# T-106.213 Ohjelmoinnin peruskurssi L1:n tentti

**Tenttipäivä: 10.12.2004, laatija: Juha Sorva**

Kirjoita kunkin vastauspaperisi alkuun kurssitunnus, päiväys, oma nimesi, opiskelijanumerosi (selkeästi) ja allekirjoituksesi. Eri tehtäville tässä tehtäväpaperissa lueteltuja pistearvoja saatetaan vielä muuttaa arvostelun yhteydessä. Tentin arvosanarajat määritetään vasta tentin jälkeen. Tenttitulokset julkaistaan kuukauden sisällä tenttipäivästä kurssin WWW-sivuilla, ja oman suorituksen arvosteluun voi tutustua erillisessä tilaisuudessa, jonka aika ja paikka tullaan myös ilmoittamaan kurssisivuilla.

## Tehtävä 1 (40 pistettä/100)

Lue seuraavat väittämät huolellisesti ja kerro ovatko ne totta vai tarua. Anna kussakin kohdassa myös lyhyt selitys/perustelu vastauksellesi. Selityksen toivottu pituus on yksi virke per kohta - tarkoitus ei ole antaa tyhjentävää selontekoa kustakin aiheesta vaan lähinnä osoittaa, että et ole vastannut vain arvaamalla. Jos keksit väitteen todeksi näyttävän esimerkin tai sen taruksi osoittavan vastaesimerkin, niin esimerkki riittää perusteluksi. Kustakin kohdasta saa oikealla vastauksella viisi pistettä. Jos perustelu puuttuu tai ei ole hyväksyttävä, jäävät pisteet kyseisestä kohdasta saamatta.

a) Ohjelmoija voi määritellä Java-kieltä käyttäen itse omia tietotyyppejä (engl. *data type*).

b) Kun konstruktorille annetaan parametreja, pitää parametriarvot sijoittaa olion kenttien (ilmentymämuuttujien, engl. *field*, *instance variable*) arvoiksi.

c) Jos `toimi` on kuormitettu (engl. *overloaded*) metodi(nimi), niin seuraavat kaksi peräkkäistä koodiriviä voivat (ainakin joillain muuttujien arvoilla) johtaa eri `toimi`-nimisten metoditoteutusten suorittamiseen

```
this.toimi((a && b) || c);
this.toimi(!((a && d) || !c));
```

d) Konstruktorin määrittely abstraktille luokalle voi olla perusteltua.

e) Kun tätä koodinpätkää yritetään käyttää, se aiheuttaa eräänlaisen poikkeusolion (engl. *exception*) syntymisen.

```
String teksti = "kissa";
Object olio = teksti;
teksti = olio;
```

f) Olkoon käytössä ohjelman käsittelemää dataa kuvaava luokka L, jonka ilmentymiä halutaan varastoida linkitettyyn listaan (engl. *linked list*). Jotta tämä onnistuisi, luokkaan L on lisättävä tätä tarkoitusta varten sopiva kenttä (ilmentymämuuttuja; engl. *field*, *instance variable*).

g) Parametrimuuttujat ovat ohjelman rakennetta (luokkakokonaisuutta) suunniteltaessa oleellisempia kuin muut paikalliset muuttujat.

h) Javan graafisissa käyttöliittymäkirjastoissakin käytetyn tapahtumankäsittelyn (engl. *event handling*) ydinajatus on, että laaditaan tapahtumaolioita luovalle (tapahtumien lähteenä toimivalle) luokalle aliluokka, jossa toteutetaan sopiva tapahtumankuuntelijarajapintaluokka (esim. `ActionListener`).

## Tehtävä 2 (20 pistettä/100)

Tutustu liitteessä 1 kuvattuun luokkaan `Henkilo` ja vastaa siihen liittyviin osatehtäviin, jotka on annettu alla. Kukin osatehtävä on viiden pisteen arvoinen. Henkilöluokan käyttämää `Ominaisuus`-luokkaa ei ole tässä tarkasti kuvattu, mutta siitä ei tarvitse tietää muuta kuin että siinä on metodi `public String kerroNimi()`, joka palauttaa ominaisuuden nimen.

a) Luokassa on yksi virhe joka estää sen järkevän käytön. Mistä virheestä on kysymys, mitä siitä seuraa ja miten sen voi korjata? (Virhe ei ole `teeJotainTuntematonta`-metodissa.)

b) Kirjoita `Henkilo`-luokan `kerroOminaisuus`-metodin julkinen dokumentaatio. Ts. dokumentoi `Henkilo`-luokan julkinen liityntä mainitun metodin osalta. (Jos osaat, niin voit kirjoittaa Javadoc-dokumentaatiokommentin, mutta muukin selkeä esitysmuoto kelpaa mainiosti.)

c) Millaisia (Java perus-API:stakin löytyviä) perusvälineitä apuna käyttäen henkilöluokka voitaisiin toteuttaa kätevämmin tai muuten paremmin, kuitenkin muuttamatta luokan julkista liityntää (eli julkista rajapintaa, engl. *public interface*)? Perustele lyhyesti. Java API:n luokkien tai niiden metodien nimiä ei ole tarpeen esittää.

d) Metodi `teeJotainTuntematonta` hoitaa hommansa iteratiivisesti. Laadi metoditoteutus, joka hoitaa saman tehtävän rekursiivisesti. Nimeä metoditoteutuksesi niin, että antamasi nimi kuvaa paremmin metodin tehtävää.

## Tehtävä 3 (28 pistettä/100)

Kuvattakoon erilaisia tavaroita luokan `Tavara` ja sen aliluokkien avulla. `Tavara`-luokka ja pari esimerkkiä sille laadittavista aliluokista on oleellisilta osiltaan kuvattu liitteessä 2. Oletettakoon, että aliluokkia on olemassa paljon muitakin kuin liitteessä mainitut `Vaatekappale` ja `Kirja`. Samaisessa liitteessä on myös pari esimerkkiä tavaroihin liittyvistä toiminnoista (metodit `maaritaKuljetushinta`, `maaritaTekijanNimi`), jotka on toteutettu tavaroita kuvaavien luokkien ulkopuolelle. Oletettakoon jälleen, että myös vastaavia, erilaisia tavaroihin kohdistettavia toimintoja on olemassa paljon muitakin kuin liitteessä esitetyt.

a) Metodeissa `maaritaKuljetushinta` ja `maaritaTekijanNimi` esiintyy molemmissa rakenne (`if-else-if-else-`), joka käy läpi kaikki erilaiset tavaratyypit ja toimii tietyllä tavalla parametrinsa tavaratyypistä riippuen (muistutus: Javan `instanceof`-operaattori selvittää onko olio tiettyä tyyppiä vai ei). Kirjoita tavaroita kuvaavista luokista (`Tavara`, `Vaatekappale`, `Kirja`) uusitut versiot siten, että tavaraoliot osaavat itse huolehtia kuljetushinnan ja tekijän nimen määrittämisestä ja tarjoavat tähän julkiset metodit, eikä kaikkien eri vaihtoehtojen läpikoluamiseen `if-` (tai `switch`-)lauseella siis ole tarvetta. (16p)

b) Liitteen ohjelmakoodissa toteutettiin tavaroihin liittyvät toiminnot tavaroita kuvaavien luokkien ulkopuolelle. Tämän tehtävän a-kohdassa taas toiminnot toteutettiin ko. luokkiin itseensä. Kumpi näistä menettelytavoista on enemmän "olio-ohjelmoinnin hengen mukainen" eli noudattaa lähemmin olio-ohjelmoinnin pääperiaatteita? Perustele lyhyesti. (4p)

Oletetaan, että ohjelman eri luokkien toteutuksista on vastuussa eri ohjelmoijia. Kukin ohjelmoija tuntee toisten ohjelmoijien laatimista luokista vain niiden julkisen liitynnän (eli julkisen rajapinnan, engl. *public interface*).

c) Ajatellaan tilannetta, jossa ohjelmaan halutaan lisätä uusi tavaratyyppi (uusi `Tavara`-luokan aliluokka). Vertaa liitteen ohjelmakoodin lähestymistapaa ja itse a-kohdassa laadittua versiota sen suhteen, kuinka kätevää tai vaivalloista tällaisen lisäyksen tekeminen on. Perustele väitteesi lyhyesti. (4p)

d) Ajatellaan tilannetta, jossa ohjelmaan halutaan lisätä uusi tavaroihin liittyvä toiminto. Vertaa liitteen ohjelmakoodin lähestymistapaa ja itse a-kohdassa laadittua versiota sen suhteen, kuinka kätevää tai vaivalloista tällaisen lisäyksen tekeminen on. Perustele väitteesi lyhyesti. (4p)

## Tehtävä 4 (12 pistettä/100)

Liitteessä 3 on esitetty tämän syksyn luentomateriaalia koskevaa kritiikkiä (hieman alkuperäisestä muokattuna), jonka joku opiskelija lähetti kurssin anonyymin pikapalautelomakkeen kautta. Liitteestä löytyvät myös ne luentomateriaalin sivut, joita kritiikki koskee. Ovatko kirjoittajan esittämät faktat kunnossa? Osuuko kritiikki nähdäksesi oikeaan (eli onko esimerkissä jotain arveluttavaa), vai onko se väärässä, vai onko totuus jossain sillä välillä? *Miksi?* Huomaa, ettei tarkoitus ole selittää yleisesti sitä, mikä kyseisessä luentomateriaalissa on mielestäsi hyvää tai huonoa, vaan ottaa kantaa oheiseen kritiikkiin liittyviin asioihin.

**Muista käydä täyttämässä kurssikyselylomake kurssisivuilla. Kyselyyn vastaamisesta saa 200 harjoitustehtäväpistettä.**

## e 1: Henkilöluokka. Liittyy tehtävään 2

```
Lic class Henkilo {

ublic static final int OMINAISUUKSIA_ENINTAAN = 1000;
rivate String nimi;
rivate Henkilo isa;
rivate Ominaisuus[] ominaisuudet;
rivate int ominaisuuksia;

public Henkilo(String nimi, Henkilo isa) {
  this.nimi = nimi;
  this.isa = isa;
  Ominaisuus[] ominaisuudet = new Ominaisuus[Henkilo.OMINAISUUKSIA_ENINTAAN];
  this.ominaisuuksia = 0;
}

public Henkilo(String nimi) {
  this(nimi, null);
}

public String kerroNimi() {
  return this.nimi;
}

public boolean lisaaOminaisuus(Ominaisuus uusiOminaisuus) {
  if (this.ominaisuuksia < this.ominaisuudet.length) {
    this.ominaisuudet[this.ominaisuuksia] = uusiOminaisuus;
    this.ominaisuuksia++;
    return true;
  } else
    return false;
}

public Ominaisuus kerroOminaisuus(String ominaisuudenNimi) {
  for (int indeksi = 0; indeksi < this.ominaisuuksia; indeksi++) {
    Ominaisuus tutkittava = this.ominaisuudet[indeksi];
    if (tutkittava.kerroNimi().equals(ominaisuudenNimi))
      return tutkittava;

  }
  return null;
}

public Henkilo teeJotainTuntematonta() {
  Henkilo henkilo = this;
  while (henkilo.isa != null) {
    henkilo = henkilo.isa;
  }
  return henkilo;
}
}
```

## Liite 2: Ohjelmakoodia. Liittyy tehtävään 3.

Luokat kuvattu vain tehtävänannon kannannalta oleellisin osin.

Tavaroita kuvaavat luokat:

```
public abstract class Tavara {
   // jotain kaikkien tavaroiden yhteisiä piirteitä...
}

public class Vaatekappale extends Tavara {
   // ...
   public String kerroKoko() {
     return this.koko;
   }

   public String kerroMerkkinimi() {
     return this.merkkinimi;
   }
   // ... jne. muita vaatteille ominaisia piirteitä
}

public class Kirja extends Tavara {
   // ...
   public Kirjailija kerroKirjailija() {
     return this.tekija;
   }

   public String kerroNimi() {
     return this.nimi;
   }
   // ... jne. muita kirjoille ominaisia piirteitä
}
```

Tavaroihin liittyviä toimintoja (muita metodeita, joista käsin tavaraluokkia käytetään).

```
public class JokuMuuLuokka {
   // ...
   public String maaritaKuljetushinta(Tavara tavara) {
     if (tavara instanceof Vaatekappale)
       return 5;
     else if (tavara instanceof Kirja)
       return 3;
     else
       // ... jne. määritellään kuljetushinta muille tavarakategorioille
       // samaan tyyliin
   }
   // ...
}

public class VielaJokuLuokka {
   // ...
   /* Määrittää tavaran tekijän nimen: se, mitä "tekijällä" tarkoitetaan riippuu
    * tavaratyypistä: vaatteille niiden valmistaja, kirjoille kirjailija jne.    */
   public String maaritaTekijanNimi(Tavara tavara) {
     if (tavara instanceof Vaatekappale)
       return ((Vaatekappale)tavara).kerroMerkkinimi();
     else if (tavara instanceof Kirja)
       return ((Kirja)tavara).kerroKirjailija().kerroNimi();
     else
       // ... jne. käsitellään muut tavarakategoriat samaan tyyliin
   }
   // ...
}
```