9. Periytyminen Javassa (osa 2)

Sisällys

- Piirteiden näkyvyys periytymisessä.
- Ilmentymämetodien korvaaminen.
  - **Super**-attribuutti.
  - *Override*-annotaatio.
- Rakentajat ja periytyminen.
- Periytymisen estäminen.

Piirteiden näkyvyys periytymisessä

- Javan näkyvyysmääreet säätelevät piirteiden näkyvyyttä sekä luokkahierarkian sisällä että pakkausten välillä.
  - **Public**-määreellä esitellyt luokan piirteet näkyvät aina jälkeläisille ja ovat käytettävissä pakkauksesta riippumatta kaikissa muissa luokissa.
  - **Private**-määre kätkee yliluokan piirteet kaikilta muilta luokilta, myös jälkeläisiltä, pakkauksista riippumatta.
  - **Protected**-määre on edellisten välimuoto, jolla on näkyvää vaikutusta näkyvyyteen vasta, kun luokat pakataan.
- Pakkausten välistä näkyvyyttä pohditaan lisää myöhemmin.

Piirteiden näkyvyys periytymisessä

- **Private**-määreellä kätketty tieto saadaan käyttöön luokan jälkeläisissä julkisten aksessoreiden kautta.
- // Metodin korvaus Kissa-
  // luokassa.
  public boolean onkolso() {
  // Kätketty tieto voidaan
  // lukea perityllä metodilla.
  return paino() > 10;
  }
- Jos julkisten aksessoreiden määrittely ei ole järkevää, on attribuutit esiteltävä **protected**-määreellä ja luokat pakattava.
  - Näin kätketty tieto on huonommassa tallessa kuin **private**-määreellä esitelty.
  - **Protected**-määrettä ei saa käyttää siksi, että ei jaksa kirjoittaa aksessoreita!

Korvaaminen

- Aliluokan metodi korvaa (override) yliluokasta perimänsä ilmentymämetodin, kun aliluokan metodi esitellään pääsään-töisesti täsmälleen samalla otsikolla kuin yliluokan metodi.
  - Luokka- tai rajapintatyyppinen tyyppimääre voidaan korvata yhteensopivalla tyypillä (covariant return type).
  - Näkyvyyttä voi laajentaa (**protected** → **public**), mutta ei supistaa.
  - Korvaaminen on eri asia kuin kuormittaminen (overloading), jossa metodin nimi pysyy samana, mutta parametrilista vaihtelee.
  - Luokkametodia ei voi korvata. Korvaamisen sijasta tapahtuu metodin peittäminen (hiding).
- Korvaamismekanismin avulla voidaan toteuttaa yliluokasta peritty toiminnallisuus aliluokalle ominaisella tavalla.

Korvaaminen

- Korvaamisen muoto ja sen tarve on tapauskohtainen.
  - Korvattua versiota voidaan kutsua tarvittaessa.
- Korvataan *Kissa*-luokassa peritty *syö*-metodi siten, että metodissa kuvataan kuinka nimenomaan kissat syövät.
  - Metodin runko on kirjoitettu täysin uudestaan.
- **public void** syö() {

  System.out.println("Syön kuin kissa...");

  kehrää();

  }

Korvaaminen

```
public class Testi {
   public static void main(String[] args) {
      Nisakas nisse = new Nisakas();
      Kissa mussu = new Kissa();
      nisse.syö();                  // Syön kuin nisäkäs...
      mussu.syö();                  // Syön kuin kissa...
   }                                // murr, murrrr...
}
```

Korvaaminen

- Korvattu metodi ei hävitä peitettyä versiota, vaan syrjäyttää sen siten, että peritty metodi on tarvittaessa saatavilla.
- Perittyä ilmentymämetodia voidaan kutsua **super**-attribuutin kautta, joka on aina automaattisesti käytettävissä Javan toimesta kuten **this**-attribuutti.
  - **Super**-attribuutilla voidaan viitata kaikkiin yliluokan näkyviin piirteisiin.
  - **Super**-attribuuttia ei voi käyttää luokkametodissa, koska se on ilmentymäattribuutti.
    - Luokkametodissa voidaan kutsua yliluokan metodia luokan nimen kautta.

Korvaaminen

- Korvataan *Kissa*-luokassa sinne peritty *syö*-metodi siten, että ensin syödään kuten nisäkkäät yleensä ja sitten aletaan kehrätä tyytyväisenä.
  - Aluksi kutsutaan korvattua versiota.
- **public void** syö() {

  super.syö();

  kehrää();

  }

Korvaaminen

```
public class Testi {
   public static void main(String[] args) {
      Nisakas nisse = new Nisakas();
      Kissa mussu = new Kissa();
      nisse.syö();                  // Syön kuin nisäkäs...
      mussu.syö();                  // Syön kuin nisäkäs...
   }                                // murr, murrrr...
}
```

Korvaaminen

- Annotaatiot (annotation) ovat kommenttien tapaan ohjelmaan liittyvää metatietoa eli tietoa tiedosta.
- Annotaatioilla voidaan ohjailla Java-kääntäjän toimintaa.
- Annotaation tunnus alkaa @-merkillä.
  - Esimerkiksi @Override, @Deprecated ja @SuppressWarnings.
- Ohjelmoija lisää annotaatiot itse lähdekoodiin.
- Tällä kurssilla hyödyllisin annotaatio on Java-kielen oma *Override*-annotaatio joka pakottaa kääntäjän ilmoittamaan virheestä, jos annotoitu metodi ei korvaa yliluokan metodia.
  - Estää pienellä vaivalla korvauksen yhteydessä tavallisia kirjoitus- ja ajatusvirheitä.

Korvaaminen

- *Override*-annotaatio annetaan omalle rivilleen välittömästi ennen metodin otsikkoa.
- @Override

  public void syö() {

    super.syö();

    kehrää();

  }
- Annotaatio estää *Kissa*-luokan kääntämisen, jos *syö*-metodin korvaus on epäonnistunut esimerkiksi kirjoitusvirheen vuoksi siten, että korvauksen asemasta määritellään uusi metodi.

Periytyminen ja rakentajat

- Rakentajat eivät periydy yliluokalta aliluokalle.
- Aliluokan oliota luotaessa kutsutaan automaattisesti yliluokan oletusrakentajaa.
  - Automaattista kutsua ei tapahdu, jos aliluokan rakentajassa kutsutaan **super**-metodin avulla yliluokan rakentajaa.
    - Yliluokan rakentajaa ei voi kutsua sen nimellä.
    - Yliluokan rakentajan kutsuminen ohjelmoijan toimesta on pakollista vain, jos yliluokkaan on tehty parametrillinen rakentaja, mutta ei oletusrakentajaa.
- Yliluokan rakentaja alustaa yliluokan attribuutit, jonka jälkeen palataan suorittamaan aliluokan rakentajan sisältö.
- Rakentajan pitäisi alustaa vain oman luokkansa tietoja.

Periytyminen ja rakentajat

- Yliluokan rakentajan kutsun on oltava aliluokan rakentajan ensimmäinen lause.
- Luokan oma rakentajaa voidaan kutsua luokan toisesta rakentajasta **this**-metodilla.
  - Myös tämän kutsun on oltava rakentajan ensimmäinen lause.

- **public** Aliluokka(…) {
  super(…);
  …
  }
- **public** Luokka() {
  this(…);
  …
  }
- **public** Luokka(…) {
  this();
  …
  }

Periytyminen ja rakentajat

- Rakentajat suoritetaan ketjussa.
  - Yliluokka kutsuu oman yliluokkansa rakentajaa joko automaattisesti tai ohjelmoijan toimesta kunnes päädytään *Object*-luokan rakentajaan.
  - Luokkahierarkiassa edetään takaisin suorittamalla esivanhempien rakentajat, joista kukin alustaa oman osuutensa aliluokan perimistä attribuuteista.
  - Suoritetaan aliluokan rakentajan loput lauseet.
- Ketjuttamisen ansiosta
  - luokka voi ottaa aina vastuun vain omien tietojensa alustamisesta ja
  - samaa koodia ei tarvitse kirjoittaa useampaan paikkaan.

Nisäkäs ja kissa (Nisakas.java)

```
public class Nisakas {
   ...
   public Nisakas() {
      elossa = true;
      paino = 0.1;
   }
   public Nisakas(boolean e, double kg) {
      elossa(e);
      paino(kg);
   }
   ...
}
```

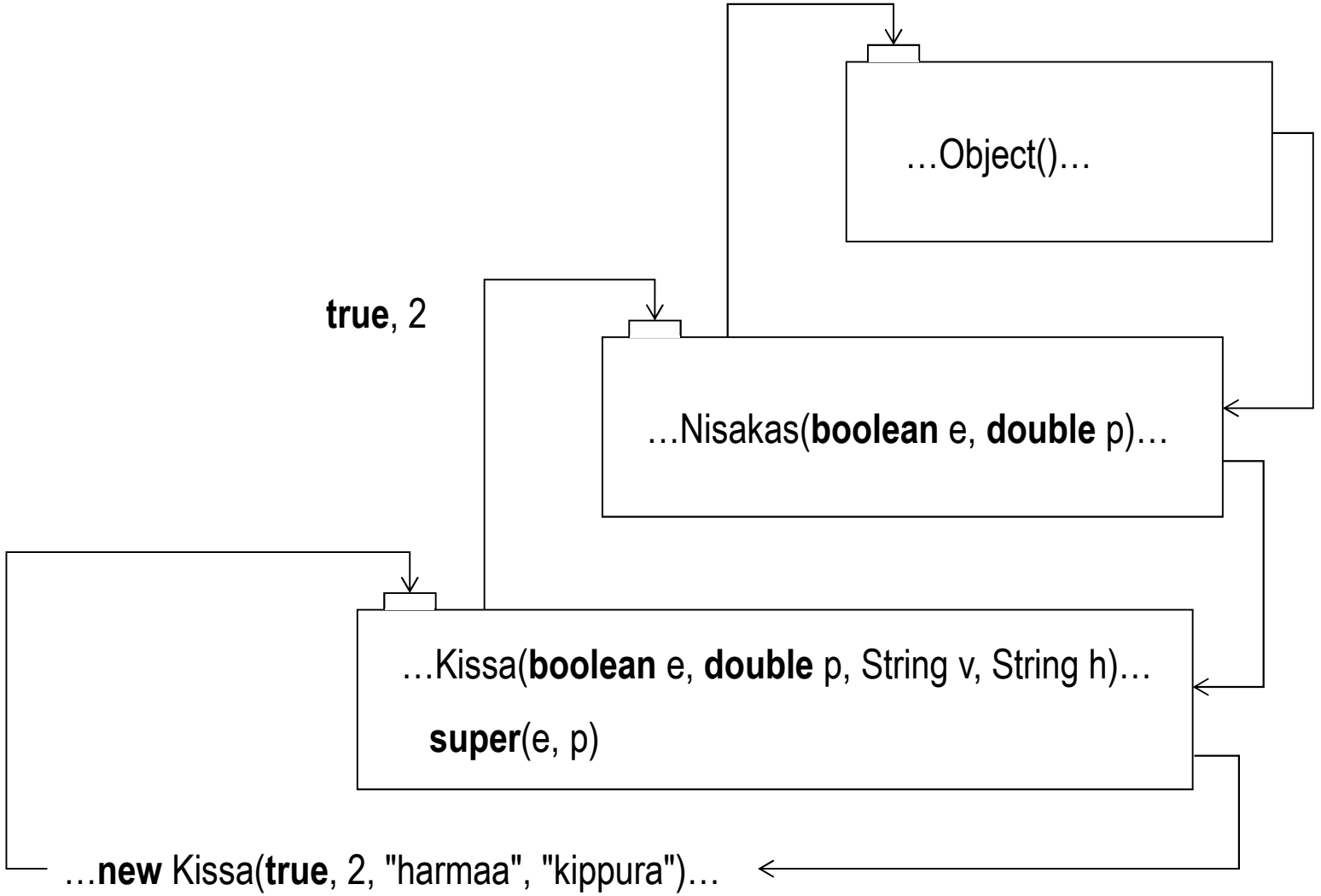
Nisäkäs ja kissa (Kissa.java)

```
public class Kissa extends Nisakas {
   ...
   public Kissa(boolean e, double kg, String v, String h) {
      // Kutsutaan yliluokan rakentajaa,
      // jolloin koodia ei tarvitse monistaa aliluokkaan.
      super(e, kg);
      // Vältetään koodin toistoa aksessoreita.
      väri(v);
      häntä(h);
   }
   ...
}
```

Nisäkäs ja kissa (Testi.java)

```
public class Testi {
   public static void main(String[] args) {
      Kissa hassu = new Kissa(true, 2, "harmaa", "kippura");
      System.out.println(hassu.elossa());      // true
      System.out.println(hassu.paino());       // 2.0
      System.out.println(hassu.väri());        // harmaa
      System.out.println(hassu.häntä());       // kippura
   }
}
```

Nisäkäs ja kissa



Periytymisen estäminen

- Luokan käyttö yliluokkana voidaan estää **final**-määreellä.
- **public final class** ...
- Metodin korvaaminen kielletään varatulla sanalla **final**.