Ohjelmointi 1 / syksy 2007
11/20: Konepelti auki

Paavo Nieminen

nieminen@jyu.fi

Tietotekniikan laitos

Informaatioteknologian tiedekunta

Jyväskylän yliopisto

Tämän luennon rakenne

Tutustutaan debuggeriin ja sitä käyttäen laskukoneen (engl. computer) toimintaan, alkaen HelloWorldistä:

- Debuggerin idea; esimerkkinä Eclipsen debuggeri.
- Suorituskohta, kontrollin siirtyminen, kutsupino.
- Pinokehys; paikallinen muuttuja ja parametri
- Muistimallin idea. Javan muistimalli: pino, keko.
- Muuttujan näkyvyysalue
- Viitteen käsite. Primitiivityypin ja olioviitteen eron kristallisointi.
- Joitain alustavia huomioita: Dynaaminen muistinvaraus ja roskienkeruu

Debuggerilla tutkitaan suoritusta

$\rightarrow$ Katsotaan Eclipsen debuggerin käyttöä erilaisten Java-ohjelmien kanssa.

- Debuggerin avulla voidaan tutkia koodin suoritusjärjestystä ja muistin sisältöä.
- Suoritus voidaan saada pysähtymään ennaltamäärättyyn ohjelmakoodin kohtaan
- Suoritusta voidaan askeltaa yksi suorite kerrallaan.
- Hyötyä viallisten ohjelmien korjaamisessa (tutkitaan, miten suoritus etenee kohti kaatumista)
- Hyötyä on myös ohjelmoinnin yksityiskohtien oppimisessa!

Kontrolli ja rakenteet

- Suorituskohta, kontrollin siirtyminen
- Rakenteiden suoritus: Silmukat, ehdot ym. $\rightarrow$ tutkitaan mm. aiempia esimerkkikoodeja debuggerilla.
- Aliohjelmien kutsupino: viimeksi kutsuttu päällimmäisenä; alimpana main(), jota ajoympäristö kutsuu.
- Aliohjelman kutsuminen laittaa uusimman kutsun pinon päälle. Paluussa pinon päältä lähtee päällimmäinen pois.

Aliohjelma-aktivaatio

- Hieno nimi aliohjelmakutsun suorittamiselle on “aktivaatio”. Siinä:
  - Parametrien lukuarvot (viite on toteutuksen kannalta luku siinä missä primitiivitkin) siirtyvät kopioina aliohjelman käyttöön.
  - Kontrolli siirtyy aliohjelman koodiin; ts. kone alkaa suorittaa aliohjelman koodia
  - Koodi voi aktivoida muita aliohjelmia (tai itseään rekursiivisesti, mutta silti jokainen aktivaatio on oma yksilönsä kutsupinossa ja parametrit välittyvät normaalisti!)
  - Lopulta aktivaatio päättyy: Kontrolli siirtyy kutsujalle siihen kohtaan, jossa kutsu oli. Paluuarvo sijoittuu lausekkeeseen.
- Jokaisella aliohjelman aktivaatiolla on oma kehys paikallisille muuttujille. Parametrit ovat aivan kuin paikalliset muuttujat.
- Paikallinen muuttuja on olemassa vain aktivaation ajan!

Näkyvyysalue 1/2: Vielä paikallisempia muuttujia?

- Lohkorakenteisessa kielessä, kuten Java, muuttujan laajin näkyvyysalue (engl. scope) on se rakenne, jossa se on esitelty, esim. aliohjelman runko (paikallinen muuttuja / parametri) tai metodimäärittelyn ulkopuolinen luokan runko (luokan attribuutti). Näkyvyysalue voi olla myös esim. for-lause, jossa esitellään pelkästään silmukalle paikallinen indeksimuuttuja.
- Javan syntaksissa lohko on yksi mahdollinen lauserakenne, joten muuttujista voi tehdä milloin vain ”paikallisempia” kuin aliohjelman parametrit tai esim. aliohjelmarungon alussa esitellyt paikalliset muuttujat. Tarpeettomasti tällaista ei sovi tehdä . . .

Näkyvyysalue 2/2: Syntyvät, kuolevat, pinossa elävät

- Muuttujaa ei voi käyttää sen näkyvyysalueen ulkopuolella. Itse asiassa suorituksen aikana muuttujan lukuarvon tallentamiseen on ylipäätään olemassa tallennustilaa vain esittelyhetkestä siihen saakka kun sitä ympäröivän rakenteen suoritus loppuu. Paikallinen muuttuja on olemassa vain sen aikaa kuin sen näkyvyysalueen suoritus kestää!
- Näkyvyysalueensa sisälle sijoittuvissa lohkoissa muuttuja näkyy
- Sisemmässä rakenteessa voi esitellä ulomman lohkon muuttujanimen uudelleen, jolloin nimi peittyy (sarjaa ”ei yleensä kovin selkeä koodiratkaisu” . . .)

Muistimalli: missä bitit ovat?

- Tietokoneessa muisti on sähkökomponentteja . . . matalimmalla tasolla kone käsittelee sitä kuin siinä olisi vain peräkkäin numeroituja 8-bittisiä muistipaikkoja, joista jokaiseen voi kirjoittaa uuden sisällön tai lukea edellisen. Ei mennä siihen aiheeseen nyt. Päällä on abstraktioita, kuten kaikessa.
- Muistimalli (engl. memory model) tarkoittaa tapaa, jolla ajatellaan asioiden sijoittuvan koneen muistissa. Se on lähellä tai kaukana laitteiston näkökulmaa käytetystä työkalusta riippuen. Java-ohjelmoijan tarvitsema muistimalli on Javan virtuaalikoneen määritelmän mukainen.

Javan muistimalli: pino ja keko

Javan virtuaalikoneen (ja siis Java-kielen) muistimallissa on kaksi erillistä aluetta:

- Pinomuisti (engl. stack), jossa on aliohjelmien (metodien) paikalliset muuttujat ja parametrit — kullakin aktivaatiolla on oma kehys, joka sisältää muistipaikat primitiiveille ja viitteille.
- Kekomuisti (engl. heap), jossa on olioiden sisäiset tilat. Olion luonti varaa muistia keosta vähintään niin paljon kuin uuden yksilön tila tarvitsee.

Paikalliset pinoon, oliot kekoon

- Kun metodi käyttää paikallista muuttujaa tai parametria, se käyttää pinoa.
- Kun (ei-staattinen) metodi käyttää olion attribuuttia, se käyttää kekoa (erityisesti sen olioyksilön muistipaikkoja, johon viittaa metodin aktivaatioon aina implisiittisesti liittyvä ”this”-viite. Joissain kielissä vastaava ”this” tai ”self” -viite tulee parametrina, mutta Javassa se ”kuvitellaan”; esim. jokuSkänneri.nextLine() voisi kutsua metodia java.util.Scanner.nextLine() siten että ”this”-viitteeksi tulee sama mikä viitemuuttuja jokuSkänneri kutsuhetkellä on)

Alustavia huomioita

- Olion luonti on esimerkki dynaamisesta muistinvarauksesta: aina kun tarvitaan uusi olio, sen voi tehdä new:llä.
- Oliot myös kuolevat! Kuka vapauttaa tarpeettomasti varatun muistin?
- Nykyaikaisissa ohjelmointikielissä on ns. automaattinen roskienkeruu, joka vapauttaa käyttämättömäksi jääneille olioille dynaamisesti varatun muistin.
- Virtuaalikone osaa päätellä, että jos ei ole yhtään viitettä olioon, sitä oliota ei enää tarvita ja sen tarvitsema kekomuisti voidaan vapauttaa uusien, syntyvien olioiden käyttöön.