TIE-02400 Ohjelmoinnin tekniikat

(Matti Rintala)

Tentti 3.9.2014

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

   a) Yksikkötestaus *(unit testing)*
   b) Qt:n "signaalit"ja "slotit"*(signals and slots in Qt)*
   c) Keskeytyskohta *(breakpoint)*
   d) C++:n älykkäät osoittimet *(smart pointers)*
   e) Luokkainvariantti *(class invariant)*
   f) Syväkopiointi *(deep copying)*

2. Periytyminen.

   a) Mitä on periytyminen? Luettele 4 mielestäsi *olennaisinta* periytymiseen liittyvää termiä ja selitä jokaisesta 2-3 lauseella, mitä termi tarkoittaa.
   b) Mitä ovat rajapintaluokat (*interface classes*) ja miten ne eroavat tavallisista luokista? Mihin niitä voi käyttää ja miksi niitä tarvitaan?

3. Modulaarisuus. Vastaa lyhyehkösti ja ytimekkäästi, mutta kuitenkin koko kysymykseen.

   a) Millä tavoin poikkeukset ja niitä käyttävä virheenkäsittely helpottaa modulaarisuutta eli sitä, että ohjelma voidaan jakaa toisistaan mahdollisimman riippumattomiin osiin?
   b) Mitä on sopimussuunnittelu? Miten sopimussuunnittelu auttaa modulaarisuutta? Mitä hyötyä siitä saadaan ohjelman jakamisessa riippumattomiin osiin?
   c) Entä miten periytyminen helpottaa modulaarisuuden saavuttamista?

KÄÄNNÄ!

4. Koodinlukutehtävä

   a) Alla oleva listaus kuvaan yksinkertaisen luokan, joka käsittelee etu- ja sukunimiä. Siinä on kerrottu operaatioiden esi- ja jälkiehdot. Mitä virheitä niistä löydät (koodiin verrattuna) ja miten ne korjaisit? *Huom,* saat koskea vai esi- ja jälkiehtoihin, et itse koodiin

   b) Nyt vastaavasti esi- ja jälkiehdot on kiinnitetty alkuperäisiksi, mutta koodia saa muokata. Millaisia muutoksia koodiin pitäisi tehdä, jotta alkuperäiset esi- ja jälkiehdot riittäisivät? Koodia ei ole pakko kirjoittaa, yksityiskohtainen sanallinen kuvauskin riittää. Voi myös olla, että joissain tapauksissa korjaus ei ole edes mahdollinen (kerro tästäkin).

   c) Kerro jokaisesta operaatiosta, minkä *poikkeustakuun* se tarjoaa ja miksi.

   Tässä tehtävässä saa olettaa, että `string::length` ei heitä poikkeuksia ja että muut käytetyt `string`:n operaatiot jättävät merkkijonon ennalleen, jos heittävät poikkeuksen.

```
1  #include <string>
   #include <stdexcept>
3
   class Nimi
5  {
   public:
7    // Esiehto: -
     // Jalkiehto: Olion etu- ja sukunimet
9    //   alustettu tyhjiksi
     // Poikkeukset: -
11   Nimi() : etunimet_(0), sukunimi_(0)
     {
13   }

15   // Esiehto: -
     // Jalkiehto: Olio siivottu
17   // Poikkeukset: -
     ~Nimi()
19   {
     }
21
     // Esiehto: -
23   // Jalkiehto: palauttaa n:nnen etunimen
     //   pituuden
25   // Poikkeukset: -
     unsigned int etunimen_pituus(int n) const
27   { // Lasketaan etunimien pituuksia,
       // lopetetaan n:nnen jalkeen
29     unsigned int pituus = 0;
       for (unsigned int i = 0;
31          i < etunimet_->length(); ++i)
       {
33       if (etunimet_->at(i) == ' ')
         { // Etunimi loppui. Palaa, jos n:s,
35         // muuten siirrytaan seuraavaan
           if (--n == 0) { return pituus; }
37         else { pituus = 0; }
         }
39       else { ++pituus; }
       }
41     // Tanne, koska viim. nimi ei lopu sanavaliin
       return pituus;
43   }

45   // Esiehto: s ei ole tyhja merkkijono
     // Jalkiehto: Sukunimi on vaihdettu annetuksi
47   // Poikkeukset: muistin loppuminen
     void vaihda_sukunimi(std::string const& s)
49   {
       delete sukunimi_; sukunimi_ = 0;
51     // Tee kopio
       sukunimi_ = new std::string(s);
53   }

55   // Esiehto: -
     // Jalkiehto: Palauttaa merkkijonon, jossa
57   //   etu+suku sanavalilla erotettuna
     // Poikkeukset: -
59   std::string kokonimi()
     {
61     return *etunimet_ + ' ' + *sukunimi_;
     }
63
     // Esiehto: s:ssa ei sanavaleja
65   // Jalkiehto: etunimi s lisatty muiden peraan
     // Poikkeukset: muistin loppuminen
67   void lisaa_etunimi(std::string const& s)
     {
69     if (etunimet_ == 0) {
         etunimet_ = new std::string(s);
71     } else {
         etunimet_->push_back(' '); // Vali
73       etunimet_->append(s); // Etunimi
       }
75   }

77 private:
     std::string* etunimet_;
79   std::string* sukunimi_;
   };
```