Tampereen yliopisto
Tampere University

Kopiointi, sijoittaminen ja siirtäminen

9.10.2019

Sijoitus ja kopiointi

- Yleistä “normaalissa” (imperatiivisessa) ohjelmoinnissa
  - Erityisesti perustietotyypeissä
  - Samoin abstrakteissa tietotyypeissä
- Olioiden sijoitus ja kopiointi?

Tampereen yliopisto
Tampere University

Olioiden kopiointi

- Kopiointia tapahtuu automaattisesti arvoparametreissa ja paluuarvoissa
- Mikä oikein on *kopio*?
- Perustyypit: kopio kopioimalla alkuperäisen muistin sisältö

Tampereen yliopisto
Tampere University

Olioiden kopiointi

- Muistikopiointi?: **Ei** — olioon voi kuulua muutakin kuin jäsenmuuttujat
- Kopion tulee olla identtinen?: **Ei** — kopio ei ole sama kuin alkuperäinen
- **Kopion *arvon* tai *tilan* tulee olla sama kuin alkuperäisen**

Olioiden kopiointi

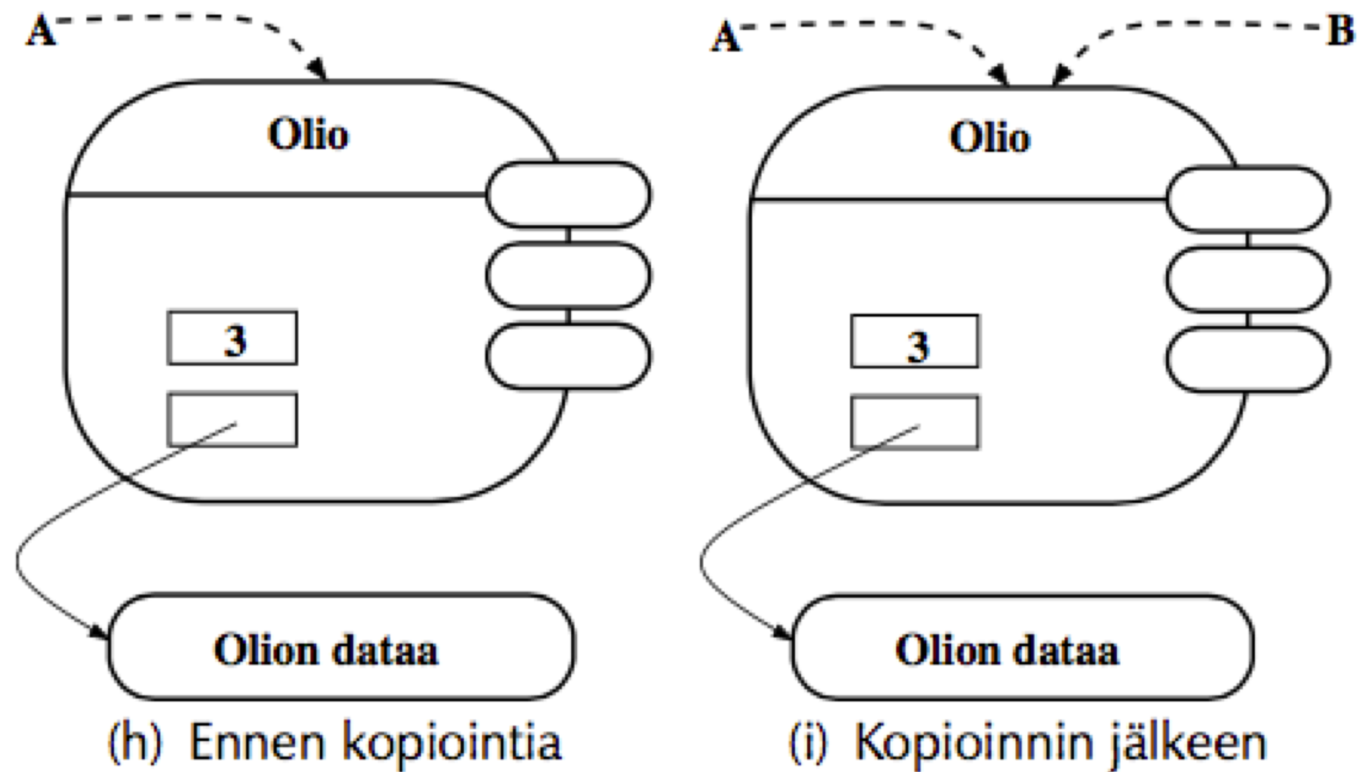
- Kopioinnin merkitys ja tapa riippuu olion tyypistä!
  - Kääntäjä ei välttämättä osaa automaattisesti luoda kopiota
  - Ohjelmoijan täytyy kertoa miten olio kopioidaan
  - Kaikkia olioita ei edes järkevä kopioida → kopioinnin estäminen

Viitekopiointi

- **Idea:** “Kopioidaan” käyttämällä uutena oliona viitettä vanhaan olioon
- Erityisesti kielissä, joissa muuttujat aina viitteitä (Smalltalk, Java)

Viitekopiointi

Kuva 7.1 s. 167 (7.1 s. 187):



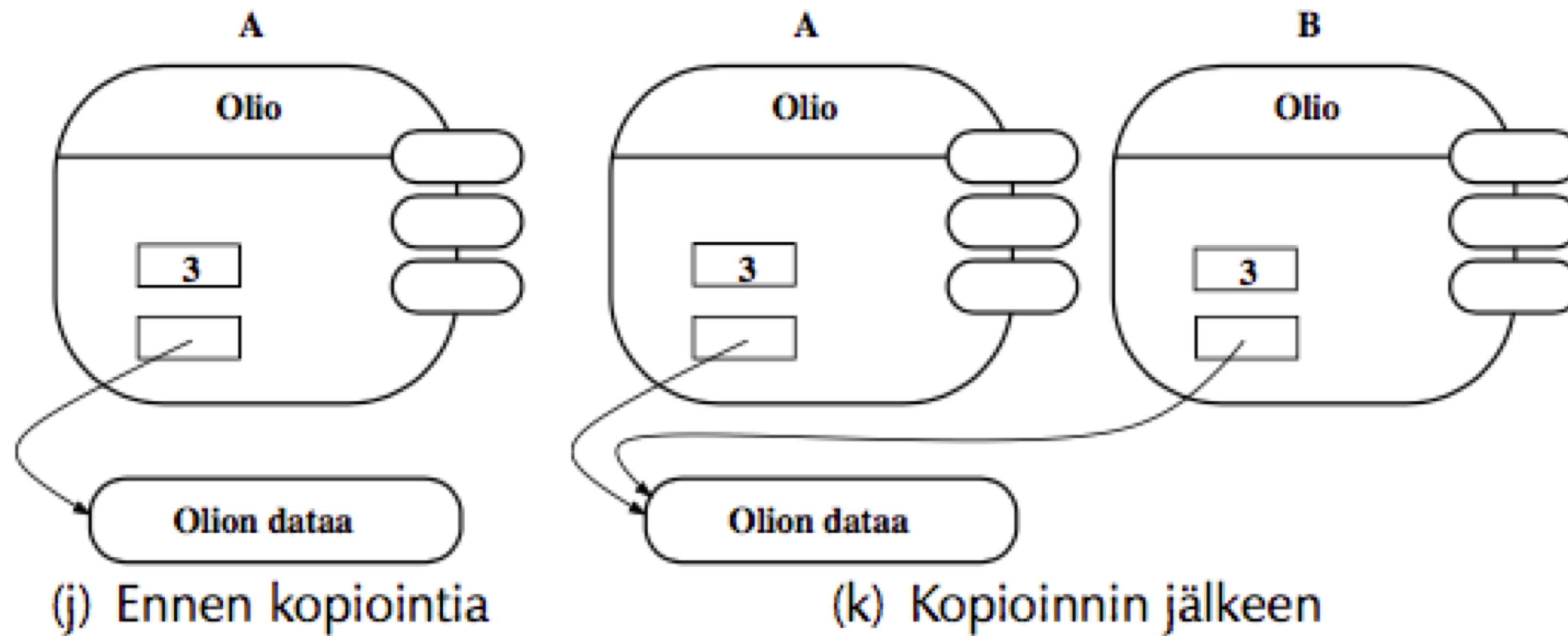
(h) Ennen kopiointia (i) Kopioinnin jälkeen

Matalakopiointi

- **Idea:** Tehdään kopio itse oliosta ja jäsenmuuttujista, olion ulkopuolista dataa ei kopioida
- Helppo toteuttaa ohjelmointikielissä
- **Ongelma:** Osa olion tilaa kuvaavasta datasta voi olla olion ulkopuolella

Tampereen yliopisto
Tampere University

Matalakopiointi



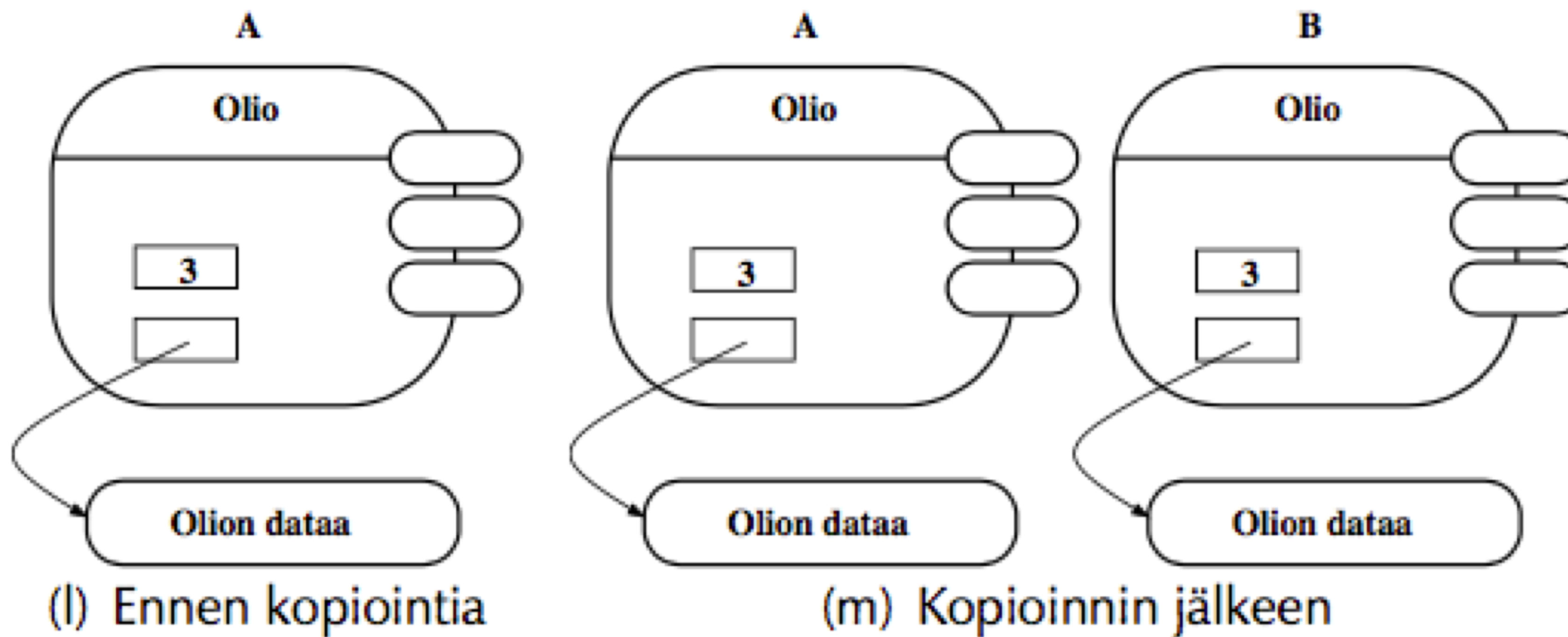
Tampereen yliopisto
Tampere University

Syväkopiointi

- **Idea:** Tehdään kopio myös olion tilaan kuuluvasta ulkopuolisesta datasta → “oikea” tapa kopioida olio
- **Ongelma:** Mitkä olion ulkopuoliset osat ovat osa olion tilaa?
- **Vastaus:** Kääntäjä ei voi päätellä → ohjelmoijan kirjoitettava toteutus

Tampereen yliopisto
Tampere University

Syväkopiointi



C++: Kopiorakentaja

- Olio kopioidaan kopiorakentajalla
- Kopiorakentaja saa viitteen alkuperäiseen olioon
  → voi alustaa uuden olion samanlaiseksi

C++: Kopiorakentaja

- Kopiorakentajan toimintatapoja:
  - Jäsenmuuttujien alustaminen (kopiointi) suoraan alkuperäisistä
  - Muistin varaaminen ja olion ulkopuolisen data kopiointi
    - ...

Merkkijonon kopiorakentaja

```
class Mjono
{
public:
    Mjono(char const* merkit);
    Mjono(Mjono const& vanha); // Kopiorakentaja
    virtual ~Mjono();
private:
    unsigned long koko_;
    char* merkit_;
};
```

Tampereen yliopisto
Tampere University

Merkkijonon kopiorakentaja

```
Mjono::Mjono(Mjono const& vanha) : koko_(vanha.koko_),
                                   merkit_(0)
{
    if (koko_ != 0)
    { // Varaa tilaa, jos koko ei ole nolla
        merkit_ = new char[koko_ + 1];
        for (unsigned long i = 0; i != koko_; ++i) {
            merkit_[i] = vanha.merkit_[i]; // Kopioi merkit
        }
        merkit_[koko_] = '\0'; // Loppumerkki
    }
}
```

Periytyminen ja kopiorakentaja

- Kopiorakentaja on rakentaja → **Aliluokassa kopiorakentajan kutsuttava kantaluokan kopiorakentajaa**
- Kantaluokan kopiorakentaja: kantaluokkaosan alustaminen kopioksi
- Aliluokan kopiorakentaja: aliluokkaosan alustaminen kopioksi

Tampereen yliopisto
Tampere University

Päivätyn merkkijonon kopiorakentaja

```
class PaivattyMjono : public Mjono {
  public:
    PaivattyMjono(char const* merkit, Paivays const& paivays);
    PaivattyMjono(PaivattyMjono const& vanha); // Kopiorakentaja
    virtual ~PaivattyMjono();
  private:
    Paivays paivays_;
};

// Olettaa, että Paivays-luokalla on kopiorakentaja
PaivattyMjono::PaivattyMjono(PaivattyMjono const& vanha) :
                             Mjono(vanha),paivays_(vanha.paivays_)
{
}
```

Oletuskopiorakentaja

- Luokalle ei kirjoitettu kopiorakentajaa → kääntäjä tarjoaa “oletusarvoisen” kopiorakentajan
- Oletusarvoinen kopiorakentaja toimii kopioimalla jäsenmuuttujat suoraan
  - Toimii yksinkertaisissa luokissa
  - Jos luokassa osoittimia tai luokka muuten monimutkainen → Oletusarvoinen kopiointi toimii yleensä väärin!

Tampereen yliopisto
Tampere University

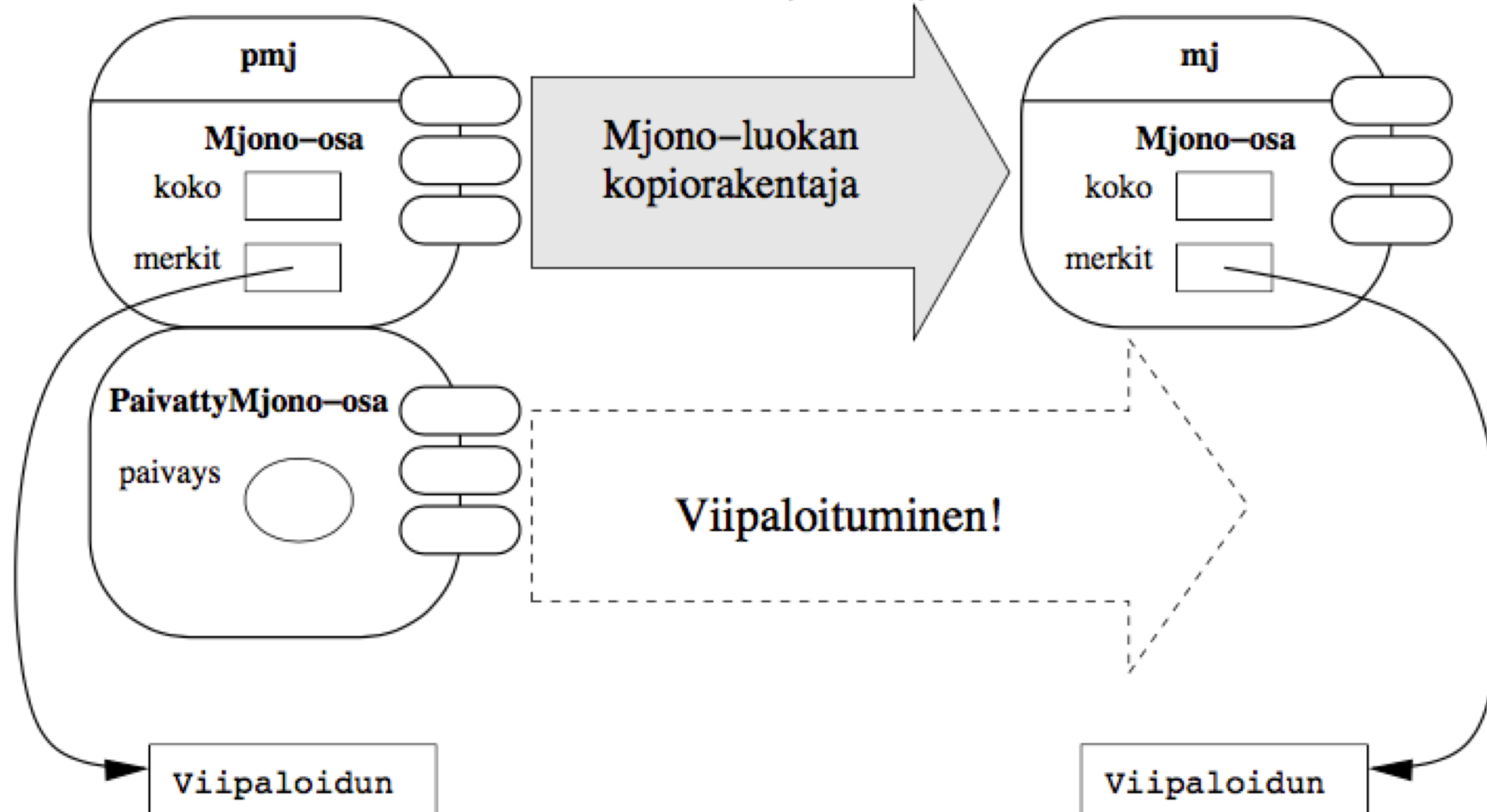
Kopioinnin estäminen

- Joskus kopiointi ei mielekästä → tulisi olla mahdotonta
- Kopiorakentajaa ei kirjoitettu → oletusarvoinen kopiorakentaja → kopiointi mahdollista (väärin)
- Kopiointi estetään lisäämällä **julkiseen** rajapintaan (C++11):

```
Luokka(const Luokka& arvo) = delete;
```

Viipaloituminen

Kuva 7.4 s. 173 (7.4 s. 194):



Tampereen yliopisto
Tampere University

Viipaloitumisen kiertäminen

```
class Mjono :
{
public:
    Mjono(Mjono const& m);
    virtual Mjono* kloonaa() const;
    …
};

Mjono* Mjono::kloonaa() const
{
    return new Mjono(*this);
}
```

```
class PaivattyMjono : public Mjono
{
public:
    PaivattyMjono(PaivattyMjono const& m);
    virtual PaivattyMjono* kloonaa() const;
    …
};

PaivattyMjono* PaivattyMjono::kloonaa() const
{
    return new PaivattyMjono(*this);
}
```

Tampereen yliopisto
Tampere University

Viipaloitumisen kiertäminen

```
void kaytakopiota(Mjono const& mj)
{
    Mjono* kopiop = mj.kloonaa(); // Tulos voi
                                   // olla periytetyn
                                   // kopio
    // Täällä sitten käytetään kopiota
    delete kopiop; kopiop = 0; // Pitää muistaa
                               // myös tuhota
}
```

Tampereen yliopisto
Tampere University

Olioiden sijoittaminen

- Sijoituksen lopputulos sama kuin kopioinnin
- Sijoituksessa alkuperäinen olio olemassa ennestään → sillä myös “vanha arvo”
- Sijoituksen ongelmat usein juuri vanhaan arvoon liittyviä

Tampereen yliopisto
Tampere University

Olioiden sijoittaminen

- Jos varauduttava virheisiin:
  - Mitä jos virhe puolessa välissä sijoitusta?
  - Palautetaanko vanha arvo virheen sattuessa? → Vaikeaa
- Joskus sijoittaminen ei mielekästä → estettävä
- Joskus sijoittaminen ei mielekästä vaikka kopiointi onkin

Tampereen yliopisto
Tampere University

C++: Sijoitusoperaattori

- C++:ssa sijoitus **sijoitusoperaattorilla** (assignment operator):

  operator = (tai **operator=**)

- Yleensä yhdistelmä purkajan ja kopiorakentajan toimintaa

Tampereen yliopisto
Tampere University

a=b

- Kutsutaan **a**:n sijoitusoperaattoria, jolle **b** viiteparametrina
- Toiminta: tuhotaan **a**:n vanha arvo, korvataan se **b**:n arvolla
- Palauttaa paluuarvona viitteen olioon itseensä, tässä **a**:han (ketjusijoitus a = b = c)
- "Oliomainen" syntaksi: a.**operator** =(b)

Merkkijonon sijoitusoperaattori

```
class Mjono
{
public:
    …
    Mjono& operator =(Mjono const& vanha);
private:
    unsigned long koko_;
    char* merkit_;
};
```

Tampereen yliopisto
Tampere University

Merkkijonon sijoitus-operaattori

```
Mjono& Mjono::operator =(Mjono const& vanha) {
    if (this != &vanha) { // Jos ei sijoiteta itseen
        delete[] merkit_; merkit_ = 0; // Vapauta vanha
        koko_ = vanha.koko_; // Sijoita koko
        if (koko_ != 0) { // Varaa tilaa, jos koko ei nolla
            merkit_ = new char[koko_ + 1];
            for (unsigned long i = 0; i != koko_; ++i)
            { // Kopioi merkit
              merkit_[i] = vanha.merkit_[i];
            }
            merkit_[koko_] = '\0'; // Loppumerkki
        }
    }
    return *this;
}
```

Itseen sijoittaminen

- Sijoitus **a = a** typerä, mutta mahdollinen
- Vaara normaalissa sijoitusoperaatiossa:
  - Aluksi vapautetaan vanhaan arvoon liittyvät muistipaikat ja muut resurssit
  - Sitten varataan uutta muistia ja tehdään varsinainen sijoitus
  - Itseensijoituksessa uusi arvo on sama kuin vanha arvo
  - → Ei toimi itseensijoituksessa!

Tampereen yliopisto
Tampere University

Itseen sijoittaminen

- Helppo ratkaisu:
  - Itseensijoituksen ei pitäisi tehdä mitään

  → Tarkastetaan, ollaanko sijoittamassa itseen
  → Jos ollaan, ei tehdä mitään
  → Jos ei, tehdään normaali sijoitus
  – Tarkastus vertailemalla **this**-osoitinta ja osoitinta parametriin

Tampereen yliopisto
Tampere University

Periytyminen ja sijoitusoperaattori

- Sijoituksessa sama vastuujako kuin kopioinnissa:
  - Aliluokan sijoitusoperaattori: kutsuu kantaluokan sijoitusta, sijoittaa aliluokkaosan
  - Kantaluokan sijoitusoperaattori: sijoittaa kantaluokkaosan
  - Aliluokan sijoitusoperaattorissa muistettava kutsua kantaluokan sijoitusoperaattoria!
- Kääntäjä ei varoita, jos kutsu unohtuu!

Tampereen yliopisto
Tampere University

Aliluokan sijoitus-operaattori

```
class PaivattyMjono : public Mjono {
  public:
    PaivattyMjono& operator =(PaivattyMjono const& vanha);
    // ...
  private:
    Paivays paivays_;
};

PaivattyMjono& PaivattyMjono::operator =(PaivattyMjono const& vanha) {
    if (this != &vanha) { // Jos ei sijoiteta itseen
        Mjono::operator =(vanha); // Kantaluokan sijoitusoperaattori
        // Oma sijoitus, oletetaan että Paivays-luokalla on
        // sijoitusoperaattori
        paivays_ = vanha.paivays_;
    }
    return *this;
}
```

Oletussijoitusoperaattori

- Luokalle ei kirjoitettu sijoitusoperaattoria → kääntäjä tarjoaa “oletusarvoisen” sijoitusoperaattorin
- Oletusarvoinen sijoitusoperaattori toimii sijoittamalla jäsenmuuttujat suoraan
  - Toimii yksinkertaisissa luokissa
  - Jos luokassa osoittimia tai luokka muuten monimutkainen → Oletusarvoinen sijoitus toimii yleensä väärin!

Tampereen yliopisto
Tampere University

Sijoituksen estäminen

- Joskus sijoitus ei mielekästä → tulisi olla mahdotonta
- Sijoitusoperaattoria ei kirjoitettu → oletusarvoinen sijoitusoperaattori → sijoitus mahdollista (väärin)
- Estäminen: julkiseen rajapintaan

```
Luokka& operator=(const Luokka& arvo) = delete;
```

Tampereen yliopisto
Tampere University

Viipaloituminen ja sijoitus

- Viipaloituminen vaarana myös sijoituksessa
- Aliluokan olio on kantaluokan olio → sen voi sijoittaa kantaluokan olioon
- Kantaluokkaviitteiden ja -osoittimien kautta mahdollista myös aliluokkaolion sijoittaminen toisen aliluokan olioon!
- Sijoituksen viipaloituminen vaarana myös muissa oliokielissä (mutta niissä ei yleensä sisäänrakennettua oliosijoitusta)

Siirtäminen (C++11)

- "The purpose of a move constructor is to steal as many resources as it can from the original object as fast as possible, because the original does not need to have a meaningful value …"
- Nopeampaa kuin kopiointi!

Tampereen yliopisto
Tampere University

Siirtäminen

- Siirtorakentaja:

```
    Luokka( Luokka&& vanha );
```

- Siirtosijoitus:

```
    Luokka& operator =( Luokka&& toinen );
```

- Olennaista harkita, haluaako siirtämistä tapahtuvan