Ohjelmointi 1 / syksy 2007

13/20: Kierrätys kannattaa koodaamisessakin

Paavo Nieminen

nieminen@jyu.fi

Tietotekniikan laitos

Informaatioteknologian tiedekunta

Jyväskylän yliopisto

Tämän luennon rakenne

- Eilisestä pari juttua vielä:
  - RunTimeException vs. Exception
  - ”throws Jotakin” API-dokumentaatiossa
  - Tiedostoon tallentaminen
- Tämän päivän asia (melko lyhyt):
  - Moduulit ja yleiskäyttöisyys
  - Paketit ja ohjelman hierarkia
  - API
  - javadoc

Kierrätys kannattaa:

Moduulit

- Aliohjelmia kannattaa koota ”moduuleiksi” (Javassa luokiksi), joissa sijaitsee aina samankaltaiseen tarkoitukseen sopivia aliohjelmia.
- ”Moduuli” on yleisnimi, joka soveltuu riippumatta ohjelmointikielestä ja -alustasta.
- Olio-ohjelmoinnin ”luokka” on ”moduuli” siinä mielessä, että olion idea on tarjota sekä tietorakenne että algoritmit jonkun tietyn asiakokonaisuuden mallintamiseen.
- Instantoitava olioluokka on kuitenkin monipuolisempi käsite kuin silkka ”aliohjelmakirjasto”

Kierrätys kannattaa: Yleiskäyttöisyys

- Modulaarisuudesta saatavia hyötyjä:
  - Ohjelma jäsentyy selkeämmin.
  - sama moduuli saattaa käydä sellaisenaan uusien sovellusten tekemiseen!
  - Siis työmäärä vähenee!
- Mitä hyötyjen saanti edellyttää:
  - Ohjelman parempi jäsentyminen edellyttää vain sitä, että moduulijako ylipäätään tehdään
  - Selkeyden parantamiseksi moduulien keskinäisiä riippuvuuksia kannattaa välttää. Esim. ”A voi tarvita B:tä mutta silloin B:n ei tule tarvita A:ta.”
  - Uudelleenkäyttö edellyttää, että moduulien rajapinta ei juurikaan muutu päivityksen yhteydessä (muuten moduulia käyttävät ohjelmat ”menevät rikki”). Uudelleenkäytettävä moduuli on siis hiukan pidempiaikaisen kehityksen tulosta! Kuitenkin jokainen uusi toiminnallisuus on jo hyvä sijoittaa sopivimpaan moduuliin, vaikka tietäisi rajapinnan vielä pari kertaa muuttuvan!

Hierarkkinen mallintaminen tässäkin

- Luokkia (moduuleita) voi koota paketeiksi. Olennaisesti ne ovat ”isompia moduuleja”, joissa on sisällä pienempiä moduuleja.
- Tällainen ns. moduulijako voi sisältää useita hierarkiatasoja. Ks. esim. Javan APIn paketit ja niiden jäsentyminen.
- Selkeätä on, jos lähdekoodien hakemistorakenne vastaa moduulihierarkiaa. Tämä on normaalisti käytäntö kielestä riippumatta; Javassa työkalut jopa pakottavat noudattamaan sitä.

Kirjasto

- Puhutaan apukirjastoista (joko aliohjelma- tai luokkakirjastoista), joilla on API (Application Programming Interface) eli rajapinta. Javan laaja peruskirjasto on esimerkki tällaisesta. Muita on paljon.
- Kirjasto tarjoaa yleiskäyttöistä (moneen eri tarkoitukseen soveltuvaa) toiminnallisuutta, jota tietty sovellus käyttää tarjotakseen jonkun tietyn palvelun loppukäyttäjälle.
- Järkevä sovellus on sellainen, joka käyttää valmiita kirjastoja apunaan, ja joka itsekin jakautuu hierarkkisesti ”kirjastoiksi” toiminnallisten osioidensa mukaan.

API ja dokumentaatio

- Kirjasto koostuu siis tietorakenteiden, luokkien, aliohjelmien, vakioarvojen, ym. määrittelyistä
- API on kirjaston rajapinta: Julkiset asiat, joita voidaan käyttää ja kutsua. (Kirjastossa voi hyvin olla paljonkin sisäistä, piilotettua, toiminnallisuutta; API on se julkinen puoli, jonka käyttäytymisestä on sopimus eli ”rajapinta”)
- Kaoottisen kirjaston käyttö ja ylläpito on vaikeata, joten suunnittelu ja dokumentointi on tärkeää.
- APIn dokumentointiin on hyvä käyttää aputyökaluja; Javassa tämä on nimeltään javadoc ja se on sisäänrakennettu työkalustoon.

Käytännön esimerkki

$\rightarrow$ Muutetaan pari aiemmin nähtyä koodia sellaisiksi, että ne hyödyntävät yleiskäyttöistä moduulia.

$\rightarrow$ Tehdään javadoc -dokumentaatio myös.