

OHJ-1450 Olio-ohjelmoinnin jatkokurssi (Matti Rintala)

Tentti 14.5.2012

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 kysytyihin 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 olio-ohjelmoinnin/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) Pysyvyys- ja vaihtelevuusanalyysi
(<i>commonality and variability analysis</i>) | d) Suunnittelumalli (<i>design pattern</i>) |
| b) Viipaloituminen (<i>slicing</i>) | e) Laiska kopiointi (<i>copy-on-write</i>) |
| c) Metaohjelmointi (<i>metaprogramming</i>) | f) Poikkeushierarkia (<i>exception hierarchy</i>) |

2. Termipareja

Alla on muodostettu pareja kurssiin liittyvistä termeistä. Selitä ko. termipareista, miten termit liittyvät yhteen. (Huomaa, että joissain termeissä yhtenemiskohtia voi olla useampi kuin yksi.)

- Periytyminen — Olioiden kopiointi
- Abstrakti kantaluokka — Virtuaalifunktiot
- Kopiorakentaja — Funktion paluuarvo
- Funktio-olio — Geneerisyys
- Sopimussuunnittelu — Poikkeukset
- Osoittimet — Olioiden omistus/elinkaari

3. Sopimussuunnittelu ja poikkeustakuut

- a) Mistä sopimussuunnittelussa oikein on kyse, mitä siihen liittyy ja mitä hyötyä siitä on ohjelman suunnittelussa?
- b) Miten sopimussuunnittelu voi hyödyttää myös ohjelman testausvaihetta?
- c) Millaisia vaikutuksia periytymisellä on sopimussuunnitteluun?
- d) Mitä ja mitkä ovat poikkeustakuut, mitä hyötyä niistä on ja miten ne helpottavat luotettavan ohjelman suunnittelemista?

..... **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
4 class Nimi
5 {
6 public:
7     // Esiehto: -
8     // Jalkiehto: Ollon etu- ja sukunimet
9     // alustettu tyhjiksi
10    // Poikkeukset: -
11    Nimi() : etunimet_(0), sukunimi_(0)
12    {
13    }
14
15    // Esiehto: -
16    // Jalkiehto: Ollon etu- ja sukunimet
17    // Poikkeukset: -
18    ~Nimi()
19    {
20        delete etunimet_;
21        delete sukunimi_;
22    }
23
24    // Esiehto: -
25    // Jalkiehto: palauttaa n:n etunimen
26    // pituuden
27    // Poikkeukset: -
28    unsigned int etunimen_pituus(int n) const
29    { // Lasketaan etunimien pituuksia,
30      // lopetetaan n:n jälkeen
31      unsigned int pituus = 0;
32      for (unsigned int i = 0;
33           i < etunimet_>length(); ++i)
34      {
35          if (etunimet_>at(i) == ' ')
36              // Etunimi loppui. Palaa, jos n:s,
37              // muuten siirrytään seuraavaan
38              if (--n == 0) { return pituus; }
39              else { pituus = 0; }
40      }
41      else { ++pituus; }
42    }
43
44    // Tanne, koska viim. nimi ei lopu sanavaliin
45    return pituus;
46 }
47
48 // Esiehto: s ei ole tyhja merkkijono
49 // Jalkiehto: Sukunimi on vaihdettu annetuksi
50 // Poikkeukset: muistin loppuminen
51 void vaihda_sukunimi(std::string const& s)
52 {
53     delete sukunimi_; sukunimi_ = 0;
54     // Tee kopio
55     sukunimi_ = new std::string(s);
56 }
57
58 // Esiehto: -
59 // Jalkiehto: Palauttaa merkkijonon, jossa
60 // etu+suku sanavallilla erotettuna
61 // Poikkeukset: -
62 std::string kokonimi()
63 {
64     return *etunimet_ + ' ' + *sukunimi_;
65 }
66
67 // Esiehto: s:ssa ei sanavaleja
68 // Jalkiehto: etunimi s lisetty muiden peraan
69 // Poikkeukset: muistin loppuminen
70 void lisaa_etunimi(std::string const& s)
71 {
72     if (etunimet_ == 0) {
73         etunimet_ = new std::string(s);
74     } else {
75         etunimet_>push_back(' '); // Vali
76         etunimet_>append(s); // Etunimi
77     }
78 }
79 private:
80     std::string* etunimet_;
81     std::string* sukunimi_;
82 };

```