JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Ohjelmointi 2 – Luento 2

Olio-ohjelmoinnin perusteet

Luennon sisältö

- Olio-ohjelmoinnin perusajatus
- Luokka ja olio
- Kapselointi

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Olio-ohjelmoinnin perusajatus

Olio-ohjelmoinnin perusajatus

- Ohjelmat yleensä koostuvat datasta ja sitä käsiteltävästä koodista
- Kuitenkin samaa ohjelmaa voidaan suunnitella eri tavoin
- Esimerkki: Kurssien ilmoittautumisjärjestelmä
  - Opiskelijasta tulee tallentaa nimi ja opiskelijanumero
  - Kurssista tulee tallentaa kurssikoodi ja nimi
  - Ohjelma pitää kirjaa, kuka on ilmoittautunut kursseille

Olio-ohjelmoinnin perusajatus

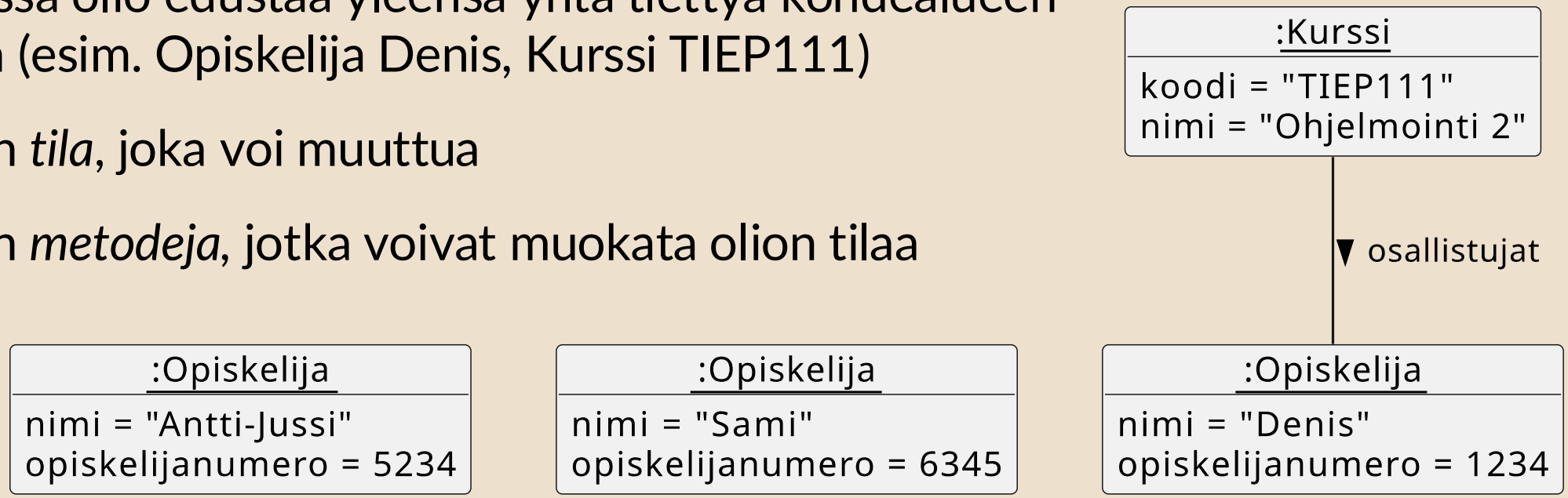
```
void main() {
    // Data eli tietorakenteet
    String[] opiskelijat = {"Denis", "Antti-Jussi", "Sami"};
    int[] opiskelijanumerot = {1234, 5234, 6345};
    String[] kurssit = {"Ohjelmointi 1", "Ohjelmointi 2", "Tietokannat"};
    List<List<Integer>> ilmoittautumiset = new ArrayList<>(List.of(
        new ArrayList<>(List.of(0, 1)),
        new ArrayList<>(List.of(1, 2)),
        new ArrayList<>(List.of(2))
    ));

    ilmoittaudu(2, 0, ilmoittautumiset);
}

// Proseduurit eli tietorakenteita käsittelevät aliohjelmat
void ilmoittaudu(int opiskelija, int kurssi, List<List<Integer>> ilmoittautumiset) {
    ilmoittautumiset.get(opiskelija).add(kurssi);
}
```

Proseduraalinen ohjelmointi: tietorakenteet + aliohjelmat

Olio-ohjelmoinnin perusajatus

Olio-ohjelmointi on *eräs* vaihtoehtoinen tapa suunnitella ja jäsentää ohjelmia

- Olio on rakenne, jonka voi ajatella edustavan yksittäistä asiaa
  - Ohjelmassa olio edustaa yleensä yhtä tiettyä kohdealueen kohdetta (esim. Opiskelija Denis, Kurssi TIEP111)
  - Oliolla on *tila*, joka voi muuttua
  - Oliolla on *metodeja*, jotka voivat muokata olion tilaa



Olio-ohjelmoinnin perusajatus

```
void main() {
    Opiskelija denis = new Opiskelija("Denis", 1234);
    Opiskelija aj = new Opiskelija("Antti-Jussi", 5234);
    Opiskelija sami = new Opiskelija("Sami", 6345);

    Kurssi ohj2 = new Kurssi("TIEP111", "Ohjelmointi 2");
    sami.ilmoittaudu(ohj2);
}
```

Tämän luennon lopputavoite: oliot ja niiden metodit

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Luokka

Luokat

- Java on ns. *luokkapohjainen* olio-ohjelmointikieli
- Ennen olioiden luomista ohjelmoija määrittää olion rakenteen (tila ja metodit) kirjoittamalla *luokka* (eng. class)
- Kirjoitettu luokka toimii ikään kuin "pohjapiirroksena", jonka pohjalta voidaan luoda useita olioita, joilla on sama tila ja samat metodit

Luokat

Uuden luokan määrittely ja olioiden luominen

```
// Luokka määritellään avainsanalla class
class Opiskelija {
}

void main() {
    // Uusi olio luodaan luokan pohjalta käyttäen new-avainsanaa
    Opiskelija denis = new Opiskelija();
    Opiskelija aj = new Opiskelija();
    Opiskelija sami = new Opiskelija();
}
```

Luokat
Attribuutit

```
class Opiskelija {
    // Attribuutti = oliolle kuuluva muuttuja
    String nimi;
    int opiskelijanumero;
}

void main() {
    Opiskelija denis = new Opiskelija();
    denis.nimi = "Denis";
    denis.opiskelijanumero = 1234;

    Opiskelija aj = new Opiskelija();
    aj.nimi = "Antti-Jussi";
    aj.opiskelijanumero = 5234;

    Opiskelija sami = new Opiskelija();
    sami.nimi = "Sami";
    sami.opiskelijanumero = 6345;
}
```

Luokat
Metodit

```
class Opiskelija {
    String nimi;
    int opiskelijanumero;

    // Metodi = oliolle kuuluva aliohjelma
    void tulostaTiedot() {
        IO.println(nimi + ", " + opiskelijanumero);
    }
}

void main() {
    Opiskelija denis = new Opiskelija();
    denis.nimi = "Denis";
    denis.opiskelijanumero = 1234;

    denis.tulostaTiedot();
}
```

Luokat
this-viite

```
class Opiskelija {
    String nimi;
    int opiskelijanumero;

    void tulostaTiedot() {
        String nimi = this.nimi.toUpperCase(); // this = olio, jonka metodia kutsuttiin
        // "nimi" nyt viittaa paikalliseen muuttujaan
        IO.println(nimi + ", " + opiskelijanumero);
    }
}

void main() {
    Opiskelija denis = new Opiskelija();
    denis.nimi = "Denis";
    denis.opiskelijanumero = 1234;

    denis.tulostaTiedot();
}
```

Luokat
Muodostaja

```
class Opiskelija {
    String nimi;
    int opiskelijanumero;

    // Muodostaja => kutsutaan new-operaation yhteydessä
    Opiskelija(String nimi, int opiskelijanumero) {
        this.nimi = nimi;
        this.opiskelijanumero = opiskelijanumero;
    }

    void tulostaTiedot() {
        String nimi = this.nimi.toUpperCase();
        IO.println(nimi + ", " + opiskelijanumero);
    }
}

void main() {
    // Opiskelijan tiedot voi nyt asettaa muodostajan kautta
    Opiskelija denis = new Opiskelija("Denis", 1234);
    denis.tulostaTiedot();
}
```

Luokat
Julkisuusmääreet

```
// public = kaikkien nähtävissä
public class Opiskelija {
    // private = vain tämän luokan sisällä nähtävissä
    private String nimi;
    private int opiskelijanumero;

    // Muodostaja => kutsutaan new-operaation yhteydessä
    public Opiskelija(String nimi, int opiskelijanumero) {
        this.nimi = nimi;
        this.opiskelijanumero = opiskelijanumero;
    }

    // Saantimetodi = metodi, joka palauttaa attribuutin arvon
    public String getNimi() {
        return nimi;
    }

    public void tulostaTiedot() {
        String nimi = this.nimi.toUpperCase();
        IO.println(nimi + ", " + opiskelijanumero);
    }
}
```

Luokat
Luokkametodit ja luokka-attribuutit

```
public class Opiskelija {
    private static final String YLIOPISTO = "Jyväskylän yliopisto";

    // -----8<----- Muut koodit piilotettu ----->8-----

    // Static = luokkametodi, joka ei liity mihinkään tiettyyn olioon
    public static int satunnainenOpiskelijanumero() {
        return new Random().nextInt(9000);
    }
}

void main() {
    Opiskelija denis = new Opiskelija("Denis", Opiskelija.satunnainenOpiskelijanumero());
    denis.tulostaTiedot();
}
```

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Kapselointi

Kapselointi

- Luokkien määrän kasvaessa on helppoa palata ”tieto ja proseduurit” -tyyliseen ohjelmointiin
- Olio-ohjelmointi ei ole pelkästään tapa kirjoittaa koodia, vaan myös tapa suunnitella koodia
- Kapselointi (eng. *encapsulation*) kehottaa suunnitelemaan oliot mahdolisimman itsenäiseksi ja vaihdettavaksi

Kapselointi

Kapseloinnin kaksi päämerkitystä:

- Korkea *koheesio* (eng. cohesion): luokan attribuuttien ja metodien on sovittava mahdollisimman hyvin yhteen.
- Tiedon piilottaminen: olion oma tila on piilossa muilta olioilta. Vastuu olion tilasta kuuluu oliolle itselleen.

Mitä seuraavaksi

- Tee osan 2 harjoitustehtävät