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

Tenttipäivä: 14.12.2010

Yleistä

Tentissä on kaksi tehtävää. Ensimmäinen tehtävä on tärkeämpi: sillä pyritään varmistamaan, että jokaisella kurssin läpäisevällä opiskelijalla on vähintäänkin auttavat taidot ohjelmakoodin lukemisessa ja kirjoittamisessa. Toinen tehtävä liittyy aiheisiin, jotka ovat hyödyllisiä, mutta eivät tämän kurssin kannalta aivan välttämättömimpiä. Sen suorittamalla voi korottaa kurssiarvosanaansa.

Tentistä tulee arvosanaksi jokin kolmesta vaihtoehdosta: *hylätty, hyväksytty* tai *+1*. Kunhan ensimmäisestä tehtävästä ei tule hylättyä arvosanaa, niin tentistä pääsee hyväksytysti läpi. Arvosanan *+1* ja korotuksen kurssiarvosanaan saa ratkaisemalla myös jälkimmäisen tehtävän.

Hyvää tenttiä!

Tehtävä 1

Tarkastellaan ohjelmaa, joka käsittelee (yksinkertaistetulla tavalla) bussien pysäkkiaikatauluja. Tutustu huolellisesti oheisiin luokkiin `BusTest`, `Departure` ja `BusStop`. Kukin `BusStop`-olio kuvaa yksittäistä bussipysäkkiä, jolta lähtee busseja tiettyihin vakioaikoihin päivittäin. Kukin `Departure`-olio kuvaa yhtä "kohtaa" tällaisessa pysäkkiaikataulussa: tiettyä päivittäistä lähtökertaa tietyltä pysäkiltä. Luokka `BusTest` sisältää käynnistysmetodin, jossa on esimerkkejä `BusStop`- ja `Departure`-luokkien käytöstä.

Toteuta luokan `BusStop` metodi `getDailySchedule` siten, että se toimii kuvatulla tavalla. Toteutustapa on muuten vapaa.

Vinkkejä:

- On tärkeää, että tutustut annettuihin luokkiin huolella hahmottaaksesi, miten ne toimivat! Katso ajatuksella läpi `BusTest`-luokan esimerkkitapaukset.
- Metodi `getNextDepartures` on jo toteutettu valmiiksi. Se on tietyillä tavoilla samankaltainen kuin `getDailySchedule`, joka sinun tulee laatia.
- Et tarvitse mitään sellaisia listaluokan `ArrayList` piirteitä, joista ei olisi esimerkkiä annetussa koodissa.
- Muistutus: operaattori `%` laskee jakojäännöksen: esimerkiksi `17 % 5` on kaksi ja `28 % 7` on nolla. Voit hyödyntää tätä operaattoria, jos haluat (mutta se ei ole välttämätöntä).

Tehtävästä saa palautteena jonkin näistä kolmesta:

- **hylätty:** Metodin toteutus on selvästi puutteellinen ja toimimaton. Vastauksen perusteella ei saa käsitystä, että vastaaja osaa lukea/kirjoittaa Java-ohjelmakoodia. Tenttisuoritusta ei voida hyväksyä.
- **hyvä:** Vastaus kelpaa, vaikka siinä onkin joitain puutteita tai virheitä.
- **erinomainen:** Vastaus on käytännöllisesti katsoen täysin oikein. (Arvosanan kannalta on ihan sama, saatko hyvän vai erinomaisen, mutta on varmaan silti kiva kuulla, jos vastaus on ollut mainio.)

Yksittäisistä pikku- ja pilkkuvirheistä ei sakoteta, mutta yritä silti kirjoittaa koodi niin täsmällisesti oikein kuin pystyt. Kokonaisuus ratkaisee. Arvostelu tässä ensimmäisessä tehtävässä ei tule olemaan valtavan ankara, joten turha stressi pois!

Tehtävä 2

Tässäkin tehtävässä tutustut annettuun koodiin ja täydennät sitä. Tutustu aluksi oheisiin luokkiin `CalculationTest` ja `Data`. Kukin `Data`-olio kuvaa kokonaislukujen kokoelmaa, johon voi kohdistaa erilaisia laskutoimituksia. Luokassa `CalculationTest` on esimerkkiohjelma, jossa erästä dataoliota käsitellään kolmella eri tavalla. Kutakin näistä erilaisista laskutoimitustyypeistä kuvaa yksi olio, joka toteuttaa `Calculation`-rajapintamäärittelyn (engl. *interface*).

Jotta `CalculationTest` toimisi, tarvitaan kolme luokkaa – `Product`, `Count` ja `SumSquaresOfPositives` – joista ainoastaan `Product` on annettu valmiina.
Lisäksi rajapintamäärittely `Calculation` puuttuu. Tehtäväsi on laatia rajapintamäärittely `Calculation` sekä luokat `Count` ja `SumSquaresOfPositives` siten, että ne saavat esimerkkiohjelman toimimaan kuvatulla tavalla. Laadi näihin luokkiin ja rajapintamäärittelyyn vain tehtävän kannalta oleelliset metodit. Älä muuta valmiina annettuja luokkia.

Vinkkejä:

- Varmista, että ymmärrät, miten tulon laskeminen toimii luokkien `Data` ja `Product` avulla. Voit muokata tätä perusajatusta sopivasti toteuttaaksesi luokat `Count` ja `SumSquaresOfPositives`.
- Huomaa rekursio `Data`-luokan `process`-metodissa. (Varmista, että ymmärrät sen!)
- Laadittavat luokat ja rajapintamäärittely ovat lyhyitä. Jos huomaat kirjoittavasi isoja määriä koodia, olet varmastikin tekemässä jotain pieleen.
- Näihin luokkiin ei tarvitse laatia konstruktoreja.

Tehtävästä saa palautteena jonkin näistä kolmesta:

- **puutteellinen:** Vastaus on selvästi puutteellinen tai väärin. (Ei arvosanakorotusta.)
- **sinnepäin:** Vastauksessa on jotain oikeaa, mutta myös jotain oleellista puuttuu. (Tämäkään ei vielä riitä arvosanakorotukseen.)
- **oikein:** Vastauksessa on kaikki pyydetyt osiot toimiviksi toteutettuina . (Tästä saa +1 kurssiarvosanaan.)

Arvosanakorotus edellyttää siis, että kaikki tehtävänannossa pyydetyt asiat ovat kunnossa. Tässäkään tehtävässä ei kuitenkaan muotoseikoista sakoteta. Jos et muista varmasti, miten jokin tietty asia kirjataan Java-ohjelmakoodiin, voit yrittää paikata tätä selittämällä sen, mitä koodisi tekee.

Muista käydä täyttämässä kurssipalautekysely viimeistään maanantaina 20.12. kurssisivuilla! Käy vaikka heti tentin jälkeen! (Kurssipalaute on pakollinen kurssin osasuoritus.)

```
// TEHTÄVÄN 1 KOODI ALKAA

/*
 * Esimerkkiohjelma, joka käyttää BusStop- ja Departure-luokkia.
 * Jos nuo luokat on toteutettu oikein, niin tämä esimerkkiohjelma tulostaa:
 *
 * Kolme seuraavaa alkaen indeksistä 1.
 * 913 07:15
 * 912 12:30
 * 912 18:00
 * Seitsemän seuraavaa alkaen indeksistä 4.
 * 913 19:00
 * 912 07:00
 * 913 07:15
 * 912 12:30
 * 912 18:00
 * 913 19:00
 * 912 07:00
 * Linjan 913 lähtöajat alkaen indeksistä 0.
 * 913 07:15
 * 913 19:00
 * Linjan 912 lähtöajat alkaen indeksistä 3.
 * 912 18:00
 * 912 07:00
 * 912 12:30
 */
public class BusTest {

  public static void main(String[] args) {
    BusStop testStop = new BusStop("Jälki-Hikiä");
    testStop.addDailyDeparture(new Departure(912, "07:00")); // indeksille 0
    testStop.addDailyDeparture(new Departure(913, "07:15"));
    testStop.addDailyDeparture(new Departure(912, "12:30"));
    testStop.addDailyDeparture(new Departure(912, "18:00"));
    testStop.addDailyDeparture(new Departure(913, "19:00")); // indeksille 4

    System.out.println("Kolme seuraavaa alkaen indeksistä 1.");
    for (Departure current : testStop.getNextDepartures(1, 3)) {
      System.out.println(current.toString());
    }

    System.out.println("Seitsemän seuraavaa alkaen indeksistä 4.");
    for (Departure current : testStop.getNextDepartures(4, 7)) {
      System.out.println(current.toString());
    }

    System.out.println("Linjan 913 lähtöajat alkaen indeksistä 0.");
    for (Departure current : testStop.getDailySchedule(913, 0)) {
      System.out.println(current.toString());
    }

    System.out.println("Linjan 912 lähtöajat alkaen indeksistä 3.");
    for (Departure current : testStop.getDailySchedule(912, 3)) {
      System.out.println(current.toString());
    }

  }

}
```

```
import java.util.ArrayList;

/* Kuvaa bussipysäkkejä. Kultakin bussipysäkiltä lähtee busseja
 * tiettyihin aikoihin päivittäin. */
public class BusStop {

  private String name;
  private ArrayList<Departure> dailyDepartures;

  public BusStop(String name) {
    this.name = name;
    this.dailyDepartures = new ArrayList<Departure>();
  }

  /* Lisää uuden päivittäisen lähtöajan pysäkille. Metodi on toteutettu tässä yksin-
   * kertaisesti niin, että lähtöajat pidetään lisäysjärjestyksessä. Oikeassa järjes-
   * tyksessä lisääminen jää metodin kutsujan huoleksi (ks. esim. luokka BusTest). */
  public void addDailyDeparture(Departure newDeparture) {
    this.dailyDepartures.add(newDeparture);
  }

  /* Palauttaa (uuden) listan, joka sisältää seuraavia bussien lähtöaikoja tältä
   * pysäkiltä. Listan sisältämät lähtöajat määräytyvät päivittäisen lähtöaika-
   * luettelon sekä kahden metodiparametrin perusteella. Ensimmäinen parametri
   * (start) määrää, mistä kohdasta lähtöaikaluetteloa aloitetaan ja toinen (howMany)
   * määrittää montako lähtöaikaa listaan otetaan yhteensä mukaan.
   *
   * Esimerkiksi jos start on 0 ja howMany on 6, niin aloitetaan aivan
   * päivittäisten lähtöaikojen luettelon alusta ja otetaan listaan mukaan
   * päivän ensimmäisen lisäksi viisi sitä seuraavaa lähtöä tältä pysäkiltä.
   * HUOM! Esimerkkejä metodin käytöstä luokassa BusTest! */
  public ArrayList<Departure> getNextDepartures(int start, int howMany) {
    ArrayList<Departure> nexts = new ArrayList<Departure>();
    for (int count = 0; count < howMany; count++) {
      int index = (start + count) % this.getTraffic();
      nexts.add(this.getNth(index));
    }
    return nexts;
  }

  /* Palauttaa (uuden) listan, joka sisältää kaikki tietyn
   * bussilinjan päivittäiset lähtöajat tältä pysäkiltä.
   *
   * Ensimmäinen parametri (serviceNumber) määrittää, minkä bussilinjan lähtöajat
   * haetaan. Toinen parametri (start) määrittää, mistä kohdasta (siis mistä
   * indeksistä) päivittäistä aikataulua aloitetaan. Palautettu lista sisältää
   * kaikki kyseisen bussilinjan lähtöajat järjestyksessä siten, että ensin ovat
   * lähtöajat kohdasta start päivittäisen aikataulun loppuun asti, ja sitten
   * lähtöajat päivittäisen aikataulun alkupäästä (startia pienemmiltä indekseiltä).
   * HUOM! Konkreettisia esimerkkejä metodin käytöstä luokassa BusTest! */
  public ArrayList<Departure> getDailySchedule(int serviceNumber, int start) {
    // TOTEUTUS PUUTTUU: TÄYDENNÄ!
  }

  /* Kertoo, montako kertaa päivässä tältä pysäkiltä lähtee bussi. */
  public int getTraffic() {
    return this.dailyDepartures.size();
  }

  /* Palauttaa yhden tämän pysäkin päivittäisistä lähtöajoista
   * (metodin parametri määrää monennenko). */
  public Departure getNth(int n) {
    return this.dailyDepartures.get(n);
  }

}
```

```java
/* Kuvaa bussin lähtökertoja. Yksi Departure-olio vastaa yhtä
 * bussin päivittäistä lähtökertaa joltakin pysäkiltä. */
public class Departure {

  private int serviceNumber; // bussilinjan numero
  private String time;       // lähtöaika tekstinä kuvattuna

  public Departure(int serviceNumber, String time) {
    this.serviceNumber = serviceNumber;
    this.time = time;
  }

  public int getServiceNumber() {
    return this.serviceNumber;
  }

  public String toString() {
    return this.serviceNumber + " " + this.time;
  }

}

// TEHTÄVÄN 1 KOODI LOPPUU
```

```java
// TEHTÄVÄN 2 KOODI ALKAA

/*
 * Kun luokat Product, Count ja SumSquaresOfPositives sekä
 * rajapintamäärittely Calculation on toteutettu oikein,
 * tämän esimerkkiohjelman pitäisi tulostaa kolme lukua,
 * joista ensimmäinen on kaikkien lukujen tulo (engl. product),
 * toinen on lukujen lukumäärä, ja kolmas on lukujen toisten
 * potenssien summa, kun vain positiiviset luvut huomioidaan:
 *
 * -360
 * 4
 * 145
 */
public class CalculationTest {

  public static void main(String[] args) {
    Data data = new Data(new int[] { 3, -2, 10, 6 });
    System.out.println(data.process(new Product()));
    System.out.println(data.process(new Count()));
    System.out.println(data.process(new SumSquaresOfPositives()));
  }

}


public class Data {

  private int[] numbers;

  public Data(int[] numbers) {
    this.numbers = numbers;
  }

  public int process(Calculation calculation) {
    return this.process(calculation, 0, calculation.getInitial());
  }

  private int process(Calculation calculation, int current, int total) {
    if (this.numbers.length == current) {
      return total;
    } else {
      return this.process(calculation,
                          current + 1,
                          calculation.apply(total, this.numbers[current]));
    }
  }

}


public class Product implements Calculation {

  public int apply(int result, int current) {
    return result * current;
  }

  public int getInitial() {
    return 1;
  }

}

// TEHTÄVÄN 2 KOODI LOPPUU
```