Ohjelmoinnin jatkokurssi, kurssikoe 9.12.2013

Vastaa tehtäväiin 1, 2 ja 3 **erillisille** konsepteille. Kirjoita jokaiseen konseptiin kurssin nimi, kokeen päivämäärä, oma nimi, nimikirjoitus ja opiskelijanumero. Vastaukset palautetaan tehtäväkohtaisiin pinoihin.

Vaikka jättäisit johonkin tehtävään vastaamatta, tulee vastauspaperi siinäkin tapauksessa palauttaa.

Huom: kokeen viimeisellä sivulla on pääohjelmarunko sekä lista HashMap:in ja ArrayList:in yleisimmistä komennoista.

1. Käsitteistöä (6p)

   Vastaa jokaiseen kohtaan lyhyesti ja ytimekkäästi. Koko tehtävän vastauksen pituudeksi riittää noin kaksi sivua.

   (a) (3p) Javassa on joskus tarve määritellä luokalle metodit `equals` ja `compareTo`. Kerro milloin metodeja tarvitaan ja miten metodit tulee määritellä. Anna esimerkki yksinkertaisesta luokasta, jolle määrittelet em. metodit.

   (b) (2p) Mitä ovat poikkeukset (engl. exceptions) ja millaisissa tilanteissa ohjelmoija joutuu niiden kanssa tekemisiin? Miten poikkeusten käsittely ja niiden heittäminen tapahtuu? Anna pieni koodiesimerki, joka valaisee tilannetta.

   (c) (1p) Mitä ovat tapahtumankäsittelijät (engl. action listener) ja minkälaisissa tilanteissa niitä käytetään?

2. (11p) Ohjelma saa syötteeksi tiedoston, joka sisältää twiittejä eli twitter-viestejä.

   Tiedosto on muotoa:

   ```
   arto:anybody #beer in #kallio tomorrow?
   pekka:today in #tktl exam of #programming
   arto:great selection of german #beer now in #oljenkorsi
   pekka:exam now done, no more #programming for some time
   heikki:some new courses on #programming at #tktl check out the website!
   ```

   Kukin rivi siis sisältää yhden twiitin. Rivi alkaa twiitin lähettäjän nimellä jota seuraa twiitin varsinainen sisältö. Twiittien sisällössä #-merkillä merkityt sanat ovat tägejä. Esim. ensimmäiseen twiittiin liittyvät tägit **#beer** ja **#kallio**.

   Tehtävässä tehdään ohjelma, joka lukee twiitit sisältävän tiedoston ja tulostaa twiiteissä käytetyt tägit ja niiden esiintymislukumäärän.

   Ohjelma toimii seuraavasti (käyttäjän syöte *vinonnettuna*):

   ```
   give a tweetfile: tweets.txt
   programming 3
   beer 2
   tktl 2
   oljenkorsi 1
   kallio 1
   ```

   KÄÄNNÄ!

Jos käyttäjä yrittää lukea tiedoston, jota ei ole olemassa tai jota ei kyetä lukemaan, huomauttaa ohjelma tästä ja pyytää uutta tiedostoa:

```
give a tweetfile: nonexistentfile.txt
file does not exist!
give a tweetfile:
```

Saadaksesi täydet pisteet, tulee tägit tulostaa esimerkin tapaan esiintymislukumäärän mukaisessa järjestyksessä! Jos tulostus ei ole järjestyksessä, on tehtävän maksimipistemäärä 9.

Voit olettaa että käytössäsi on apuluokka ja siinä metodi, joka palauttaa syötteenään saamasta tekstimuotoisesta twiitistä siihen liittyvät tägit:

```
public public TweetParser {
    public static List<String> getTags(String tweet){
        // ...
    }
}
```

HUOM: jos et osaa toteuttaa tiedostosta tapahtuvaa syötteen lukemista, voit lukea syöterivit komentoriviltä. Tällöin tehtävän maksimipistemäärä on 8.

3. Kasveja ja eläimiä (13p)

   (a) (2p) Toteuta abstrakti luokka `Elio`.

   Eliolla on oliomuuttujina sijainnin x- ja y-koordinaatit (kokonaislukuja), sekä paino (double). Luokalla on konstruktori `Elio(int x, int y)` jonka avulla asetetaan olion sijainti. Eliolla on *abstrakti* metodi `void elele()`.

   Huom: jos et osaa tehdä abstraktia luokaa, voit tehdä eliöstä normaalin luokan, jotta pystyt jatkamaan ohjelman laajentamista seuraavissa kohdissa.

   (b) (3p) Toteuta luokka `Kasvi`, joka *perii* luokan `Elio`. Kasvin paino on aluksi 1. Kasvin metodi `elele()` kasvattaa kasvin painoa 0.1:llä.

   Kasvin `toString()`-metodi tuottaa allaolevan esimerkin mukaisen merkkijonoesityksen kasvista:

   ```
   Kasvi kasvi = new Kasvi(10,12);
   kasvi.elele();
   kasvi.elele();
   System.out.println(kasvi);
   ```

   suoritettaessa koodi tulostuu

   ```
   kasvi: sijainti(10,12), paino 1.2 kiloa
   ```

   (c) (1p) Tee rajapinta `Liikkuva`, joka määrittelee metodin `public void siirry(int x, int y)`

   (d) (4p) Tee luokka `Elain`, joka *perii* luokan `Elio` ja *toteuttaa* rajapinan `Liikkuva`.

   Eläimen paino on aluksi 10. Eläimen metodi `elele()` pienentää painoa 0.1:llä, mutta ei kuitenkaan laske painoa alle nollan. Eläimen metodi `siirry(int x, int y)` siirtää eläimen parametrien määrittelemään sijaintiin. Eläin ei kuitenkaan siirry, jos eläimen paino on nolla eli eläin on kuollut.

   KÄÄNNÄ!

Eläimen `toString()`-metodi tuottaa allaolevan esimerkin mukaisen merkkijonoesityksen eläimestä:

```
Elain elain = new Elain(10, 15);
System.out.println(elain);
elain.siirry(2, 3);
elain.elele();
System.out.println(elain);
// eletään niin kauan että paino laskee nollaan ja eläin kuolee
for (int i = 0; i < 102; i++) {
    elain.elele();
}
System.out.println(elain);
```

suoritettaessa koodi tulostuu

```
eläin: sijainti (10,15) paino 10.0
eläin: sijainti (2,3) paino 9.9
kuollut eläin: sijainti (2,3)
```

(e) (3p) Tee luokka `JaljitettavaElain` joka *perii* luokan `Elain`. Jäljitettävät eläimet ovat muuten samanlaisia kuin eläimet, mutta ne muistavat kaikki sijaintinsa. Sijainnit, eli jäljitettävän eläimen "jälki" selviää olion `toString()`-metodin tuottamasta merkkijonoesityksestä:

```
JaljitettavaElain jalkiElain = new JaljitettavaElain(0, 0);
jalkiElain.siirry(1, 1);
jalkiElain.siirry(2, 3);
jalkiElain.siirry(5, 3);
jalkiElain.siirry(2, 2);
System.out.println(jalkiElain);
```

suoritettaessa koodi tulostuu

```
eläin: sijainti (2,2) paino 10.0
  jälki: (0,0), (1,1), (2,3), (5,3), (2,2)
```

Pääohjelmarunko

```
import java.util.Scanner;

public class KoeOhjelma {

    public static void main(String[] args) {
        Scanner lukija = new Scanner(System.in);

        int luku = Integer.parseInt( lukija.nextLine() );
        String merkkijono = lukija.nextLine();
    }
}
```

java.util.HashMap

- public HashMap<K,V>() luo tyhjän HashMap-olion, jossa `K`-tyyppiset oliot ovat avaimina ja niillä on `V`-tyyppisiä olioita arvoina.
- public V put(K key, V value) tallentaa HashMap-olioon avain-arvo-parin. Palautuva viite `V`-tyyppiseen olioon on viite vanhaan olioon, joka korvattiin tai `null`.
- public V get(Object key) palauttaa viitteen `V`-tyyppiseen olioon, joka liittyy `key`-avaimeen. Jos avaimella `key` ei löydy arvoa, palautetaan arvo `null`.
- public boolean containsKey(Object key) kertoo löytyykö HashMap:ista tiettyä avainta
- public V remove(Object key) poistaa avaimen `key` ja siihen liittyneen arvon. Palauttaa viitteen `V`-tyyppiseen olioon, joka on poistettu arvo tai `null`.
- public Set<K> keySet() palauttaa kaikkien HashMapissa olevien avaimien joukon.
- public Collection<V> values() palauttaa kaikkien HashMapissa olevien arvojen joukon.

java.util.ArrayList

- public ArrayList<T>() luo uuden ArrayList-olion jossa listan elementit ovat tyyppiä `T`.
- public boolean add(T x) lisää listan loppuun olion `x`.
- public boolean contains(Object o) tarkistaa onko listassa oliota `o`.
- T get(int i) palauttaa listan alkion indeksistä `i`.