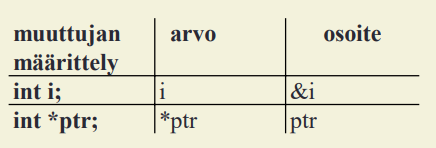
**Osoittimet**

Mikä osoitin on?

Osoitin eli pointteri on muuttuja, jonka arvo on toisen muuttujan (muisti)osoite.

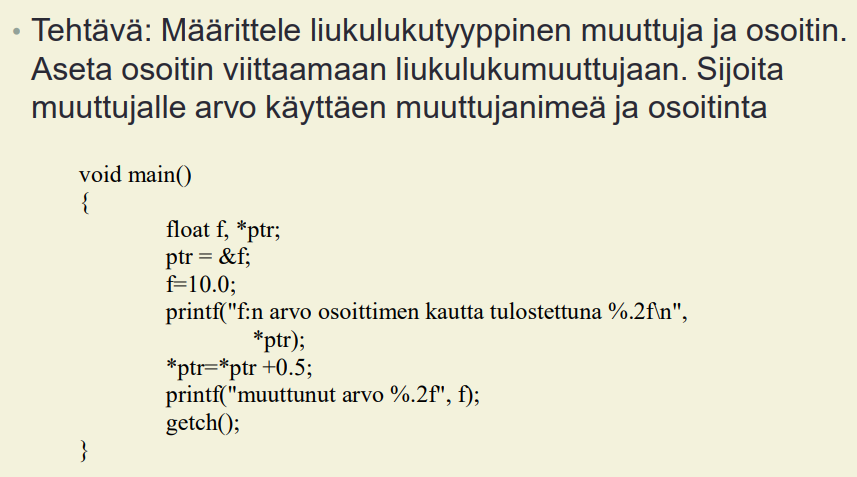
Osoittimen määrittely



Muuttujan määrityksen yhteydessä ilmoitetaan, minkä tyyppiseen muuttujaan osoitin tulee osoittamaan.

int \*ptr;

Esimerkki osoittimen käytöstä



Laskutoimitukset osoittimella

-Sallitut + ja ++ sekä – ja - -

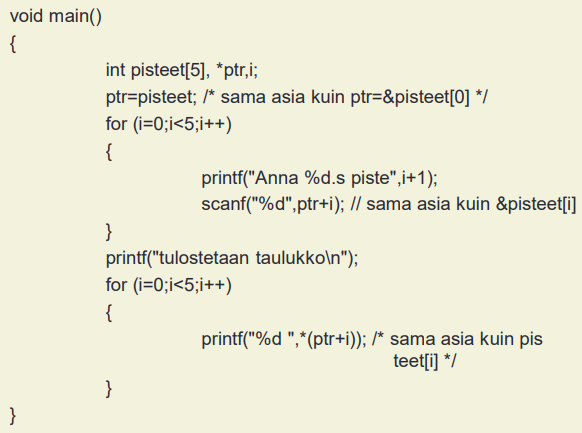
- toisena operandina voi olla ainoastaan kokonaisluku

Taulukko ja osoitin

Taulukon määrittelystä: *float pisteet[10];*

Alustus: *int taulu[]={5, 10, 15, 20, 25};*

Taulukon nimi = osoitin taulukon alkuun



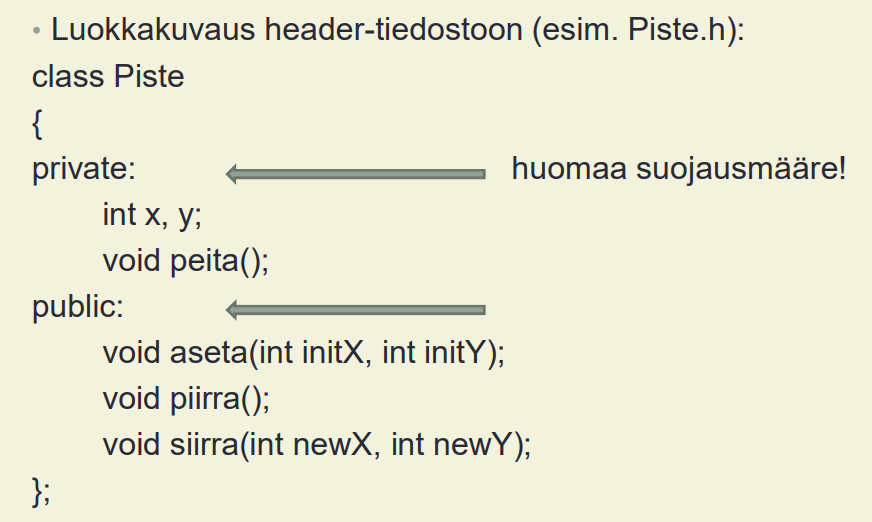
Muistivuoto

Muistivuoto tarkoittaa tilannetta, jossa kasasta varattua muistialuetta ei vapauteta.

“kaikki mitä malloc-funktiolla varataan, pitää free-funktiolla vapauttaa”

**Luokka ja olio**

luokan määrittely



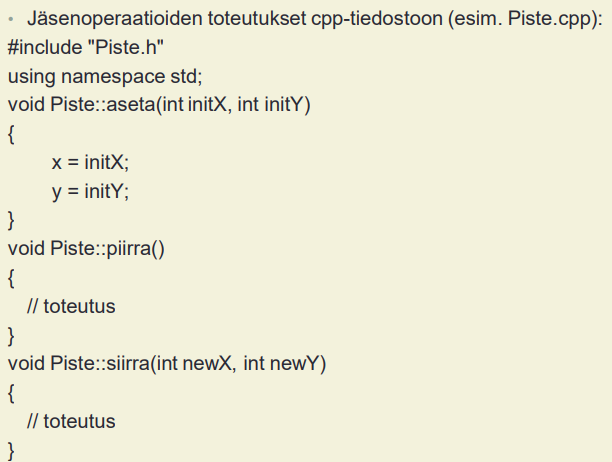
-h, cpp-tiedostot

luokan kuvaus kirjoitetaan h tiedostoon

luokan operaatiot cpp tiedostoon

luokka voidaan kirjoittaa myös yhteen (header) htiedostoon

-luokan jäsenfunktion kirjoitusasu

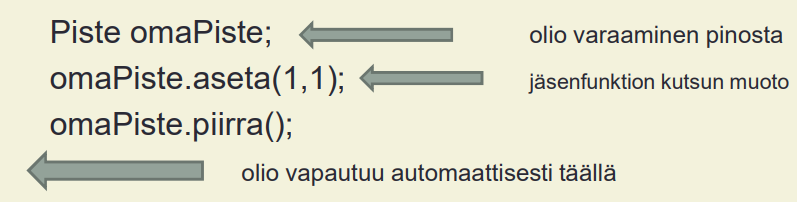
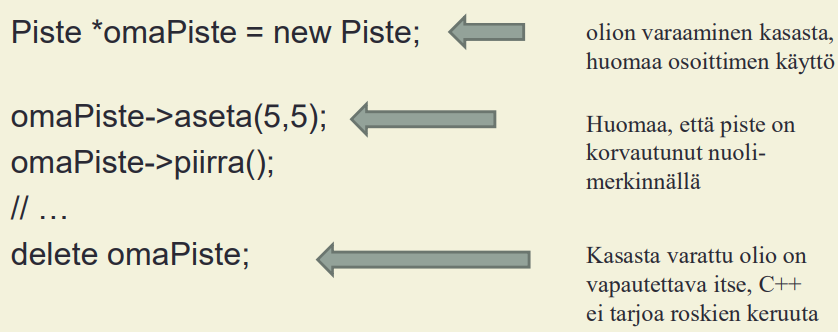


Olion määrittely

Olio on luokan ilmentymä (instance)

Olion varaaminen pinosta/kasasta

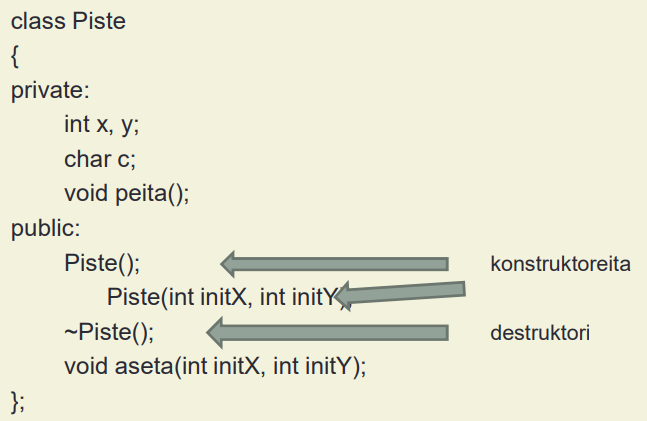
Pino: Kasa:

Konstruktori ja destruktori

**Konstruktori** alustusfunktio (muodostin), jolla voidaan toteuttaa kaikki olion luontiin liittyvät halutut toimenpiteet. Suoritetaan automaattisesti olion luonnin yhteydessä. Voi olla useita, jolloin ne poikkeavat toisistaan parametriensa perusteella. Paluuarvoa ei määritellä.

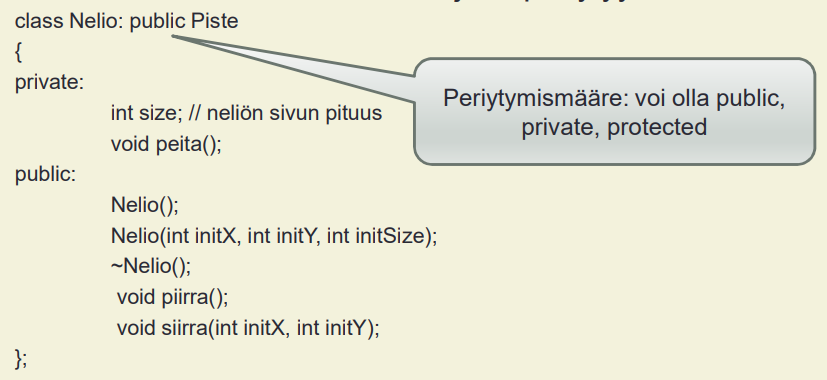
**~ Destruktori** on konstruktorin vastakohta: sitä kutsutaan, kun olio hävitetään. Destruktori on pakko kirjoittaa, mikäli oliossa on rakenteita, jotka on osoittimien avulla varattu kasasta. Destruktoreita voi olla vain yksi ja se on aina parametriton.

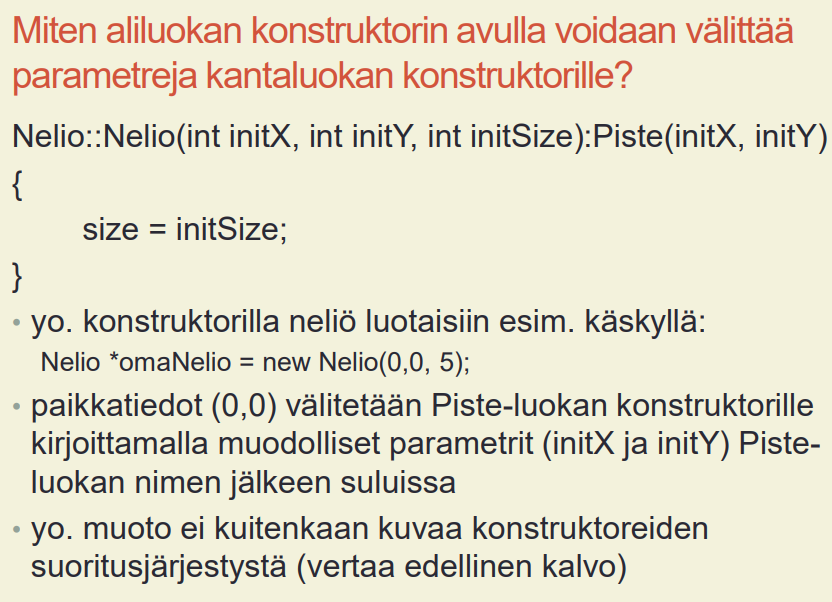


**Periytyminen**

Neliö on **aliluokka** joka periytyy **kantaluokasta** Piste

Konstruktorien ja destruktorien käyttäytyminen:





Suojausmääreiden käyttäytyminen

PUBLIC: kaikki kantaluokan publicmääreellä varustetut jäsenet ovat julkisia myös aliluokassa

PRIVATE: kaikki kantaluokan julkiset jäsenet ovat yksityisiä aliluokassa

…kummassakaan tapauksessa kantaluokan privatemääreellä varustetut jäsenet eivät ole viitattavissa aliluokasta

Muuttuja tyyppi PROTECTED: protected-määre toimii kuten private, mutta mahdollistaa protected-typpisten jäsenien periyttämisen aliluokkaan. Kantaluokan ja aliluokan ulkopuolella protected-tyyppiset jäsenet eivät ole viitattavissa.

Jos periytymismäärä aliluokalla protected: public -> suojatut jäsenet suojattuja aliluokassa, private -> suojatut jäsenet yksityisiä aliluokass, protected -> suojatut ja julkiset jäsenet suojattuja aliluokassa.

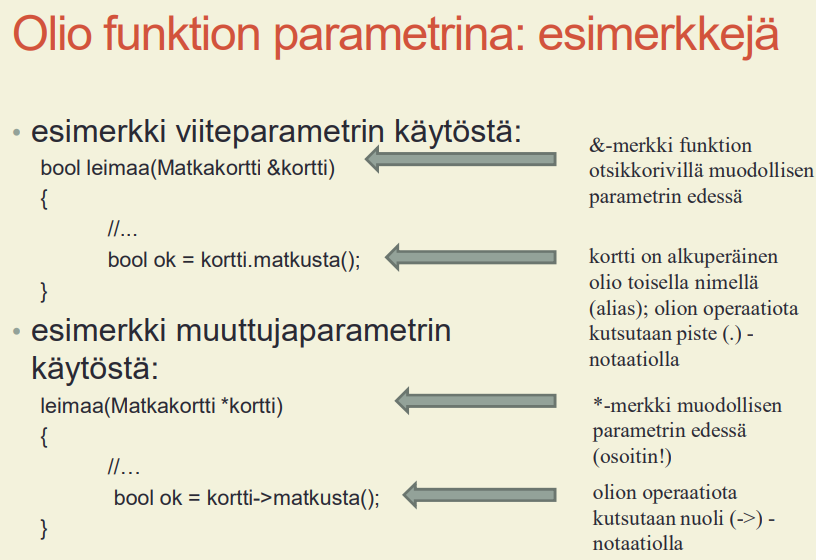
**Parametrien välittäminen**

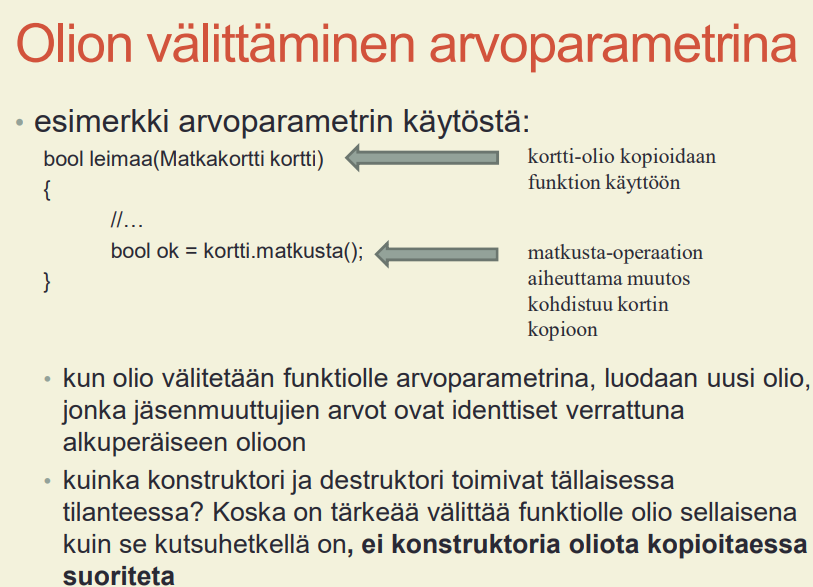
Eri parametrityypit

Arvo, muuttuja, viite, (Olio voidaan välittää kaikilla näillä tavoilla)

Viiteparametri = välitetään viite alkuperäiseen tietoon eli välitetään tieto viiteparametrina. Alkuperäiselle tiedolle annetaan aliohjelmassa (=funktio) uusi nimi, jonka avulla alkuperäiseen tietoon voidaan suoraan viitata uudella nimellä.

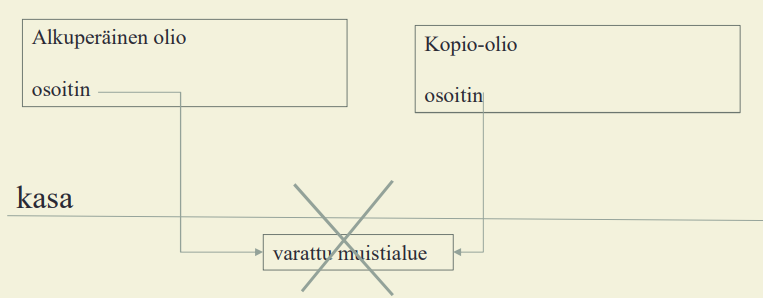
Olion välittäminen funktion parametrina

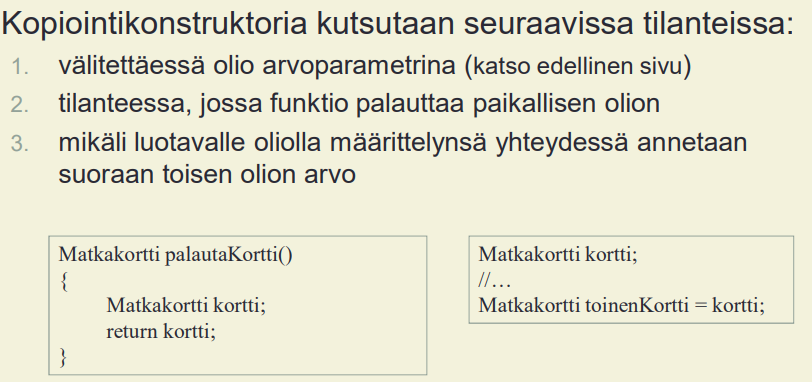




Kopiointikonstruktori

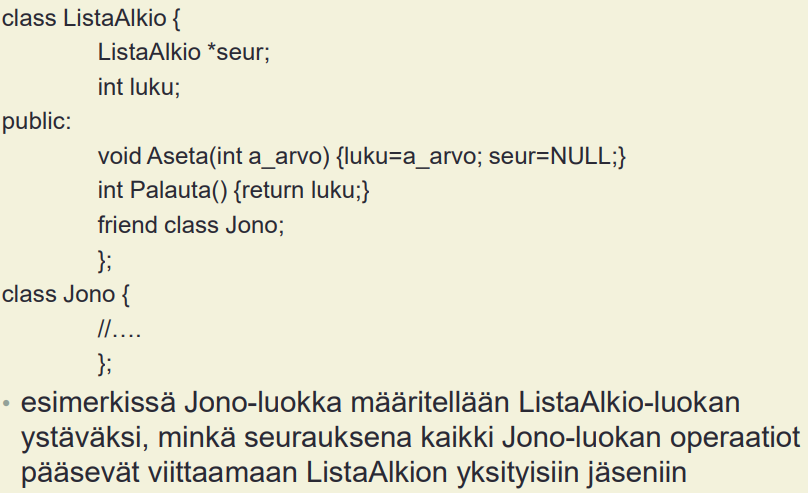
Kun kopio-olio vapautetaan, vapautetaan myös alkuperäisen olion varaama muistialue => alkuperäinen olio korruptoituu





**Ystaväluokka**

= Ystäväluokka on luokka, joka näkee toisen luokan kaikki tietojäsenet ja voi viitata yksityisiin jäseniin julkisten lisäksi (ystäväsuhteen käyttö rikkoo kapseloinnin)



Ystäväfunktio

Luokan yksityisiin jäseniin pääsevät viittaamaan ainoastaan kyseisen luokan jäsenfunktiot. C++:ssa on mahdollisuus määritellä tämän lisäksi **ystäväfunktioita,** joilla on myös oikeus viitata luokan yksityisiin jäseniin.

Prototyyppi esitellään siinä luokassa, jonka ystävä funktio on: avainsana **friend**

Ystäväfunktiossa täytyy aina viitata luokan yksityisiin jäseniin luokan olion avulla

Puolesta/vastaan

**Funktion ja operaattorin kuormittaminen**

Funktion kuormittaminen: esimerkki

Operaattorifunktio

Operaattorifunktio kirjoitetaan samoin kuin tavallinen funktio, paitsi että funktion nimenä on avainsana **operator** ja kuormitettava operaattorimerkki.

-kirjoitusasu

-eri operandityypit

-esimerkki

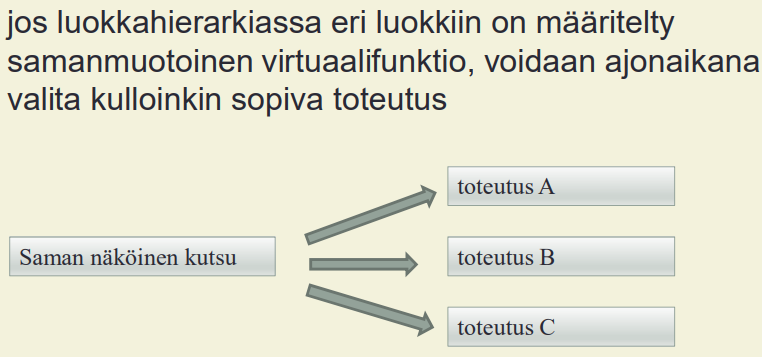
**Myöhäinen sidonta ja virtuaalifunktio**

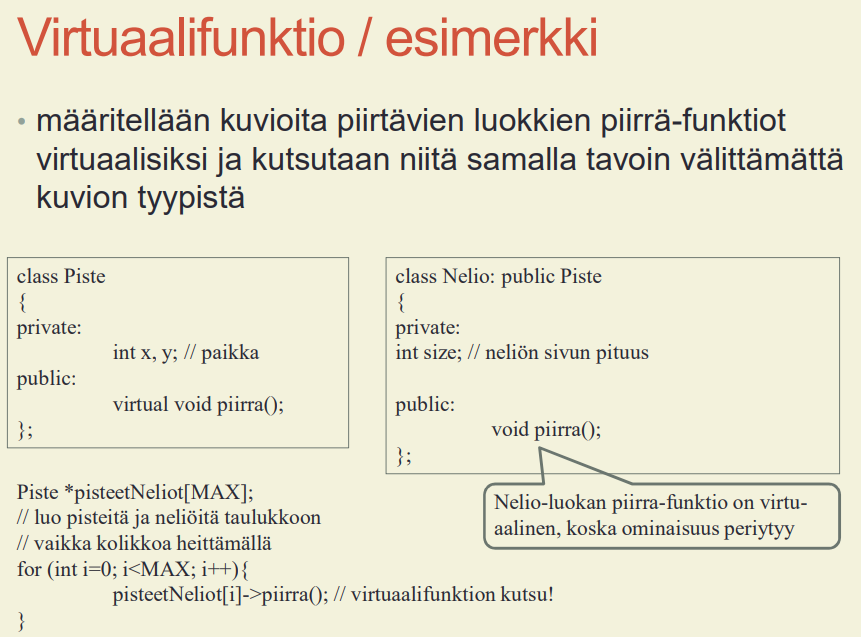
Myöhäinen sidonta?

Toteutus sidotaan funktion kutsuun vasta ohjelman ajon aikana

Virtuaalifunktio

Virtuaalifunktio on funktio, jonka toteutus sidotaan funktion kutsuun vasta ohjelman ajon aikana.





Aito virtuaalifunktio

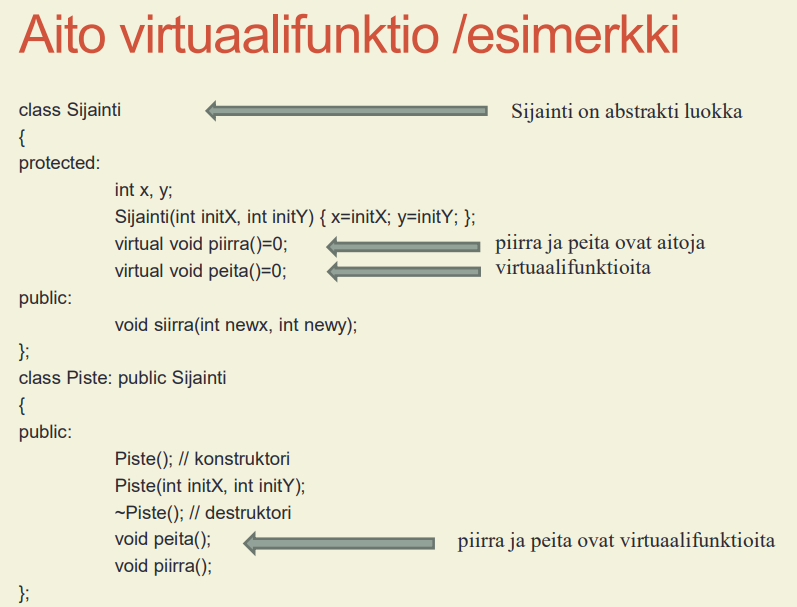
Aito virtuaalifunktio on toteutus, jonka toteutus jätetään kirjoittamatta kantaluokassa -> aliluokasta täytyy löytyä funktion toteutus.

Aidosta virtuaalifuktiosta periytettyjen funktioiden kutsut ovat myöhään sidottuja

Abstrakti luokka

Mikäli luokka sisältää yhdenkin aidon virtuaalifunktion, luokkaa kutsutaan abstraktiksi luokaksi.

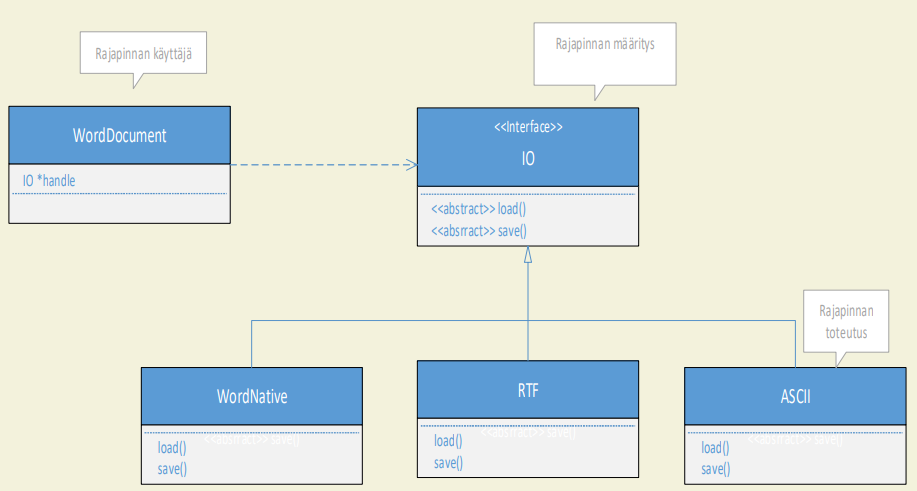
Abstraktia luokkaa ei voi instantioida (luoda siitä oliota), mutta sen tyyppinen osoitin voidaan määritellä.



Rajapinta

Rajapinta kuvaa luokan julkisen käyttäytymisen, jota ohjelmiston muut osat voivat käyttää hyväkseen. Usein käyttäytyminen ymmärretään joukoksi ennalta määriteltyjä operaatioita =funktioita.

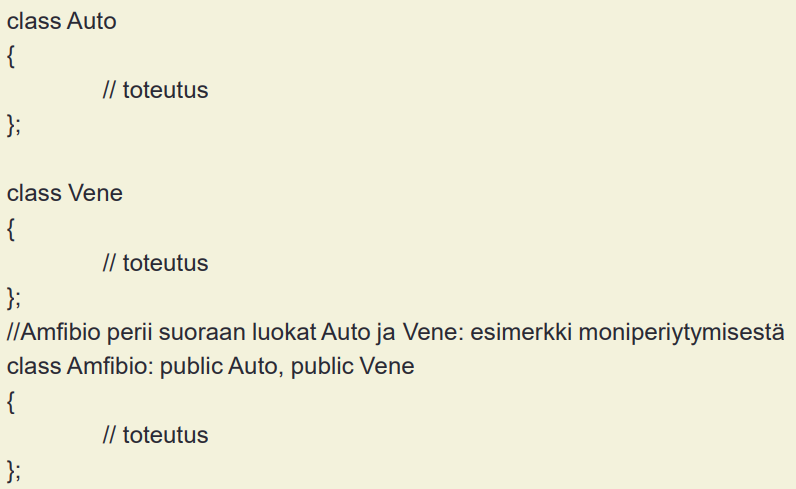
Myöhäisen sidonnan avulla voidaan luoda ajonaikana merkitykseltään muuttuvia rajapintoja.



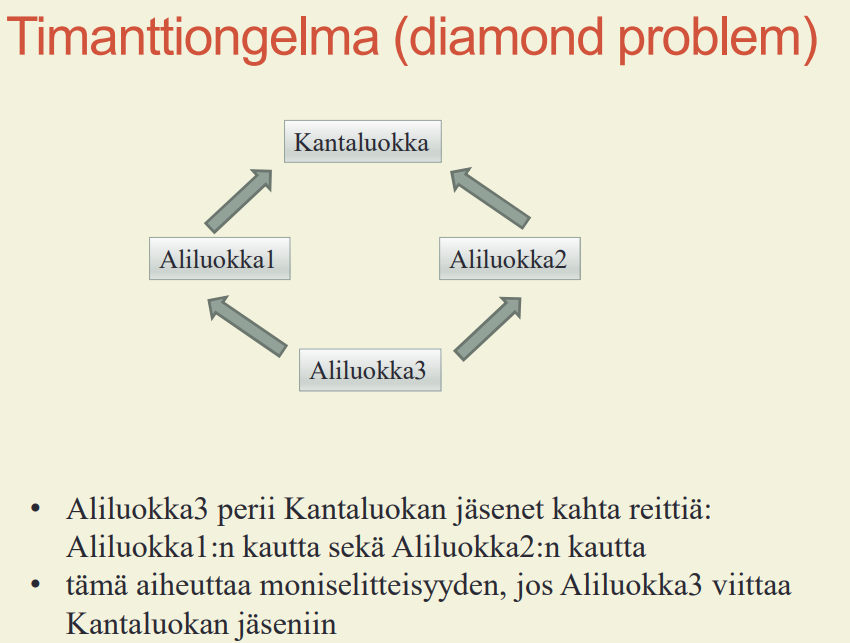
**Moniperiytyminen**

Mitä se on?

Moniperiytyminen tarkoittaa tilannetta, jossa luokka perii useamman kuin yhden konkreettisen kantaluokan.



Moniperiytymiseen liittyvä ongelma



Virtuaaliluokka

Ratkaisu moniperiytymiseen: useampaan kertaan peritystä luokasta nähdään aliluokassa vain yksi kopio.

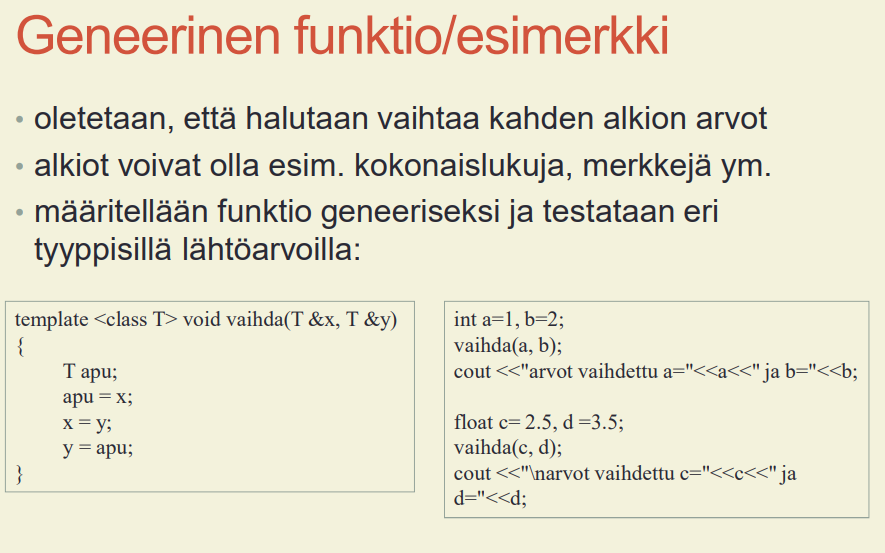
Kantaluokka määritellään virtuaaliseksi sen aliluokan määrityksen yhteydessä, joka kantaluokan suoraan perii. Miikäli periytymisketjussa tämän jälkeen esiintyy tilanne, jossa virtuaaliluokaksi määritelty kantaluokka periytyisi tietylle aliluokalle useampina kopioina, käytetään vain yhtä kopiota. (Virtuaalisuus osoitetaan kummallakin periytymisreitillä, vasemman tai oikeanpuoleisen reitin yhteydessä tehtävä määrittely ei riitä.)

**Geneeriset luokat ja funktiot**

Mitä ovat Geneeriset funktiot?

“samat toiminnot toteutetaan monen tyyppiselle tiedolle”

Geneerinen funktio määrittelee mallin tai pohjan, josta generoidaan geneerisen funktion instanssi eli generoitu funktio, joka sopii funktion kutsussa esiintyviin parametreihin. (Geneerisessä funktiossa käsiteltävän datan tyyppi määrittyy funktion kutsussa esiintyvien parametrien tyypin perusteella.)



Geneeriset luokat

Määritellään samaan tapaan kuin geneerinen funktiokin -> luokan jäsenfunktiot ovat automaattisesti geneerisiä, kun luokka on määritelty geneeriseksi. Tyypillinen toteutus: yleinen tietorakenne, joka tallettaa tietoalkioita niin, että alkion tyyppi on valittavissa vapaasti.