour>) load.readObject();
18                         load.close();
19                 } catch (FileNotFoundException e) {
20                         colours = new ArrayList<Colour>(Arrays.asList
```

```
21                      (new Colour("Red"), new Colour("Blue"),
22                      new Colour("Yellow"), new Colour("Purple"),
23                      new Colour("Green"), new Colour("Orange"),
24                      new Colour("Brown"), new Colour("Black"),
25                      new Colour("Gray"),
26                      new Colour("White")));
27                      try {
28                              FileOutputStream file =
29                              new FileOutputStream("Colours.txt");
30                              ObjectOutputStream save =
31                              new ObjectOutputStream(file);
32                              save.writeObject(colours);
33                              save.close();
34                      } catch (IOException e1) {
35
36                      }
37              } catch (IOException e) {
38                      colours = new ArrayList<Colour>(Arrays.asList
39                      (new Colour("Red"), new Colour("Blue"),
40                      new Colour("Yellow"), new Colour("Purple"),
41                      new Colour("Green"), new Colour("Orange"),
42                      new Colour("Brown"), new Colour("Black"),
43                      new Colour("Gray"),
44                      new Colour("White")));
45              } catch (ClassNotFoundException e) {
46                      colours = new ArrayList<Colour>(Arrays.asList
47                      (new Colour("Red"), new Colour("Blue"),
48                      new Colour("Yellow"), new Colour("Purple"),
49                      new Colour("Green"), new Colour("Orange"),
50                      new Colour("Brown"), new Colour("Black"),
51                      new Colour("Gray"),
52                      new Colour("White")));
53              }
```

```
54          }
55          /**
56           * This method returns true if the Colour can be used and
57           * false if it cannot be used.
58           * @param Colour c
59           * @return true if can be used, otherwise false
60           */
61          public static boolean canUseColour(Colour colour) {
62                  if(colour == null) return false;
63                  else if(colours.contains(colour)) return true;
64                  return false;
65          }
66          /**
67           * This method returns true if the Colour can be used and
68           * false if it cannot be used.
69           * @param Colour c
70           * @return true if can be used, otherwise false
71           */
72          public static boolean canUseRandomColour(Colour colour) {
73                  if(colour == null) return false;
74                  else if(randomColours.contains(colour)) return true;
75                  return false;
76          }
77          /**
78           * This method is used to add a colour to the colours ArrayList
79           * @.post colours = OLD.colours + colour
80           * @param colour (to add)
81           */
82          public static void addColour(Colour colour){
83                  if(colour == null) return;
84                  colours.add(colour);
85                  try {
86                          FileOutputStream colours =
```

```
87                              new FileOutputStream("Colours.txt");
88                      ObjectOutputStream save =
89                              new ObjectOutputStream(colours);
90                      save.writeObject(colours);
91                      save.close();
92              } catch (IOException e1) {
93
94              }
95      }
96      /**
97       * This method is used to add a colour to the colours ArrayList
98       * @.post colours = OLD.colours + colour
99       * @param colour (to add)
100      */
101     @Test
102     public static void addRandomColour(Colour colour){
103             if(colour == null) return;
104             randomColours.add(colour);
105     }
106     /**
107      * A method that will randomize colours for an ArrayList that
108      * contains a random amount of colours from the colours
109      * ArrayList.
110      * @.post The contents of the randomColours ArrayList changes.
111      */
112     public static void createRandomColours() {
113             randomColours = new ArrayList<Colour>();
114             Random r = new Random();
115             for(int i = r.nextInt(colours.size()-1);
116              i < colours.size(); i++) {
117                     Colour c =
118                     colours.get(r.nextInt(colours.size()));
119                     if(!randomColours.contains(c))
```

```
120                                 randomColours.add(c);
121                 }
122         }
123         /**
124          * A getter for the randomColours ArrayList.
125          * @return randomColours
126          */
127         public static ArrayList<Colour> getRandomColours(){
128                 return randomColours;
129         }
130         /**
131          * A getter for the colours ArrayList.
132          * @return colours
133          */
134         public static ArrayList<Colour> getColours(){
135                 return colours;
136         }
137 }
```

1.4 Kohta (1.d)

Kohdassa (1.d) pyydettiin luomaan metodi, joka arpoo käyttäjän käyttöön satunnaiset värit. Toteutin sen Colours-luokassa näin:

```
1  /**
2   * A method that will randomize colours for an ArrayList that
3   * contains a random amount of colours from the colours
4   * ArrayList.
5   * @.post The contents of the randomColours ArrayList randomly changes.
6   */
7  public static void createRandomColours() {
8          randomColours = new ArrayList<Colour>();
9          Random r = new Random();
10         for(int i = r.nextInt(colours.size()-1);
11          i < colours.size(); i++) {
12                 Colour c =
13                 colours.get(r.nextInt(colours.size()));
14                 if(!randomColours.contains(c))
15                         randomColours.add(c);
16         }
17 }
```

Metodi arpoo satunnaisen määrän satunnaisia värejä Random r-olion avulla. Koska lopputulos on satunnainen, totean @.post-kohdassa lopputuloksen olevan randomColours ArrayList-olion satunnainen muutos.

1.5 Kohta (1.e)

Kohdassa (1.e) pyydettiin luomaan JUnit testi equals()-metodille, joka luotiin tehtävässä **Kohta (1.b)**.

Tässä testiluokkani ja sen tulostukset

```
1  class ColoursTest {
2
3          @BeforeAll
4          static void setUpBeforeClass() throws Exception {
5                  Colours.init();
6          }
7
8          @Test
9          void test() {
10                 System.out.print("This␣should␣be␣true:␣");
11                 System.out.println(Colours.getColours().get(0)
12                         .equals(Colours.getColours().get(0)));
13                 System.out.print("This␣should␣be␣false:␣");
14                 System.out.println(Colours.getColours().get(0)
15                         .equals(Colours.getColours().get(1)));
16         }
17
18 }
19
20 This should be true: true
21 This should be false: false
22
23 0 errors 0 failures
```

Tekijänä Santeri Loitomaa (516587).

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
3       j lkeen onko ruudukko oikein t ytetty ja t ysi.
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 v  r in  t ytetty"); ||
8       System.out.println("Sudoku on oikein t ytetty");) &&
9       (System.out.println("Sudoku on  t ysi"); ||
10      System.out.println("Sudoku ei ole  t ysi"); )
11    */
```

```
12  public static void sudokuTila(Object o){
13    boolean t y s i = true;
14    boolean oikein = true;
15    boolean brake = false;
16    if (o instanceof Sudoku) {
17        System.out.println("Objekti␣on␣Sudoku");
18        // t  s t   alkaa tarkistus onko sudoku
19         t y t e t t y vai kesken
20        for (int x=0; x<9; x++) {
21          for (int y=0; y<9; y++) {
22            if (((Sudoku) o).k e n t t [x][y]==0) {
23              t y s i = false;
24            }
25          }
26        }
27        //T  s t   alkaa tarkistus onko sudoku oikein
28        t y t e t t y (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).k e n t t [x][y]==luku) {
35                toistox ++;
36              }
37              if (((Sudoku) o).k e n t t [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 v  r i n
46 t y t e t t y");
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).kentt [x+a*3]
59                  [y+b*3]==luku) {
60                    toistoz ++;
61                  }
62                  if (toistoz>1) {
63                    System.out.println("Sudoku
64 on v  r i n t y t e t t y");
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 ␣␣␣␣␣␣␣␣␣␣t ytetty");
90          }
91          if (t ysi) {
92            System.out.println("Sudoku␣on␣t ysi");
93          }else {
94            System.out.println("Sudoku␣ei␣ole␣t ysi");
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   * Siirt   sudokun muuttujan kentt  lukujen paikkaa niin kuin
3     oliota k  nnett isiin 90 astetta.
4   * @.pre true
5   * @.post s.kentt [x][y] --> s.kentt [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.t  yt  (8-y,x,((Sudoku)s).kentt [x][y]);
14       }
15     }
16     for (int x=0;x<9;x++) {
17       for (int y=0;y<9;y++) {
18         ((Sudoku)s).t  yt  (x,y,clone.kentt [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 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.

Tekijänä Pasi Toivanen (517487)