



### Luokkakaavion tarkoitus

- Järjestelmän tietosisällön kuvaaminen:
  - tiedot ja niiden väliset kytkennät
  - järjestelmän tiedot kuvaavat kohdealueiden ilmiötä, joten luokkakaavion tulisi määrittellä kohdealueen rakenne



### Luokkakaavion tarkoitus

- Ohjelman rakenteen kuvaaminen:
  - tiedot ja niiden väliset kytkennät
  - tietojen yhteys toimintaan
  
  - Olio-ohjelman idea on usein simuloida vastaavaa reaali maailman (kohdealueen) ilmiötä, joten luokat löytyvät reaali maailmaa analysoimalla.
  - Kohdealueen rakenteen mukaiset luokat muodostavat tällöin ohjelman luokkarakenteen ytimen.



### Luokkakaavion laatiminen

- Kokonaisvaltainen lähestymistapa:
  - pyritään löytämään kerralla koko kohdealuetta kuvaava malli
  - hankalaa, jos kohdealue on laaja
  - ensin karkea yleiskuva, sitten lisää yksityiskohtia
- Osista kokonaisuuteen:
  - jaetaan kokonaisuus osiin ja tehdään osakohtaisia malleja, jotka sitten yhdistetään kokonaisuudeksi
  - osa voisi olla käyttötapaus
  - yksityiskohdista yleiskuvaan



### Luokkakaavion laatiminen

- Kartoita luokkaehdokkaita.
- Karsi ehdokkaita.
- Tunnista olioiden väliset yhteydet.
- Täsmennä luokkakuvauksia määrittelemällä attribuutit.
- Määrittele yhteyksiin liittyvät osallistumisrajoitteet.
- Liitä luokkiin palvelut.
- Varmista palvelujen ja tietosisällön yhteensopivuus.



### Kartoita luokkaehdokkaita

- Laadi luettelo tarkasteltavan ilmiön kannalta keskeisistä kohteista tai ilmiöistä, jotka voisivat tulla kyseeseen luokkina tai olioina:
  - osallistujat,
  - toiminnan kohteet,
  - toimintaan liittyvät tapahtumat,
  - materiaalit,
  - tuotteet ja välituotteet,
  - toiminnalle edellytyksiä luovat asiat.



### Kartoita luokkaehdokkaita

- Kartoituksen pohjana voi käyttää **vapaa-muotoista tekstikuvausta** tarkasteltavasta ilmiöstä.
  - Kuvauksesta alleviivataan luokkaehdokkaita ja kerätään ne luetteloon.
  - Luokkaehdokkaat esiintyvät kuvauksessa usein substantiiveina.
  - Verbit voivat ilmaista yhteyksiä.
  - Alustavaa karsintaa voi tehdä sen perusteella, onko asia lainkaan oleellinen mallinnettavan ilmiön kannalta.

**JSS** **Karsi ehdokkaita**

- Löydetyt ehdokkaat käydään läpi ja arvioidaan, voisiko ehdokas tulla kyseeseen luokkana.
  - Liittyykö luokan ilmentymiin tietosisältöä, joka on välttämätöntä järjestelmän kannalta (yleensä on oltava useita attribuutteja)?
  - Tarvitaanko tietoa ilmentymien olemassaolosta?
  - Onko asia riittävän tärkeä kohdealueen kannalta?
  - Karsitaan synonyymit.
- Karsintaa ja ehdokkaiden kartoitusta voidaan joutua tekemään iteratiivisesti. Ensimmäinen karsintakerros ei välttämättä tuota lopullista tulosta.

**JSS** **Tunnista yhteydet**

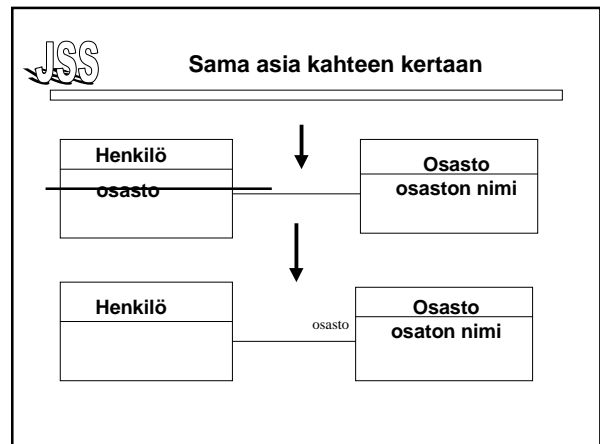
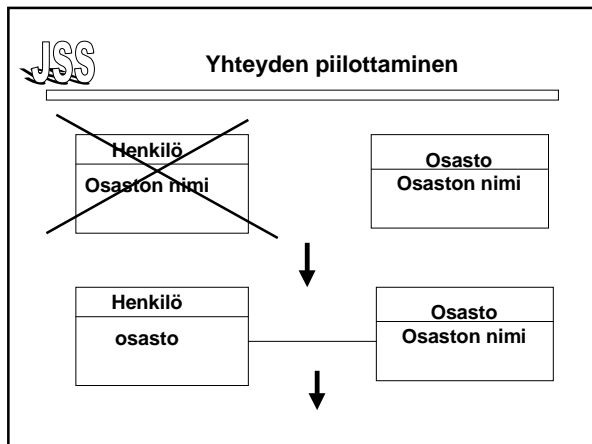
- Yhteyksiä (ilmentymien välillä) voi etsiä vapaamuotoisesta kuvauksesta:
  - *verbit*
  - *genetiivit*
  - *muut ilmaukset, jotka kuvaavat kytkeä*
- Yhteyksienkin suhteen tulisi miettiä
  - *onko yhteys oleellinen tarkasteltavan ilmiön kannalta?*
  - *onko se rakenteellinen?*
- Asia pitäisi esittää vain kertaalleen:
  - *johdettavissa olevat yhteydet?*
  - *karsitaan tai merkitään*
- Älä piilota yhteyksiä attribuuteiksi!

**JSS** **Määrittele attribuutit**

- Attribuutteja saattaa löytyä vapaamuotoisesta kuvauksesta.
- Yleensä niiden löytäminen edellyttää lisäselvityksiä kohdealueesta, esimerkiksi toiminnan osapuolten haastatteluja.
- Attribuuttien kohdalla pitäisi myös selvittää, mihin niitä tarvitaan.
- Ja uudelleen:
  - Älä piilota yhteyksiä attribuuteihin!

**JSS** **Määrittele osallistumisrajoitteet**

- Osallistumisrajoitteiden avulla ilmaistaan rakenteellisia **sääntöjä**.
- Säännöt eivät välttämättä tule esiin vapaamuotoisessa kuvauksessa, vaan edellyttävät tarkempaa kohdealueen analysointia.



**JSS**

---

- Puutarhatonttu esimerkki

**JSS** **Käyttötapaukset ja tietosisältö**

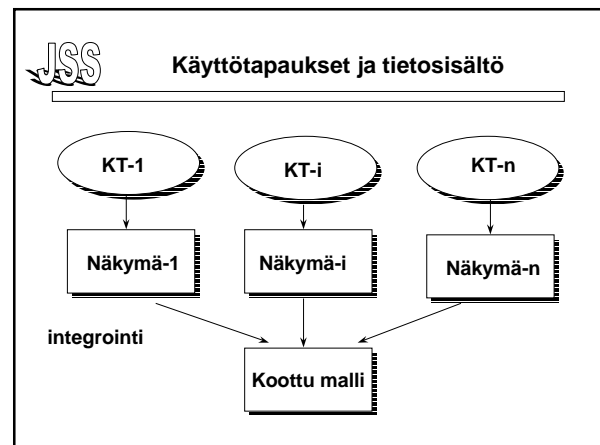
---

- Käyttötapaukset määrittelevät toiminnalliset vaatimukset järjestelmälle, mutta ne asettavat vaatimuksia myös järjestelmän tietosisällölle, koska toiminnot edellyttävät tietojen olemassaoloa.
- Eräs tapa järjestelmän sisältöluokkien suunnitteluun on tehdä suunnittelu käyttötapausittain ja sitten yhdistää käyttötapauskohtaiset mallit.

**JSS** **Käyttötapaukset ja tietosisältö**

---

- Lähtökohtana käyttötapaukset:
  - Mitä tietoja ja tietokokonaisuuksia tarvitaan käyttötapausten hoitamiseksi?
- Jokaisen käyttötapausten perusteella laaditaan käyttötapauskohtainen näkymä tarvittavista tiedoista, eli pelkästään käyttötapausten tarpeet huomioon ottava luokkakaavio.
- Näkymät yhdistetään sisällön kokonaismallin aikaansaamiseksi.



**JSS** **Käyttötapaukset ja tietosisältö**

---

- Käyttötapausittain sisältömallia muodostettaessa saadaan ainakin periaatteessa malliin mukaan kaikki käyttötapausten tarvitsemat tiedot.
- Käyttötapausmallin ja tietosisältömallin yhteensopivuuden varmistamiseksi voidaan käyttää riippuvuusmatriisia.

**JSS**

---

- Kampaamoesimerkki

**JSS Käyttötapaukset ja tietosisältö**

- **Riippuvuusmatriiseja:**
  - Luokat ja yhteydet / käyttötapaukset
    - *Matriisin alkiona riippuvuustieto:*
      - luo, muuttaa, poistaa, käyttää
  - Matriisit voidaan esittää myös attribuuttitasolla, jolloin näkyisi attribuuttien käsittely. Tällöin matriisit on kuitenkin syytä pilkkoa luokkakohtaisiksi.

**JSS Käyttötapaukset ja tietosisältö**

Olioluokat	Käyttötapaukset															
	Uus artikkele	Uus artikkeleversio	Tiedustelu artikkelin tilasta	Luovutuksen tila	Luovutuksen tilan vaihto	Luovutuksen tilan vaihto	Muutokset luovutukseen	Luovutuksen tilan vaihto	Luovutuksen tilan vaihto	Luovutuksen tilan vaihto	Luovutuksen tilan vaihto	Luovutuksen tilan vaihto	Luovutuksen tilan vaihto	Luovutuksen tilan vaihto	Luovutuksen tilan vaihto	Luovutuksen tilan vaihto
Article	L	M	K	K	K	K	K	K	K	K	K	K	K	K	K	K
Article version	L	L	K	M	K	K	K	K	K	K	K	K	K	K	K	K
Person	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Reference			K	L	M	M	M	M	M	M	M	M	M	M	M	M
Journal																

K= Käyttää, L = Luo, M= Muuttaa, P= Poistaa, X=Luo tai muuttaa

**JSS Käyttötapaukset ja tietosisältö**

- Jokaiselle luokalle ja yhteydelle täytyisi löytyä käyttötapaus, jolla voidaan luoda luokan ilmentymiä ja kytkeä olioita ko. yhteyteen.
- Jokaiselle luokalle ja yhteydelle täytyisi löytyä käyttötapaus, joka hyödyntää luokan tietoja ja yhteyksiä.
- Jos ilmentymät eivät ole ikuisia, pitää löytyä käyttötapaus, jolla hävitetään ilmentymiä.

**JSS Käyttötapaukset ja tietosisältö**

- Jos ilmentymän tietosisältö ei ole muuttumaton, pitää löytyä käyttötapaus, jolla tietosisältöä voidaan muuttaa.
- Jos osapuoli voidaan irroittaa yhteydestä, täytyy tähän tarkoitukseen löytyä käyttötapaus.
- Toisaalta jokaisen käyttötapauksen täytyy jollain tavoin liittyä sisältöluokkiin.

**JSS Luokkahierarkia**

- Luokkia määrittelemällä luodaan luokitusjärjestelmä.
- Eri menetelmät asettavat erilaisia vaatimuksia luokitusjärjestelmälle:
  - joissakin edellytetään, että luokat ovat erillisiä (ei yhteisiä ilmentymiä)
  - toisissa sallitaan päällekkäisyys (yhteiset ilmentymät)

**JSS Luokkahierarkia**

- Luokkahierarkiassa luokka voidaan määritellä toisen luokan alaluokaksi
- esim.
  - luokka nainen on luokan henkilö alaluokka
  - luokka johtaja on luokan henkilö alaluokka
  - luokat auto, laiva ja lentokone ovat luokan kulkuväline alaluokkia

**JSS** **Luokkahierarkia**

- Jos luokka A on luokan B alaluokka, siitä seuraa, että jokainen A:n ilmentymä on myös B:n ilmentymä.
  - **jokainen johtaja on myös henkilö**
- Tästä taas seuraa, että kaikki attribuutit, palvelut ja yhteydet, jotka on määritelty luokan B ilmentymille, liittyvät myös A:n ilmentymiin.

**JSS** **Luokkahierarkia**

- Jos henkilölle on määritelty attribuutti Nimi, niin myös johtajalla on automaattisesti Nimi-attribuutti.
- Jos henkilö on määritelty osapuoleksi työsuhde-yhteyteen, myös johtaja voi olla osapuolena työsuhde-yhteydessä.
- Tätä ilmiötä kutsutaan periytymiseksi (inheritance).

**JSS** **Luokkahierarkia**

- Periytymisessä yläluokkaan liitetyt attribuutit, palvelut ja yhteydet periytyvät alaluokalle.
- **Huom.** periytyminen on luokkien välistä määrittelyjen periytymistä - ilmentymät eivät peri mitään toisilta ilmentymiltä.

**JSS** **Luokkahierarkia**

- Yläluokan ja alaluokan välinen riippuvuussuhde kuvataan kaaviossa:

```
graph LR; Alaluokka --> Yläluokka; Johtaja --> Henkilö;
```

**JSS** **Luokkahierarkia**

- Alaluokkaan voidaan liittää yläluokalta perittyjen attribuuttien, palvelujen ja yhteyksien lisäksi omia attribuutteja, yhteyksiä ja palveluja.
  - **Johtajalla on ominaisuudet vastuualue ja alaisten määrä, joita ei ole henkilöllä yleisesti.**
  - **Vain johtajalla voi olla oikeus käyttää edustustiliä.**
  - **Johtajalla voi olla alaisia.**

**JSS** **Luokkahierarkia**

```
classDiagram; Johtaja <|-- Henkilö; Johtaja "1" -- "0..*" Henkilö : alainen; Johtaja "0..*" -- "0..1" Edustustili : Käyttöoikeus; Johtaja --> Johtaja : vastuualue, alaisten määrä;
```

**JSS Luokkahierarkia**

- Alaluokkaan voidaan liittää uusia palveluita, joita yläluokalla ei ole.
- Alaluokassa voidaan myös **syрjäyttää** (override) yläluokassa määritelty palvelu. Syрjäyttäminen on sisällön uudelleenmäärittelyä.
- Henkilöllä voisi olla palvelu viikkoraportti:
  - kerro ajankäyttö työtehtäviin
- Johtajan viikkoraportti-palvelun sisältö voisi olla:
  - kerro ajankäyttö työtehtäviin
  - laadi yhteenveto alaisten viikkoraporteista
  - raportoi edustustilain käyttö

**JSS Luokkahierarkia**

- Syрjäyttäminen on erityisesti olio-ohjelmoinnissa hyödyllinen tekniikka.
- Olio-ohjelmoinnin merkittävimmät hyödyt tulevat esiin juuri syрjäyttämismahdollisuuden kautta:
  - muokattavat kirjastopalvelut
  - yksinkertaisemmat ohjelmat

**JSS Luokkahierarkia**

- Tilannetta, jossa luokka on useamman kuin yhden luokan välitön alaluokka, kutsutaan **moniperiytymiseksi (multiple inheritance)**.
  - Kaikki olio-ohjelmointikieliet eivät tarjoa moniperintää (esim. Javassa ei ole, C++:ssa on).

```

classDiagram
    kenraali --|> johtaja
    kenraali --|> sotilas
    
```

**JSS Luokkahierarkia**

- Järjestelmää mallinnettaessa luokkahierarkiasta on eniten hyötyä:
  - sääntöjen ilmaisemisessa
  - kuvauksen ekonomisuudessa (ei tarvitse toistaa samoja asioita useassa luokassa)

**JSS Luokkahierarkia**

- Esimerkki säännöstä: Autolla täytyy olla omistaja. Omistaja voi olla yritys tai henkilö.

```

classDiagram
    yritys --|> mahdollinen_omistaja
    henkilo --|> mahdollinen_omistaja
    mahdollinen_omistaja "1..*" -- "0..*" auto
    
```

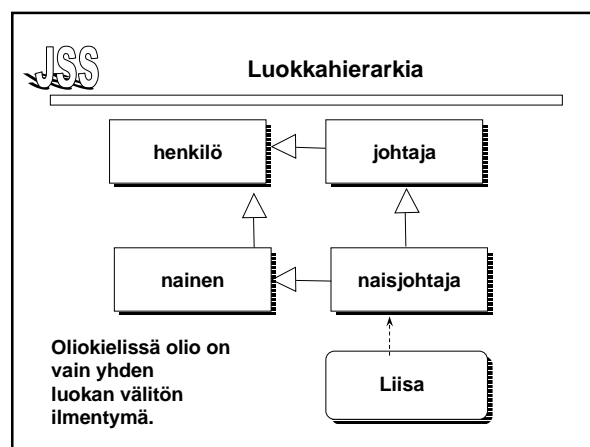
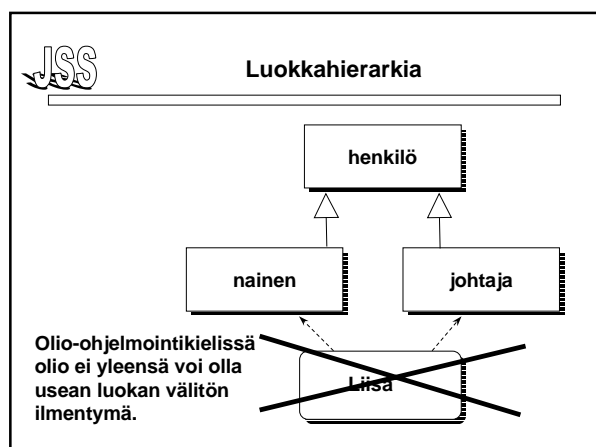
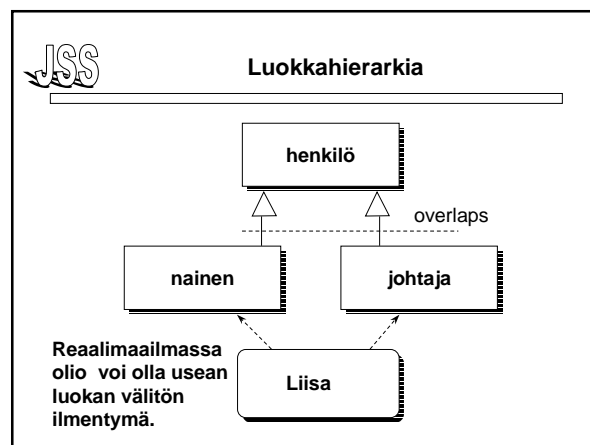
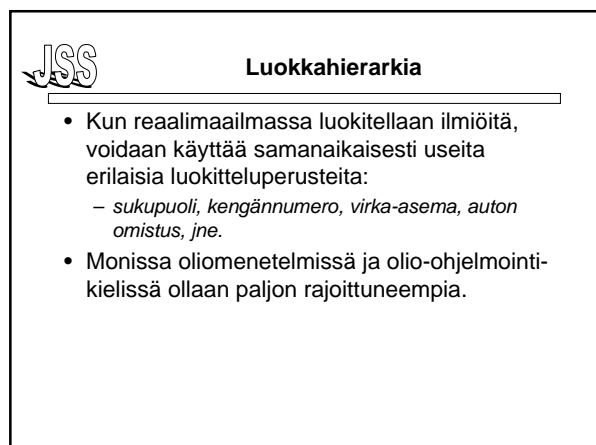
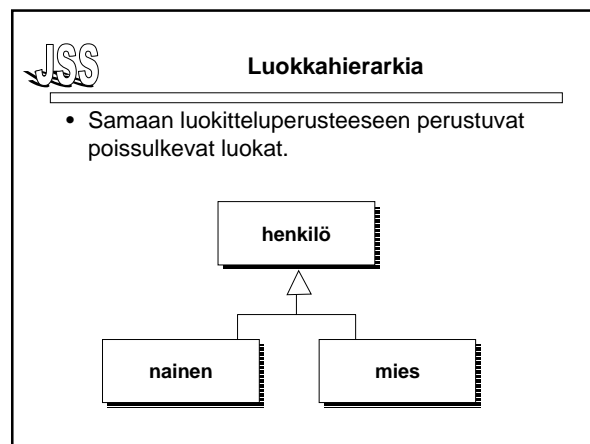
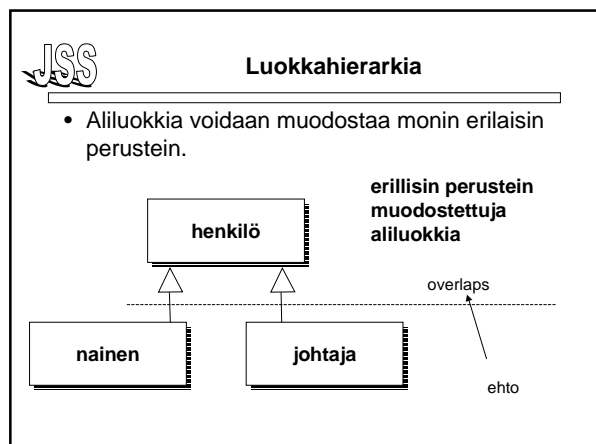
**JSS Luokkahierarkia**

- Ellei yläluokkaa käytettäisi:

```

classDiagram
    yritys "0..*" -- "0..*" auto : yritysomistus
    henkilo "0..*" -- "0..*" auto : henkilöomistus
    
```

Voiko autolla olla  
- sekä yritys- että henkilöomistaja?  
- nolla omistajaa?





### Luokkahierarkia

- Edellinen tilanne johtuu siitä, että olio-ohjelmoinnissa olio on esiteltävä johonkin (yhteen) luokkaan kuuluvaksi ja tämä luokka perii vain yläluokkiensa ominaisuudet.
- Kohdealueen analyysin kannalta rakenne on kömpelö.