Ohjelmistojen mallintaminen

Luento 6, 19.11.

Kertaus:
käsiteanalyysi eli menetelmä luokkamallin muodostamiseen

1. Etsi luokkaehdokkaat tekstikuvauksista (substantiivit)
2. Karsi luokkaehdokkaita (mm. yhdistä synonyymit)
3. Tunnista olioiden väliset yhteyksiä (verbit ja genetiivit vihjeenä)
4. Lisää luokille attribuutteja
5. Tarkenna yhteyksiä (kytkentärajoitteet, kompositiot)
6. Etsi ”rivien välissä” olevia luokkia
7. Etsi yläkäsitteitä (ensi viikolla)
8. Toista vaiheita 1-7 riittävän kauan

- Aloitetaan vaiheella 1, sen jälkeen edetään muihin vaiheisiin peräkkäin, rinnakkain tai/ja sekalaisessa järjestyksessä toistaen
- Lopputuloksena alustava toteutusriippumaton sovelluksen kohdealueen malli
  - Malli tulee tarkentumaan ja täsmentymään suunnitteluvaiheessa
  - Siksi ei edes kannata tähdätä ”täydelliseen” malliin

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

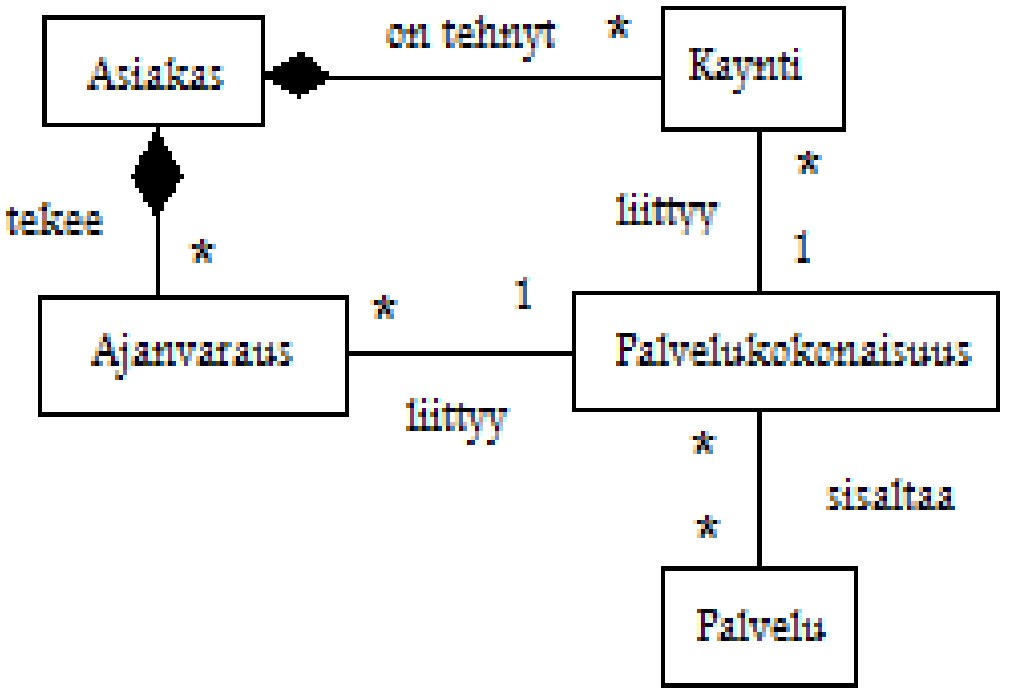
- ~~Kampaamo~~
  - *Liian yleinen*
- Varauspalvelu
- Asiakas, *attribuutteja:*
  - Asiakastunnus
  - Nimi
  - Osoite
  - Puhelinnumero
- Palvelu, *attribuutteja*:
  - Hinta
  - Kesto
- Hinnasto
- Ajanvaraus, *attribuutti:*
  - Toive
- ~~Toimenpide~~
  - *Palvelun synonyymi*
- Palvelukokonaisuus
- ~~Toimenpidekokoelma~~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

Alustavasti valitut luokat

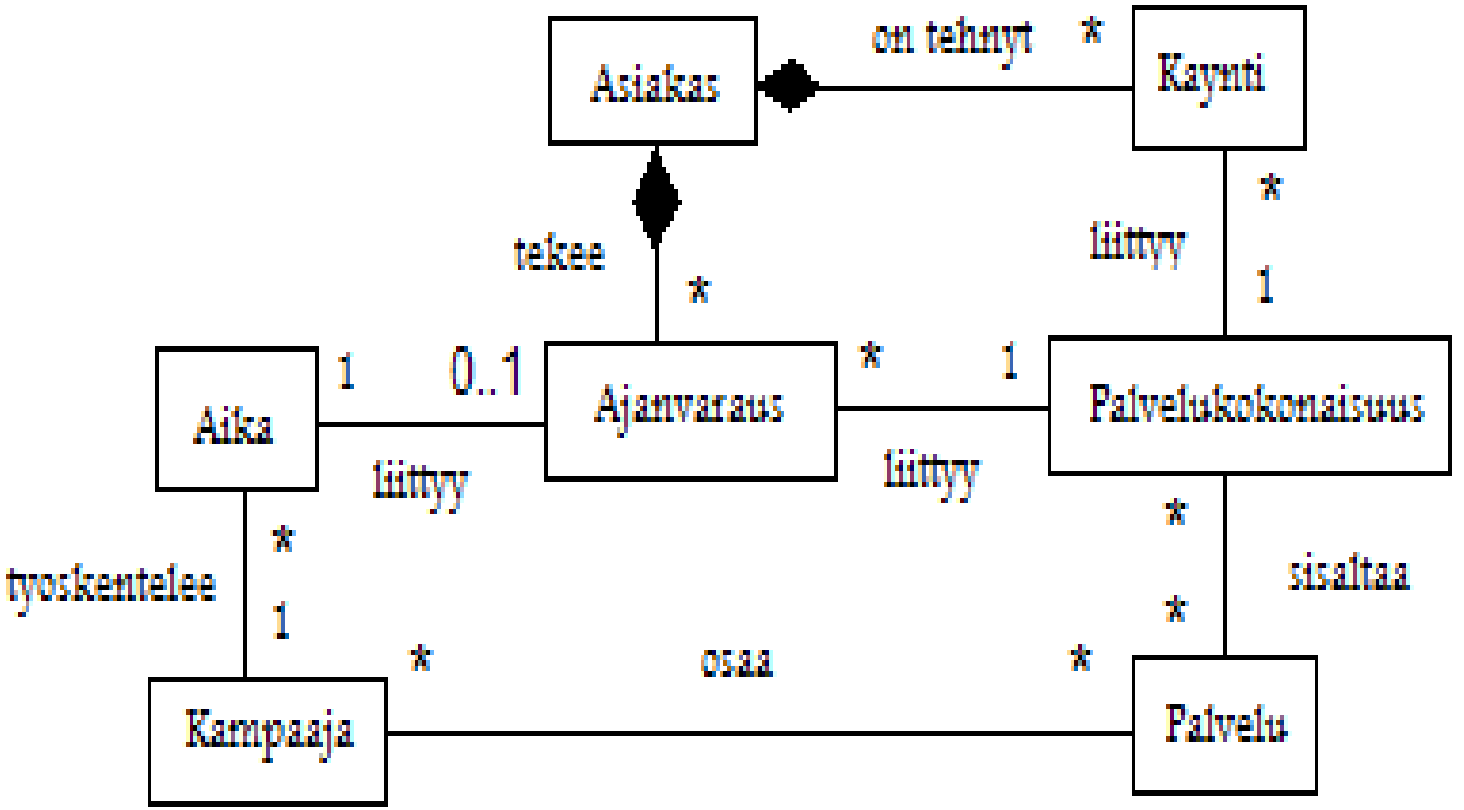
- Varauspalvelu
  - Itse järjestelmää kuvaava luokka
- Asiakas
- Palvelu
- Hinnasto
- Palvelukokonaisuus
- Ajanvaraus
- Käynti
- Kampaaja
- Aika

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



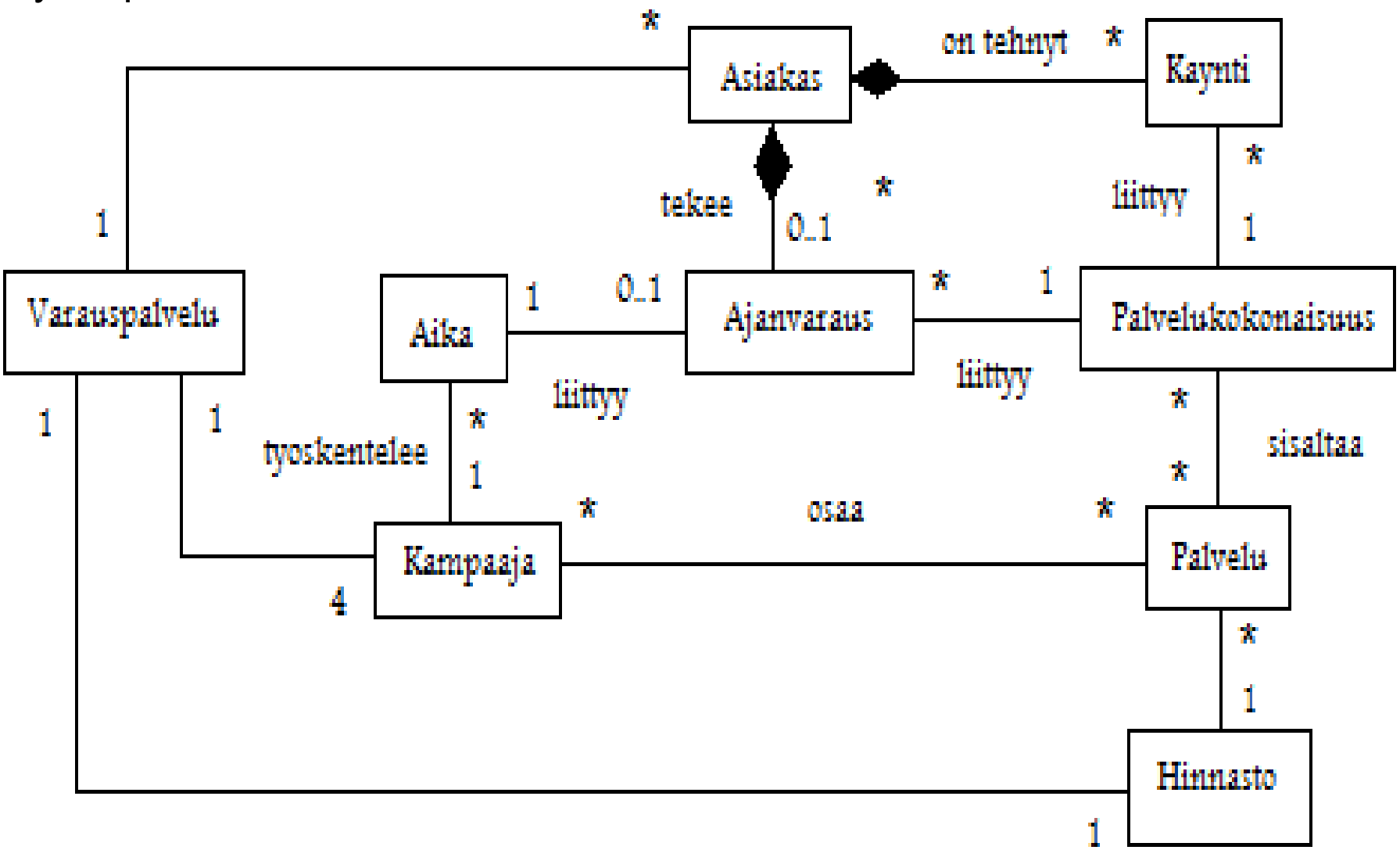
- Tarkastellaan luokkia aika, kampaaja
  - Kampaamossa työskentelee 4 kampaajaa, jotka ovat työssä epäsäännöllisesti. Kaikki kampaajat eivät tee kaikkia tarjolla olevia toimenpiteitä.
  - Varaustilanteessa järjestelmän pitää kyetä näyttämään asiakkaalle, milloin halutun palvelun suorittamaan pystyviltä 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



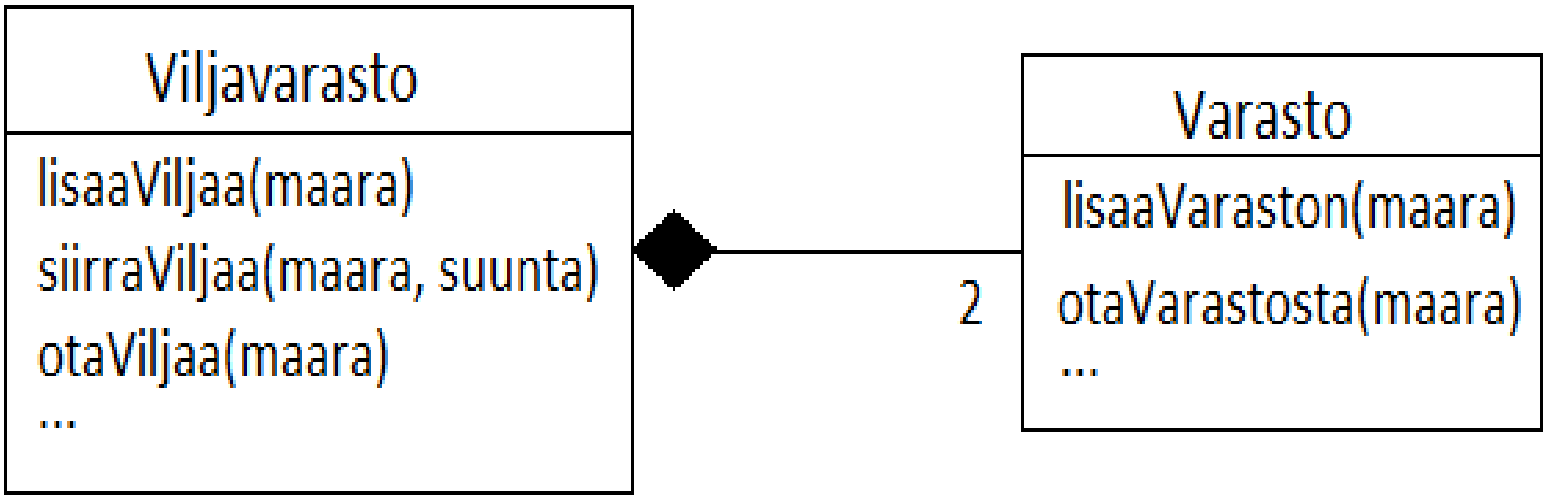
Mallinnuksen eteneminen

- Isoa ongelmaa kannattaa lähestyä pienin askelin, esim:
  - Yhteydet ensin karkealla tasolla, tai
  - Tehdään malli pala palalta, lisäten siihen muutama luokka yhteyksineen kerrallaan

- Mallinnus iteratiivisesti etenevässä ohjelmistokehityksessä
  - Ketterissä menetelmissä suositaan *iteratiivista* lähestymistapaa ohjelmistojen kehittämiseen
    - kerralla on määrittelyn, suunnittelun ja toteutuksen alla ainoastaan osa koko järjestelmän toiminnallisuudesta
  - Jos ohjelmiston kehittäminen tapahtuu ketterästi, kannattaa myös ohjelman luokkamallia rakentaa iteratiivisesti
  - Eli jos ensimmäisessä iteraatiossa toteutetaan ainoastaan muutaman käyttötapauksen kuvaama toiminnallisuus, esitetään iteraation luokkamallissa vain ne luokat, jotka ovat merkityksellisiä tarkastelun alla olevan toiminnallisuuden kannalta
  - Luokkamallia täydennetään myöhempien iteraatioiden aikana niiden mukana tuoman toiminnallisuuden osalta

Olioiden yhteistyön mallintaminen

- Luokkakaaviosta käy hyvin esille ohjelman *rakenne*
  - minkälaisia luokkia on olemassa
  - miten luokat liittyvät toisiinsa
- Entä ohjelman toiminta?
  - Luokkakaaviossa voi olla metodien nimiä
  - Pelkät nimet eivät kuitenkaan kerro juuri mitään
- Ohjelman toiminnan kuvaamiseen tarvitaan jotakin muuta
  - Tarve esim. kuvata skenaario ”Viljavarastoon lisätään 500 kiloa viljaa siten, että ne varastoidaan toiseen varastoon” (lisääViljaa tekee lisäyksen aina ensimmäiseen varastoon...)



Olioiden yhteistyö

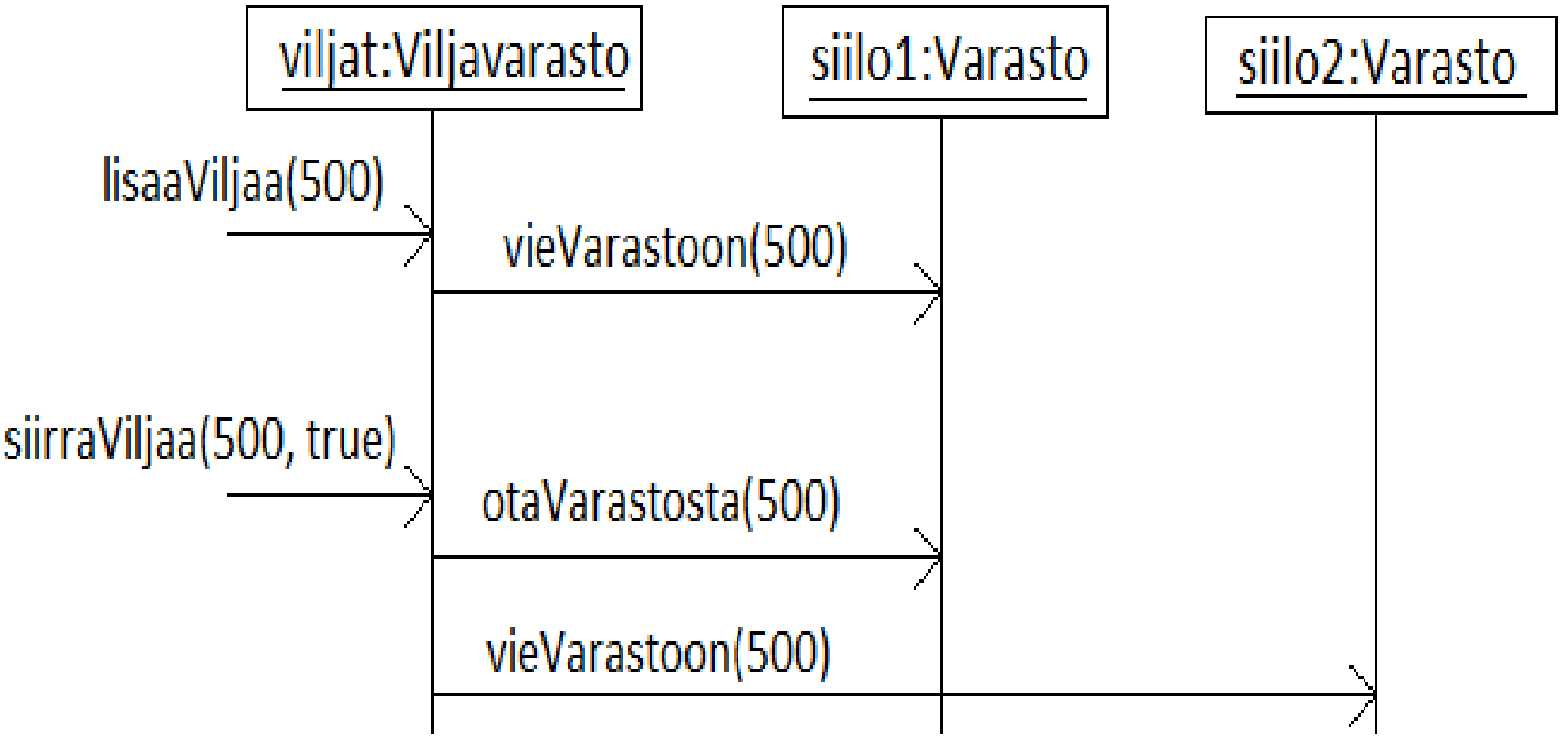
- Oliopohjaisuus siis perustuu seuraavaan olettamukseen:
  - *Minkä tahansa järjestelmän katsotaan voivan muodostua olioista, jotka yhteistyössä toimien ja toistensa palveluja hyödyntäen tuottavat järjestelmän tarjoamat palvelut*

- Koska järjestelmän toiminnan kulmakivenä on järjestelmän sisältämien olioiden yhteistyö, *tarvitaan menetelmä yhteistyön kuvaamiseen*
- UML tarjoaa kaksi menetelmää, joita kohta tarkastelemme:
  - sekvenssikaavio
  - kommunikaatiokaavio

- Huomionarvoista on, että luokkakaaviossa tarkastelun pääkohteena olivat luokat ja niiden suhteen. Yhteistyötä mallintaessa taas fokuksessa ovat oliot eli luokkien instanssit
  - Luokkahan ei tee itse mitään, ainoastaan oliot voivat toimia

Sekvenssikaavio

- Palataan skenaarioon ”Viljavarastoon lisätään 500 kiloa viljaa siten että ne varastoidaan toiseen varastoon”
  - Lukemalla koodia (ks. mallivastaus ohpe viikko 4) huomataan, että vilja on vietävä ensi ensimmäiseen varastoon, josta se siirretään toiseen varastoon
- Tilanteen kuvaava *sekvenssikaavio* alla



Sekvenssikaavio

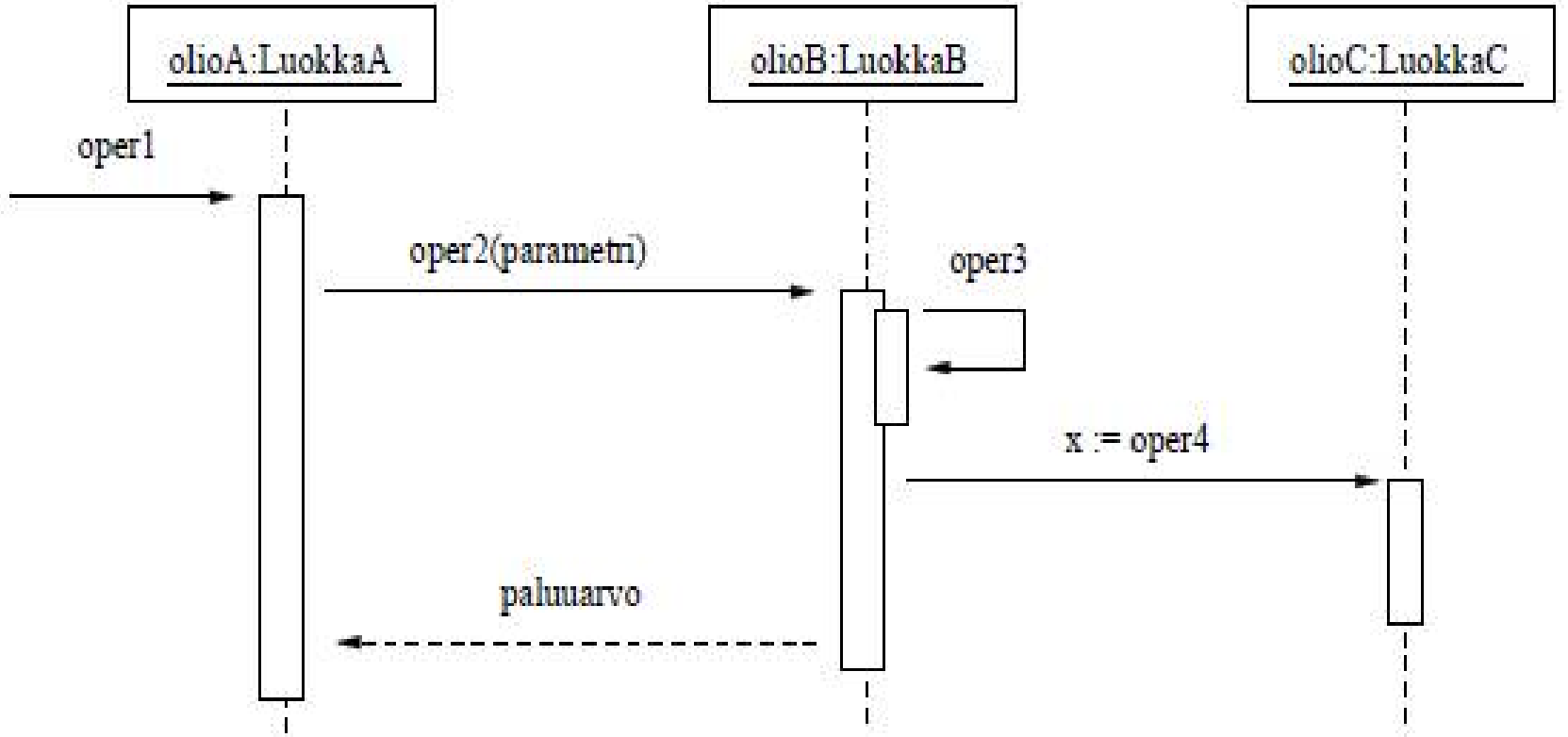
- Sekvenssikaaviossa kuvataan tarkasteltavan skenaarion aikana tapahtuva olioiden vuorovaikutus
- Oliot esitetään kuten oliokaaviossa, eli laatikkoina, joissa alleviivattuna olion nimi ja tyyppi
- Sekvenssikaaviossa oliot ovat (yleensä) ylhäällä rivissä
- Aika etenee kaaviossa alaspäin
- Jokaiseen olioon liittyy katkoviiva eli elämänviiva (engl. lifeline), joka kuvaa sitä, että olio on olemassa ”ylhäältä alas asti”
- Metodikutsu piirretään nuolena, joka kohdistuu kutsuttavan olion elämänlankaan
  - Usein kutsujana joku toinen olio
  - Joskus kutsu tulee kuvattavien olioiden ulkopuolelta määrittelemättömästä kohteesta (esim. viljavaraston käyttäjältä)
- Tyypillisesti yksi sekvenssikaavio kuvaa järjestelmän yksittäisen toimintaskenaarion
  - Jokaiselle toimintaskenaariolle tarvitaan oma sekvenssikaavio

Viljavaraston sekvenssikaavio

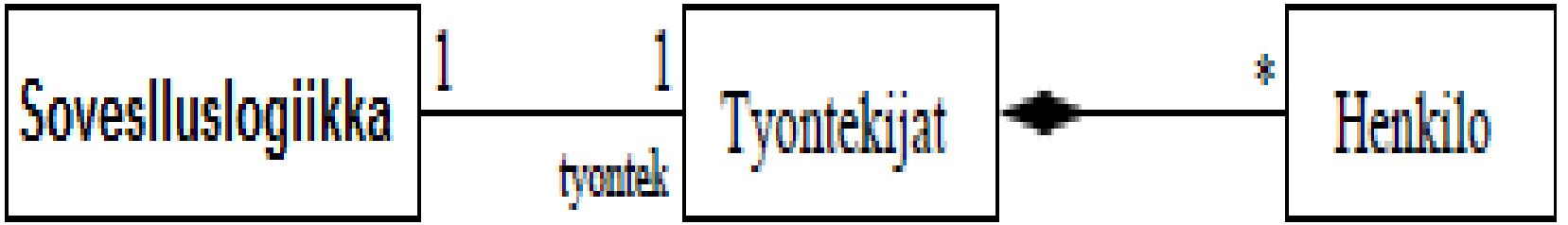
- Esimerkissä toiminta alkaa sillä, että joku (esim. pääohjelma) kutsuu Viljavarasto-olion metodia lisääViljaa(500)
  - Seurauksena Viljavarasto-olio kutsuu ensimmäisen Vvarasto-olion (nimeltään siilo1) metodia vieVarastoon(500)
- Toiminta jatkuu kun Viljavarasto-olion metodia siirräViljaa(500,true) kutsutaan
  - Tästä seurauksena Viljavarasto-olio ottaa siilo1:stä 500 kiloa ja siirtää ne siiloon2
- Lopulta siis 500 kiloa viljaa on siirtynyt toiseen Varastoon

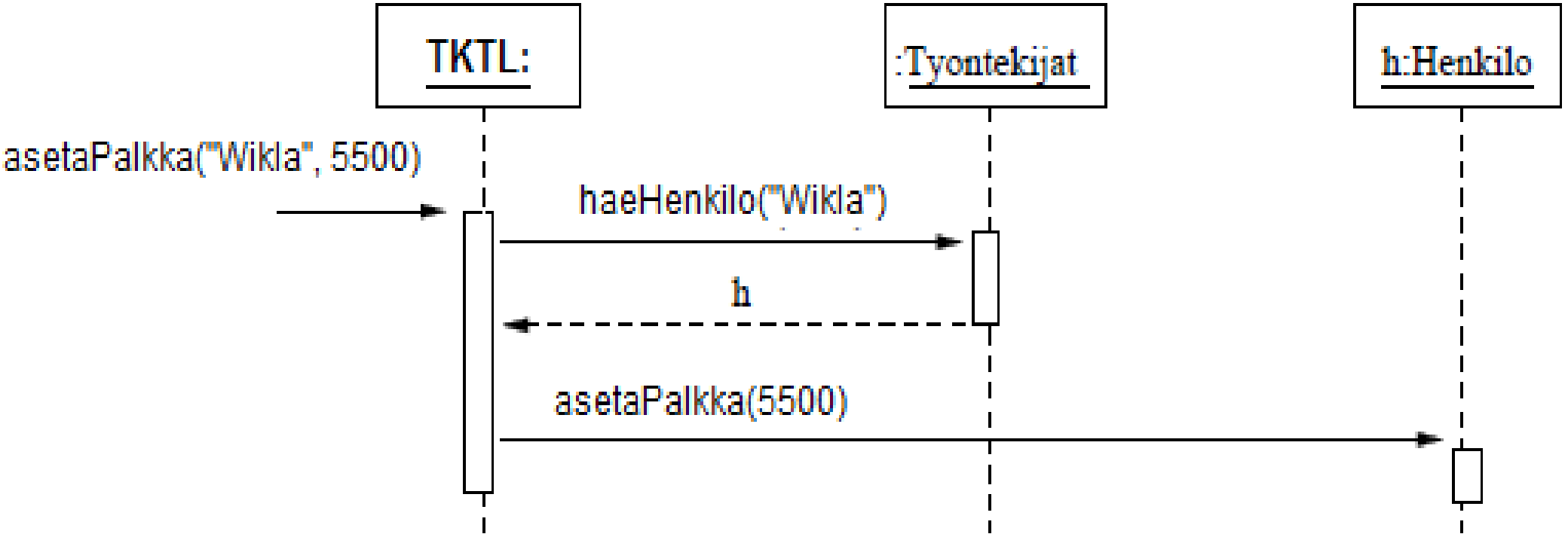
Lisää syntaksia

- Seuraavat käsitteet on selitetty monisteessa. Selitys tulee luennolla, mutta en toista samaa tähän kalvolle.
  - Joskus hyödyllistä piirtää *aktivaatiopalkki*
  - Olion *oman metodin* kutsu
  - Kaksi tapaa *paluuarvon ilmaisemiseen*



- Esimerkkisovellus:
  - Työntekijät-olio pitää kirjaa työntekijöistä, jotka Henkilö-oliota
  - Sovelluslogiikka-olio hoitaa korkeamman tason komentojen käsittelyn
- Tarkastellaan operaatiota lisaaPalkka(nimi, palkka)
  - Lisätään parametrina annetulle henkilölle uusi palkka
- Suunnitellaan, että operaatio toimii seuraavasti:
  - Ensin sovelluslogiikka hakee Tyontekijat-oliolta viitteen Henkilo-olion
  - Sitten sovelluslogiikka kutsuu Henkilo-olion palkanasetusmetodia
- Seuraavalla sivulla operaation suoritusta vastaava sekvenssikaavio
  - Havainnollistuksena myös osa luokan Sovelluslogiikka koodista






```
class Sovelluslogiikka{
    Tyontekijat tyontek;     // attribuutti, jonka kautta sovelluslogiikka tuntee työntekijät
    void lisaaPalkka(String nimi, int palkka ){
        Henkilo h = tyontek.haeHenkilo( nimi );
        h.asetaPalkka( palkka );
    }
}
```

Oliosuunnittelua!

- Tässä oli oikeastaan jo kyse oliosuunnittelusta
  - Alunperin oli ehkä päätetty luokkarakenne
  - Tiedettiin, että tarvitaan toiminto, jolla lisätään henkilölle palkka
  - Suunniteltiin, miten palkan asettaminen tapahtuu olioiden yhteistyönä
  - Suunnittelu tapahtui ehkä sekvenssikaaviota hyödyntäen
  - Siitä saatiin helposti aikaan koodirunko
- Sekvenssikaaviot ovatkin usein käytössä oliosuunnittelun yhteydessä
  - Kuten kohta näemme, voidaan niitä käyttää myös määrittelyssä kuvaamaan käyttötapauksen kulkua
- huomaa parametrin h käyttö edellisen sivun kuvassa
- haeHenkilo()-metodikutsun paluuarvo on h
- Kyseessä on sama h, joka on sekvenssikaaviossa esiintyvän (Wiklan tiedot sisältävän) olion nimi!

Käyttötapaus ja sekvenssikaavio

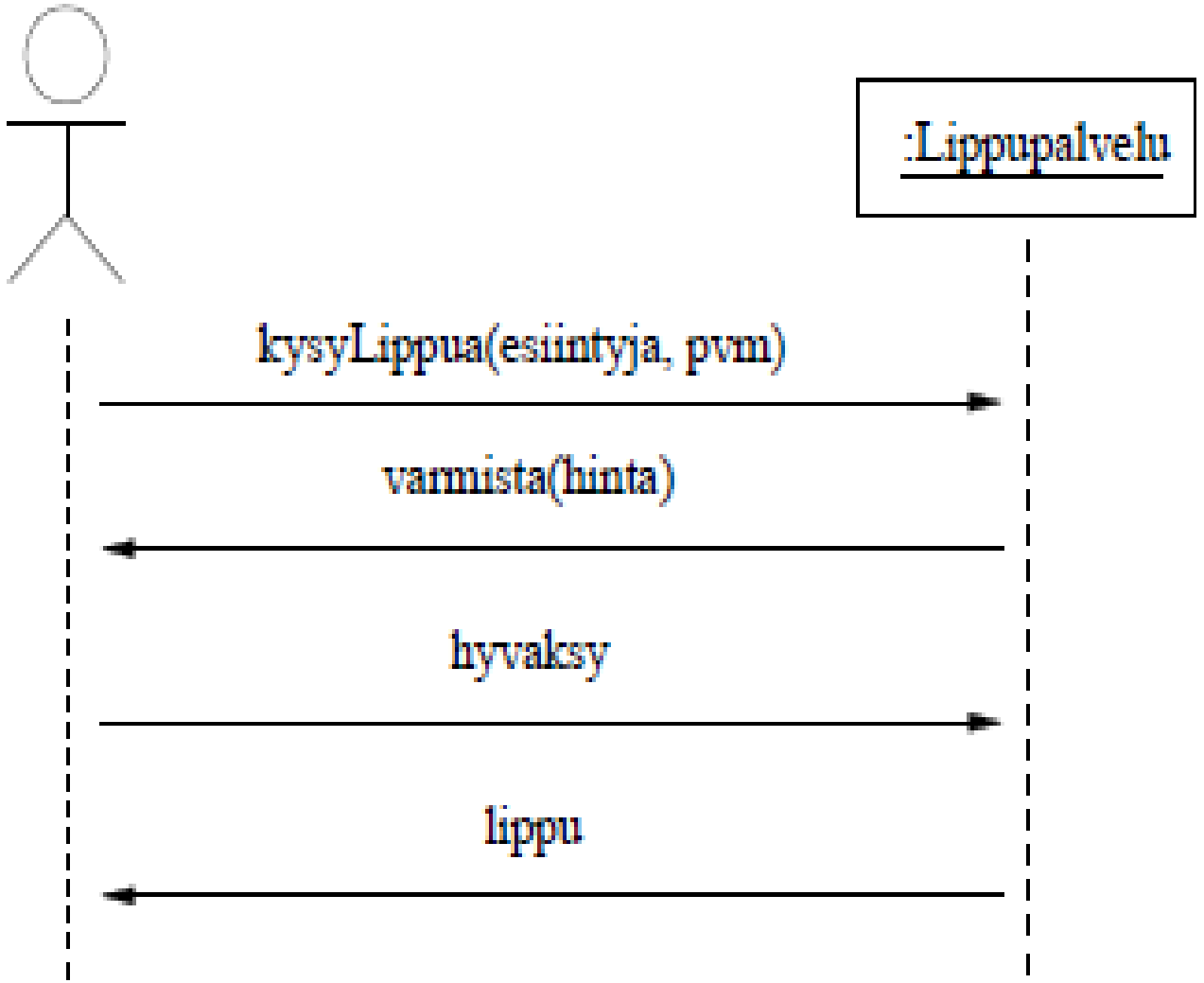
- Tarkastellaan alkeellista lippupalvelun tietojärjestelmää ja sen käyttötapausta

Lipun varaus, tilanne missä lippuja löytyy

- Käyttötapauksen kulku:
  1. Käyttäjä kertoo tilaisuuden nimen ja päivämäärän
  2. Järjestelmä kertoo, minkä hintainen lippu on mahdollista ostaa
  3. Käyttäjä hyväksyy lipun
  4. Käyttäjälle annetaan tulostettu lippu

- Käyttötapauksen kulun voisi kuvata myös sekvenssikaavion avulla ajatellen koko järjestelmän yhtenä oliona
  - *Järjestelmätason sekvenssikaavio*

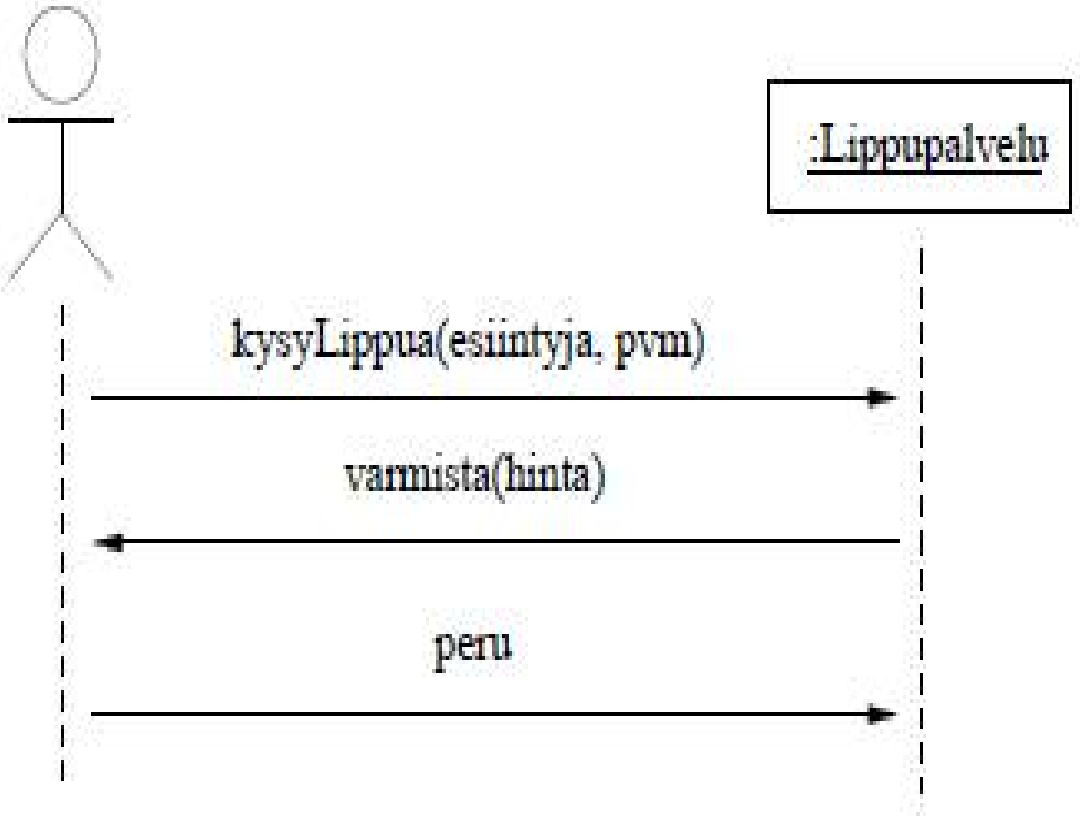
Käyttötapauksen *Lipun varaus* kuvaava sekvenssikaavio



- Huom: Olioiden aktivaatiopalkit on jätetty kuvaamatta, sillä niille ei ole tarvetta esimerkissä

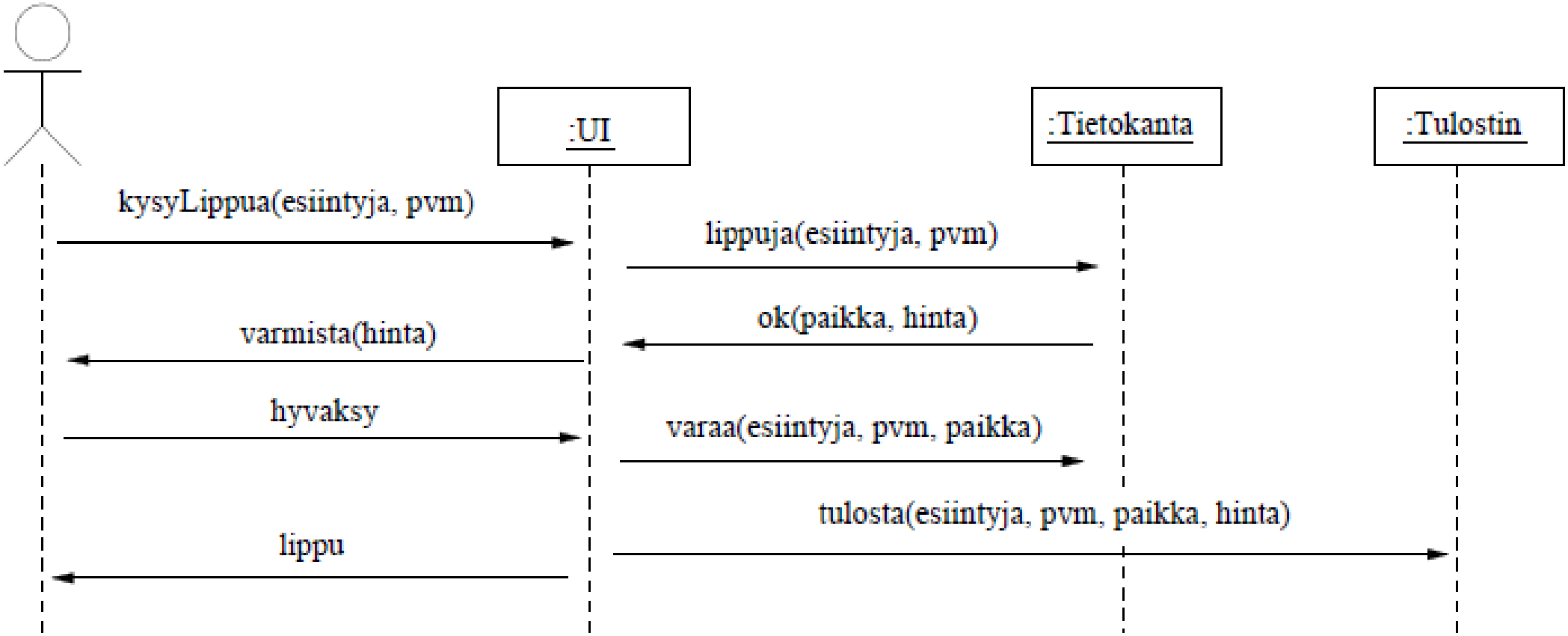
Vaihtoehtoinen skenaario

- Kuten kohta huomaamme, on myös yhteen sekvenssikaavioon mahdollista sisällyttää valinnaisuutta
- Toinen, usein selkeämpi vaihtoehto on kuvata vaihtoehtoiset skenaariot omina kaavioinaan
- Alla järjestelmätason sekvenssikaaviona tilanne, jossa asiakas hylkää tarjotun lipun



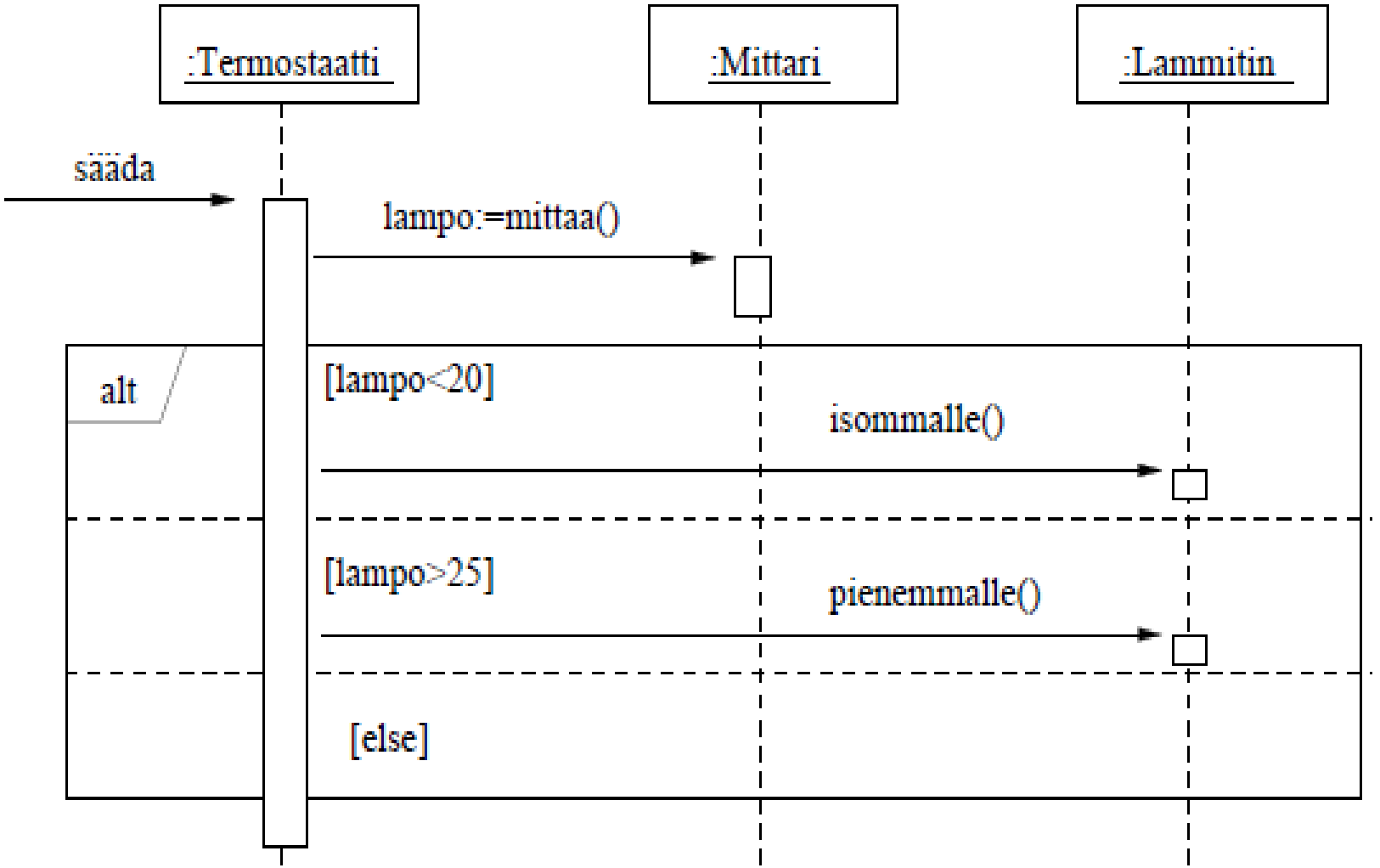
Järjestelmätasolta suunnittelutasolle

- Järjestelmätason sekvenssikaaviosta käy selkeästi ilmi käyttäjän ja järjestelmän interaktio
- Järjestelmän sisälle ei vielä katsota
- Seuraava askel on siirtyä suunnitteluun ja tarkentaa miten käyttötapauksen skenaario toteutetaan suunniteltujen olioiden yhteistyönä
- Alla yksinkertaistettu esimerkki, miten lippupalvelu voisi olla toteutettu:



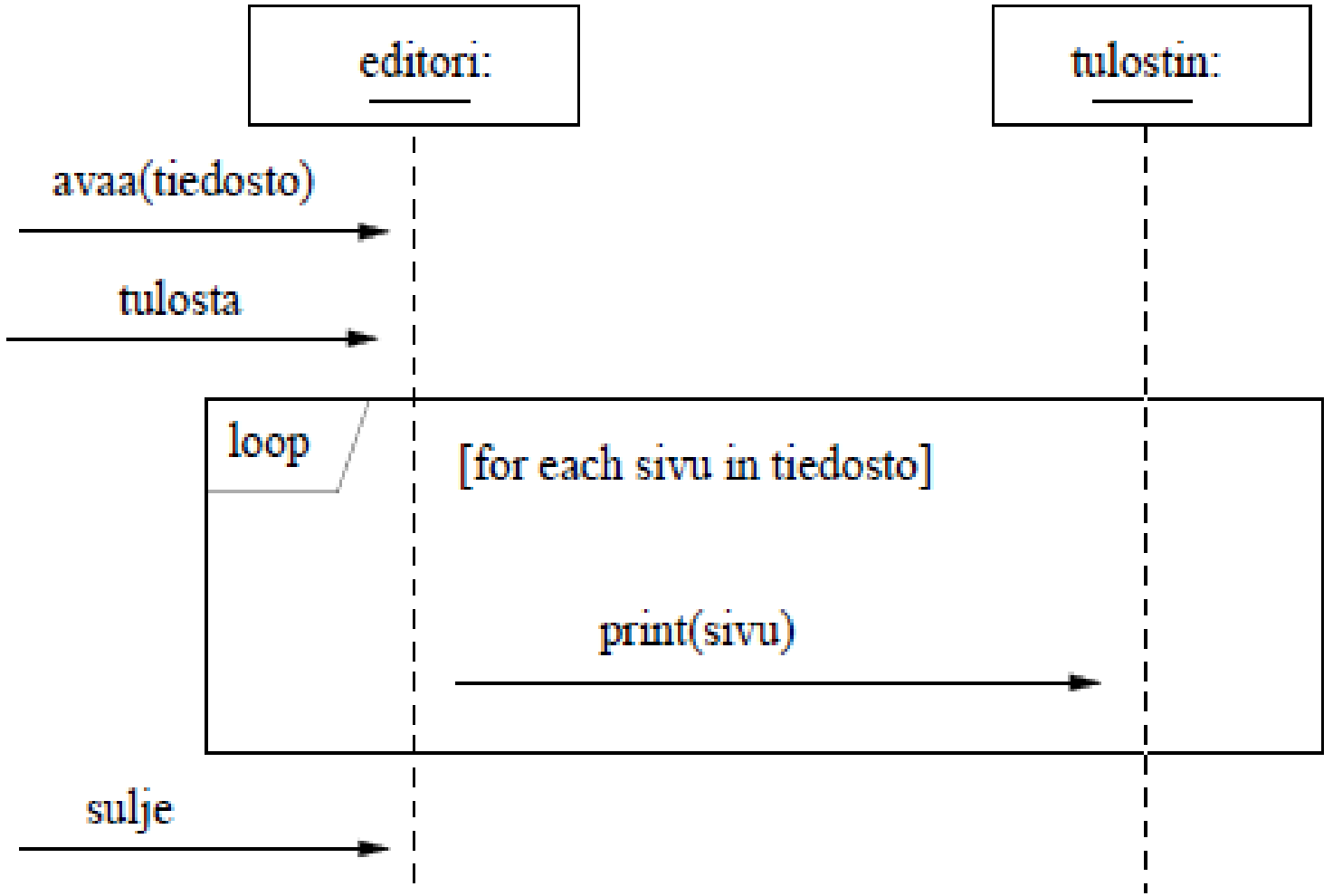
Valinnaisuus sekvenssikaaviossa

- Kaavioihin voidaan liittää lohko, jolla kuvataan valinnaisuutta
  - Vähän kuin if-else
  - Eli parametrina saadun arvon perusteella valitaan jokin kolmesta katkoviivan erottamasta alueesta



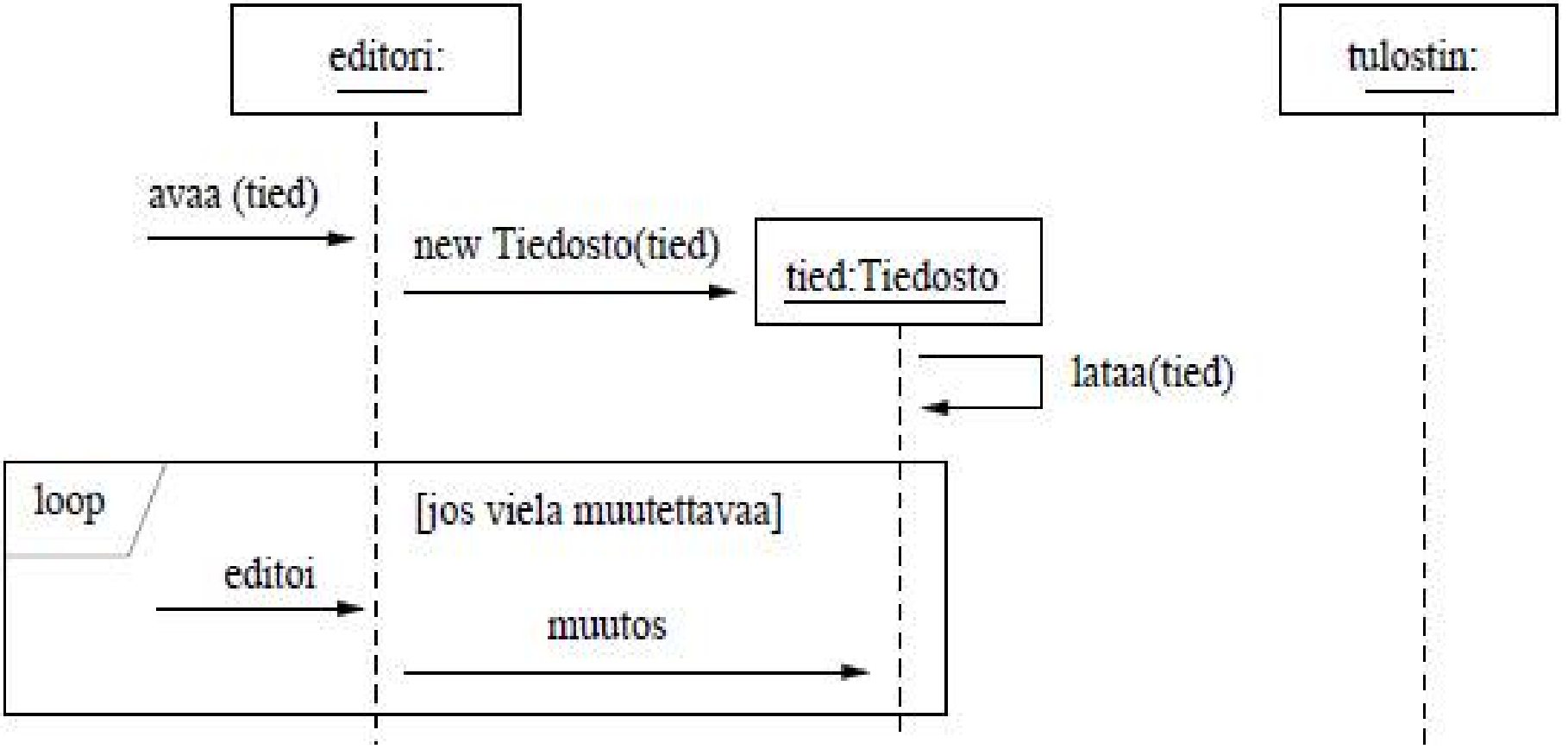
Toisto

- Myös toistolohko mahdollinen (vrt. for, while tai do-while)
  - Huomaa miten toiston määrä on ilmaistu [ja] -merkkien sisällä
  - Voidaan käyttää myös vapaamuotoisempaa ilmausta, kuten ”tulostetaan kaikki sivut erikseen”

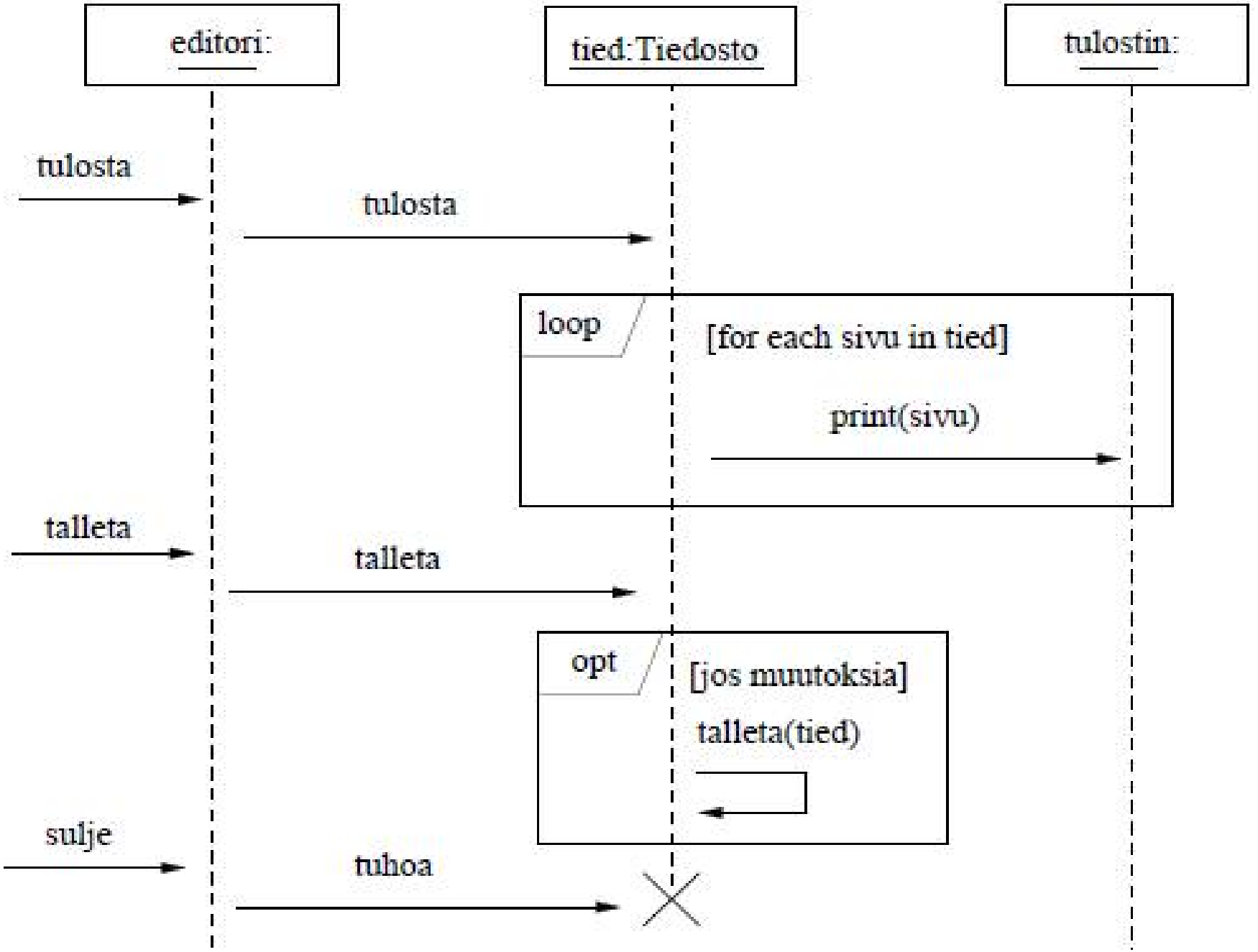


Olioiden luominen ja tuhoaminen

- Kun tiedosto avataan, luo editori ensin tiedostolle olion
- Tiedosto-olio lataa tiedoston sisällön levyltä kutsumalla omaa metodiaan
- *Huomaa kuinka olion luominen merkitään*
  - Uusi olio ei aloita ylhäältä vaan vasta siitä kohtaa milloin se luodaan
- Kaavio jatkuu seuraavalla sivulla, jossa nähdään miten olion tuhoutuminen merkitään

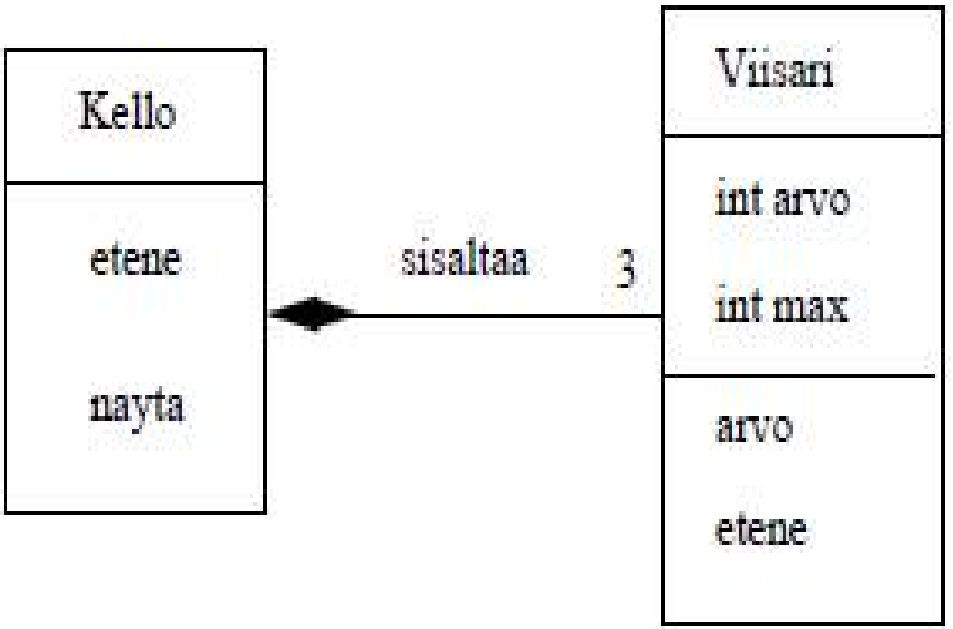


- Editoriesimerkki jatkuu
- Mukana myös *valinnainen (opt) lohko*, joka suoritetaan jos ehto tosi

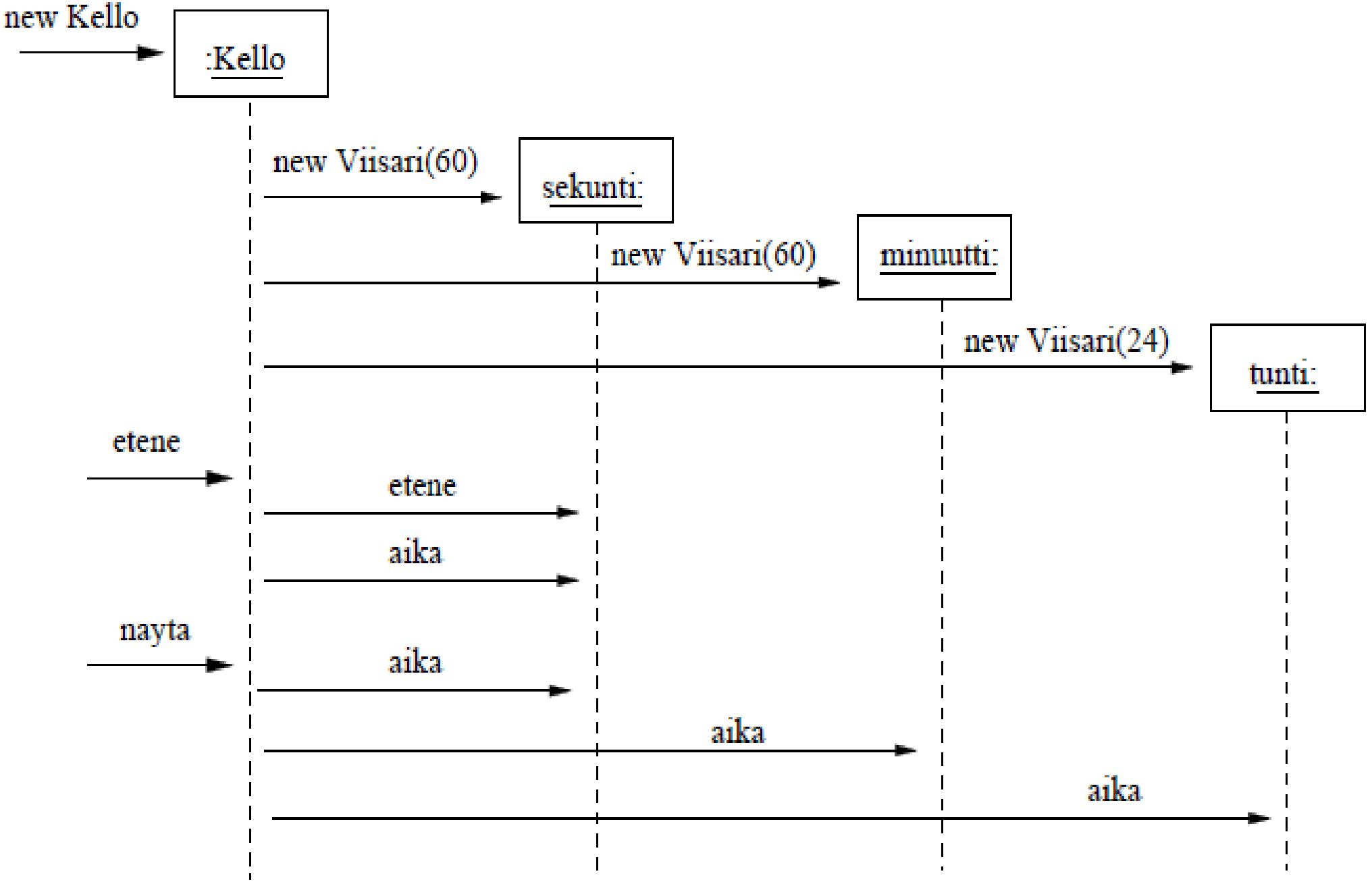


Takaisinmallinnus

- *Takaisinmallinnuksella* (engl. reverse enginering) tarkoitetaan mallien tekemistä valmiina olevasta koodista
  - Erittäin hyödyllistä, jos esim. tarve ylläpitää huonosti dokumentoitua koodia
- Monisteesta löytyy Javalla toteutettu kello, joka nyt takaisinmallinnetaan
- Luokkakaavio on helppo laatia
  - Kello koostuu kolmesta viisarista
- Luokkakaaviosta ei vielä saa kuvaa kellon toimintalogiikasta joten tarvitaan sekvenssikaavioita

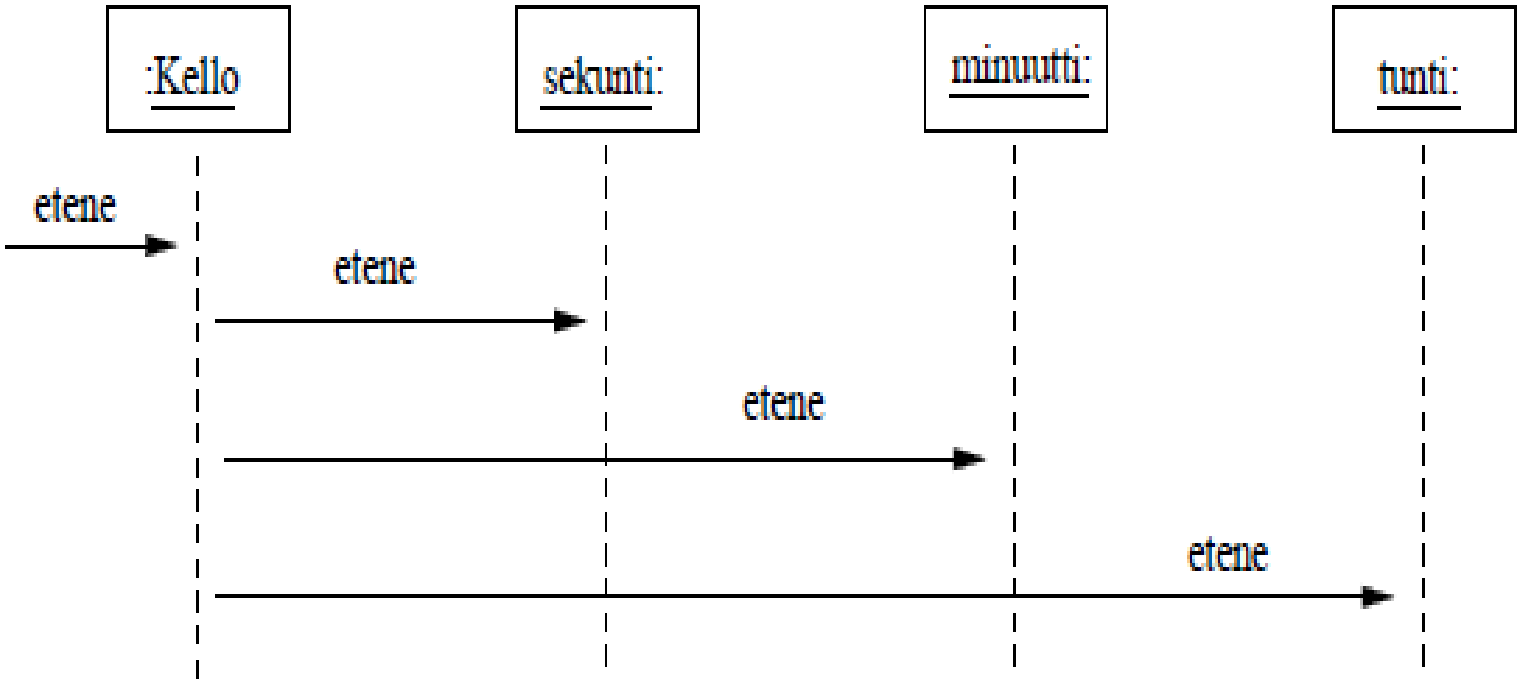


Kellon syntyy ja lähtee käymään



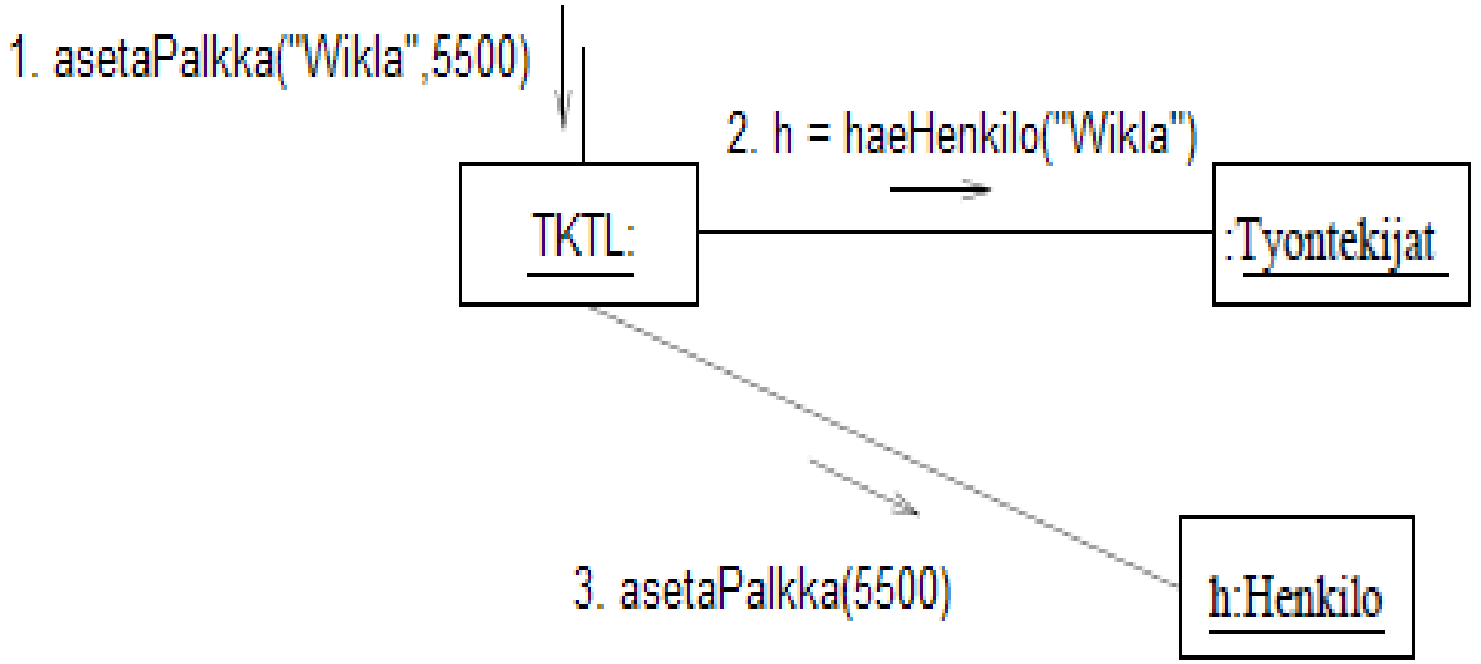
Kellon eteneminen tasatunnilla

- Keskiyöllä kaikki viisarit pyörähtävät eli ”etenevät” nollaan, tilannetta kuvaava sekvenssikaavio alla
- Kaaviosta jätetty pois aika()-metodikutsut
- Samoin edelliseltä sivulta on jätetty pois Java-standardikirjaston out-oliolle suoritetut print()-metodikutsut
- Eli jotta sekvenssikaavio ei kasvaisi liian suureksi, otetaan mukaan vain olennainen

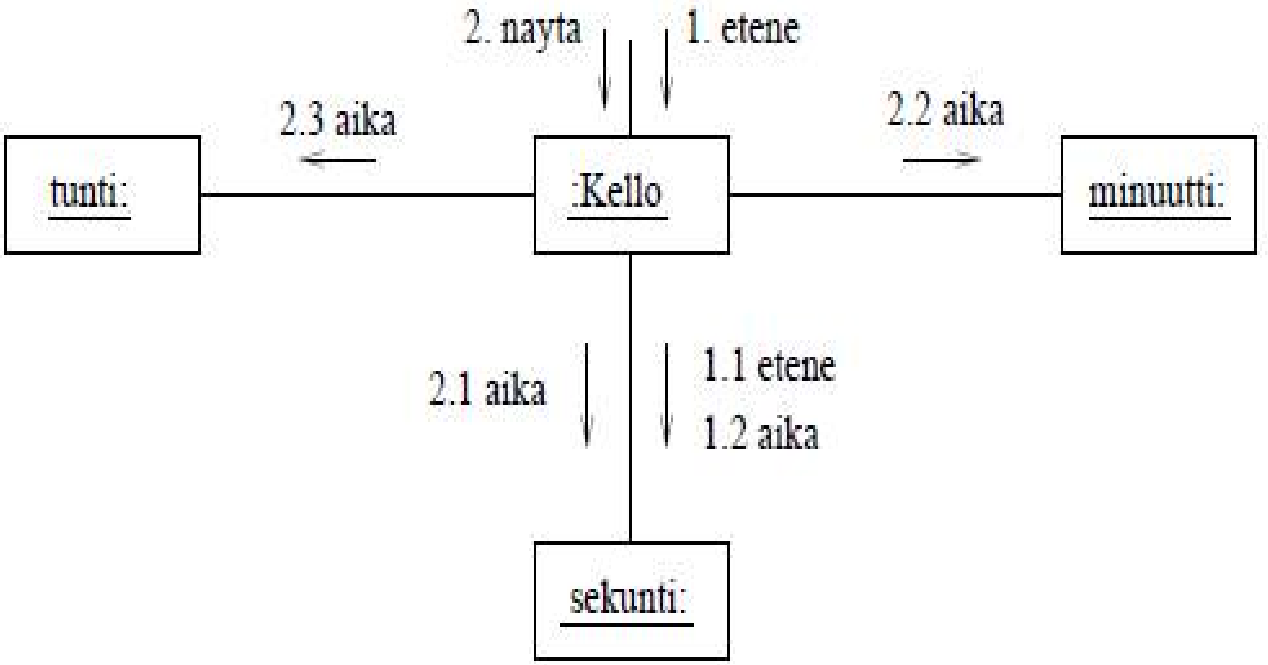


Kommunikaatiokaavio

- Toinen tapa olioiden yhteistyön kuvaamiseen on kommunikaatiokaavio (communication diagram)
- Alla muutaman sivun takainen esimerkki, jossa henkilölle asetetaan palkka
- Mukana skenaarioon osallistuvat oliot
  - Olioiden sijoittelu on vapaa
  - Kommunikoivien olioiden väliin on piirretty viiva
  - Muistuttaa oliokaaviota!
- Metodien suoritusjärjestys ilmenee numeroinnista



- Viestien järjestyksen voi numeroida juoksevasti: 1, 2, 3, …
- Tai allaolevan esimerkin (vanha tuttu Kello) tyyliin hierarkkisesti:
  - Kellolle kutsutaan metodia etene(), tällä numero 1
  - Eteneminen aiheuttaa sekuntiviisarille suoritetut metodikutsut etene() ja näytä(), nämä numeroitu 1.1 ja 1.2
  - Seuraavaksi kellolle kutsutaan metodia näytä(), numero 2
  - Sen aiheuttamat metodikutsut numeroitu 2.1, 2.2, 2.3, ...



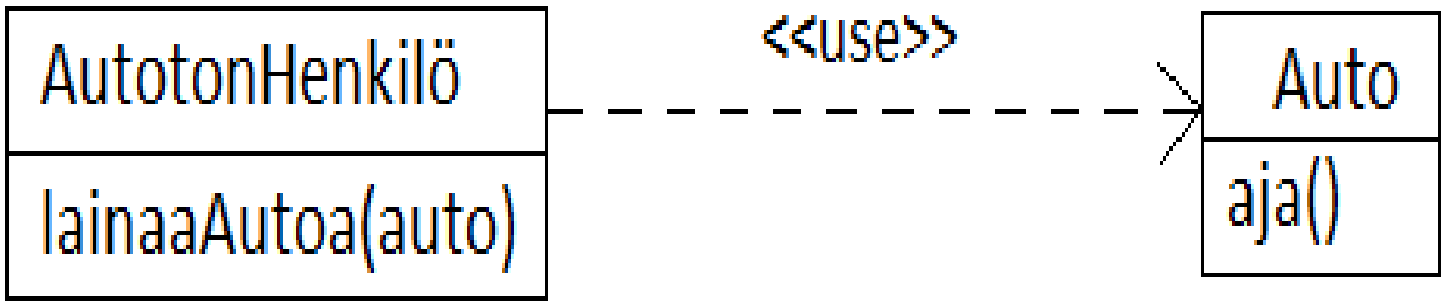
Yhteenveto olioiden yhteistoiminnan kuvaamisesta

- Sekvenssikaavioita käytetään useammin kun kommunikaatiokaavioita
  - Sekvenssikavio lienee luokkakaavioiden jälkeen eniten käytetty UML-kaaviotyyppi
- Sekä sekvenssi- että kommunikaatiokaavioilla erittäin tärkeä asema oliosuunnittelussa
- Kaaviot kannattaa pitää melko pieninä ja niitä ei kannata tehdä kuin järjestelmän tärkeimpien toiminnallisuuksien osalta
  - Kommunikaatiokaaviot ovat yleensä hieman pienempiä, mutta toisaalta metodikutsujen ajallinen järjestys ei käy niistä yhtä hyvin ilmi kuin sekvenssikaavioista
- On epäselvää missä määrin sekvenssikaavioiden valinnaisuutta ja toistoa kannattaa käyttää

- Sekvenssikaaviot on alunperin kehitetty tietoliikenneprotokollien kuvaamista varten

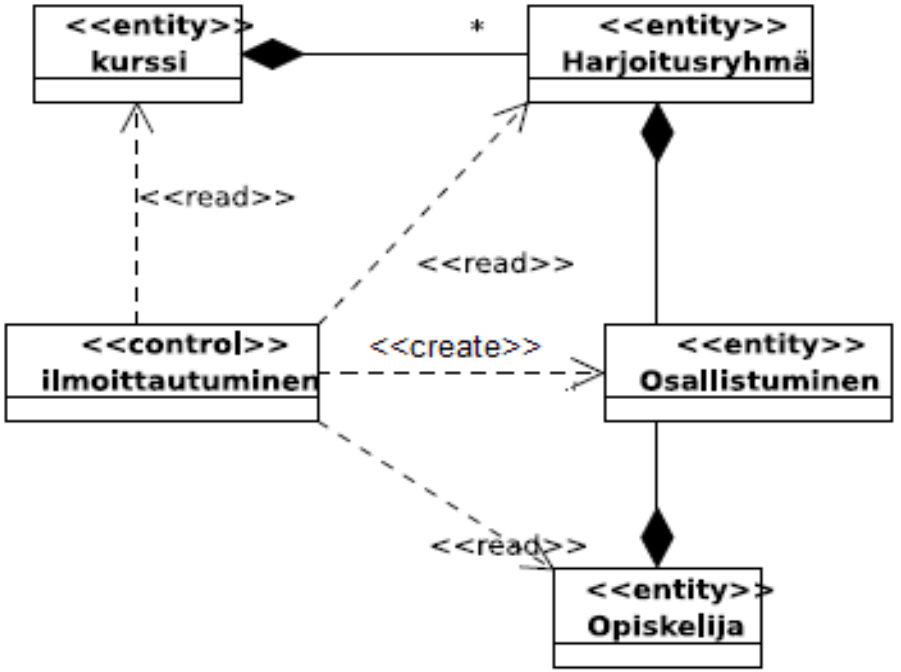
UML:n erikoistaminen

- Olemme jo pariin kertaan törmänneet kaavioissa merkkien << ja >> sisään kirjoitettuun tekstiin
  - Alla muistutuksena tilanne, jossa AutotonHenkilo- ja Auto-luokkien välisen *riippuvuuden* (katkoviiva) luonne on tarkennettu
- Kyseessä on UML:n peruskäsitteen riippuvuus laajennus erikoiskäsitteeksi, eli riippuvuudeksi joka johtuu siitä, että kohteena olevan luokan olioa käytetään parametrina
- Laajennus ilmaistaan *stereotyyppinä*, jolla tarkoitetaan tietyn lisämerkityksen antamista halutulle symbolille
  - Stereotyyppi merkitään kaavioon << ja >> välissä olevana ”kuvaavana” sanana



UML:n erikoistaminen

- Edellisessä esimerkissä stereotyyppi liittyy luokkien väliseen riippuvuuteen eli se tarkentaa minkälaisesta riippuvuudesta on kyse
- Stereotyyppi voi liittyä oikeastaan mihin tahansa UML-symboliin tai symbolin osaan
- Alla esimerkki, jossa stereotyyppi tarkentaa luokan roolia:
  - Luokka on joko *entity* eli tietosisältöluokka, *control* eli toiminnanohjausluokka
  - Riippuvuuden yhteydessä on myös tarkennettu riippuvuuden laatu



UML:n erikoistaminen

- UML:ssä on joukko valmiiksi määriteltyjä stereotyyppejä (kuten create)
- Stereotyyppejä voi määritellä halutessaan myös itse
- Stereotyypin voikin ajatella kommentin tapaiseksi lisäselitykseksi, jolla on ”hyvin määritelty” merkitys omissa kaavioissa
  - Eli jos tarvetta, kannattaa tarkentaa mitä määritellyllä stereotyypillä tarkoitetaan
- Erikoistetuille käsitteille voidaan myös määritellä oma graafinen symboli
  - Oikeastaan käyttötapauskaavion tikku-ukko on UML-standardin itse määrittelemä graafinen laajennussymbooli