

Flyweight

String ja border

Kalle Mustonen | Suunnittelumallit | 25.04.2019

# Johdanto

Flyweight (hiutale suomeksi) on ohjelmistokielen suunnittelumalli, jossa yhteiskäyttöistä oliota käytetään samanaikaisesti useissa eri yhteyksissä. Olio kuitenkin käyttäytyy jokaisessa käyttöyhteydessä itsenäisesti.

Hiutaleoliolla on kaksi eri tilaa; sisäinen ja ulkoinen. Tarkastellaan esimerkkinä höyhen oliota. Sisäinen tila sisältää tietoa, mikä on jaettu jokaisen hiutaleen kanssa esim. höyhenen koko ja väri. Ulkoinen tila sisältää tietoa, mikä on jokaisen hiutaleen oma tieto esim. höyheneen sijainti.

Flyweight suunnittelumallin tarkoituksena on saavuttaa parempaa suorituskykyä, koska voidaan aina viitata samaan ilmentymään, eikä jokaista oliota varten tarvitse luoda uutta ilmentymää.

# Tehtävä

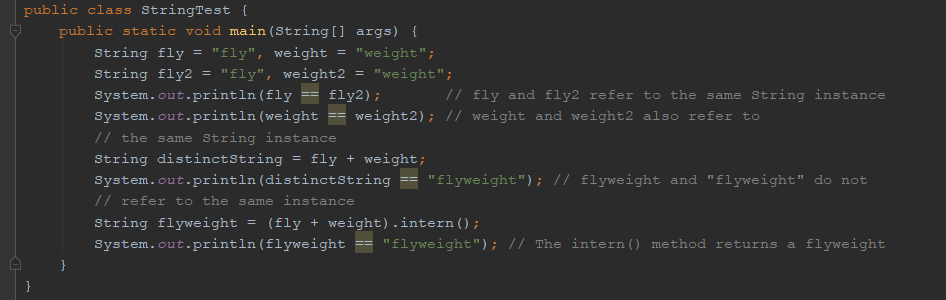
Tehtävänä oli lukea Flyweight-mallia käsittelevä artikkeli osoitteessa: <https://www.javaworld.com/article/2073632/make-your-apps-fly.html> . Artikkelissa käydään läpi, kuinka Flyweight-mallin voi implementoida Java ohjelmointikielellä ja kuinka se auttaa suorituskyvyssä. Artikkeli sisältää myös usean koodiesimerkin, joista oli tarkoitus tarkastella lähempää kahta. String ja Border. Näiden lähdekoodi oli tarkoitus kopioida omaan editoriin, testata ja todeta mitä näistä testeistä voi päätellä.

# Testit

Kopioin String ja Border esimerkkien lähdekoodit käyttämääni IntelliJ IDEA kehitysympäristöön, ajoin koodit ja kävin läpi mitä havaitsin esimerkeissä.

## String

String esimerkissä luodaan aluksi erilaisia String muuttujia eri arvoilla ja vertaillaan ”==” vertailulla. ”==” vertailu Javassa vertailee kahden muuttujan muistipaikkoja keskenään. Kuva 1 on kuva String esimerkin lähdekoodista.



Kuva StringTest

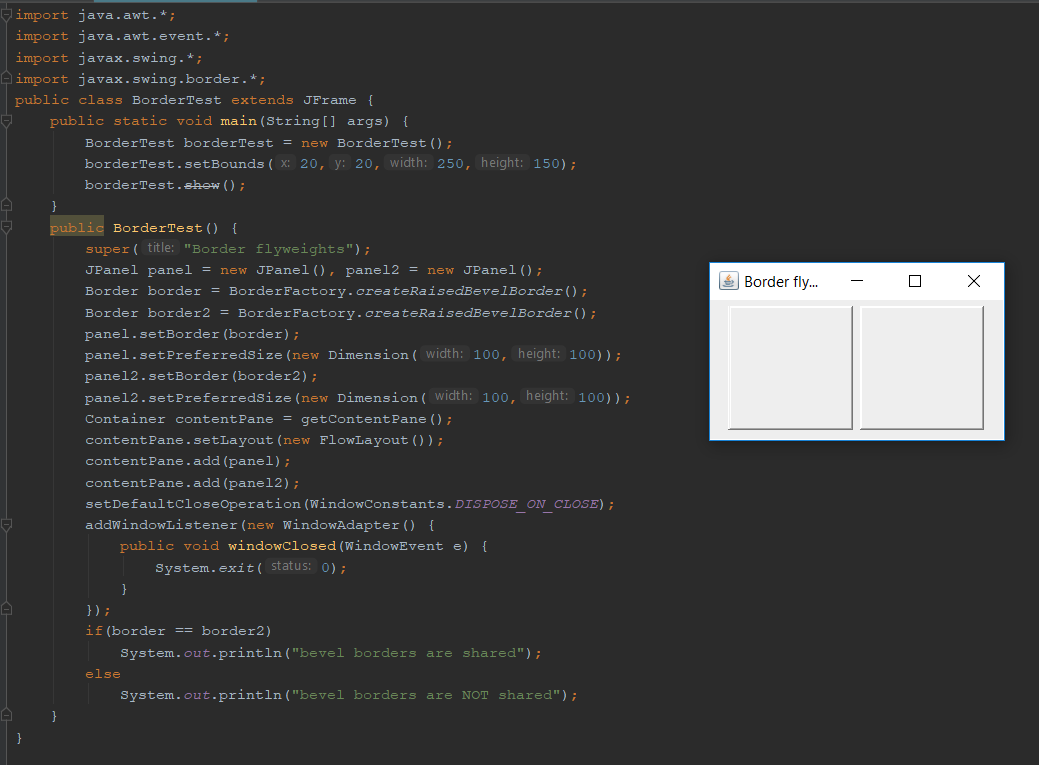
Huomataan, että fly == fly2 ja weight == weight2 ovat molemmat totta sillä molemmat muuttujat sisältävät viittauksen samaan merkki jonoon, eli hiutaleeseen.

Seuraavaksi havaitaan, että distinctString == ”flyweight” ei ole totta, koska distinctString sisältää viittauksen kahteen eri muuttujaan. Jos vertailuun käytettäisiin distinctString.equals(”flyweight”), niin tämä palauttaisi true, siää equals vertailee merkki jonojen sisältöä.

Viimeisessä kohdassa fly ja weight muuttujien merkki jonot yhdistetään yhdeksi ja samaksi merkki jonoksi ja tämän arvo sijoitetaan flyweight muuttujaan. Nyt flyweight == ”flyweight” palauttaa arvon true, koska ne viittaavat samaan paikkaan muistissa.

## border

Border esimerkissä luodaan Javan swing kirjastoa käyttäen pieni GUI, missä on kaksi eri paneelia, joiden ympärille asetetaan Border reunukset. Kuva 2 on Border esimerkin lähdekoodi.



Kuva 2 BorderTest

Bordereiden luontien jälkeen näiden muistipaikkoja vertaillaan keskenään ”==” operaattorilla ja tämä palauttaa arvon true, koska Javan Swing kirjaston Border on toteutettu Lightweight-mallilla.

Syynä tähän on suorituskyvyn optimointi. Jokaiselle widgetille, jolle haluat reunat luoda, niin voidaan viitata samaan Border olioon, eikä tarvitse luoda aina uutta ilmentymää.