Tentti: T-106.1200/1203 Ohjelmoinnin perusteet T/L

Tenttipäivä: 15.12.2008

Yleistä

Tentissä on kaksi tehtävää. Niillä pyritään varmistamaan, että jokaisella kurssin läpäisevällä opiskelijalla on vähintäänkin auttavat taidot ohjelmakoodin lukemisessa ja kirjoittamisessa. Kummastakin tehtävästä saa palautteena jonkin näistä kolmesta:

- **hylätty:** Vastaus on selvästi puutteellinen tai väärin. Vastauksen perusteella ei saa sitä käsitystä, että vastaaja osaa lukea/kirjoittaa Java-ohjelmakoodia.
- **hyvä:** Vastaus kelpaa, vaikka siinä onkin joitain puutteita tai virheitä.
- **erinomainen:** Vastaus on käytännöllisesti katsoen täysin oikein.

Tentin arvosanaksi tulee kahden tehtävän perusteella joko hyväksytty tai hylätty. Numeroarvosanaa siis ei ole. Hyväksytyn arvosanan saa, kunhan kumpaakaan tehtävää ei hylätä. (Koko kurssisuorituksen arvosanahan määräytyy harjoitustehtäväkierrosten pisteiden perusteella. Kurssiarvosanan kannalta ei ole merkitystä sillä, arvioitiinko tenttitehtävät hyviksi vai erinomaisiksi.)

Tehtävä 1: lukutaito

Tutustu tarkasti oheisessa liitteessä annettuihin luokkiin.

Mitä ohjelma tulostaa, kun ajetaan luokan `GameTest` sisältämä käynnistysmetodi?

Annetuista luokista tiedetään seuraavaa:

- Ne ovat osa ohjelmaa, jolla voi pitää kirjaa (kuvitteellisen) Cryptix-seurapelin etenemisestä ja pelaajien saamista pisteistä.
- Kukin luokan `Game` ilmentymä kuvaa yksittäistä Cryptix-pelikertaa, johon osallistuu muutama pelaaja.
- Kukin luokan `Player` ilmentymä edustaa yksittäistä tällaiseen pelisessioon osallistuvaa pelaajaa.
- Pelaaja voi liittyä mukaan Cryptix-peliin pelikerran alussa. Lisäksi pelin jo ollessa käynnissä peliin voi tulla mukaan uusia pelaajia joko lisäpelaajina tai peliin aiemmin liittyneiden tilalle.
- Cryptix-pelikerta koostuu eri pelaajien vuoroista (*turn*), joiden aikana pelaajilla on mahdollisuus kerätä pisteitä.
- Luokassa `GameTest` on käynnistysmetodi, joka koekäyttää kahta muuta luokkaa.

Kaikki muu tarvittava tieto Cryptix-pelin vuorojen etenemisestä ja pistelaskusta sinun on pääteltävä itse annetun ohjelmakoodin perusteella.

Tässä muistin virkistykseksi muutama `ArrayList`-luokan metodi, ettei vastaaminen jää ulkomuistista kiinni:

- Yksiparametrinen `add`-metodi lisää alkion (*element*) listan perään.
- `get`-metodi kertoo, mikä arvo on tallennettu määrätylle listan indeksille.
- `set`-metodi korvaa määrätyllä indeksillä olevan alkion määrätyllä uudella arvolla.
- `size`-metodi kertoo listan alkioiden lukumäärän.

Täysiin pisteisiin riittää oikean tulosteen kirjaaminen vastauspaperiin. Oikean tulosteen päätteleminen edellyttää huolellisuutta. Jos haluat, voit ottaa riskin ja vastata pelkällä ohjelman tulosteella. Kuitenkin kannattanee perustella vastaustasi sanallisesti tentin arvostelijalle. Vähän virheelliselläkin vastauksella voi päästä läpi, kunhan perusteluista käy ilmi, että kykenet riittävästi ymmärtämään Java-ohjelmakoodia. Sen sijaan virheellinen tuloste ilman perusteluja voi olla tenttisuorituksen kannalta kohtalokas.

Tehtävä 2: kirjoitustaito

Oletetaan, että ollaan laatimassa urheilutuloksia käsittelevää ohjelmaa. Ohjelmassa joukkueiden viimeisimmät ottelutulokset on kuvattu merkkijonoina (*string*), joissa voitot (W), tappiot (L) ja tasapelit (D) ovat aikajärjestyksessä yksittäisinä kirjaimina. Halutaan laatia metodi, jolla voidaan etsiä tällaisesta merkkijonosta joukkueen voitto-, tappio- ja tasapeliputkia eli peräkkäisten samankaltaisten tulosten sarjoja. Esimerkiksi merkkijonosta DDWWWDDLLWWWWWL löytyy kolmen W:n mittainen voittoputki alkaen indeksistä 2 ja viiden mittainen voittoputki alkaen indeksistä 9. (Merkkijonon merkkien indeksit alkavat nollasta.)

Kirjoita Java-kielinen metodi, joka:

- Ottaa parametreinaan kolme arvoa: merkkijonon `data`, merkin `target` ja kokonaisluvun `minimumLength`.
- Etsii `data`-merkkijonosta `target`-merkin esiintymiä: yritetään löytää merkkijonon sisältä sellainen kohta, jossa merkki esiintyy vähintään `minimumLength` kertaa peräkkäin.
- Palauttaa indeksin, josta löydetty peräkkäisten merkkien sarja alkaa. Jos tällaisia kohtia on annetussa merkkijonossa useita, palauttaa näiden alkuindekseistä pienimmän.
- Palauttaa luvun -1, jos yhtään ehdot täyttävää kohtaa ei löydy annetusta merkkijonosta.
- Ei kaadu ajonaikaiseen virheeseen silloinkaan, kun annettu `data`-merkkijono on tyhjä eli nollan mittainen. (Kuitenkin saat olettaa, että `data` ei ole `null` ja että `minimumLength` on positiivinen luku.)

Esimerkkejä: Jos `data` on DDWWWDDLLWWWWWL ja `target` on W, niin:

- `minimumLength`in ollessa 3 metodi palauttaa 2.
- `minimumLength`in ollessa 1 metodi palauttaa 2.
- `minimumLength`in ollessa 4 metodi palauttaa 9.
- `minimumLength`in ollessa 6 metodi palauttaa -1.

Tässä tehtävässä riittää kun laadit yksittäisen metodin. Nimeä se itse. Kokonaista luokkaa ei tarvitse laatia.

Metodin toteuttamiseen on monia tapoja. Mikä tahansa toimiva ratkaisutapa kelpaa. Esimerkiksi näitä `String`-luokan metodeita voi hyödyntää:

- Parametriton `length`-metodi palauttaa merkkijonon pituuden (eli merkkien määrän).
- `charAt`-metodille annetaan parametriksi indeksi, ja se palauttaa sen merkin, joka on merkkijonossa kyseisellä indeksillä.

Vaikka toimivuus on ratkaisua arvosteltaessa tärkeintä, muista myös hyvä koodinkirjoitustyyli. Kirjoita koodi siten, että arvostelija pystyy lukemaan sen ilman suurempia ponnistuksia. Epäselvä tyyli huomioidaan arvostelussa. Kauhealla tyylillä kirjoitettu ohjelma voidaan jopa hylätä.

Yksittäisistä pienistä "pilkkuvirheistä" ei arvostelussa sakoteta, mutta yritä silti kirjoittaa koodi niin täsmällisesti oikein kuin pystyt. Kokonaisuus ratkaisee.

(Jos haluat hieman lisäharjoitusta, voit miettiä seuraavaa: Miten metodista voisi tehdä sellaisen, että se etsiikin kaikkein pisimmän voitto-, tappio- tai tasapeliputken? Tähän lisäkysymykseen ei kuitenkaan tarvitse tässä tenttitehtävässä löytää vastausta.)

Hyvää tenttiä!

```
1: import java.util.ArrayList;
2:
3:
4: public class Game {
5:
6:   private ArrayList<Player> players;
7:   private int turn;
8:
9:   public Game() {
10:    this.players = new ArrayList<Player>();
11:    this.turn = 0;
12:  }
13:
14:  public void addPlayer(String newPlayerName) {
15:    Player newPlayer = new Player(newPlayerName);
16:    this.players.add(newPlayer);
17:  }
18:
19:  public void playTurn(int points) {
20:    this.getCurrent().addPoints(points);
21:    this.turn = this.turn + 1;
22:    if (this.turn >= this.players.size()) {
23:      this.turn = 0;
24:    }
25:  }
26:
27:  public int getTurn() {
28:    return this.turn;
29:  }
30:
31:  public Player getCurrent() {
32:    return this.players.get(this.turn);
33:  }
34:
35:  public Player getLeader() {
36:    Player leader = null;
37:    for (Player candidate : this.players) {
38:      if (leader == null || candidate.getPoints() > leader.getPoints()) {
39:        leader = candidate;
40:      }
41:    }
42:    return leader;
43:  }
44:
45:  public void enterNewPlayer(String newPlayerName) {
46:    this.addPlayer(newPlayerName);
47:    this.turn = this.players.size() - 1;
48:  }
49:
50:  public void replacePlayer(Player oldPlayer, String newPlayerName) {
51:    Player newPlayer = new Player(newPlayerName);
52:    newPlayer.addPoints(oldPlayer.getPoints());
53:    for (int index = 0; index < this.players.size(); index++) {
54:      Player candidate = this.players.get(index);
55:      if (candidate == oldPlayer) {
56:        this.players.set(index, newPlayer);
57:      }
58:    }
59:  }
60:
61: }
62:
63:
64:
65:
66:
67:
```

```
68: public class Player {
69:
70:   private String name;
71:   private int score;
72:
73:   public Player(String name) {
74:     this.name = name;
75:     this.score = 0;
76:   }
77:
78:   public void addPoints(int newPoints) {
79:     this.score = this.score + newPoints;
80:   }
81:
82:   public void setPoints(int newPoints) {
83:     this.score = newPoints;
84:   }
85:
86:   public String getName() {
87:     return this.name;
88:   }
89:
90:   public int getPoints() {
91:     return this.score;
92:   }
93: }
94:
95: public class GameTest {
96:
97:   public static void main(String[] args) {
98:     Game exampleGame = new Game();
99:     exampleGame.addPlayer("Jussi");
100:    exampleGame.addPlayer("Kalle");
101:    exampleGame.addPlayer("Laura");
102:    exampleGame.addPlayer("Minna");
103:
104:    Player who1 = exampleGame.getCurrent();
105:    System.out.println(who1.getName());
106:
107:    exampleGame.playTurn(3);
108:    exampleGame.playTurn(5);
109:    exampleGame.playTurn(4);
110:    exampleGame.playTurn(10);
111:    exampleGame.playTurn(5);
112:    exampleGame.playTurn(7);
113:
114:    Player who2 = exampleGame.getCurrent();
115:    System.out.println(who2.getName());
116:    Player who3 = exampleGame.getLeader();
117:    System.out.println(who3.getName());
118:
119:    exampleGame.replacePlayer(who3, "Niklas");
120:
121:    Player who4 = exampleGame.getLeader();
122:    System.out.println(who4.getName());
123:
124:    exampleGame.enterNewPlayer("Olivia");
125:    exampleGame.playTurn(15);
126:    exampleGame.playTurn(2);
127:
128:    Player who5 = exampleGame.getCurrent();
129:    System.out.println(who5.getName());
130:    Player who6 = exampleGame.getLeader();
131:    System.out.println(who6.getName());
132:  }
133: }
134:
```