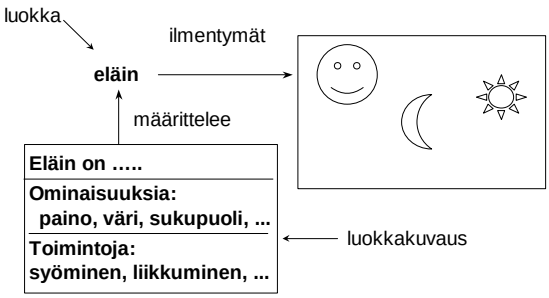
Luokka- ja oliokaaviot

- 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

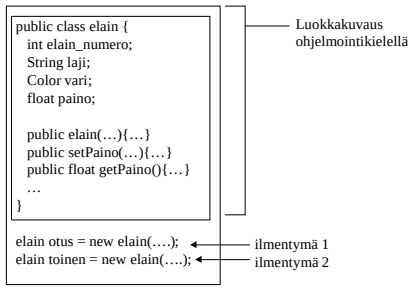
Luokka- ja oliokaaviot

- (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.

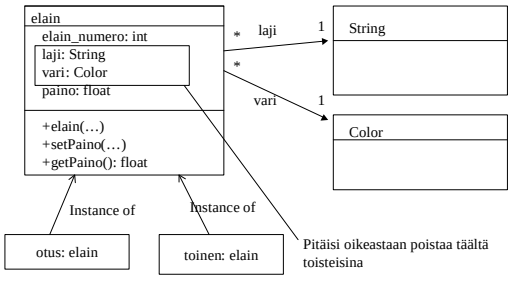
Reaalimaailma ja luokka



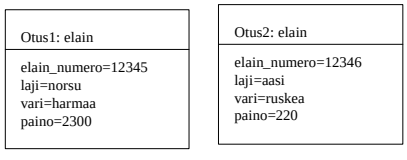
Olio-ohjelma ja luokka



Luokkakuvaus UML:llä

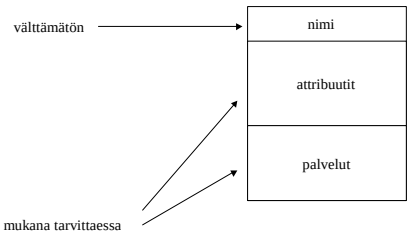


Ilmentymän kuvaaminen



Ilmentymiä esitetään yleensä vain esimerkkeinä

Luokkakuvaus UML:ssä



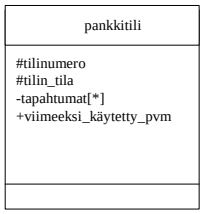
Luokan attribuutit

- 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) käyttöä.
  - private (-) rajaa käytön luokan omiin palveluihin
  - protected (#) sallii käytön 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



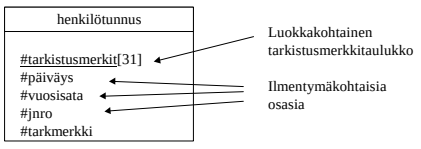
Attribuutit tilinumero ja tilin_tila ovat käytettävissä luokan pankkitili ja sen aliluokkien palveluissa (esim. luokassa käyttötili)

Moniarvoiseen attribuuttiin tapahtumat voi viitata pankkitilin omissa palveluissa

Atribuuttiin viimeeksi_käytetty voi viitata myös ulkopuolisten luokien palveluissa

Luokan attribuutit

- UML:ssä on mahdollista määritellä
  - luokka-attribuutteja (class attribute),
    - *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
  - **perustietotyyppin arvo**
    - *UML ei ota kantaa tietotyyppeihin vaan sallii mitä tahansa perustietotyyppejä*
    - *yllä int ja char*

Luokan attribuutit

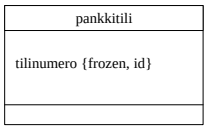
- **Olio, jolloin tietotyyppinä on olion luokka**
  - *Olioarvoinen attribuutti muodostaa **yhteyden** attribuutin sisältävän olion ja attribuutin arvona olevan olion välille. Tällaiset yhteydet pitäisi tuoda selkeästi esiin ja kuvata omalla tekniikallaan (luokkia yhdistävinä viivoina)*
  - *Jos attribuutin 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

Luokan attribuutit ja Java ohjelma

- UML-luokka vastaa Java-luokkaa.
  - Static-muuttujat vastaavat luokka-attribuutteja
  - muut luokassa määritellyt muuttujat vastaavat ilmentymäattribuutteja

Luokan palvelut

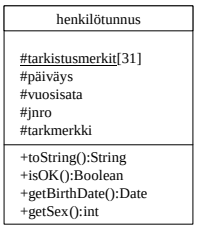
- Palvelusta UML:ssä voidaan kuvata
  - näkyvyys - kuten attribuuttien kohdalla
  - nimi (välttämätön)
  - parametrit
  - paluuarvon tyyppi
  - muut määreet

    [näkyvyys] nimi [(parametrit)] [:paluuarvon tyyppi] [{muut määreet}]
  - parametrimäärittelyn muoto on

    [suunta] nimi: tyyppi [= oletusarvo]

Luokan palvelut

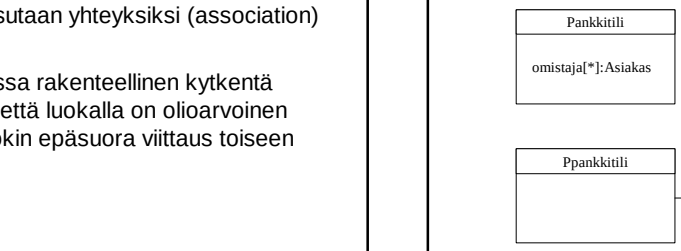
- Esim.



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



Yhden tilin voi omistaa useita asiakkaita

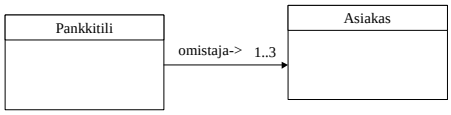
Olioiden väliset yhteydet

Oletetaan, että Java-ohjelmassa pankkitilin omistaja olisi määritelty seuraavasti

Asiakas[] omistaja = new Asiakas[3];

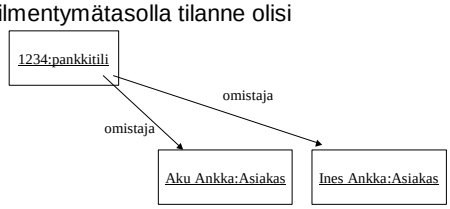
eli tilillä voi olla enintään 3 omistajaa. Pankkitilillä täytyy aina olla omistaja (tämä on Java-ohjelmassa tarkistettava ohjelmassa)

Ylläoleva esitettäisiin:



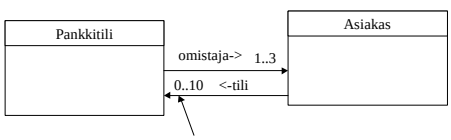
Olioiden väliset yhteydet

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



Olioiden väliset yhteydet

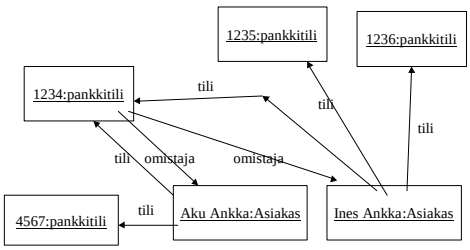
- Jos haluttaisiin yhteys myös toiseen suuntaan (ja edelleen taulukoilla), voisimme määritellä asiakkaalle attribuutin
- Pankkitili[] tili=new Pankkitili[10];



Koska asiakkaalla ei välttämättä ole yhtään tiliä

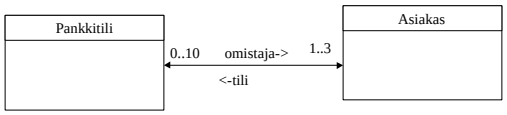
Olioiden väliset yhteydet

- Ilmentymätasolla tilanne voisi olla

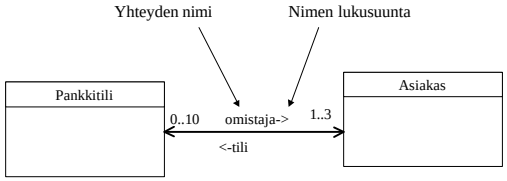


Olioiden väliset yhteydet

- Huomaamme, että tili ja omistaja yhteydet kuvaavat samaa asiaa ja voimme yhdistää ne. Ilmentymätasolla mikään ei muutu



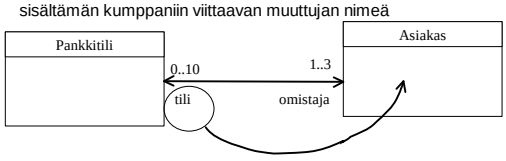
Olioiden väliset yhteydet



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

Olioiden väliset yhteydet

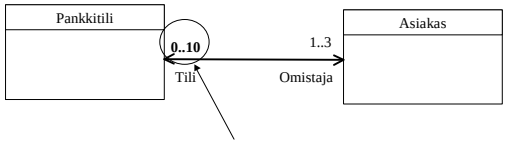
Tässä tili ja omistaja ovat roolinimiä,
Ohjelmaa kuvattaessa kumppanin roolinimenä voi käyttää olion sisältämän kumppaniin viittaavan muuttujan nimeä



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

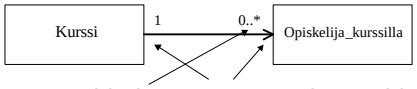
Tärkeää



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



Kurssin opiskelijoiden ylärajaa ei ole kiinnitetty (*)

Kurssista pääsee kurssin_opiskelijaan mutta ei toisinpäin

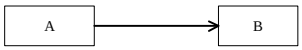
Olioiden väliset yhteydet

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

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

Olioiden väliset yhteydet

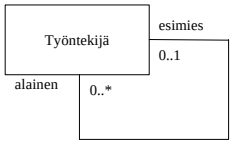
Jos yhteysviivan toisessa päässä kuvataan navigointimahdollisuus se tulkitaan kuvatuksi myös toisessa päässä



näin esitettynä B-olioista ei ole suoraa pääsyä A-olioihin

Olioiden väliset yhteydet

- Molemmat yhteyden osapuolina olevat oliot voivat kuulua samaan luokkaan.

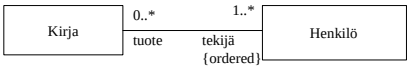


Työntekijällä voi olla vain yksi välitön esimies

Työntekijällä voi olla useita alaisia, ei välttämättä yhtään

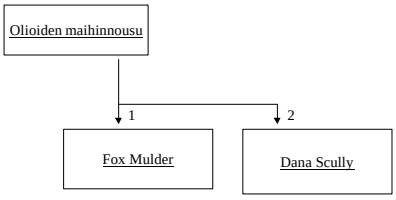
Olioiden väliset yhteydet

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



Kirjan tekijöiden joukko on järjestetty

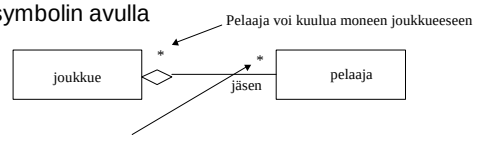
Olioiden väliset yhteydet



Järjestetty joukko

Olioiden väliset yhteydet

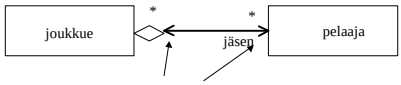
- Jos yhteyden osapuolta voidaan pitää osana toista osapuolta on kyseessä **kooste** (aggregate). Tällainen yhteyden erikoistapaus voidaan esittää yhteysviivan kokonaisuuden puoleiseen päähän sijoitetun salmiakki-symbolin avulla



Pelkkä * tarkoittaa samaa kuin pari 0..*

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.



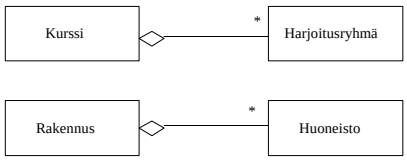
Pelaajasta on suora pääsy pelaajan joukkueeseen ja joukkueesta sen pelaajiin

Olioiden väliset yhteydet

- Ohjelman toiminnan kannalta koostetta merkittävämpi yhteys on **kompositio**-yhteys (composition). Kompositio voidaan ajatella koosteen erikoistapauksena, 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

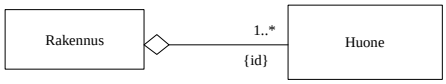
Olioiden väliset yhteydet

- 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 kompositiota syytä käyttää 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 komposition.
- Perus-UML ei tarjoa keinoa komposition kautta tapahtuvan ulkoisen identifioinnin kuvaamiseen, joten tällä kurssilla otetaan käyttöön yhteyteen osan puolelle liitettävä lisämääre {id} esittämään tätä.

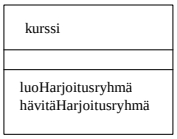
Olioiden väliset yhteydet



Huoneen yksikäsitteiseen identifiointiin tarvitaan tieto rakennuksesta, jossa huone sijaitsee

Olioiden väliset yhteydet

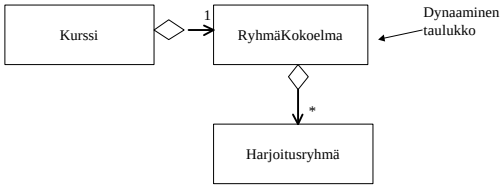
- Ohjelmassa komposition tapauksessa kokonaisuusolio on vastuussa osiensa luonnista ja hävittämisestä.



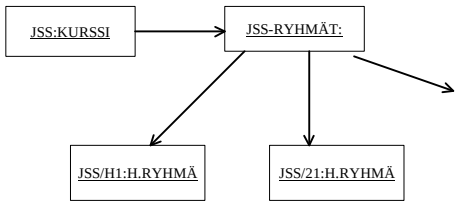
Yhteydet ja abstraktiotasot

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:



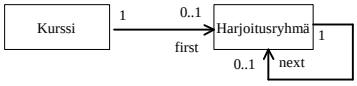
Yhteydet ja abstraktiotasot



ilmentymätasolla

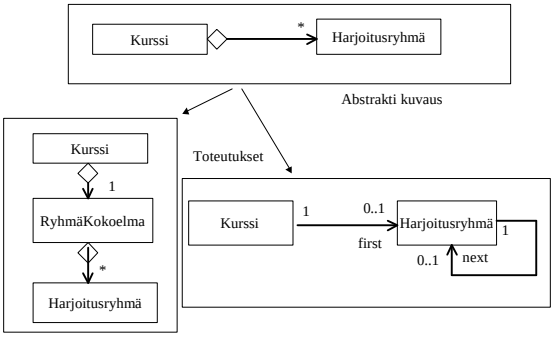
Yhteydet ja abstraktiotasot

Sama asia voitaisiin toteuttaa myös vaikkapa listarakenteena.



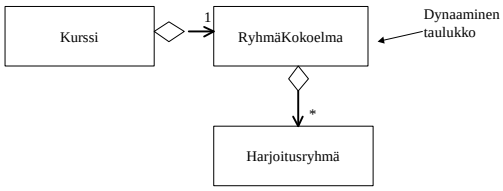
Näissä kummassakin kuvauksessa, erityisesti ylläolevassa, ratkaisun yksityiskohdat vaikeuttavat oleellisen kokonaisuuden hahmotusta sitä, että kurssiin liittyy useita harjoitusryhmiä, jotka ovat olemassaoloriippuvia kurssista. Kuvauksen abstraktiotasoa on tällöin syytä nostaa häivyttämällä tekninen ratkaisu.

Yhteydet ja abstraktiotasot



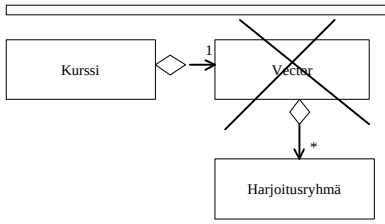
Kokoelman toteutus

- Tarkastellaan vielä aiempaa esimerkkiä



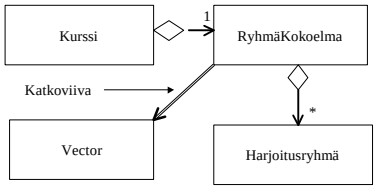
Tässä ratkaisussa RyhmäKokoelma todennäköisesti toteutettaisiin ohjelmointikielen tarjoamaa valmista dynaamista taulukkoa käyttäen (Javassa Vector) - miten tämä kuvataan UML:llä?

Kokoelman toteutus



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

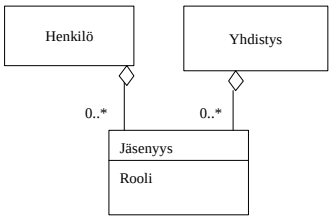
- Java ohjelmana edellisen kalvon esimerkki toteutuisi seuraavasti

```
public class Kurssi {
   Vector RyhmäKokoelma;

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

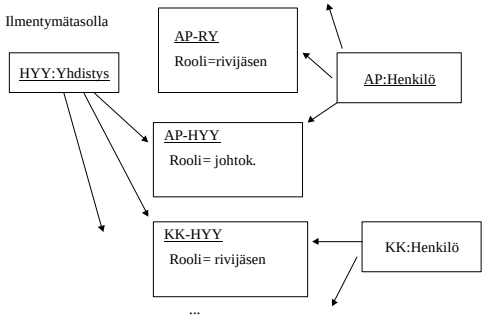
Tässä siis luokkaa RyhmäKokoelma ei ole määritelty. On vain Vector luokan ilmentymä, mutta koska Vector on geneerinen yleiskäyttöinen luokka sen ilmentymä voidaan ajatella tarkoitukseen sidotuksi luokaksi

Yhteydet ja abstraktiotasot



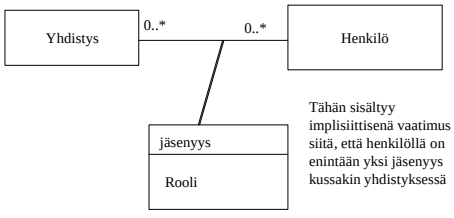
Tässä esimerkissä jäsenyys olioiden avulla kuvataan henkilön jäsenyyttä yhdistyksessä. Henkilö voi olla rivijäsen, johtokunnassa tai kunniajäsen.

Yhteydet ja abstraktiotasot



Yhteydet ja abstraktiotasot

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



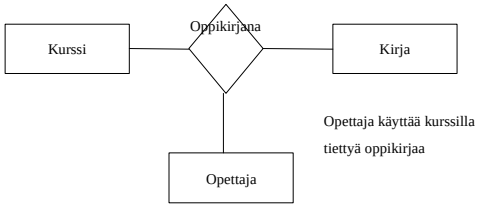
Tähän sisältyy implisiittisenä vaatimus siitä, että henkilöllä on enintään yksi jäsenyys kussakin yhdistyksessä

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

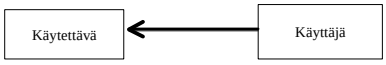


Luokkien väliset suhteet

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

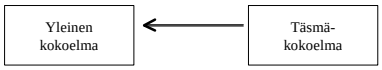
- Esimerkki riippuvuudesta on palvelun parametrin aiheuttama riippuvuus, jossa palvelun tarjoava luokka tulee riippuvaksi parametrin luokasta.

```
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