OHJ-1450 Olio-ohjelmoinnin jatkokurssi (Matti Rintala)

Tentti 20.5.2009

Tentissä ei saa käyttää ylimääräistä kirjallista materiaalia, laskimia, tietokoneita tai muita lunttausvälineitä.

Muutama sana tenttivastauksen kirjoittamisesta:

1. Mieti etukäteen esim. ranskalaisilla viivoilla vastauksesi pääkohdat ja lajittele ne johdonmukaiseen järjestykseen — älä kirjoita yhteen pötköön kaikkea mieleen tulevaa, se on varma tapa unohtaa olennaista.
2. Muista vastata kaikkiin tehtävän kysymyksiin, täysiä pisteitä ei voi saada jos kaikkiin kysyttyihin asioihin ei ole vastattu.
3. Jos vastaus vaatii ohjelmakoodin kirjoittamista, sen **ei** tarvitse olla pilkulleen syntaksiltaan oikein.

1. Seuraavassa on joukko väittämiä olio-ohjelmoinnista ja C++:sta. Mitkä väittämät ovat oikein, mitkä väärin? Perustele mielestäsi vääristä väittämistä n. 3–6 rivillä, *miksi/miten* väittämä on väärin ja miten asia todellisuudessa on.

   a) Abstraktit kantaluokat (*abstract base classes*) ovat luokkia, joista ei enää voi periyttää aliluokkia.

   b) Julistamalla itsensä toisen luokan ystäväksi (*friend*) luokka saa pääsyn tämän toisen luokan private-osaan.

   c) Suunnittelumallit (*design patterns*) ovat oliosuunnitelmaesimerkkejä, jotka on tarkoitettu helpottamaan ohjelman suunnittelussa alkuunpääsemistä.

   d) Kun funktion paluutyyppi on luokka, luodaan kutsujan puolelle väliaikaisolio, joka alustetaan return-lauseella palautettavasta oliosta kopiorakentajalla.

   e) Poikkeuksien hyötynä paluuarvona välitettyyn virhekoodiin verrattuna on se, että paluukoodi voi jäädä huomioimatta, mutta poikkeukset eivät.

   f) Pysyvyys- ja vaihtelevuusanalyysissä tutkitaan, mitkä jäsenfunktiot luokassa voidaan varustaa const-määreellä.

2. Lyhyehköjä esseitä (mutta esseitä kuitenkin, vastaa kattavasti).

   a) Mitä on viipaloituminen *(slicing)* olio-ohjelmoinnissa? Mitä ongelmia se tuo C++:aan ja millaisia keinoja on sen estämiseen?

   b) Mitä ja mitkä ovat poikkeustakuut, mihin niitä käytetään ja miten ne helpottavat luotettavan ohjelman suunnittelemista?

   c) Mistä sopimussuunnittelussa oikein on kyse? Mitä ovat luokkainvariantit, esi- ja jälkiehdot?

KÄÄNNÄ!

3. Selitä (max. 6 riviä/kohta) seuraavat olio-ohjelmoinnin ja C++:n käsitteet ja mitä hyötyä/haittaa niistä on olio-ohjelmoinnissa. *Älä* selitä niistä pelkkää syntaksia tms, vaan kerro etupäässä, mitä ko. käsitteet *tarkoittavat*.

   a) Dynaaminen sitominen *(dynamic binding)*
   b) Syväkopiointi *(deep copying)*
   c) Poikkeushierarkia *(exception hierarchy)*
   d) Rajapintaluokka *(interface class)*
   e) Luokkafunktio *(static member function)*
   f) Funktio-olio *(function object, functor)*

4. Sopimussuunnittelu, poikkeukset ja vähän periytymistä.

   a) Alla oleva listaus esittelee "yksinumeroisen murtolukuluokan", jonka olioiden on tarkoitus esittää murtolukuja a/b, jossa a ja b ovat molemmat yksinumeroisia kokonaislukuja. Kirjoita luokalle luokkainvariantti (sanallinen tai kaavoina, kunhan asia käy selväksi.)
   b) Lisää luokan jäsenfunktioihin tarvittava virhekäsittely. Koko listausta ei tarvitse kopioida, vaan ainoastaan uudet tai muuttuneet rivit (ilmoita rivinumerot niin, että riveistä näkee minne ne kuuluvat).
   c) Kirjoita luokan operaatioille esi- ja jälkiehdot. (Jälleen sanallisena tai kaavoina, kunhan ovat yksityiskohtaisia ja selkeitä.)
   d) Ohjelmaan haluttaisiin myös normaali murtolukuluokka, jonka osoittajan ja nimittäjän arvoja ei ole rajoitettu yksinumeroisiksi. Kumpaan suuntaan periytyminen tapahtuisi YksMurto-luokasta? Perustele!

```
1  // class YksMurto
   {
3    // Alustus kokonaisluvusta
     YksMurto(int i)
5      : oso_(i), nim_(1)
     {
7    }
     // Alustus osoittaja & nimittäjä
9    YksMurto(int o, int n)
       : oso_(o), nim_(n)
11   {
     }
13   // Purkaja
     ~YksMurto()
15   {
     }
17   // Lisätään kokonaisluku
     void lisaa(int i)
19   { // Lisätään i lavennettuna
       oso_ += i * nim_;
21   }
     // Jaetaan murtoluvulla
23   void jaa(YksMurto const& m)
     { // Jaetaan kertomalla ristiin
25     oso_ *= m.nim_;
       nim_ *= m.oso_;
27     supista(); // Supistetaan
     }
29   // Muutetaan käänteisluvukseen
     void kaanna()
31   { // Vaihdetaan osoittaja ja nimittäjä
       int vanha_oso = oso_;
33     oso_ = nim_;
       nim_ = vanha_oso;
35   }
     // Palautetaan arvo liukulukuna
37   double annaLiukulukuna() const
     {
39     return static_cast<double>(oso_) /
              static_cast<double>(nim_);
41   }
   private:
43   // Supistaa osoittajan ja nimittäjän
     // niin, että ne ovat mahd. pieniä
45   void supista(); // Toteutus ohitetaan
     // Luvun arvo on oso_ / nim_
47   int oso_; // Osoittaja
     int nim_; // Nimittäjä
49 };
```