Tampereen yliopisto
Tampere University

Abstraktit kantaluokat
Periytyminen

18.9.2019

Tampereen yliopisto
Tampere University

Abstraktit kantaluokat

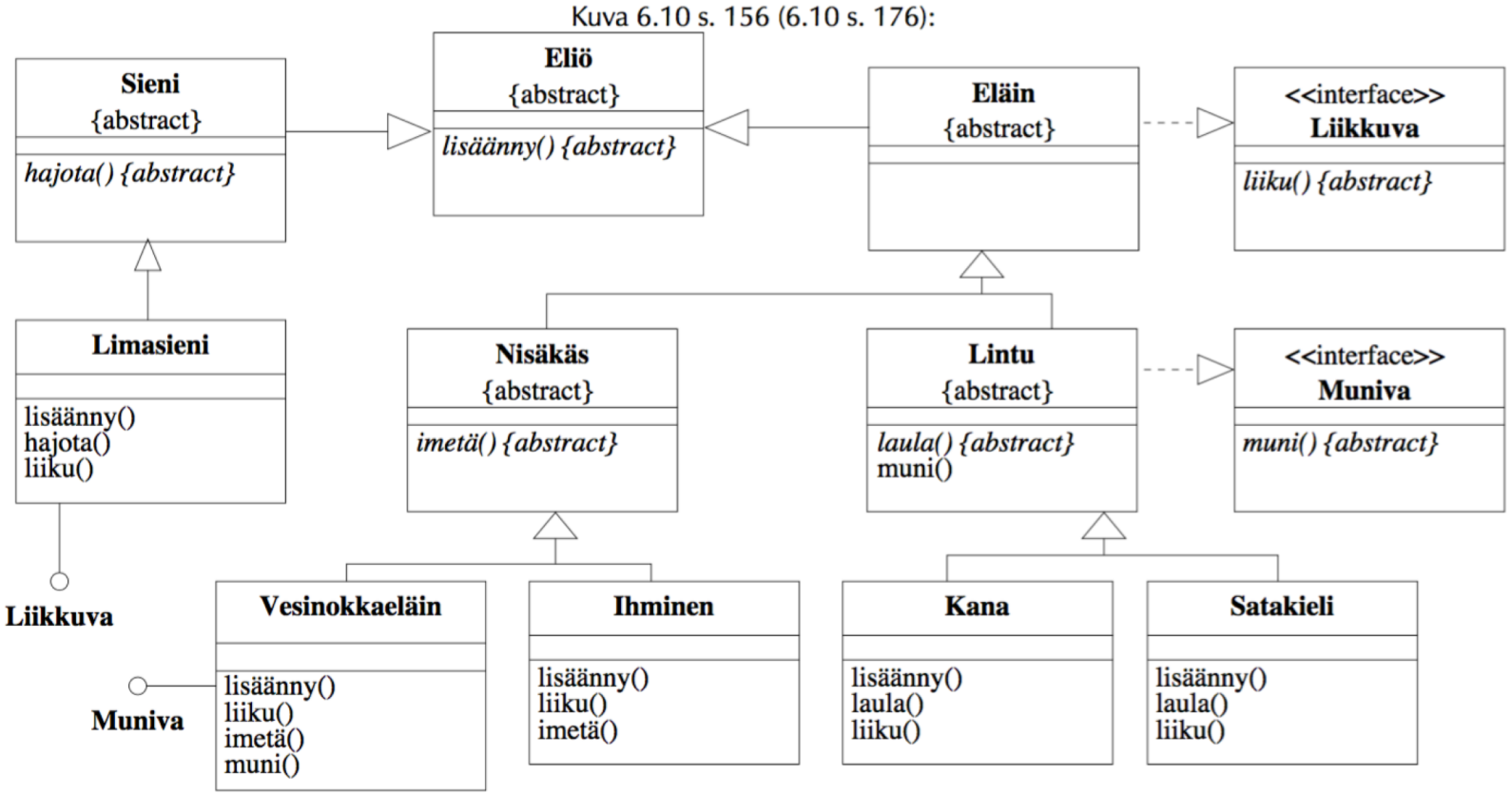
- Abstrakti kantaluokka (abstract base class): ei jäsenmuuttujia, ei jäsenfunktioiden toteutuksia
  - Tarkoitettu käytettäväksi vain kantaluokkana
  - Luokasta ei voi luoda olioita
  - Tyypillisesti rajapintafunktioita, joilla ei (riittävää) toteutusta

Abstraktit kantaluokat

- Paljastaa:
  - pelkkä hierarkkinen rajapinta
- Kätkeä:
  - rajapintafunktioiden toteutus konkreettisiin luokkiin

Rajapintaluokat

Kuva 6.10 s. 156 (6.10 s. 176):



Rajapintaluokan määrittely

```
class Liikkuva
{
public:
     // Kääntäjä tuottaa automaattisesti tyhjän
     //oletusrakentajan
    virtual ~Liikkuva() { } // Upotettu tyhjä
                            // virtuaalipurkaja (inline)
    virtual void liiku(Sijainti paamaara) = 0;
};
```

Periytyminen

Olio-ohjelmoinnin ominaisuus, jossa tehdään uusi luokka olemassa olevan mallin (luokan) pohjalta.

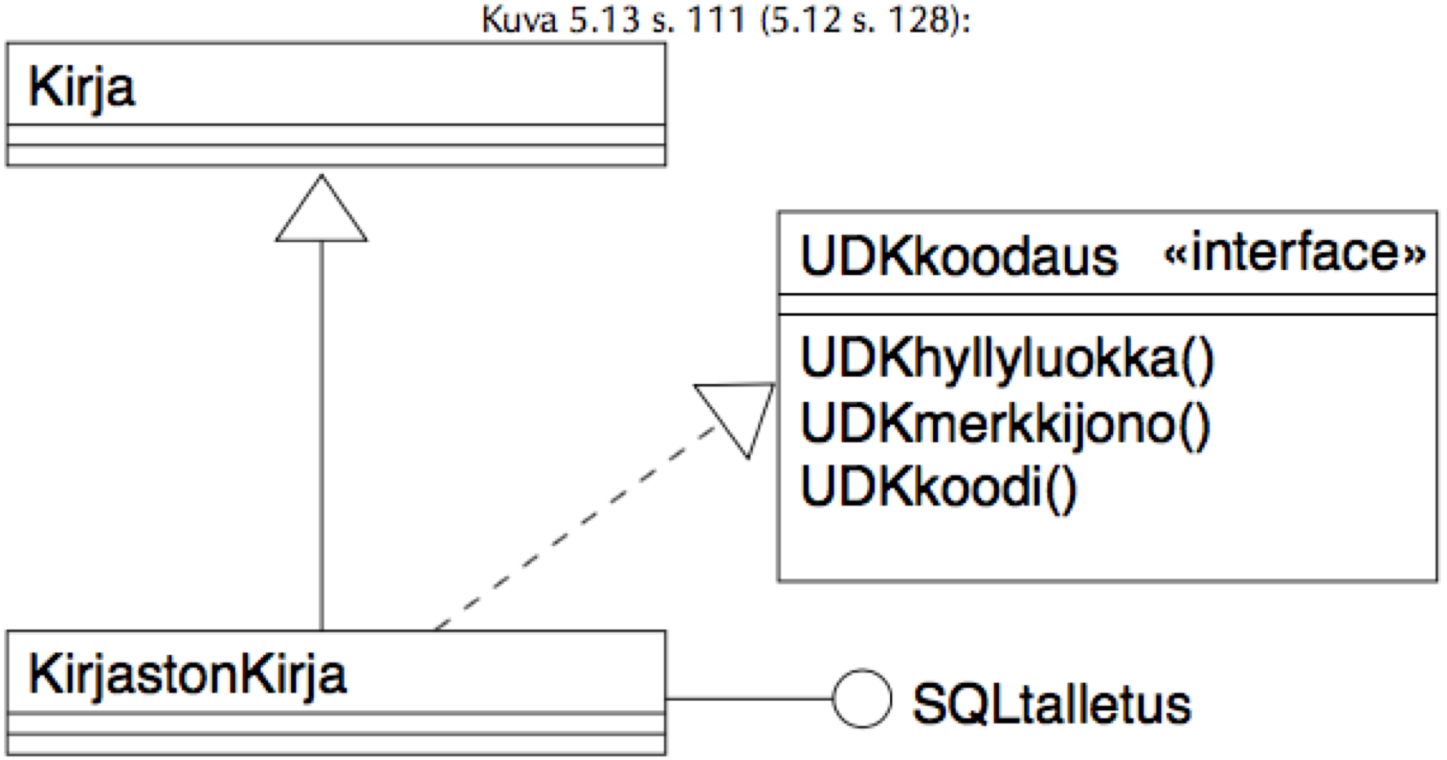
- Aliluokka (periytetty luokka) sisältää kaikki kantaluokan (mallin) ominaisuudet (attribuutit ja rajapinnan)
- Aliluokkaan voidaan lisätä uusia ominaisuuksia ja kantaluokan ominaisuuksia tarvittaessa muuttaa

“is-a”-suhde

- Yksi olio-ohjelmoinnin ja oliosuunnittelun tärkeimmistä ominaisuuksista
- Periytymisellä luodun uuden luokan rajapinta on oletuksena sama kuin kantaluokalla ⇒ uuden luokan sanotaan olevan käyttäytymiseltään myös kantaluokan olio (laajennettu versio siitä)

Periytyminen ja rajapinnat

Kuva 5.13 s. 111 (5.12 s. 128):



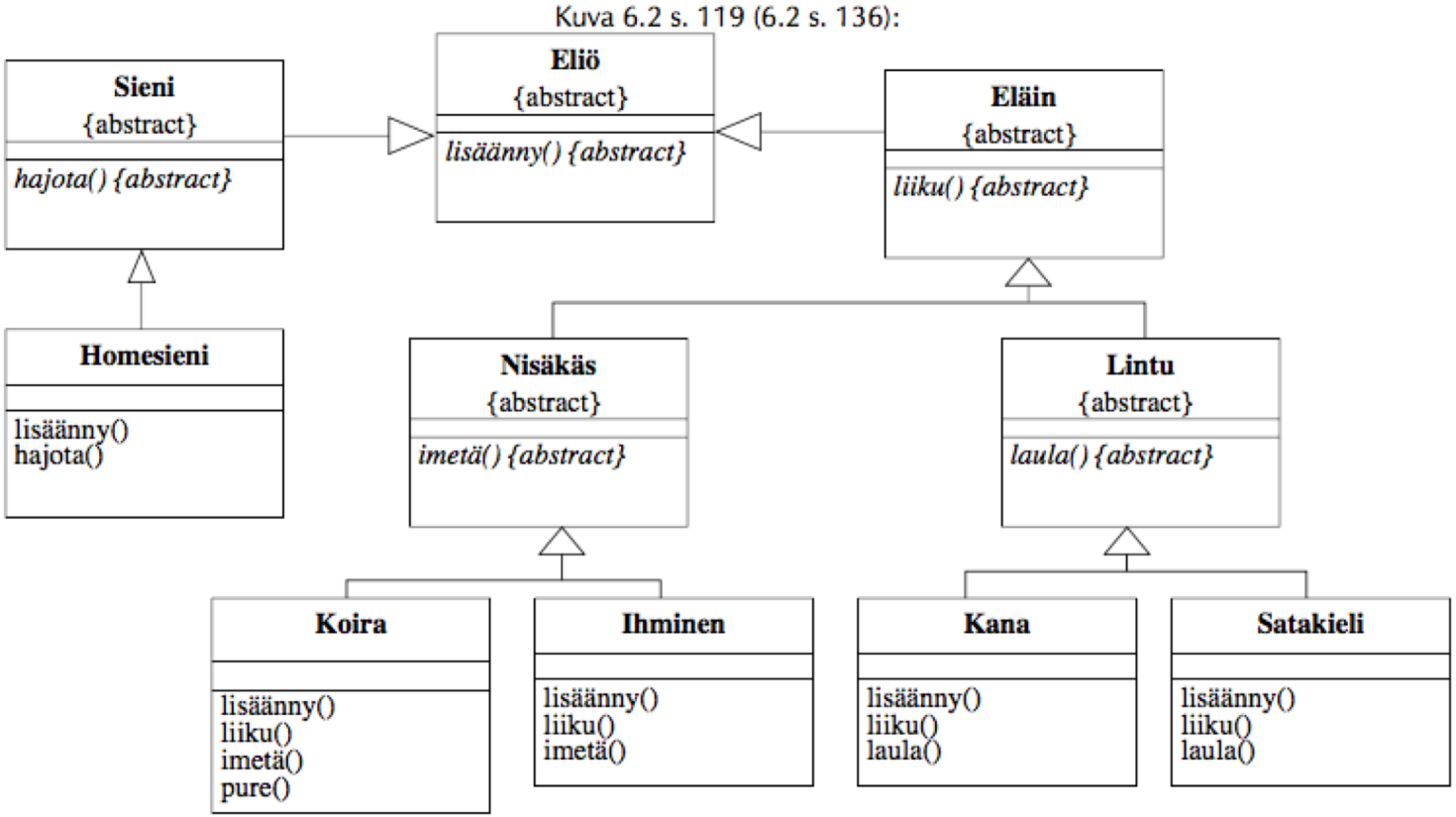
Periytymisen termejä

- **Periytyminen** (*inheritance*), johtaminen
- **Periytymishierarkia** (*inheritance hierarchy*)
- **Kantaluokka** (*base class*), yliluokka (*superclass, parent class*)
- **Aliluokka** (*subclass*), periytetty/johdettu luokka (*derived class*)
- **Esi-isä** (*ancestor*)
- **Jälkeläinen** (descendant)

Periytyminen

- Asioiden luokittelu ja kategorisointi luonnollista ihmiselle
- Käytetty laajalti tieteissä, kielissä jne.
- Olio-ohjelmointi:
  - Ryhmittely yhteisten rajapintojen perusteella
  - Ryhmittely yhteisen toteutuksen perusteella
- Useimmissa kielissä molemmilla sama mekanismi: periytyminen (*inheritance*)

Luokkahierarkia



Luokkahierarkiat

- Erilaisia luokkien rooleja:
  - Rajapintaluokat
    - vain puhtaita virtuaalifunktioita
  - Abstraktit kantaluokat
    - ainakin yksi puhdas virtuaalifunktio
  - Konkreettiset luokat
    - ei puhtaita virtuaalifunktioita

Luokkahierarkiat

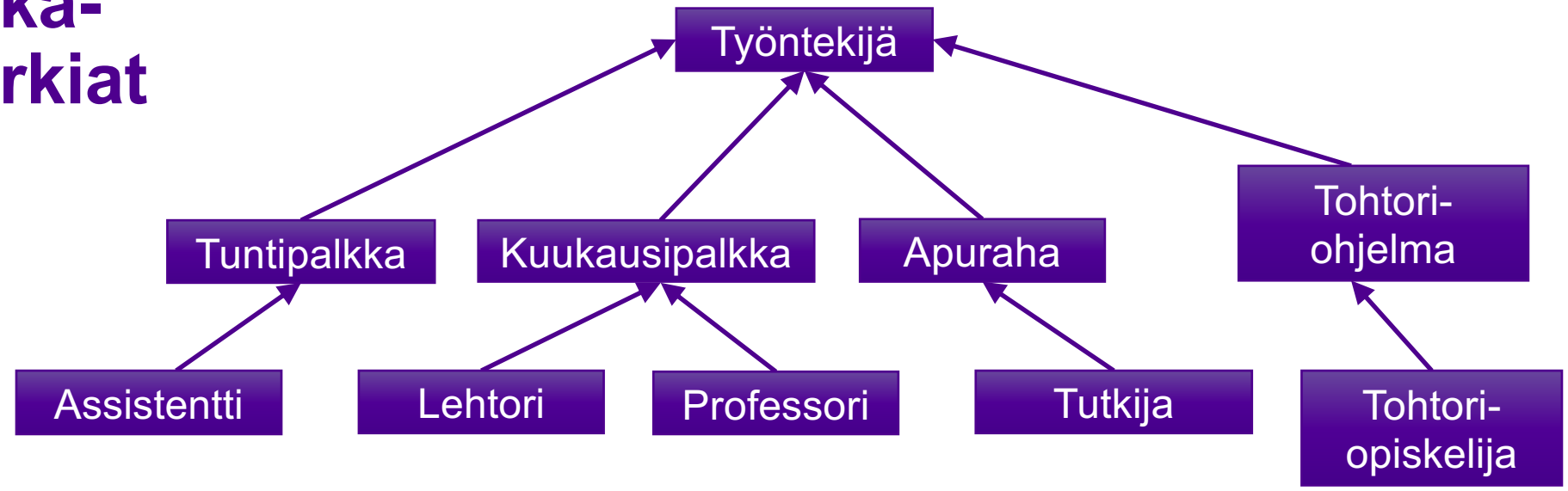
- Hierarkian yläosan luokat
  - “yläkäsitteitä”
  - Määräävät alempien luokkien rajapintaa
  - Polymorfismi: olioihin viittaaminen “yhteisellä nimellä”

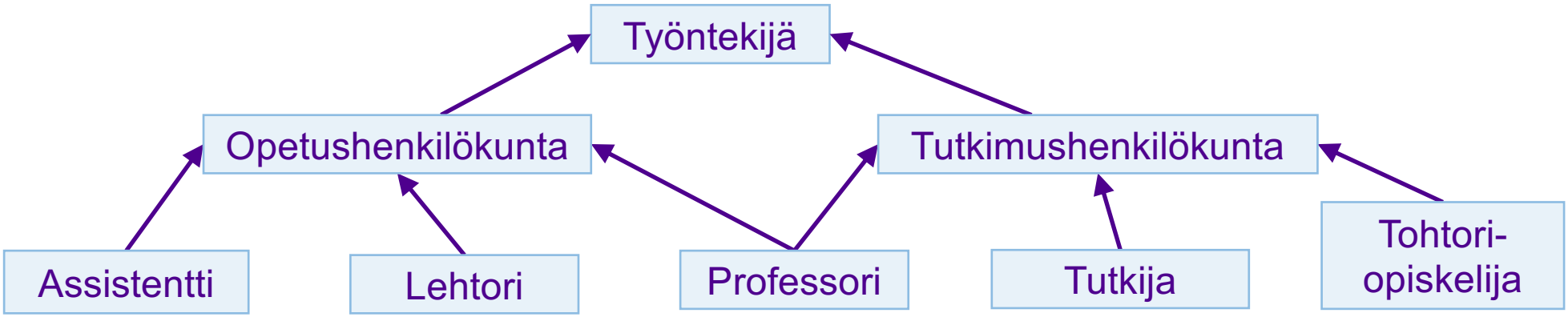
Luokkahierarkiat

- Hierarkian alaosan luokat
  - Palveluiden toteutus
  - Erikoistaminen (*specialization*)
  - Dynaaminen sitominen

Tampereen yliopisto
Tampere University

Luokka-hierarkiat





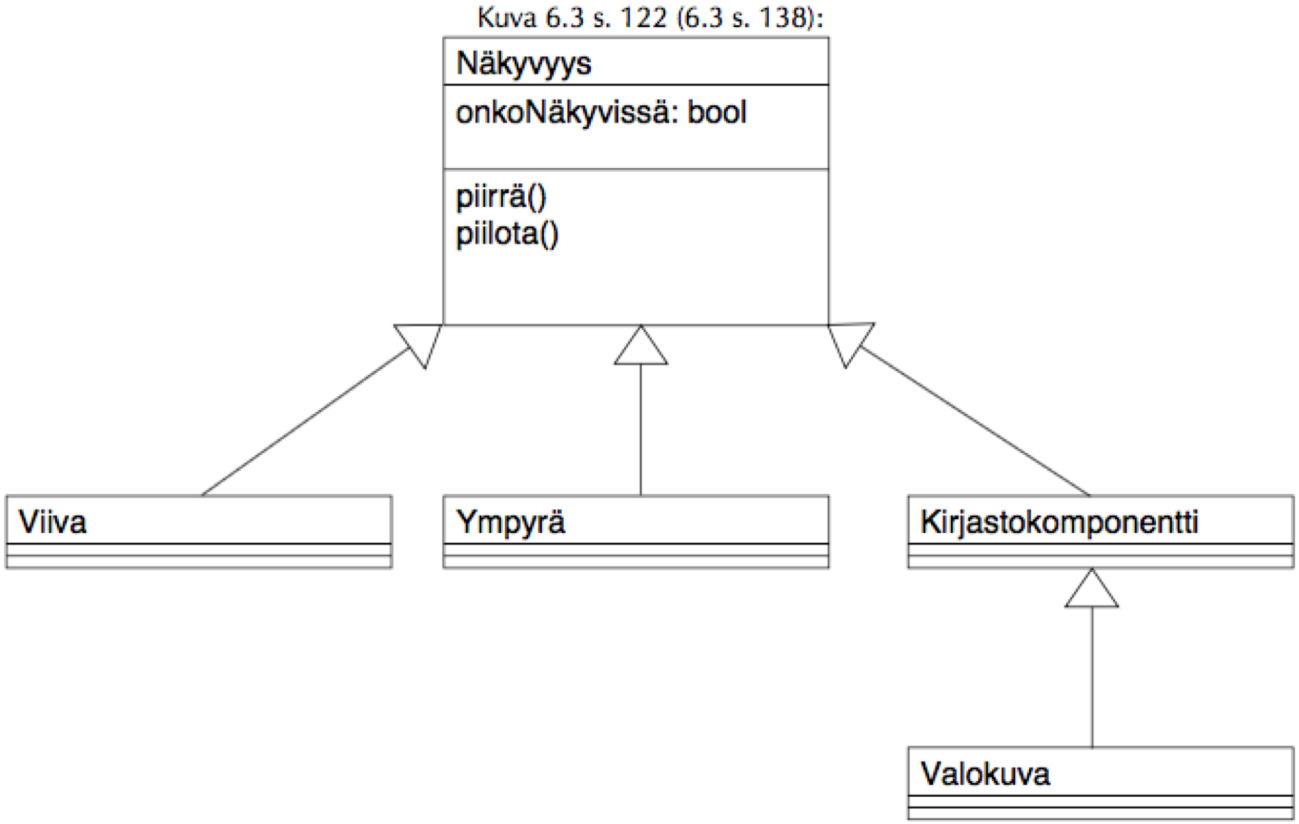
Periytyminen ja uudelleenkäyttö

- Usein luokilla yhteisiä ominaisuuksia
- Yleistys (*generalization*)
  - Yhteiset osat omaan luokkaansa
  - Osat periyttämällä mukaan varsinaisiin luokkiin

Periytyminen ja uudelleenkäyttö

- Aliluokan ei tarvitse kirjoittaa uudelleen kantaluokan jo toteuttamia palveluita
  - Ohjelmakoodin uudelleenkäyttö ⇒ Ei koodin toistoa
- Vaarana koodin pirstoutuminen

Periytyminen ja uudelleenkäyttö



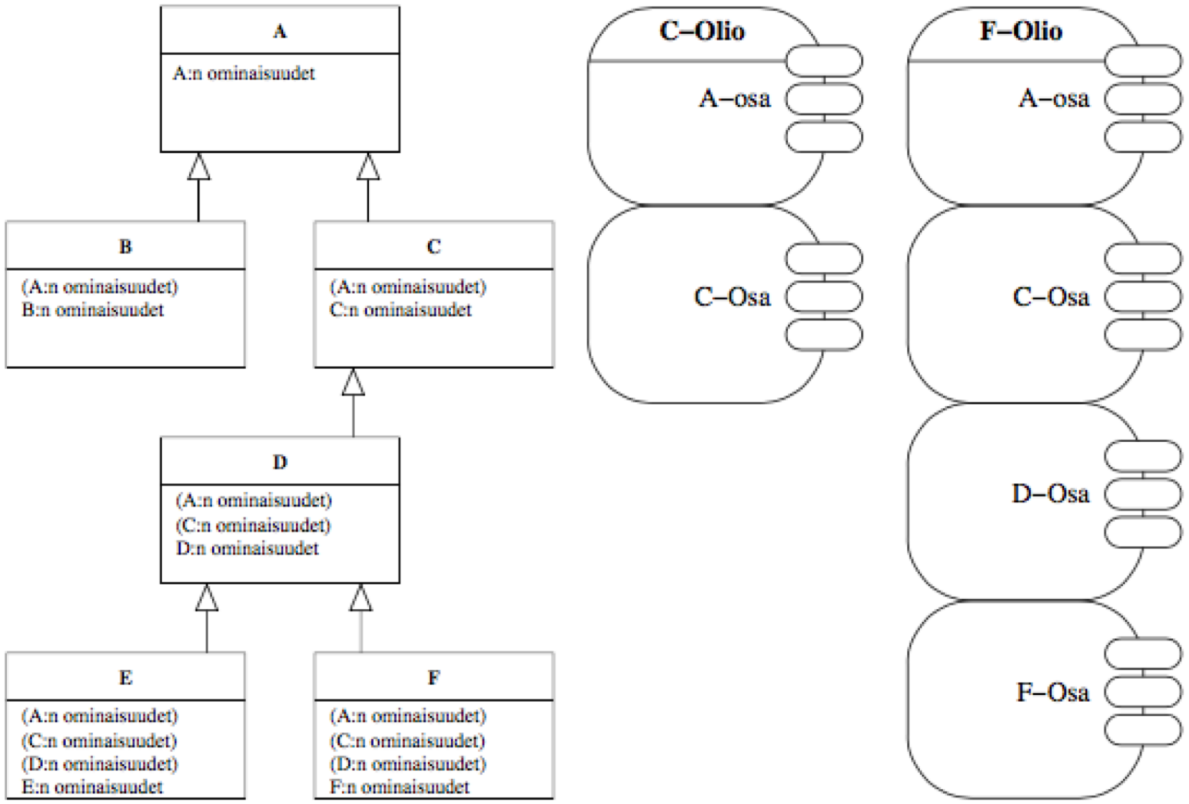
C++: Periytymisen perusteet

- Aliluokka “perii” kantaluokan kaikki ominaisuudet, voi myös lisätä uusia
  - kaikkiin kantaluokan ominaisuuksiin ei pääse käsiksi (suoraan)
- Moniperiytyminen
- Näkyvyysmääreet (public, private, protected) periytymisessä
  - ominaisuuksien näkyvyys
  - periytymistyyppi

Tampereen yliopisto
Tampere University

Periytyminen ja oliot

Kuva 6.4 s. 123 (6.4 s. 139):



C++: Periytymisen syntaksi

```
class A {
    // Luokan A ominaisuudet
};
class B : public A {
    // Luokan B A:han lisäämät ominaisuudet
};
class C : public A {
    // Luokan C A:han  lisäämät ominaisuudet
};
class D : public C {
    // Luokan D C:hen lisäämät ominaisuudet
};
```

C++: Periytymisen syntaksi

```
// ...
class E : public D {
    // Luokan E D:hen lisäämät ominaisuudet
};
class F : public D {
    // Luokan F D:hen lisäämät ominaisuudet
};
```