CoMa - Ohjelmoinnin tyyliohje

Mindmap - Valtteri Tyrsky

Helsinki 16.12.2005

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Antti Kavonen
Ilari Nieminen
Tiina Torvinen
Valtteri Tyrsky
Kari Velling
Antti Vähäkotamäki

Asiakas

Anni Rytkönen

Johtoryhmä

Juho Teuho (ohjaaja)
Juha Taina

Kotisivu

`http://www.cs.helsinki.fi/group/mindmap`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
1.0	16.12.2005	Ensimmäinen versio

Sisältö

1 Tyyliohje

Koodaustyylin pohjana käytetään Sun:in määrittämiä java-koodauskonventioita, jotka ovat luettavissa osoitteessa http://java.sun.com/docs/codeconv Näistä konventioista käytetään jatkossa nimitystä javakonventiot.

Koodaus tehdään kokonaan englannin kielellä. Tämä koskee sekä varsinaista koodia että kommentteja.

Javakonventioita ei oteta sellaisenaan käyttöön tässä projektissa, vaan niitä täydennetään tarkemmilla tai kokonaan uusilla ohjeilla, ja toisaalta joitakin kohtia jätetään pois tai lievennetään. Seuraavissa kappaleissa kuvaillaan nämä lisäykset ja poikkeamat javakonventioista.

1.1 Java-tiedostojen alkukommentit

Tiedoston alkukommentin versionumero ja päiväys automatisoidaan käyttäen CVS:n keywordeja "$Revision : 1.2$"ja "$Date : 2005/12/1613 : 52 : 47$".

Kommentin muoto on seuraava:

```
/*
 * Classname.java
 *
 * $Revision: 1.2 $
 * $Date: 2005/12/16 13:52:47 $
 *
 * Copyright (c) 2005 Mindmap Group / University of Helsinki
 * + tietoa lisenssistä
 */
```

1.2 Metodien kommentointi

Metodia ei tarvitse kommentoida jos se on nimensä puolesta rittävän yksiselitteinen. Jos metodi kuitenkin kommentoidaan, se tehdään javadoc-tyylisesti.

1.3 Import-lauseet

Kooditiedostossa importoidaan vain ne luokat ja rajapinnat joita koodi tarvitsee. Kokonaista pakkausta ei siis koskaan importoida, joten '*'-symbolia ei käytetä import-lauseissa.

1.4 Sarkain

Sarkainta (TAB) ei käytetä koodin sisennykseen eikä mihinkään muuhunkaan. Koodi sisennetään neljä (4) välilyöntiä kerrallaan.

1.5 Koodirivin pituus

Koodirivin pituus ei saa ylittää 80 merkkiä. (Vihje: Eclipsestä saa näkyviin "print margin-apuviivan, joka on oletusarvoisesti 80 merkin kohdalla).

1.6 Pitkän lausekkeen jakaminen riveille

Javakonventioissa esitetyt ohjenuorat ovat hyviä, mutta niitä ei tarvitse noudattaa orjallisesti.

1.7 Kommentit koodin seassa

Kahdella vinoviivalla alkavaa kommentointia käytetään a) koodin poistamiseen käytöstä, b) koodirivin perään kirjoitettavassa kommentissa. Esimerkkejä:

```
//int x;   <--- nämä koodirivit on kommentoitu pois käytöstä
//int y;
Point location;

Animal a = AnimalFactory.createAnimal(
    Animal.CAT,
    "Misse",
    null     // owner not known, thus null
);
```

Muissa tapauksissa koodin sekaan kirjoitettavissa kommenteissa käytetään muotoa:

```
/* Check animal type */
if (animal instanceof Cat) {
    /*
     * For cats, we need to make sure that
     * blaablaa .... blaa blaa...
     * blaablaa blaa
     */
}
```

Toisin kuin Sun:in konventioissa esitetään, yksirivistä kommenttia edeltämään ei tarvitse laittaa tyhjää riviä.

1.8 Muuttujien määrittely

Yhdellä rivillä määritellään vain yksi muuttuja.

```
int x, y;  // ei näin!

int x;     // ..vaan näin!
int y;
```

Ryhmää muuttujamäärittelyitä voi selkiyttää sijoittamalla muuttujan tyypin ja nimen väliin välilyöntejä (EI tabeja), kuten seuraavassa esimerkissä:

```
int number;
double fraction;
MyObject entry;
```

Tällöin välilyöntien määrä määräytyy pisimmän tyypin nimen mukaan, jota tulee seurata kaksi välilyöntiä kuten esimerkissä. Käytä kuitenkin ryhmittelyä välttääksesi todella pitkiä välejä:

```
AbsolutelyFarTooLongClassName instance1;
AbsolutelyFarTooLongClassName instance2;

int n; // Näitä ei järkeä siirtää
long delta;
long price;
```

1.9 Castin jälkeen ei tarvitse aina välilyöntiä

Esimerkkejä:

```
Node n = (Node) iterator.next(); // Välilyönti OK

doComputing((int) (a / b)); // Välilyönti outo
doComputing((int)(a / b)); // Parempi ilman
```

1.10 Tyhjät rivit metodien välissä

Luokassa metodien väliin sijoitetaan yleensä yksi tyhjä rivi. Tyhjä rivi voidaan kuitenkin jättää pois, jos näin saavutetaan ohjelmoijan mielestä selkeämpi lopputulos.

1.11 Metodien parametrilista

Metodin parametrilistassa parametreja erottavien pilkkujen jälkeen täytyy sijoittaa yksi välilyönti, mikäli seuraava parametri sijaitsee samalla rivillä. Parametrilistaa ympäröivien sulkeiden ja itse parametrien väliin SAA sijoittaa yhden välilyönnin. Välilyöntejä on tällöin oltava tasan yksi kummassakin päässä parametrilistaa. Esimerkki:

```
public void doStuff( Object obj ) {
    /* Välilyönnit parametrilistan päissä: OK*/
    int result  = calculateResult( someValue, someOtherValue );

    /* Ei välilyöntejä parametrilistan päissä: MYÖS OK */
    int result2 = calculateResult(someValue, someOtherValue);
}
```

Jos metodin määrityksessä parametrilista ylittää rivin maksimipituuden, täytyy parametrit jakaa riveille. Tällöin parametrit sijoitetaan alekkain, ja erotetaan sulkeista välilyönneillä. Esimerkki:

```
public static void calculateSomething( SomeObject something,
                                       SomeOtherObject other,
                                       int integerValue ) {
    ....
}
```

Myös metodikutsun jakautuessa usealle riville, olisi suositeltavaa pyrkiä sijoittamaan parametrit alekkain. Esimerkki:

```
/* EI NÄIN: */
SomeResultObject result = calculateResult(Object firstParam,
        Object secondParam, Object thirdParam);

/* VAAN ESIM NÄIN: */
SomeResultObject result = calculateResult(
    Object firstParam,
```

```
    Object secondParam,
    Object thirdParam
);

/* TAI NÄIN: */
SomeResultObject result =
    calculateResult(
        Object firstParam,
        Object secondParam,
        Object thirdParam
    );
```

1.12 Muuttujien nimeäminen

Muuttujat nimetään pääsääntöisesti tyyliin:

```
undoStackPointer
```

eli aloitetaan pienellä alkukirjaimella ja siitä eteenpäin jokainen sana alkaa isolla kirjaimella. Kuitenkin myös alaviivaa voidaan käyttää sanoja erottamaan silloin kun luettavuus sitä vaatii. Tämä tulee vastaan erityisesti tapauksia joissa loppuosa muuttujan nimestä on vain yhden tai kaksi merkkiä pitkä:

```
member_id // memberId näyttää sekavalta
count_a   // countA näyttää sekavalta
```

Lyhenteiden kohdalla myös pelkkien isojen kirjaimien käyttäminen on sallittua:

```
memberID
```

1.13 Taulukoiden ja listojen nimeäminen

Taulukot ja listat nimetään monikossa, esim:

```
int[] numbers;
List objects;
```

1.14 Listojen tyyppi

Dynaamista listaa tarvittaessa käytetään normaalisti luokkaa ArrayList. Vector-luokkaa voidaan käyttää jos listalta vaaditaan säieturvallisuutta. LinkedList-luokkaa voidaan käyttää jos listaan tullaan kohdistamaan paljon insertointeja tai deletointeja (muualle kuin listan loppuun).

1.15 Aksessoi rajapinnan kautta aina kun mahdollista

Olioita käytetään mahdollisimman korkealla tasolla Esimerkki:

```
List nodes = new Vector(); // OK

Vector nodes = new Vector(); // EI NÄIN, ELLEI PAKKO
```

1.16 Listojen läpikäynti

Listoja käydään läpi iteraattoreiden avulla while- tai for-silmukassa, ellei tilanne vaadi muuta. Esimerkkejä:

```
Iterator iter = nodes.iterator();
while (iter.hasNext()) {
    Node n = (Node) iter.next();
    n.doSomething();
}

for (Iterator i=nodes.iterator(); i.hasNext();) {
    Node n = (Node) i.next();
    n.doSomething();
}
```

Seuraavassa esimerkissä tarvitaan tieto indeksistä, joten iteraattori ei ole paras ratkaisu:

```
public int getIndexOf(Node n) {
    for (int i=0; i<nodes.size(); i++) {
        if (nodes.get(i).equals(n)) {
            return i;
        }
    }
    return -1;
}
```

1.17 Päättymättömät silmukat

Silmukalle tulisi aina pyrkiä määrittämään poistumisehto. Joissakin harvinaisissa tilanteissa on kuitenkin luontevampaa jättää silmukan poistumisehto pois, ja huolehtia silmukasta poistumisesta silmukan sisällä, kutsuen break-lausetta jollakin ehdolla. Tällöin silmukka toteutetaan while-lauseella seuraavasti:

```
while (true) {
    ...
}
```

Muita muotoja, kuten for-lausetta, ei sallita:

```
for (;;) {  // ei näin!
    ...
}
```