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

Tenttipäivä 15.12.2009 – Kirjoita jokaiseen paperiin oma nimesi, opiskelijanumerosi tarkistusmerkkeineen, koulutusohjelmasi, opintojaksokoodi ja –nimi, päivämäärä, Sali, **palauttamiesi papereiden lukumäärä** sekä **allekirjoituksesi.**

Yleistä

Tentissä on kaksi osiota ja kolme tehtävää. Tehtävät 1 ja 2 kuuluvat ensimmäiseen ja tehtävä 3 toiseen osioon.

Ensimmäinen osio

Ensimmäinen osio varmistaa että jokaisella kurssin läpäisevällä opiskelijalla on vähintään auttavat taidot ohjelmakoodin lukemisessa ja kirjoittamisessa. Ensimmäinen osio sisältää kaksi tehtävää, joista toinen testaa koodin lukemista ja ymmärtämistä ja toinen koodin kirjoittamista.

Kummastakin näistä kahdesta 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 on joitain puutteita tai virheitä.
- Erinomainen: Vastaus on käytännöllisesti katsoen täysin oikein

Mikäli kumpikaan 1. osion tehtävistä ei ole hylätty on osio hyväksytty ja vastaavasti myös tentti hyväksytty A-tasolla. *(Kurssiarvosanan kannalta ei ole merkitystä arvosteltiinko tehtävät hyviksi vai erinomaisiksi.)* 1. osion läpäiseminen tarkoittaa että saat kurssin kokonaisarvosanaksi harjoitusarvosanasi, kuitenkin niin että korkein saavutettava arvosana on 2.

Toinen osio

Toinen osio varmistaa että tentin suorittajalla on kurssiarvosanoihin 3-5 riittävät taidot. (ts. jatkokursseilla tarvittava osaamisen taso) Osion läpäiseminen tarkoittaa että saat kurssin kokonaisarvosanaksi harjoitusarvosanasi sellaisenaan.

Myös ensimmäinen osio tulee olla hyväksytysti suoritettu.

Taulukko 1: tentin ja harjoituspisteiden vaikutus arvosanaan (ylläoleva selitys taulukon muodossa)

Tentti			Harjoitusarvosana				
1.osio	2. osio	yhteensä	1	2	3	4	5
hylätty	-	hylätty	0	0	0	0	0
hylätty	hyväksytty	hylätty	0	0	0	0	0
hyväksytty	-	hyväksytty A-taso	1	2	2	2	2
hyväksytty	hylätty	hyväksytty A-taso	1	2	2	2	2
hyväksytty	hyväksytty	hyväksytty	1	2	3	4	5

Ensimmäinen osio

Tehtävä 1 : Koodin lukeminen **(hylätty / hyvä /erinomainen)**

Tutustu tarkasti liitteessä A, sekä alla annettuihin luokkiin `Test`, `Elevator` ja `Building`. Luokka `Building` kuvaa (suuria) taloja, joissa voi olla useita kerroksia (numeroitu nollasta alkaen) ja useita rinnakkaisia hissejä. Hissitön 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, fields, kentät).*

Käytä selostuksessa apuna annettuun ohjelmakoodiin merkittyjä rivinumeroita.

Huomaa: ei riitä että selostat vain `Test`-luokassa sijaitsevat käynnitysmetodin 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 5, available: true
Elevator #2: floor 4, available: true
```

Tämän tehtävän on tarkoitus varmistaa, että jokainen kurssin läpäisevä opiskelija ymmärtää perusasiat Java-ohjelmakoodista ja ohjelman suorittamisesta. Tehtävä kuuluu tentin ensimmäiseen osioon ja se on suoritettava tasolla Hyvä tai Erinomainen, jotta tenttisuoritus hyväksytään.

Tehtävä arvostellaan asteikolla : hylätty / hyvä /erinomainen

```
110 public class Test {
111   public static void main(String[] args) {
112      Building office = new Building("Test Office", 7);
113      office.addElevator();
114      office.addElevator();
115      Elevator test1 = office.orderElevatorToFloor(3);
116      Elevator test2 = office.orderElevatorToFloor(4);
117      test2.closeDoor();
118
119      test1.travelTo(6);
120      test1.closeDoor();
121      Elevator test3 = office.orderElevatorToFloor(5);
122      test3.closeDoor();
123      office.printElevatorDescription();
124   }
125}
```

Muu koodi liitteessä A

Ensimmäinen osio

Tehtävä 2 : Koodin kirjoittaminen **(hylätty / hyvä /erinomainen)**

Kirjoita Java-kielinen metodi nimeltään **pituusjako**, joka

- Saa parametrinaan listan jonka alkiot ovat merkkijonoja (String), sekä kokonaisluvun.
- Palauttaa kahden alkion mittaisen kokonaislukutaulukon. Taulukon ensimmäisessä alkiossa on niiden merkkijonojen lukumäärä, jotka ovat pienempiä tai yhtäsuuria kuin pituusjako-metodille parametrina annettu kokonaisluku. Toisessa palautettavan taulukon alkiossa on vastaavasti niiden alkioiden määrä jotka ovat aidosti suurempia kuin annettu luku.
- Listassa on myös null-arvoja, joita ei tule laskea mukaan kumpaankaan taulukossa palautettavaan arvoon. Ohjelma ei myöskään saa kaatua tällaisen arvon kohdatessaan.
- Esimerkiksi , jos metodin saama lista sisältää seuraavat alkiot "hei", "hoi", null, "hauki" ja metodin saama kokonaislukuargumentti on 3, metodin tulisi palauttaa taulukko, jonka ensimmäinen alkio on 2 ja toinen 1.

Tämän tehtävän on tarkoitus varmistaa, että jokainen kurssin läpäisevä opiskelija ymmärtää perusasiat Java-ohjelmakoodista ja ohjelman suorittamisesta. Tehtävä kuuluu tentin ensimmäiseen osioon ja se on suoritettava tasolla Hyvä tai Erinomainen, jotta tenttisuoritus hyväksytään.

Tehtävä arvostellaan asteikolla : hylätty / hyvä /erinomainen

MUISTA VASTATA KURSSIKYSELYYN TENTIN JÄLKEEN!

KURSSIKYSELY ON PAKOLLINEN OSASUORITUS JA

ON AUKI 23.12 ASTI.

Toinen osio

Tehtävä 3 : Koodin lukeminen ja kirjoittaminen **(hylätty / hyvä /erinomainen)**

tehtävä: **Kirjoita kaikki liitteessä B olevasta koodista puuttuvat osiot 3a – 3f**

Tässä tehtävässä on tarkoitus täydentää liitteestä B löytyvää ohjelmaa. (liitteessä C on tarvittavia luokkia) Tämän ohjelman tehtävä on kuljettaa postilähetyksiä rautateitse asemalta toiselle. Perusidea on seuraavanlainen:

- Postipaketit, luokka `Package` ovat todella yksinkertaisia olioita, niiden ainoa sinulle näkyvä ominaisuus on niiden lopullinen määränpää, jonka saa selville metodilla `getDestination()`
- Junanvaunut, `Traincar`, kuljettavat em. paketteja. Nämä lastataan asemalla jo valmiiksi ja mikä tahansa juna, jonka reitillä on vaunun kohdeasema vain poimii ne mukaansa. *(metodi loadTrain)*
- Junalla on reitti *(lista asemista)*, jolla se etenee asema kerrallaan. Juna vetää ainoastaan sellaisia vaunuja(TrainCar) jotka ovat matkalla johonkin sen loppureitin varrella.
- Kun vaunut ovat saapuneet kohteeseensa, juna hoitaa metodissa `unloadTrain()` sen että vaunut tulevat purettua.
- Jos vaunuista puretut paketit tulivat pääteasemalleen, ne säilötään asemalle. Muussa tapauksessa konsultoidaan "reittitaulua" ja sijoitetaan paketit uuteen vaunuun.
- Joka asemalla on tällainen tietorakenne (HashMap), jonka avulla paketin lopullisesta määränpäästä saa selville seuraavan aseman, johon paketti tulee seuraavaksi lähettää.
- Tämän perusteella määräytyy myös paketin tuleva vaunu.

Liitteessä toteutettavat metodit on pyritty kuvaamaan vähän tarkemmin.

Vinkki: pohdi miten metodit kutsuvat toisiaan – älä keksi pyörää uudestaan jos voit käyttää johonkin olemassaolevaa metodia.

Pyri keskittymään vain yhteen metodiin kerrallaan mikäli mahdollista.

Tämän tehtävän on tarkoitus varmistaa, että jokainen kurssin läpäisevä opiskelija osaa hahmottaa hieman perustasoa laajempaa kokonaisuutta ja kirjoittaa koodia useammasta luokasta koostuvaan ohjelmaan. Tehtävä kuuluu tentin toiseen osioon ja se on suoritettava tasolla Hyvä tai Erinomainen, jotta tentin kokonaisarvosanaksi tulisi **hyväksytty.**

Tehtävä arvostellaan asteikolla : hylätty / hyvä /erinomainen

```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);
   }

   public boolean travelTo(int destination) {
        if (destination >= 0 && destination <= this.topFloor) {
             this.closeDoor();
             this.currentFloor = destination;
             this.openDoor();
             return true;
        } else {
             return false;
        }
   }

   public boolean orderTo(int destination) {
        if (this.isAvailable() && destination >= 0 &&
                                  destination <= this.topFloor) {
             this.currentFloor = destination;
             this.openDoor();
             return true;
        } else {
             return false;
        }
   }
}
```

Liite A

```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++;
      }
   }
}
```

Liite B : Tehtävään 3 liittyvä lähdekoodi

```
import java.util.ArrayList;

public class Traincar {

    private Station             destination;
    private ArrayList<Package> packages;
    public static final int     CAPACITY = 250;

    public Traincar(Station destination) {
        3a) Kirjoita tänne koodi joka alustaa tarvittavat kentät
    }

    public boolean addPackage(Package box) {
        3b) Tämä metodi lisää uuden paketin tähän junanvaunuun. Metodi palauttaa true jos lisääminen
        onnistui. Huomaa pakettien maksimimäärä.
    }

    public void unloadTraincar(Station s) {
        3c) Tämä metodi purkaa junanvaunun sisällön muihin oikeaan suuntaan matkalla oleviin
        vaunuihin, jotka seisovat parametrina annetulla asemalla.

        Paketit jotka ovat saapuneet perille säilötään (store) vaunujen sijaan asemalle.
    }

    public Station getDestination() {
        return this.destination;
    }

    public int getFreeSize() {
        return CAPACITY - packages.size();
    }
}
```

```
import java.util.*;

public class Station {

    private ArrayList<Traincar> traincars;       // Asemalla seisovat vaunut
    private HashMap<Station, Station> routeMap; // Reittikartta, jonka avulla voidaan
                                                // hakea seuraava vaihtopaikka

    public Station() {
        // Voit olettaa että kentät traincars ja routemap alustetaan
    }   // täällä. Sinun ei tarvitse kirjoittaa tätä konstruktoria.

    public void store(Package box) {
        // Säilöö paketin asemalle. Vain paketit joiden lopullinen määränpää on
    }   // tämä asema säilötään tänne. Muut paketit laitetaan aina vaunuihin.

    public void newPackage(Package box) {
        3d) Kun asemalle saapuu uusi paketti, se tulee lastata johonkin junanvaunuun odottamaan junaa. Ikävä kyllä paketin saa
        harvoin laitettua vaunuun joka menee suoraan perille, joten kuljetus täytyy tehdä useampana pikku matkana.

        Aseman tietorakenteessa routeMap on tieto siitä, mille asemalle paketti täytyy kuljettaa, jotta matka etenisi kohti
        lopullista kohdetta. Käyttämällä loppuasemaa avaimena tähän rakenteeseen saat viittauksen seuraavaan väliasemaan.
        Lastaa paketti johonkin saamaasi välikohteeseen matkalla olevaan vaunuun tai uuteen asemalle lisättävään vaunuun jos
        löydetyt vaunut ovat kaikki täysiä, tai sopivia vaunuja ei muuten löydy.
    }

    public void loadTrain( Train train ) {
        3e) Lisää parametrina annettuun junaan kaikki vaunut jotka ovat matkalla asemille junan loppureitillä. (Huomaa että juna
        on jo kulkenut osan reitistä) Junaan lisätyt vaunut myös poistuvat asemalta.

    }

    public void unloadTrain( Train train ) {
        3f) Poistaa junasta, ja purkaa kaikki vaunut, jotka ovat saapuneet perille. Purettujen vaunujen sisältö lastataan muihin
        asemalla odottaviin vaunuihin.
    }
}
```

Liite C : Tehtävään 3 liittyvä dokumentaatio

Alla on esitelty tehtävissä ehkä tarvittavia metodeja jotka ovat olemassa valmiina. Näitä ei siis tule toteuttaa.

Train

Luokassa `Train` on kolme metodia, joita tehtävässä on pakko käyttää

`public void addTraincar(Traincar car)`
Lisää parametrina annetun vaunun junaan.

`public ArrayList<Station> getRoute()`
Palauttaa junan reitin listana asemia (Station). Reitissä on mukana myös jo ohitetut asemat.

`public ArrayList removeTraincars(Station destination)`
Poistaa ja palauttaa junan kaikki vaunut, jotka ovat matkalla annetulle asemalle.

Package

Luokassa `Package` on vain yksi metodi.

`public Station getDestination()`
Palauttaa paketin lopullisen määränpään.

HashMap <KeyType, ValueType>

HashMap-luokkaa käytetään asemalla reittien seuraavan pisteen tallentamiseen. Huomaa kuitenkin että kaikki ohjelman tarvitsema tieto on jo tallennettu rakenteeseen. Voit siis keskittyä tiedon hakemiseen.

`public ValueType get(KeyType key)`

Palauttaa annettua avainta vastaavan arvon.

`public boolean put(KeyType key, ValueType value)`

Tallentaa `HashMap`iin (hajautustauluun) avain-arvo parin. Metodi palauttaa arvon **`true`** jos kyseisellä avaimella taulussa oli jo tallennettuna arvo.

Tässä tehtävässä sekä avaimen tyyppi, että arvon tyyppi ovat tyyppiä Station

Muut luokat ja metodit joita et muista

Hätätilanteessa keksi unohtamillesi metodeille ja luokille jonkinlaiset pseudokoodinimet ja käytä niitä. (siis tee näin vain jos et muista oikeita ja mikäli voit ainoastaan siten tehdä tehtävän loppuun.)