Ohjelmistojen mallintaminen

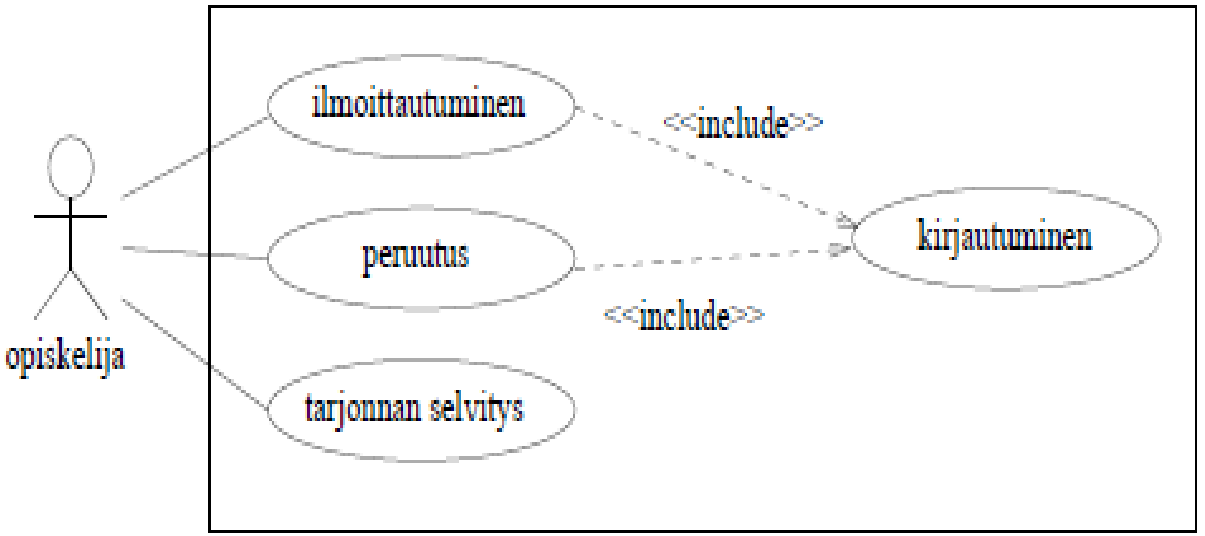
Luento 3, 9.11.

Kertausta: **Ohjelmistotuotantoprosessin vaiheet**

- Vaatimusanalyysi- ja määrittely
  - Mitä halutaan?
- Suunnittelu
  - Miten tehdään?
- Toteutus
  - Ohjelmointi
- Testaus
  - Varmistetaan että toimii niin kuin halutaan
- Ylläpito
  - Korjataan bugit ja tehdään laajennuksia

Viimeksi: **käyttötapaukset**

- Tapa dokumentoida järjestelmän toiminnalliset vaatimukset
  - Käytetään siis *vaatimusmäärittelyssä*
- Käyttötapaus dokumentoi miten *käyttäjä* kommunikoi järjestelmän kanssa suorittaessaan jotain tehtävää
  - Yksi käyttötapaus kuvaa isomman tavoitteellisen kokonaisuuden
  - *Ei kuvaa järjestelmän sisäistä rakennetta tai toimintaa*
- Yleiskuva järjestelmän toiminnallisuudesta *käyttötapauskaaviona*
  - Ellipsit käyttötapauksia, tikkuihmiset käyttäjiä
  - Kuvassa *kirjautuminen* on apukäyttötapaus jonka *ilmoittautuminen* ja *peruutus* sisällyttävät



Käyttötapaus kuvataan tekstinä

- Ohjelmistoprojektissa pitää kuvata kaikki käyttötapaukset tekstuaalisesti saman kaavan mukaan:
  - *käyttötapauspohja*
- Esim. käyttötapaus **Ilmoittautuminen**:

- *Käyttäjä:* opiskelija
- *Tavoite:* saada kurssipaikka
- *Esiehto:* opiskelija on ilmoittautunut kuluvalla lukukaudella läsnäolevaksi
- *Jälkiehto:* opiskelija on lisätty haluamansa ryhmän ilmoittautujien listalle
- *Käyttötapauksen kulku:*

1. Opiskelija aloittaa kurssi-ilmoittautumistoiminnon
2. Järjestelmä näyttää kurssitarjonnan
3. Opiskelija tutkii kurssitarjontaa
4. Opiskeljia valitsee ohjelmiston esittämästä tarjonnasta kurssin ja ryhmän
5. **Suoritetaan käyttötapaus kirjautuminen**
6. Järjestelmä ilmoittaa opiskelijalle ilmoittautumisen onnistumisesta.

Oliot ja luokat

- Kuten viikko sitten mainittiin, nykyään ohjelmistokehitys perustuu usein seuraavaan oletukseen
  - *Minkä tahansa järjestelmän katsotaan voivan muodostua olioista, jotka yhteistyössä toimien ja toistensa palveluja hyödyntäen tuottavat järjestelmän tarjoamat palvelut*

- *Mikä siis on olio?*
  - jokin ohjelman tai sen sovellusalueen kannalta mielenkiintoinen asia tai käsite, tai ohjelman osa
  - yleensä yhdistää tietoa ja toiminnallisuutta
  - omaa identiteetin, eli erottuu muista olioista omaksi yksilökseen

Oliot ja luokat

- Jokainen olio kuuluu johonkin luokkaan
- *Luokka kuvaa minkälaisia siihen kuuluvat oliot ovat tietosisällön ja toiminnallisuuden suhteen*
- Luokka Javassa:

```
class Henkilo {
    String nimi;
    int ika;
    Henkilo(String n) {  nimi = n;  }
    void vanhene() {  ika++;  }
}
```

- ja Henkilö-olioita:

```
Henkilo arto    = new Henkilo(”Arto”);
Henkilo heikki = new Henkilo(”Heikki”);
arto.vanhene();
heikki.vanhene();
```

Suunnittelu ja toteutusvaiheen oliot ja luokat

- Oliota ja luokkia ajatellaan usein ohjelmointitason käsitteinä
  - Esim. Javassa määritellään luokka class Henkilo { … }
    - *Huom: kalvojen esimerkeissä jätetään usein näkyvyysmääreet public/private pois jotta koodi olisi tiiviimpää. Näin ei tule tehdä oikeasti ohjelmoidessa*
    - *Huom2: Jos näkyvyyttä ei ole merkitty, on käytössä ns. pakkaustason näkyvyys*
  - Ja luodaan olioita Henkilo arto = new Henkilo(”Arto”);
  - Ohjelma siis muodostuu olioista
  - Oliot elävät koneen muistissa
  - Ohjelman toiminnallisuus muodostuu olioiden toiminnallisuudesta

- Ohjelmiston suunnitteluvaiheessa suunnitellaan mistä oliosta ohjelma koostuu ja miten oliot kommunikoivat
  - Nämä oliot sitten toteutetaan ohjelmointikielellä toteutusvaiheessa

- *Mistä suunnitteluvaiheen oliot tulevat? Miten ne keksitään?*

Vaatimusanalyysivaiheen oliot ja luokat

- Vaatimusmäärittelyn yhteydessä tehdään usein *vaatimusanalyysi*
  - Kartoitetaan ohjelmiston sovellusalueen (eli sovelluksen kohdealueen) kannalta tärkeitä *käsitteitä* ja *niiden suhteita*
- Eli mietitään mitä tärkeitä asioita sovellusalueella on olemassa
  - Esim. kurssihallintojärjestelmän käsitteitä ovat
    - Kurssi
    - Laskariryhmä
    - Ilmoittautuminen
    - Opettaja
    - Opiskelija
    - Sali
    - Salivaraus
- Nämä käsitteet voidaan ajatella luokkina

Vaatimusanalyysivaiheen oliot ja luokat

- Vaatimusanalyysivaiheen luokat ovat *vastineita reaalimaailman käsitteille*
- Kun edetään vaatimuksista ohjelmiston suunnitteluun, monet vaatimusanalyysivaiheen luokista saavat vastineensa ”ohjelmointitason” luokkina, eli luokkina, jotka on tarkoitus ohjelmoida esim. Javalla
- Eli riippuen katsantokulmasta luokka voi olla joko
  - reaalimaailman käsitteen vastine, tai
  - suunnittelu- ja ohjelmointitason ”tekninen” asia
- Tyypillisesti ohjelmatason olio on vastine jollekin todellisuudessa olevalle ”oliolle”
  - Ohjelma simuloi todellisuutta
- Ohjelmissa on myös paljon luokkia ja olioita, joille ei ole vastinetta todellisuudessa
  - Esim. käyttöliittymän toteuttavat oliot

Oliomallinnus

- **Olioperustainen ohjelmistokehitys** etenee yleensä seuraavasti:
  1. Luodaan **määrittelyvaiheen oliomalli** sovelluksen käsitteistöstä
     - Mallin oliot ja luokat ovat rakennettavan sovellusksen kohdealueen käsitteiden vastineita
  2. Suunnitteluvaiheessa tarkennetaan edellisen vaiheen oliomalli **suunnitteluvaiheen oliomalliksi**
     - Oliot muuttuvat yleiskäsitteistä teknisen tason olioiksi
     - Mukaan tulee olioita, joilla ei suoraa vastinetta reaalimaailman käsitteistössä
  3. Toteutetaan suunnitteluvaiheen oliomalli jollakin **olio-ohjelmointikielellä**

- Voidaankin ajatella, että *malli tarkentuu* muuttuen koko ajan ohjelmointikieliläheisemmäksi/teknisemmäksi siirryttäessä määrittelystä suunnitteluun ja toteutukseen

Luokka- ja oliokaaviot

- Palaamme oliomallinnukseen ja olioiden käyttöön määrittely- ja suunnittelutasolla myöhemmin
- Nyt tarkastelemme miten olioita ja luokkia kuvataan UML:ssä
- Sama mallinnustekniikka kelpaa sekä teknisen tason että korkeamman tason olioille ja luokille

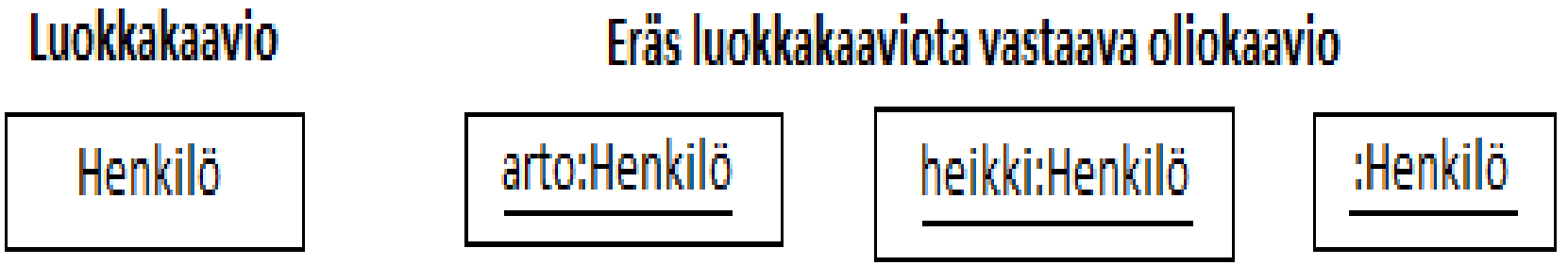
Luokka- ja oliokaaviot

- Järjestelmän luokkarakennetta kuvaa **luokkakaavio** (engl. class diagram)
  - Mitä luokkia olemassa
  - Minkälaisia luokat ovat
  - Luokkien suhteet toisiinsa
- Luokkakaavio on UML:n eniten käytetty kaaviotyyppi

- Luokkakaavio kuvaa ikäänkuin kaikkia mahdollisia olioita, joita järjestelmässä on mahdollista olla olemassa
- **Oliokaavio** (engl. object diagram) taas kuvaa mitä olioita järjestelmässä on tietyllä hetkellä

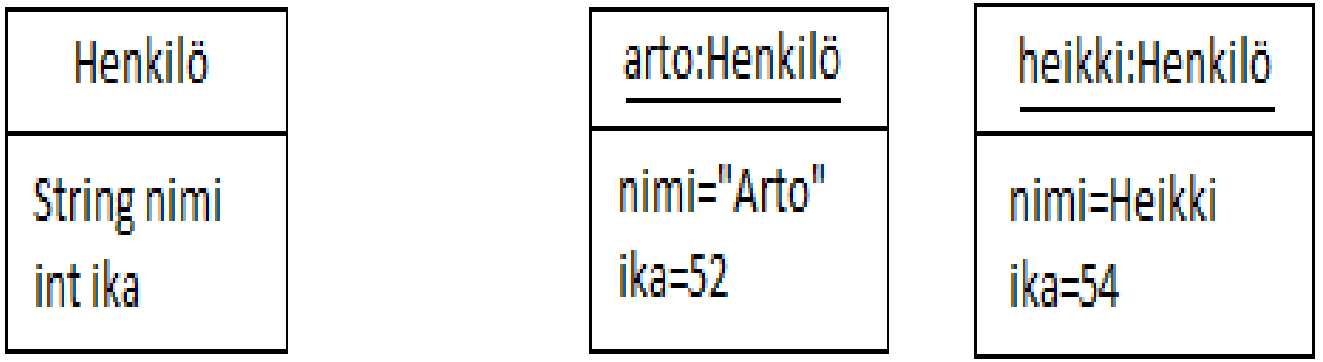
Luokka- ja oliokaaviot

- Luokkaa kuvataan laatikolla, jonka sisällä on luokan nimi
  - Kuten pian näemme, laatikossa voi olla myös muuta
- Luokan instansseja eli siitä luotuja olioita kuvataan myös laatikolla, erona on nimen merkintätapa
  - Nimi alleviivattuna, sisältäen mahdollisesti myös olion nimen
- Kuvassa Henkilö-luokka ja kolme Henkilö-olioa
  - Kolmas olioista on nimetön
- Luokkia ja olioita ei sotketa samaan kuvaan, kyseessä onkin kaksi kuvaa: vasemmalla luokkakaavio ja oikealla oliokaavio
  - Oliokaavio kuvaa tietyn hetken tilanteen, olemassa 3 henkilöä



Luokka- ja oliokaaviot: attribuutit

- Luokan olioilla on attribuutteja eli muuttujia ja operaatiota eli metodeja
- Nämä määritellään luokkamäärittelyn yhteydessä
  - Aivan kuten Javassa kirjoitettaessa class Henkilö{ … } määritellään kaikkien Henkilö:n attribuutit ja metodit luokan määrittelyn yhteydessä
- Luokkakaaviossa attribuutit määritellään luokan nimen alla omassa osassaan laatikkoa
  - Attribuutista on ilmaistu nimi ja tyyppi (voi myös puuttua)
- Oliokaaviossa voidaan ilmaista myös attribuutin arvo

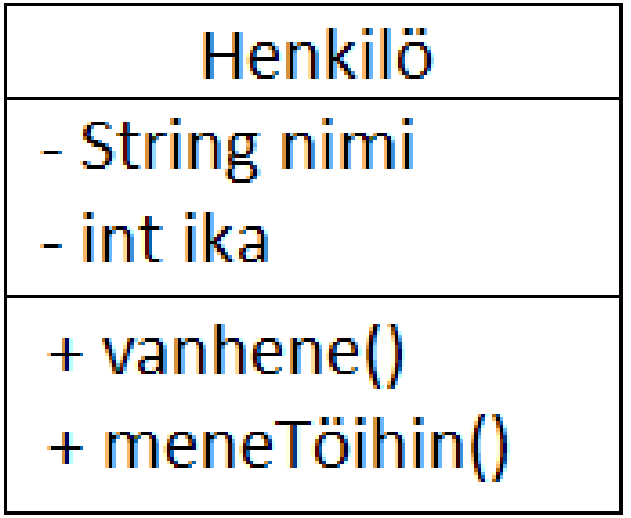


Luokka- ja oliokaaviot: metodit

- Luokan olioiden metodit merkitään laatikon kolmanteen osaan
- Luokkiin on pakko merkitä ainoastaan nimi
  - Attribuutit ja metodit merkitään jos tarvetta
  - Usein metodeista merkitään ainoastaan nimi, joskus myös parametrien ja paluuarvon tyyppi
- Attribuuttien ja operaatioiden parametrien ja paluuarvon tyyppeinä voidaan käyttää valmiita tietotyyppejä (int, double, String, …) tai rakenteisia tietotyyppejä (esim. taulukko, ArrayList).
- Tyyppi voi olla myös luokka, joko itse määritelty tai asiayhteydestä ”itsestäänselvä” (alla Väri ja Piste)

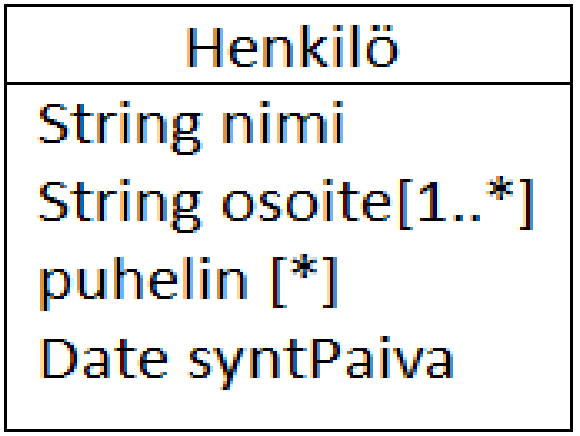
Luokkakaavio: attribuuttien ja operaatioiden näkyvyys

- Ohjelmointikielissä voidaan attribuuttien ja metodien näkyvyyttä muiden luokkien olioille säädellä
  - Javassa private, public, protected
- UML:ssa näkyvyys merkitään attribuutin tai metodin eteen: *public +, private -, protected #, package ~*
  - Jos näkyvyyttä ei ole merkitty, sitä ei ole määritelty
  - Kovin usein näkyvyyttä ei viitsitä merkitä
- Esim. alla kaikki attribuutit ovat private eli eivät näy muiden luokkien olioille, metodit taas public eli kaikille julkisia



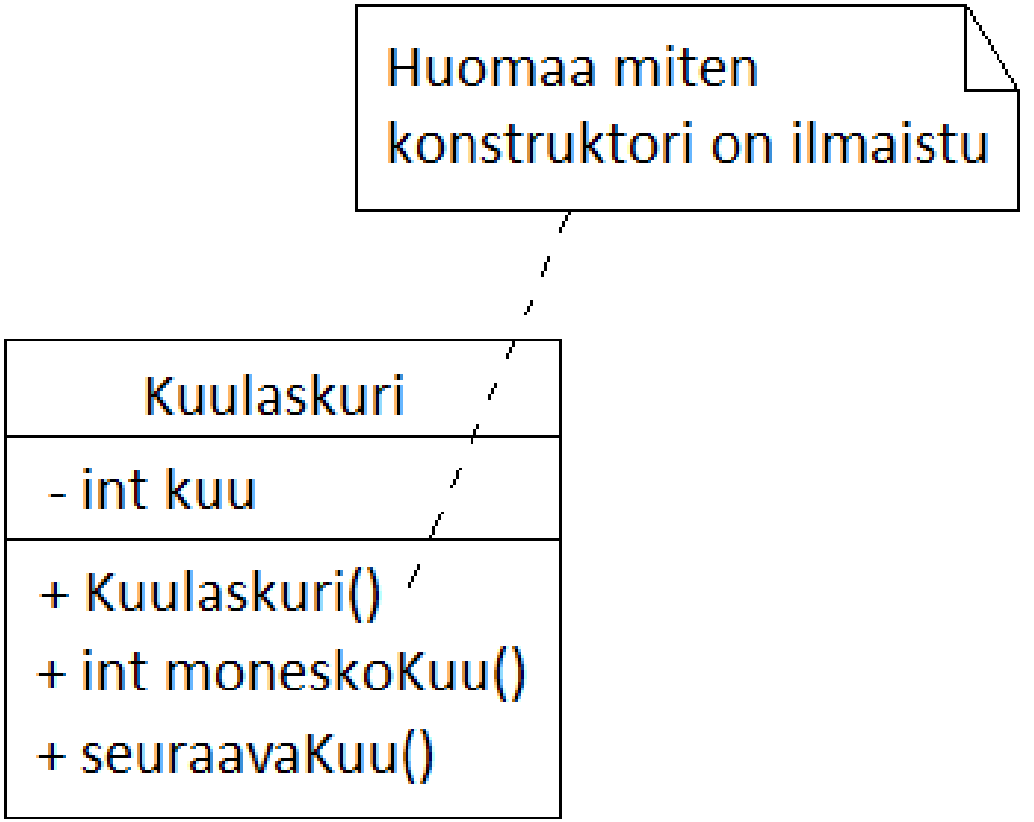
Luokkakaavio: attribuutin moniarvoisuus

- Jos attribuutti on kokoelma samanlaisia arvoja (esim. taulukko), voidaan se merkitä luokkakaavioon
  - Kirjoitetaan attribuutin perään hakasulkeissa kuinka monesta ”asiasta” attribuutti toistuu
  - * tarkoittaa tuntematonta
- Henkilöllä on vähintään 1 osoite, mutta osoitteita voi olla useita
- Puhelinnumeron kohdalle on kirjoitettu [*], joka tarkoittaa, että numeroa ei välttämättä ole tai numeroita voi olla useita
- Moniarvoisuus on asia joka merkitään kaavioon melko harvoin



Esim: Kurssilta ohjelmoinnin perusteet tuttu Kuulaskuri

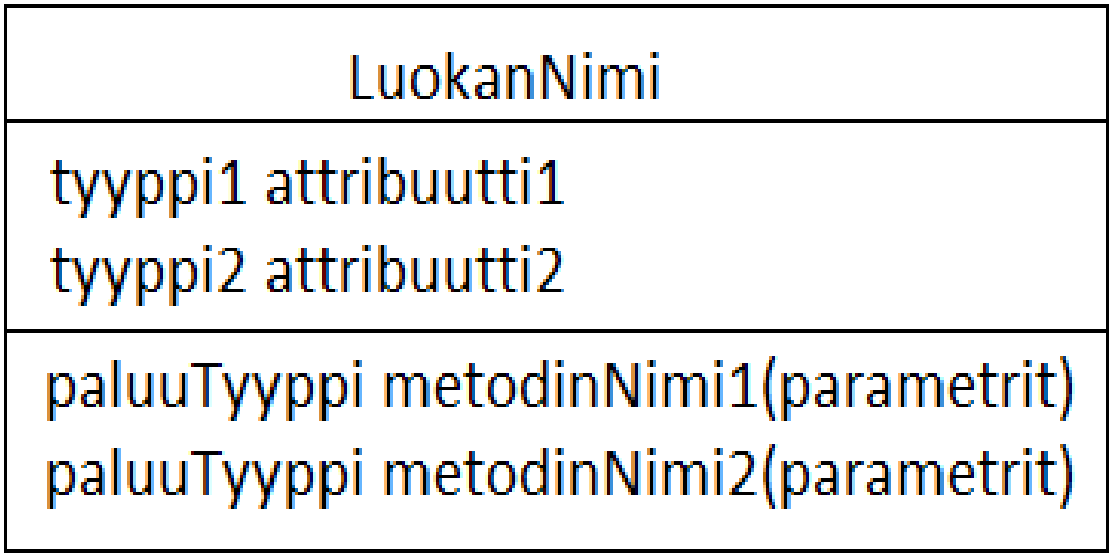
```
public class Kuulaskuri {
    private int kuu;

    public Kuulaskuri() {
        kuu = 1;
    }

    public int moneskoKuu() {
       return kuu;
    }

    public void seuraavaKuu() {
        kuu++;
        if (kuu == 13)  kuu = 1;
    }
}
```



Kuvassa mukana UML-kommenttisymboli

Kertaus: Luokan määrittely luokkakaaviossa

- Eli luokka on laatikko, jossa luokan nimi ja tarvittaessa attribuutit sekä metodit
- Attribuuttien ja metodien parametrien ja paluuarvon tyyppi ilmaistaan tarvittaessa
  - Näkyvyysmääreet ilmaistaan tarvittaessa
- Jos esim. metodeja ei haluta näyttää, jätetään metodiosa pois, vastaavasti voidaan menetellä attribuuttien suhteen



Olioiden väliset yhteydet

- Ohjelmat sisältävät useita olioita ja olioiden välillä on yhteyksiä:
  - Työntekijä *työskentelee* Yrityksessä
  - Henkilö *omistaa* Auton
  - Henkilö *ajaa* Autolla
  - Auto *sisältää* Renkaat
  - Henkilö *asuu* Osoitteessa
  - Henkilö *omistaa* Osakkeita
  - Työntekijä *on* Johtajan *alainen*
  - Johtaja *johtaa* Työntekijöitä
  - Johtaja *erottaa* Työntekijän
  - Opiskelija *on ilmoittautunut* Kurssille
  - Viljavarasto *sisältää* kaksi Varastoa
  - Lukijalaite *tuntee* Päivämäärän
- Yhteys voi olla pysyvämpiluontoinen eli rakenteinen tai hetkellinen
  - Tällä luennolla fokus pysyvämpiluontoisissa yhteyksissä

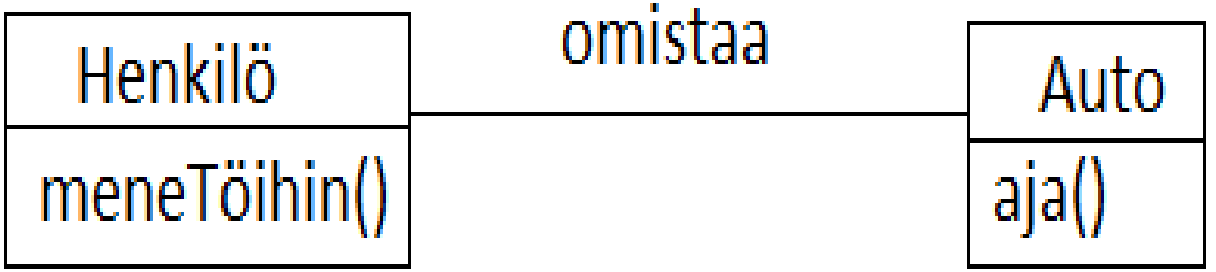
Olioiden väliset yhteydet

- Ohjelmakoodissa pysyvä yhteys ilmenee yleensä luokassa olevana *olioviitteenä*, eli attribuuttina jonka tyyppinä on luokka
- Henkilö omistaa Auton:

```
class Auto{
   void aja(){ System.out.println("liikkuu"); }
}

class Henkilö {
   Auto omaAuto;   // viite olioon, jonka tyyppinä Auto

   Henkilö(Auto a) {  omaAuto = a;  }

   void meneTöihin() {
      omaAuto.aja();      // metodissa henkilö käyttää omistamaansa autoa
   }
}
```

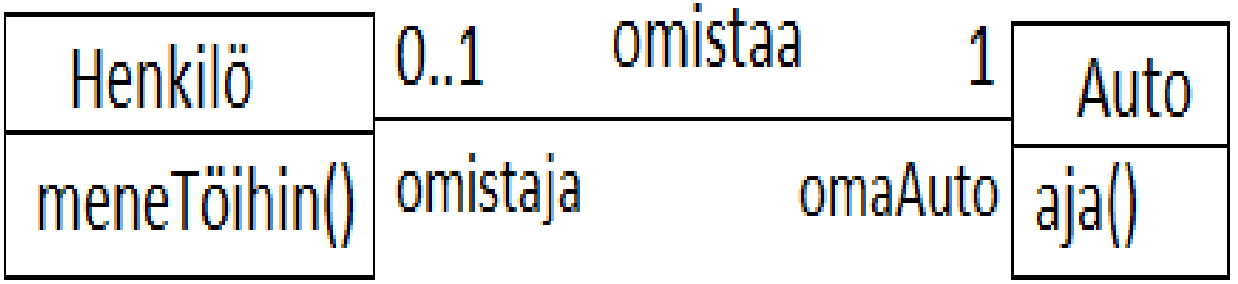
Olioiden väliset yhteydet

- Olioviite voitaisiin periaatteessa merkitä luokkakaavioon attribuuttina, kyseessähän on teknisessä mielessä attribuutti
  - Näin ei kuitenkaan ole tapana tehdä
- Parempi tapa on kuvata olioiden välinen yhteys luokkakaaviossa
  - Jos Henkilö- ja Auto-olion välillä voi olla yhteys, yhdistetään Henkilö- ja Auto-luokat viivalla
- Tilanne kuvattu alla
  - Yhteydelle on annettu nimi *omistaa*
  - Eli Henkilö-olio omistaa Auto-olion



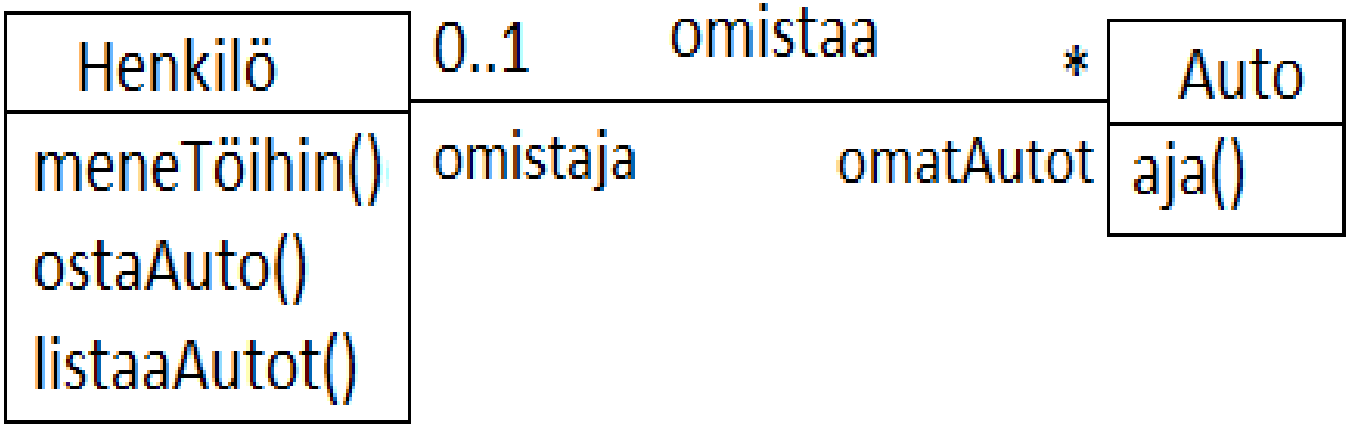
Yhteys: kytkentärajoitteet ja roolit

- Ohjelmakoodissa jokaisella henkilöllä on täsmälleen yksi auto
  - ja auto liittyy korkeintaan yhteen henkilöön
- Nämä voidaan kuvata luokkakaaviossa *kytkentärajoitteina*
  - Alla yhteyden oikeassa päässä on numero 1, joka tarkoittaa, että yhteen Henkilö-olioon liittyy täsmälleen yksi Auto-olio
  - Yhteyden vasemmassa päässä 0..1, joka tarkoittaa, että yhteen Auto-olioon liittyy 0 tai 1 Henkilö-olioa
- Auton *rooli* yhteydessä on olla henkilön omaAuto, rooli on merkitty Auton viereen
  - Huom: roolin nimi on sama kun luokan Henkilö attribuuti jonka tyyppinä Auto
- Henkilön rooli yhteydessä on olla omistaja



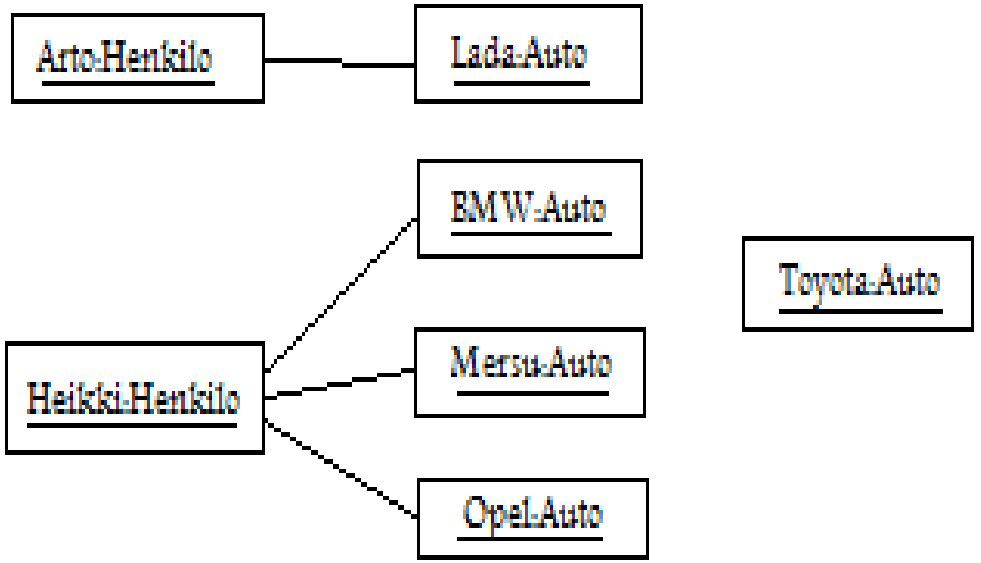
Yhteys

- Esimerkissämme yhdellä Henkilö-oliolla on yhteys täsmälleen yhteen Auto-olioon
  - Eli yhteyden Auto-päässä on kytkentärajoitteena 1
- Jos halutaan mallintaa tilanne, jossa kullakin Henkilö-oliolla voi olla mielivaltainen määrä autoja (nolla tai useampi), niin kytkentärajoitteeksi merkitään *
  - Alla tilanne, jossa henkilö voi omistaa useita autoja
  - Kullakin autolla joko 0 tai 1 omistajaa



Yhteydet oliokaaviossa

- Luokkakaavio kuvaa luokkien olioiden kaikkia mahdollisia suhteita
  - Edellisessä sivulla sanotaan vaan, että tietyllä henkilöllä voi olla useita autoja ja tietyllä autolla on ehkä omistaja
- Jos halutaan ilmaista asioiden tila jollain ajanhetkellä, käytetään oliokaaviota
  - Mitä olioita tietyllä hetkellä on olemassa
  - Miten oliot yhdistyvät
- Alla tilanne, jossa Artolla on 1 auto ja Heikillä 3 autoa, yhdellä autolla ei ole omistajaa



Yhden suhde moneen -yhteyden toteuttaminen Javassa

- Jos henkilöllö on korkeintaan 1 auto, on Henkilö-luokalla siis attribuutti, jonka tyyppi on Auto

```
class Henkilö {
    Auto omaAuto;   // viite olioon, jonka tyyppinä Auto
    // ,,,
}
```

- Jos henkilöllä on monta autoa, on Javassa yksi ratkaisu lisätä Henkilö-luokalle attribuutiksi listallinen (esim. ArrayList) autoja:

```
class Henkilö {
    ArrayList<Auto> omatAutot;
    // ...
}
```

- Esimerkki seuraavalla sivulla, huomaa miten listalle lisätään uusi auto ja miten lista käydään läpi
- ArrayLististä tarkemmin Ohjelmointikurssin materiaalista

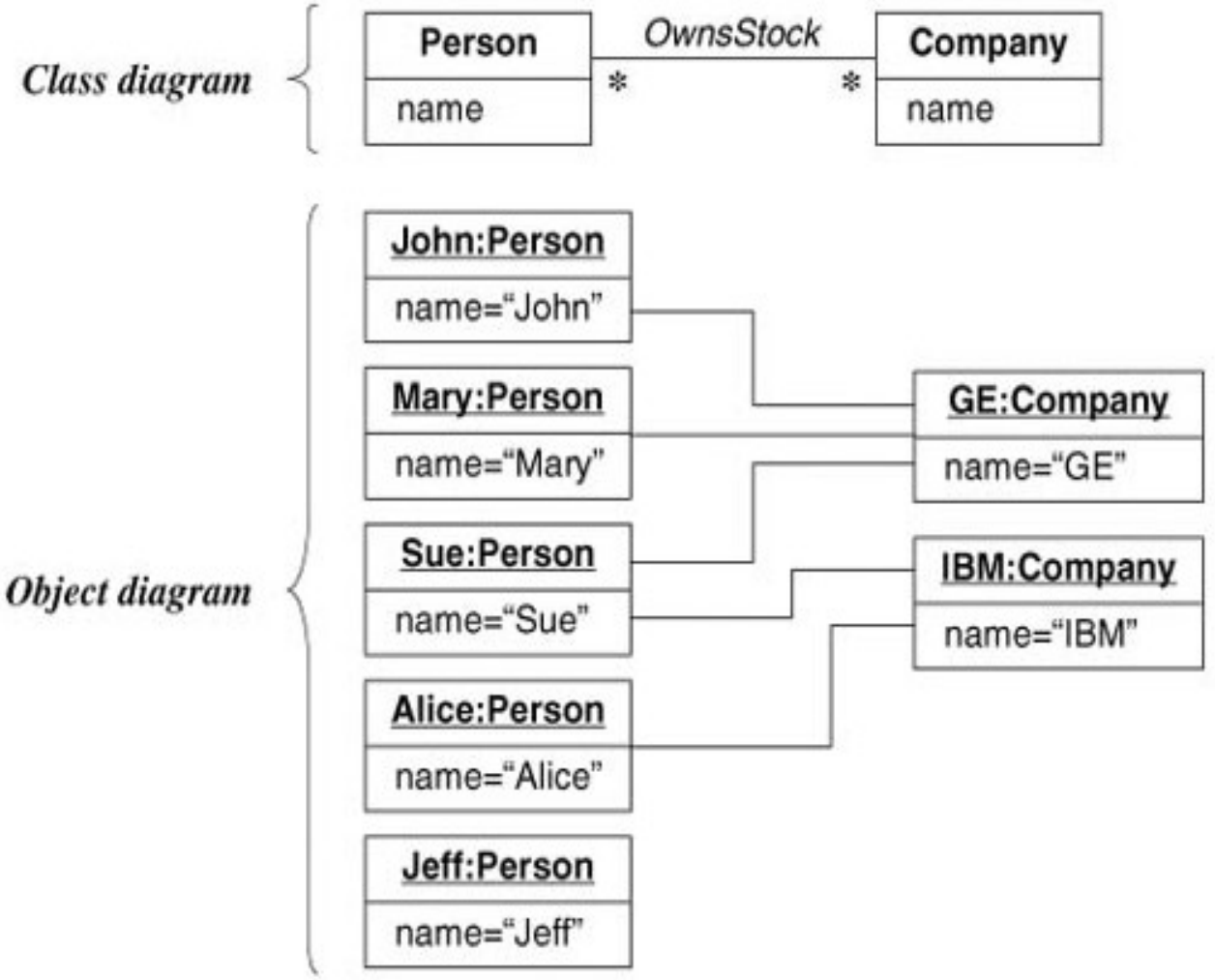
```
class Henkilö {
    ArrayList<Auto> omatAutot;

    Henkilö () { omatAutot = new ArrayList<Auto>(); }

    void ostaAuto(Auto uusi){   omatAutot.add(uusi);  }

    void listaaAutot() {
        for ( Auto auto : omatAutot )
            System.out.prinln(auto);
    }

    void meneToihin(){
        if ( omatAutot.isEmpty()  )
            //  käytä julkista liikennettä
        else
            // … valitaan auto jolla mennään töihin
    }
```

Yhteyksistä

- Kuten niin moni asia UML:ssä, on myös yhteyden nimen ja roolinimien merkintä vapaaehtoista
  - Joskus asia on niin ilmeinen, että nimeämistä ei tarvita
- Jos kytkentärajoite jätetään merkitsemättä, niin silloin yhteydessä olevien olioiden lukumäärä on määrittelemätön
  - Kytkentärajoitteet ilmaistaankin melkein aina
  - Poikkeuksena tilanne, jossa luodaan luokkakaavio askeleittain tarkentaen karkeammasta tarkempaan

- Seuraavassa joukko esimerkkejä
- Monissa tapauksissa esitetty myös oliokaavio selkiyttämään tilannetta

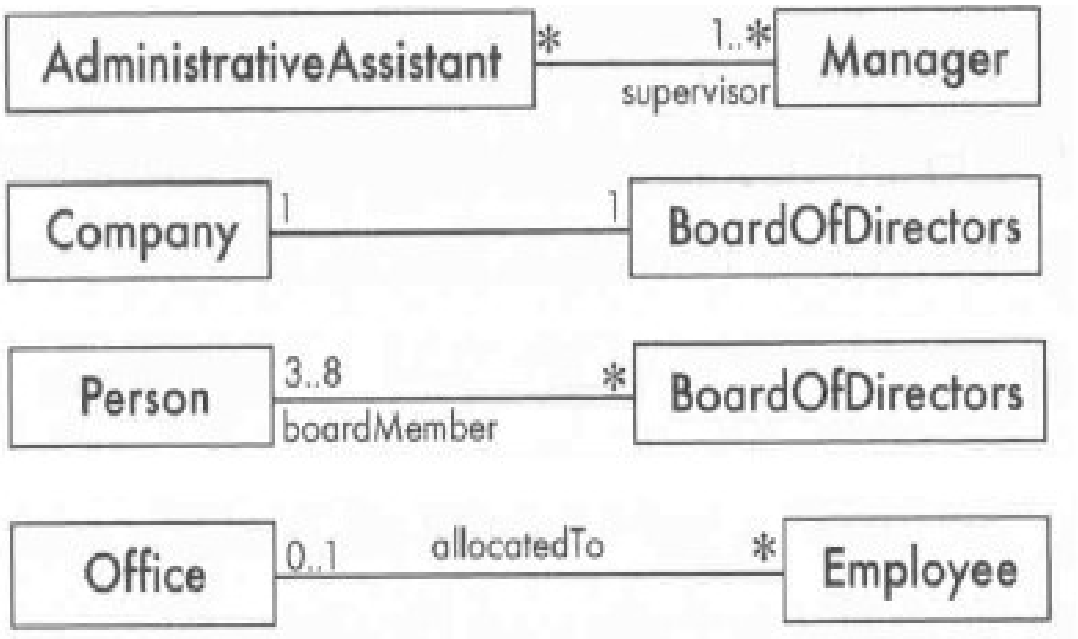
- Henkilö voi omistaa usean yhtiön osakkeita
- Yhtiöllä on monia osakkeenomistajia
- Eli yhteen Henkilö-olioon voi liittyä monta Yhtiö-olioa
- Ja yhteen Yhtiö-olioon voi liittyä monta Henkilö-olioa



- Yhtiössä työskentelee useita ihmisiä
- Ihminen työskentelee korkeintaan yhdessä yrityksessä
- Huomaa roolinimet:
  - Ihmisen rooli yhteyden suhteen on työntekijä, *employee*
  - Yrityksen rooli yhteydessä on työnantaja, *emplyer*

employee	employer
Joe Doe	Simplex
Mary Brown	Simplex
Jean Smith	United Widgets

- Lisää bisnesmaailman esimerkkejä
  - Manageria kohti on useita assistentteja, assistentin johtajana (supervisor) toimii vähintään yksi manageri
  - Yhtiöllä on yksi johtokunta, joka johtaa tasan yhtä yhtiötä
  - Johtokuntaan kuuluu kolmesta kahdeksaan henkeä. Yksi henkilö voi kuulua useisiin johtokuntiin, muttei välttämättä yhteenkään.
  - Toimistoon on sijoitettu (allocated to) useita työntekijöitä. Työntekijällä on paikka yhdessä toimisto tai ei missään



Esimerkki

- Vaaka koostuu kahdesta varastosta
- ks. Ohjelmoinnin perusteiden materiaali
  - http://www.cs.helsinki.fi/u/wikla/ohjelmointi/materiaali/III_oliot/#15
  - http://www.cs.helsinki.fi/u/wikla/ohjelmointi/materiaali/III_oliot/#22

- Luokkakaavio tehdään luennolla

- Perjantaina jatkamme tutustumista luokkakaavioihin