hyväksymispäivä arvosana

arvostelija

COBOL ja Python: Datan kapselointi

Erkki Heino, Tero Huomo, Eeva Terkki

Helsinki 20.2.2013

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 COBOL

1.1 Perinteinen COBOL

COBOLissa tietoa voidaan ryhmitellä muodostamalla tietueita (record) ja ryhmiä (group). `DATA DIVISION` -osiossa käytettävät muuttujien tasonumerot (level number) määrittävät, minkälaisia kokonaisuuksia muuttujat muodostavat. Seuraavassa esimerkissä on kuvattu opiskelijatietue.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OPISKELIJA.
   05 NIMI.
       10 ETUNIMI  PIC A(10).
       10 SUKUNIMI PIC A(10).
   05 NUMERO       PIC 9(10).
   05 SUKUPUOLI    PIC A.
       88 MIES     VALUE "M".
       88 NAINEN   VALUE "N".
```

`OPISKELIJA` on tietue, joka koostuu nimestä, opiskelijanumerosta ja sukupuolesta. Nimi on jaettu kahteen osaan: etunimi ja sukunimi.

Tasonumerot kuvaavat tiedon hierarkian, mutta tietyillä numeroilla on erityismerkitys. Hierarkiaa kuvaavat tasonumerot 02-49. Tasonumero 01 tarkoittaa, että kyseessä on tietue. Merkitsemällä tasonumeroksi 66 voidaan edeltävälle muuttujalle antaa toinen nimi. 77 kuvaa muuttujaa, joka ei ole osa mitään rakennetta. Tason 88 avulla voidaan määrittää muuttujaan liittyviä ehtoja: yllä olevassa esimerkissä opiskelijan sukupuoli on mies, jos muuttujan `SUKUPUOLI` arvo on M, ja nainen, jos arvo on N [Ans74, s. I-84]. Esimerkiksi voitaisiin tarkistaa, onko opiskelija mies:

```
IF MIES OF OPISKELIJA
```

Todellisuudessa tietue ei ole juuri muuta kuin merkkijono, jonka eri osat kuvaavat tietueen kenttiä. Tietueelle voidan antaa arvot kenttä kerrallaan:

```
MOVE "ARTO" TO ETUNIMI OF OPISKELIJA.
```

Toisaalta koko tietueelle voidaan antaa arvo kerralla:

```
MOVE "ARTO      WIKLA     1234567890M" TO OPISKELIJA.
```

Esimerkin tietue sisältää kerrallaan yhden opiskelijan tiedon — tällaisenaan koodissa ei siis voida käsitellä useampaa opiskelijaa samanaikaisesti. Rakenne voidaan määritellä taulukoksi (table), jolloin samaa rakennetta voidaan käyttää useamman samanrakenteisen tiedon käsittelemiseen ilman saman rakenteen uudelleenmäärittelyä.

Muuttuja määritellään taulukoksi `OCCURS`-lauseen avulla. `OPISKELIJA` voitaisiin muuttaa taulukoksi muokkaamalla määrittelyä seuraavasti:

```
01 OPISKELIJA OCCURS 5 TIMES.
```

Esimerkiksi taulukon toiseen opiskelijaan viitattaisiin indeksillä 2: `OPISKELIJA(2)`. Taulukon sisällä voidaan myös määritellä sisäkkäisiä taulukoita.

1.2 COBOLin oliolaajennus

COBOL ei alunperin ole olio-ohjelmointikieli. COBOLista on kuitenkin kehitetty olio-ohjelmoinnin mahdollistava laajennus, Object-oriented COBOL. Ensimmäiset oliolaajennukset COBOLiin tehtiin jo vuonna 1997 [Wik13]. Kieltä laajensivat esimerkiksi Micro Focus sekä IBM. Lopullinen ISO standardi Object-Oriented COBOLille tehtiin vuonna 2002.

Object-Oriented COBOL (jatkossa OO COBOL) lisää kieleen kolme olio-ohjelmointikielen perustavanlaatuista ominaisuutta - tiedon kapseloinnin, perinnän sekä polymorfismin. Olio-ohjelmoinnin lisäyksien syntaksi sekä tarjotut ominaisuudet vaihtelevat suuresti eri toteuttajien välillä. Niiden yhtenäisyys vuoden 2002 ISO standardin kanssa on vaihteleva.

Oliolaajennoksen tärkeimpiä lisäyksiä on mahdollisuus luokkien ja olioiden määrittelyyn. Kuten normaali COBOL-ohjelma, luokka määritellään neljän *osion* (division) avulla. IBM'n toteutus [IBM13] ja Micro Focus'n toteutus [Mic13] vastaavat pitkälti toisiaan. `IDENTIFICATION DIVISION`-osiossa ohjelman tunnuksen sijasta annetaan luokan tunnus, `CLASS ID`, sekä mahdolliset periytymisestä kertovat tiedot. Luokan yhteinen tietosisältö ja metodit luetellaan `FACTORY`-osiossa, olioiden tietosisältö ja metodit `OBJECT`-osiossa. Muihin luokkiin viittaamiseen on varattu erillinen `REPOSITORY`-osio.

Olioiden metodien kutsumiseksi on OO COBOLiin lisätty avainsana `INVOKE`.

```
INVOKE ARTONTILI "TALLETA" USING KATEINEN RETURNING KUITTI
```

Esimerkissä kutsutaan olion `ARTONTILI` metodia `TALLETA`, jolle annetaan parametriksi muuttujan `KATEINEN` arvo. Metodin palauttama arvo sijoitetaan muuttujaan `KUITTI`.

Periminen ja rajapintojen toteutus vaihtelee runsaasti. Micro Focus'n Visual COBOL ei mahdollista moniperintää, mutta luokat voivat toteuttaa Javan tyyliin usean rajapinnan [Mic13]. IBM:n Enterprise COBOL ei määrittele dokumentoinnissa rajapintoja lainkaan, mutta estää moniperinnän [IBM13]. Michael Kastenin [Kas98] mukaan C++:n tavoin IBM:n OO COBOL kuitenkin luettelisi erilaisia sääntöjä, joilla usean luokan yhtäaikaista perimisestä kumpuava niin kutsuttu timanttiongelma ratkaistaan. Tästä voidaan päätellä, että OO COBOL on myös tukenut moniperintää. Myös Micro Focus'n varhainen dokumentaatio mainitsee moniperinnän tuen [Mic06].

2 Python

Pythonissa muuttujien arvot ovat viitteitä olioihin, ja kielessä kaikki esitetään olioina tai olioiden välisinä suhteina [].

2.1 Rakenteiset tyypit

Pythonin tarjoamia sekvenssityyppejä ovat muun muassa lista (`list`) ja monikko (`tuple`). Muita kielen tietorakenteita ovat esimerkiksi joukko (`set`) ja sanakirja (`dict`).

Lista muistuttaa jossain määrin Javan `ArrayList`ia. Listan alkioiden ei kuitenkaan tarvitse olla keskenään samaa tyyppiä [Pyt13c]. Listan alkiot erotellaan toisistaan pilkuilla ja listaa ympäröivät hakasulut. Merkintä `[]` tarkoittaa tyhjää listaa. Listan sisältöä voi muokata sen luomisen jälkeen.

Listoja voi käsitellä monella tavalla. Listaa voidaan esimerkiksi järjestää ja listan alkioita voi lisätä ja poistaa. Listan muokkaamiseen tarkoitettujen metodien lisäksi Pythonissa on listojen käsittelyyn myös funktionaaliselle ohjelmoinnille tyypillisiä funktioita, kuten `map` ja `filter`. Funktio `map` tuottaa listan, jonka alkiot ovat tuloksia funktion parametrina saaman funktion soveltamisesta funktion listaparametrin alkioihin. Funktio `filter` puolestaan tuottaa listan suodattaen alkioita parametrina saamansa funktion mukaisesti listaparametrista. Seuraavassa esimerkissä `map`-funktiolla luodaan luvun kaksi potensseja sisältävä lista:

```
>>> map(lambda x: 2**x, range(0, 11))
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

Pythonissa `lambda` määrittelee anonyymin funktion, jolla on tässä esimerkissä yksi parametri. `range(0,11)` palauttaa kokonaisluvut nollasta kymmeneen.

Listakomprehensiot (list comprehension) ovat näppärä tapa luoda listoja. Listakomprehensioita käytetään samaan tarkoitukseen kuin `map`- ja `filter`-funktioita, mutta ne ilmaisevat asian tiiviimmin ja selkeämmin. Listakomprehensioita voi olla myös sisäkkäin. Seuraavassa esimerkissä luodaan listakomprehensiolla samanlainen lista, joka yllä luotiin `map`-funktion avulla:

```
>>> [2**x for x in range(0, 11)]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

Kun edellisen esimerkin listakomprehensioon lisätään ehtolause, voidaan luoda lista, joka sisältää vain ne luvut, joissa eksponentti on parillinen:

```
>>> [2**x for x in range(0,11) if x % 2 == 0]
[1, 4, 16, 64, 256, 1024]
```

Monikkoa ympäröivät kaarisulut, ja alkiot erotellaan toisistaan pilkuilla. Monikot muistuttavat hieman listoja, mutta niitä käytetään yleensä eri tarkoituksiin. Toisin kuin listan tapauksessa, kerran luotua monikkoa ei voi muokata [Pyt13b]. Monikko voi kuitenkin sisältää muuttuvia tietotyyppejä, kuten listoja. Monikon alkiot voivat olla myös toisia monikkoja.

Seuraava esimerkki esittelee monikon purkamista:

```
>>> monikko = ("a", 123, 3.14)
>>> x, y, z = monikko
>>> x
"a"
>>> y
123
>>> z
3.14
```

Esimerkissä monikon alkiot sidotaan järjestyksessä muuttujiin `x`, `y` ja `z`. Kun monikko puretaan, vasemmalla puolella on oltava yhtä monta muuttujaa kuin monikossa on alkioita. Muussa tapauksessa tapahtuu suoritusaikainen virhe.

Joukko on järjestämätön tietorakenne, ja se voi sisältää saman alkion korkeintaan kerran. Joukkoihin voi käyttää matemaattisia joukko-operaatioita, kuten yhdistettä, leikkausta ja erotusta. Python tukee myös joukkokomprehensioita (set comprehension), jotka toimivat listakomprehensioiden tapaan:

```
>>> joukko = {x for x in "nakkivanukas" if x not in "aeiouyää"}
>>> joukko
set(['k', 'v', 's', 'n'])
```

Esimerkissä luodaan joukko, joka sisältää merkkijonon `nakkivanukas` ne merkit, jotka eivät ole pieniä vokaaleja. Esimerkistä huomataan, että kukin ehdon täyttävä merkki esiintyy luodussa joukossa vain kerran.

Sanakirja sisältää avain-arvo-pareja, ja se vastaa esimerkiksi Javan `HashMap`ia. Sanakirjaan voi lisätä uuden avain-arvo-parin ja sieltä voi hakea arvoa avaimen perusteella. Pareja voi myös poistaa, ja olemassa olevaan avaimeen liittyvää arvoa voi päivittää. Avaimena voi käyttää arvoa, joka ei ole muokattavaa tyyppiä; avaimiksi kelpaavat esimerkiksi merkkijonot, luvut sekä monikot, joiden alkiot eivät ole muokattavia.

Avaimen ja siihen liittyvän arvon välissä on kaksoispiste, ja parit erotetaan toisistaan pilkuilla. Sanakirjaa ympäröivät aaltosulut, ja merkintä `{}` tarkoittaa tyhjää sanakirjaa. Seuraavassa esimerkissä käytetään sanakirjaa eläinten ja niiden ääntelyn tallentamiseen:

```
>>> elaimet = {"koira": "hau", "kissa": "miu"}
>>> elaimet["seeprapeippo"] = "tööt"
>>> elaimet
{"seeprapeippo": "tööt", "koira": "hau", "kissa": "miu"}
>>> elaimet["seeprapeippo"]
"tööt"
```

Esimerkissä luodaan aluksi sanakirja, jonka avaimet ja arvot ovat merkkijonoja. Sanakirjaan lisätään uusi avain-arvo-pari, ja lopuksi sieltä haetaan avaimeen `seeprapeippo` liittyvä arvo.

Myös sanakirjoja voi luoda käyttämällä komprehensioita (dict comprehension):

```
>>> {x: 2**x for x in (0,1,2,3)}
{0: 1, 1: 2, 2: 4, 3: 8}
```

Esimerkissä luodaan sanakirja, jonka avaimina ovat kokonaisluvut nollasta kolmeen, ja arvoina luku kaksi korotettuna avaimen osoittamaan potenssiin.

Yleiskäyttöisten kokoelmien lisäksi Pythonissa on myös tehokkaita, tiettyihin käyttötarkoituksiin erikoistuneita kokoelmatyyppejä. Standardikirjaston `collections`-moduulissa on muun muassa tehokas tietorakenne jonojen ja pinojen toteuttamiseen (`deque`) sekä `OrderedDict`, joka toimii kuten tavallinen sanakirja, mutta muistaa lisäksi, missä järjestyksessä alkiot on lisätty [Pyt13a].

2.2 Oliot ja luokat

Ennen versiota 2.2 Pythonin luokat erosivat hieman nykyisistä luokista. Itse asiassa vanhanmalliset luokat olivat käytössä rinnakkain uudenmallisten luokkien kanssa Python 3:een asti [Pyt13d]. Seuraavassa käsitellään uudenmallisia luokkia, ellei toisin mainita.

Python on dynaaminen kieli ja sen oliot ovat hyvin joustavia. Luokka määritellään seuraavalla tavalla:

```
class Luokka(object):
    def __init__(self, parametri):
        self.muuttuja = parametri
```

Yllä olevan esimerkin luokassa on määritelty myös konstrukorinomainen `__init__`-metodi. Tätä metodia kutsutaan olion luomisen yhteydessä. Metodi saa ensimmäisenä parametrinaan (self) luokan ilmentymän. Luokasta `Luokka` luotaisiin instanssi seuraavalla tavalla: `l = Luokka("arvo")`. Parametreille ja muuttujille ei määritetä tyyppiä, joten esimerkinkin metodi hyväksyy minkä tyyppisen muuttujan tahansa — tässä tapauksessa merkkijonon. `__init__`-metodia ei kuitenkaan vaadita luokan määrittelyssä.

Muuttujaa `muuttuja` ei ole määritelty `__init__`-metodin ulkopuolella. Muuttujia voidaan määritellä myös metodin ulkopuolella, mutta silloin ne ovat käytettävissä myös ilman luokan instanssia.

Muuttujille ja metodeille ei voi määrittää näkyvyysalueita: kaikki luokan muuttujat ja metodit ovat käytettävissä kaikkialla, missä luokka on käytettävissä. Konvention mukaan muuttujat ja metodit, joita ei ole tarkoitettu käytettäväksi luokan ulkopuolella, nimetään siten, että ne alkavat alaviivalla. Tämä ei kuitenkaan estä niiden käyttöä.

Itse asiassa luokkia tai niiden ilmentymiä ei ole mitenkään suojattu muutoksilta: niiden muuttujia ja metodeja voidaan korvata ja uusia lisätä täysin vapaasti. Jos esimerkiksi luokan `Luokka` `__init__`-metodi ei miellytä, sen voi korvata vaikka seuraavasti:

```
>>> def uusi_init(self):
        print "Uusi metodi!"

>>> Luokka.__init__ = uusi_init
>>> l = Luokka()
Uusi metodi!
```

Metodin tai muuttujan voi myös poistaa: `del Luokka.__init__`. Toisaalta myös täysin uuden metodin lisääminen onnistuu helposti:

```
>>> Luokka.puhu = uusi_init
>>> l = Luokka()
>>> l.puhu()
Uusi metodi!
```

- name mangling

Olion muuttujat sijaitsevat assosiaatiotaulussa. Muuttu

- x.y() = X.y(x)

2.3 Periytyminen

Python tukee moniperintää.

- yläluokan init pitää kutsua eksplisiittisesti

3 Etuja ja haittoja

COBOLin olio-ohjelmoinnin mahdollistava laajennus on monimutkaistaa COBOLin syntaksia entisestään. Lisäksi COBOLille tyypillisesti jo yksinkertaisten luokkien määritteleminen tuottaa huomattavan paljon luettavaa koodia, jos vertailukohteena on esimerkiksi Javan tai varsinkin Pythonin tiivis ilmaisutapa.

Object-Oriented COBOLin toteutukset ovat monimuotoisia ja poikkeavat toisistaan ominaisuuksiltaan. On mahdollista kiistellä, onko IBM:n toteutuksessa mahdollista käyttää polymorfismia kovinkaan laajasti, sillä kielen referenssissä ei ole lainkaan kuvailtu rajapintoja. Lisäksi muun muassa Micro Focus tukee COBOLin yhteistyötä C-kielen kanssa, kun taas IBM mahdollistaa Javan ja COBOLin yhdistämisen. Yhteistyötuki eri kielten välille muuttamat implementaatiota suuresti.

Lähteet

Ans74 *American national standard programming language COBOL.* ANSI, New York, USA, 1974.

Pyt13a The python standard library – collections – high-performance container datatypes, http://docs.python.org/2/library/collections.html. [20.2.2013].

Pyt13b The python tutorial – data structures, http://docs.python.org/2/tutorial/datastructures.html. [19.2.2013].

IBM13 Writing object-oriented programs, http://pic.dhe.ibm.com/infocenter/pdthelp/v1r1/index.jsp?topic=%2Fcom.ibm.entcobol.doc_4.1%2FPGandLR%2Ftasks%2Ftpoot02.htm. [19.2.2013].

Kas98 Oo cobol: Comparison to c++, http://mck9web.com/cobol/ooc/oocvsc++.html. [19.2.2013].

Pyt13c The python tutorial – an informal introduction to python – lists, http://docs.python.org/2/tutorial/introduction.html#lists. [19.2.2013].

Mic13 Micro focus cobol language reference, http://documentation.microfocus.com/help/index.jsp?topic=%2Fcom.microfocus.eclipse.infocenter.visualcobol.vs%2FHRLHLHPDFX03.html. [19.2.2013].

Mic06 Oo programming with object cobol, https://supportline.microfocus.com/Documentation/books/nx50/opconc.htm. [19.2.2013].

Pyt13d Python v2.7.3 documentation language reference - data model, http://docs.python.org/2/reference/datamodel.html. [19.2.2013].

Wik13 Cobol, http://en.wikipedia.org/wiki/COBOL. [19.2.2013].