12. Monimuotoisuus (osa 1)

Sisällys

- Johdanto.
- Periytymismekanismi määrittää alityypityksen.
  - Viitteiden sijoitus ja vertailu.
- Staattinen ja dynaaminen luokka.
- Myöhäinen ja aikainen sidonta.

Johdanto

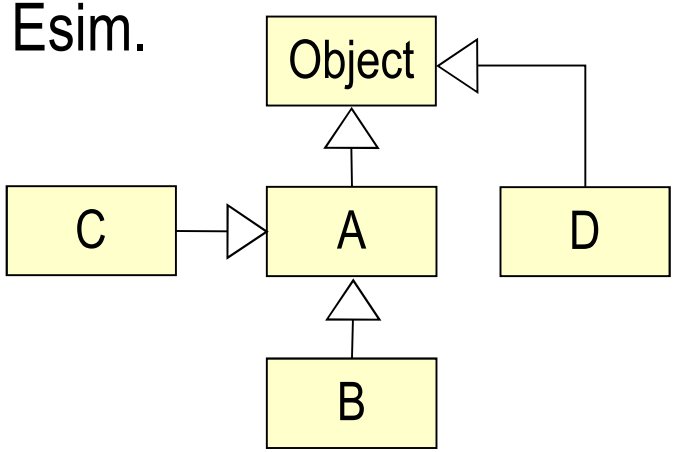
- Monimuotoisuus eli polymorfismi (polymorphism) on yleisesti sitä, että ohjelman tunnus voi tarkoittaa erilaisia asioita.
- Monimuotoisuus on monimuotoinen käsite.
  - Metodien kuormitus ja korvaaminen eräänlaista monimuotoisuutta.
- Monimuotoisuus liittyy erityisen kiinteästi periytymiseen:
  - Luokkahierarkia määrittelee alityypityksen, jossa kukin jälkeläinen edustaa alityyppiä ja kukin esi-isä ylityyppiä.
- Tarkastellaan jatkossa monimuotoisuutta Java-kielen näkökulmasta.

Alityypitys

- Määritellään tyyppien yhteensopivuus (<:) hyvin vapaamuotoisesti luokkien periytymissuhteen avulla:
  - Tyyppi *T* (luokka) on yhteensopiva itsensä kanssa.
    - *T* <: *T* (refleksiivisyys).
  - Alityyppi *S* (aliluokka) on yhteensopiva ylityyppinsä *T* (yliluokkansa) kanssa.
    - *S* <: *T*.
  - Alityyppi *S* (jälkeläinen) on yhteensopiva kaikkien ylityyppiensä *Q* (esi-isiensä) kanssa.
    - Jos *S* <: *T* ja *T* <: *Q*, niin *S* <: *Q*, (transitiivisuus).
  - Ylityyppi *Q* (esi-isä) ei ole yhteensopiva alityyppiensä *T* (jälkeläistensä) kanssa (antisymmetrisyys).

Alityypitys

- Esim.



- Koska *Object*-luokka on *A*-, *B*-, *C*- ja *D*-luokkien esi-isä (*Object* > { *A*, *B*, *C*, *D* }), niin *A*-, *B*-, *C*- ja *D*-tyypit ovat yhteensopivia *Object*-tyypin kanssa.
- Samoin koska *A*-luokka on *B*- ja *C*- luokkien esi-isä (*A* > { *C*, *B* }), niin *B*- ja *C*-tyypit ovat yhteensopivia *A*-tyypin kanssa.
- Toisaalta esimerkiksi *A*-tyyppi ei ole yhteensopiva *B*-tyypin kanssa, koska *A*-luokka ei ole *B*-luokan jälkeläinen.
- Huomaa myös, että esimerkiksi *C*- ja *D*-tyypit eivät sovi yhteen, koska niitä vastaavien luokkien välillä ei ole periytymissuhdetta.

Alityypitys

- Alityypityksen seurauksena viitteiden *x* (*X*-luokka) ja y (*Y*-luokka) sijoitus *x* = *y*; on sallittu, jos *Y* <: *X* eli *X* ≥ *Y*.
  - Alityyppi käy oman tyyppinsä lisäksi ylityyppiinsä.
- Esimerkki:

```
Object o = null; A a = null;
B b = null; C c = null;
D d = null;
// Oikeellisia sijoituksia.
o = a; // A <: Object
o = b; // B <: Object
o = c; // C <: Object
o = d; // D <: Object
a = b; // B <: A
a = c; // C <: A
// Virheellisiä sijoituksia.
a = o; // Object > A
b = a; // A > B
c = d; // ei periytymissuhdetta
```

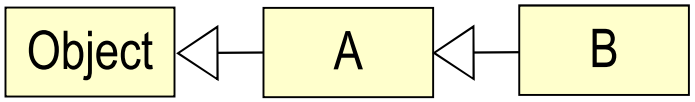
- Vertailussa operadien oltava yhteensopivia, mutta järjestyksellä ei ole väliä.

Alityypitys

- Viite ja siihen liittyvä olio voivat olla eri luokista, kunhan tyypit vain ovat sijoitus-yhteensopivia.
  - Viitteeseen voi liittyä viitteen luokan olioiden lisäksi myös olioita viitteen luokan jälkeläisluokista.
- Nisakas nisse = **new** Kissa();
  - **New**-operaattori palauttaa *Kissa*-tyyppisen nimettömän viitteen. *Nisse*-viitteen sijoitus onnistuu, koska *Kissa*-tyyppi on yhteensopiva *Nisakas*-tyypin kanssa eli Kissa <: Nisakas.
- Object apu = **new** String("abc");
  - Mihin tahansa luokan olioon voidaan sijoittaa *Object*-tyyppinen viite, koska kaikki luokkatyypit ovat yhteensopivia *Object*-tyypin kanssa.

Staattinen ja dynaaminen luokka

- Erotetaan jatkossa viitteen luokka (staattinen luokka) ja olion perusluokka (dynaaminen luokka) toisistaan.
- Dynaaminen luokka vaihtelee staattisen luokan asettamissa rajoissa.
  - Staattinen luokka on dynaamisen luokan "yläraja".



Staattinen luokka	Dynaaminen luokka voi olla
Object	Object, A, B
A	A, B
B	B

```
// Oikein (Object > A)
Object o = new A();
// Väärin (A > B)
B p = new A();
```

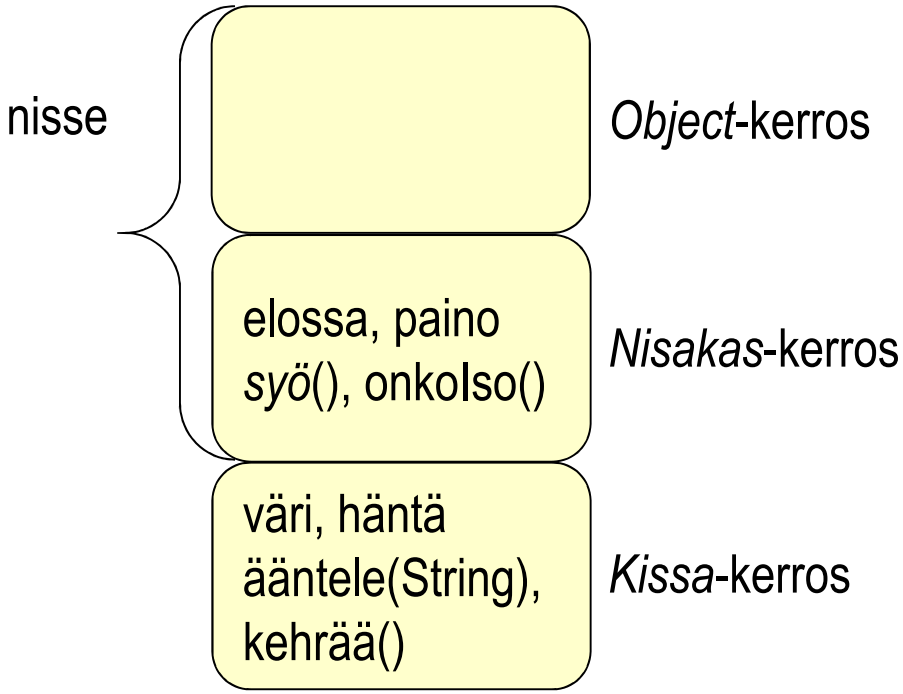
Staattinen ja dynaaminen luokka

Esimerkki	Staattinen luokka	Dynaaminen luokka
Kissa molli = **new** Kissa();	Kissa	Kissa
Nisakas nisse = **new** Kissa();	Nisakas	Kissa
Object eläin; eläin = **new** Kissa(); eläin = **new** Koira();	Object	 Kissa Koira

// Tämä ei käänny,
// koska Object > Kissa.
Kissa mussu = **new** Object();
	
	(Kissa)
		
		(Object)

Staattinen ja dynaaminen luokka

- Java “näkee” olion viitteen luokan (staattinen luokka) kautta: ilman tyyppimuunnosta käytettävissä ovat vain staattisen luokan piirteet.
- // Asetetaan kissaan
  // Nisakas-luokan viite.
  Nisakas nisse = **new** Kissa();
  // Kissa ei osaa nyt esimerkiksi
  // kehrätä, koska staattinen
  // luokka on Nisakas. Alla oleva
  // lause tuottaa käännösvirheen.
  nisse.kehrää();



Myöhäinen ja aikainen sidonta

- Monimuotoisuuden seurauksena
  - metodista voi olla saatavilla useita versioita (korvaus) ja
  - viitteeseen liittyvän olion tyyppi selviää usein vasta ohjelman ajon aikana,

  jolloin metodikutsut täytyy liittää metodien määrittelyihin vasta ohjelman ajon aikana.
- Metodien myöhäinen sidonta tehdään automaattisesti Javan virtuaalikoneen toimesta.
- Myöhäinen sidonta hidastaa ohjelmaa hieman.

Myöhäinen ja aikainen sidonta

- Aikainen sidonta on tyypillistä proseduraalisissa ohjelmointikielissä, joissa operaatioiden kutsut voidaan liittää määrittelyihin jo käännösaikana monimuotoisuuden puuttuessa.
- Aikainen sidonta nopeuttaa ohjelmaa.
- Javan luokkametodit eivät ole tietyn olion, vaan luokkansa omaisuutta ja voidaan siksi sitoa aikaisessa vaiheessa.
  - Ohjelman toimintaa ei ole tarpeen tehostaa tällä kurssilla käyttämällä ilmentymämetodeja, koska runsas **static**-määreiden käyttö tekee ohjelmasta vähemmän oliomaisen.