Ohjelmoinnin jatkokurssi, kurssikoe 28.4.2014

Kirjoita jokaiseen palauttamaasi konseptiin kurssin nimi, kokeen päivämäärä, oma nimi ja opiskelijanumero. **Vastaa kaikkiin tehtäviin omille konsepteilleen.**

Huom: kokeen viimeisellä sivulla on pääohjelmarunko sekä lista HashMap:in ja ArrayList:in yleisimmistä komennoista. Pääohjelmarunkoa ei tarvitse vastauksiin kirjoittaa.

Voit lyhentää komennon `System.out.println` kirjoittamalla `sout` ja käyttää muitakin lyhenteitä, mikäli selität lyhenteen jokaisessa paperissa, jossa sitä käytät.

1. (3p) Selitä lyhyesti seuraavat käsitteet. Anna jokaisen soveltamisesta konkreettinen koodiesimerkki. Koko tehtävän vastausten sopiva yhteenlaskettu pituus on 1-2 sivua.

   (a) Poikkeus

   (b) Rajapinta `Comparable`

   (c) tapahtumankäsittelijä ja `ActionListener`

2. (8p) Ohjelma saa syötteekseen tiedoston, joka sisältää joukon pelaajien otteluissa saamia pistemääriä.

   Tiedosto on muotoa

   ```
   pekka;0;1
   arto;2;0
   jukka;1;1
   pekka;1;0
   arto;1;3
   pekka;1;1
   jukka;0;3
   mikko;0;0
   ```

   Tiedoston yksi rivi kertoo pelaajan yksittäisessä pelissä tekemät maalit ja syötöt. Esimerkin ensimmäinen rivi tarkoittaa, että *pekka* ei tehnyt ottelussa yhtään maalia mutta sai yhden syöttöpisteen. Toinen rivi taas tarkoittaa, että *arto* teki ottelussa kaksi maalia mutta ei saanut yhtään syöttöpistettä. Kolmas rivi taas tarkoittaa että *jukka* teki yhden maalin ja sai yhden syöttöpisteen.

   Luettuaan syötteen, muodostaa ohjelma *pistepörssin*, eli laskee yhteen pelaajan kaikista otteluissa saamat maalit ja syöttöpisteet ja tulostaa pelaajat ja pelaajan pisteet pisteiden mukaan järjestettynä.

   Ohjelman toiminta voi näyttää esim seuraavalta:

   ```
   tiedosto: pisteet.txt
   pistepörssi:
   arto 6
   jukka 5
   pekka 4
   mikko 0
   ```

   Esimerkissämme *arto* on ollut mukana kahdessa ottelussa

   ```
   arto;2;0
   arto;1;3
   ```

saaden ensimmäisestä 2 pistettä (2 maalia + 0 syöttöä) ja toisesta 4 pistettä (1 maali + 3 syöttöä), eli yhteensä artolla on 6 pistettä.

Täysiin pisteisiin edellytetään pelaajien tulostamista esimerkin järjestyksessä. Jos ohjelmasi tulostaa pelaajat jossain muussa järjestyksessä, on tehtävän maksimipistemäärä 6.

Jos käyttäjä antaa tiedoston, jota ei voida lukea, raportoi ohjelma tästä ja lopettaa:

```
tiedosto: eiolemassa.txt
tiedostoa ei voida lukea
```

Vihje: saat pilkottua tiedostossa olevat merkkojonot seuraavasti:

```
String rivi = // lueRiviTiedostosta, oletetaan että rivin sisältö on arto;2;0
String[] palat = rivi.split(";");
// palat[0] on nyt merkkijono arto
// palat[1] on nyt merkkijono 2
// palat[2] on nyt merkkijono 0
```

3. Käytössäsi on seuraava abstrakti luokka

```
public abstract class Tyontekija {
    private String nimi;

    public Tyontekija(String nimi) {
        this.nimi = nimi;
    }

    public abstract int palkka();

    public abstract String tyosuhde();

    @Override
    public String toString() {
        return nimi+" "+tyosuhde()+", ansiot: "+palkka()+" euroa";
    }
}
```

Jos et muista mitä abstrakti luokka tarkoittaa, voit kuvitella että kyseessä on normaali luokka ja abstraktien metodien toteutukset eivät tee mitään, eli ovat seuraavanlaisia:

```
    public String tyosuhde() {
        return "";
    }
```

Perivien luokkien on toteutettava abstraktit metodit jotta ohjelma toimisi.

(a) (2p) Tee luokka `Harjoittelija`, joka perii luokan `Tyontekija`. Kun suoritetaan koodi

```
Tyontekija jarmo = new Harjoittelija("Jarmo");
System.out.println(jarmo);
```

tulee tulostuksen olla seuraava:

```
Jarmo harjoittelija, ansiot: 0 euroa
```

Saadaksesi täydet pisteet tulee luokan hyväksikäyttää mahdollisimman paljon abstraktilta luokalta perimiään asioita.

(b) (3p) Tee luokka `Viranhaltija`, joka myös perii luokan `Tyontekija`. Kun suoritetaan koodi

```
Tyontekija arto = new Viranhaltija("Arto", 3500);
System.out.println(arto);
```

oikein toteutetulla luokalla tulostus on

```
Arto virka, ansiot: 3500 euroa
```

Konstruktorin `public Viranhaltija(String nimi, int palkka)` toinen parametri siis kertoo viranhaltijan palkan.

Kuten edellä, saadaksesi täydet pisteet tulee luokan hyväksikäyttää mahdollisimman paljon abstraktilta luokalta perimiään asioita.

(c) (3p) Tee luokka `Sivutoiminen`, joka myös perii luokan `Tyontekija`.

Kun suoritetaan seuraava koodi

```
Sivutoiminen pekka = new Sivutoiminen("Pekka", 20);
pekka.lisaaTunteja(2);
System.out.println(pekka);
pekka.lisaaTunteja(3);
System.out.println(pekka);
pekka.lisaaTunteja(5);
System.out.println(pekka);
```

tulee tulostua:

```
Pekka sivutoiminen 20 euroa/h, ansiot: 40 euroa
Pekka sivutoiminen 20 euroa/h, ansiot: 100 euroa
Pekka sivutoiminen 20 euroa/h, ansiot: 200 euroa
```

Konstruktorin `public Sivutoiminen(String nimi, int tuntipalkka)` toinen parametri siis kertoo sivutoimisen työntekijän tuntipalkan.

Luokan olioilla on myös metodi `public void lisaaTunteja(int maara)`, jonka avulla sivutoimiselle työntekijälle lisätään tehtyjä työtunteja. Sivutoimisen ansiot ovat luonnollisestikin tuntipalkka kertaa tehtyjen työtuntien määrä.

Ja jälleen sama huomautus, saadaksesi täydet pisteet tulee luokan hyväksikäyttää mahdollisimman paljon abstraktilta luokalta perimiään asioita.

4. Luodaan joukko kokonaisluvuilla operoivia luokkia. Kaikki tämän tehtävän luokat toteuttavat seuraavan rajapinnan:

```
public interface Funktio {
    int laske(int luku);
}
```

Luokan instanssit ovat siis "funktioita", eli ne kuvaavat eli muokkaavat parametrinaan saamansa kokonaisluvun jonkun säännön perusteella toiseksi kokonaisluvuksi.

(a) (2p) Tee ensin rajapinnan toteuttava luokka `Itseisarvo`. Luokka toimii seuraavasti:

```
int[] luvut = {3, -7, 2, -11, -2};
Funktio itseisarvo = new Itseisarvo();

for (int luku : luvut) {
    System.out.print(itseisarvo.laske(luku)+ " ");
}
```

tulostuu

```
3 7 2 11 2
```

Metodi `laske` siis palauttaa parametrina saamansa luvun itseisarvon, eli luvun itsensä jos luku on positiivinen ja luvun kerrottuna miinus yhdellä jos luku on negatiivinen.

(b) (3p) Jatka tekemällä rajapinnan toteuttava luokka `Kertoja`. Luokka toimii seuraavasti:

```
int[] luvut = {3, -7, 2, -11, -2};
Funktio kolmellaKertoja = new Kertoja(3);

for (int luku : luvut) {
    System.out.print(kolmellaKertoja.laske(luku)+
    " ");
}
```

tulostuu

```
9 -21 6 -33 -6
```

Metodi `laske` siis kertoo parametrina saamansa luvun Kertoja-olion konstruktoriparametrina saamalla luvulla. Esimerkissä olio luotiin kutsumalla `new Kertoja(3)`, eli metodi `laske` palauttaa parametrina saamansa luvun kerrottuna kolmella.

(c) (3p) Tee sitten rajapinnan toteuttava luokka `Summaaja`. Luokka toimii seuraavasti

```
int[] luvut = {1, 2, 3, 4, 5, 6, 7};
Funktio summaaja = new Summaaja();

for (int luku : luvut) {
    System.out.print(summaaja.laske(luku)+ " ");
}
```

tulostuu

```
1 3 6 10 15 21 28
```

Metodi `laske` palauttaa summan kaikista niistä luvuista, joilla kyseisen olion metodia on kutsuttu. Esimerkissä kutsutaan ensin `laske(1)` ja metodi palauttaa 1. Seuraava kutsu on `laske(2)`, metodi palauttaa 3 koska sitä on tässä vaiheissa kutsuttu parametreilla 1 ja 2. Seuraava kutsu on `laske(3)`, joten metodi palauttaa 1+2+3=6.

Toinen esimerkki metodin toiminnasta:

```
int[] luvut = {2, 2, 2, 2, 1, 1};
Funktio summaaja = new Summaaja();

for (int luku : luvut) {
    System.out.print(summaaja.laske(luku)+ " ");
}
```

tulostuu

```
2, 4, 6, 8, 9, 10
```

(d) (3p) Tee lopuksi rajapinnan toteuttava luokka `Yhdistelma`, jonka avulla on mahdollista yhdistää kahden muun funktion toiminta.

Seuraavassa käyttöesimerkki:

```
int[] luvut = {2, -3, 7, -99, 0};
Funktio kertaaKaksi = new Kertoja(2);
Funktio itseisarvo = new Itseisarvo();
Funktio yhdistelma = new Yhdistelma(kertaaKaksi, itseisarvo);

for (int luku : luvut) {
    System.out.print(yhdistelma.laske(luku)+ " ");
}
```

tulostuu

```
4 6 14 198 0
```

Esimerkissä muodostetaan yhdistelmä, jonka metodi `laske` kertoo ensin parametrinaan saamansa luvun kahdella ja ottaa vielä tuloksesta itseisarvon. Eli esim. kutsuttaessa `laske(-3)` palauttaa metodi 6.

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`.

pääohjelmarunko

```
public class Ohjelma {
    public static void main(String[] args) {
       Scanner lukija = new Scanner(System.in);
       // int luku = Integer.parseInt( lukija.nextLine() );
       // String rivi = lukija.nextLine();
    }
}
```