Tampereen yliopisto
Tampere University

Dynaaminen sitominen ja virtuaalifunktiot

25.9.2019

Tampereen yliopisto
Tampere University

Virtuaalifunktiot

- Aliluokalla oma *toteutus* kantaluokan palvelulle
- Aliluokka perii *rajapinnan,* ei toteutusta
- Mahdollista C++:ssa, jos kantaluokassa jäsenfunktio **virtuaalinen**, eli se on **virtuaalifunktio** (*virtual function*)
  - avainsana: `virtual`

Tampereen yliopisto
Tampere University

Virtuaalifunktiot

- Aliluokan vaihtoehdot:
  - Hyväksyä kantaluokan toteutus
  - Kirjoittaa oma toteutus (usein kutsuu kantaluokan toteutusta)
  - Parametreja ja paluutyyppiä **ei voi** muuttaa

Tampereen yliopisto
Tampere University

Dynaaminen sitominen

- Virtuaalifunktiot → jäsenfunktion rajapinta voi olla eri tasolla kuin toteutus
- Kutsuttavan toteutuksen käännösaikainen päättely usein mahdotonta
- Kutsuttava funktio sidotaan (valitaan) ajoaikana (dynaamisesti)

Dynaaminen sitominen

- Päätös, mitä toteutusta kutsua, *ajoaikainen*
- Osoittimet/viitteet:
  - Osoittimen päässä kantaluokan tai aliluokan olio
  - Kutsuttava toteutus riippuu olion luokasta

    → sama kutsu, eri toteutus oliosta riippuen

Lisäys luokkaan Kirja

```
class Kirja
{
  public:
    virtual void tulostaTiedot(std::ostream& virta) const;
    virtual bool sopiikoHakusana(std::string const& sana) const;
  private:
    void tulostaVirhe(std::string const& virheteksti) const;
};
```

Lisäys luokkaan Kirja

```
void Kirja::tulostaVirhe(string const& virheteksti) const {
    cerr << "Virhe: " << virheteksti << endl;
    cerr << “kirjassa: ";
    tulostaTiedot(cerr);
    cerr << endl;
}
void Kirja::tulostaTiedot(ostream& virta) const {
    virta << tekija_ << " : \"" << nimi_ << "\"";
}
bool Kirja::sopiikoHakusana(string const& sana) const {
    return nimi_.find(sana) != string::npos || tekija_.find(sana) !=
                                               string::npos;
}
```

Tampereen yliopisto
Tampere University

Lisäys luokkaan KirjastonKirja

```
class KirjastonKirja : public Kirja
{
    // ...
    virtual void tulostaTiedot(std::ostream& virta) const;
};

void KirjastonKirja::tulostaTiedot(ostream& virta) const
{
    Kirja::tulostaTiedot(virta);
    virta << ", palautus " << palpvm_;
}
```

Tampereen yliopisto
Tampere University

Dynaaminen sitominen

```
void tulostaKirjat(vector<Kirja*> const& kirjat)
{
    for (unsigned int i = 0; i != kirjat.size(); ++i)
    {
        kirjat[i]->tulostaTiedot(cout);
        cout << endl;
    }
}
```

Tampereen yliopisto
Tampere University

Dynaaminen sitominen

```
int main() {
    vector<Kirja*> kirjaHylly;
    kirjaHylly.push_back( new Kirja("Axiomatic", "Greg Egan"));
    kirjaHylly.push_back( new KirjastonKirja("Matemaattisia olioita",
                                             "Leena Krohn",
                                             Paivays(31,10,1999)));

    tulostaKirjat(kirjaHylly); // Tulostetaan kirjat
    for (unsigned int i = 0; i != kirjaHylly.size(); ++i) {
        delete kirjaHylly[i];
        kirjaHylly[i] = 0;
    }
}
```

Termejä

- Virtuaalifunktio
  - dynaamisesti sidottava funktio
- Dynaaminen (=ajoaikainen) sitominen
  - kutsuttava funktio valitaan olion senhetkisen luokan perusteella
  - mahdollistaa polymorfismin
- Polymorfismi (=monimuotoisuus)
  - oliokielissä: aliluokan olio voi esiintyä kantaluokan olion paikalla

Tampereen yliopisto
Tampere University

Olion tyypin ajoaikainen tarkastaminen

- ISO C++aan lisätty **RTTI** (*Run-Time Type Identification*)
- Vaatii toimiakseen luokkiin ainakin yhden virtuaalifunktion

Olion tyypin ajoaikainen tarkastaminen

- Aliluokan olio kantaluokkaosoittimen päässä:
  - Vain kantaluokan rajapinta näkyy
  - Normaalitapauksessa se(n pitäisi) myös riittää
- Tarve päästä aliluokan rajapintaan → tyyppimuunnos

Olion tyypin ajoaikainen tarkastaminen

- Tyyppimuunnos:
  - Järkevä vain, jos olio todella ko. tyyppiä → voi epäonnistua
  - `dynamic_cast<Aliluokka*>(kantaosoitin)`
  - Jos olio ei oikeaa tyyppiä → palautetaan 0
- Tyyppimuunnoksia syytä välttää, jos mahdollista!

Olion tyypin ajoaikainen tarkastaminen

```
bool myohassako(Kirja* kp, Paivays const& tanaan)
{
    KirjastonKirja* kkp = dynamic_cast<KirjastonKirja*>(kp);
    if (kkp != 0)
    { // Jos tultiin tänne, kirja on kirjastonkirja
        return kkp->onkoMyohassa(tanaan);
    }
    else
    { // Jos tultiin tänne, kirja ei ole kirjastonkirja
        return false; // Ei siis ole myöhässä
    }
}
```

Olion luokan selvittäminen

- **`dynamic_cast`** testaa kuuluuko olio *tiettyyn* luokkaan (tai sen aliluokkaan)

→ Sillä ei saa selville, *mihin* luokkaan olio kuuluu

- Tähän C++:ssa operaattori **`typeid`** ja luokka **`type_info`**
  - Käyttö: `#include <typeinfo>`

Tampereen yliopisto
Tampere University

Olion luokan selvittäminen

- Luokan **`type_info`** oliot
  - “Edustavat” jokainen tiettyä luokkaa
  - Saadaan tuloksena lausekkeista:
    `typeid``(olio)` ja **`typeid`**`(luokka)`
  - Vertailu operaattoreilla == ja `!=`
  - Luokan nimen selvittäminen jäsenfunktiolla **`name`**

Olion luokan selvittäminen

- typeid *testaa eri asiaa* kuin dynamic_cast

```
if( typeid(*kp) ==
typeid(KirjastonKirja))…
if( dynamic_cast<*KirjastonKirja>(kp)
!= 0 ) …
```

Ei-virtuaalifunktio ja peittäminen

- Virtuaalifunktion kutsussa ajoaikainen tarkastus, tavallisessa ei
- Aliluokassa samanniminen jäsenfunktio kuin kantaluokan *ei-virtuaalinen* jäsenfunktio
  - Aliluokan toteutus **peittää** (*hide*) kantaluokassa olevan
  - *Dynaamista sitomista* **ei** käytetä
  - Kutsuttava toteutus riippuu kutsutavasta

Tampereen yliopisto
Tampere University

Ei-virtuaalifunktio ja peittäminen

- Välttääksesi virheitä anna vain *virtuaalifunktioille* uusi toteutus aliluokassa
- Virtuaalisuutta ei voi lisätä enää aliluokassa

→ Kantaluokassa muistettava merkitä **`virtual`**-sanalla *kaikki* sellaiset jäsenfunktiot, joiden toteutus saatetaan määritellä uudelleen aliluokissa

Tampereen yliopisto
Tampere University

Virtuaalipurkajat

- Kantaluokkaosoitin `new`’llä luotuun olioon
- Ongelma: miten tuhota olio, kun sen luokkaa (tyyppiä) ei tunneta?
- Tuhoamistoimet ajoaikana määräytyviä
- Jotta toimisi, *purkajan oltava virtuaalinen kantaluokassa*
- Kantaluokan purkaja ei virtuaalinen → toiminta määrittelemätön

Virtuaalifunktioden hinta

- Ajoaikainen tarkastus → kustannus

1. Tarkastaminen hidastaa ohjelmaa:
   - Kokonaishidastus pientä
   - Ei oleellinen, jos ajoaikainen tarkastus väistämätön

Virtuaalifunktioiden hinta

2. Olion tyyppitieto vie muistia:
    - Yleensä osoittimen verran (4 tavua) oliota kohti
    - Riippumaton virtuaalifunktioiden määrästä
    - Myös pientä luokkakohtaista muistinkulutusta

- Kääntäjällä lupa optimoida kulutus ja hidastus pienemmäksi

Virtuaalifunktiot rakentajissa ja purkajissa

- Rakentajat suoritetaan kantaluokasta aliluokkaan päin
- Kantaluokan rakentajassa aliluokkaosat eivät valmiita

Virtuaalifunktiot rakentajissa ja purkajissa

→ Olio “ei vielä aliluokan olio”

→ Olio käyttäytyy kuin kantaluokan olio

→ Dynaamisessa sitomisessa aliluokan toteutukset eivät käytettävissä

→ ***Vältä virtuaalifunktioiden kutsuja rakentajissa!***

- Sama pätee purkajiin

Abstraktit kantaluokat

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

Abstraktit kantaluokat

- **Puhdas virtuaalifunktio** (*pure virtual function*)
  - Toteutus *pakko* määritellä aliluokissa
  - Kantaluokassa ei usein toteutusta
  - Luokan esittelyyn lisämääre `=0`
- Luokat abstrakteja, kunnes kaikilla puhtailla virtuaalifunktioilla toteutus

Tampereen yliopisto
Tampere University

Puhtaat virtuaalifunktiot

```
class Elain : public Elio {
public:
    virtual ~Elain();
    virtual void liiku(Sijainti paamaara) = 0;
};

class Lintu : public Elain {
public:
    virtual ~Lintu();
    virtual void laula() = 0;
};
```

Puhtaat virtuaalifunktiot

```
class Kana : public Lintu
{
public:
    virtual ~Kana();
    virtual void lisaanny(); // Toteutus lisääntymisfunktiolle
    virtual void liiku(Sijainti paamaara); // Toteutus liikkumiselle
    virtual void laula(); // Toteutus laulamiselle

private:
    // Tänne tarvittavat yksityiset ominaisuudet
};
```

Tampereen yliopisto
Tampere University

Puhdas virtuaalifunktio, jolla toteutus

```
class Elain : public Elio {
public:
    virtual ~Elain();
    virtual void liiku(Sijainti paamaara) = 0;
private:
    Sijainti paikka_;
};
```

Puhdas virtuaalifunktio, jolla toteutus

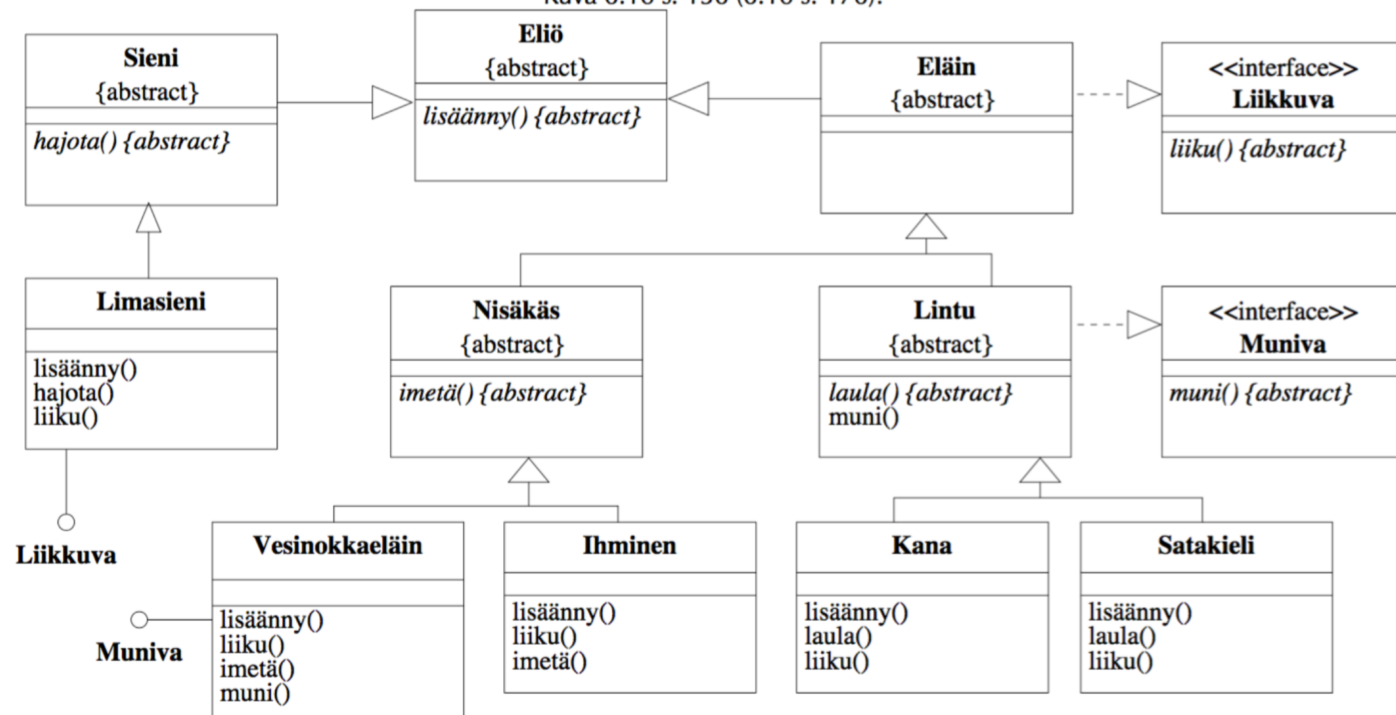
```
void Elain::liiku(Sijainti paamaara)
{
    paikka_ = paamaara;
}

void Kana::liiku(Sijainti paamaara)
{
    // Tähän kanaan liittyvät erityistoimet, käveleminen jne.
    Elain::liiku(paamaara); // Kantaluokka suorittaa kaikille yhteiset
}
```

Periytyminen ja rajapintaluokat

- Kantaluokassa vain rajapinnan määrittely → **rajapintaluokka** (*interface class*)
- Esim. Javassa eivät edes luokkia (oma syntaksi)
- Toisistaan riippumattomat yhteiset rajapinnat ongelma luokkahierarkiassa → *erillisen rajapinnan* käsite

Rajapintaluokat

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



C++: abstraktit kantaluokat ja moniperiytyminen

```
class Liikkuva {
public:
    virtual ~Liikkuva();
    virtual void liiku(Sijainti paamaara) = 0;
};
class Muniva
{
public:
    virtual ~Muniva();
    virtual void muni() = 0;
};
```

C++: abstraktit kantaluokat ja moniperiytyminen

```
class Elain : public Elio,
              public Liikkuva
{
public:
    virtual ~Elain();
private:
};
```

```
class Vesinokkaelain : public Nisakas,
public Muniva
{
public:
    virtual ~Vesinokkaelain();
    virtual void lisaanny();
    virtual void liiku(Sijainti paamaara);
    virtual void imeta();
    virtual void muni();
};
```