Turun yliopisto
University of Turku

Olio-ohjelmoinnin metodiikka
Loppuraportti

Ryhmä:
Mikko Mallikas (123456)

Loppuraportti
23. lokakuuta 2018
8 s., 1 liites.

Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietotekniikka
Olio-ohjelmoinnin metodiikka

1 Johdanto

Huom, yhteistyö kurssilaisten kanssa ei ole kiellettyä, mutta lopullinen raportti tulisi kirjoittaa omin sanoin ja ajatuksin.

Tämä johdantoteksti on tehtävänannossa lähinnä ohjeistukseksi ja koko luvun voi karsia pois, jos haluaa. Työtä ei arvostella tutkielmana. Raportilla on vaikutusta kurssiarvosanaan korkeintaan ± 1 yksikköä.

Kurssi on nyt melkein takanapäin ja aika laskea, mitä kaikesta opetetusta jäi lopulta käteen. Kurssilla tausta-ajatuksena on ollut valmistaa juuri Teitä moneen eri suuntaan jatkoa varten. Yksi aspekti on toimia jatkoharjoituksena ohjelmointitaitojen kartuttamisessa seuraaville ohjelmointia sivuaville kursseille tai työtehtäviin.

Konkreettisen ohjelmoinnin lisäksi kurssilla on otettu askeleita ohjelmistojen määrittämisen, mallintamisen ja testaamisen suuntiin sekä tutustuttu ulkoapäin tuleviin määrittelyihin ja koodeihin. Harjoitukset ovat myös testanneet taitojen soveltamista (yksinkertaisissa) ”käytännön” ongelmissa. Joskus rajallisen aikataulun vuoksi tehtävät ovat olleet lähinnä äärimmäisen riisuttuja ja/tai puujalkaisia karikatyyrejä – mutta pientä esimakua siitä, mitä haasteita voi vastaan tulla ja miten ongelmat voivat vaikeutua kerrostumalla tuttujen abstraktioiden päälle, jolloin mahdollisesti tarvitsee selvyyden vuoksi abstrahoida lisää järjestyksen luomiseksi.

Tässä on tärkeä huomata, että informaatioteknologiaan liittyvä osaaminen mo-

nimutkaistuu päivä päivältä. Esimerkiksi vanhoihin ohjelmointikieliin tulee ominaisuuksia lähinnä lisää ja ohjelmat pitenevät. Oli jokin kurssi miten pitkä tahansa, ennen pitkää kurssi ei voi vastata enää kaikkiin kysymyksiin, vaan parempi tavoite on ohjastaa oppijoita oikeaan suuntaan, jotta uuden löytäminen ja vanhan osaamisen arviointi helpottuisi. Köydellä on kuitenkin vaikea työntää, vedetään itse mukana.

Teknisen osaamisen lisäksi yhtenä aspektina kurssilla on myös akateeminen, ehkäpä filosofinenkin puoli: Miten määritellään mielekkäästi ohjelmistotekninen ongelma? Miksi jokin asia koetaan ongelmaksi? Miten jokin tekniikka valitaan ongelman ratkaisuksi? Mitä sen käyttö tarkoittaa? Mitä käytöstä seuraa? Ratkeaako ongelma ja mitkä ovat kunkin ratkaisun edut ja haitat? Onko olemassa muita ratkaisuja? Miksi pohdimme asioita kun samassa ajassa olisi jo kirjoittanut monta riviä koodia?

1.1 Rakenne

Työ koostuu seuraavista 5 luvusta, jotka käsittelevät melko tarkassa järjestyksessä kurssin luentoja, kurssikirjaa ja viikkoharjoituksia. Osioiden yhteydessä on erilaisia pohtivia kysymyksiä. Huomaa, että kysymyksiin ei ole Oikeita Vastauksia^{TM}, vaan kyse on pohdinnasta. Vastaa kuhunkin lukuun lyhyesti ja ytimekkäästi, korkeintaan 2 sivua ($<$ 400 sanaa) tämän ohjedokumentin formaatilla. Voit käyttää omaa ja mallivastausten koodia tekstin tukena, jos tarvetta. Jos nyt huomaat, että huomenna on raportin deadline, kannattaa varmistua että kaupasta ehtii vielä ostaa kahvia. Kannattaa myös muistaa, että ylipitkä vireystila ja valvominen ovat terveydelle haitallista. Ne välttää yleensä aloittamalla aikaisemmin.

Kaikesta huolimatta, jaksamista kirjoittamiseen. Loppu häämöttää.

1.2 Lyhyesti:

- Kirjoita oma teksti (plagiointi on tulosten sepittämisen ohella pahimpia akateemisia rikkeitä)
- Kysymyksiin ei ole Oikeita Vastauksia, vaan kyse on pohdinnasta
- Fontti normaali leipätekstifontti, esim. 12pt
- Marginaalit noin suunnilleen kuten tässä pohjassa
- Tekstin sanamäärä: 200-400 sanaa per luku
- Harjoitustehtävien kysymysten/vastausten materiaalia kannattaa käyttää apuna mahdollisuuksien mukaan (koodisanoja ei lasketa mukaan edelliseen sanamäärään, koska esimerkit voi copypastea valmiista materiaalista)
- Deadline 25.11.

2 Ohjelmakoodin määrittely

- Esiintyy kurssilla
  - Kurssikirja, luvut 1–3
  - Luennot 1–2
  - Harjoituskerrat 1→ (erityisesti 1–2)
- Pohdi seuraavia kysymyksiä
  - Mitä haasteita huomasit määrittelyjen laatimisessa? tarvitaanko määrittelyä? miksi huonolta vaikuttava määrittely on huono?
  - Miten nyt kurssin jälkeen näet seuraavien auttavan ohjelman oikeellisuuden takaamisessa (ei hauki on kala -copypastea vaan omia ajatuksia)
    * sopimuspohjainen ohjelmointi (valitse käsitteletkö kommentteihin kirjoitettuja ja/tai suoritettavia ehtoja: cofoja / eiffel)
    * yksikkötesti (unit test) / ominaisuustesti (property based test) (siltä osin kuin kurssin tehtävissä ehdittiin nämä lyhyesti käsitellä, jälleen voi pohtia erillisenä tai yhdessä sopimuksien kanssa)
    * suoritusaikaiset kielen tietotyypit
    * käännösaikaiset kielen tietotyypit
    * muita hyviä tekniikoita?

3 Olio-ohjelmoinnin peruspilarit

= (periytyminen, polymorfismi, kapselointi)

- Esiintyy kurssilla
  - Kurssikirja, luvut 3–5
  - Luennot 3–5
  - Harjoituskerrat 3–4
- Pohdi seuraavia kysymyksiä
  - Milloin kannattaa/ei kannata käyttää periytymistä?
  - Mikä on polymorfismin suurin anti olio-ohjelmoinnissa (ei ehkä välttämättä suurin, mutta jokin mielestäsi tärkeä seikka)

4 Luokkakokonaisuudet ja -hierarkiat

- Esiintyy kurssilla
  - Kurssikirja, luvut 4–5
  - Luennot 3–4, 7, 8
  - Harjoituskerrat 3→
- Pohdi seuraavia kysymyksiä
  - Yleisesti, mitä asioita mielestäsi kannattaa kuvata luokkien hierarkialla ja mitä ei? Esim.: (abstraktin) kantaluokan sijaan voisi vaihtoehtona olla rajapintaluokka?
  - Miten koet kapseloinnin tukevan erilaisten luokkien erilaisia sidoksia keskenään silloin kun samaan aikaan jollakin pitää olla olioon pääsy ja toisella oliolla ei? Esim. lompakko, omistaja, varas.

5 Olioiden perustoiminnot

- Esiintyy kurssilla
  - Kurssikirja, luvut 6, 8
  - Luennot 6, sovelluksia: 7, 8
  - Harjoituskerta 5
- Pohdi seuraavia kysymyksiä
  - Käsittele ajatuksiasi Javan olioiden perustoiminnoista (kuten listattu kurssimateriaalissa). Ota esimerkiksi jokin muu luennolla listattu olion perusominaisuus kuin .toString() ja pohdi miksi se on osa kaikkia luokkia Objectin kautta / erillinen entiteetti (siis esim. interface)
  - Mitä mieltä olet universaalista ekvivalenssista – ts. Java antaa verrata mitä tahansa kahta oliota keskenään. Vaihtoehtoinen toteutus voisi olla multiversaali ekvivalenssi, jossa kahta oliota ei voi lähtökohtaisesti verrata, mikäli luokkien välille ei määritetä erikseen ekvivalenssia.
    Vinkki: mieti esim. mitä ajatusvirheitä Java mahdollistaa.

6 Geneeriset tietorakenteet ja algoritmit

- Esiintyy kurssilla
  - Kurssikirja, luvut 7–8
  - Luennot 7–8
  - Harjoituskerta 5
- Pohdi seuraavaa
  1. Käytännön työssä hyödylliset tietorakenteet ja algoritmit ovat monesti erittäin vaikeita toteuttaa oikeellisesti, tehokkaasti ja mielekkäästi myös muiden ominaisuuksiensa suhteen.
  2. Geneerisyyskin on melko haastava konsepti, varsinkin kun otetaan huomioon varianssi ja mahdollisimman laaja uudelleenkäytettävyys.

  - Mieti, miten polymorfismi ja geneerinen olio-ohjelmointi voivat auttaa tässä. Ota esimerkkejä vaikkapa Javan kokoelmakirjastosta.

Liite A Liitedokumentti

Täytä jos tarvitset.