Turun yliopisto
University of Turku

Olio-ohjelmoinnin metodiikka
Viikkoharjoitustyö 5

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

Viikkoharjoitustyö 5
28. lokakuuta 2018
17 sivua

Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietotekniikka
Olio-ohjelmoinnin metodiikka

Sisältö

1 Tehtävä 1

Tehtävässä muokataan Gorilla -pelin rakennusluokkia niin, että niihin voidaan toteuttaa järjestelyalgoritmeja.

1.1 Leveyden ja korkeuden mukainen järjestys

Lajittelua varten muodostetaan ensiksi Rakennus -luokkaan Comparator, niin leveys kuin korkeusjärjestyksen muodostamiseen.

```
1  public static Comparator<Rakennus> LeveysJärjestys = new Comparator<Rakennus>() {
2          public int compare(Rakennus eka, Rakennus toka) {
3                  return eka.getLeveys() - toka.getLeveys();
4          }
5  };
6
7  public static Comparator<Rakennus> KorkeusJärjestys = new Comparator<Rakennus>() {
8          public int compare(Rakennus eka, Rakennus toka) {
9                  return eka.getKorkeus() - toka.getKorkeus();
10         }
11 };
```

Jos halutaan, että Arrays.sort() -metodi järjestää rakennukset ensisijaisesti korkeuden mukaan, niin Rakennus -luokan tulee toteuttaa Comparable -rajapinta ja ylikirjoittaa metodi compareTo() seuraavalla tavalla:

```
1  /**
2   * Paleutetaan ensisijaisesti korkeusjärjestys, mutta jos rakennukset ovat
3   * yhtä korkeita, niin silloin palautetaan leveysjärjestys.
4   */
```

```
5  @Override
6  public int compareTo(Rakennus that) {
7          int korkeusjärjestys = Rakennus.KorkeusJärjestys.compare(this,that);
8          return korkeusjärjestys != 0 ?
9                  korkeusjärjestys : Rakennus.LeveysJärjestys.compare(this, that);
10 }
```

Jolloin tilanteet, joissa rakennukset ovat yhtä korkeita arvioidaan rakennusten leveyden perusteella.

1.2 Järjestysnumeron esittäminen

Ohjelmassa järjestysnumerot voidaan piirtää rakennusten alapuolelle käyttämällä edellisessä luvussa esiteltyä compareTo() -metodia listan järjestämisessä. Nyt listan sort() -metodi on tosin staattinen. Rakennusten piirtämistä voidaan helposti muokata näyttämään järjestysnumero kirjoittamalla drawForegroundContent() -metodiin muunneltu silmukka:

```
1  ArrayList<Rakennus> järjestettyKopio = new ArrayList<>(rakennukset);
2  järjestettyKopio.sort(null);
3
4  for (Rakennus r : rakennukset) {
5          Rakennus skaalattu = r.skaalaa(suhdeLeveys, suhdeKorkeus);
6          piirräRakennus(paikka, skaalattu);
7          drawText(paikka.add(skaalattu.leveys / 2, -32), CoreColor.White,
8                  järjestettyKopio.indexOf(r) + "#", 24, true, true); paikka =
9                  paikka.add(skaalattu.leveys, 0);
10 }
```

Silmukkaa ennen rakennusten järjestys laitetaan piirtämistä varten muistiin. Tämä muistinkäyttö olisi järkevämpi tehdä jossain muussa luokassa, mutta koska tässä esimerkissä rakennukset on tallennettu PiirtelyPinta -luokan sisään, niin käytämme sitten myös PiirtelyPinta -luokkaa järjestyksen tallentamiseen.

Jotta saataisiin järjestystä vaihdettua ajon aikana GorillaPiirtopinta -luokassa tulee myös tietää missä järjestyksessä järjestysnumero pitää esittää, joten tallennetaan

staattiseen muuttujaan käytettävä Comparator.

```
1  public class GorillaPiirtopinta extends PiirtelyPinta {
2      private final List<Rakennus> rakennukset;
3      private Comparator<Rakennus> järjestys = Rakennus.KorkeusJärjestys;
4      jne
5  }
```

Ja muokataan piirtämistä hiukan:

```
1  ArrayList<Rakennus> järjestettyKopio = new ArrayList<>(rakennukset);
2  järjestettyKopio.sort(järjestys);
3
4  for (Rakennus r : rakennukset) {
5          Rakennus skaalattu = r.skaalaa(suhdeLeveys, suhdeKorkeus);
6          piirräRakennus(paikka, skaalattu);
7          drawText(paikka.add(skaalattu.leveys / 2, -32), CoreColor.White,
8                  järjestettyKopio.indexOf(r) + "#", 24, true, true); paikka =
9                  paikka.add(skaalattu.leveys, 0);
10 }
```

jossa oletusjärjestyksen (null-arvon) sijaan käytetäänkin staattiseen muuttujaan tallennettua järjestystä (ks. rivi 2). Jäljelle jää tehdä toiminnallisuus alla oleviin painikkeisiin niin, että järjestystä voidaan muuttaa ajon aikana.

Koska luokkarakenne ei ole erityisen eheä (edellämainittu ristiriita piirtämiseen tarkoitetun luokan sisälle tallennettu datarakenne), niin tehdään järjestäRakennukset() -metodi, jonka argumenttina välitetään oikea järjestys:

```
1  void järjestäRakennukset(Comparator<Rakennus> c) {
2                  rakennukset.sort(c); // tullaan siirtämään seuraavassa tehtävässä pois täältä
3                  järjestys = c;
4  }
```

jolloin ohjelman logiikan puolella voidaan muuttaa staattista järjestystä:

```
1  @Override
2  public List<Node> bottomBarContent() {
3          List<Node> napit = new ArrayList<Node>(Arrays.asList(
4                  new Button("Järjestys␣#1") {
5                          { setOnAction(e ->
6                          appLogic.järjestäRakennukset(Rakennus.KorkeusJärjestys));
```

```
7                 } },
8                 new Button("Järjestys␣#2") {
9                         { setOnAction(e ->
10                        appLogic.järjestäRakennukset(Rakennus.LeveysJärjestys)); }
11                },
12                new Button("Lajittelu") {
13                        { setOnAction(e ->
14                        System.out.println("TODO")); }
15                }
16        ));
17        napit.addAll(basicButtons());
18        return napit;
19 }
```

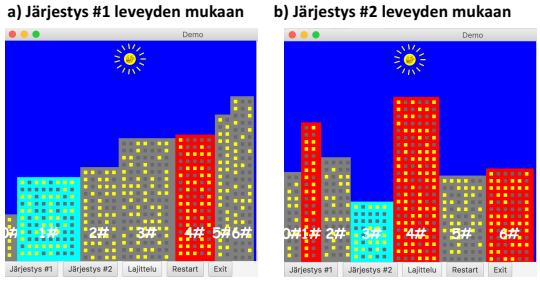
1.3 Lajittelu ajon aikana

Lisätään lajittelu-painikkeelle oma metodi, joka päivittää rakennusten järjestyksen. Tämä tehdään GorillaPiirtopinta ja GorillaLogic -luokkiin. Samalla eritellään järjestyksen ja lajittelun käsitteet toisistaan:

```
1 void järjestäRakennukset(Comparator<Rakennus> c) {
2                 järjestys = c;
3 }
4
5 public void lajitteleRakennukset() {
6                 rakennukset.sort(järjestys);
7 }
```

jolloin nyt nappien toiminta kirjoitetaan näin:

```
1 new Button("Järjestys␣#1") {
2     { setOnAction(e ->
3     appLogic.järjestäRakennukset(Rakennus.KorkeusJärjestys)); }
4 },
5 new Button("Järjestys␣#2") {
6     { setOnAction(e ->
7     appLogic.järjestäRakennukset(Rakennus.LeveysJärjestys)); }
8 },
9 new Button("Lajittelu") {
```



Kuva 1.1: Lajittelunäkymä korkeus- ja leveysjärjestyksessä

```
10      { setOnAction(e ->
11      appLogic.lajitteleRakennukset()); }
12  }
```

Lajiteltu näkymä on esitetty kuvassa 1.1.

1.4 Rakennusten sekoittaminen

Jos rakennukset halutaan sekoittaa silloin, kun halutaan järjestää jo järjestyksessä olevaa listaa rakennuksista, niin tämä voidaan toteuttaa muokkaamalla lajittelu -metodia:

```
1  public void lajitteleRakennukset() {
2          ArrayList<Rakennus> rakennuksetOld = new ArrayList<>(rakennukset);
3          rakennukset.sort(järjestys);
4          if ( rakennukset.equals(rakennuksetOld) ) {
5                  Collections.shuffle(rakennukset);
6          }
7  }
```

Jolloin rakennukset sekoitetaan vain, jos muutosta järjestykseen ei tapahdu.

Tehnyt Pasi Toivanen (517487)

2 Tehtävä 2

2.1 a-kohta

Loin luokan Kuvio, jolla on aiemmassa tehtävässä määritellyt ominaisuudet väri, täytetty, keskipiste ja lisäksi kulmien määrä, joka määrittää mikä kuvion tyyppi on kyseessä. Tyypit olivat tehtävässä enumeina YMPYRÄ, NELIÖ, KOLMIO.

```
1  public class Kuvio{
2    int kulmaMaara;
3    Point keskusta;
4    Color vari;
5    double koko;
6    boolean taytetty;
7    enum Tyyppi{
8      NELIÖ, YMPYRÄ, KOLMIO
9    }
10   Tyyppi tyyppi;
11   /**
12    *
13    * Luo uuden kuvio olion, katkaisten luonnin jos kulmaMaara ei täsmää kuvioihin.
14    * @.pre kulmaMaara = 0 || 3 || 4
15    * @.post new Kuvio
16    */
17   public Kuvio(int kulmaMaara, Point keskusta, Color vari, boolean taytetty) {
18     if (kulmaMaara != 0 && kulmaMaara != 3 && kulmaMaara != 4) {
19       System.out.println("Kuvio␣ei␣ole␣ympyrä,␣neliö␣tai␣kolmio!");
20       return;
21     }
22     this.kulmaMaara = kulmaMaara;
23     this.keskusta = keskusta;
24     this.vari = vari;
```

```
25      this.taytetty = taytetty;
26      if (kulmaMaara == 0) {
27        this.tyyppi = Tyyppi.YMPYRÄ;
28        this.koko = 100;
29      }
30      else if(kulmaMaara == 3){
31        this.tyyppi = Tyyppi.KOLMIO;
32        this.koko = 30;
33      }
34      else if(kulmaMaara == 4){
35        this.tyyppi = Tyyppi.NELIÖ;
36        this.koko = 60;
37      }
38    }
39  }
```

Kuviolla on etukäteen määritetty koko, jotta tehtävä täyttäisi suuruusjärjestykseen liittyvät ehdot.

Kuviopari perii luokan kuvio ja toimii hyvin samalla tavalla, mutta sillä on lisää muuttujia sisemmän kuvion ominaisuuksille. Sisemmän ja ulomman kuvion kokoja muokataan tässä sen mukaan onko sisäkuvio sama vai eri.

```
1  public class KuvioPari extends Kuvio{
2    int sisaKulmaMaara;
3    Color sisaVari;
4    double sisaKoko;
5    boolean sisaTaytetty;
6    enum Tyyppi{
7      NELIÖ, YMPYRÄ, KOLMIO
8    }
9    Tyyppi sisaTyyppi;
10   /**
11    *
12    * Luo uuden kuviopari olion, katkaisten luonnin jos kulmaMaara ei täsmää kuvioihin.
13    * @.pre kulmaMaara = 0 || 3 || 4 && sisaKulmaMaara = 0 || 3 || 4
14    * @.post new KuvioPari
15    */
16   public KuvioPari(int kulmaMaara, int sisaKulmaMaara, Point keskusta, Color vari,
17   Color sisaVari, boolean taytetty, boolean sisaTaytetty) {
18     super(kulmaMaara,keskusta,vari,taytetty);
```

```
19      if (sisaKulmaMaara != 0 && sisaKulmaMaara != 3 && sisaKulmaMaara != 4) {
20        System.out.println("Sisäkuvio␣ei␣ole␣ympyrä,␣neliö␣tai␣kolmio!");
21        return;
22      }
23      this.sisaKulmaMaara = sisaKulmaMaara;
24      this.sisaVari = sisaVari;
25      this.sisaTaytetty = sisaTaytetty;
26      if (sisaKulmaMaara == 0) {
27        this.sisaTyyppi = Tyyppi.YMPYRÄ;
28        this.sisaKoko = 100;
29      }
30      else if(kulmaMaara == 3){
31        this.sisaTyyppi = Tyyppi.KOLMIO;
32        this.sisaKoko = 30;
33      }
34      else if(kulmaMaara == 4){
35        this.sisaTyyppi = Tyyppi.NELIÖ;
36        this.sisaKoko = 60;
37      }
38      if (kulmaMaara == sisaKulmaMaara) {
39        this.koko = this.koko + 20;
40        this.sisaKoko = this.sisaKoko + 10;
41      }else {
42        this.koko = this.koko + 10;
43        this.sisaKoko = this.sisaKoko - 10;
44      }
45    }
46  }
```

Koska kuviopari perii kuvion, molempia voi käyttää Kuvion metodissa vertaile, joka laskee kuvioiden pinta-alan ja määrittelee siten mikä on suurin.

```
1   /**
2    * Vertailee kahta kuviota toisiinsa koon suhteen
3    * @.post System.out.println("Toinen kuvio on isompi." || "Ensimmäinen kuvio on
4    isompi." || "Kuviot ovat yhtä suuret");
5    */
6   public void vertaile(Kuvio kuvio) {
7     double suuruus1 = 0;
8     double suuruus2 = 0;
9     if (this.tyyppi == Tyyppi.KOLMIO) {
10      suuruus1 = this.koko*this.koko*0.433;
```

```
11      }
12      if (this.tyyppi == Tyyppi.YMPYRÄ) {
13        suuruus1 = (this.koko/2)*(this.koko/2)*Math.PI;
14      }
15      if (this.tyyppi == Tyyppi.NELIÖ) {
16        suuruus1 = this.koko*this.koko;
17      }
18      if (kuvio.tyyppi == Tyyppi.KOLMIO) {
19        suuruus2 = kuvio.koko*kuvio.koko*0.433;
20      }
21      if (kuvio.tyyppi == Tyyppi.YMPYRÄ) {
22        suuruus2 = (kuvio.koko/2)*(kuvio.koko/2)*Math.PI;
23      }
24      if (kuvio.tyyppi == Tyyppi.NELIÖ) {
25        suuruus2 = kuvio.koko*kuvio.koko;
26      }
27      if (suuruus1 < suuruus2) {
28        System.out.println("Toinen kuvio on isompi.");
29      } else if(suuruus1 > suuruus2) {
30        System.out.println("Ensimmäinen kuvio on isompi.");
31      } else {
32        System.out.println("Kuviot ovat yhtä suuret");
33      }
34    }
```

Lopuksi testasin vertailun toimivuutta luomalla kuviot testi1-9 jotka ovat kukin joko pelkkä kuvio, kuvio joka sisältää eri kuvion kuin itsensä ja kuvio joka sisältää itsensä muotoisen kuvion. Testasin aluksi vartavasten test2 ja test1 kokoaeroa ja sitten test2 itsensä kanssa varmistaakseni vertailumetodin antavan kaikki halutut tulokset. Sitten testasin loput järjestyksessä, varmistaakseni niiden järjestyksen olevan tehtävänannon ohjeiden mukainen.

Alla kutsupino ja sitten tulostus

```
1       Point keski = new Point(100, 100);
2       Kuvio test1 = new Kuvio(3,keski,CoreColor.Red,true);
3       KuvioPari test2 = new KuvioPari(3,4,keski,CoreColor.Red,CoreColor.Red,true,true);
4       KuvioPari test3 = new KuvioPari(3,3,keski,CoreColor.Red,CoreColor.Red,true,true);
5       Kuvio test4 = new Kuvio(4,keski,CoreColor.Red,true);
6       KuvioPari test5 = new KuvioPari(4,0,keski,CoreColor.Red,CoreColor.Red,true,true);
```

```
7       KuvioPari test6 = new KuvioPari(4,4,keski,CoreColor.Red,CoreColor.Red,true,true);
8       Kuvio test7 = new Kuvio(0,keski,CoreColor.Red,true);
9       KuvioPari test8 = new KuvioPari(0,4,keski,CoreColor.Red,CoreColor.Red,true,true);
10      KuvioPari test9 = new KuvioPari(0,0,keski,CoreColor.Red,CoreColor.Red,true,true);
11      test2.vertaile(test1);
12      test2.vertaile(test2);
13      test2.vertaile(test3);
14      test3.vertaile(test4);
15      test4.vertaile(test5);
16      test5.vertaile(test6);
17      test6.vertaile(test7);
18      test7.vertaile(test8);
19      test8.vertaile(test9);
20  ''Ensimmäinen kuvio on isompi.
21    Kuviot ovat yhtä suuret
22    Toinen kuvio on isompi.
23    Toinen kuvio on isompi.
24    Toinen kuvio on isompi.
25    Toinen kuvio on isompi.
26    Toinen kuvio on isompi.
27    Toinen kuvio on isompi.
28    Toinen kuvio on isompi.''
```

2.2 b-kohta

Kuvio voi olla vain ympurä, neliö tai kolmio, sillä konstruktori varmistaa kulmamäärän heti alussa ja katkaisee koodin toiminnan jos lukema ei täsmää.

```
1     if (kulmaMaara != 0 && kulmaMaara != 3 && kulmaMaara != 4) {
2       System.out.println("Kuvio␣ei␣ole␣ympyrä,␣neliö␣tai␣kolmio!");
3       return;
4     }
```

2.3 c-kohta

Kaikki luokat toimivat ArrayListissa, HashMapissa ja TreeMapissa, niin kauan kun tiesi joko objektin indexin listassa tai avaimen jolla sen sai haettua, sillä silloin ne sai ulos

alkuperäisessä muodossaan ja pystyi castaamaan tarvittaessa. HashSetissä ja TreeSetissä, ei objekteja saanut palautettua samalla tavalla, joten mikään ei toiminut.

2.4 d-kohta

Luokassa Tallennettava on kaksi Serializable arvoa jotka tallennetaan, ja yksi transient arvo jota ei tallenneta, mainissa oleva tallennusmetodi ajettaessa syntyi tallennus.tal tiedosto.

```
1  import java.io.FileOutputStream;
2  import java.io.IOException;
3  import java.io.ObjectOutputStream;
4  import java.io.Serializable;
5  public class Main {
6    public static void main(String[] args) {
7      Tallennettava t = new Tallennettava();
8      try {
9        FileOutputStream tiedosto = new FileOutputStream("tallennus.tal");
10       ObjectOutputStream tallenna = new ObjectOutputStream(tiedosto);
11       tallenna.writeObject(t);
12       tallenna.close();
13     } catch (IOException e) {
14       e.printStackTrace();
15     }
16   }
17 }
18 class Tallennettava implements Serializable{
19   private static final long serialVersionUID = 1L;
20   int arvo1 = 24;
21   transient int arvo2 = 10;
22   int arvo3 = 12;
23 }
```

Tekijänä Tommi Heikkinen (517749)

3 Tehtävä 3

3.1 A.kohta

Javassa voi käyttää instanceof-operaattoria rivin sisäisiin tutkimuksiin, jos oletetaan, että tehtävän rivit ovat toisensa periviä luokkia. Rivin välisiin tutkimuksiin käyttäisin jonkinlaista yksilöityä metodia Kuluttajille ja Tuottajille Tuotettavan luokan sisälle.

Metodi Humppa-luokan sisällä. Tarkistaa, kuunteleeko Kuluttaja humppaa.

```
1 <X extends Kuluttaja> boolean käykö(X x) {
2   if(x.kuluta() instanceof Humppa) {
3     return true;
4 }
```

Testi Main, joka tulostaa true molemmissa tapauksissa.

```
1 public class Main {
2   public static void main(String[] args) {
3     System.out.println(new Humppa() instanceof Musiikki);
4     System.out.println(new Humppa().käykö(new HumppaFani()));
5   }
6 }
```

Tekijänä Santeri Loitomaa (516587)

4 Tehtävä 4

Tehtävänä oli luoda geneerinen Hajautustaulu ja Linkitetty lista ja kuvata niiden toimintaa visuaalisesti OOMKitillä.

4.1 Hajautustaulu

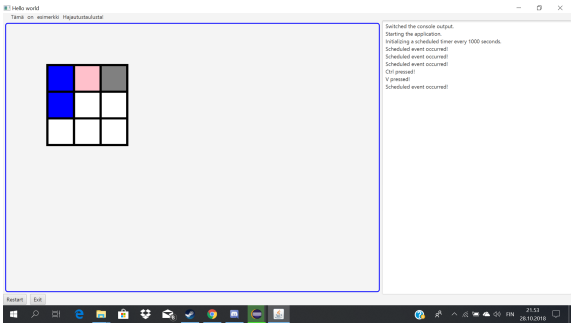
Hajautustaulu on luotu olio taulukko ArrayListina.(java ei anna tehdä geneerisiä taulukoita). Jossa samanlaiset oliot tallennetaan samalle sarakeelle.

```
1  import java.util.ArrayList;
2
3
4  public class Hajautustaulu<E> {
5          ArrayList<Object[]> list = new ArrayList<Object[]>();
6
7          public Hajautustaulu() {
8
9          }
10         public boolean lisää(E arvo) {
11                 for (Object[] taulu : list) {
12                         if(taulu[0].equals(arvo)) {
13                                 for(int i=1;i<taulu.length;i++) {
14                                         if(taulu[i] == null) {
15                                                 taulu[i]=arvo;
16                                                 return true;
17                                         }else {
18                                                 return false;
19                                         }
20                                 }
21                         }
```

```
22                 }
23                 Object[] t = {arvo,null,null};
24                 list.add(t);
25                 return true;
26
27         }
28         public boolean poista(E arvo) {
29                 for(Object[] taulu : list) {
30                         if(taulu[0].equals(arvo)) {
31                                 for(int i=0;i<taulu.length;i++) {
32                                         if(taulu[i+1] == null) {
33                                                 taulu[i]=null;
34                                                 if(i == 0) {
35                                                         list.remove(taulu);
36                                                 }
37                                                 return true;
38                                         }
39                                 }
40                         }
41                 }
42                 return false;
43         }
44
45
46 }
```

Ja visuaalisen kuvauksen tein omana metodinaan joka piirtää taulukon SimpleCanvasiin.

```
1  public void piirräHajautustaulu(GraphicsContext gc,Hajautustaulu<CoreColor> h) {
2                  for (int i=0;i<h.list.size();i++) {
3                                  for(int j=0;j<h.list.get(i).length;j++) {
4                                          if(h.list.get(i)[j] != null) {
5                                                  gc.setFill(((CoreColor)(h.list.get(i)[j]))
6                                                                  .toFXColor());
7                                          gc.fillRect(i*60+100,j*60+100,60,60);
8                                          }else {
9                                                  gc.setFill(Color.WHITE);
10                                                 gc.fillRect(i*60+100,j*60+100, 60, 60);
11                                         }
12                                         gc.strokeRect(i*60+100,j*60+100,60,60);
13                                 }
```



```
14                          }
15                  }
```

Kuva miltä hajautustaulu näyttäisi.

4.2 Linkitetty lista

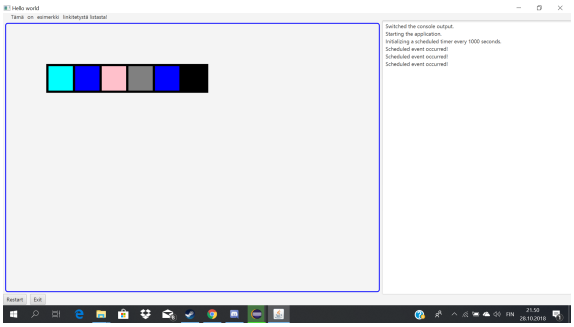
Toinen tehtävän osa oli linkitetty lista, joka oli mielästäni paljon helpompi toteutaa. Tein perus yksisuuntaisen linkitetyn listan jossa alkio tietää oman arvonsa ja mitä tulee sen jälkeen.

```
1
2   public class LinkitettyLista<E> {
3           E arvo;
4           LinkitettyLista<E> seuraava;
5
6           public LinkitettyLista(E arvo) {
7                   this.arvo = arvo;
8                   seuraava = null;
9           }
10
11          public void lisää(E arvo) {
12                  E tempt = this.arvo;
13                          if(seuraava != null) {
14                                  tempt = seuraava.arvo;
15                                  seuraava.lisää(arvo);                   }
16                          else {
17                                  seuraava = new LinkitettyLista<E>(arvo);
18                  }
19          }
```

```
20
21
22
23          public boolean poista(E arvo) {
24                  LinkitettyLista<E> edellinen = null;
25                  LinkitettyLista<E> temp = this;
26                  do {
27                          if(temp.arvo.equals(arvo)) {
28                                  if(edellinen != null) {
29                                          if(temp.seuraava != null) {
30                                                  edellinen.seuraava = temp.seuraava;
31                                                  return true;
32                                          }else {
33                                                  edellinen.seuraava = null;
34                                                  return true;
35                                          }
36                                  }
37                          }
38                          edellinen = temp;
39                          temp = edellinen.seuraava;
40                  }
41                  while(temp.seuraava != null);
42                  return false;
43          }
44          public void tulosta() {
45                  LinkitettyLista solmu = this;
46                  System.out.println(solmu.arvo.toString());
47
48                  while(solmu.seuraava != null) {
49                          solmu = solmu.seuraava;
50                          System.out.println(solmu.arvo.toString());
51                  }
52          }
53  }
```

Linkitetylle listalle on myös sen oma piirtämismetodi. Linkitetty lista piirretään ainasamaan kohtaan ja sen lopuun tulee musta neliö.

```
1
2   public void piirräLinkitettyLista(GraphicsContext gc,LinkitettyLista<CoreColor> lista) {
3                   LinkitettyLista solmu = lista;
4                   int x = 100;
```



```
5                int y = 100;
6                        //System.out.println(solmu.arvo.toString());
7                gc.setFill(((CoreColor)(solmu.arvo)).toFXColor());
8                gc.fillRect(x, y, 60, 60);
9                gc.strokeRect(x,y,60,60);
10               x = x+60;
11
12                       while(solmu.seuraava != null) {
13                               solmu = solmu.seuraava;
14                               gc.setFill(((CoreColor)(solmu.arvo)).toFXColor());
15                       gc.fillRect(x, y, 60, 60);
16                       gc.strokeRect(x,y,60,60);
17                       x = x+60;
18                       }
19                       gc.setFill(Color.BLACK);
20               gc.fillRect(x, y, 60, 60);
21               gc.strokeRect(x,y,60,60);
22
23           }
```

Kuva miltä linkitetty lista näyttää OOMKitillä.

Tekijänä Janina Kuosmanen (516580)