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

Tenttipäivä: 08.01.2009

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 liitteessä 1 annettuihin luokkiin. Luokka `Building` kuvaa (suuria) taloja, joissa voi olla useita kerroksia (numeroitu nollasta alkaen) ja useita rinnakkaisia hissejä. Hissistön käyttäjä voi tilata lähimmän vapaana olevan hissin käyttöönsä luokasta `Building` löytyvällä metodilla. Hissejä kuvaa luokka `Elevator`. Luokan `Test` avulla voi koekäyttää em. kahta luokkaa.

Mitä ohjelma tulostaa, kun ajetaan luokan `Test` käynnistysmetodi?

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

Kirjoita Java-kielinen metodi, joka:

- Saa parametrikseen merkkijonoja (`String`) sisältävän listan. Annetussa listassa voi olla alkioina myös `null`-arvoja, mistä metodin on selviydyttävä kaatumatta.
- Palauttaa pisimmän ja lyhimmän listassa olevan merkkijonon pituuksien erotuksen. `null`-arvot listassa tulkitaan tässä "nollan pituisiksi", vaikka ne eivät oikeasti merkkijonoja olekaan.
- Palauttaa nolla, jos listassa ei ole alkioita (tai jos pisin ja lyhin jono ovat yhtä pitkät).

Esimerkiksi jos metodin parametrina saamassa listassa on "abc", "muu", "ananasakäämä", "lordi" ja "humppa", niin metodin kuuluu palauttaa luku 9 (eli 12 - 3). Jos taas parametrilistassa olisi lisäksi ollut `null`-arvo, niin metodin tulisi palauttaa luku 12 (eli 12 - 0).

Riittää kun laadit yksittäisen metodin. Nimeä se itse. Kokonaista luokkaa ei tarvitse laatia. Metodin toteuttamiseen on monia tapoja. Mikä tahansa toimiva ratkaisutapa kelpaa.

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

Hyvää tenttiä!

```java
public class Elevator {

  private int currentFloor;  // most-recent holder
  private int topFloor;      // fixed value
  private boolean doorOpen;  // most-recent holder, "flag"

  public Elevator(int topFloor) {
    this.currentFloor = 0;
    this.topFloor = topFloor;
    this.doorOpen = false;
  }

  private void openDoor() {
    this.doorOpen = true;
  }

  public void closeDoor() {
    this.doorOpen = false;
  }

  public boolean isAvailable() {
    return this.doorOpen == false;
  }

  public int getFloor() {
    return this.currentFloor;
  }

  public int getDistance(int destination) {
    //  Math.abs returns the absolute value (Finnish: itseisarvo) of its parameter.
    return Math.abs(this.currentFloor - destination);
  }

  // "A button is pressed inside the elevator, so it travels
  // to the given destination."
  public boolean travelTo(int destination) {
    if (destination >= 0 && destination <= this.topFloor) {
      this.closeDoor();
      this.currentFloor = destination;
      this.openDoor();
      return true;
    } else {
      return false;
    }
  }

  // "A button is pressed outside, ordering the elevator to
  // come to the given destination."
  public boolean orderTo(int destination) {
    if (this.isAvailable() && destination >= 0 &&
        destination <= this.topFloor) {
      this.currentFloor = destination;
      this.openDoor();
      return true;
    } else {
      return false;
    }
  }

  public String getDescription() {
    return "floor: " + this.getFloor() +
           ", available: " + this.isAvailable();
  }

}
```

```java
import java.util.ArrayList;

public class Building {

  private String name;                        // fixed value
  private int height;                         // fixed value
  private ArrayList<Elevator> elevators;  // container

  public Building(String name, int height) {
    this.name = name;
    this.height = height;
    this.elevators = new ArrayList<Elevator>();
  }

  public void addElevator() {
    Elevator newElevator = new Elevator(this.height - 1);
    this.elevators.add(newElevator);
  }

  public String getName() {
    return this.name;
  }

  public Elevator orderElevatorToFloor(int destination) {
    Elevator closest = null;                    // most-wanted holder
    for (int idx = this.elevators.size() - 1; idx >= 0; idx--) { // idx: stepper
      Elevator current = this.elevators.get(idx); // current: most-recent holder
      if ((closest == null || current.getDistance(destination) <
                              closest.getDistance(destination))) {
        if (current.isAvailable()) {
          closest = current;
        }
      }
    }
    if (closest != null) {
      closest.orderTo(destination);
    }
    return closest;
  }

  public String getDescription() {
    String description = "Elevators in " + this.getName() + ":\n"; // gatherer
    int elevatorNumber = 1;                                        // stepper
    for (Elevator current : this.elevators) {    // current: most-recent holder
      description = description + "  #" + elevatorNumber + ": ";
      description = description + current.getDescription() + "\n";
      elevatorNumber++;
    }
    return description;
  }
}
```

```
135: public class Test {
136:
137:   public static void main(String[] args) {
138:     Building office = new Building("Office", 10);
139:     office.addElevator();
140:     office.addElevator();
141:     Elevator test1 = office.orderElevatorToFloor(8);
142:     test1.travelTo(6);
143:     test1.closeDoor();
144:     System.out.println(office.getDescription());
145:     Elevator test2 = office.orderElevatorToFloor(2);
146:     Elevator test3 = office.orderElevatorToFloor(2);
147:     test2.closeDoor();
148:     System.out.println(office.getDescription());
149:     test3.travelTo(1);
150:     office.orderElevatorToFloor(0);
151:     System.out.println(office.getDescription());
152:   }
153:
154: }
155:
```