T-106.1250 Informaatioverkostojen ohjelmointikurssi
T-106.2001 Informaatioverkostojen studio 1

Tentti 11. tammikuuta 2008

Vastaa kaikkiin kysymyksiin. 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 vaikutus määreellä `static` on Java-metodin määrittelyssä?
c) Miten sijoitusoperaattori (=) toimii Java-ohjelmoinnissa? Mikä vaikutus operaattorin vasemmalla puolella olevan muuttujan tyypillä on operaattorin toimintaan?
d) Mikä voisi aiheuttaa sen, että seuraava koodirivi heittää ajettaessa `NullPointerException`in? Millainen ohjelmointivirhe todennäköisesti on kyseessä?

```
1  System.out.println(this.taulukko[x][y]);
```

2. **Väittämät (8p)**

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

a) Näkyvyysmääreiden tehtävä on estää muita ohjelmoijia näkemästä ja muuttamasta laatimaamme koodia.

b) Java-luokka ei voi periä montaa eri yläluokkaa, mutta samankaltainen toiminnallisuus voidaan silti saavuttaa. Tähän tarvitaan muun muassa rajapintoja (engl. *interface*).

c) Tämän luokan kääntäminen Java-kääntäjällä ei aiheuta käännösvirheitä.

```
1  public class HeiMaailma {
2
3      private char[] merkit = new char[] { 'H', 'e', 'i', ' ',
4              'm', 'a', 'a', 'i', 'l', 'm' ,'a', '!' };
5
6      private void heiMaailma() {
7          for (char c : merkit) {
8              System.out.print(c);
9          }
10         System.out.println();
11     }
12
13     public static void main(String[] args) {
14         heiMaailma();
15     }
16
17 }
```

d) Oletetaan, että olemme laatimassa pasianssipeliä Java-kielellä. Paras tapa mallintaa pelikorttien maat olisi luoda seuraavanlainen luokka.

```
1  public class Maa {
2      public static final int HERTTA = 0;
3      public static final int PATA = 1;
4      public static final int RISTI = 2;
5      public static final int RUUTU = 3;
6  }
```

3. **Mitä tämä koodi tulostaa? (8p)**

Kirjoita näkyviin tuloste, jonka `Tulostaja`-luokan ajaminen aiheuttaa.

```
1   public abstract class Tulostaja {
2
3       private boolean b;
4
5       public Tulostaja(boolean b) {
6
7           this.b = b;
8       }
9
10      public abstract int laske(int alku);
11
12      public void poikkea() {
13
14          try {
15
16              System.out.println("a");
17
18              if (b) {
19                  throw new Exception("b");
20              }
21
22              System.out.println("c");
23
24          } catch (Exception e) {
25
26              System.out.println("catch");
27
28          } finally {
29
30              System.out.println("finally");
31
32          }
33      }
34
35      public static void main(String[] args) {
36
37          Tulostaja oletus = new Oletustulostaja();
38          Tulostaja erikois = new Erikoistulostaja();
39
40          System.out.println("-----");
41          System.out.println(oletus.laske(8));
42          System.out.println("-----");
43          System.out.println(erikois.laske(6));
44          System.out.println("-----");
45          oletus.poikkea();
46          System.out.println("-----");
47          erikois.poikkea();
48      }
49
50  }
51  // Jatkuu seuraavalla sivulla
```

```
52 public class Oletustulostaja extends Tulostaja {
53
54     public Oletustulostaja() {
55         super(true);
56     }
57
58     public int laske(int alku){
59
60         int tulos = 0;
61
62         for (int i = 0; i <= alku; i += 2) {
63
64             if (i == 2) {
65                 continue;
66             }
67
68             System.out.println(tulos);
69             tulos += alku * i;
70
71         }
72
73         return tulos;
74     }
75
76 }
77
78 public class Erikoistulostaja extends Tulostaja {
79
80     public Erikoistulostaja() {
81         super(false);
82     }
83
84     public int laske(int alku) {
85
86         System.out.println(alku);
87
88         if (alku < 1) {
89             return 2;
90         } else if (alku < 5) {
91             return this.laske(alku - 2);
92         } else {
93             return this.laske(alku - 1);
94         }
95
96     }
97
98 }
```

4. **Virheet (8p)**

Alla on jätkänshakkipelin toteuttavan luokan koodi. Koodissa on virheitä, eikä luokka toimi JavaDoc-kommenteissa kuvatulla tavalla. Luettele ja perustele virheelliset kohdat.

```
1   import java.util.Random;
2
3   /**
4    * Luokka kuvaa jätkänshakkipeliä. Jätkänshakissa kaksi pelaajaa, X ja O,
5    * tekevät vuorotellen siirtoja 3x3-kokoisella laudalla. Pelin tavoitteena
6    * on saada kolmen suora.
7    */
8   public class Jatkanshakki {
9
10      /**
11       * Jätkänshakin pelaajia kuvaava enumeraatio.
12       */
13      private enum Pelaaja {
14
15          X, O;
16
17          public Pelaaja toinen() {
18              if (this == X) {
19                  return O;
20              }
21              return X;
22          }
23
24      }
25
26      private static final Pelaaja ALOITTAJA = Pelaaja.X;
27
28      private Pelaaja[][] lauta;
29
30      private Pelaaja seuraavaPelaaja;
31
32      private int pelatutMerkit;
33
34      /**
35       * Luo uuden jätkänshakkipelin.
36       */
37      public Jatkanshakki() {
38          this.lauta = new Pelaaja[3][3];
39          this.seuraavaPelaaja = ALOITTAJA;
40      }
41
```

```
42      /**
43       * Pelaa yhden siirron. Siirron aikana vuorossa oleva pelaaja lisää
44       * laudalle merkin.
45       * <p>
46       * Jos annettu pelaaja ei ole vuorossa, tai peli on päättynyt, siirtoa
47       * ei voi tehdä. Siirtoa ei voi tehdä myöskään, jos annettu laudan ruutu
48       * ei ole tyhjä.
49       *
50       * @param pelaaja
51       *            pelaaja, jonka siirto pelataan.
52       * @param x
53       *            pelattavan ruudun x-koordinaatti.
54       * @param y
55       *            pelattavan ruudun y-koordinaatti.
56       * @return true jos siirto onnistui, muuten false.
57       */
58      public boolean pelaaSiirto(Pelaaja pelaaja, int x, int y) {
59
60          if (seuraavaPelaaja != pelaaja ||
61                  annaMerkki(x, y) != null ||
62                  peliPaattynyt()) {
63              return false;
64          }
65
66          lauta[x][y] = pelaaja;
67          seuraavaPelaaja = pelaaja.toinen();
68          return true;
69      }
70
71      /**
72       * Palauttaa annetuissa koordinaateissa olevan merkin, mikäli sellainen
73       * on sinne pelattu. Jos annetut koordinaatit ovat laittomat,
74       * palautetaan null.
75       *
76       * @param x
77       *            x-koordinaatti.
78       * @param y
79       *            y-koordinaatti.
80       * @return koordinaatteihin pelattu merkki, tai null, jos sellaista ei
81       *         ole.
82       *
83       */
84      private Pelaaja annaMerkki(int x, int y) {
85
86          if (x < 0 && y < 0 && x > 2 && y > 2) {
87              return null;
88          }
89
90          return lauta[y][x];
91      }
```

```
92      /**
93       * Palauttaa pelin voittajan, tai null, jos peli on kesken tai päättynyt
94       * tasan.
95       * <p>
96       * Pelin voittaa se pelaaja, joka ensimmäisenä saa laudalle kolmen
97       * merkin suoran joko pystyyn, vaakaan tai vinottain.
98       *
99       * @return pelin voittanut pelaaja, tai null, jos pelillä ei ole
100      *         voittajaa.
101      */
102     public Pelaaja annaVoittaja() {
103
104         for (int x = 0; x < 3; ++x) {
105             if (annaMerkki(x, 0) != null &&
106                     annaMerkki(x, 0) == annaMerkki(x, 1) &&
107                     annaMerkki(x, 1) == annaMerkki(x, 2)) {
108
109                 return annaMerkki(x, 0);
110             }
111         }
112
113         for (int y = 0; y < 3; ++y) {
114             if (annaMerkki(y, 0) != null &&
115                     annaMerkki(y, 0) == annaMerkki(y, 1) &&
116                     annaMerkki(y, 1) == annaMerkki(y, 2)) {
117
118                 return annaMerkki(y, 0);
119             }
120         }
121
122         if (annaMerkki(0, 0) != null &&
123                 annaMerkki(0, 0) == annaMerkki(1, 1) &&
124                 annaMerkki(1, 1) == annaMerkki(2, 2)) {
125
126             return annaMerkki(0, 0);
127         }
128
129         if (annaMerkki(0, 2) != null &&
130                 annaMerkki(0, 2) == annaMerkki(1, 1) &&
131                 annaMerkki(1, 1) == annaMerkki(2, 0)) {
132
133             return annaMerkki(0, 0);
134         }
135
136         return null;
137     }
```

```
138     /**
139      * Palauttaa true, jos peli on päättynyt, tai false, jos peli on kesken.
140      * Peli on päättynyt, mikäli jompikumpi pelaaja on voittanut tai lauta
141      * on pelattu täyteen.
142      *
143      * @return true, jos peli on päättynyt, tai false, jos peli on kesken.
144      */
145     public boolean peliPaattynyt() {
146         return annaVoittaja() != null && pelatutMerkit == 9;
147     }
148
149     /**
150      * Palauttaa pelin merkkijonokuvauksen. Kuvauksessa on pelin nykyinen
151      * tilanne ja sen alla pelin voittaja.
152      *
153      * Kuvaus voi olla esimerkiksi:
154      *
155      * <pre>
156      * -----
157      * |XOX|
158      * |X O|
159      * |XO |
160      * -----
161      * Voittaja: X
162      * </pre>
163      *
164      * @return pelin merkkijonokuvaus.
165      */
166     public String toString() {
167
168         String kuvaus = "-----\n";
169
170         for (Pelaaja[] rivi : lauta) {
171
172             for (Pelaaja merkki : rivi) {
173                 kuvaus += "|";
174                 kuvaus += merkki == null ? " " : merkki;
175             }
176             kuvaus += "|\n";
177         }
178
179         kuvaus += "-----\n";
180
181         kuvaus += "Voittaja: " +
182             (annaVoittaja() == null ? annaVoittaja() : "Ei kukaan");
183
184         return kuvaus;
185     }
```

```
186     /**
187      * Metodi, joka testaa Jatkanshakki-luokan toimintaa. Metodi luo uuden
188      * pelin, ja pelaa siirtoja vuorotellen satunnaisiin ruutuihin kunnes
189      * peli on päättynyt. Joka kierroksen jälkeen tulostetaan pelitilanne.
190      *
191      * @param args
192      *            ei käytössä
193      */
194     public static void main(String[] args) {
195
196         Jatkanshakki j = new Jatkanshakki();
197
198         Random rand = new Random();
199
200         while (!j.peliPaattynyt()) {
201             for (Pelaaja p : Pelaaja.values()) {
202                 if (j.pelaaSiirto(p, rand.nextInt(2), rand.nextInt(2))) {
203                     System.out.println(j);
204                 }
205                 if (j.peliPaattynyt()) {
206                     break;
207                 }
208             }
209         }
210
211     }
212
213 }
```