# 12. Monimuotoisuus (osa 2)

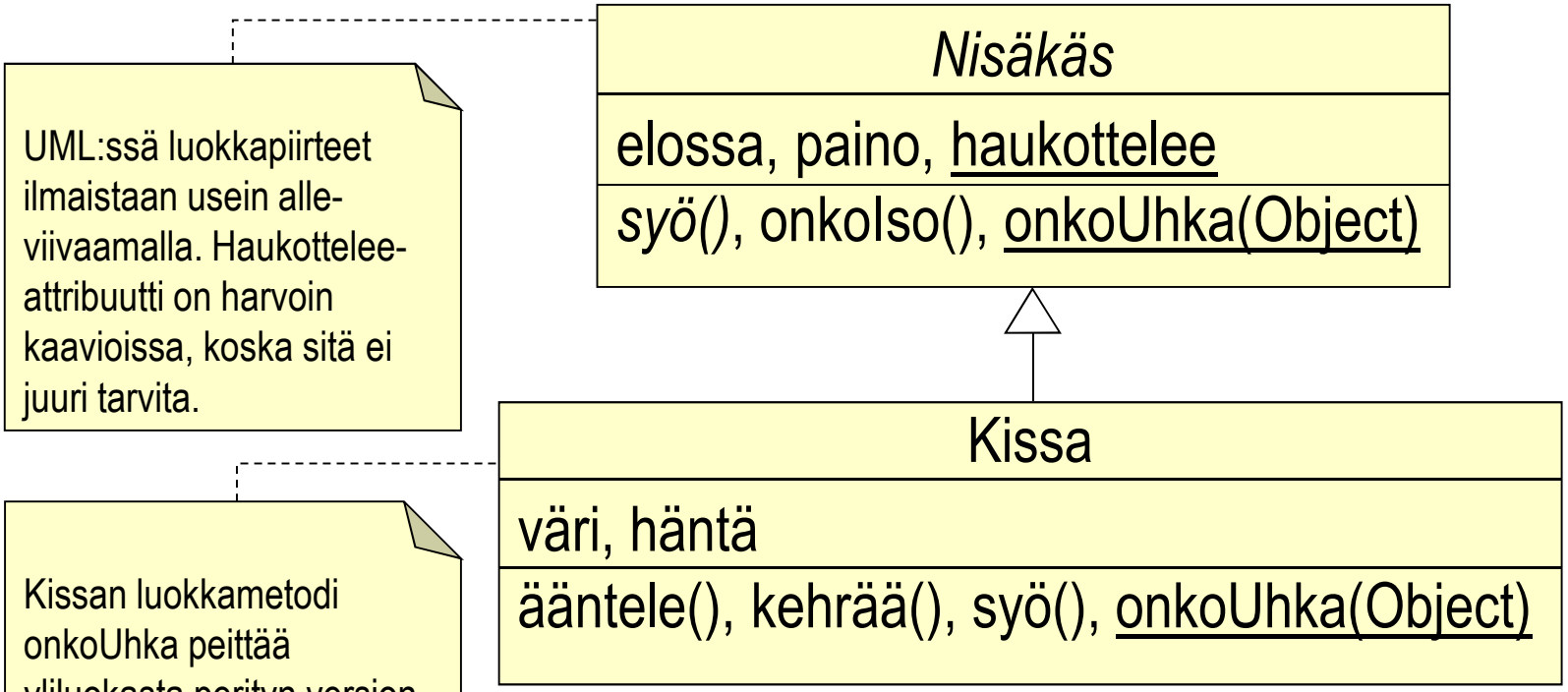
# Sisällys

- Myöhäinen ja aikainen sidonta.
- Parametrinvälitys eräs monimuotoisuuden sovellus.
- Tyyppimuunnos.
- Rajapinnat ja alityypitys.

# Myöhäinen ja aikainen sidonta

- Luokkametodien lisäksi myös **private**- tai **final**-määreellä esitellyt ilmentymämetodit voidaan sitoa käännösaikana, koska tällaisia metodeja ei voida kuormittaa.
  - Kuormittaminen estetään ohjelmoijan toimesta **final**-määreellä.
- Ilmentymämetodien kuormitukset voidaan sitoa aikaisessa vaiheessa, koska ennen ohjelman suoritusta on selvitettävä onko kuormitus mahdollinen.
- Olion viitteen tyyppi on keskeisessä asemassa aikaisessa sidonnassa.
  - Esimerkiksi peitettyjen luokkametodien välillä valitaan viitteeseen perustuen.

# Nisäkäs ja kissa




Kissan piirteitä ovat muun muassa:
elossa, paino, syö(), onkolso(), onkoUhka(Object)
väri, häntä, ääntele(), kehrää()

# Parametrinvälitys

- Monimuotoisuutta hyödynnetään usein parametrinvälityksessä.
- Parametrin tyypiksi asetetaan jokin riittävän lähellä luokkahierarkian juurta oleva luokka, jolloin metodille voidaan antaa parametriksi jälkeläisluokkien viitteitä ja viitteisiin liittyviä olioita.

- // Tämä metodi olisi esimerkiksi
  // monitaitoista lääkäriä
  // mallintavassa luokassa.
  **public void** rokota(Nisakas n) {
  // Täällä rokotetaan mm.
  // kissoja, koiria ja ihmisiä.
  }
- Kissa rontti = **new** Kissa();
  // Toimii, koska Kissa <: Nisakas.
  rokota(rontti);
  …

# Parametrinvälitys

- **Instanceof**-operaattorilla voidaan selvittää parametriin liittyvän olion luokka, jos tiedetään mitä luokkia oliot edustavat.
- x **instanceof** Y
- Palauttaa totuusarvon **true**, jos viite *x* liittyy olioon, jonka luokan tai esivanhemman luokan nimi on *Y*.
  - Testaa onko X <: Y.

```
public void rokota(Nisakas n) {
  if (n instanceof Kissa) {
    // Täällä rokotetaan kissa.
    …
  }
  else if (n instanceof Ihminen) {
    // Täällä rokotetaan ihminen
    …
```

- Operaattorin runsas käyttö ei ole suotavaa, koska yleensä pyritään mahdollisimman erillisiin luokkiin.

# Tyyppimuunnos

- Eksplisiittiset tyyppimuunnokset ()-operaattorilla ovat tuttuja jo alkeistyyppien yhteydestä.
- // 1.25

```
double osamäärä = 5 / (double)4;
// Väliaikainen muunnos, osamäärän palaa arvoon 1.25
// tämän lauseen jälkeen.
System.out.println((int)osamäärä); // 1
System.out.println(osamäärä); // 1.25
// Tallennetaan muunnoksen tulos.
int kokonaisosa = (int)osamäärä; // 1
```

# Tyyppimuunnos

- Myös viitteen luokka voidaan tarvittaessa muuttaa toiseksi tyyppimuunnosoperaatiolla.
- (Y)x tai ((Y)x).piirre
- Operaattori palauttaa *Y*-tyyppisen viitteen viitteeseen *x*-liittyvään olioon, jos *Y* on viitteen luokan esivanhemman tai jälkeläisen nimi.
  - // Luodaan Kissa-luokan olio ja asetetaan siihen Object-viite.
    Object mussu = **new** Kissa();
    // Ei käänny. Staattinen luokka on Object, joka ei kehrää.
    mussu.kehrää();
    // Kääntyy. Staattinen luokka tässä lauseessa Kissa.
    ((Kissa)mussu).kehrää();

# Tyyppimuunnos

- Alkuperäinen luokka palautuu muunnoksen jälkeen, mutta tarvittaessa muunnoksen tulos voidaan sijoittaa toiseen viitteeseen, jonka luokka on sama kuin muunnoksessa käytetty luokka.
  - // Asetetaan olioon viite, jonka luokka on Kissa.

    Kissa mussu2 = (Kissa)mussu;

    // Onnistuu. Staattinen luokka Kissa.

    mussu2.kehrää();
- Yleensä luokkatyyppi muunnetaan siten, että viitteen tyyppi muunnetaan alityypikseen (downcasting), koska moni-muotoisuutta hyödynnetään usein parametrinvälityksessä.

# Tyyppimuunnos

- Tyyppimuunnos on vaarallinen operaatio, koska kääntäjä tarkistaa vain onko tyyppimuunnos kieliopillisesti oikein.
- Tyyppimuunnosten avulla viite voidaan liittää täysin väärän tyyppiseen olioon, jolloin staattisen luokan piirteiden käsittely aiheuttaa ohjelman pysäyttävän poikkeuksen.
- // Staattinen luokka Object, dynaaminen Kissa.

  Object katti = **new** Kissa();

  // Voidaan muuntaa, koska Object > String,

  // mutta ohjelma kaatuu (ClassCastException).

  ((String)katti).length();

# Tyyppimuunnos

- Ajonaikaisia tyyppitarkistuksia voidaan tehdä **instanceof**-operaattorilla ennen tyyppimuunnosta.
- ```
  public void rokota(Nisakas n) {
    if (n instanceof Kissa) {
      // Tyyppimuunnos turvallinen, kun tiedetään
      // dynaamiseksi luokaksi Kissa.
      ((Kissa)n).sähise(); …
    }
    else if (n instanceof Ihminen) {
      // Täällä rokotetaan ihminen
      …
  ```
  ```

# Rajapinnat ja alityypitys

- Rajapinnat noudattavat alityypityssääntöjä.
- Rajapinnan toteuttava luokka ja sen jälkeläiset katsotaan rajapinnan kanssa yhteensopiviksi.
- Oletetaan, että *Kissa*-luokka toteuttaa *Tervehtiva*-rajapinnan.

```
// Staattinen ja dynaaminen luokka Kissa.
Kissa mussu = new Kissa();
// Staattinen luokka Tervehtiva, dynaaminen luokka Kissa.
Tervehtiva terve = mussu;
terve.moikkaa();    // Kääntyy; kutsutaan rajapinnan metodia.
terve.syö();        // Ei käänny; metodi ei viitteen luokassa.
```