Turun yliopisto
University of Turku

Olio-ohjelmoinnin metodiikka
Viikkoharjoitustyö 4

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

Viikkoharjoitustyö 4
21. lokakuuta 2018
22 sivua

Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietotekniikka
Olio-ohjelmoinnin metodiikka

Sisältö

1 Tehtävä 1

Tehtävänä on vähentää OOMKitin DemoApp4:n riippuvuutta OOMKittiin.

1.1 Kohta (a)

Näitä metodeja voisi käyttää tehtävän kuvaamissa tilanteissa, kunhan on olemassa GraphicsContext canvas johon vaikuttaa. Demo käyttää SimpleCanvas-luokkaa, joka periaattessa vain helpottaa värien vaihtoa (setFill() ja setStroke()) ja muuta vastaavaa ylläpitoa.

Piirtävät luokat

```
1  // piirtää pallukan pisteeseen p värillä väri
2    public void piirräPallukka(Point p, Color väri, double koko, boolean täytetty,
3        GraphicsContext canvas) {
4      if (täytetty){
5        if (debuggaus) System.out.println("Piirretään täytetty " + (koko < 15 ?
6            "pieni pallukka" : (int)koko+" cm - säteinen ympyrä") + " pisteeseen " + p);
7        setFill(väri);
8        canvas.fillOval(p.x - koko / 2, p.y - koko / 2, koko, koko);
9      } else {
10       if (debuggaus) System.out.println("Piirretään ontto " + (koko < 15 ?
11          "pieni pallukka" : (int)koko+" cm - säteinen ympyrä") + " pisteeseen " + p);
12       canvas.setLineWidth(2);
13       setStroke(väri);
14       canvas.strokeOval(p.x - koko / 2, p.y - koko / 2, koko, koko);
15     }
16   }
```

```
17  // piirtää viivaan pisteestä alku pisteeseen lopppu värillä väri
18  public void piirräViiva(Point alku, Point loppu, Color väri, GraphicsContext canvas) {
19    if (debuggaus) System.out.println("Piirretään viiva pisteestä " + alku +
20        " pisteeseen " + loppu);
21    canvas.setLineWidth(2);
22    setStroke(väri);
23    canvas.strokeLine(alku.x, alku.y, loppu.x, loppu.y);
24  }
25  // piirtää viivaan pisteestä alku pisteeseen lopppu värillä väri
26  public void piirräLaatikko(Point vasenYlä, Point oikeaAla, Color väri, boolean täytetty,
27      GraphicsContext canvas) {
28    if (täytetty) {
29      if (debuggaus) System.out.println("Piirretään täytetty laatikko pisteiden " + vasenYlä
30          + " ja " + oikeaAla + " välille");
31      setFill(väri);
32      canvas.fillRect(vasenYlä.x, vasenYlä.y, oikeaAla.sub(vasenYlä).x,
33          oikeaAla.sub(vasenYlä).y);
34    } else {
35      if (debuggaus) System.out.println("Piirretään ontto laatikko pisteiden " + vasenYlä
36          + " ja " + oikeaAla + " välille");
37      canvas.setLineWidth(2);
38      setStroke(väri);
39      canvas.strokeRect(vasenYlä.x, vasenYlä.y, oikeaAla.sub(vasenYlä).x,
40          oikeaAla.sub(vasenYlä).y);
41    }
42  }
43  public void piirräKolmio(Point keski, double koko, GraphicsContext canvas) {
44    Point piste1 = kaarelta(keski, Math.PI * 1 / 6, koko / 2 / 37 * 50);
45    Point piste2 = kaarelta(keski, Math.PI * 5 / 6, koko / 2 / 37 * 50);
46    Point piste3 = kaarelta(keski, Math.PI * 9 / 6, koko / 2 / 37 * 50);
47
48    int siirtymäY = keski.y - (piste2.y + piste3.y) / 2;
49    piste1 = piste1.add(0, siirtymäY);
50    piste2 = piste2.add(0, siirtymäY);
51    piste3 = piste3.add(0, siirtymäY);
52
53    canvas.fillPolygon(new double[]{piste1.x, piste2.x, piste3.x}, new double[]{piste1.y,
54        piste2.y, piste3.y}, 3);
55  }
56  public void updateContent() {
57    if (kulma > 0.5) debuggaus = false;
```

```
58    kulma += 0.1;
59  }
60  // palauttaa pistettä keski ympäröivän säde-säteisen ympyrän kaarelta pisteen kulmasta
61  protected static Point kaarelta(Point keski, double kaariKulma, double säde) {
62    return keski.add(
63        (int) (säde * Math.cos(kaariKulma)),
64        (int) (säde * Math.sin(kaariKulma))
65    );
66  }
67  protected static String fyysisesti(Point p) {
68    return "kohtaan, joka on " + p.x + " cm kohteen vasemmasta reunasta ja " + p.y +
69        " cm yläreunasta.";
70  }
71  protected void drawBackgroundContent(GraphicsContext canvas) {
72    setFill(CoreColor.Black);
73    canvas.fillRect(0, 0, getWidth(), getHeight());
74  }
75  protected void drawForegroundContent(GraphicsContext canvas) {
76    Point keski1 = new Point(100, 100);
77    Point keski2 = new Point(300, 150);
78    Point keski3 = new Point(150, 350);
79
80    {
81      Point keski = keski1;
82      int kulmia = 5;
83      int kulmaHyppy = 2;
84      double kulmaLisäys = kulma;
85      double koko = 32;
86
87      if (debuggaus) System.out.println("Seuraavilla ohjeilla piirretään 5-kulmio kohtaan "
88          + fyysisesti(keski));
89      for (double i = 0; i < kulmia; i++) {
90        Point p1 = kaarelta(keski, kulmaLisäys + i / kulmia * 2 * Math.PI, koko);
91        Point p2 = kaarelta(keski, kulmaLisäys + (i + kulmaHyppy) / kulmia * 2 * Math.PI,
92            koko);
93        piirräViiva(p1, p2, CoreColor.Red, canvas);
94        piirräPallukka(p1, CoreColor.LightRed, 10, true, canvas);
95        piirräPallukka(p2, CoreColor.LightRed, 10, true, canvas);
96      }
97    }
98    {
```

```
99        Point keski = keski2;
100       int kulmia = 7;
101       int kulmaHyppy = 1;
102       double kulmaLisäys = -kulma / 2;
103       double koko = 80;
104
105       if (debuggaus) System.out.println("Seuraavilla␣ohjeilla␣piirretään␣7-kulmio␣kohtaan␣"
106           + fyysisesti(keski));
107       for (double i = 0; i < kulmia; i++) {
108         Point p1 = kaarelta(keski, kulmaLisäys + i / kulmia * 2 * Math.PI, koko);
109         Point p2 = kaarelta(keski, kulmaLisäys + (i + kulmaHyppy) / kulmia * 2 * Math.PI,
110             koko);
111         piirräViiva(p1, p2, CoreColor.Red, canvas);
112         piirräPallukka(p1, CoreColor.LightRed, 10, true, canvas);
113         piirräPallukka(p2, CoreColor.LightRed, 10, true, canvas);
114       }
115     }
116     {
117       Point sijainti = keski3.add((int) (Math.cos(kulma / 2) * 180 + 100),
118           (int) (Math.sin(kulma / 2) * 40));
119       int koko = 80;
120       Color väri = CoreColor.Orange;
121
122       if (debuggaus) System.out.println("Seuraavilla␣ohjeilla␣piirretään"+
123           "␣'Clavicula␣Nox'␣kohtaan␣" + fyysisesti(sijainti));
124       piirräPallukka(sijainti, väri, koko, false, canvas);
125       piirräPallukka(sijainti.add(0, koko / 2), väri, koko * 2 / 3, false, canvas);
126       piirräLaatikko(sijainti.add(-koko / 2, -koko / 2 - 5), sijainti.add(koko / 2, 0),
127           CoreColor.Black, true, canvas);
128       piirräViiva(sijainti.add(0, koko * 2), sijainti.add(0, -koko / 4), väri, canvas);
129     }
130   }
```

1.2 Kohta (b)

Näistä metodeista ohjeiden tulostus ei olisi vaikeaa, sillä se tekee sen jo dubuggaus-booleanin ollessa true. Tietysti joitain tekstejä tulisi muokata "ohjemaisemmiksi"mutta muuten en näe ongelmaa.

Tekijänä Santeri Loitomaa (516587)

2 Tehtävä 2

2.1 a-kohta

Alla valmiskoodista otettu ympyrän piirämismetodi ja sen määritys.

```
/**
* Piirtää tietyn värisen joko kokonaan väritetyn tai onton ympyrän, siten että annettu
  piste on keskipiste.
* @.pre koko>=0.0
* @.post Piirtää ympyrän annetulla värillä, koolla, sijainnilla (keskipisteenä) ja joko
  kokonaan väritettynä tai
* pelkillä reunoilla.
*/
  public void piirräPallukka(Point p, Color väri, double koko, boolean täytetty,
  GraphicsContext canvas) {
        if (täytetty){
            if (debuggaus) System.out.println("Piirretään täytetty " + (koko < 15 ?
            "pieni pallukka" : (int)koko+" cm - säteinen ympyrä") + " pisteeseen " + p);
            setFill(väri);
            canvas.fillOval(p.x - koko / 2, p.y - koko / 2, koko, koko);
        } else {
            if (debuggaus) System.out.println("Piirretään ontto " + (koko < 15 ?
            "pieni pallukka" : (int)koko+" cm - säteinen ympyrä") + " pisteeseen " + p);
            canvas.setLineWidth(2);
            setStroke(väri);
            canvas.strokeOval(p.x - koko / 2, p.y - koko / 2, koko, koko);
        }
    }
```

Alla valmiskoodista otettu neliön piirtämismetodi ja sen määrittely. Muutin sen toimintaa siten että se ottaa vastaan keskipisteen ylä ja alakulman sijasta ja koon, niin-

kuin tehtävänannossa vaadittiin.

```
/**
* Piirtää tietyn värisen joko kokonaan väritetyn tai onton neliön, siten että annettu
  piste on keskipiste.
* @.pre koko>=0.0
* @.post Piirtää neliön annetulla värillä, koolla, sijainnilla (keskipisteenä) ja joko
  kokonaan väritettynä tai
* pelkillä reunoilla.
*/
    public void piirräLaatikko(Point p, Double koko, Color väri, boolean täytetty,
    GraphicsContext canvas) {
        if (täytetty) {
            if (debuggaus) System.out.println("Piirretään täytetty laatikko pisteiden "
            + (int)(p.x - koko/2) + "," + (int)(p.y - koko/2) + " ja " +
            (int)(p.x + koko/2) + "," + (int)(p.y + koko/2) + " välille");
            setFill(väri);
            canvas.fillRect(p.x - koko/2, p.y - koko/2, koko, koko);
        } else {
            if (debuggaus) System.out.println("Piirretään ontto laatikko pisteiden " +
            (int)(p.x - koko/2) + "," + (int)(p.y - koko/2) + " ja " +
            (int)(p.x + koko/2) + "," + (int)(p.y + koko/2) + " välille");
            canvas.setLineWidth(2);
            setStroke(väri);
            canvas.strokeRect(p.x - koko/2, p.y - koko/2, koko, koko);
        }
    }
```

Alla valmiskoodista otettu kolmionpiirtämismetodi ja sen määrittely, koodi ei täyttänyt tehtävänannon vaatimuksia joten päivitin sen ottamaan vastaan etukäteen määritetyn värin ja tiedon siitä onko se ontto voi väritetty.

```
/**
* Piirtää tietyn värisen joko kokonaan väritetyn tai onton kolmion, siten että annettu
piste on keskipiste.
* @.pre koko>=0.0
* @.post Piirtää kolmion annetulla värillä, koolla, sijainnilla (keskipisteenä) ja
  joko kokonaan väritettynä tai
* pelkillä reunoilla.
*/
    public void piirräKolmio(Point keski, Color väri, boolean täytetty, double koko,
    GraphicsContext canvas) {
```

```
11          Point piste1 = kaarelta(keski, Math.PI * 1 / 6, koko / 2 / 37 * 50);
12          Point piste2 = kaarelta(keski, Math.PI * 5 / 6, koko / 2 / 37 * 50);
13          Point piste3 = kaarelta(keski, Math.PI * 9 / 6, koko / 2 / 37 * 50);
14          int siirtymäY = keski.y - (piste2.y + piste3.y) / 2;
15          piste1 = piste1.add(0, siirtymäY);
16          piste2 = piste2.add(0, siirtymäY);
17          piste3 = piste3.add(0, siirtymäY);
18      if (täytetty){
19        setFill(väri);
20        canvas.fillPolygon(new double[]{piste1.x, piste2.x, piste3.x}, new
21        double[]{piste1.y, piste2.y, piste3.y}, 3);
22      } else{
23        canvas.setLineWidth(2);
24              setStroke(väri);
25              canvas.strokePolygon(new double[]{piste1.x, piste2.x, piste3.x},
26              new double[]{piste1.y, piste2.y, piste3.y}, 3);
27      }
28
29      }
```

2.2 b-kohta

Perusasetelman kuviot saadaan aikaan sijoittamalla piirrä-metodit piirtelypinann draw-foregroundin sisään.

```
1       protected void drawForegroundContent(GraphicsContext canvas) {
2           {
3               Point keski = new Point(100, 100);
4               double koko = 32;
5               piirräLaatikko(keski, koko, CoreColor.Red, true, canvas);
6           }
7           {
8               Point keski = new Point(300, 150);
9               double koko = 80;
10              piirräKolmio(keski, CoreColor.Green, true, koko, canvas);
11          }
12          {
13              Point keski = new Point(150, 350);
14              int koko = 80;
```

```
15              piirräPallukka(keski, CoreColor.Yellow, koko, true, canvas);
16          }
17      }
```

2.3 c-kohta

Asetelma laittaa sisäkkäiset kuviot, joista sisempi on puolet pienempi ennalta määrättyihin sijainteihin. Kunkin kuvion lokaatio määritellään randomilla, hyväksikäyttäe switchia, varmistaen ettei sama ole missään kahdesti.

```
1  Random num = new Random();
2        int x = 0;
3        int y = 0;
4        double koko = 30;
5        int kohta = num.nextInt(9);
6        ArrayList käytetyt = new ArrayList();
7        CoreColor color = CoreColor.Red;
8        for (int i = 0; i<3;i++) {
9          x = x + 50;
10         for (int j = 0; j<3;j++) {
11           y = y + 50;
12           Point p = new Point(x,y);
13           while (käytetyt.contains(kohta)) {
14             kohta = num.nextInt(9);
15           }
16           käytetyt.add(kohta);
17           switch(kohta){
18           case 0: piirräKolmio(p, color, false, koko, canvas);
19           piirräKolmio(p, color, true, koko/2, canvas);
20           break;
21           case 1: piirräKolmio(p, color, false, koko, canvas);
22           piirräPallukka(p, color, koko/2, true, canvas);
23           break;
24           case 2: piirräKolmio(p, color, false, koko, canvas);
25           piirräLaatikko(p, koko/2, color, true, canvas);
26           break;
27           case 3: piirräPallukka(p, color, koko, false, canvas);
28           piirräKolmio(p, color, true, koko/2, canvas);
29           break;
```

```
30          case 4: piirräPallukka(p, color, koko, false, canvas);
31          piirräPallukka(p, color, koko/2, true, canvas);
32          break;
33          case 5: piirräPallukka(p, color, koko, false, canvas);
34          piirräLaatikko(p, koko/2, color, true, canvas);
35          break;
36          case 6: piirräLaatikko(p, koko, color, false, canvas);
37          piirräKolmio(p, color, true, koko/2, canvas);
38          break;
39          case 7: piirräLaatikko(p, koko, color, false, canvas);
40          piirräPallukka(p, color, koko/2, true, canvas);
41          break;
42          case 8: piirräLaatikko(p, koko, color, false, canvas);
43          piirräLaatikko(p, koko/2, color, true, canvas);
44          break;
45        }
46        }
47        y = 0;
48      }
```

Ratkaisussa ei tarvinnut käyttää periytymistä aiempia enempää, joskin PiirtelyPinta, jolla kuviot piirretään perii luokan SimpleCanvas.

2.4 d-kohta

Yksi tapa ratkaista tilanne on luoda uudet metodit KolmioNeliö ja Neliökolmio yms ja laittaa ne kutsumaan jo olemassaolevia metodeja, jolloin voidaan hyväksikäyttää jo olemassaolevia materiaaleja. esim.

```
1 KolmioNeliö(Point p, double koko, Color color1, Color color2, boolean
2 filled1, boolean filled2, GraphicsContext canvas)
```

Elegantimpi ratkaisu olisi uudenmallinen metodi, kenties jo olemassaolevista muokattu tai uusi, piirräMuoto, joka ottaisi vastaan halutun muodon (tyypit tulisi olla määritetty etukäteen), kaikki muodolle tarpeelliset tiedot ja myös mahdollisen sisämuodon tiedot, jotka nulleiksi jätettyinä tarkoittaisivat ettei sisämuotoa tule.

Tekijänä Tommi Heikkinen (517749)

3 Tehtävä 3:

3.1 yhdistä() -metodin palautuvat viittaukset

Muodostetaan ensin tehtävän tarvitsema oliohierarkia kirjoittamalla luokat muotoon:

```
1 class Eucarya {}
2 class Animalia extends Eucarya {}
3 class Chordata extends Animalia {}
4 ja niin edelleen
```

Tuloksena saadaan oliohierarkia (kuva 3.1).

Yhdistä() -metodi voidaan toteuttaa ylikuormitettuna käyttämällä apuna metodia getLevel(), joka antaa olion lajitason kokonaislukuna niin, että Eucarya -luokan oliot palauttavat luvun 0, Animalia 1:sen jne... getLevel() ja yhdistä() -metodit esitetään seuraavaksi koostetusti niin, että luokkahierarkia jätetään esittämättä.

```
1  /**
2   * Level kuvaa kuinka korkealla eliö on lajistotasolla
3   * @return 0 ellei eliö ole korkeampaa luokkaa
4   */
5  public int getLevel() {
6          return 0;
7  }
8
9  /**
10  * Rekursiivinen määrittely korkeammille luokille
```

Kuva 3.1: Luokkakaavio lajiston periytyvyydestä

```
 */
@Override
public int getLevel() {
        return super.getLevel()+1;
}

/**
 *
 * @param input Saman lajitason eliö
 * @return Taulukko muodossa {this,eliö}, jos eliö on samalla tasolla.
 */
public Eucarya[] yhdistä(Eucarya eliö) {
        if (this.getLevel() == eliö.getLevel()) {
                return new Eucarya[] {this,eliö};
        }
        return new Eucarya[] {};
}

/**
 *
 * @param eliöTaulukko
 * @return Taulukko, johon on lisätty this. Palauttaa tyhjän taulukon jos eliöt
 * eivät ole samalla tasolla.
 */
public Eucarya[] yhdistä(Eucarya... eliöTaulukko) {
        Eucarya[] result = new Eucarya[eliöTaulukko.length + 1];
        result[0] = this;

        for (int i = 0; i < eliöTaulukko.length; i++) {
                if (this.getLevel() != eliöTaulukko[i].getLevel()) {
                        return new Eucarya[] {};
                }
                result[i+1] = eliöTaulukko[i];
        }
        return result;
}
```

Testausta varten käytetään tulosta() -metodia joka selittää lyhyesti eliöiden lajitason.

```
public static void tulosta(String otsikko, Eucarya... taulukko) {
        System.out.println(otsikko);
```

```
3         for (int i = 0; i < taulukko.length; i++) {
4                 System.out.println("Eliön lajitaso " + i + ": " + taulukko[i].getClass());
5         }
6 }
```

Jolloin voidaan suorittaa tehtävässä kysytty algoritmi:

```
1  Eucarya[] taulukko = maunkiharapyrstö.yhdistä(meksikonkarhu);
2  tulosta("maunkiharapyrstön ja meksikonkarhun yhdistelmä",taulukko);
3
4  taulukko = kapinleijona.yhdistä(meksikonkarhu);
5  tulosta("kapinleijonan ja meksikonkarhun yhdistelmä",taulukko);
6
7  taulukko = kapinleijona.yhdistä(maunkiharapyrstö.yhdistä(meksikonkarhu));
8  tulosta("meksikonkarhun, maunkiharapyrstön ja kapinleijonan yhdistelmä",taulukko);
9
10 //maunkiharapyrstön ja meksikonkarhun yhdistelmä
11 //Eliön lajitaso 0: class Bishop
12 //Eliön lajitaso 1: class Arctos
13 //kapinleijonan ja meksikonkarhun yhdistelmä
14 //Eliön lajitaso 0: class Leo
15 //Eliön lajitaso 1: class Arctos
16 //meksikonkarhun, maunkiharapyrstön ja kapinleijonan yhdistelmä
17 //Eliön lajitaso 0: class Leo
18 //Eliön lajitaso 1: class Bishop
19 //Eliön lajitaso 2: class Arctos
```

Huomion arvoista on, että taulukkojen yhdistäminen tapahtuu niin, että jokainen eliö tulee olla samalla lajitasolla. Jos ylläolevan koodin rivillä 7 tapahtuisi tilanne, jossa sisempien sulkujen sisällä yhdistäminen ei onnistuisi, niin tulostuisi pelkästään taulukko, jossa olisi kapinleijona. Olisi siis turvallisempaa toteuttaa rivi 7 käyttäen yhdistä() -metodia seuraavalla tavalla:

```
1 taulukko = kapinleijona.yhdistä(new Eucarya[] {maunkiharapyrstö,meksikonkarhu});
```

Jolloin taulukko olisi varmasti tyhjä, jos jokin eliöistä on mahdotonta yhdistää kapinleijonan kanssa.

3.2 Olio-taulukon varianssi

Jos ajetaan tehtävänohjeistuksen mukainen koodi:

```
1 Olio[] oliot = new Olio[] { new Olio() };
2 oliot[0] = new Olio(); // OK
3 Object[] objektit = oliot;
4 objektit[0] = new Olio(); // OK
5 objektit[0] = new Object();
6 System.out.println("Onnistui");
```

Jossa Olio -luokka on Object -luokan alaluokka, koska kaikki luokat, joilla on olio-esiintymiä ovat Object -luokan alaluokkia. Listan luomisen jälkeen luodaan viittaus objektit oliot-taulukkoon. Viittaus ei kumminkaan pysty muuttamaan taulukkoa ylemmän luokan taulukoksi vaan tämä tulisi konstruoida uudelleen. Ohjelman ajo siis keskeytyy ajonaikaiseen virheeseen jossa objektit viittauksella yritetään ylikirjoittaa Object -luokan olio taulukkoon, jossa voi olla vain Olio -luokkaa tai tämän aliluokkia, jolloin luokkaviittauksen invarianssi ei päde.

Koodi voidaan korjata toimivaksi kirjoittamalla rivi 3 seuraavalla tavalla:

```
1 Object[] objektit = new Object[]{oliot[0]};
```

Jolloin objektit viittaa uuteen taulukkoon, jossa voi olla mitä tahansa olioita.

3.3 Solmu -luokan periytymisongelmat

Ensimmäisenä ongelmana voisi olla solmujen väliset toimenpiteet. Voisi olla epäselvää, että onko erikoistettujen solmujen equals -metodi sama, kuin Solmu- luokan solmujen. Siinä tapauksessa equals -metodin palautus erikois- ja normaalisolmun välillä riippuisi siitä kummalta kysytään. Tämä voi olla ongelmallista, kun halutaan vertailla mitkä solmut ovat oikeasti samoja.

Jos yritetään mallintaa vaikka tiekarttaa jota kuvataan suunnattuna graafina, niin solmujen erikoistaminen siinä tapauksessa, että kaupungin läpi ajaminen vaihtelisi

riippuen siitä löytyykö kehätietä vai tuleeko autoilijan ajaa keskustaan. Tätä voitaisiin kuvata solmun ominaisuutena, mutta silloin tulisi kirjoittaa paljon suunnattuja graafeja uudelleen, koska matkan laskemiseen käytettävä algoritmi tulisi toimia eri tavalla.

Voitaisiinkin miettiä voisiko kaupunkien läpi ajaessa parempi kuvata kaarilla ja merkata niihin, että matka on pidempi. Se tarkoittaisi sitä, että jokainen kaari joka on vaikka yhteydessä Varkauteen tulisi painottaa suuremmaksi. Nyt matkan laskemisalgoritmia ei tarvitsisi kirjoittaa uudestaan.

3.4 Periytymisratkaisujen arvioimista

3.4.1 PeliObjekti matopelissä

Jos haluttaisiin tehdä PeliObjekti, jossa olisi yhtenäisenä piirteenä ainoastaan se, että niillä kaikilla olisi olemassa jotkin koordinaatit, jotka kertovat missä kohtaa pelialueella ne ovat. Tämä olisi huono ratkaisu, koska erikoistumista ei tapahdu tarpeeksi ja eteen tulisi peliä laajentaessa jossain vaiheessa invarianssiongelmia.

Olisi järkevämpää tehdä rajapinta “koordinaatillinen”, jolla merkattaisiin kaikki luokat joilla olisi koordinaatit. Rajapintaan voisi tehdä säännön, että jokainen tämän rajapinnan luokka toteuttaa erilaiset koordinaatteihin liittyvät metodit. Jos PeliObjekti vaikka koostuisi useasta osasta, niin annaKoordinaatit() -metodi voisi palauttaa esimerkiksi näiden usean osan koordinaattien vektori-keskiarvon. Näin useasta osasta koostuva peliobjekti voisi säilyttää oman abstraktionsa useampiosaisena systeeminä.

3.4.2 Banaani-luokka toteuttaa rajapinnan Painovoima

Jos Banaani-luokka toteuttaa rajapinnan painovoima, niin voidaan kohdata ongelmia pelimoottorin kanssa. Fysiikan lakien mukaan painovoima, on enemmänkin massallisten kappaleiden väliseen vuorovaikutukseen liittyvä ominaisuus. Tämä ratkaisu rajaa pelimekaniikan tilanteeseen, jossa painovoima voi tapahtua vain alaspäin. Lisäksi

Painovoima rajapinta tekisi pelimoottorin kannalta hyvin vähän asioita: lähinnä määräisi, että tähään olioon vaikuttaisi alaspäin suuntautuva kiihtyvyys.

Koska kaikki kappaleet luonnossa kiihtyvät yhtä nopeasti alaspäin, niin mielekkäämpää olisi vain tehdä luokka Kappale, johon olisi sisäänkirjoitettuna painovoiman aiheuttama kiihtyvyys vektorina.

3.4.3 Ympyrä perii ellipsin

Ympyrä on ellipsin erikoistapaus, jossa säteen suuruus pysyy vakiona. Jos halutaan mallintaa ympyrää matemaattisesti, niin tämä voi olla hyvä idea. Toisaalta näin tehdessä laskennallinen rutiini hidastuu, koska jokaisen ympyrän kuvaaminen ellipsinä on hitaampaa, kuin että tehtäisiin oma Ympyrä-luokka. Omassa Ympyrä -luokassa voitaisiin laskentaa nopeuttaa ja erikoistaa.

3.4.4 Ellipsi perii ympyrän

Ellipsin periessä ympyrän erikoistaminen tapahtuu toisin. Tässä tilanteessa ympyrän käsitettä yleistetään lisäämällä siihen ominaisuuksia niin, että ellipsille ominaiset piirteet voidaan mallintaa. Tämä on hyvä idea, koska ellipsillä on paljon ominaisuuksia, jotka ovat samankaltaisia ympyrän kanssa: esim. keskeissäde ja keskipiste.

3.4.5 Neperin luku peritään luokasta Number

Luonnon vakioiden tai matemaattisten tunnuslukujen tallentaminen kannattaa tehdä joko omaksi singulaariseksi olioksi tai Number -luokan staattiseksi ominaisuudeksi. Neper -olioita ei tarvita laskutoimituksissa useita kappaleita, joten tällainen periminen on aivan turhaa.

3.4.6 Stack perii Vector -luokan

Stack -luokalla halutaan lisätä Vector -luokkaan ominaisuus poistaa viimeinen jäsen tai lisätä uusi jäsen ensimmäiseksi listassa. Vector -luokassa jäseniä poistetaan tietyistä indekseistä ja uusi jäsen lisätään aina viimeiseksi. Koska Stack -luokassa halutaan käyttää Vector luokan toiminnallisuutta periminen on perusteltua.

Stack -luokka voisi periä myös jonkin muun List -luokan, jolloin pystyttäisiin muodostamaan sama toiminnallisuus. Tähän voitaisiin päätyä, jos Stack -luokan käytännöt eivät muistuttaisi niin paljon Vector -luokkaa.

4 Tehtävä 4

Tehtävän tarkoituksena oli demostroida javan metodikutsurakenteen toimintaaa ja mahdollisia rajoitteita. Tehtävässä on luotu maailma jossa on Hiiri, Kissa ja Hipsteri olioita. Kun kaksi oliota tapaavat tulostuu näytölle testi joka määräytyy olioiden luokan mukaan.

4.1 Kuormitus(a)

Loin jokaiselle mahdolliselle olion yhdistelmälle kohtaa-metodin. Kaikilla metodeilla on täysin sama nimi ja vain parametrit muuttuvat.

```
1  public class Hipsteri extends Olio {
2          public Hipsteri() {}
3
4          public void kohtaa(Hipsteri h) {
5                  System.out.println("Hipsteri␣kohtaa␣hipsterin");
6          }
7      public void kohtaa(Kissa k) {
8          System.out.println("Hipsteri␣kohtaa␣eläimen");
9          }
10     public void kohtaa(Hiiri h) {
11         System.out.println("Hipsteri␣kohtaa␣eläimen");
12         }
13 }
14 public class Kissa extends Olio{
15         public Kissa() {}
16
17         public void kohtaa(Hipsteri h) {
18                 System.out.println("Hipsteri␣kohtaa␣eläimen");
```

```
19          }
20          public void kohtaa(Kissa k) {
21                  System.out.println("Kissoja syntyy mahdollisesti nyt");
22          }
23          public void kohtaa(Hiiri h) {
24                  System.out.println("Hiiri syödään");
25          }
26  }
27  public class Hiiri extends Olio {
28          public Hiiri() {}
29
30          public void kohtaa(Hipsteri h) {
31                  System.out.println("Hipsteri kohtaa eläimen");
32          }
33          public void kohtaa(Hiiri h) {
34                  System.out.println("Hiiriä syntyy mahdollisesti nyt");
35          }
36          public void kohtaa(Kissa k) {
37                  System.out.println("Hiiri syödään");
38          }
39  }
```

Ja sitten yksin kertainen tarkistus että koodi toimii niin kuin ajattelin.

```
1   public static void main(String[]args) {
2           Hipsteri harri = new Hipsteri();
3           Hipsteri tiina = new Hipsteri();
4           Hiiri pip = new Hiiri();
5           Kissa misu = new Kissa();
6           harri.kohtaa(misu);
7           pip.kohtaa(harri);
8           misu.kohtaa(pip);
9           tiina.kohtaa(harri);
10        }
```

4.2 Yliluokan käyttö(b)

Seuraavaksi laitoin olioita taulukkoon ja valitsin kahdesta taulukon indeksista olion ja laitoin ne kohtaamaan.

```
1  public class Maailma {
2          public Olio[] map;
3
4      public Maailma() {
5          map = new Olio[10];
6      }
7      public void lisää(Olio o,int i) {
8          map[i]=o;
9      }
10     public void poista(int i) {
11         map[i]=null;
12     }
13     public Olio get(int i) {
14         return map[i];
15     }
16      public static void main(String[]args) {
17
18         Maailma m = new Maailma();
19         Hipsteri harri = new Hipsteri();
20         Hipsteri tiina = new Hipsteri();
21         Hiiri pip = new Hiiri();
22         Hiiri pup = new Hiiri();
23         Kissa misu = new Kissa();
24         Kissa kisu = new Kissa();
25
26         m.lisää(harri, 0);
27         m.lisää(tiina, 1);
28         m.lisää(pip, 2);
29         m.lisää(misu, 3);
30         m.lisää(pup, 4);
31         m.lisää(kisu, 5);
32
33         m.get(0).kohtaa(m.get(1));
34         m.get(0).kohtaa(m.get(2));
35         m.get(0).kohtaa(m.get(3));
36         m.get(2).kohtaa(m.get(4));
37         m.get(2).kohtaa(m.get(3));
38         m.get(3).kohtaa(m.get(3));
39         m.get(3).kohtaa(m.get(1));
40     }
```

Tässä tilanteessa kohtaa-metodit eivät toimi sellaisenaan sillä parametri on olio-tyyppisenä. Tässä tilanteessa voitaisiin dynaamisesti muuttaa parametrin tyyppiä, mutta tämä toimii vain jos parametrin tyyppi on ennestään tiedossa. Itse tekisin yläluokkaan yhden metodin joka tarkistaa molempien olioden tyypit.

```
1  public void kohtaa(Olio o) {
2                  if(this instanceof Kissa) {
3                          if(o instanceof Kissa) {
4                                  System.out.println("Kissoja syntyy mahdollisesti nyt");
5                          }else if(o instanceof Hiiri) {
6                                  System.out.println("Kissa syö hiiren");
7                  }else if(o instanceof Hipsteri) {
8                          System.out.println("Hipsteri kohtaa eläimen");
9              }
10         } if(this instanceof Hiiri) {
11                 if(o instanceof Kissa) {
12                         System.out.println("Kissa syö hiiren");
13                 }else if(o instanceof Hiiri) {
14                         System.out.println("Hiiriä syntyy mahdollisesti nyt");
15         }else if(o instanceof Hipsteri) {
16                 System.out.println("Hipsteri kohtaa eläimen");
17 }
18         } if(this instanceof Hipsteri) {
19                 if(o instanceof Hipsteri) {
20                         System.out.println("Hipsteri kohtaa hipsterin");
21                 }else {
22                         System.out.println("Hipsteri kohtaa eläimen");
23                 }
24
25                 }
26 }
```

4.3 Vierailija(c)

```
1  interface Visitor{
2          void visit(Kissa k,Olio o);
3          void visit(Hiiri h,Olio o);
4          void visit(Hipsteri h,Olio o);
```

```
5
6  }
7
8  public class OlioVisitor implements Visitor {
9
10         @Override
11         public void visit(Kissa k, Olio o) {
12                         if(o instanceof Kissa) {
13                                 System.out.println("Kissoja␣syntyy␣mahdollisesti␣nyt");
14                         }else if(o instanceof Hiiri) {
15                                 System.out.println("Kissa␣syö␣hiiren");
16                 }else if(o instanceof Hipsteri) {
17                         System.out.println("Hipsteri␣kohtaa␣eläimen");
18             }
19         }
20
21         @Override
22         public void visit(Hiiri h, Olio o) {
23                 if(o instanceof Kissa) {
24                         System.out.println("Kissa␣syö␣hiiren");
25                 }else if(o instanceof Hiiri) {
26                         System.out.println("Hiiriä␣syntyy␣mahdollisesti␣nyt");
27         }else if(o instanceof Hipsteri) {
28                 System.out.println("Hipsteri␣kohtaa␣eläimen");
29     }
30
31         }
32
33         @Override
34         public void visit(Hipsteri h, Olio o) {
35                 if(o instanceof Kissa) {
36                         System.out.println("Hipsteri␣kohtaa␣eläimen");
37                 }else if(o instanceof Hiiri) {
38                         System.out.println("Hipsteri␣kohtaa␣eläimen");
39         }else if(o instanceof Hipsteri) {
40                 System.out.println("Hipsteri␣kohtaa␣hipsterin");
41     }
42
43         }
44
45 }
```

ja jokaiselle oliolle vastaanottaja metodi

```
1 public void Accept(OlioVisitor visitor,Olio o) {
2         visitor.visit(this,o);
3         }
```

Tekijänä Janina Kuosmanen (516580)