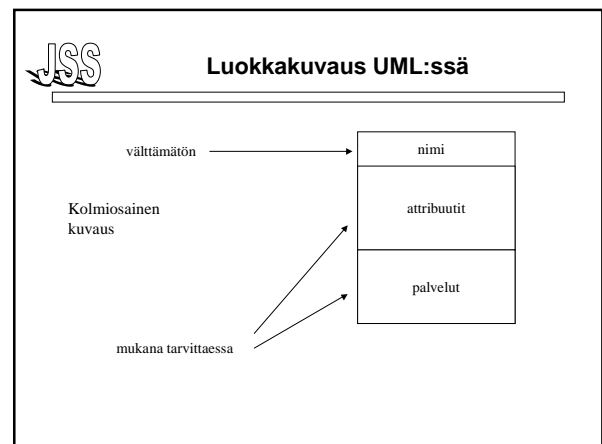
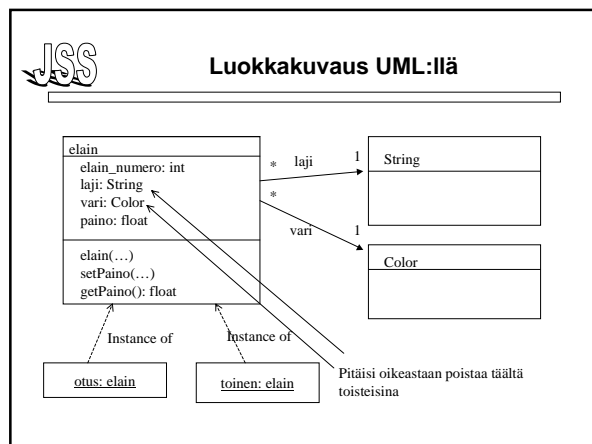
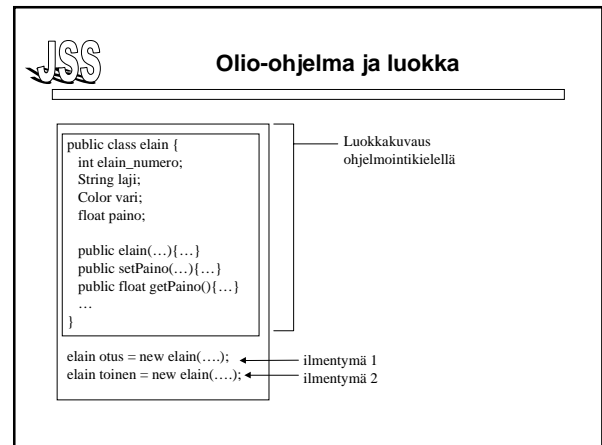
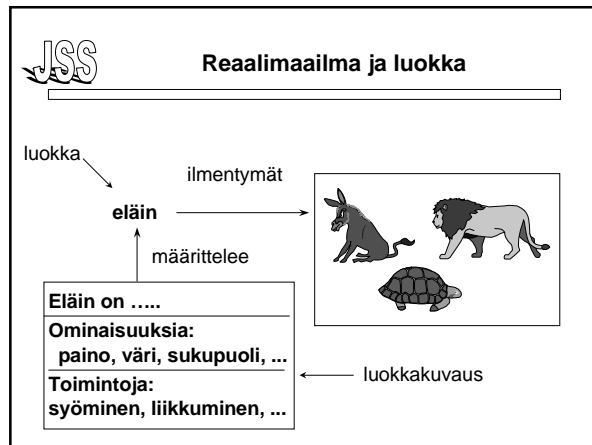


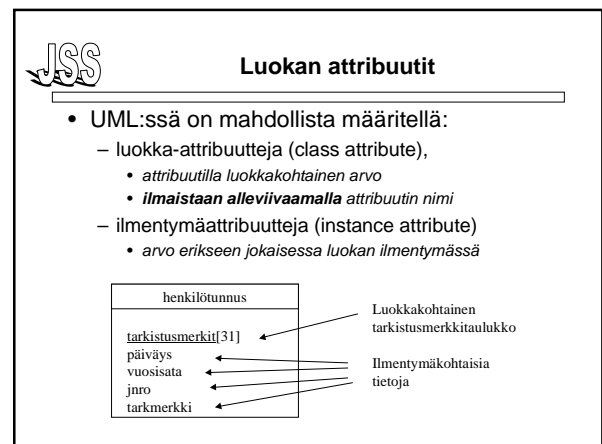
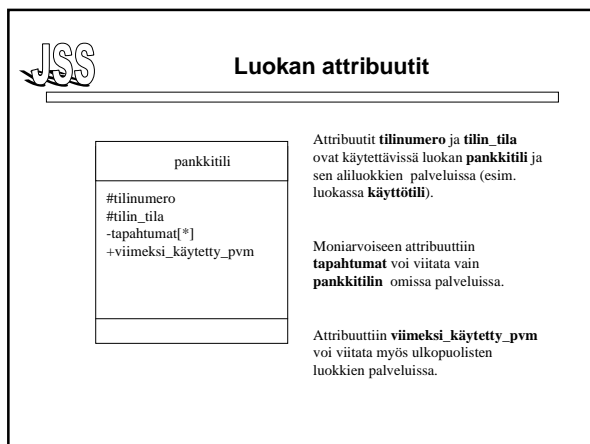
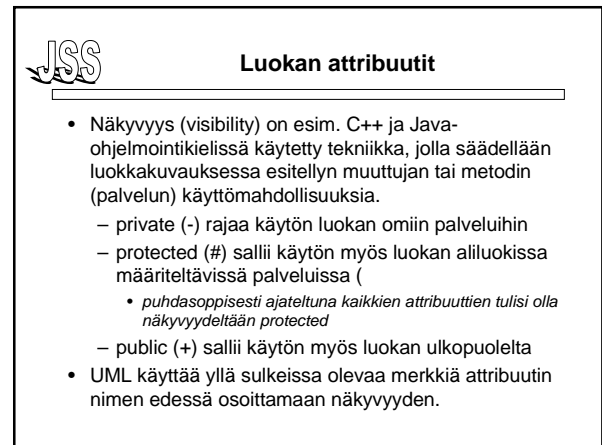
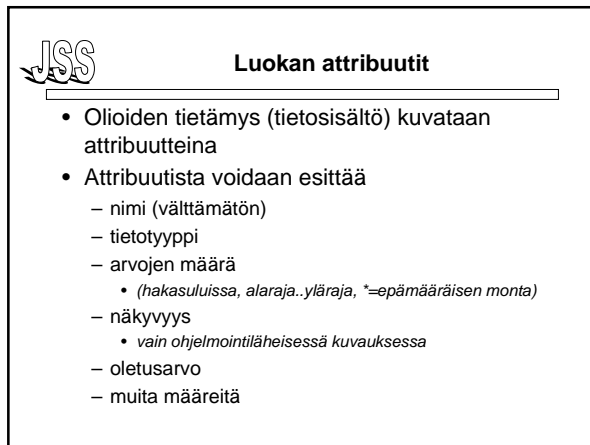
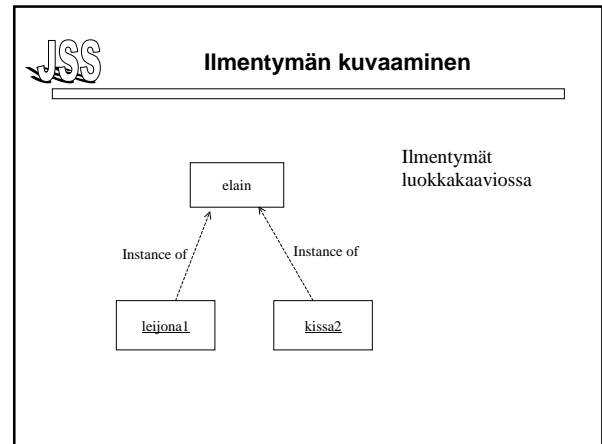
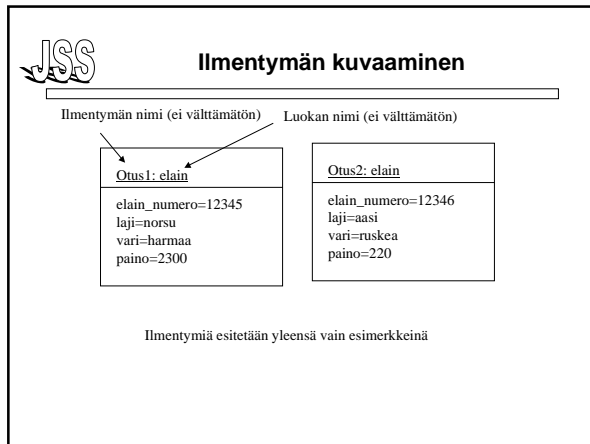
JSS Luokka- ja oliokaaviot

- 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

JSS 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.





JSS **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 tietotyyppiin, vaan sallii mitä tahansa perustietotyyppiä.
 - yllä int ja char

JSS **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

JSS **Luokan attribuutit**

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

JSS **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

pankkitili
tilinumero {frozen, id}

Tilinumeroa ei voi muuttaa, se yksilöi tilit.

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

JSS **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

JSS **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 määreitä]]

- parametrin määrittelyn muoto on [suunta] nimi: tyyppi [= oletusarvo]

JSS Luokan palvelut

- Esim.

```

classDiagram
    class Henkilötunnus {
        #tarkistusmerkit[31]
        #päiväys
        #vuosisata
        #jnro
        #tarkmerkki
        +toString():String
        +isOK():Boolean
        +getBirthDate():Date
        +getSex():int
    }
    
```

JSS 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.

JSS Olioiden väliset yhteydet

```

classDiagram
    class Pankkitili {
        omistaja[*]:Asiakas
    }
    class Asiakas
    Pankkitili "1" -- "*" Asiakas : omistaja->
    
```

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

Yhden tilin voi omistaa useita asiakkaita.

JSS Olioiden väliset yhteydet

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

```

classDiagram
    class Pankkitili
    class Asiakas
    Pankkitili "1" -- "1..3" Asiakas : omistaja->
    
```

JSS Olioiden väliset yhteydet

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

```

classDiagram
    class Pankkitili {
        1234:pankktili
    }
    class Asiakas {
        Aku Ankka:Asiakas
        Ines Ankka:Asiakas
    }
    Pankkitili --> Asiakas : omistaja
    
```

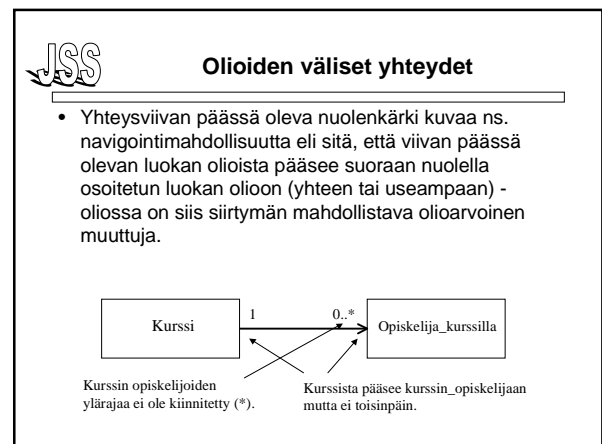
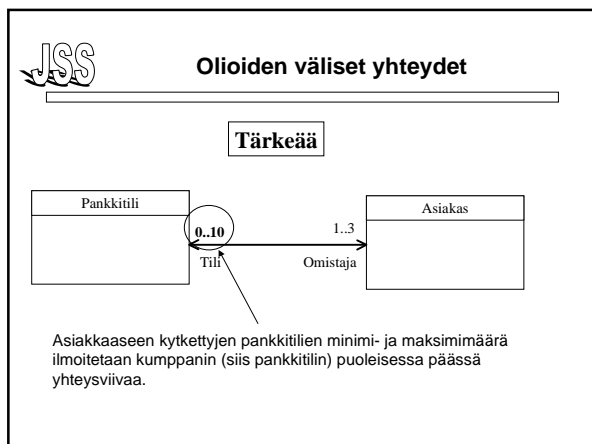
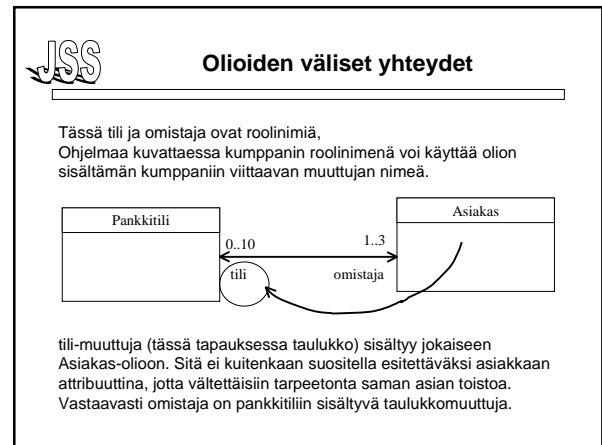
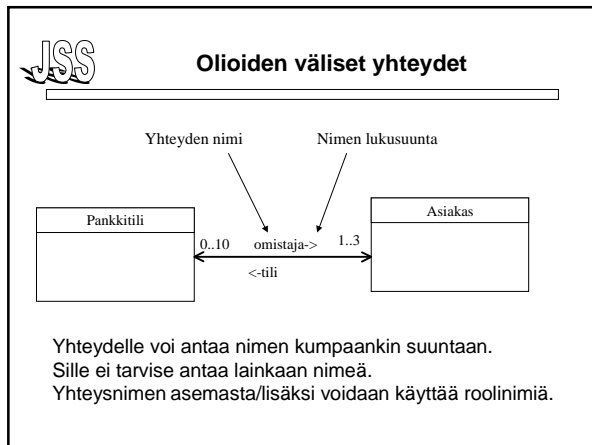
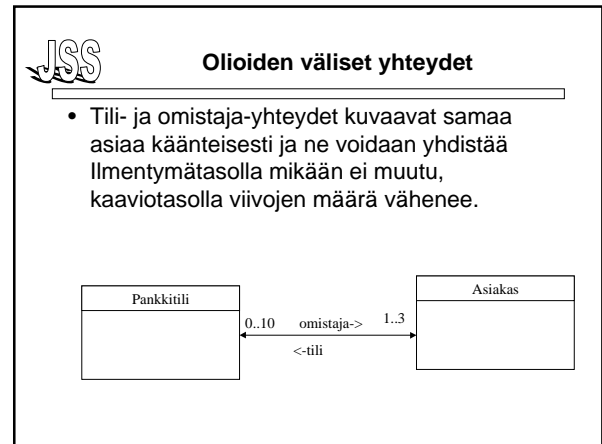
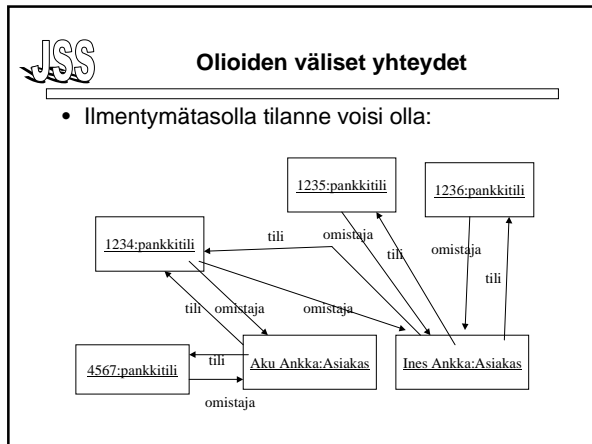
JSS Olioiden väliset yhteydet

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

```

classDiagram
    class Pankkitili
    class Asiakas
    Pankkitili "1" -- "1..3" Asiakas : omistaja->
    Asiakas "0..10" --> "1" Pankkitili : <-tili
    
```

Asiakkaalla ei välttämättä ole yhtään tiliä



JSS **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.

JSS **Olioiden väliset yhteydet**

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

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

JSS **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.

JSS **Olioiden väliset yhteydet**

JSS **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.

JSS **Olioiden väliset yhteydet**

Järjestetty joukko

JSS 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.

Pelaaja voi kuulua moneen joukkueeseen.

Pelkkä * tarkoittaa samaa kuin pari 0..*

JSS 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.

JSS 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

JSS Olioiden väliset yhteydet

- Esimerkkejä:

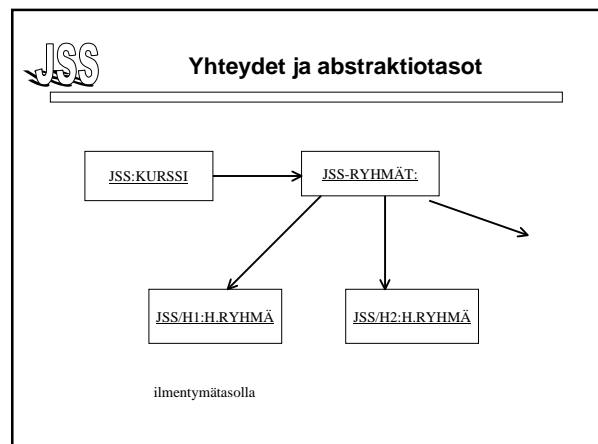
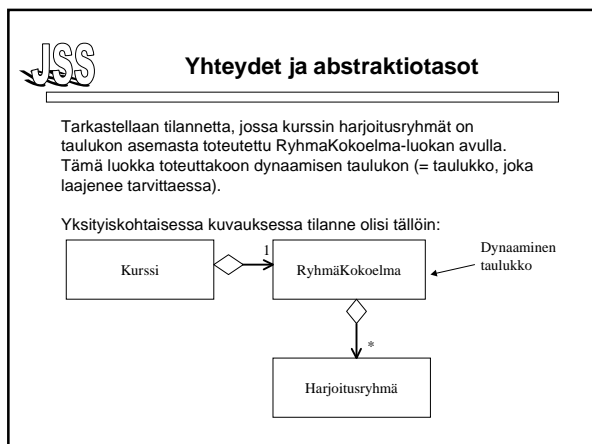
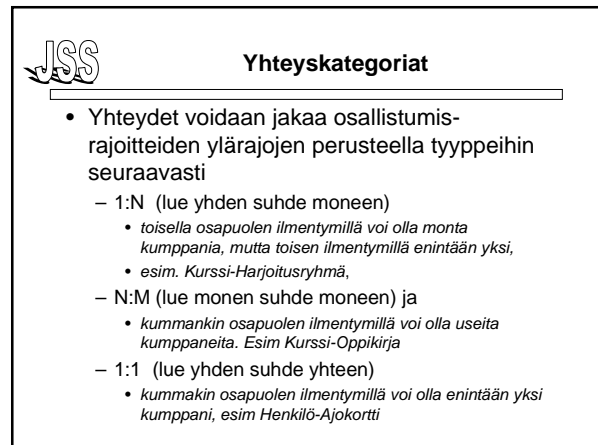
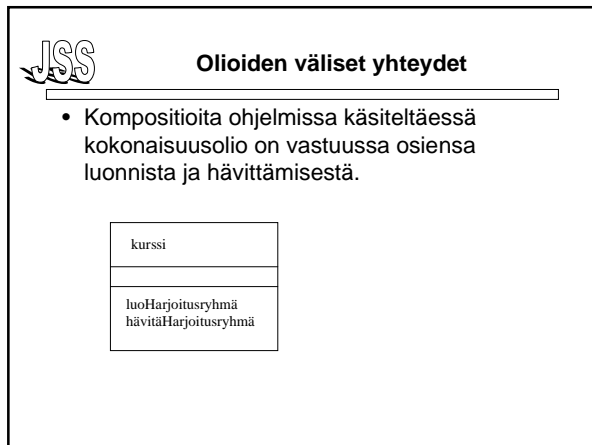
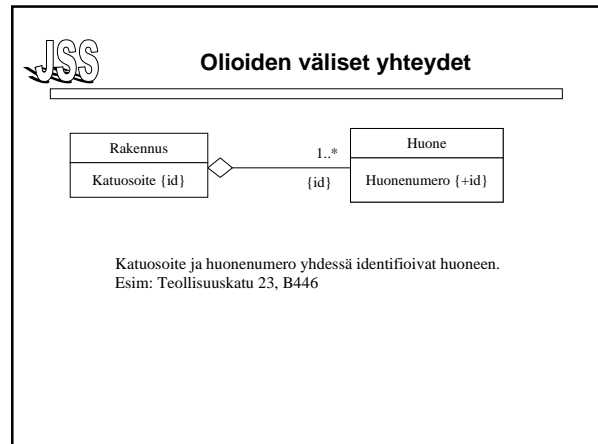
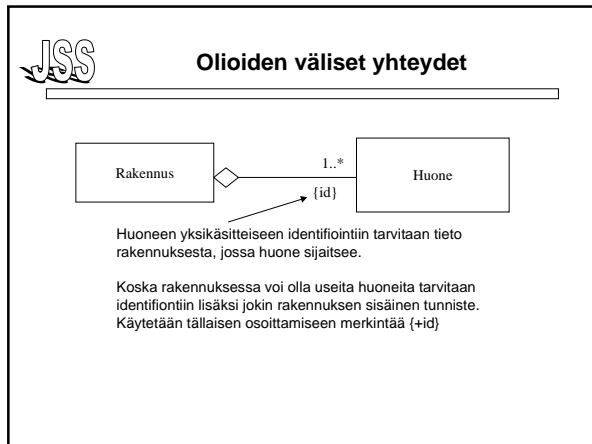
Kompositio esitetään mustalla salmiakilla kokonaisuuden puoleisessa päässä.

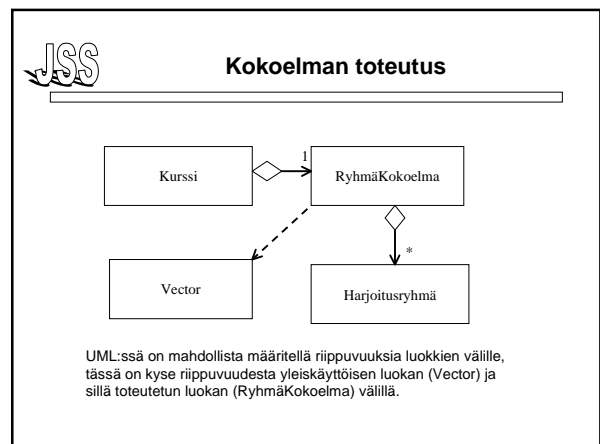
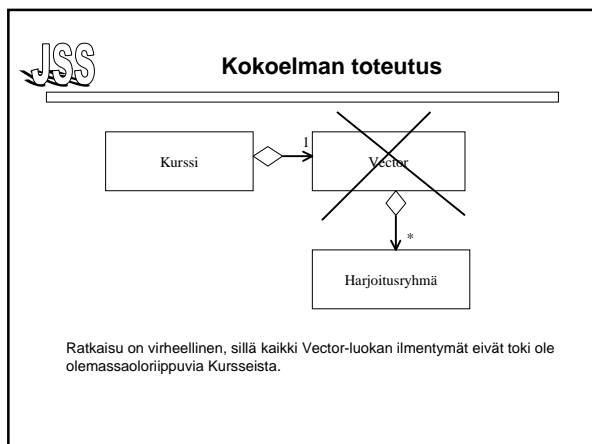
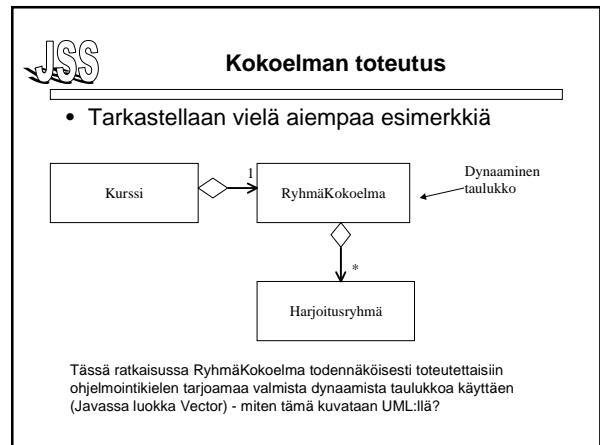
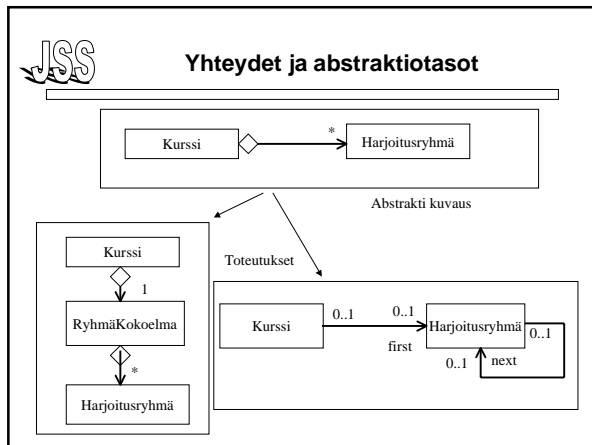
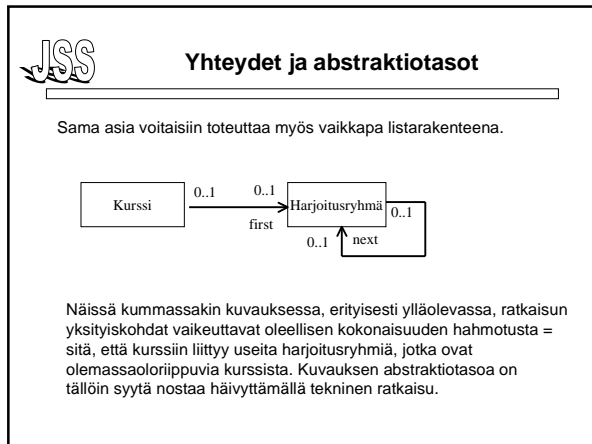
JSS Olioiden väliset yhteydet

- 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.

JSS Olioiden väliset yhteydet

- Kompositiota käytetään reaali maailmassa 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 yhteydessä osan puolelle liitettävä lisämääre {id} esittämään tätä.





JSS **Kokoelman toteutus**

- Java-ohjelmana edellisen kalvon esimerkki toteutuisi seuraavasti:

```
public class Kurssi {
    Vector RyhmäKokoelma;

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

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

JSS **Yhteydet ja abstraktiotasot**

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

JSS **Yhteydet ja abstraktiotasot**

Ilmentymätasolla

JSS **Yhteydet ja abstraktiotasot**

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

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

JSS **Yhteydet ja abstraktiotasot**

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

JSS **Yhteydet**

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

Opettaja käyttää kurssilla tiettyä oppikirjaa.

JSS 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

JSS Riippuvuus

- Luokien välisellä riippuvuudella tarkoitetaan tilannetta, jossa luokan määrittelyssä 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.

JSS Riippuvuus

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

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

JSS 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.