Tentti: T-106.1203 Ohjelmoinnin perusteet L

Tenttipäivä: 14.12.2006

Huom! Tämä on L-peruskurssin tentti.
Kurssilla T-106.1200 Ohjelmoinnin perusteet T on erillinen tenttipaperi.

Yleistä

Tehtävät 1 ja 2 muodostavat tentin pakollisen osan. Kummastakin näistä tehtävistä on saatava vähintään yksi piste päästäkseen läpi tentistä. Tehtäviä 3 ja 4 ei edes arvostella, ellei tätä pakollista osaa läpäise.

Toisaalta, jos molemmista pakollisista tehtävistä saa vähintään yhden pisteen, niin myös varmasti pääsee tentistä läpi vähintään arvosanalla 1. Arvosana määrittyy tällöin kaikkien tehtävien yhteenlasketun pistemäärän perusteella. Arvosanojen pisterajat määrätään vasta tentin jälkeen. Tenttitulokset julkaistaan kuukauden sisällä tenttipäivästä kurssien WWW-sivuilla, ja oman suorituksen arvosteluun voi tutustua erillisessä tilaisuudessa, jonka aika ja paikka tullaan myös ilmoittamaan kurssisivuilla.

Tehtävä 1 (2 pistettä)

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.

Kirjoita yksityiskohtainen selostus siitä, mitä tapahtuu, kun tätä hissiohjelmaa suoritetaan luokan `Test` käynnistysmetodista alkaen. Selostuksesta tulee käydä ilmi tarkasti se ***järjestys, jossa ohjelman suoritus etenee*** riviltä toiselle em. kolmessa luokassa. Sinun tulee myös kertoa huolellisesti ***kaikki arvot, joita muuttujat saavat*** ohjelman suorituksen eri vaiheissa. Muuttujiksi luetaan niin parametrimuuttujat, muut paikalliset muuttujat (*local variables*) kuin ilmentymämuuttujatkin (*instance variables*).

Käytä selostuksessa apuna annettuun ohjelmakoodiin merkittyjä rivinumeroita.

Huomaa: ei riitä, että selostat vain `Test`-luokassa sijaitsevan käynnistysmetodin toiminnan, vaan sinun tulee selittää kaiken ohjelma-ajon aikana suoritetun ohjelmakoodin toiminta. Joudut siis kirjoittamaan varsin paljon.

Vinkki: ohjelman tuottama tuloste on:

```
Elevator #1: floor 4, available: true
Elevator #2: floor 0, available: true
```

Tämän tehtävän tarkoitus on varmistaa, että jokainen kurssin läpäisevä opiskelija ymmärtää perusasiat Java-ohjelmakoodista ja ohjelman suorittamisesta. Tehtävä kuuluu tentin pakolliseen osaan ja siitä on saatava vähintään yksi piste läpäistäkseen tentin. Tehtävä arvostellaan asteikolla: 0 (hylätty) / 1 (hyväksytty) / 2 (erinomainen).

(LOPUT TEHTÄVÄT TOISELLA PUOLELLA)

Tehtävä 2 (2 pistettä)

Muokkaa edellisessä tehtävässä esiteltyä luokkaa `Elevator` siten, että sillä voi pitää kirjaa hissin matkustajista ja eritoten heidän painostaan. Käytä matkustajia kuvaamaan apuna liitteestä 2 löytyvää yksinkertaista luokkaa `Passenger`.

Tarkalleen ottaen sinun pitää tehdä luokkaan `Elevator` seuraavat lisäykset:

1. Lisää hisseille kaksi uutta ilmentymämuuttujaa (*instance variable*): lista hississä olevista matkustajista sekä painoraja, joka määrää, kuinka raskas kuorma hissillä saa korkeintaan olla kannettavanaan. Muista päivittää myös konstruktoria asiaankuuluvalla tavalla.
2. Lisää metodi `addPassenger`, jolla voi lisätä hissiin uuden matkustajan. Lisäys kuitenkin onnistuu vain, jos hissin painoraja ei lisäyksestä ylittyisi.

Määritä itse mielekkäät muuttujannimet, parametrit jne. sikäli kun niitä ei yllä ole määrätty. Muita lisätoimintoja kuin mainitut ei tarvitse tehdä (esim. matkustajien poistamista hissistä).

Kirjaa vastaukseesi ne muutokset, jotka luokan `Elevator` ohjelmakoodiin täytyy tehdä. Luokan valmiita metodeita ei tarvitse muuttaa.

Tämän tehtävän tarkoitus on varmistaa, että jokaisella kurssin läpäisevällä opiskelijalla on vähintäänkin auttavat taidot Java-ohjelmakoodin kirjoittamisessa. Tehtävä kuuluu tentin pakolliseen osaan ja siitä on saatava vähintään yksi piste läpäistäkseen tentin. Tehtävä arvostellaan asteikolla: 0 (hylätty) / 1 (hyväksytty) / 2 (erinomainen).

Tehtävä 3 (4 pistettä)

Esitä perusteltu mielipiteesi seuraavista väitteistä, kustakin ***korkeintaan parilla virkkeellä***.

a) Väite: Näkyvyysmääreiden – esim. `private` ja `public` – käytöllä pyritään mm. ehkäisemään ennalta ohjelmointivirheitä

b) Olkoon tutkittavana metodi:

```
public static int doStuff(int number, int another) {
  if (number < another) {
    return 0;
  } else {
    return doStuff(number - 1, another + 1);
  }
}
```

Väite: kun tätä metodia kutsutaan parametriarvoilla 5 ja 1, syntyy ns. loputon rekursio, joka johtaa ohjelman käytössä olevien muistiresurssien loppumiseen.

c) Väite: metodin määrittely staattiseksi voi olla perusteltua, jos metodin tehtävä on vain laskea ja palauttaa jotain saamiensa parametrien perusteella.

d) Väite: abstraktissa luokassa määriteltyjä olioiden ilmentymämuuttujia (*instance variable*) ei voi käyttää missään samaisen abstraktin luokan ohjelmakoodissa määritellyistä metodeista.

Tehtävä 4 (2 pistettä)

Olkoon tutkittavana eräs ohjelma, jossa on rivi:

```
this.stringArray[index] = input;
```

Oletetaan lisäksi, että on havaittu kyseisen ohjelman kaatuvan ajettaessa ajonaikaiseen poikkeukseen (*exception*). Tarkemmin ottaen ottaen ohjelma kaatuu `NullPointerException`-poikkeustilanteeseen, joka syntyy juuri yllä olevalla rivillä. Arvioi, mistä virhe voisi johtua. Mistä päin ohjelmaa lähtisit etsimään virheen pohjimmaista syytä? Koko ohjelmaa ei ole tässä esitetty; kerro perusidea lyhyesti, korkeintaan muutamalla virkkeellä.

Muista täyttää kurssipalautelomake kurssin kotisivuilla joulukuun 22. päivään mennessä! Palautteen antaminen on ensiarvoisen tärkeää kurssin jatkokehittämisen kannalta. Se on myös tämän kurssin pakollinen osasuoritus.

```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;
    }
  }

}
```

```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 (Elevator current : this.elevators) { // 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 Elevator orderElevatorToLobby() {
    return this.orderElevatorToFloor(0);
  }

  public void printElevatorDescription() {
    int elevatorNumber = 1;                   // stepper
    for (Elevator current : this.elevators) {   // current: most-recent holder
      System.out.println("Elevator #" + elevatorNumber +
                         ": floor " + current.getFloor() +
                         ", available: " + current.isAvailable());
      elevatorNumber++;
    }
  }
}

public class Test {
  public static void main(String[] args) {
    Building office = new Building("Test Office", 7);
    office.addElevator();
    office.addElevator();
    Elevator test1 = office.orderElevatorToFloor(3);
    test1.travelTo(6);
    test1.closeDoor();
    Elevator test2 = office.orderElevatorToFloor(4);
    test2.closeDoor();
    office.printElevatorDescription();
  }
}
```

```
1: public class Passenger {
2:
3:   private double weight; // most-recent holder
4:
5:   public Passenger(double weight) {
6:     this.setWeight(weight);
7:   }
8:
9:   public void setWeight(double newWeight) {
10:    this.weight = newWeight;
11:  }
12:
13:  public double getWeight() {
14:    return this.weight;
15:  }
16:
17: }
```

LITE 2