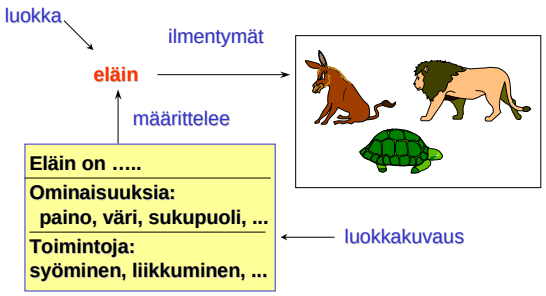
Luokka- ja oliokaaviot

- Luokka- ja oliokaavioilla kuvataan ohjelmiston / järjestelmän koostuminen olioista, olioiden tietämys, niiden tarjoamat palvelut sekä olioiden väliset yhteydet
- Olio
  - tiedon ja sen käsittelyyn liittyvien palveluiden muodostama kokonaisuus
  - olio pitää sisällään tietoja, joita olion palvelut käsittelevät / hyödyntävät
  - Puhtaassa olio-ohjelmoinnissa olion tietoihin pääse käsiksi vain olion palveluiden kautta

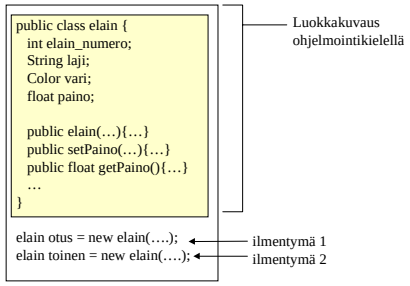
Luokka- ja oliokaaviot

- (Olio)**luokka** (class, object class) on samankaltaisten olioiden malli.
- Saman mallin mukaiset oliot kuuluvat samaan luokkaan. Ne ovat kyseisen luokan ilmentymiä (instance).
- Luokkakuvaus määrittelee luokan.

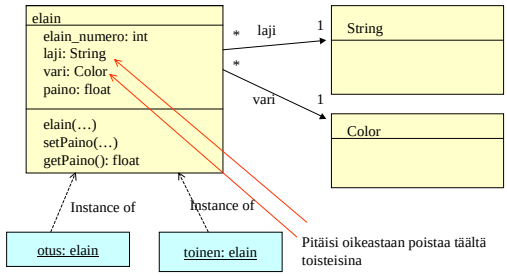
Reaalimaailma ja luokka



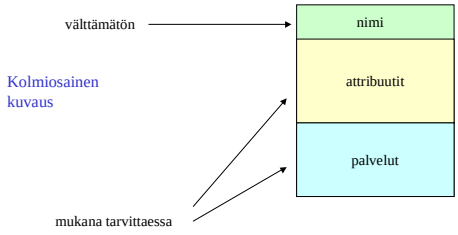
Olio-ohjelma ja luokka



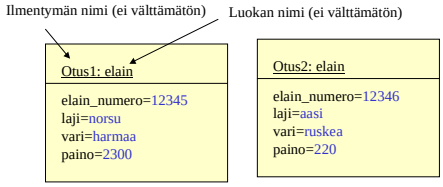
Luokkakuvaus UML:llä



Luokkakuvaus UML:ssä



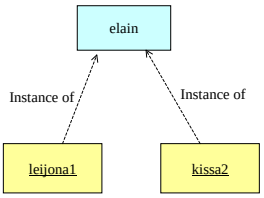
Ilmentymän kuvaaminen



Ilmentymiä esitetään yleensä vain esimerkkeinä

Ilmentymän kuvaaminen

Ilmentymät luokkakaaviossa



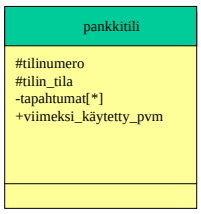
Luokan attribuutit

- Olioiden tietämys (tietosisältö) kuvataan attribuutteina
- Attribuutista voidaan esittää
  - nimi (välttämätön)
  - tietotyyppi
  - arvojen määrä
    - *(hakasuluissa, alaraja..yläraja, *=epämääräisen monta)*
  - näkyvyys
    - *vain ohjelmointiläheisessä kuvauksessa*
  - oletusarvo
  - muita määreitä

Luokan attribuutit

- Näkyvyys (visibility) on esim. C++ ja Java-ohjelmointikielissä käytetty tekniikka, jolla säädellään luokkakuvauksessa esitellyn muuttujan tai metodin (palvelun) käyttömahdollisuuksia.
  - private (-) rajaa käytön luokan omiin palveluihin
  - protected (#) sallii käytön myös luokan aliluokissa määriteltävissä palveluissa (
    - *puhdasoppisesti ajateltuna kaikkien attribuuttien tulisi olla näkyvyydeltään protected*
  - public (+) sallii käytön myös luokan ulkopuolelta
- UML käyttää yllä sulkeissa olevaa merkkiä attribuutin nimen edessä osoittamaan näkyvyyden.

Luokan attribuutit



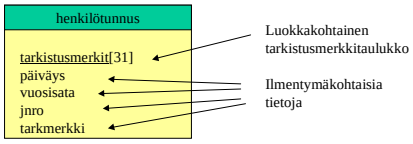
Attribuutit **tilinumero** ja **tilin_tila** ovat käytettävissä luokan **pankkitili** ja sen aliluokkien palveluissa (esim. luokassa **käyttötili**).

Moniarvoiseen attribuuttiin **tapahtumat** voi viitata vain **pankkitilin** omissa palveluissa.

Attribuuttiin **viimeksi_käytetty_pvm** voi viitata myös ulkopuolisten luokkien palveluissa.

Luokan attribuutit

- UML:ssä on mahdollista määritellä:
  - luokka-attribuutteja (class attribute),
    - *attribuutilla luokkakohtainen arvo*
    - ***ilmaistaan alleviivaamalla** attribuutin nimi*
  - ilmentymäattribuutteja (instance attribute)
    - *arvo erikseen jokaisessa luokan ilmentymässä*



Luokan attribuutit

- Attribuutin arvo on jotain tyyppiä. Tyyppi ilmaistaan antamalla se nimen perässä kaksoispisteellä erotettuna:
  - päiväys:Date
  - jnro:int
  - tarkastusmerkki: char
- Arvo voi olla
  - **perustietotyypin arvo**
    - *UML ei ota kantaa tietotyyppeihin, vaan sallii mitä tahansa perustietotyyppejä.*
    - *yllä int ja char*

Luokan attribuutit

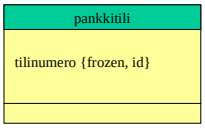
- **Olio, jolloin tietotyyppinä on olion luokka**
  - *Olioarvoinen attribuutti muodostaa **yhteyden** attribuutin sisältävän olion ja attribuutin arvona olevan olion välille. Tällaiset yhteydet pitäisi tuoda selkeästi esiin ja kuvata omalla tekniikallaan (luokkia yhdistävinä viivoina).*
  - *Jos attribuutin arvona on olio, tämä tulisi esittää attribuuttimäärittelyn yhteydessä vain, mikäli olion luokka on perustietotyyppimäinen, esim Date, String, Color, Point, …*
  - *päiväys:Date*
  - *nimi:String*

Luokan attribuutit

- Attribuuttimäärittelyyn voi liittää muiden määrittelyjen jälkeen kaarisulkeissa {...} lisämääreitä.
- UML tarjoaa valmiina lisämääreet:
  - changeable (oletus)
    - *arvon muuttamiseen ei ole rajoitteita*
  - frozen
    - *arvoa ei voi muuttaa*
  - addOnly
    - *moniarvoiselle attribuutille voi vain lisätä arvoja, ei koskaan poistaa*

Luokan attribuutit

- UML-mallin soveltaja voi ottaa käyttöön omia lisämääreitä. Hyödyllisiä voisivat olla
  - id
    - *ulkoinen tunniste, attribuutin arvo ei voi olla sama kahdella eri ilmentymällä*



Tilinumeroa ei voi muuttaa, se yksilöi tilit.

Jos usealla attribuutilla on id-määre, ne yhdessä yksilöivät kohteen

Luokan attribuutit ja Java ohjelma

- UML-luokka vastaa Java-luokkaa.
  - Javan static-muuttujat vastaavat luokka-attribuutteja.
  - Muut Java-luokassa määritellyt muuttujat vastaavat ilmentymäattribuutteja.
  - Javan protected tulkinta on hieman edellä esitetystä poikkeava

Luokan palvelut

- Palvelusta UML:ssä voidaan kuvata:
  - näkyvyys - kuten attribuuttien kohdalla
  - nimi (välttämätön)
  - parametrit
  - paluuarvon tyyppi
  - muut määreet

Hakasulut ilmaisevat valinnaisen elementin

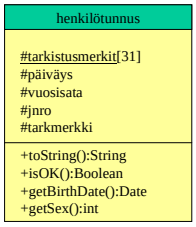
[näkyvyys] nimi [(parametrit)] [:paluuarvon tyyppi] [{muut määreet}]

  - parametrimäärittelyn muoto on

[suunta] nimi: tyyppi [= oletusarvo]

Luokan palvelut

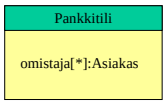
- Esim.



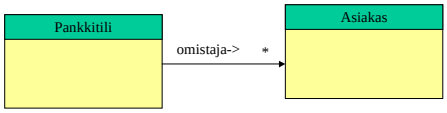
Olioiden väliset yhteydet

- Olioiden (ilmentymien) välisiä rakenteellisia kytkentöjä kutsutaan yhteyksiksi (association).
- Ohjelmakoodissa rakenteellinen kytkentä ilmenee siten, että luokalla on olioarvoinen attribuutti tai jokin epäsuora viittaus toiseen olioon.

Olioiden väliset yhteydet



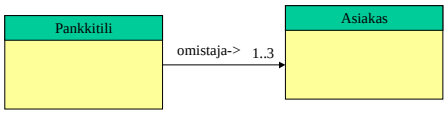
Tarkoittaa, että pankkitili-olioiden ja asiakas-olioiden välillä on kytkentä, joka voidaan kuvata seuraavasti:



Yhden tilin voi omistaa useita asiakkaita.

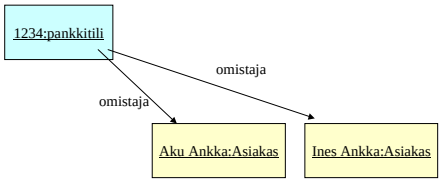
Olioiden väliset yhteydet

Määritellään Java-ohjelmassa pankkitilin omistaja
Asiakas[] omistaja = new Asiakas[3];
eli tilillä voi olla enintään 3 omistajaa. Pankkitilillä täytyy aina olla omistaja (tämä on tarkistettava ohjelmassa). Ylläoleva esitettäisiin kaaviona:



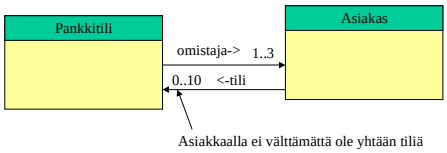
Olioiden väliset yhteydet

- Edellä on kyseessä suunnattu yhteys: pankkitilistä päästään asiakkaaseen, mutta asiakkaasta ei päästä pankkitiliin, eli ilmentymätasolla tilanne olisi seuraava:



Olioiden väliset yhteydet

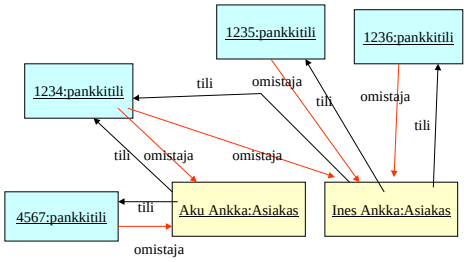
- Jos haluttaisiin yhteys myös toiseen suuntaan (ja edelleen taulukoilla), voisimme määritellä asiakkaalle attribuutin
- Pankkitili[] tili=new Pankkitili[10];



Asiakkaalla ei välttämättä ole yhtään tiliä

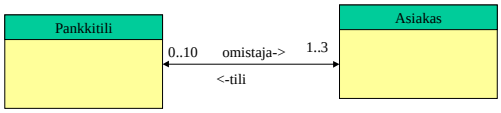
Olioiden väliset yhteydet

- Ilmentymätasolla tilanne voisi olla:

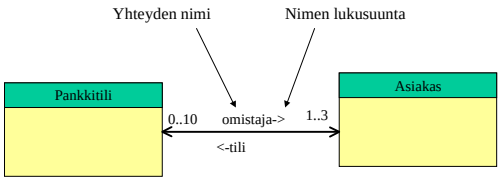


Olioiden väliset yhteydet

- Tili- ja omistaja-yhteydet kuvaavat samaa asiaa käänteisesti ja ne voidaan yhdistää Ilmentymätasolla mikään ei muutu, kaaviotasolla viivojen määrä vähenee.



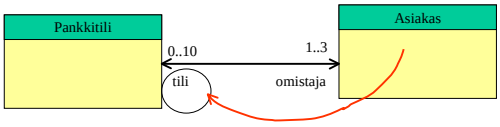
Olioiden väliset yhteydet



Yhteydelle voi antaa nimen kumpaankin suuntaan.
Sille ei tarvise antaa lainkaan nimeä.
Yhteysnimen asemasta/lisäksi voidaan käyttää roolinimiä.

Olioiden väliset yhteydet

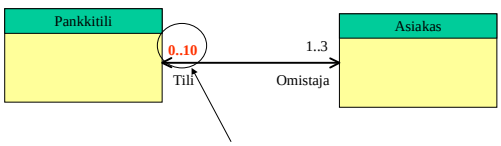
Tässä tili ja omistaja ovat roolinimiä,
Ohjelmaa kuvattaessa kumppanin roolinimenä voi käyttää olion sisältämän kumppaniin viittaavan muuttujan nimeä.



tili-muuttuja (tässä tapauksessa taulukko) sisältyy jokaiseen Asiakas-olioon. Sitä ei kuitenkaan suositella esitettäväksi asiakkaan attribuuttina, jotta vältettäisiin tarpeetonta saman asian toistoa. Vastaavasti omistaja on pankkitiliin sisältyvä taulukkomuuttuja.

Olioiden väliset yhteydet

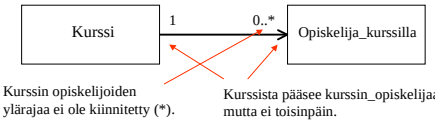
Tärkeää



Asiakkaaseen kytkettyjen pankkitilien minimi- ja maksimimäärä ilmoitetaan kumppanin (siis pankkitilin) puoleisessa päässä yhteysviivaa.

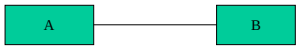
Olioiden väliset yhteydet

- Yhteysviivan päässä oleva nuolenkärki kuvaa ns. navigointimahdollisuutta eli sitä, että viivan päässä olevan luokan olioista pääsee suoraan nuolella osoitetun luokan olioon (yhteen tai useampaan) - oliossa on siis siirtymän mahdollistava olioarvoinen muuttuja.



Olioiden väliset yhteydet

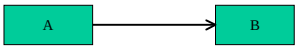
- Kun kuvauksen abstraktiotasoa nostetaan, jätetään navigointimahdollisuus yleensä kuvaamatta. Tällöin yhteysviivan kummassakaan päässä ei ole nuolenkärkeä.



Tämän ei tarkoita sitä, ettei A:sta pääse B:hen eikä päinvastoin, vaan sitä, että navigointimahdollisuus on jätetty esittämättä. Navigointimahdollisuuden voi tällöin ajatella olevan molemminsuuntainen.

Olioiden väliset yhteydet

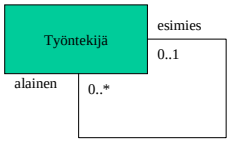
Jos yhteysviivan toisessa päässä kuvataan navigointimahdollisuus, se tulkitaan kuvatuksi myös toisessa päässä.



Näin esitettynä B-olioista ei ole suoraa pääsyä A-olioihin.

Olioiden väliset yhteydet

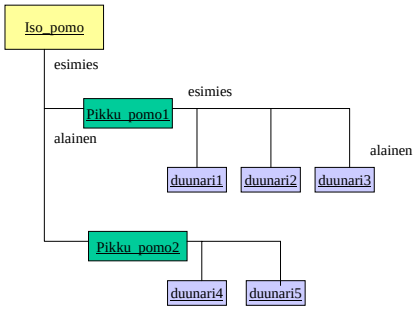
- Molemmat yhteyden osapuolina olevat oliot voivat kuulua samaan luokkaan.



Työntekijällä voi olla vain yksi (välitön) esimies.

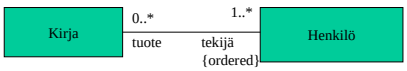
Työntekijällä voi olla useita alaisia, ei välttämättä yhtään.

Olioiden väliset yhteydet



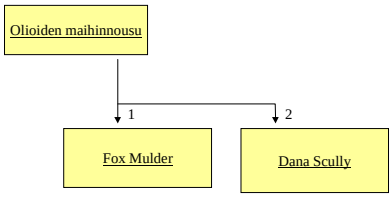
Olioiden väliset yhteydet

- Yhteyteen voidaan liittää järjestysmääre kuvaamaan sitä, että samaan olioon yhteydessä olevien olioiden joukko on jollain perusteella järjestetty.



Kirjan tekijöiden joukko on järjestetty.

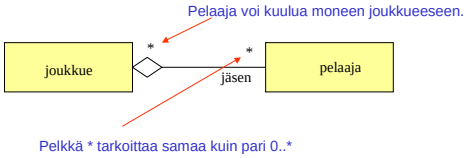
Olioiden väliset yhteydet



Järjestetty joukko

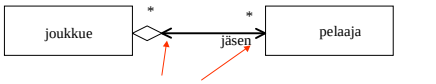
Olioiden väliset yhteydet

- Jos yhteyden osapuolta voidaan pitää osana toista osapuolta, on kyseessä **kooste** (aggregate). Tällainen yhteyden erikoistapaus voidaan esittää yhteysviivan kokonaisuuden puoleiseen päähän sijoitetun salmiakki-symbolin avulla.



Olioiden väliset yhteydet

- Kooste auttaa hahmottamaan sitä, miten oliot semanttisesti kytkeytyvät toisiinsa. Sillä ei välttämättä ole mitään vaikutusta esimerkiksi navigointimahdollisuuksiin. Yleensä kuitenkin kokonaisuudesta on aina pääsy osiinsa.



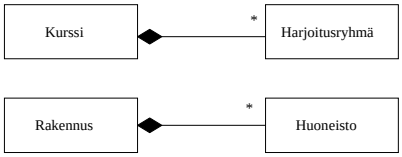
Pelaajasta on suora pääsy pelaajan joukkueseen ja joukkueesta sen pelaajiin.

Olioiden väliset yhteydet

- Ohjelman toiminnan kannalta koostetta merkittävämpi yhteys on **kompositio**-yhteys (composition).
- Kompositio on koosteen erikoistapaus, jossa
  - **osan olemassaolo** on kytketty kokonaisuuden olemassaoloon - kun kokonaisuus hävitetään, häviävät myös sen osat (tavallisen koosteen kohdalla näin ei käy)
  - osa voi sisältyä vain yhteen samantyyppisen kompositioon
  - osa ei voi vaihtaa kompositiotaan, vaan on aina osa samaa kokonaisuutta

Olioiden väliset yhteydet

- Esimerkkejä:



Kompositio esitetään mustalla salmiakilla kokonaisuuden puoleisessa päässä.

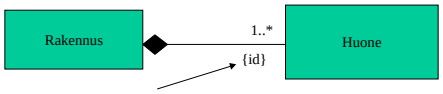
Olioiden väliset yhteydet

- Kompositio on UML:ssä ainoa tapa ilmaista olemassaoloriippuvuus.
- Olemassaoloriippuvuus on mallintamisessa hyvin tärkeä asia, tärkeämpi kuin osan ja kokonaisuuden yhteys.
- Jos olemassaoloriippuvuus pitää kyetä esittämään, on syytä käyttää kompositiota, vaikka osa-kokonaisuus-yhteys ei aivan selvältä näyttäisikään.

Olioiden väliset yhteydet

- Kompositiota käytetään reaalimaailmassa usein hyväksi olioiden identifioinnissa.
- ‘Rakennuksen Teollisuuskatu 23 huone B446’ pitää sisällään kompositionin.
- Perus-UML ei tarjoa keinoa kompositionkautta tapahtuvan ulkoisen identifioinnin kuvaamiseen, joten tällä kurssilla otetaan käyttöön yhteytessä osan puolelle liitettävä lisämääre {id} esittämään tätä.

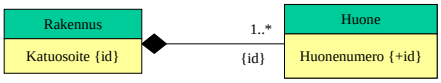
Olioiden väliset yhteydet



Huoneen yksikäsitteiseen identifiointiin tarvitaan tieto rakennuksesta, jossa huone sijaitsee.

Koska rakennuksessa voi olla useita huoneita tarvitaan identifiontiin lisäksi jokin rakennuksen sisäinen tunniste. Käytetään tällaisen osoittamiseen merkintää {+id}

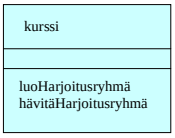
Olioiden väliset yhteydet



Katuosoite ja huonenumero yhdessä identifioivat huoneen.
Esim: Teollisuuskatu 23, B446

Olioiden väliset yhteydet

- Kompositioita ohjelmissa käsiteltäessä kokonaisuusolio on vastuussa osiensa luonnista ja hävittämisestä.



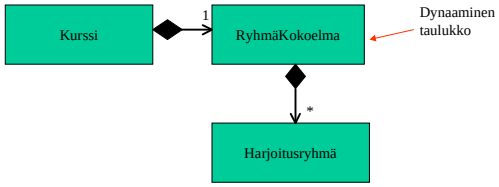
Yhteyskategoriat

- Yhteydet voidaan jakaa osallistumis-rajoitteiden ylärajojen perusteella tyyppeihin seuraavasti
  - 1:N (lue yhden suhde moneen)
    - *toisella osapuolen ilmentymillä voi olla monta kumppania, mutta toisen ilmentymillä enintään yksi,*
    - *esim. Kurssi-Harjoitusryhmä,*
  - N:M (lue monen suhde moneen) ja
    - *kummankin osapuolen ilmentymillä voi olla useita kumppaneita. Esim Kurssi-Oppikirja*
  - 1:1 (lue yhden suhde yhteen)
    - *kummakin osapuolen ilmentymillä voi olla enintään yksi kumppani, esim Henkilö-Ajokortti*

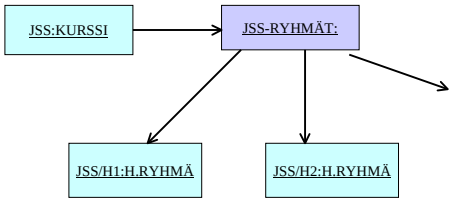
Yhteydet ja abstraktiotasot

Tarkastellaan tilannetta, jossa kurssin harjoitusryhmät on taulukon asemasta toteutettu RyhmaKokoelma-luokan avulla. Tämä luokka toteuttakoon dynaamisen taulukon (= taulukko, joka laajenee tarvittaessa).

Yksityiskohtaisessa kuvauksessa tilanne olisi tällöin:



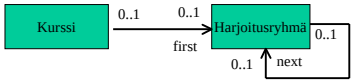
Yhteydet ja abstraktiotasot



ilmentymätasolla

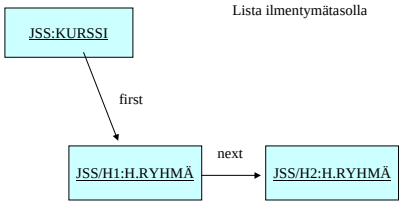
Yhteydet ja abstraktiotasot

Sama asia voitaisiin toteuttaa myös vaikkapa listarakenteena.

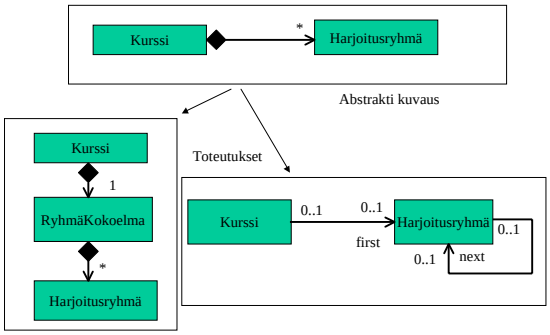


Näissä kummassakin kuvauksessa, erityisesti ylläolevassa, ratkaisun yksityiskohdat vaikeuttavat oleellisen kokonaisuuden hahmotusta = sitä, että kurssiin liittyy useita harjoitusryhmiä, jotka ovat olemassaoloriippuvia kurssista. Kuvauksen abstraktiotasoa on tällöin syytä nostaa häivyttämällä tekninen ratkaisu.

Yhteydet ja abstraktiotasot

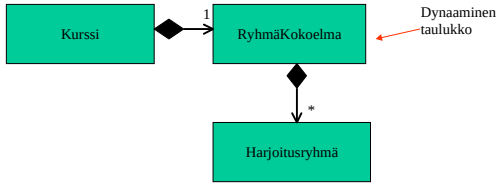


Yhteydet ja abstraktiotasot



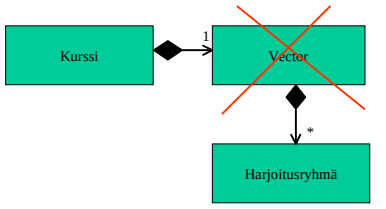
Kokoelman toteutus

- Tarkastellaan vielä aiempaa esimerkkiä



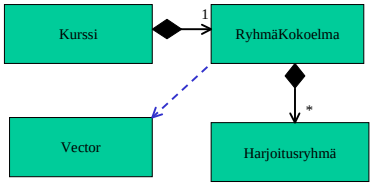
Tässä ratkaisussa RyhmäKokoelma todennäköisesti toteutettaisiin ohjelmointikielen tarjoamaa valmista dynaamista taulukkoa käyttäen (Javassa luokka Vector) - miten tämä kuvataan UML:llä?

Kokoelman toteutus



Ratkaisu on virheellinen, sillä kaikki Vector-luokan ilmentymät eivät toki ole olemassaoloriippuvia Kursseista.

Kokoelman toteutus



UML:ssä on mahdollista määritellä riippuvuuksia luokkien välille, tässä on kyse riippuvuudesta yleiskäyttöisen luokan (Vector) ja sillä toteutetun luokan (RyhmäKokoelma) välillä.

Kokoelman toteutus

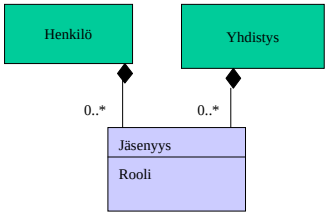
- Java-ohjelmana edellisen kalvon esimerkki toteutuisi seuraavasti:

```
public class Kurssi {
   Vector RyhmäKokoelma;

   public void  lisääRyhmä(HarjoitusRyhmä h) {
      RyhmäKokoelma.addItem(h);
   }
```

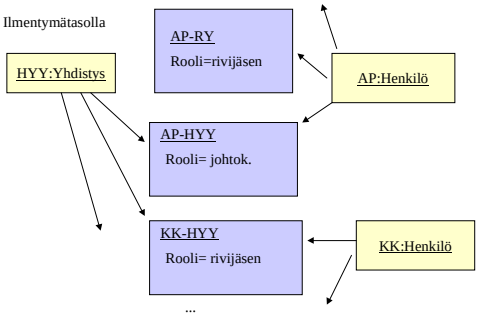
Tässä siis luokkaa RyhmäKokoelma ei ole määritelty. On vain Vector-luokan ilmentymä, mutta koska Vector on geneerinen yleiskäyttöinen luokka, sen ilmentymä voidaan ajatella tarkoitukseen sidotuksi luokaksi.

Yhteydet ja abstraktiotasot



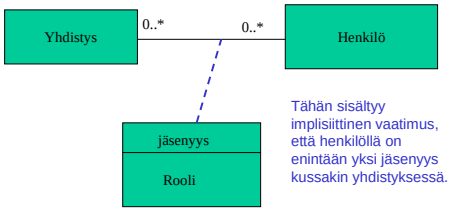
Tässä esimerkissä jäsenyys-olioiden avulla kuvataan henkilön jäsenyyttä yhdistyksessä. Henkilö voi olla rivijäsen, johtokunnassa tai kunniajäsen.

Yhteydet ja abstraktiotasot



Yhteydet ja abstraktiotasot

Tilanne voitaisiin mallintaa myös ns. yhteysluokan (association class) avulla.



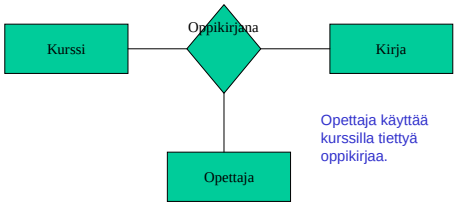
Tähän sisältyy implisiittinen vaatimus, että henkilöllä on enintään yksi jäsenyys kussakin yhdistyksessä.

Yhteydet ja abstraktiotasot

- Yhteysluokkaa käyttäen voidaan kahden olion väliseen kytkentään liittää kytkennän laatua kuvaavia attribuutteja.

Yhteydet

- UML:ssä on mahdollista kuvata myös useamman kuin kahden olion välisiä kytkentöjä.



Opettaja käyttää kurssilla tiettyä oppikirjaa.

Luokkien väliset suhteet

- Edellä on tarkasteltu olioiden (ilmentymien) välisiä yhteyksiä. Luokkakaaviossa nämä kuvataan luokkien välillä.
- UML:ssä voidaan lisäksi esittää
  - luokkien välisiä riippuvuuksia (dependency) ja
  - luokkahierarkia

Riippuvuus

- Luokien välisellä riippuvuudella tarkoitetaan tilannetta, jossa luokan määrittelyissä tapahtuvalla muutoksella voi olla vaikutuksia toisen luokan toimintaan. Riippuvuus kuvataan katkoviivalla, jonka päässä oleva nuolenkärki osoittaa siihen luokkaan, josta viivan toisessa päässä oleva on riippuva.

Käyttäjä on riippuvuussuhteessa käytettävään.

Riippuvuus

- Esimerkki riippuvuudesta on palvelun parametrin aiheuttama riippuvuus, jossa palvelun tarjoava luokka tulee riippuvaksi parametrin luokasta.

```
Class Käyttäjä {

  public omaPalvelu(Käytettävä k) {
     k.vierasPalvelu();
  }
}
```

Riippuvuus

- Aiemmin oli jo esillä riippuvuus yleiskäyttöisestä luokasta kokoelman toteutuksessa.

UML:ssä on nimetty 8 erilaista luokkakaaviossa mahdollista riippuvuutta. Näiden määrittelyt ovat kuitenkin osin varsin epämääräisiä.