Tentti: CSE-A1110 Ohjelmointi 1

Tenttipäivä: 16.12.2013

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.

Tässä tentissä ei saa käyttää apuvälineitä (tietokone, oppikirja yms.)

Hyvää tenttiä!

Tehtävä 1

Tutustu tarkasti liitteessä annettuun ohjelmakoodiin. Koodin luokka `City` kuvaa kaupunkeja, `Hotel` niissä olevia hotelleja ja `Loc` koordinaattipareja, joita käytetään kuvaamaan kaupunkien ja hotellien sijainteja. Lisäksi mukana on pieni testiohjelma `CityTest`.

Täydennä luokkaan `City` puuttuva toteutus metodille `averageHotelPrice`. Metodin tulee toimia dokumentaation mukaisesti eli näin:

- Metodi palauttaa niitten kyseisen kaupungin hotellien hintakeskiarvon, joilla on täsmälleen parametrina annettu määrä tähtiä ja jotka lisäksi ovat korkeintaan annetun etäisyyden päässä kaupungin keskipisteestä (linnuntietä).
- Metodi palauttaa kuitenkin nollan, jos ehdot täyttäviä hotelleja ei ole kaupungissa lainkaan.

Ohjeita ja vinkkejä:

- Huomaa, että `CityTest`-testiohjelmassa on muutama esimerkki metodin käytöstä. Koodisi tulee olla yhteensopiva tämän testiohjelman kanssa.
- Käytä jompaakumpaa annetuista `distance`-metodeista etäisyyden laskemiseen.
- Toteutustapa on muuten vapaa.
- Vastaukseksi riittää Scala-ohjelmakoodi.
- Yksittäisistä pilkku- ja pikkuvirheistä 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!

Tehtävästä saa palautteena jonkin näistä kolmesta:

- **hylätty:** Toteutus on selvästi puutteellinen ja toimimaton. Vastauksen perusteella ei saa käsitystä, että vastaaja osaa lukea ja kirjoittaa Scala-kielistä 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.)

Tehtävä 2

Tässäkin tehtävässä tutustut annettuun koodiin ja täydennät sitä. Aloitetaan kuitenkin aihepiirin kuvauksella. Kuvittele aarteenetsintäleikki, jossa erinimisiä esineitä (*item*) on piilotettu luolaan (*dungeon*). Luolassa on tunneleita, jotka jakautuvat toistuvasti kahtia, kuitenkin niin, etteivät haarat kerran erottuaan koskaan kohtaa. Voidaan ajatella, että luola muodostuu risteyksistä (*fork*) ja umpikujista (*dead end*). Kuhunkin risteykseen ja umpikujaan on sijoitettu nimetty esine. Luola voi siis näyttää siltä kuin seuraavan sivun kuvassa.

Aarteenetsintää varten esineet on sijoitettu luolaan huolellisesti: risteyksestä vasemmalle käännyttyään voi löytää vain sellaisia esineitä, joiden nimi on aakkosissa ennen risteyksessä olevan esineen nimeä. Vastaavasti oikealta löytyy vain aakkosissa myöhempiä esineitä. Esimerkiksi kuvan luolassa heti luolan suulla olevassa risteyksessä on esine *rubber chicken* ja kaikki sitä myöhemmät esineet (*sword, skull, t-shirt*) ovat siitä oikealla. Vastaavasti *skull* on *sword*ista vasemmalle ja *t-shirt* oikealle.

Koodi tehtävään 1

```
/** This singleton object contains a simple program that uses the class City. */
object CityTest extends App {
  val testHotels = Vector(new Hotel("A", new Loc(105.0, 100.0), 2,  90.0),
                          new Hotel("B", new Loc( 95.0, 100.0), 5, 900.0),
                          new Hotel("C", new Loc(100.0,  55.0), 2, 120.0),
                          new Hotel("D", new Loc( 88.8, 111.1), 2, 120.0),
                          new Hotel("E", new Loc( 95.0, 100.0), 3, 150.0))
  val testCity = new City("Test City", new Loc(100.0, 100.0), testHotels)

  // The following should print 110.0, because all the three 2-star hotels are within a distance of 50:
  println(testCity.averageHotelPrice(2, 50.0))
  // The following should print 90.0, because the only 2-star hotel so close to the center is hotel A:
  println(testCity.averageHotelPrice(2, 15.0))
  // The following should print 0.0, because there are no two-star hotels THAT close to the center:
  println(testCity.averageHotelPrice(2, 2.5))
  // The following should print 0.0; there are no four-stars hotels in the test data at all:
  println(testCity.averageHotelPrice(4, 15.0))
}

/**
 * Each instance of this class represents a city with some hotels in it.
 *
 * @param name   the name of the city
 * @param center the coordinates of the official center of the city
 * @param hotels the hotels in the city
 */
class City(val name: String, val center: Loc, val hotels: Vector[Hotel]) {

  /**
   * Determines the average price of a standard room in this city's hotels. Only hotels that have exactly
   * the given number of stars and that are located within a given radius of the city center are considered.
   * (Distances are calculated "as the crow flies"; see class Hotel.)
   *
   * @param stars                a number of stars
   * @param maxDistanceFromCenter the maximum distance from the city center
   * @return the average, or 0.0 if there are no matching hotels
   */
  def averageHotelPrice(stars: Int, maxDistanceFromCenter: Double) = {
    // TODO: method implementation missing
  }

  /** Returns a textual description of the city. */
  override def toString = this.name

}

/**
 * Each instance of this class represents a hotel.
 *
 * @param name              the name of the hotel
 * @param location          the location of the hotel
 * @param stars             the star rating of the hotel (up to a maximum of five stars)
 * @param standardRoomPrice the price of a night in one of the hotel's standard rooms
 */
class Hotel(val name: String, val location: Loc, val stars: Int, val standardRoomPrice: Double) {

  /** Returns the distance between this hotel and a given location, "as the crow flies". */
  def distance(location: Loc) = location.distance(this.location)

  /** Returns a textual representation of this hotel. */
  override def toString = this.name + " (" + this.stars + "/5): " + this.standardRoomPrice + "e"

}

/**
 * Each instance of this class represents a location, that is, a pair of coordinates on a two-dimensional plane.
 */
class Loc(val x: Double, val y: Double) {

  /** Returns the distance between this and another location, "as the crow flies". */
  def distance(another: Loc) = scala.math.hypot(another.x - this.x, another.y - this.y)

  /** Returns a textual representation of this location. */
  override def toString = x + ", " + y

}
```