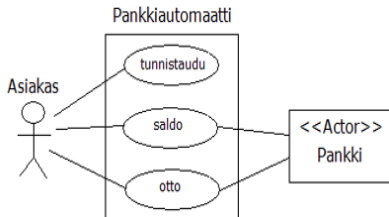


# Ohjelmistotekniikan menetelmät

luento 3, 15.11.2016

# Kertaus - käyttötapausmalli

- ▶ 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*
  - ▶ Käyttäjäksi luokitellaan myös ulkoiset järjestelmät, joihin kuvattava järjestelmä on yhteydessä
- ▶ Yleiskuva järjestelmän tarjoamasta toiminnallisuudesta voidaan esittää *käyttötapauskaaviona*



# Kertaus - käyttötapausmalli

- Ohjelmistoprojektissa tulee kuvata kaikki käyttötapaukset tekstuaalisesti saman kaavan, eli *käyttötapauspohjan*, mukaan

## Käyttötapaus 1: otto

**Tavoite** Asiakas nostaa tililtään haluamansa määrän rahaa

**Käyttäjät** Asiakas, Pankki

**Esiehto** Kortti syötetty ja asiakas tunnistautunut

**Jälkiehto** käyttäjä saa tililtään haluamansa määrän rahaa.  
Jos saldo ei riitä, tiliä ei veloiteta

### Käyttötapauksen kulku:

1. asiakas valitsee otto-toiminnon
2. automaatti kysyy nostettavaa summaa
3. asiakas syöttää haluamansa summan
4. pankilta tarkistetaan riittääkö asiakkaan saldo
5. summa veloitetaan asiakkaan tililtä
6. kuitti tulostetaan ja annetaan asiakkaalle
7. pankkikortti palautetaan asiakkaalle
8. rahat annetaan asiakkaalle

### Poikkeustapaukset:

**4a** asiakkaan tilillä ei tarpeeksi rahaa, palautetaan kortti

# Oliot ja luokat

## Olio...

- ▶ on 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
- ▶ kuuluu johonkin **luokkaan**

# Oliot ja luokat

## Henkilö-luokka:

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

## Henkilö-olioita:

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

# Oliot ja luokat

## Suunnittelu ja toteutusvaiheen oliot ja luokat

- ▶ *Luokka* kuvaa minkälaisia siihen kuuluvat oliot ovat tietosisällön ja toiminnallisuuden suhteen
- ▶ Oliota ja luokkia ajatellaan usein ohjelmointitason käsitteinä
  - ▶ Ohjelma 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?

# Oliot ja luokat

## Vaatusanalyysivaiheen oliot ja luokat

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

# Oliot ja luokat

## Vaatimusanalyysivaiheen oliot ja luokat

- ▶ **Vaatimusanalyysivaiheen luokat ovat vastineita reaailmaailman 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
  - ▶ reaailmaailman käsitteen vastine, tai
  - ▶ suunnittelu- ja ohjelmointitason "tekninen" asia
- ▶ Tyypillisesti ohjelmatuson olio on vastine jollekin todellisuudessa olevalle "oliolle" (*simuloidaan todellisuutta*)
- ▶ Ohjelmissa on myös paljon "tekniisiä" luokkia ja olioita, joille ei ole vastinetta todellisuudessa
  - ▶ esim. käyttöliittymän toteuttavat oliot, tiedostojen ja tietokantojen kanssa kommunikoinnin hoitavat oliot



# Oliot ja luokat

## Oliomallinnus ja olioperustainen ohjelmistokehitys

- ▶ Olioperustainen ohjelmistokehitys etenee yleensä seuraavasti:
  1. Luodaan **määrittelyvaiheen oliomalli** sovelluksen käsitteistöstä
    - ▶ Mallin oliot ja luokat ovat rakennettavan sovelluksen kohdealueen käsitteiden vastineita
  2. Suunnitteluvaiheessa tarkennetaan edellisen vaiheen malli **suunnitteluvaiheen oliomalliksi**
    - ▶ Oliot muuttuvat yleiskäsitteistä teknisen tason olioiksi
    - ▶ Mukaan tulee olioita, joilla ei suoraa vastinetta reaali maailmassa
    - ▶ Osa olioista on luonteeltaan *pysyviä* ja niitä tulee vastaamaan jokin rakenne ohjelman tietokannassa
  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

Olioden ja luokkien kuvaus UML:ssä

# Luokka- ja oliokaaviot

## Olioiden ja luokkien kuvaus UML:ssä

- ▶ Miten olioita ja luokkia voidaan kuvata UML:ssä
- ▶ Järjestelmän luokkarakennetta kuvaa **luokkakaavio** (engl. *class diagram*)
  - ▶ Mitä luokkia on olemassa
  - ▶ Minkälaisia luokat ovat
  - ▶ Luokkien ja niiden olioiden suhteet toisiinsa
- ▶ Luokkakaavio on UML:n eniten käytetty kaaviotyyppi
- ▶ Luokkakaavio kuvaa järjestelmän kaikkia mahdollisia olioita, ja niiden välisiä suhteita
- ▶ **Oliokaavio** (engl. *object diagram*) taas kuvaa mitä olioita järjestelmässä on tietyllä hetkellä

# Luokka- ja oliokaaviot

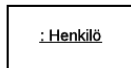
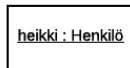
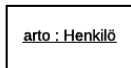
## Olioiden ja luokkien kuvaus UML:ssä

- ▶ Luokkaa kuvataan laatikolla, jonka sisällä on luokan nimi
- ▶ Luokasta luotuja olioita kuvataan myös laatikolla, erona on nimen merkintätapa
  - ▶ Nimi alleviivattuna, sisältäen mahdollisesti myös olion nimen
- ▶ Kuvassa Henkilö-luokka ja kolme Henkilö-olioa
- ▶ Luokkia ja olioita ei sotketa samaan kuvaan, kyseessä onkin kaksi kuvaa: vasemmalla luokkakaavio ja oikealla oliokaavio
  - ▶ Oliokaavio kuvaa tietyn hetken tilanteen, olemassa 3 henkilöä, joista yksi on nimetön

Luokkakaavio



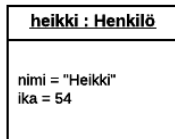
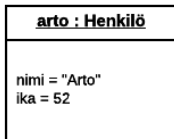
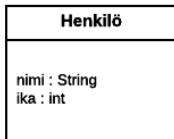
Eräs luokkakaaviota vastaava oliokaavio



# Luokka- ja oliokaaviot

## Attribuutit

- ▶ Luokan olioilla on attribuutteja eli oliomuuttujia ja operaatioita 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 tyyppinä 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)

Henkilö
nimi : String ikä : int
vanhene() meneToihin()

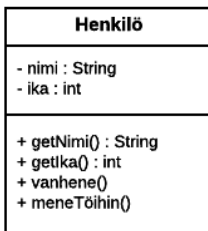
Tiedosto
nimi kokoTavuina muokkausAika
tulosta()

Kuvio
vari : Vari sijainti : Piste
kierra(kulma : double) siirra(suunta)

# Luokka- ja oliokaaviot

## 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



# Luokka- ja oliokaaviot

## VahenevaLaskuri esimerkki

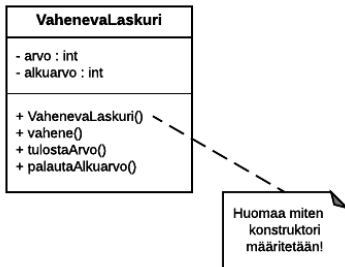
```
public class VahenevaLaskuri {  
    private int arvo;  
    private int alkuarvo;  
  
    public VahenevaLaskuri(int arvo) {  
        this.arvo = arvo;  
        this.alkuarvo = arvo;  
    }  
  
    public void vahene() {  
        if ( this.arvo>0 ) {  
            this.arvo--;  
        }  
    }  
  
    public void tulostaArvo() {  
        System.out.println(this.arvo);  
    }  
  
    public void palautaAlkuarvo() {  
        this.arvo = this.alkuarvo;  
    }  
}
```



# Luokka- ja oliokaaviot

## VahenevaLaskuri esimerkki

```
public class VahenevaLaskuri {  
    private int arvo;  
    private int alkuarvo;  
  
    public VahenevaLaskuri(int arvo) {  
        this.arvo = arvo;  
        this.alkuarvo = arvo;  
    }  
  
    public void vahene() {  
        if ( this.arvo>0 ) {  
            this.arvo--;  
        }  
    }  
  
    public void tulostaArvo() {  
        System.out.println(this.arvo);  
    }  
  
    public void palautaAlkuarvo() {  
        this.arvo = this.alkuarvo;  
    }  
}
```



*Kuvassa mukana  
UML-kommenttisyntaksi!*

# Luokka- ja oliokaaviot

## Yhteenveto

- ▶ Eli luokka on laatikko, jossa luokan nimi ja tarvittaessa attribuutit sekä metodit
- ▶ Attribuuttien ja metodien parametrien ja paluuarvon tyyppi ilmaistaan tarvittaessa
  - ▶ Näkyvyysmääreet ilmaistaan tarvittaessa
- ▶ Parametrin tyyppi voidaan myös merkitä "Javamaisesti", esim. Date aika tai int ikä
- ▶ Jos esim. metodeja ei haluta näyttää, jätetään metodiosa pois, vastaavasti voidaan menetellä attribuuttien suhteen

LuokanNimi
attribuutinNimi : attribuutinTyyppi attribuutinNimi2 : attribuutinTyyppi2
metodinNimi(parametrit) : paluuTyyppi

# Olioiden väliset yhteydet

## Yleisesti yhteyksistä

- ▶ 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
  - ▶ Kello *sisältää* kolme Viisaria
- ▶ Olioiden välinen yhteys voi olla pysyvämpiluontoinen (rakenteinen) tai hetkellinen
  - ▶ Aluksi keskitytään pysyvämpiluontoiisiin yhteyksiin

# Olioiden väliset yhteydet

## Yleisesti yhteyksistä

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

```
public class Auto {  
    public void aja() {  
        System.out.println("liikkuu");  
    }  
}  
  
public class Henkilö {  
    private Auto omaAuto;  
  
    public Henkilö(Auto auto) {  
        omaAuto = auto  
    }  
  
    public void meneTöihin() {  
        omaAuto.aja();  
    }  
}
```

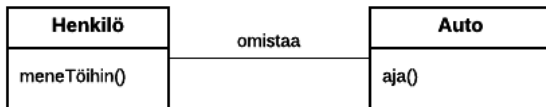
# Olioiden väliset yhteydet

Assosiaatio

# Olioiden väliset yhteydet

## Assosiaatio

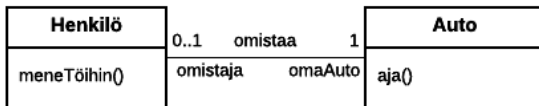
- ▶ Olioviite voitaisiin periaatteessa merkitä luokkakaavioon attribuuttina, kyseessä on teknisessä mielessä attribuutti
- ▶ Parempi tapa on kuvata olioiden välinen yhteys luokkakaaviossa
  - ▶ Jos Henkilö- ja Auto-olion välillä voi olla yhteys, yhdistetään Henkilö- ja Auto-luokat viivalla
- ▶ Tilanne kuvattu alla
  - ▶ Yhteydelle on annettu nimi omistaa, eli Henkilö-olio omistaa Auto-olion



# Olioiden väliset yhteydet

## Kytkenärajat ja roolit

- ▶ Ohjelmakoodissa jokaisella henkilöllä on täsmälleen yksi auto ja auto liittyy korkeintaan yhteen henkilöön
- ▶ Tämä kuvataan *kytkentärajotteina* (engl. multiplicity)
  - ▶ 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 oliomuuttuja, jonka tyyppinä Auto
- ▶ Henkilön rooli yhteydessä on olla omistaja



# Olioiden väliset yhteydet

## Kytkentärajoitteet ja roolit

- ▶ Edellisessä esimerkissä 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 \*

0	Ei yhtään (harvinainen!)
0..1	Ei yhtään tai yksi
1	Tasan yksi
0..*	Ei yhtään tai enemmän
*	Sama kuin 0..*
1..*	Yksi tai enemmän



# Olioiden väliset yhteydet

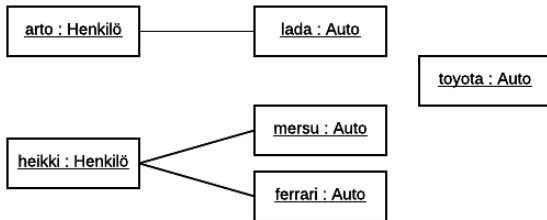
## Yhteydet oliokaaviossa

- ▶ 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 on tietyllä hetkellä olemassa ja miten ne yhdistyvät?

# Olioiden väliset yhteydet

## 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 on tietyllä hetkellä olemassa ja miten ne yhdistyvät?
- ▶ Alla tilanne, jossa Artolla on 1 auto ja Heikillä 3 autoa, yhdellä autolla ei ole omistajaa



# Olioiden väliset yhteydet

## Yhden suhde moneen -yhteyden toteuttaminen Javassa

- ▶ Jos henkilöllä on korkeintaan yksi auto, on Henkilö-luokalla siis attribuutti, jonka tyyppi on Auto
  - ▶ `private Auto omaAuto;`
- ▶ Jos henkilöllä on monta autoa, on Javassa yleinen ratkaisu lisätä Henkilö-luokalle attribuutiksi listallinen (esim. `ArrayList`) autoja:
  - ▶ `private ArrayList<Auto> omatAutot;`
- ▶ `ArrayList`istä tarkemmin Ohjelmoinnin perusteiden materiaalissa

# Olioiden väliset yhteydet

## Helppoja yhteysesimerkkejä

- ▶ Kuten niin moni asia UML:ssä, on myös yhteyden nimen ja roolinimien merkintä vapaaehtoista
- ▶ Jos kytkentärajoite jätetään merkitsemättä, niin silloin yhteydessä olevien olioiden lukumäärä on määrittelemätön
- ▶ **Seuraavaksi joukko esimerkkejä**

# Olioiden väliset yhteydet

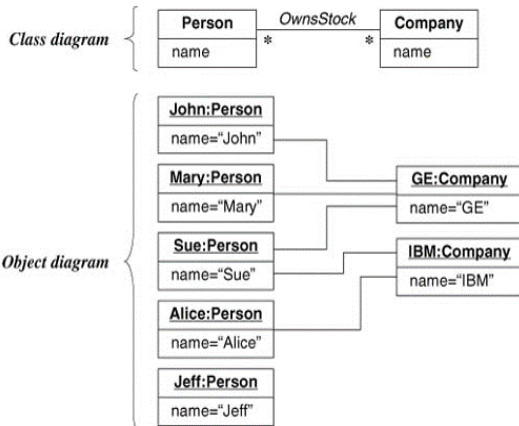
## Helppoja yhteysesimerkkejä

- ▶ 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

# Olioiden väliset yhteydet

## Helppoja yhteysesimerkkejä

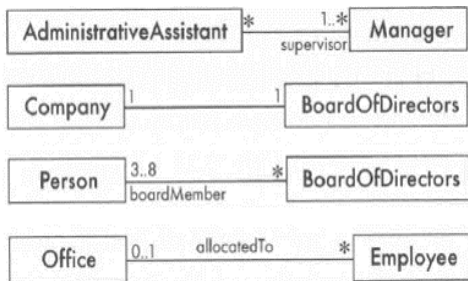
- ▶ Henkilö voi omistaa usean yhtiön osakkeita
- ▶ Yhtiöllä on monia osakkeenomistajia
- ▶ Eli yhteen Henkilö-olioon voi liittyä monta Yhtiö-olioa
- ▶ Ja yhteen Yhtiö-olioon voi liittyä monta Henkilö-olioa



# Olioiden väliset yhteydet

## Helppoja yhteysesimerkkejä

- ▶ 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 (allocatedTo) useita työntekijöitä. Työntekijällä on paikka yhdessä toimistossa tai ei missään



# Olioiden väliset yhteydet

## Yhteyden navigointisuunta

- Palautetaan mieleen Auto ja Henkilö -esimerkki

```
public class Auto {  
    public void aja() {  
        System.out.println("liikkuu");  
    }  
}  
  
public class Henkilö {  
    private Auto omaAuto;  
  
    public Henkilö(Auto auto) {  
        omaAuto = auto  
    }  
  
    public void meneTöihin() {  
        omaAuto.aja();  
    }  
}
```

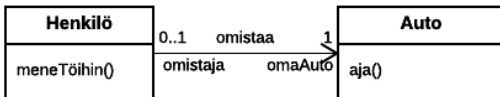
- Auto-luokan koodista huomaamme, että auto-oliot eivät tunne omistajaansa, Henkilö-oliot taas tuntevat omistamansa autot Auto-tyyppisen attribuutin omaAuto ansiosta
- Yhteys siis on oikeastaan yksisuuntainen, henkilöstä autoon



# Olioiden väliset yhteydet

## Yhteyden navigointisuunta

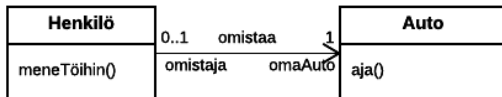
- ▶ Asia voidaan ilmaista kaaviossa tekemällä yhteysviivasta nuoli
  - ▶ Nuolen kärki sinne suuntaan, johon on pääsy oliomuuttujan avulla



# Olioiden väliset yhteydet

## Yhteyden navigointisuunta

- ▶ Asia voidaan ilmaista kaaviossa tekemällä yhteysviivasta nuoli
  - ▶ Nuolen kärki sinne suuntaan, johon on pääsy oliomuuttujan avulla



- ▶ Yhteyden navigointisuunnalla merkitystä lähinnä suunnittelu- ja toteutustason kaavioissa
  - ▶ Merkitään vain jos suunta tärkeä tietää
  - ▶ Joskus kaksisuuntaisuus merkitään nuolella molempiin suuntiin
  - ▶ Joskus taas nuoleton tarkoittaa kaksisuuntaista
- ▶ Määrittelytason luokkakaavioissa yhteyden suuntia ei yleensä merkitä ollenkaan
- ▶ Yhteyden suunnalla on aika suuri merkitys sille, kuinka yhteys toteutetaan kooditasolla

# Olioiden väliset yhteydet

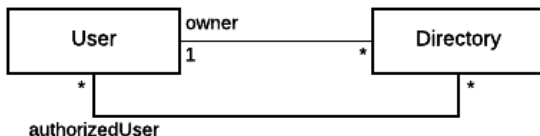
## Useampia yhteyksiä olioiden välillä

- ▶ Esim. Linuxissa jokaisella hakemistolla on tasan yksi omistaja
- ▶ Eli yhteen hakemisto-olioon liittyy roolissa owner tasan yksi käyttäjä-olio
- ▶ Jokaisella hakemistolla voi olla lisäksi useita käyttäjiä
  - ▶ Yhteen hakemistoon liittyy useita käyttäjiä roolissa `authorizedUser`
- ▶ Yksi käyttäjä voi omistaa useita hakemistoja
- ▶ Yhdellä käyttäjällä voi olla käyttöoikeus useisiin hakemistoihin
- ▶ Yhdellä käyttäjällä voi olla samaan hakemistoon sekä omistus- että käyttöoikeus

# Olioiden väliset yhteydet

## Useampia yhteyksiä olioiden välillä

- ▶ Esim. Linuxissa jokaisella hakemistolla on tasan yksi omistaja
- ▶ Eli yhteen hakemisto-olioon liittyy roolissa owner tasan yksi käyttäjä-olio
- ▶ Jokaisella hakemistolla voi olla lisäksi useita käyttäjiä
  - ▶ Yhteen hakemistoon liittyy useita käyttäjiä roolissa `authorizedUser`
- ▶ Yksi käyttäjä voi omistaa useita hakemistoja
- ▶ Yhdellä käyttäjällä voi olla käyttöoikeus useisiin hakemistoihin
- ▶ Yhdellä käyttäjällä voi olla samaan hakemistoon sekä omistus- että käyttöoikeus



# Olioiden väliset yhteydet

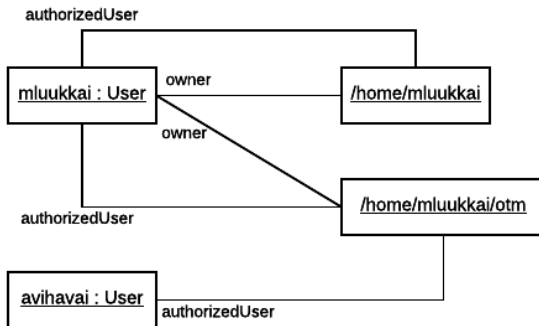
## Useampia yhteyksiä olioiden välillä

- ▶ Ohessa oliokaavio, joka kuvaa erään edellisen luokkakaavion mukaisen tilanteen
  - ▶ Käyttäjät: mluukkai ja avihavai
  - ▶ mluukkai omistaa kaksi hakemistoa
  - ▶ mluukkai omistaa kaksi hakemistoa
    - ▶ Samojen olioiden välillä kaksi eri yhteyttä!
  - ▶ avihavai:lla käyttöoikeus hakemistoon `/home/mluukkai/otm`

# Olioiden väliset yhteydet

## Useampia yhteyksiä olioiden välillä

- ▶ Ohessa oliokaavio, joka kuvaa erään edellisen luokkakaavion mukaisen tilanteen
  - ▶ Käyttäjät: mluukkai ja avihavai
  - ▶ mluukkai omistaa kaksi hakemistoa
  - ▶ mluukkai omistaa kaksi hakemistoa
    - ▶ Samojen olioiden välillä kaksi eri yhteyttä!
  - ▶ avihavai:lla käyttöoikeus hakemistoon /home/mluukkai/otm



# Olioiden väliset yhteydet

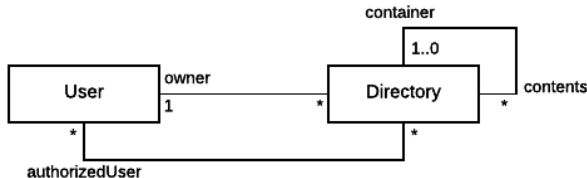
## Yhteys kahden saman luokan olion välillä

- ▶ Miten mallinnetaan se, että hakemistolla on alihakemistoja?
  - ▶ Hakemisto sisältää alihakemistoja
  - ▶ Hakemisto sisältyy johonkin toiseen hakemistoon
- ▶ Yhteen hakemisto-olioon voi liittyä 0 tai 1 hakemisto-olioa roolissa *container* (=sisältäjä), eli hakemisto voi olla jonkun toisen hakemiston alla
- ▶ Yhteen hakemisto-olioon voi liittyä mielivaltaisen määrä (\*) hakemisto-olioita roolissa *contents* (=sisältö), eli hakemisto voi sisältää muita hakemistoja

# Olioiden väliset yhteydet

## Yhteys kahden saman luokan olion välillä

- ▶ Miten mallinnetaan se, että hakemistolla on alihakemistoja?
  - ▶ Hakemisto sisältää alihakemistoja
  - ▶ Hakemisto sisältyy johonkin toiseen hakemistoon
- ▶ Yhteen hakemisto-olioon voi liittyä 0 tai 1 hakemisto-olioa roolissa *container* (=sisältäjä), eli hakemisto voi olla jonkun toisen hakemiston alla
- ▶ Yhteen hakemisto-olioon voi liittyä mielivaltainen määrä (\*) hakemisto-olioita roolissa *contents* (=sisältö), eli hakemisto voi sisältää muita hakemistoja





# Olioiden väliset yhteydet

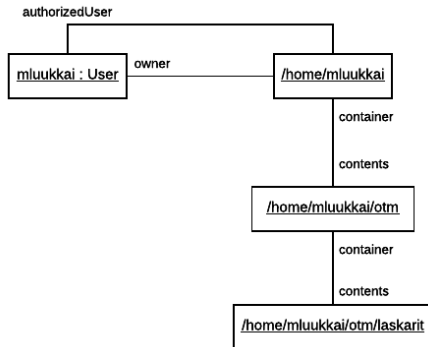
## Yhteys kahden saman luokan olion välillä

- ▶ Tilanne vaikuttaa hieman sekavalta, selvennetään oliokaavion avulla
- ▶ /home/mluukkai hakemiston
  - ▶ Alihakemistojen rooli yhteydessä on *contents* eli sisältö
  - ▶ Päähakemiston rooli yhteydessä on *container* eli sisältäjä
- ▶ /home/mluukkai/otm on edellisen alihakemisto, mutta sisältää itse alihakemiston

# Olioiden väliset yhteydet

## Yhteys kahden saman luokan olion välillä

- ▶ Tilanne vaikuttaa hieman sekavalta, selvennetään oliokaavion avulla
- ▶ /home/mluukkai hakemiston
  - ▶ Alihakemistojen rooli yhteydessä on *contents* eli sisältö
  - ▶ Päähakemiston rooli yhteydessä on *container* eli sisältäjä
- ▶ /home/mluukkai/otm on edellisen alihakemisto, mutta sisältää itse alihakemiston



# Olioiden väliset yhteydet

Monimutkaisempi esimerkki assosiaatioista

## ► Mallinnetaan seuraava tilanne:

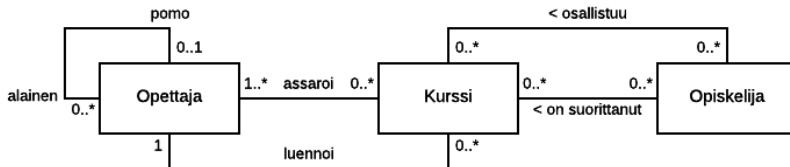
- Kurssilla on luennoijana 1 opettaja ja assarina useita opettajia
- Opettaja voi olla useiden kurssien assarina ja luennoijana
- Opettajalla voi olla pomo ja useita alaisia
- Opettajat johtavat toisiaan
- Opiskelija voi osallistua useille kursseille
- Opiskelijalla voi olla suorituksia useista kursseista

# Olioiden väliset yhteydet

## Monimutkaisempi esimerkki assosiaatioista

### ► Mallinnetaan seuraava tilanne:

- Kurssilla on luennoijana 1 opettaja ja assarina useita opettajia
- Opettaja voi olla useiden kurssien assarina ja luennoijana
- Opettajalla voi olla pomo ja useita alaisia
- Opettajat johtavat toisiaan
- Opiskelija voi osallistua useille kursseille
- Opiskelijalla voi olla suorituksia useista kursseista



# Olioiden väliset yhteydet

## Riippuvuus

# Olioiden väliset yhteydet

## Riippuvuus

- ▶ Muistellaan taas Auto-esimerkkiä:
- ▶ On olemassa henkilöitä, jotka eivät omista autoa
- ▶ Autottomat henkilöt kuitenkin välillä lainaavat jonkun muun autoa

- ▶ Koodissa asia voitaisiin ilmaista seuraavasti:

```
public class AutotonHenkilo {  
    public void lainaaJaAja( Auto lainaAuto ) {  
        lainaAuto.aja();  
    }  
}
```

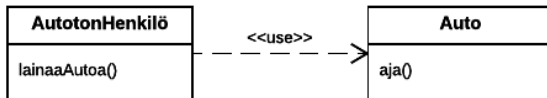
- ▶ Eli autottoman henkilön metodi lainaaJaAja saa parametrikseen auton (lainaAuto), jolla henkilö ajaa
- ▶ Auto on lainassa ainoastaan metodin suoritusajan
  - ▶ AutotonHenkilo ei siis omaa pysyvää suhdetta autoon

# Olioiden väliset yhteydet

## Riippuvuus

Riippuvuus on tavallaan myös yhteys, mutta "normaalia" yhteyttä "heikompi" (koska se ei kestä yhtä kauaa)

- ▶ Koska kyseessä ei ole pysyvämpiluontoinen yhteys, on parempi käyttää luokkakaaviossa *riippuvuussuhdetta* (engl. dependency)
- ▶ Riippuvuus merkitään katkoviivanuolena, joka osoittaa siihen luokkaan josta ollaan riippuvaisia
- ▶ Alla on ilmaistu vielä riippuvuuden laatu
  - ▶ Tarkennin (eli *stereotyyppi*) «use» kertoo että kyseessä on käyttöriippuvuus, eli AutotonHenkilö kutsuu Auto:n metodia



# Olioiden väliset yhteydet

## Riippuvuus

- ▶ Joskus riippuvuus määritellään siten, että luokka A on riippuvainen luokasta B jos muutos B:hen saa aikaan mahdollisesti muutostarpeen A:ssa
  - ▶ Näin on edellisessä esimerkissä: jos Auto-luokka muuttuu (esim. metodi aja muuttuu siten että se tarvitsee parametrin), joudutaan AutotonHenkilö-luokkaa muuttamaan
- ▶ Riippuvuus on siis jotain *heikompaa* kun tavallinen luokkien välinen yhteys
  - ▶ Jos luokkien välillä on yhteys, on niiden välillä myös riippuvuus, sitä ei vaan ole tapana merkitä (Henkilö joka omistaa Auton on riippuvainen autosta)

Huomaathan, että toisin kuin yhteyksiin, **riippuvuuksiin ei merkitä kytkentärajoitteita**



# Olioiden väliset yhteydet

## Kompositio

```
public class Auto {
    private final Rengas vEtu;
    private final Rengas oEtu;
    private final Rengas vTaka;
    private final Rengas oTaka;

    public Auto(){
        vEtu = new Rengas();
        oEtu = new Rengas();
        vTaka = new Rengas();
        oTaka = new Rengas();
    }

    public void aja() {
        vEtu.pyöri();
        oEtu.pyöri();
        vTaka.pyöri();
        oTaka.pyöri();
    }
}

class Rengas{
    public void pyöri(){
        System.out.println("pyörii");
    }
}
```

- ▶ Auto sisältää 4 rengasta
- ▶ Renkaat luodaan auton konstruktorissa
- ▶ Renkaiseen ei pääse auton ulkopuolelta käsiksi

# Olioiden väliset yhteydet

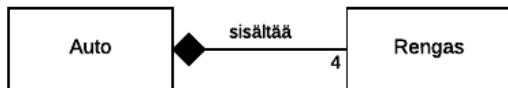
## Kompositio

- ▶ Autossa on 4 pyörää, joten tilannehan voitaisiin mallintaa tekemällä autosta yhteys Rengas-olioon ja laittamalla kytkentärajoitteeksi 4
- ▶ Renkaat ovat kuitenkin siinä mielessä erityisessä asemassa, että voidaan ajatella, että ne ovat auton komponentteja
  - ▶ Renkaat sisältyvät autoon
- ▶ Kun auto luodaan, luodaan renkaat samalla
  - ▶ Koodissa auto luo renkaat
- ▶ Renkaat ovat private, eli niihin ei pääse ulkopuolelta käsiksi
- ▶ Kun roskienkerääjä tuhoaa auton, tuhoutuvat myös renkaat
- ▶ **Eli ohjelman renkaat sisältyvät autoon ja niiden elinikä on sidottu auton elinikään** (oikeat renkaat eivät tietenkään käyttäydy näin vaan ovat vaihdettavissa)

# Olioiden väliset yhteydet

## Kompositio

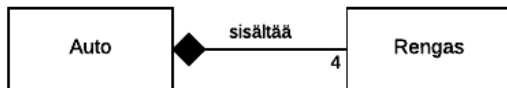
- ▶ Edellisen kalvon tilannetta nimitetään **kompositioksi** (engl. *composition*)
- ▶ Komposition symboli on "musta salmiakkikuvio", joka liitetään yhteyden siihen päähän, johon osat sisältyvät



# Olioiden väliset yhteydet

## Kompositio

- ▶ Edellisen kalvon tilannetta nimitetään **kompositioksi** (engl. *composition*)
- ▶ Komposition symboli on "musta salmiakkikuvio", joka liitetään yhteyden siihen päähän, johon osat sisältyvät

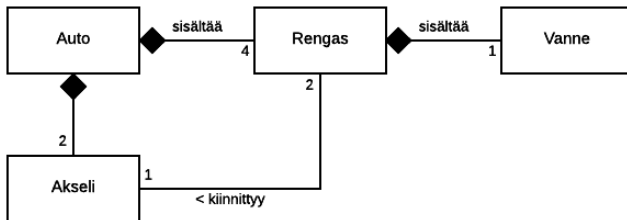


- ▶ Kompositiota käytetään kun seuraavat ehdot toteutuvat:
  1. Osat ovat olemassaoloriippuvaisia kokonaisuudesta
  2. Osa voi kuulua vaan yhteen kompositioon
    - ▶ Rengasta ei voi siirtää toiseen autoon
  3. Osa on koko elinaikansa kytketty samaan kompositioon
- ▶ Koska Rengas-olio voi liittyä nyt vain yhteen Auto-olioon, ei salmiakin puoleiseen päähän tarvita osallistumisrajoitetta, koska se on joka tapauksessa 1

# Olioiden väliset yhteydet

## Monimutkaisempi esimerkki kompositiosta

- ▶ Tarkennettu Auto sisältää 4 rengasta ja 2 akselia
- ▶ Komposition osa voi myös sisältää oliota
  - ▶ Rengas sisältää vanteen
- ▶ Komposition osilla voi olla "normaaleja" yhteyksiä
  - ▶ Akseli kiinnittyy kahteen renkaaseen
  - ▶ Rengas on kiinnittynyt yhteen akseliin



# Olioiden väliset yhteydet

## Monimutkaisempi esimerkki kompositiosta

- ▶ Onko kompositiomerkkiä pakko käyttää?
  - ▶ Ei, mutta usein sen käyttö selkiyttää tilannetta
- ▶ Kompositiota ei kannata laittaa sinne minne se ei kuulu!
  - ▶ Kompositio on erittäin rajoittava suhde olioiden välillä, toisin kuin "normaali" yhteys, käytä kompositiota vasta kun olet tarkistanut onko kaikki sen ehdot täyttyneet
- ▶ **HUOM:** auton ja renkaat sisältävä esimerkki kuvaa vain esimerkikoodin tilannetta, mutta ei tietenkään ole realistinen kuvaamaan reaali maailman autojen ja renkaiden suhdetta sillä normaalistihan renkaat eivät ole autoista olemassaoloriippuvaisia

# Olioiden väliset yhteydet

Kooste

# Olioiden väliset yhteydet

## Kooste

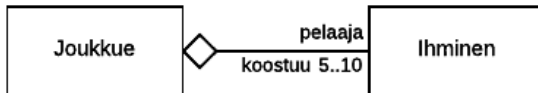
- ▶ **Koosteella** (engl. aggregation) tarkoitetaan koostumussuhdetta, joka ei ole yhtä komposition tapaan "ikuinen"
  - ▶ **HUOM:** suomenkieliset termit kooste ja kompositio ovat huonot ja jopa harhaanjohtavat
- ▶ Koostetta merkitään "valkoisella salmiakilla" joka tulee siihen päähän yhteyttä, johon osat kuuluvat
- ▶ Esimerkki: Joukkue koostuu pelaajista (jotka ovat ihmisiä)
  - ▶ Ihminen ei kuitenkaan kuulu joukkueeseen ikuisesti
  - ▶ Joukkue ei synnytä eikä tapa pelaajaa
  - ▶ Ihminen voi kuulua yhtä aikaa useampaan joukkueeseen



# Olioiden väliset yhteydet

## Kooste

- ▶ **Koosteella** (engl. aggregation) tarkoitetaan koostumussuhdetta, joka ei ole yhtä komposition tapaan "ikuinen"
  - ▶ **HUOM:** suomenkieliset termit kooste ja kompositio ovat huonot ja jopa harhaanjohtavat
- ▶ Koostetta merkitään "valkoisella salmiakilla" joka tulee siihen päähän yhteyttä, johon osat kuuluvat
- ▶ Esimerkki: Joukkue koostuu pelaajista (jotka ovat ihmisiä)
  - ▶ Ihminen ei kuitenkaan kuulu joukkueeseen ikuisesti
  - ▶ Joukkue ei synnytä eikä tapa pelaajaa
  - ▶ Ihminen voi kuulua yhtä aikaa useampaan joukkueeseen



# Olioiden väliset yhteydet

## Kooste

- ▶ Komposition (eli mustan salmiakin) merkitys on selkeä, kyseessä on olemassaoloriippuvuus
- ▶ On sensijaan epäselvää milloin koostetta (eli valkoista salmiakkia) tulisi käyttää normaalin yhteyden sijaan
- ▶ Monet asiantuntevat oliomallintajat ovat sitä mieltä että koostetta ei edes tulisi käyttää
- ▶ Koostesuhde on poistunut UML:n standardista versiosta 2.0 lähtien
- ▶ Koostesuhde on kuitenkin edelleen erittäin paljon käytetty joten on hyvä tuntea symboli passiivisesti
- ▶ **Tällä kurssilla koostetta ei käytetä eikä sitä tarvitse osata!**
- ▶ Joukkueen ja pelaajien välinen suhde voidaankin ilmaista normaalina yhteytenä

# Olioiden väliset yhteydet

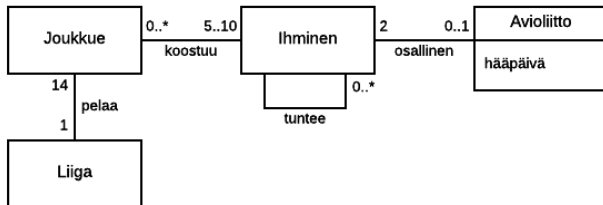
## Monimutkaisempi esimerkki Koosteen kierrosta

- ▶ Joukkue pelaa liigassa, jossa on 14 joukkuetta
- ▶ Ihminen voi kuulua mielivaltaisen moneen joukkueeseen
- ▶ Joukkueeseen kuuluu 5-10 ihmistä
- ▶ Ihminen tuntee useita ihmisiä
- ▶ Ihminen voi olla avioliitossa, mutta vain yhdessä avioliitossa kerrallaan
- ▶ Avioliitto koostuu kahdesta ihmisestä

# Olioiden väliset yhteydet

## Monimutkaisempi esimerkki Koosteen kierrosta

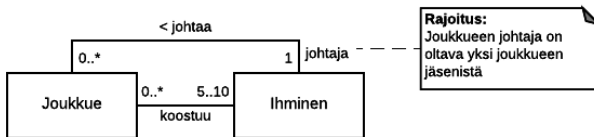
- ▶ Joukkue pelaa liigassa, jossa on 14 joukkuetta
- ▶ Ihminen voi kuulua mielivaltaisen moneen joukkueeseen
- ▶ Joukkueeseen kuuluu 5-10 ihmistä
- ▶ Ihminen tuntee useita ihmisiä
- ▶ Ihminen voi olla avioliitossa, mutta vain yhdessä avioliitossa kerrallaan
- ▶ Avioliitto koostuu kahdesta ihmisestä



# Olioiden väliset yhteydet

## Rajoitukset

- ▶ Jos haluttaisiin mallintaa tilanne, että joku joukkueen jäsenistä on joukkueen johtaja, pelkkä luokkakaavio (siinä määrin kun tällä kurssilla UML:ää opitaan) ei riitä
- ▶ Tilanne voitaisiin mallintaa alla esitetyllä tavalla
  - ▶ Eli lisätään normaali yhteys *johtaa* joukkueen ja ihmisen välille
  - ▶ Määritellään kytkentärajoite: joukkueella on tasan 1 johtaja
  - ▶ Ilmaistaan UML-kommenttina, että joukkueen johtajan on oltava joku joukkueen jäsenistä



# Olioiden väliset yhteydet

## Rajapinnat

# Olioiden väliset yhteydet

## Rajapinnat

- ▶ Javan rajapinta (engl. *interface*) määrittelee joukon metodeja, jotka rajapinnan toteuttavan luokan on toteutettava
- ▶ Rajapintaluokka siis (yleensä) listaa ainoastaan joukon metodien nimiä
  - ▶ Java 8 on tuonut tähän sen poikkeuksen, että rajapintojen metodeita voi olla oletustoteutuksia
- ▶ Yksi luokka voi toteuttaa useita rajapintoja
  - ▶ Jos luokka toteuttaa rajapinnan, sen täytyy toteuttaa kaikki rajapinnan määrittelemät metodit (paitsi ne joilla on oletustoteutus)

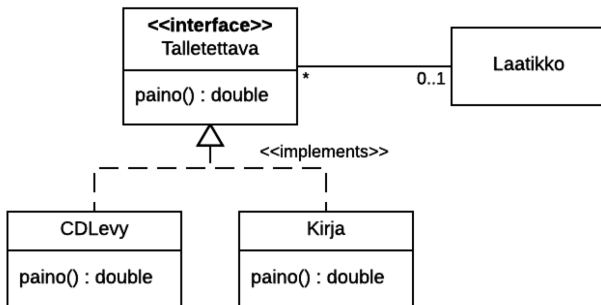
Rajapinta on sopimus, jonka toteuttaja lupaa toteuttaa ainakin rajapinnan määrittelemät metodit.

- ▶ Seuraavalla Ohjelmoinnin jatkokurssin toisen viikon tehtävä 153 Tavarointa ja Laatikoita

# Olioiden väliset yhteydet

## Rajapinta UML-kaaviossa

- ▶ Rajapinta kuvataan luokkana, johon liitetään tarkenne `<<interface>>`
  - ▶ Rajapinnalle on merkitty metodi *paino*, jonka se määrittelee
- ▶ Rajapinnan toteuttaminen merkitään katkoviivana, jonka päässä on "valkoinen" kolmio
  - ▶ Voidaan tarkentaa tarkenteella `<<implements>>`
- ▶ Rajapinnan toteuttaville luokille on merkitty myös metodi *paino*, sillä molemmat niistä toteuttavat sen omalla tavallaan





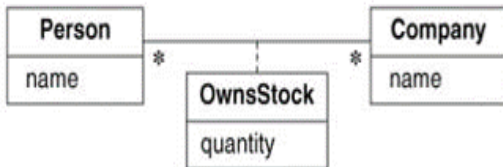
# Yhteyteen liittyvät tiedot

Yhteyden tietojen mallinnus

# Yhteyteen liittyvät tiedot

## Yhteyden tietojen mallinnus

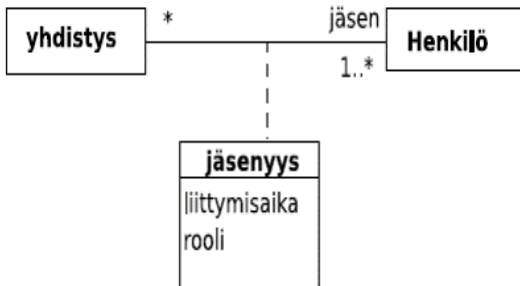
- ▶ Yhteyteen voi joskus liittyä myös tietoa
- ▶ Esim. tilanne missä henkilö voi olla (usean) yhtiön osakkeenomistaja
  - ▶ Osakkeenomistuksen kannalta tärkeä asia on omistettujen osakkeiden määrä
- ▶ Yksi tapa mallintaa tilanne on käyttää yhteysluokkaa (engl. association class), eli yhteyteen liittyvää luokkaa, joka sisältää esim. yhteyteen liittyviä tietoja
- ▶ Alla yhteysluokka sisältää omistettujen osakkeiden määrän



# Yhteyteen liittyvät tiedot

## Yhteyden tietojen mallinnus

- ▶ Luentomonisteessa mallinnetaan tilanne, jossa henkilö voi olla jäsenenä useassa yhdistyksessä
  - ▶ yhdistyksessä on vähintään 1 jäsen
- ▶ Jäsenyys kuvataan yhteytenä, johon liittyy yhteysluokka
  - ▶ jäsenyyden alkaminen (liittymisaika) sekä jäsenyyden tyyppi (rooli, eli onko rivijäsen, puheenjohtaja tms...) kuvataan yhteysluokan avulla



# Yhteyteen liittyvät tiedot

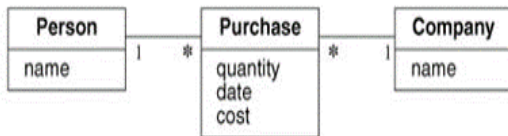
## Kannattaako yhteysluokkia käyttää?

- ▶ **Kannattaako yhteysluokkia käyttää?**
  - ▶ Korkean tason abstrakteissa malleissa ehkä
  - ▶ Suunnittelutason malleissa todennäköisesti ei, sillä ei ole selvää, mitä yhteysluokka tarkoittaa toteutuksen tasolla
- ▶ Yhteysluokan voi aina muuttaa tavalliseksi luokaksi
- ▶ Yhteysluokka joudutaankin käytännössä aina ohjelmoidessa toteuttamaan omana luokkana, joka yhdistää alkuperäiset luokat joiden välillä yhteys on
  - ▶ Tämän takia yhteysluokkia ei välttämättä kannata käyttää alunperinkään

# Yhteyteen liittyvät tiedot

## Yhteysluokasta normaaliksi luokaksi

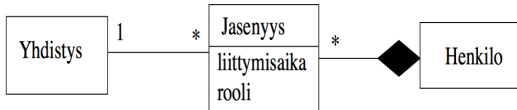
- ▶ Alla muutaman dian takainen osake-esimerkki
  - ▶ Nyt ilman yhteysluokkaa
- ▶ Henkilöllä on useita ostoksia (purchase)
- ▶ Ostokseen liittyy määrä (kuinka monesta osakkeesta kyse), päiväys ja hinta
- ▶ Yksi ostos taas liittyy tasan yhteen yhtiöön ja tasan yhteen henkilöön
- ▶ Henkilö ei ole enää suorassa yhteydessä firmaan
  - ▶ Henkilö "tuntee" kuitenkin omistamansa firmat ostosolioiden kautta



# Yhteyteen liittyvät tiedot

## Yhteysluokasta normaaliksi luokaksi

- ▶ Henkilön jäsenyys yhdistyksessä on myös luontevaa kuvata yhteysluokan sijaan omana luokkanaan:



- ▶ Jäsenyyden olemassaoloriippuvuus on nyt merkitty henkilöön
- ▶ Miksi näin? Miksei yhdistykseen tai peräti molempiin?
- ▶ Periaatteessa loogisinta olisi tehdä jäsenyydestä olemassaoloriippuvainen sekä yhdistyksestä että henkilöstä, mutta UML ei salli tätä (Sääntö 2: Osa voi kuulua vaan yhteen kompositioon)
- ▶ Olemassaoloriippuvuus saa siis olla vain yhteen olioon ja hetken mietinnän jälkeen on päätetty valita Henkilö jäsenyyden "omistavaksi" osapuoleksi, periaatteessa Yhdistys olisi ollut yhtä hyvä valinta

# Työkaluja piirtoon

Ohjelmia ja nettisivuja

# Työkaluja piirtoon

Ohjelmia ja nettisivuja

## Millä kaaviot kannattaa piirtää?

- ▶ Kynä ja paperia tai valkotaulu
- ▶ Ilmaisia työkaluja
  - ▶ Dia (win+linux)
  - ▶ Umbrello
  - ▶ ArgoUML
  - ▶ Openoffice
- ▶ Maksullisia työkaluja:
  - ▶ Visual Paradigm
  - ▶ Magic Draw
  - ▶ Microsoft Visio
  - ▶ Omnigraffle (Mac)



# Työkaluja piirtoon

Ohjelmia ja nettisivuja

## Millä kaaviot kannattaa piirtää?

- ▶ Mahdollisuuksia myös netissä:
  - ▶ <http://yuml.me/> luokka- ja käyttötapauskaavioihin
  - ▶ <https://www.websequencediagrams.com/> Sekvenssikaavioihin
  - ▶ <https://www.draw.io/> Jokapaikan höylä

Ei siis ole olemassa selkeää vastausta mitä työkalua kannattaa käyttää. Tämän kurssin tarpeisiin kynä ja paperia riittää hyvin