Tampereen yliopisto
Tampere University

Muistinhallinta ja olioiden omistus

Fiksut osoittimet

Tampereen yliopisto
Tampere University

Olioiden elinkaari

- Oliot monimutkaisia ⇒ “syntymä ja kuolema” saattavat vaatia kaikenlaista
- Staattinen vs. dynaaminen

Omistusvastuu

- Olio aina jonkin ohjelmanosan tai olion *omistuksessa*
- Dynaamisesti luodun olion tuhoamisvastuu omistajalla

Kuva: Kyle Lucen blogauksesta

Rajapinnat ja olioiden välittäminen

- Moduulit/oliot kutsuvat toisiaan kapseloitujen rajapintojen läpi
- Kutsuissa välitetään usein olioita paikasta toiseen
- Jos olion omistus (= tuhoamisvastuu) säilyy koko ajan yhdessä paikassa, hyvä juttu

Rajapinnat ja olioiden välittäminen

- Usein kuitenkin tarve luoda olio toisella puolen rajapintaa ja tuhota toisella puolella ⇒ olion omistus siirtyy rajapinnan yli
- Omistuksen siirtyminen dokumentoitava rajapintadokumentaatioon!

Omistus

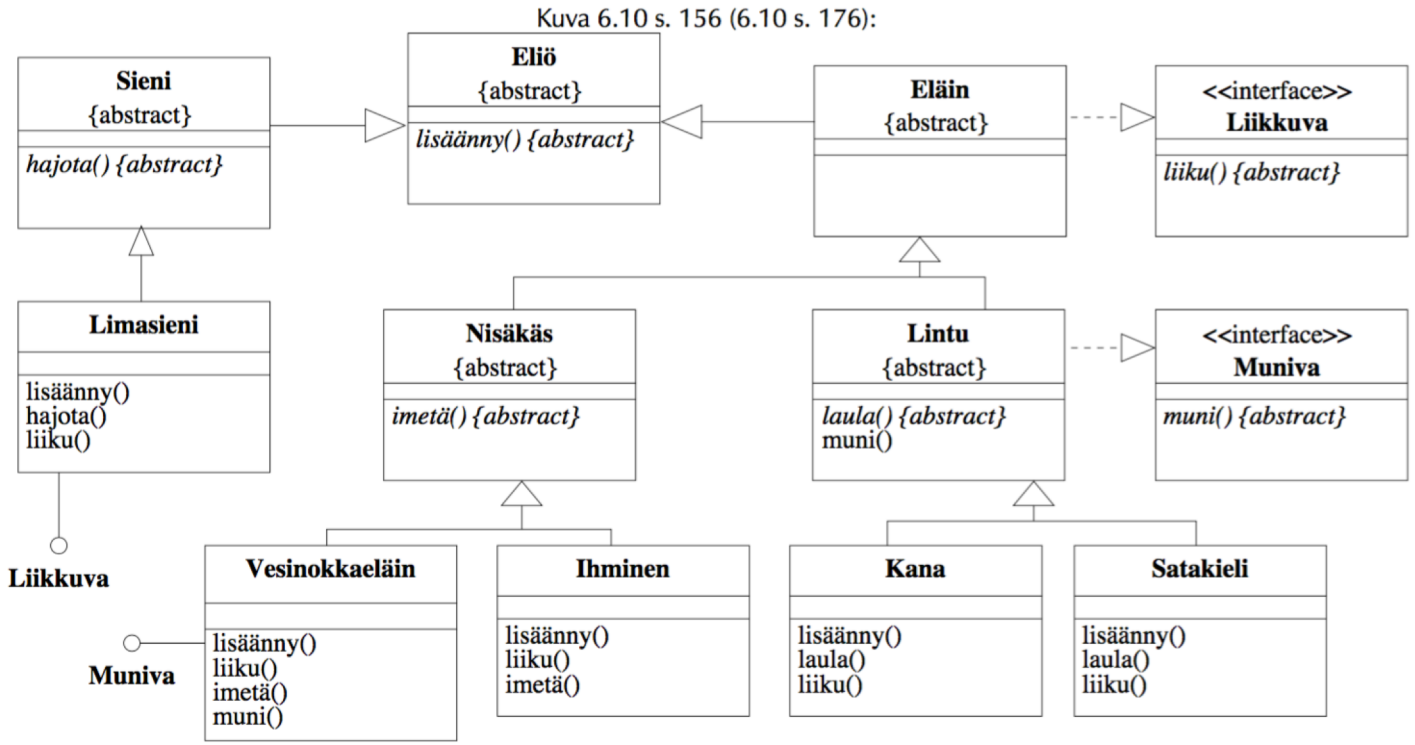
- Omistuksen dokumentointi tärkeää:
  - Olion tuhoamisvastuu
  - Olion vaatimat siivoustoimenpiteet
- Automaattinen roskienkeruu (esim. Python) hoitaa ensimmäisen, mutta ei jälkimmäistä
- C++:ssa ei (vielä) roskienkeruuta ollenkaan, toisaalta purkaja hoitaa siivouksen, kunhan olio tuhotaan

Omistus: tuhoaminen

- Perinteisesti **new**:llä luodun olion tuhoaminen ohjelmoijan vastuulla (**delete**)
- C++11 tarjoaa omistuksen hallintaan myös älykkäät osoittimet:
  - `std::shared_ptr`
  - `std::weak_ptr`
  - `std::unique_ptr`

Rakenteen dokumentointi

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



Omistuksen dokumentointi

- UML:ssä eri assosiaatiotyyppejä (tavallinen, jaettu kooste, muodostuminen...)
- Assosiaatiot vaikuttavat myös olioiden omistussuhteisiin

Omistuksen dokumentointi

- C++:ssa nykyisin useita eri tapoja viitata/osoittaa olioon (viite, osoitin, automaattiosoitin, jaettu osoitin, heikko osoitin)

⇒ C++:n osoitintyypin valinnalla voi dokumentoida koodissa UML:ssä tehtyjä suunnitteluratkaisuja. Lisäksi osoitintyyppi helpottaa ko. assosiaation toteutusta. Kätevää kuin mikä!

(Fiksut ?!) osoittimet

Kuva: DwarfVader (CC BY-NC-ND 2.0)

Omistuksen dokumentointi C++:ssa

Viite (&)

- Ei omistusta, ei voi olla NULL (0)
- Kohde säilyy samana viitteen elinkaaren ajan
- Ei käy STL-säiliöiden alkioksi (Assignable)

Osoitin (∗)

- Ei omistusta (tai omistuksen hallinta tehty käsin)
- Käy STL-säiliöiden alkioksi (Assignable)

Tampereen yliopisto
Tampere University

Omistus ja fiksut osoittimet

Jaettu omistus (shared ownership)

- monta oliota voi omistaa saman resurssin
- resurssi ei riippuvainen yhden olion elinkaaresta

Uniikki omistus (unique ownership)

- yksi olio voi omistaa resurssin

Jaettu osoitin `std::shared_ptr`

- Viitelaskurillinen fiksu osoitin: Jaettu omistus usean jaetun osoittimen kesken
- Resurssit vapautetaan, kun viitelaskuri 0
  - Viimeinen jaettu osoitin poistaa omistettavan
  - Varo syklejä!
  - Luodaan `std::make_shared<X>(...)`

Jaettu osoitin `std::shared_ptr`

Fig: Herb Sutter


```
auto sp1 = make_shared<widget>(…);
auto sp2 = sp1;
```

Muuta:

- Raakaosoitin: `spw.get()`, (ei luopumista)
- Jakotilanteen selvitys: `use_count(), unique()`

Shared_ptr: get():n käyttö

```
void output(const std::string& msg, int* pInt) {
    std::cout << msg << *pInt << std::endl;
}

int main() {
    int* pInt = new int(42);
    std::shared_ptr<int> pShared = make_shared<int>(42);
    output("Raw pointer ", pInt);
    // output("Shared pointer ", pShared); // compiler error
    output("Shared pointer with get() ", pShared.get());
    delete pInt;
    return 0;
}
```

```
Raw pointer 42
Shared pointer with get() 42
```

Heikko osoitin `std::weak_ptr`

- Osoittaa kiinnostuksen jaettuun kohteeseen, ei riitä pitämään hengissä
- Kätevä jaettu osoitin -syklien rikkomiseen
- Kätevä, jos halutaan tietää, onko kohde jo tuhottu

Tampereen yliopisto
Tampere University

Heikko osoitin `std::weak_ptr`

- Viimeinen jaettu poistaa kohteen, vaikka heikkoja jäljellä (wp.`expired()`)
- Ei suoraa pääsyä kohteeseen, mutta tuottaa jaetun osoittimen (wp.lock())

Uniikki osoitin `std::unique_ptr`

- Omistus yksinoikeudella
- Yhtä halpa kuin tavallinen osoitin (ei viitelaskureita)
- Omistuksen voi eksplisiittisesti siirtää tai vapauttaa (alkuperäinen tyhjenee)
- Olion luominen suoraan uniikkiosoittimen päähän (C++14):
  `std::make_unique<X>(...)`

Esimerkki uniikista osoittimesta

```
std::unique_ptr<Thing> p1;
std::unique_ptr<Thing> p2 (std::make_unique<Thing>(…) );
// p1 = p2;                          // Virhe! Uniikkia
osoitinta ei voi kopioida
p1 = std::move(p2);                  // p2.get() == nullptr
Thing* tp = p1.release();            // p1.get() == nullptr
…
p1.reset(new Thing(…));    // p1.get() != nullptr
```

Funktio-osoittimet

- Toiminnallisuuden välittäminen parametrina
  - funktiota ei voi välittää parametrina, mutta osoittimen funktioon voi
  - esim. STL:n algoritmeille (ja assosiatiivisille säiliöille)
- Toinen tapa: funktio-oliot

Funktio-osoittimet

```
bool onkoAlle5(int i) {
    return i < 5;
}

void tulostaAlle5(vector<int> const& v) {
    vector<int>::const_iterator i = v.begin();
    while( (i = find_if(i, v.end(), &onkoAlle5) ) != v.end() ) {
        cout << *i << ' ';
        ++i;
    }
    cout << endl;
}
```