TIE-02400 Ohjelmoinnin tekniikat

(Matti Rintala)

Tentti 21.5.2014

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. Termit

   Selitä (max. 7 riviä/kohta) seuraavat käsitteet ja mitä hyötyä/haittaa niistä on. ***Älä*** selitä niistä pelkkää syntaksia tms, vaan kerro etupäässä, mitä ko. käsitteet *tarkoittavat*.

   a) Poikkeushierarkia *(exception hierarchy)*
   b) Yksikkötestaus *(unit testing)*
   c) Tapahtumapohjainen ohjelmointi *(event driven programming)*
   d) Qt:n "signaalit"ja "slotit"
   e) Syväkopiointi *(deep copying)*
   f) C++:n älykkäät osoittimet *(smart pointers)*

2. Sopimussuunnittelu *(Contract Programming)*.

   a) Mistä sopimussuunnittelussa on kyse, ts. miten se helpottaa ohjelmointia?
   b) Mitkä ovat sopimussuunnittelun keskeiset käsitteet ja mitä ne *tarkoittavat*?
   c) C++:n `vector`-luokan indeksointi-operaattorin `[]` esiehtona on, että käytetty indeksi on pienempi kuin alkioiden lukumäärä. Miten operaation **jälkiehtoa** pitäisi muuttaa, jos esiehtoa lievennetään niin, että indeksiksi käy mikä tahansa kokonaisluku?
   d) Kirjoita (sanallisesti tai kaavalla, kuitenkin mahdollisimman täsmällisesti) seuraavien funktioiden esi- ja jälkiehdot. Tehtävässä `v` on globaali muuttuja (hyi) tyyppiä `std::vector<int>`.

      i. `double f1(double x) { return x/x; }`
      ii. `unsigned int f2(string s) { return s.length(); }`
      iii. `int f3(int i) { return v[i+1]; }`
      iv. `void f4() { v.push_back(v[0]); }`
      v. `int f5(int i) { return v.push_back(v.at(i)); }`
      vi. `int f6() { return v[0]/v[1]; }`

KÄÄNNÄ!

3. Periytyminen

   a) Mitä ovat periytyminen ja luokkahierarkiat? Miten periytyminen vaikuttaa luokkiin (ts. miten periytyminen näkyy ohjelmakoodin tasolla)?

   b) Mitä ovat dynaaminen sitominen ja virtuaalifunktiot? Millaisissa tilanteissa ja miten ne helpottavat ohjelmointia?

   c) Mitä ovat abstraktit kantaluokat? Mitä hyötyä sellaisista on?

4. Poikkeukset ja vähän muutakin

   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) Ohjelmaan haluttaisiin myös normaali murtolukuluokka, jonka osoittajan ja nimittäjän arvoja ei ole rajoitettu yksinumeroisiksi. Voisiko tämän uuden luokan periyttää YksMurto-luokasta? Perustele!

```
1  // class YksMurto
   {
3    // Alustus kokonaisluvusta
     YksMurto(int i)
5      : oso_(i), nim_(1)
     {
7    }
     // Alust. osoittaja & nimittaja
9    YksMurto(int o, int n)
       : oso_(o), nim_(n)
11   {
     }
13   // Purkaja
     ~YksMurto()
15   {
     }
17   // Lisataan kokonaisluku
     void lisaa(int i)
19   { // Lisataan 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 kaanteisluvukseen
     void kaanna()
31   { // Vaihdetaan osoittaja ja nimittaja
       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 nimittajan
     // niin, etta ne ovat mahd. pienia
45   void supista(); // Toteutus ohitetaan
     // Luvun arvo on oso_ / nim_
47   int oso_; // Osoittaja
     int nim_; // Nimittaja
49 };
```