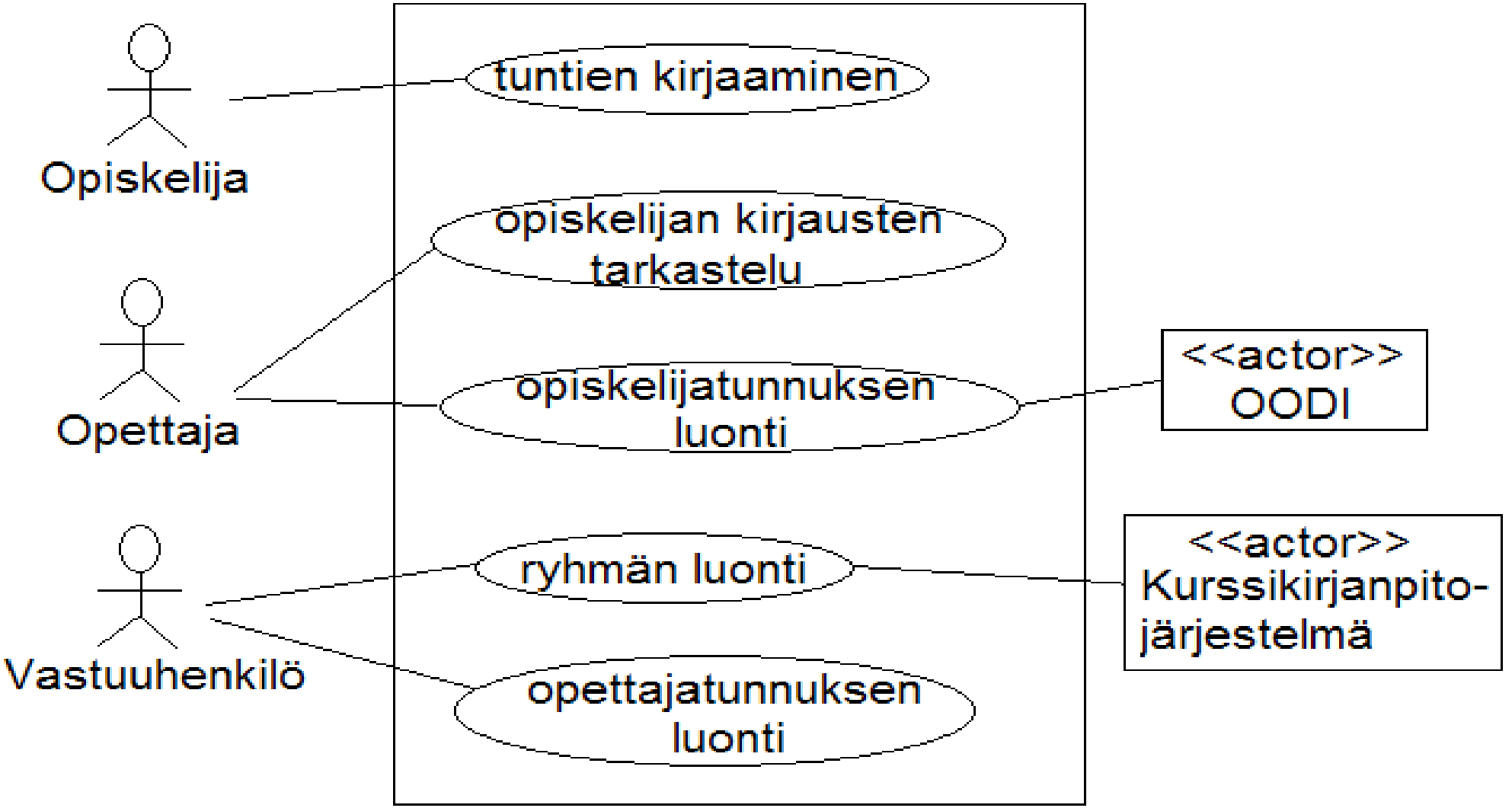
Ohjelmistojen mallintaminen

Luento 5, 16.11.

Muutama huomio/muistutus käyttötapauksista

- *Mikä/kuka on käyttäjä (engl. actor)?*
  - *Henkilö, toinen järjestelmä, laite yms. taho, joka on järjestelmän ulkopuolella, mutta tekemisissä järjestelmän kanssa*
- Käyttäjä siis ei ole välttämättä ihminen
  - Esim. lh2:n ensimmäisessä tehtävässä OODI ja kurssihallintajärjestelmä ovat käyttäjiä (ks. seuraava kalvo)
- Sana actor olisi ehkä parempi kääntää *toimijaksi*, joka sekään ei ole optimaalinen, sillä ”käyttäjä” voi myös olla passiivinen osapuoli käyttötapauksen suhteen (esim. OODI lh2:ssa)
- Käyttäjä on oikeastaan rooli
  - Yksi ihminen voi toimia useassa käyttäjäroolissa
  - Lh2: sama henkilö voi olla samaan aikaan vastuuhenkilö ja ”normaali” opettaja
- *Käyttötapausmallissa ei tule mallintaa mitään järjestelmän sisällä olevaa*, esim. tietokantaa tms.
- Käyttötapauskaavioiden ”nuolityypit”: include, extend, yleistys
  - Milloin ja miten mitäkin käytetään

Harjoitustyön tuntikirjanpitojärjestelmän käyttötapauskaavio

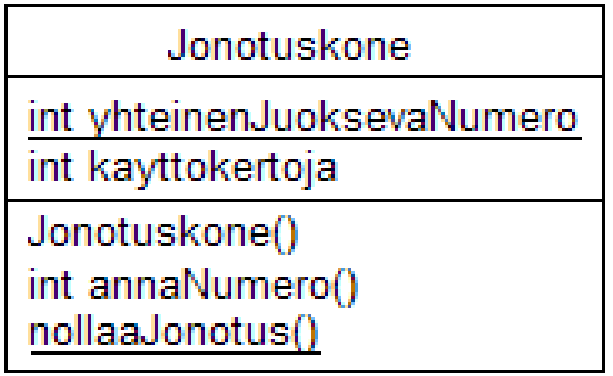


- Katso tarkemmin kurssin sivulta löytyvästä esimerkkivastauksesta

Luokkakohtaiset eli stattiset metodit ja attribuutit

- Ilmaistaan luokkakaaviossa alleviivattuina

```
public class Jonotuskone {
   private static int yhteinenJuoksevaNumero = 0;
   private int käyttökertoja;

   public static void nollaaJonotus() {   yhteinenJuoksevaNumero = 0;  }
   public Jonotuskone() {   käyttökertoja = 0;  }
   public int annaNumero() {
     ++käyttökertoja;
     return ++yhteinenJuoksevaNumero;
   }
}
```



Pääohjelma ja luokkakaavio

- Ohjelmoinnin perusteiden viikon 4 tehtävän 4 ratkaisu näyttää todennäköisesti seuraavalta:

```
public class Paaohjelma {

    public static void main(String[] args){

        Viljavarasto viljat = new Viljavarasto(10,15);

        while ( true ) {

            System.out.print(”Viljamestari, valitse toiminto: ”);

            String komento = lukija.nextLine();

            // suoritetaan valittu komento

        }

    }

}
```

- Miten tilanne tulisi mallintaa luokkakaaviona?

Pääohjelma ja luokkakaavio

- Jos haluaa käyttää UML:ää oikeaoppisesti, ei pääohjelman ja Viljavaraston suhdetta tule kuvata yhteytenä kahdesta syystä:
  - Yhteys luokan A ja luokan B välillä tarkoittaa että luokkien *olioiden* välillä on yhteys:
    - Tietty henkilöolio omistaa tietyn auto-olion
    - Tietty opiskelijaolio osallistuu tietylle kurssille
  - Viite viljavarastoon on ainoastaan metodin main() sisällä, ei luokassa Pääohjelma itsessään
- Oikeaoppinen tapa lienee kuvata Paaohjelman ja Viljavaraston suhde riippuvuutena:

- Paaohjelman ja Viljavaraston suhteen merkitseminen normaaliksi yhteydeksi on kuitenkin suhteellisen pieni rike

Miten viljamestari ohjelmoitaisiin ”oikeasti”

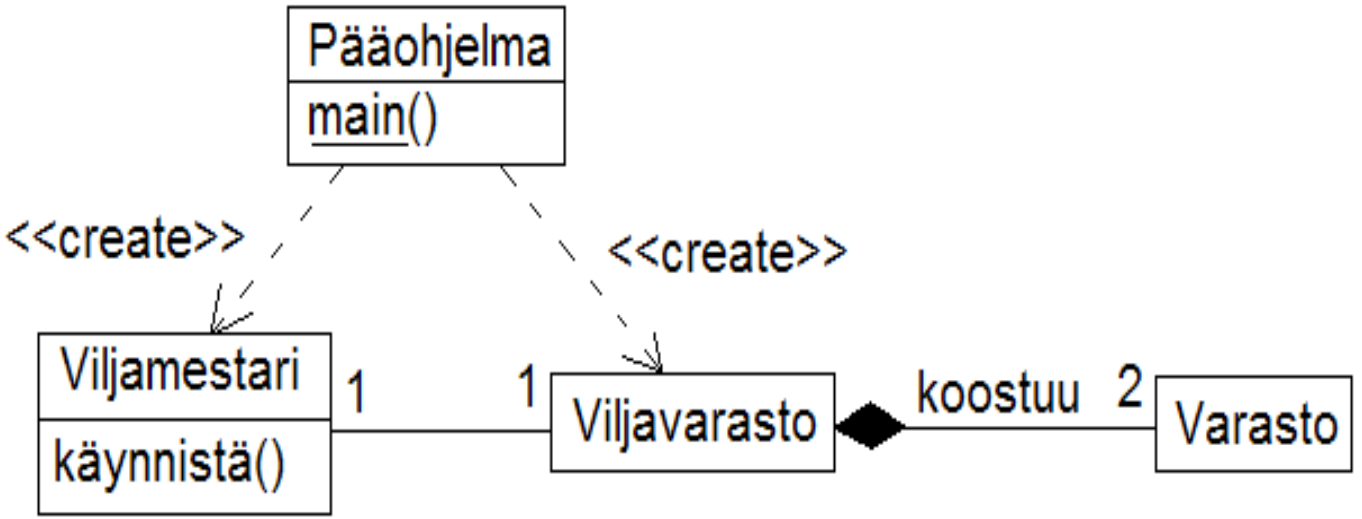
- Oikeissa ohjelmissa main ei yleensä tee mitään muuta kuin luo joukon olioita ja käynnistää sovelluksen joka on itsekin olio
- ”tosimaailman” Viljamestarisovellus voitaisiin tehdä esim. seuraavasti

```
public class Paaohjelma {

    public static void main(String[] args){

        Viljavarasto viljat = new Viljavarasto(10,15);

        Viljamestari viljamestari = new Viljamestari( viljat );

        viljamestari.käynnistä();

    }

}
```

- Eli pääohjelma luo viljavaraston ja sovelluksen ytimenä olevan viljamestarin joka saa käyttöönsä luodun viljavaraston
- Lopuksi main ”käynnistää” sovelluksen

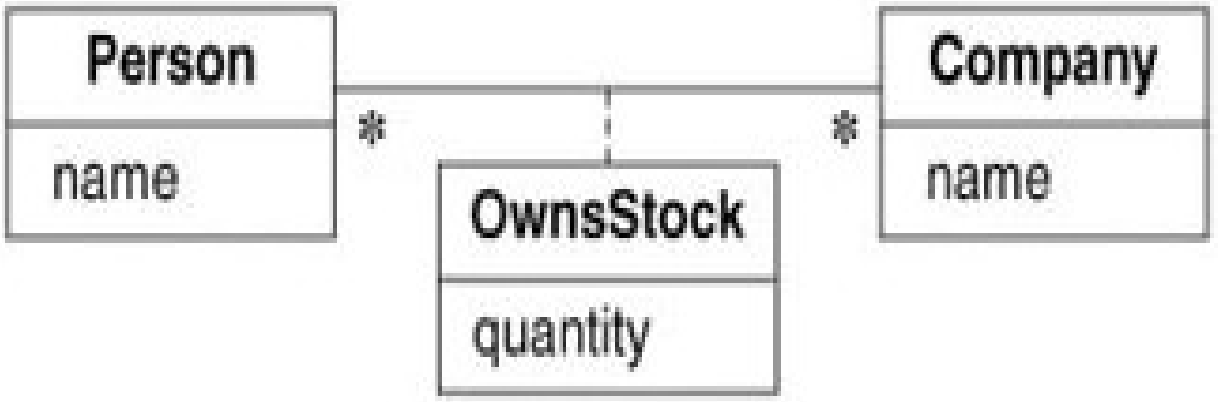
- Viljamestari-olio siis ottaa sen roolin mikä pääohjelmalla on mainiin tehdyissä ohjelmissa
- Alla luokkakaavio johon on merkitty mukaan myös main. Yleensä Pääohjelma jätetään merkitsemättä jos sen rooli on pelkkä olioiden luominen

```
public class Viljamestari {

    Viljavarasto viljavarasto;

    public Viljamestari(Viljavarasto v) { viljavarasto = v; }

    public void kaynnistä(){

        while ( true ) {

            System.out.print(”Viljamestari, valitse toiminto: ”);

            String komento = lukija.nextLine();

            // suoritetaan valittu komento

        }

    }

}
```



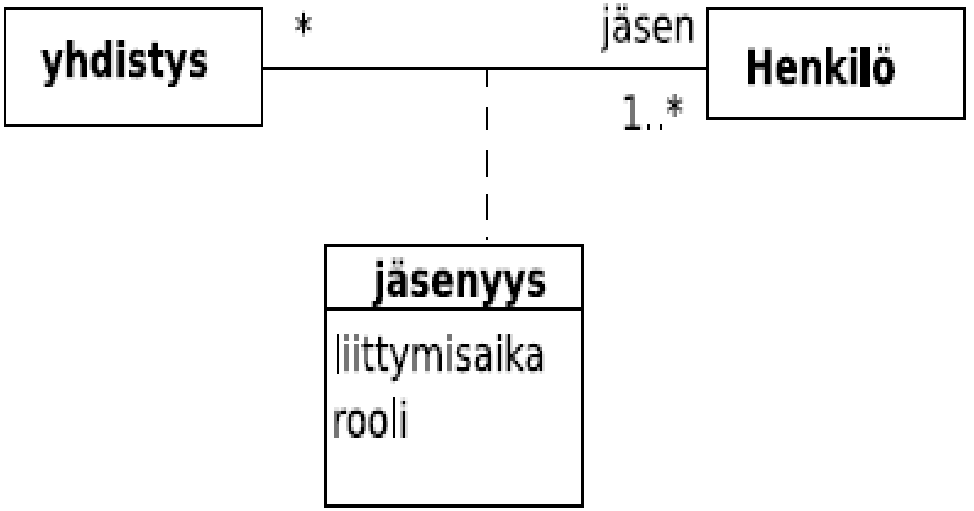
Yhteyden tietojen mallinnus

- 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



Toinen yhteysluokkaesimerkki

- 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

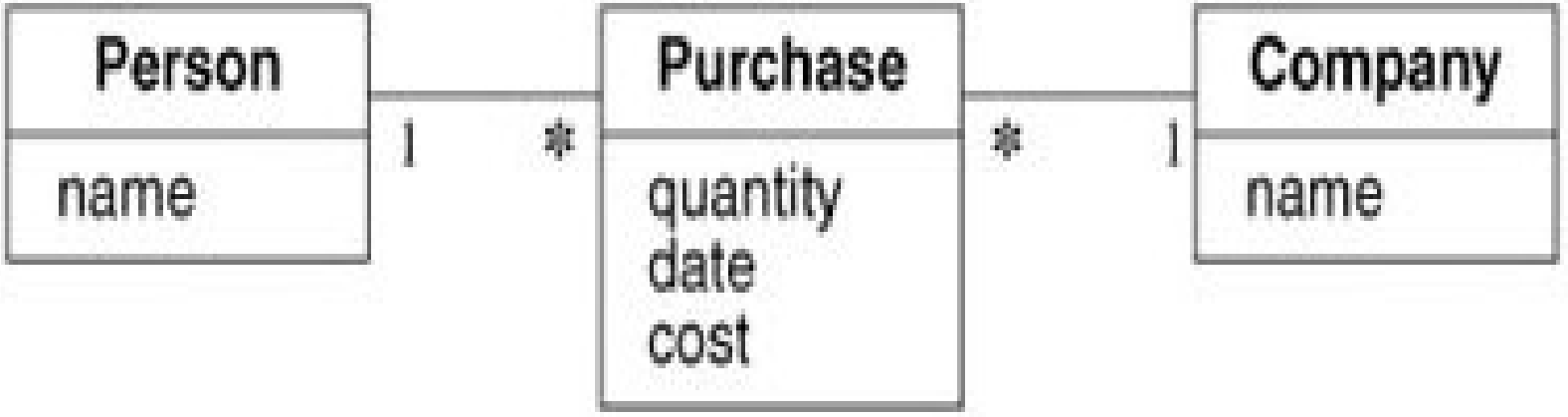


Kannattaako yhteysluokkia käyttää?

- 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 luokkanaan, joka yhdistää alkuperäiset luokat joiden välillä yhteys on
  - Tämän takia yhteysluokkia ei välttämättä kannata käyttää alunperinkään

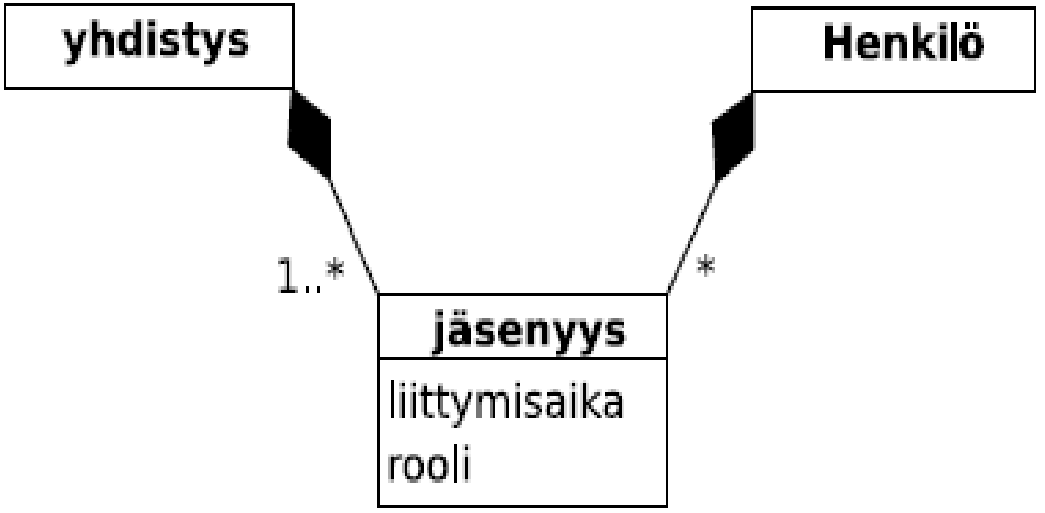
Yhteysluokasta normaaliksi luokaksi

- Alla muutaman sivun takainen osake-esimerkki 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 ostos-olioiden kautta



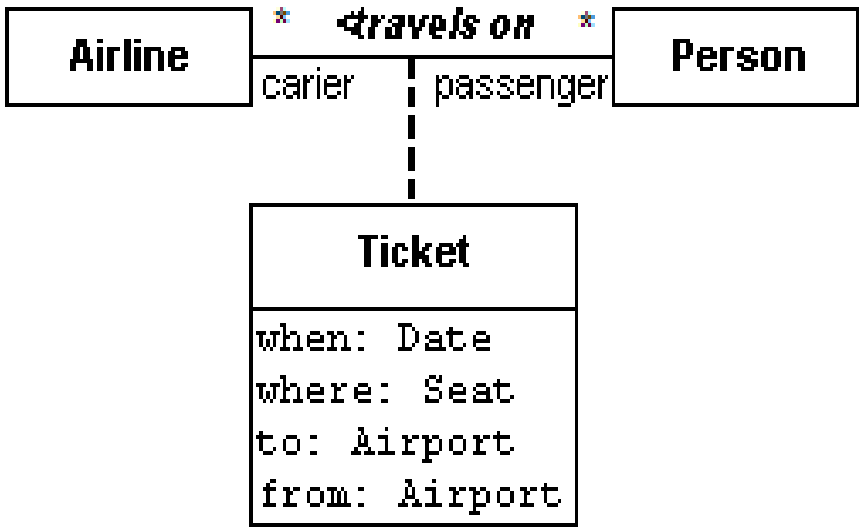
Yhteysluokasta normaaliluokaksi

- Henkilön jäsenyys yhdistyksessä on myös luontevaa kuvata yhteysluokan sijaan omana luokkanaan
- Yhteyttä kuvaava luokka Jäsenyys on nyt liitetty sekä Henkilöön että Yhdistykseen mustalla salmiakilla. Mistä on kysymys?
  - Musta salmiakki, eli kompositio siis tarkoittaa, että yhteyden toisessa päässä oleva olio on olemassaoloriippuvainen salmiakin päässä olevasta oliosta
  - Jäsenyys on olemassaoloriippuvainen sekä henkilöstä että yhdistyksestä
  - Jäsenyys ei voi siirtyä toiselle ihmiselle tai toiseen yhdistykseen
  - Jos ihminen kuolee tai yhdistys lakkautetaan tuhoutuu jäsenyys



Kaksi olioa ja yhteyksien lukumäärä

- Kurssin kotisivulle linkitetyssä Holubin UML quick referensessä on alla oleva esimerkki
- Eli henkilöllä voi olla *travels on* -yhteyksiä useiden lentoyhtiöiden kanssa
  - Yhteysluokkana Ticket on kerrottu matkan tiedot
- Tähän malliin sisältyy ongelma:
  - Henkilö voi olla *travels on* -yhteydessä moniin eri lentoyhtiöolioihin
  - Saman lentoyhtiöolion (esim. Finnair) kanssa ei kuitenkaan voi olla useampaa yhteyttä



- Siis: **jos luokkakaaviossa kahden luokan välillä on yhteys, voi kaksi luokkien olioa olla vain yhdessä yhteydessä kerrallaan**
  - Esim. olioiden arto:Person ja finnair:Airline välillä voi olla vain yksi yhteys, eli Arto voi lentää Finnairilla vain kerran
  - Tämä siitä huolimatta, että kytkentärajoitus on *
  - Artolla voi olla useita lippuja, mutta jokainen täytyy olla eri lentoyhtiöltä!
- Järkevin ratkaisu ongelmaan on kuvata yhteys omana luokkanaan
  - Henkilöllä voi olla useita lippuja
  - Lippu liittyy tiettyyn lentoyhtiöön ja tiettyyn henkilöön
  - Lentoyhtiön liittyy useita myytyjä lippuja

Luokkakaavion laatiminen

- Kuten jo muutamaan kertaan on mainittu, olioperustaisessa ohjelmistokehityksessä pyritään muodostamaan koko ajan tarkentuva luokkamalli, joka ”simuloi” sovelluksen kohdealuetta
  - Ensin luodaan **määrittelyvaiheen oliomalli** sovelluksen käsitteistöstä
  - Suunnitteluvaiheessa tarkennetaan edellisen vaiheen oliomalli **suunnitteluvaiheen oliomalliksi**
- Tarkastellaan seuraavaksi miten alustava, määrittelyvaiheen luokkamalli voidaan muodostaa
    - Ideana tunnistaa sovelluksen kohdealueen käsitteet ja niiden väliset suhteen
    - Eli karkeasti ottaen tehtävänä on etsiä todellisuutta simuloiva luokkarakenne
- Esiteltävästä menetelmästä käytetään nimitystä *käsiteanalyysi* (engl. conceptual modeling)
- Järjestelmän sovellusalueen käsitteistöä kuvaavaa luokkamallia kutsutaan usein **kohdealueen luokkamalliksi** (engl. problem domain model)

Menetelmä alustavan luokkamallin muodostamiseen

- Menetelmän voi ajatella etenevän seuraavien vaiheiden kautta
    1. Kartoita luokkaehdokkaat
    2. Karsi luokkaehdokkaita
    3. Tunnista olioiden väliset yhteyksiä
    4. Lisää luokille attribuutteja
    5. Tarkenna yhteyksiä
    6. Etsi ”rivien välissä” olevia luokkia
    7. Etsi yläkäsitteitä
    8. Toista vaiheita 1-7 riittävän kauan

- Yleensä aloitetaan vaiheella 1 ja sen jälkeen edetään sopivalta tuntuvassa järjestyksessä
- On hyvin epätyypillistä, että ensimiettimältä päädytään ”lopulliseen” ratkaisuun
- Katsotaan vaiheita hieman tarkemmin

Luokkaehdokkaiden kartoitus

- Laaditaan lista tarkasteltavan sovelluksen kannalta keskeisistä asioista, kohteista ja ilmiöistä, joita ovat esim:
  - Toimintaan osallistujat
  - Toiminnan kohteet
  - Toimintaan liittyvät tapahtumat, materiaalit ja tuotteet ja välituotteet
  - Toiminnalle edellytyksiä luovat asiat
- Kartoituksen pohjana voi käyttää esim.
  - kehitettävästä järjestelmästä tehtyä vapaamuotoista tekstuaalista kuvausta tai
  - järjestelmän halutusta toiminnallisuudesta laadittuja käyttötapauksia
- **Luokkaehdokkaat** ovat yleensä järjestelmän toiminnan kuvauksessa esiintyviä **substantiiveja**
- *Eli etsitään käytettävissä olevista kuvauksista kaikki substantiivit ja otetaan ne alustaviksi luokkaehdokkaiksi*

Esimerkki: lipun varaaminen elokuvateatterista

- Seuraavassa vapaamuotoinen tekstikuvaus elokuvalipun varaamisesta. Substantiivit alleviivattu.
- Tarkasteltavana ilmiönä on elokuvalipun varaaminen. Lippu oikeuttaa paikkaan tietyssä näytöksessä. Näytöksellä tarkoitetaan elokuvan esittämistä tietyssä teatterissa tiettyyn aikaan. Samaa elokuvaa voidaan esittää useissa teattereissa useina aikoina. Asiakas voi samassa varauksessa varata useita lippuja yhteen näytökseen.
- Löydettiin siis seuraavat substantiivit
  - elokuvalippu
  - lippu
  - näytös
  - esittäminen
  - aika
  - varaus
  - varaaminen
  - paikka
  - elokuva
  - teatteri
  - asiakas

Ehdokkaiden karsiminen

- Näin aikaansaadulla listalla on varmasti ylimääräisiä luokkakandidaatteja
- Karsitaan sellaiset jotka eivät vaikuta potentiaalisilta luokilta
- Kysymyksiä, joita karsiessa voi miettiä
  - Liittyykö käsitteeseen tietosisältöä?
    - On tosin olemassa myös tietosisällöttömiä luokkia...
  - Onko käsitteellä merkitystä järjestelmän kannalta
  - Onko kyseessä jonkin muun termin synonyymi
  - Onko kyseessä jonkin toisen käsitteen attribuutti

- Huomattavaa on, että kaikki luokat eivät yleensä edes esiinny sanallisissa kuvauksissa, vaan ne löytyvät vasta jossain myöhemmässä vaiheessa

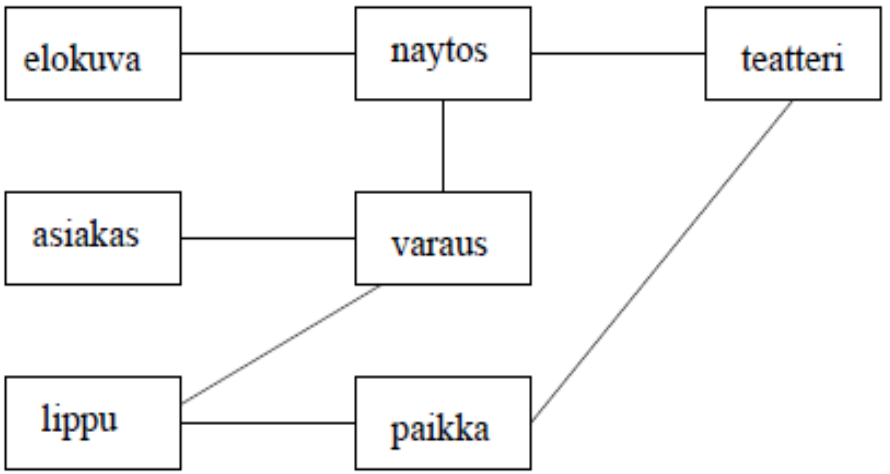
Ehdokkaiden karsiminen

- Karsitaan listalta yliviivatut
  - ~~elokuvalippu~~
    - Termin lippu synonyymi
  - ~~varaaminen~~
    - tekemistä
  - lippu
  - paikka
  - näytös
  - elokuva
  - ~~esittäminen~~
    - tekemistä
  - teatteri
  - ~~aika~~
    - Kyseessä lienee näytöksen attribuutti
  - asiakas
  - varaus

Alustava yhteyksien tunnistaminen

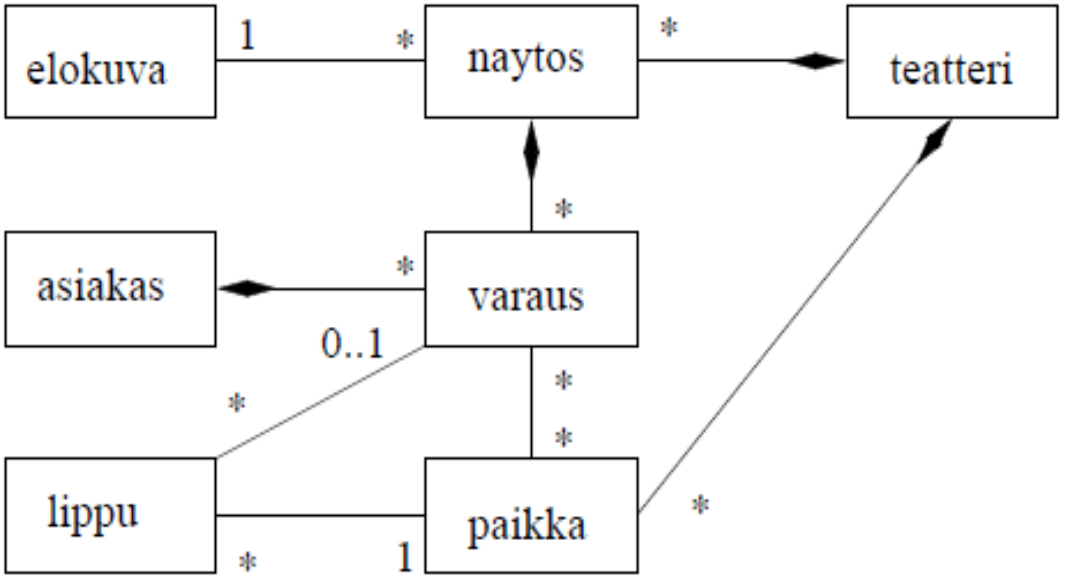
- Kun luokat on alustavasti tunnistettu, kannattaa ottaa paperia ja kynä ja piirtää alustava luokkakaavio, joka koostuu vasta luokkalaatikoista
- Tämän jälkeen voi ruveta miettimään minkä luokkien välillä on yhteyksiä
- Aluksi yhteydet voidaan piirtää esim. pelkkinä viivoina ilman yhteys- ja roolinimiä tai osallistumisrajoitteita
- Tekstuaalisessa kuvauksessa olevat **verbit ja genetiivit** viittaavat joskus olemassaolevaan **yhteyteen**
  - Lippu *oikeuttaa* paikkaan tietyssä näytöksessä
  - Näytöksellä *tarkoitetaan* elokuvan esittämistä tietyssä teatterissa tiettyyn aikaan
  - Samaa elokuvaa *voidaan esittää* useissa teattereissa useina aikoina.
  - Asiakas voi samassa varauksessa *varata* useita lippuja yhteen näytökseen
- Huom: kaikki verbit eivät ole yhteyksiä
  - Yhteydellä tarkoitetaan pysyvää suhdetta, usein verbit ilmentävät ohimeneviä asioita

- Verbien tuomat vihjeet eivät sisällä kaikkia yhteyksiä, toisaalta mukana voi olla myös ei pysyvää yhteyttä ilmentäviä asioita
  - Lippu – paikka
  - Näytös – elokuva
  - Näytös – teatteri
  - Elokuva – teatteri
  - Asiakas – varaus
  - Asiakas – lippu
  - Lippu – varaus
- Seuraavassa jonkinlainen hahmotelma. Osa yhteyksistä siis edellisen listan ulkopuolelta, ”maalaisjärellä” pääteltyjä
  - Joitain yhteyksiä (kuten elokuva–teatteri) jätetty redundantteina pois



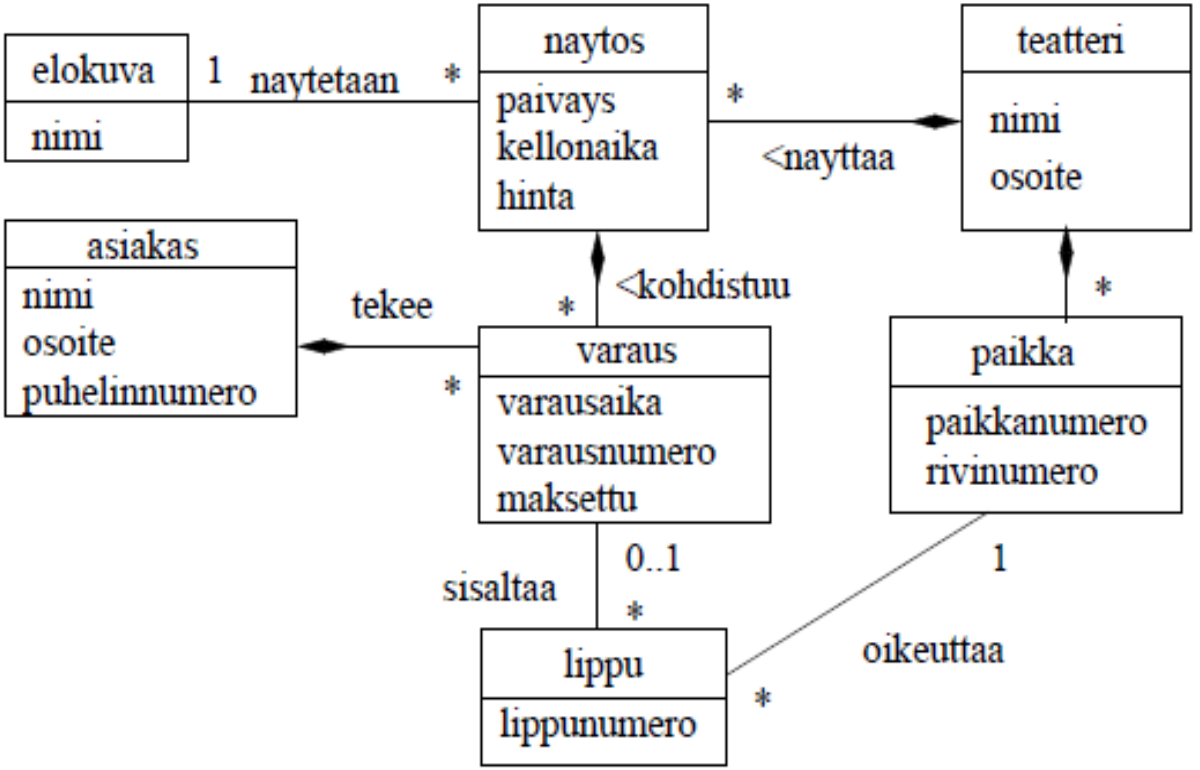
Yhteyksien tarkentaminen

- Kun yhteys tunnistetaan ja vaikuttaa tarpeelliselta, tarkennetaan yhteyden laatua ja kytkentärajoitetta
- Ei ole olemassa oikeaa etenemisstrategiaa
- Yksi mahdollisuus on tarkastella ensin luokkia esim. pareina
  - miten liittyvät toisiinsa elokuva ja näytös
  - miten liittyvät toisiinsa asiakas ja varaus
  - …
- Seuraavassa askeleen tarkentunut elokuvateatteri



Yhteyksien tarkentaminen ja attribuuttien etsiminen

- Attribuuttien löytäminen edellyttää yleensä lisätietoa, esim. asiakkaan haastatteluista
- Määrittelyvaiheen aikana tehtävää kohdealueen luokkamallia ei ole välttämättä tarkoituksenmukaista tehdä kaikin osin tarkaksi
  - Malli tarkentuu ja muuttuu jokatapauksessa suunnitteluvaiheessa
- Elokuvateatterin kolmas vaihe kuvassa

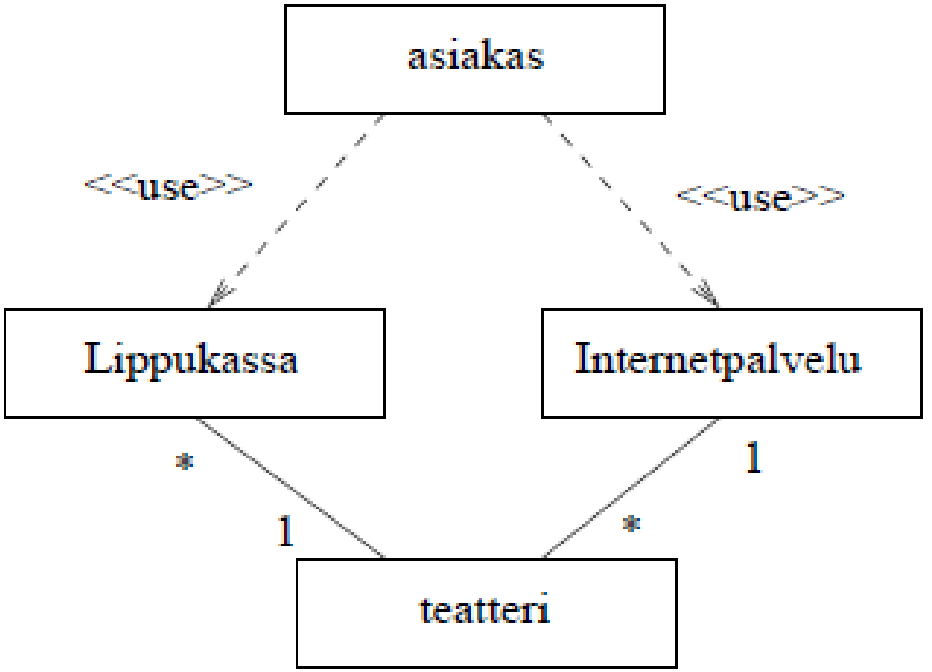


Mikä ei ole yhteys ja mikä ei?

- Oletetaan, että lipunvaraustapahtuman tekstuaaliseen kuvaukseen liittyisi myös seuraava:
  - *Asiakas tekee lippuvarauksen elokuvateatterin internetpalvelun kautta. Elokuvateatterissa on useita lippukassoja. Asiakas lunastaa varauksensa lippukassalta viimeistään tuntia ennen esitystä.*
- Tästä kuvauksesta löytyy kaksi uutta luokkakandidaattia:
  - internetpalvelu
  - Lippukassa
- Tekstuaalisen kuvauksen perusteella teatterilla on yhteys internetpalveluun sekä lippukassoihin
- Nämä ovat selkeitä "rakenteellisia" yhteyksiä jotka voidaan merkata myös luokkakaavioon

Mikä ei ole yhteys ja mikä ei?

- Tekstuaalinen kuvaus antaa viitteen, että asiakkaalla on yhteys sekä internetpalveluun että lippukassaan
- Näitä ei kuvata luokkamallissa yhteytenä sillä kyse ei ole rakenteisesta, pysyvälaatuisesta yhteydestä, vaan hetkellisestä käytöstä
- Jos se, että asiakas käyttää lippukassan palvelua halutaan merkata luokkakaavioon, tulee yhteyden sijasta käyttää riippuvuutta



Luokkakaavion laatiminen

- Muutama sivu sitten olleella listalla mainittiin yhdeksi vaiheista *yläkäsitteiden etsiminen*
  - Tämä liittyy periytymiseen ja palaamme asiaan viikon päästä
  - Lyhyesti: jos tekisimme yleistä lippupalvelujärjestelmää, olisi lippu todennäköisesti yläkäsite, joka erikoistuu esim. elokuvalipuksi, konserttilipuksi, ym...

- Määrittelyvaiheen aikana tehtävään sovelluksen kohdealueen luokkamalliin ei vielä liitetä mitään metodeja
  - Metodien määrittäminen tapahtuu vasta ohjelman suunnitteluvaiheessa
  - Palaamme aiheeseen myöhemmin
  - Suunnitteluvaiheessa luokkamalli tarkentuu muutenkin monella tapaa

Toinen esimerkki: kampaamo

Kampaamo X on kehittelemässä asiakkailleen varauspalvelua. Asiakkaat rekisteröityvät järjestelmään ensimmäisen varauksensa yhteydessä. Rekisteröityneelle asiakkaalle annetaan asiakastunnus. Asiakkaasta tallennetaan järjestelmään perustietoja, kuten nimi, osoite ja puhelinnumero.

Tarjolla olevista palveluista on olemassa hinnasto. Hinnastossa kerrotaan kunkin palvelun osalta hinta ja kesto.

Ajanvarauksen yhteydessä asiakas valitsee, mitä toimenpiteitä hän haluaa suoritettavaksi. Asiakas voi myös valita samanlaisen palvelukokonaisuuden, jonka hän on saanut jollain aiemmalla käynnillä. Tätä varten järjestelmässä on säilytettävä tiedot aiemmista käynneistä.

Käynnillä tehtävien toimenpiteiden perustana on varauksessa ilmoitettu toive. Palvelutilanteessa asiakas ja kampaaja voivat kuitenkin päätyä varauksesta poikkeavaan toimenpidekokoelmaan.

Varauksen tietoja ei tarvitse säilyttää kuin varattuun aikaan asti.

Kampaamossa työskentelee 4 kampaajaa, jotka ovat työssä epäsäännöllisesti. Kaikki kampaajat eivät tee kaikkia tarjolla olevia toimenpiteitä.

Varaustilanteessa järjestelmän pitää kyetä näyttämään asiakkaalle, milloin halutun palvelun suorittamaan pystyviltä kampaajilta löytyy vapaita aikoja.

Kuvauksesta löytyneet substantiivit

- Kampaamo
- Varauspalvelu
- Asiakas
- Asiakastunnus
- Nimi
- Osoite
- Puhelinnumero
- Palvelu
- Hinta
- Kesto
- Hinnasto
- Ajanvaraus
- Toimenpide
- Palvelukokonaisuus
- Käynti
- Tiedot
- Toive
- Palvelutilanne
- Toimenpidekokoelma
- Aika
- Kampaaja
- Työ
- Varaustilanne

Synonyymien ja attribuuttien erottaminen, turhien poisto

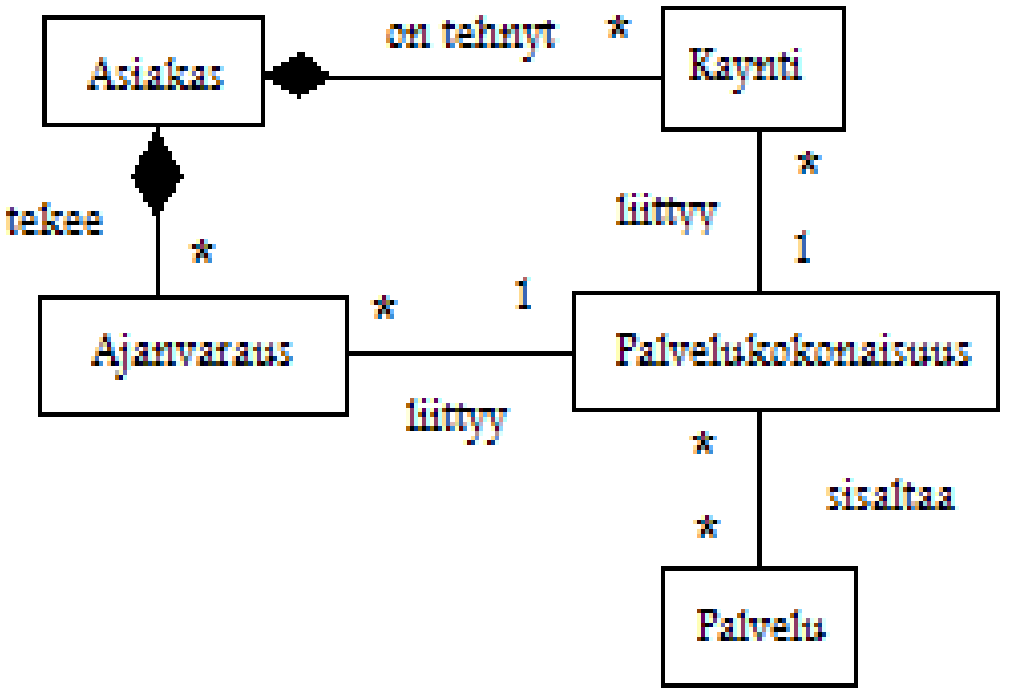
- ~~Kampaamo~~
  - *Liian yleinen*
- Varauspalvelu
- Asiakas, *attribuutteja:*
  - Asiakastunnus
  - Nimi
  - Osoite
  - Puhelinnumero
- Palvelu, *attribuutteja*:
  - Hinta
  - Kesto
- Hinnasto
- Ajanvaraus, *attribuutti:*
  - Toive
- ~~Toimenpide~~
  - *Palvelun synonyymi*
- Palvelukokonaisuus
- ~~Toimenpidekokoelma~~
  - *Palvelukokonaisuuden synonyymi*
- Käynti, *attribuutti:*
  - Tiedot
- ~~Palvelutilanne~~
  - *tekemistä*
- Kampaaja
- ~~Työ~~
  - *Epämääräinen käsite*
- Aika
- ~~Varaustilanne~~
  - *tekemistä*

Alustavasti valitut luokat

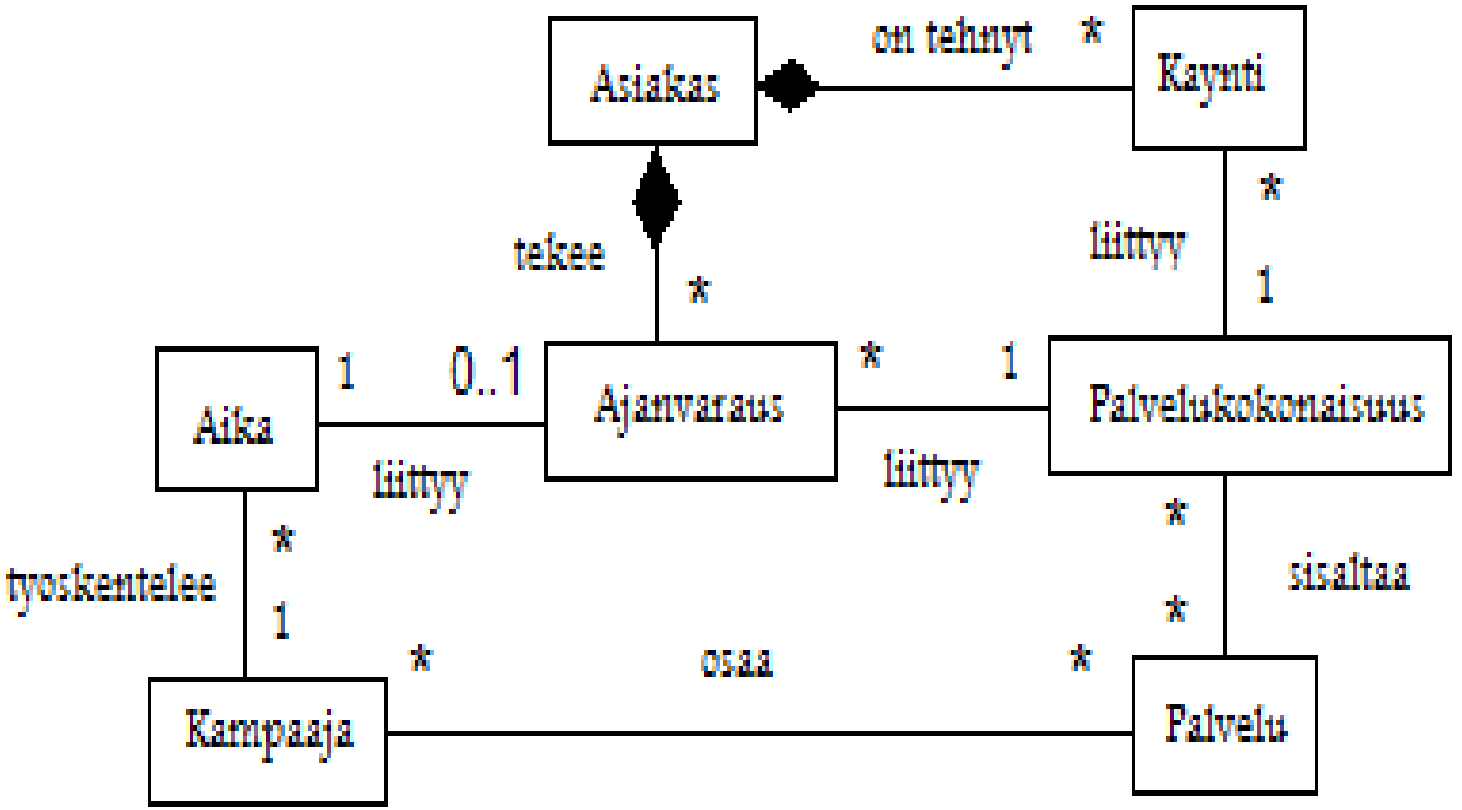
- Varauspalvelu
    - Itse järjestelmää kuvaava luokka
- Asiakas
- Palvelu
- Hinnasto
- Palvelukokonaisuus
- Ajanvaraus
- Käynti
- Kampaaja
- Aika

Yhteyksiä

- Tarkastellaan luokkia *asiakas, ajanvaraus, palvelu, palvelukokonaisuus* ja *käynti*
- Seuraavassa tekstikuvaus, jossa synonyymit korvattu valituilla termeillä:
  - Ajanvarauksen yhteydessä asiakas valitsee, mitä palveluita hän haluaa suoritettavaksi
  - Asiakas voi myös valita samanlaisen palvelukokonaisuuden, jonka hän on saanut jollain aiemmalla käynnillä
  - Tätä varten järjestelmässä on säilytettävä tiedot aiemmista käynneistä
- Asiakkaaseen liittyy ajanvarauksia ja käyntejä
- Ajanvaraukseen ja käyntiin liittyy palvelukokonaisuus
- Palvelukokonaisuus koostuu palveluista
- Päädytään seuraavan sivun alustavaan luokkakaavioon



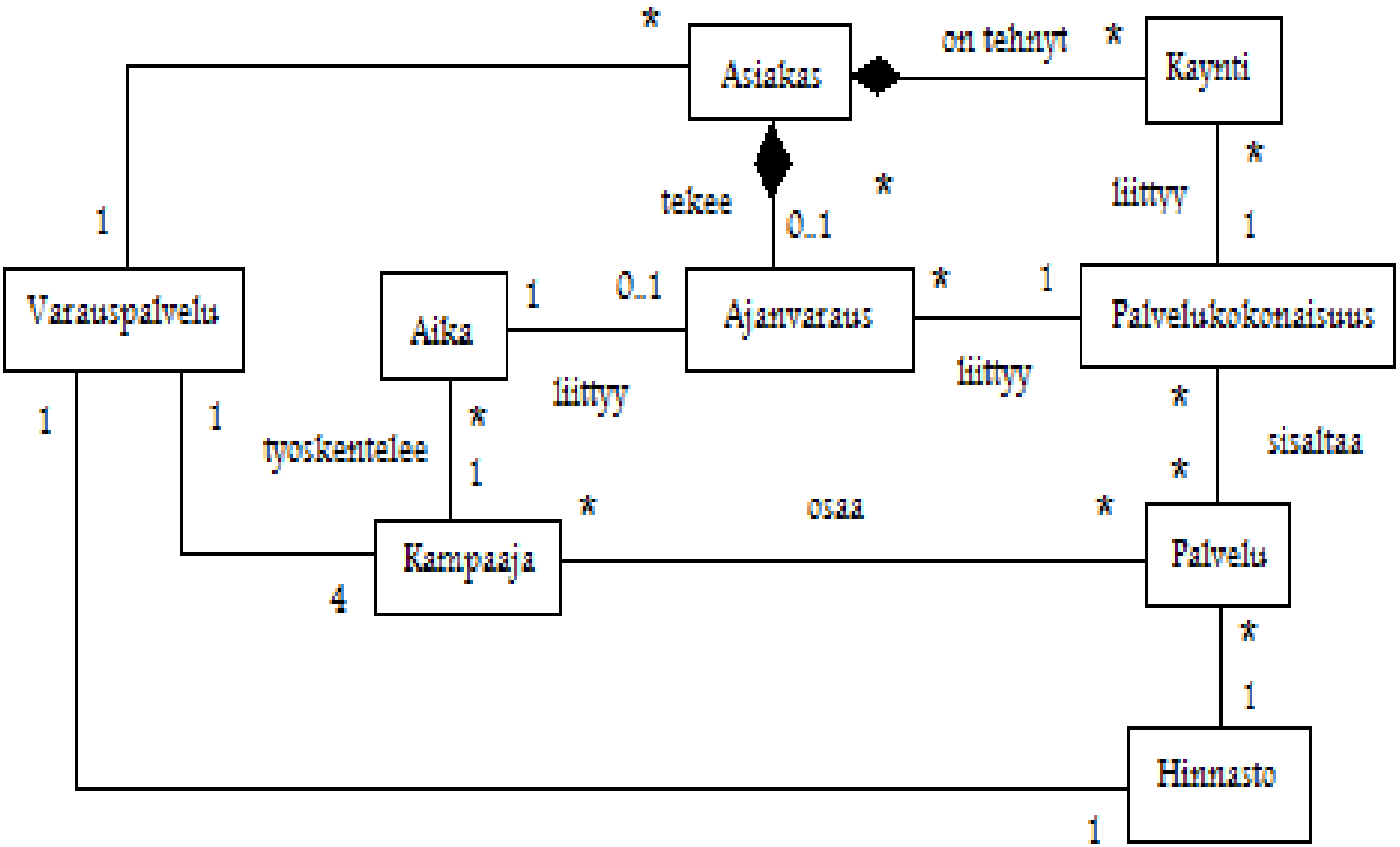
- Tarkastellaan luokkia aika, kampaaja
  - Kampaamossa työskentelee 4 kampaajaa, jotka ovat työssä epäsäännöllisesti. Kaikki kampaajat eivät tee kaikkia tarjolla olevia toimenpiteitä.
  - Varaustilanteessa järjestelmän pitää kyetä näyttämään asiakkaalle, milloin halutun palvelun suorittamaan pystyviltä kampaajilta löytyy vapaita aikoja.
- Kampaajaan liittyy aikoja jolloin hän on töissä
- Aika liittyy myös varaukseen
- Kampaajaan liittyy myös joukko palveluita joita kampaaja osaa suorittaa



- Jäljelle jäivät luokat hinnasto ja varauspalvelu
  - Tarjolla olevista palveluista on olemassa hinnasto. Hinnastossa kerrotaan kunkin palvelun osalta hinta ja kesto.
- Eli hinnasto sisältää palvelut
- Hinnasto, kampaajat ja asiakkaat ovat koko järjestelmästä huolehtivan luokan Varauspalvelu alla

Kampaamon luokkakaavion alustava versio

- Päädymme allaolevaan alustavaan luokkakaavioon
- Todellisuudessa mallin laatiminen ei edennyt yhtä suoraviivaisesti kuin näillä kalvoilla vaan sisälsi hapuilevia askelia
- Tekemällä erilaisia valintoja ja oletuksia, oltaisiin voitu päätyä hieman erilaisiin yhtä perusteltuihin ratkaisuihin



Mallinnus iteratiivisesti etenevässä ohjelmistokehityksessä

- Jos ohjelmiston kehittäminen tapahtuu ketterien menetelmien suosittamalla iteratiivisella lähestymistavalla, kannattaa myös ohjelman luokkamallia rakentaa iteratiivisesti
- Eli jos ensimmäisessä iteraatiossa toteutetaan esim. ainoastaan käyttötapausten 1 ja 2 mukainen toiminnallisuus, keskitytään luokkamallissa pääosiin vain niihin luokkiin, jotka ovat merkityksellisiä tarkastelun alla olevan toiminnallisuuden kannalta
- Luokkamallia täydennetään myöhempien iteraatioiden aikana tarpeellisilta osin