T-106.1250 Informaatioverkostojen ohjelmointikurssi
T-106.2001 Informaatioverkostojen studio 1

Tentti 8. tammikuuta 2009

Vastaa kaikkiin tehtäviin. Kirjoita jokaiseen vastauspaperiin vähintään päivämäärä, nimesi, opiskelijanumerosi ja allekirjoituksesi.

1. **Kysymykset (8p)**

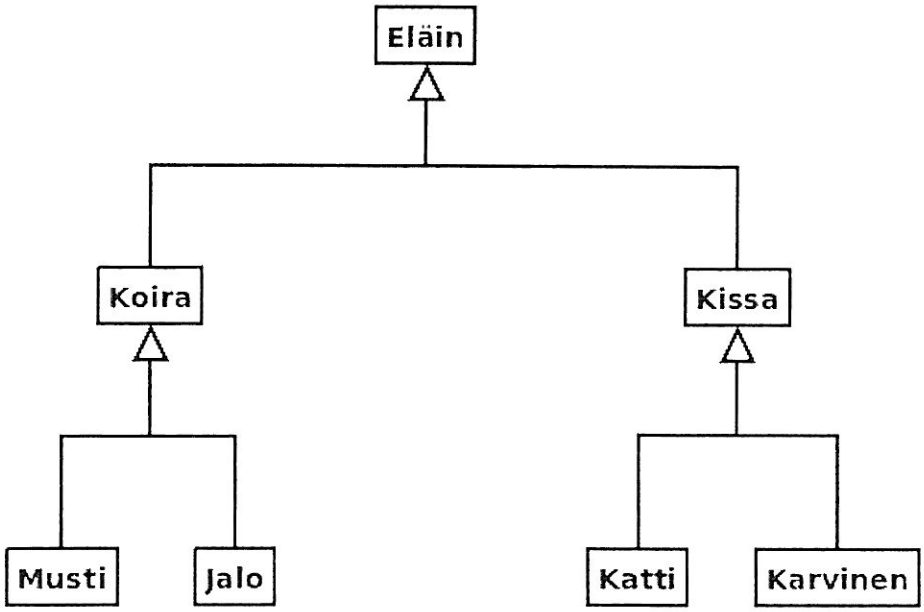
   Vastaa seuraaviin kysymyksiin. Sopiva vastauksen pituus on noin yksi kappale.

a) Määrittele lyhyesti *olion* ja *luokan* käsitteet Java-ohjelmoinnissa.
b) Mikä merkitys määreellä `abstract` on Java-metodin määrittelyssä?
c) Mitä eroa on *tarkistamattomilla* poikkeuksilla (engl. *unchecked exceptions*) verrattuna *tarkistettuihin* poikkeuksiin (engl. *checked exceptions*)?
d) Mitä on *rekursio*? Anna esimerkki ongelmasta, jonka ratkaiseminen rekursion avulla on järkevää. Itse ratkaisua ei tarvitse esittää.

2. **Väittämät (8p)**

Perustele, ovatko seuraavat väittämät totta vai eivät. Pisteet saa oikeasta perustelusta.

a) Ruokalan jonon kuvaamiseen sopii tietorakenteista paremmin lista kuin taulukko.
b) Tämän kaavion kuvaama luokkarakenne on järkevä.



Valkopäinen musta nuoli tarkoittaa kaaviossa perintää – esimerkiksi `Koira` on `Eläin`-luokan alaluokka.

c) Olkoon määritelty luokka `Joukkue`. Tällöin seuraavat ohjelmat tuottavat ajettaessa saman tulosteen.

```
1  public class Testi {
2
3      public static void main(String[] args) {
4          Joukkue joukkue = new Joukkue("Veikkolan veikot");
5          System.out.println(joukkue);
6      }
7
8  }
```

```
1  public class Testi {
2
3      public static void main(String[] args) {
4          Joukkue joukkue = new Joukkue("Veikkolan veikot");
5          System.out.println(joukkue.toString());
6      }
7
8  }
```

d) Seuraavan luokan kääntäminen Java-kääntäjällä aiheuttaa käännösvirheen.

```
1  public class HeiMaailma {
2
3      public static String tervehdi(String tervehdys1) {
4          System.out.println(tervehdys);
5          return tervehdys;
6      }
7
8      public static void tervehdi(String tervehdys2){
9          System.out.println(tervehdys);
10     }
11
12     public static void main(String[] args) {
13         HeiMaailma.tervehdi("Hei maailma!");
14     }
15
16 }
```

3. **Mitä tämä koodi tulostaa? (8p)**
Kirjoita näkyviin tuloste, jonka `Tulostaja`-luokan ajaminen aiheuttaa.

```
1   public class Tulostaja {
2
3       protected boolean totuus;
4
5       public Tulostaja(){
6           System.out.println("Tulostajan konstruktori");
7           totuus = true;
8       }
9
10      public static void main(String[] args) {
11          Tulostaja t1 = new Oliotulostaja();
12          System.out.println("-----");
13          Tulostaja t2 = new Silmukkatulostaja();
14          System.out.println("-----");
15          t1.tulosta();
16          System.out.println("-----");
17          t2.tulosta();
18      }
19
20      public void tulosta() {
21          System.out.println("tulosta()");
22      }
23
24      public String toString() {
25          return "Tulostaja";
26      }
27
28      // jatkuu seuraavalla sivulla
```

```
29      private static class Oliotulostaja extends Tulostaja {
30
31          public void tulosta() {
32
33              if (this instanceof Object && this.totuus) {
34
35                  System.out.println("instanceof");
36
37              } else {
38
39                  System.out.println("!instanceof");
40
41              }
42
43              Tulostaja tulostaja = this;
44
45              System.out.println(tulostaja.toString());
46              System.out.println(this);
47
48          }
49
50          public String toString() {
51              super.toString();
52              return "Oliotulostaja";
53          }
54
55      }
56
57      // jatkuu seuraavalla sivulla
```

```
58      private static class Silmukkatulostaja extends Tulostaja {
59
60          private int muuttuja;
61
62          public Silmukkatulostaja() {
63              super();
64              System.out.println("Silmukkatulostajan konstruktori");
65          }
66
67          public void tulosta() {
68
69              for (this.alusta(); this.ehto(); this.muuta()) {
70                  System.out.println("Vartalo");
71              }
72
73          }
74
75          private void alusta() {
76              this.muuttuja = 3;
77              System.out.println("Alustettu:" + this.muuttuja);
78          }
79
80          private boolean ehto() {
81              boolean b = this.muuttuja > 0;
82              System.out.println("Ehto: " + b);
83              return b;
84          }
85
86          private void muuta() {
87              this.muuttuja--;
88              System.out.println("Muutettu: " + this.muuttuja);
89          }
90
91      }
92
93  }
```

4. **Virheet (8p)**

Alla on ristinollapelin toteuttavan luokan koodi. Koodi kääntyy, mutta siinä on virheitä, eikä se toimi JavaDoc-kommenteissa kuvatulla tavalla. Luettele ja perustele virheelliset kohdat.

```
1  import java.util.*;
2
3  /**
4   * Luokka toteuttaa ristinollapelin. Ristinollassa kaksi pelaajaa, X ja O,
5   * asettavat merkkejään vuoronperään kaksiulotteiselle laudalle. Laudan koko
6   * on rajoittamaton. Pelin voittaa se, joka ensimmäisenä saa laudalle viisi
7   * merkkiä joko pystyyn, vaakaan tai vinottain.
8   *
9   * Pelisiirrot tehdään antamalla koordinaatteja. Koordinaatistossa x kasvaa
10  * oikealle ja y alas. Pelilaudan keskimmäisten ruutujen koordinaatit siis
11  * ovat muodossa (x,y):
12  *
13  *      (-1,-1)  (0,-1)  (1,-1)
14  *
15  *      (-1, 0)  (0, 0)  (1, 0)
16  *
17  *      (-1, 1)  (0, 1)  (1, 1)
18  *
19  */
20 public class Ristinolla {
21
22     /**
23      * Tason pistettä kuvaava luokka. Pisteellä on x- ja y-koordinaatit.
24      * Koordinaatit ovat kokonaislukuja.
25      */
26     public static class Piste {
27
28         public int x;
29         public int y;
30
31         /**
32          * Luo uuden pisteen.
33          *
34          * @param x pisteen x-koordinaatti
35          * @param y pisteen y-koordinaatti
36          */
37         public Piste (int x, int y) {
38             this.x = x;
39             this.y = y;
40         }
41
42         /**
43          * Palauttaa true, jos tämä piste on "yhtä suuri" kuin annettu olio.
44          * Olioiden ei siis tarvitse olla samat.
45          *
46          * @return true, jos tämä piste on "yhtä suuri" kuin annettu olio.
47          */
48         public boolean equals(Object o) {
49             Piste p = (Piste) o;
50             return p.x == this.x && p.y == this.y;
51         }
```

```
52
53          /**
54           * Palauttaa tämän pisteen "hajautuskoodin". Hajautuskoodin idea on,
55           * että mikäli kaksi oliota ovat equals()-metodin mukaan "yhtä
56           * suuria", niillä on myös sama hajautuskoodi.
57           */
58          public int hashCode() {
59              return this.x + this.y;
60          }
61
62      }
63
64      /**
65       * Ristinollapelin pelaajia kuvaava enumeraatio.
66       */
67      public enum Pelaaja {
68
69          X, O;
70
71          /**
72           * Palauttaa sen pelaajan, joka ei ole tämä pelaaja.
73           *
74           * @return tämän pelaajan vastapelaaja.
75           */
76          public Pelaaja toinen() {
77
78              switch(this) {
79              case X:
80                  return O;
81              case O:
82                  return X;
83              }
84              return null;
85          }
86
87      }
88
89      private static final Pelaaja ALOITTAJA = Pelaaja.X;
90
91      private static final int VOITTORIVIN_PITUUS = 5;
92
93      private Map<Piste, Pelaaja> lauta;
94
95      private Pelaaja vuorossa;
96      private Pelaaja voittaja;
97
98      private Piste vasenYlakulma;
99      private Piste oikeaAlakulma;
100
```

```
101     /**
102      * Luo uuden ristinollapelin.
103      */
104     public Ristinolla() {
105         this.lauta = new HashMap<Piste, Pelaaja>();
106         this.vuorossa = ALOITTAJA;
107     }
108
109     /**
110      * Pelaa yhden siirron. Siirron tekee aina vuorossa oleva pelaaja. Jos
111      * peli on voitettu tai annettuun paikkaan on jo pelattu, tämä metodi ei
112      * tee mitään.
113      *
114      * @param paikka
115      *            paikka, johon siirto tehdään.
116      */
117     public void pelaa(Piste paikka) {
118
119         if (this.voittaja != null) {
120             return;
121         }
122
123         if (this.lauta.get(paikka) == null) {
124             return;
125         }
126
127         this.lauta.put(paikka, this.vuorossa);
128         this.tarkistaVoitto(paikka);
129
130         this.paivitaKulmat(paikka);
131
132         System.out.println(vuorossa + " pelasi " + paikka.x + ", " +
133             paikka.y);
134         this.vuorossa = this.vuorossa.toinen();
135     }
136
137
138     /**
139      * Päivittää tiedon siitä, mitkä ovat jo pelatun alueen rajat. Alueesta
140      * on tallennettu vasen yläkulma ja oikea alakulma.
141      *
142      * @param uusiSiirto
143      *            siirto, joka mahdollisesti muuttaa pelatun alueen rajoja.
144      */
145     private void paivitaKulmat(Piste uusiSiirto) {
146         if (uusiSiirto.x <= vasenYlakulma.x &&
147             uusiSiirto.y <= vasenYlakulma.y) {
148             vasenYlakulma = uusiSiirto;
149         }
150
151         if (uusiSiirto.x >= oikeaAlakulma.x &&
152             uusiSiirto.y >= oikeaAlakulma.y) {
153             oikeaAlakulma = uusiSiirto;
154         }
155     }
156
```

```
157     /**
158      * Tarkistaa, aiheuttiko parametrina annettuun pisteeseen pelaaminen
159      * sen, että siirron tehnyt pelaaja voitti pelin.
160      *
161      * @param viimeisin
162      *            edellisen siirron koordinaatit
163      */
164     private void tarkistaVoitto(Piste viimeisin) {
165
166         List<Piste> puoletSuunnista = new ArrayList<Piste>();
167
168         puoletSuunnista.add(new Piste(0, -1));  // pohjoinen
169         puoletSuunnista.add(new Piste(1, -1));  // koillinen
170         puoletSuunnista.add(new Piste(1, 0));   // itä
171         puoletSuunnista.add(new Piste(1, 1));   // kaakko
172
173         for (Piste suunta : puoletSuunnista) {
174
175             int perakkaiset = laskePerakkaiset(viimeisin, suunta);
176             int vastakkaiset = laskePerakkaiset(viimeisin,
177                 new Piste(-suunta.x, -suunta.y));
178
179             int rivinPituus = perakkaiset + vastakkaiset;
180
181             if (rivinPituus > VOITTORIVIN_PITUUS) {
182                 this.voittaja = this.vuorossa;
183                 return;
184             }
185         }
186     }
187
188     /**
189      * Laskee alkupisteestä lähtien tiettyyn suuntaan laudalla olevien
190      * peräkkäisten samojen merkkien määrän.
191      *
192      * @param alkupiste
193      *            piste, josta lähdetään laskemaan perättäisiä
194      * @param delta
195      *            suunta, johon liikutaan. Parametrin koordinaatit
196      *            merkitsevät sitä, paljonko alkupisteeseen täytyy lisätä,
197      *            jotta päästään suunnassa seuraavaan pisteeseen.
198      * @return peräkkäisten samojen merkkien määrä.
199      */
200     private int laskePerakkaiset(Piste alkupiste, Piste delta) {
201
202         Piste testipiste = new Piste(alkupiste.x, alkupiste.y);
203         int perakkaiset = 0;
204         Pelaaja pelaaja = lauta.get(testipiste);
205
206         while (lauta.get(testipiste) == pelaaja) {
207             perakkaiset++;
208             testipiste.x += delta.x;
209             testipiste.y += delta.y;
210         }
211
212         return perakkaiset;
213
214     }
```

```
215     /**
216      * Palauttaa pelilaudan merkkijonoesityksen. Merkkijonoesityksessä
217      * pelatut siirrot ovat omilla paikoillaan niin, että x kasvaa oikealle
218      * ja y alas. Tulosteen vasemmalla ja oikealla laidalla on "seinä".
219      *
220      * @return pelilaudan merkkijonoesitys.
221      */
222     public String toString() {
223
224         String tuloste = "";
225
226         for (int x = vasenYlakulma.x; x <= oikeaAlakulma.x; ++x) {
227
228             tuloste += "|";
229
230             for (int y = vasenYlakulma.y; y <= oikeaAlakulma.y; ++y) {
231
232                 Pelaaja p = this.lauta.get(new Piste(x,y));
233
234                 if (p == null) {
235                     tuloste += " ";
236                 } else {
237                     tuloste += p;
238                 }
239             }
240
241             tuloste += "|";
242
243         }
244
245         return tuloste;
246     }
247
248     /**
249      * Palauttaa pelin voittaneen pelaajan.
250      *
251      * @return pelin voittanut pelaaja.
252      */
253     public Pelaaja annaVoittaja() {
254         return this.voittaja;
255     }
256
257 }
```