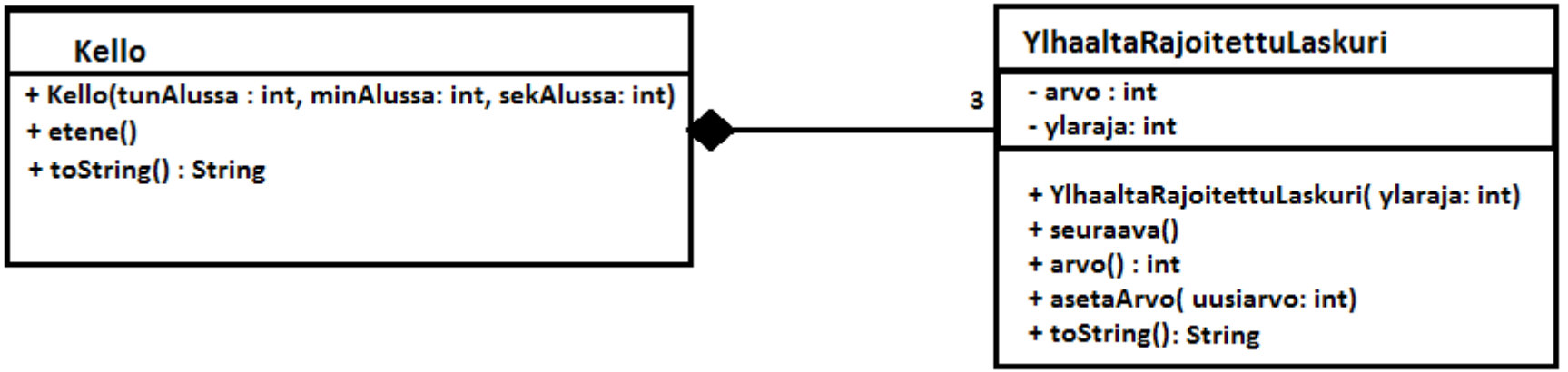
Ohjelmistojen mallintaminen

Luento 3, 12.11.

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 **opintorekisteri** on käyttäjä
- Sana actor olisi ehkä parempi kääntää *toimijaksi*, joka sekään ei ole optimaalinen, sillä ”käyttäjä” voi myös olla passiivinen osapuoli käyttötapauksen suhteen (esim. opintorekisteri)
- Käyttäjä on oikeastaan rooli
  - yksi ihminen voi toimia useassa käyttäjäroolissa
  - sama henkilö voi olla samaan aikaan ohjaaja ja vastuuhenkilö
- *Käyttötapausmallissa ei tule mallintaa mitään järjestelmän sisällä olevaa*, esim. tietokantaa tms.
- Käyttötapauskaavioiden ”nuolityypit”: include, extend, yleistys
  - Milloin ja miten mitäkin käytetään

```
public class Kello {
    private YlhaaltarajoitettuLaskuri sek;
    private YlhaaltarajoitettuLaskuri min;
    private YlhaaltarajoitettuLaskuri tun;
    public Kello(int tunnitAlussa, int minuutitAlussa, int sekunnitAlussa) {
        tun = new YlhaaltarajoitettuLaskuri(23);
        min = new YlhaaltarajoitettuLaskuri(59);
        sek = new YlhaaltarajoitettuLaskuri(59);
    }
    // ...
}
public class YlhaaltarajoitettuLaskuri {
    // ...
}
```



Luokkakaavio: kello *sisältää* kolme viisaria

- Oleellista on merkitä *olioiden* väliset yhteydet
  - Myös yhteyden osallistumisrajoitukset tulee merkitä
  - Tarvittaessa myös nimi, roolit ja salmiakki
- Oliomuuttujia, kuten YlhaaltarajoitettuLaskuri sek, min ja tun **ei ole tapana** merkitä sillä luokkakaavioon piirretty yhteys tuo ne paremmin esille
- Merkataanko muut oliomuuttujat? Entä konstruktorit, metodit ja niiden parametrit?
  - Tarpeen vaatiessa
    - Yleensä tarkkoja UML-kaavioita (joissa mukana kaikki oliomuuttujat ja metodit parametreineen) piirretään erittäin harvoin
  - Esim. jos ohjelmakoodi on olemassa, ei ainakaan kaikkia metodeja yleensä tarvitse merkitä
    - Ainakin toStringin ja konstruktorit voi rauhassa jättää pois
- ”kymmenen sekunnin sääntö”: jos huomaat käyttäväsi luokkakaavion piirtämiseen huomattavasti yli 10 sekunttia, mieti voisitko käyttää ajan hyödyllisemmin
  - Toki joskus tarvetta tarkemmalle kaaviolle ja aikaa menee yli 10 sek
  - ”Piirrä mielummin monta kaaviota nopeasti kuin vain yksi liian tarkasti”

Musta salmiakki eli kompositio

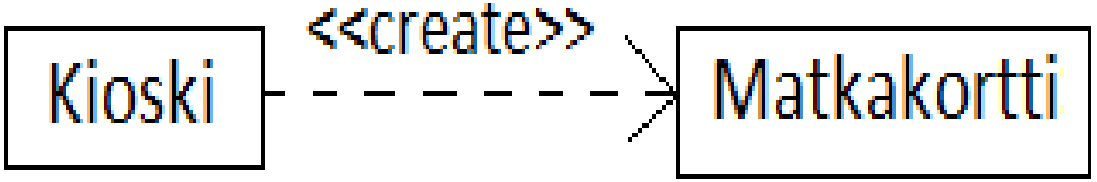
- Kompositiota käytetään kun seuraavat ehdot toteutuvat:
  - *Osat ovat olemassaoloriippuvaisia kokonaisuudesta*
    - Laskurit luodaan Kellon konstruktorissa
    - Kellon tuhoutuessa Laskurit tuhoutuvat
  - *Osa voi kuulua vaan yhteen kompositioon*
    - Laskuria ei voi siirtää toiseen kelloon
  - *Osa on koko elinaikansa kytketty samaan kompositioon*
- HUOM: salmiakki merkitään siihen päähän johon ”osat” sisältyvät

- Jos olet epävarma salmiakista, jätä merkkaamatta, se ei ole kovin paha virhe
- Virheellinen salmiakki taas on virhe
- Pari esimerkkiä:
  - Talossa on huoneita (salmiakki)
  - Joukkue koostuu pelaajista (ei olemassaoloriippuvuutta, ei salmiakkia!)

Ei yhteys vaan riippuvuus

- Luokalla Kioski on *riippuvuus* Matkakorttiin, sillä metodin sisällä käsitellään korttia:

```
public class Kioski {

    Matkakortti ostaMatkakortti( String omistaja, double arvo){

        Matkakortti uusi = new Matkakortti( omistaja );

        uusi.kasvataArvoa(arvo);

        return uusi;

    }

}
```

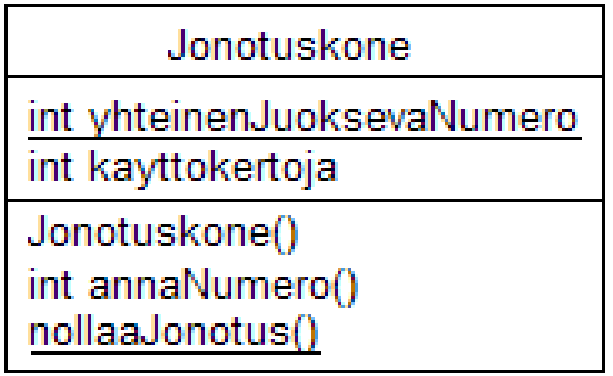


- Yhteyttä ei kuitenkaan ole sillä kioski ei muista matkakortteja
  - Yhteys tarkoittaisi että Kioskilla on kenttä joka tallettaa matkakortteja
- Matkakortin koodissa ei Kioskia mainita mitenkään, joten riippuvuutta toiseen suuntaan ei ole
- Kioskin riippuvuus voidaan varustaa tarkenteella <<create>>
- Myös Lukijalaitteen ja Lataajalaitteen suhde matkakorttiin on riippuvuus, tarkenteena <<use>>

Luokkakohtaiset eli staattiset 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

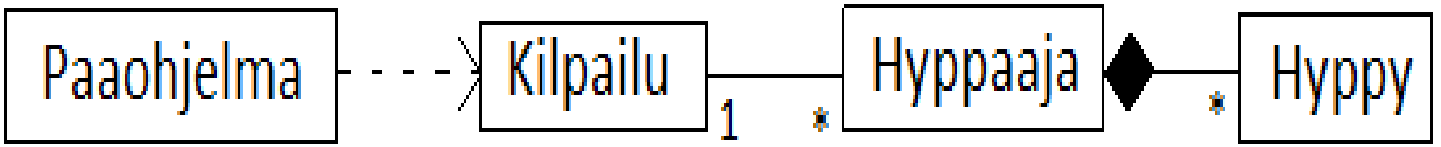
- Ohjelmoinnin jatkokurssi viikon 2 viimeisen ratkaisu näyttää todennäköisesti seuraavalta:

```
public class Paaohjelma {

    public static void main(String[] args){

        Kilpailu kilpailu = new Kilpailu();

        kilpailu.kaynnista();

    }

}
```

- Miten tilanne tulisi mallintaa luokkakaaviona?

Pääohjelma ja luokkakaavio

- Jos haluaa käyttää UML:ää oikeaoppisesti, ei pääohjelman ja Kilpailun 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 kilpailuun on ainoastaan metodin main() sisällä, ei luokassa Pääohjelma itsessään
- Oikeaoppinen tapa lienee kuvata Paaohjelman ja Kilpailun suhde riippuvuutena:



- Paaohjelman ja Kilpailun suhteen merkitseminen normaaliksi yhteydeksi on kuitenkin suhteellisen pieni rike
- Yleensä mainia ei luokkakaavioon merkitä

Määrittelyvaiheen luokkakaavion laatiminen

Luokkakaavion laatiminen

- Kuten jo muutamaan kertaan on mainittu, olioperustaisessa ohjelmistokehityksessä pyritään muodostamaan koko ajan tarkentuva luokkamalli, joka ”simuloi” sovelluksen kohdealuetta
  - Ensin luodaan **määrittelyvaiheen luokkamalli** sovelluksen käsitteistöstä
  - Suunnitteluvaiheessa tarkennetaan edellisen vaiheen luokkamalli **suunnitteluvaiheen luokkamalliksi**
- 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

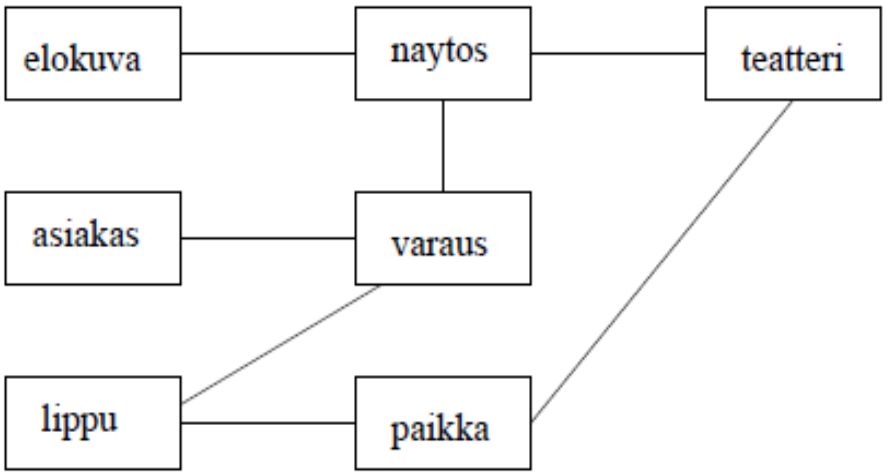
Ehdokkaiden karsiminen

- 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

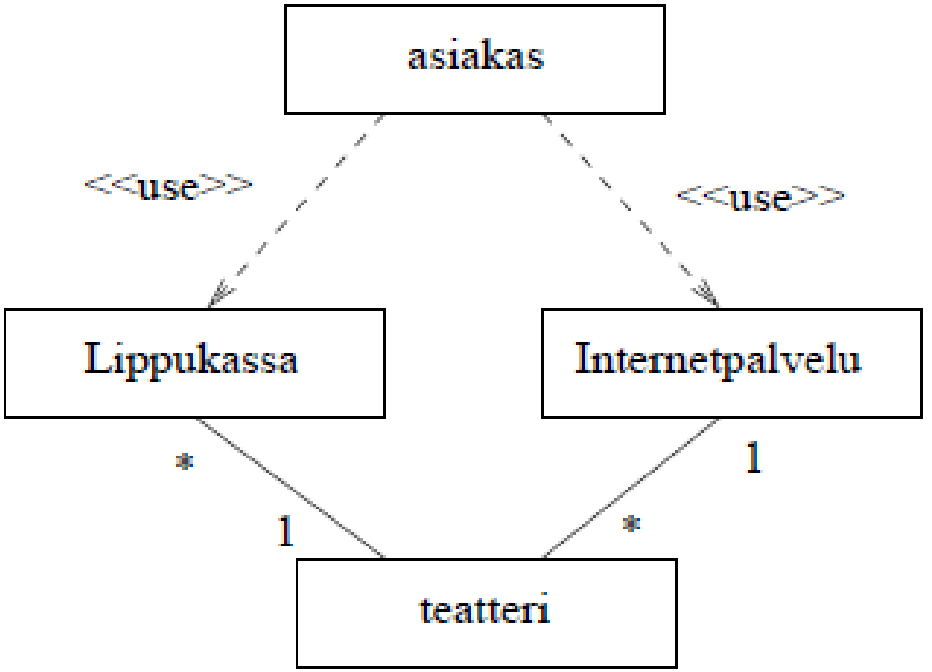
- Attribuuttien löytäminen edellyttää yleensä lisätietoa, esim. asiakkaan haastatteluista
- Määrittelyvaiheen aikana tehtävää kohdealueen luokkamallia ei ole välttämättä tarkoituksenmukaista tehdä kaikin osin tarkaksi
  - Malli tarkentuu ja muuttuu jokatapauksessa suunnitteluvaiheessa
- Elokuvateatterin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



Luokkien Asiakas ja Lippukassa suhde

- Edellisen sivun kaltaisessa tilanteessa ei olisi todennäköisesti mielekästä merkitä Asiakkaasta edes riippuvuutta Lippukassaan
- Luokka Asiakas on nimittäin järjestelmässä oikean elokuvateatterin asiakkaan representaatoi joka ei kuitenkaan itsessään ”suorita” mitään toienpiteitä
  - Suurin osa varsinaisen toiminnallisuuden suorittavista olioista tulee vasta suunnittelutason luokkamalleihin
- Hieman samaan tapaan yliopiston kuhunkin opiskelijaan liittyy OODI-järjestelmässä oma ”olio”, kuitenkaan tuo OODI:ssa oleva olio ei tee mitään, esim. käy luennoilla tai suorita kursseja
- Luokkamallin yhteyksissä siis ei tule kuvata toiminnallisuutta, tyyliin Asiakas tekee varauksen Internetpalvelussa tai Opiskelija käy kurssin Luennolla vaan olioiden välisiä suhteita (jotka voivat olla toiminnan seuraus):
  - Asiakkaaseen liittyy Varaus
  - Opiskelija on suorittanut Kurssin

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*

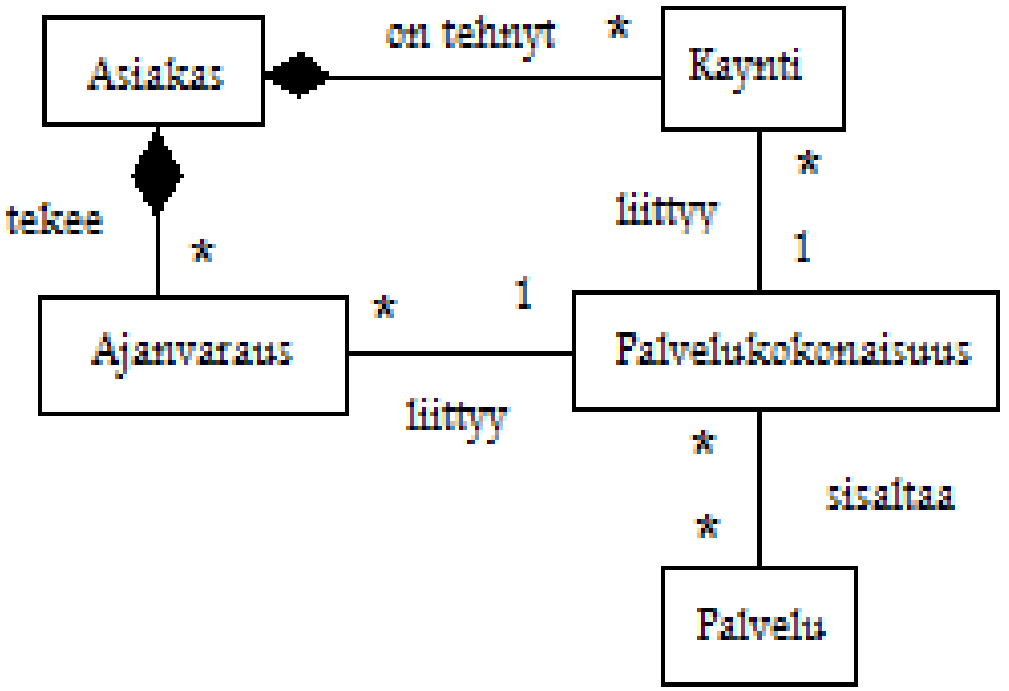
Alustavasti valitut luokat

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

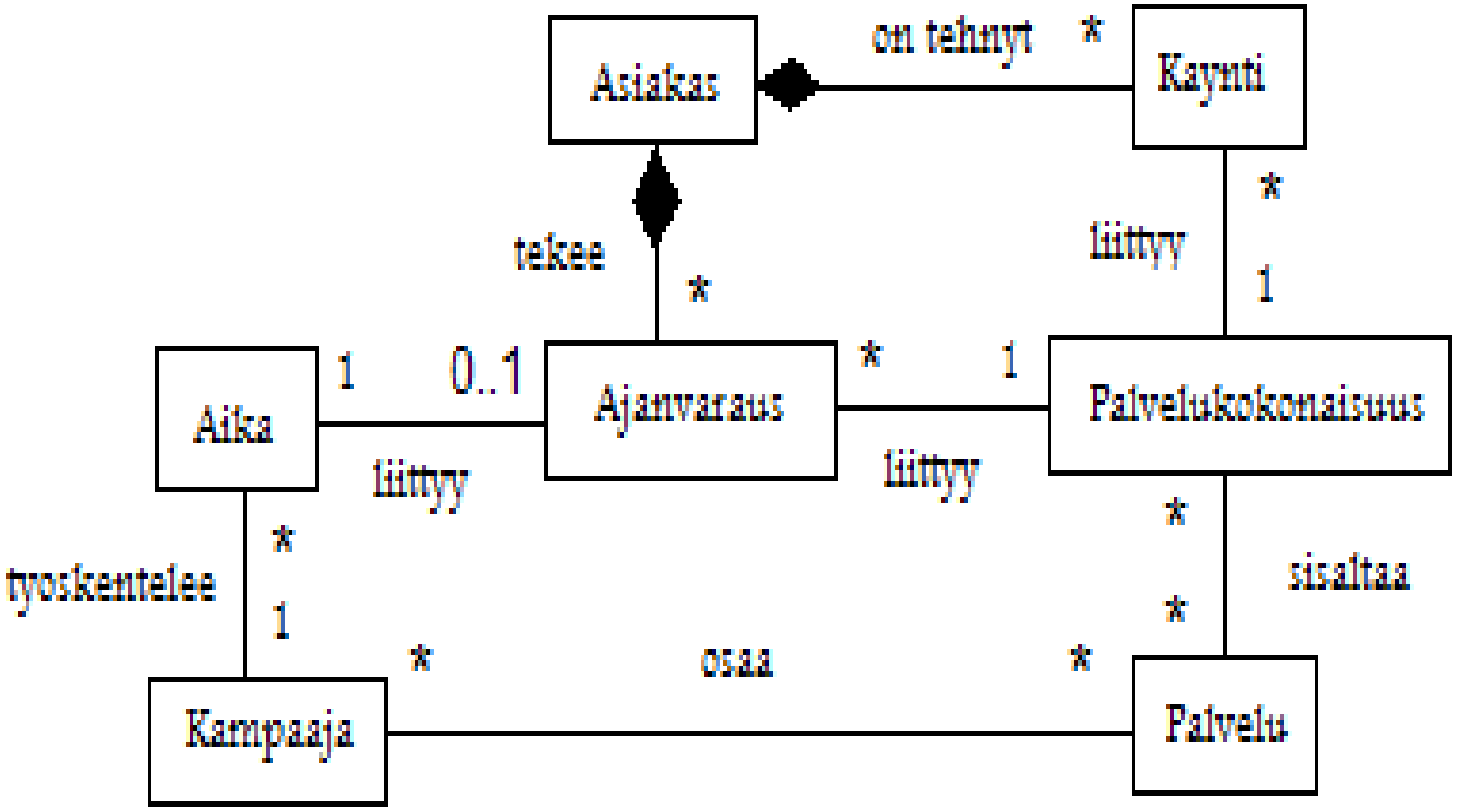
- **HUOM: yleensä ei jakseta kerätä kaikkia substantiiveja vaan tehdään esim. synonyymien ja tekemistä tarkoittavien sanojen karsinta samalla kun ”kunnollisten” substantiivien etsintä etenee**

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



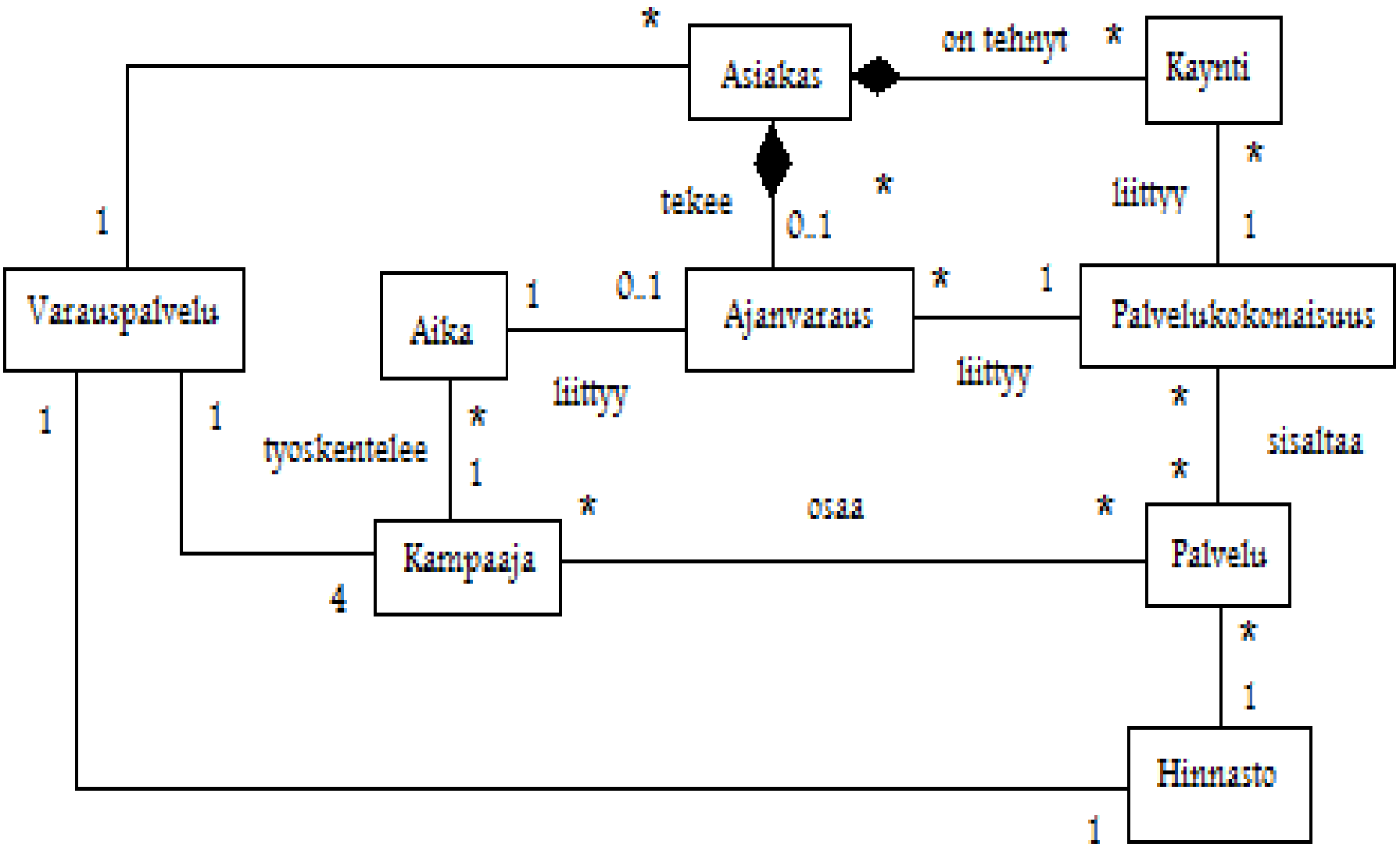
- Tarkastellaan luokkia aika, kampaaja
  - Kampaamossa työskentelee 4 kampaajaa, jotka ovat työssä epäsäännöllisesti. Kaikki kampaajat eivät tee kaikkia tarjolla olevia toimenpiteitä.
  - Varaustilanteessa järjestelmän pitää kyetä näyttämään asiakkaalle, milloin halutun palvelun suorittamaan pystyviltä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

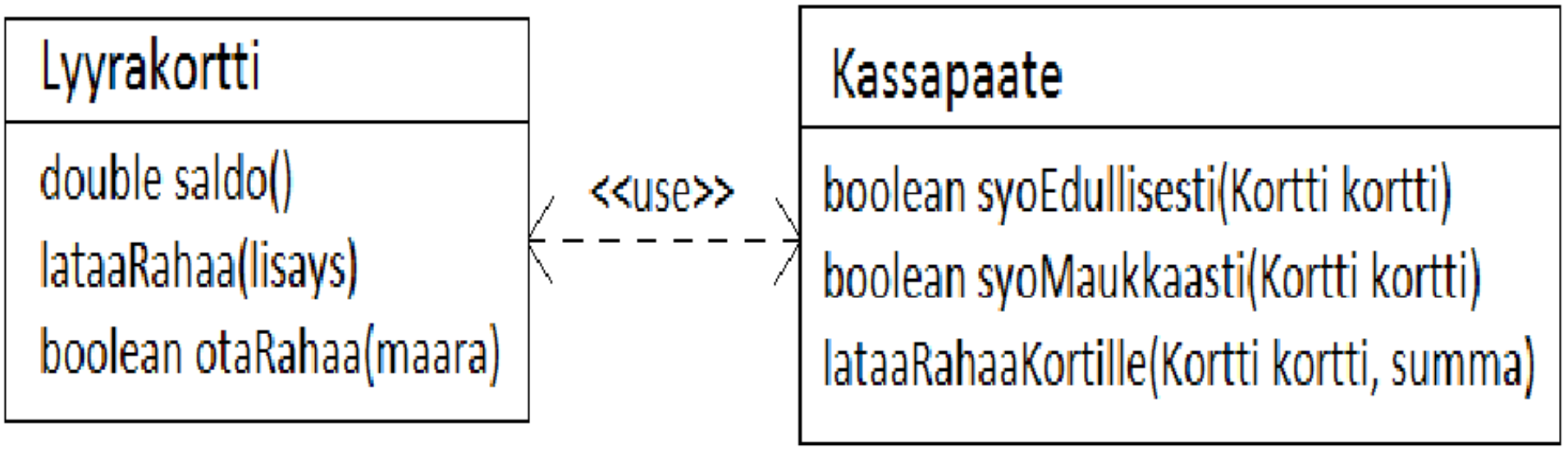
- Isoa ongelmaa kannattaa lähestyä pienin askelin, esim:
  - Yhteydet ensin karkealla tasolla, tai
  - Tehdään malli pala palalta, lisäten siihen muutama luokka yhteyksineen kerrallaan

- 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

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 kuvata esim. skenaario ”ostetaan 3 euroa sisältävällä lyyrakortilla edullinen lounas”



Olioiden yhteistyö

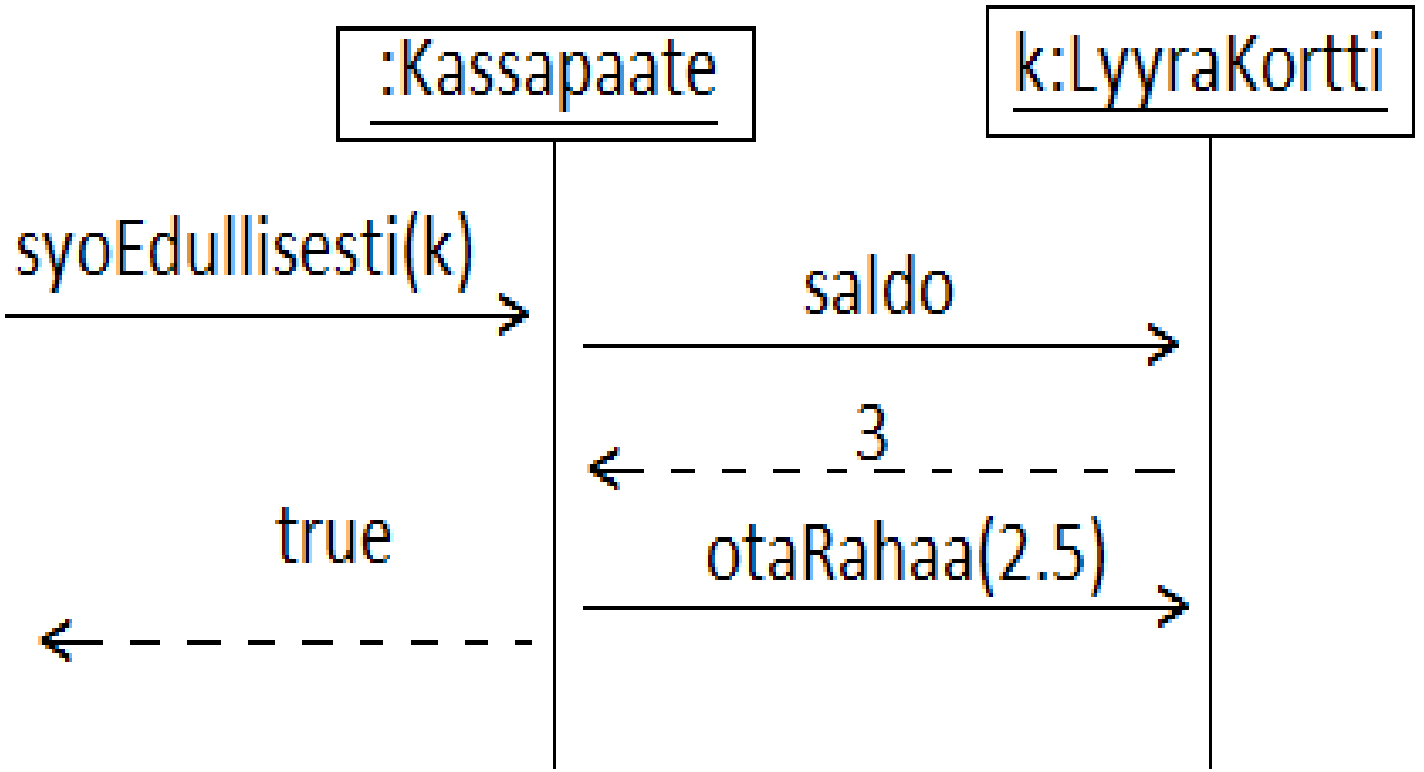
- Oliopohjaisuus siis perustuu seuraavaan olettamukseen:
  - *Minkä tahansa järjestelmän katsotaan voivan muodostua olioista, jotka yhteistyössä toimien ja toistensa palveluja hyödyntäen tuottavat järjestelmän tarjoamat palvelut*

- 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 ”ostetaan 3 euroa sisältävällä lyyrakortilla edullinen lounas”
  - Lukemalla koodia (ks. mallivastaus ohpe viikko 5) huomataan, että kassapääte kysyy ensin kortin saldon ja huomatessaan sen riittävän, vähentää kortilta edullisen lounaan hinnan
- Tilanteen kuvaava *sekvenssikaavio* alla



Sekvenssikaavio

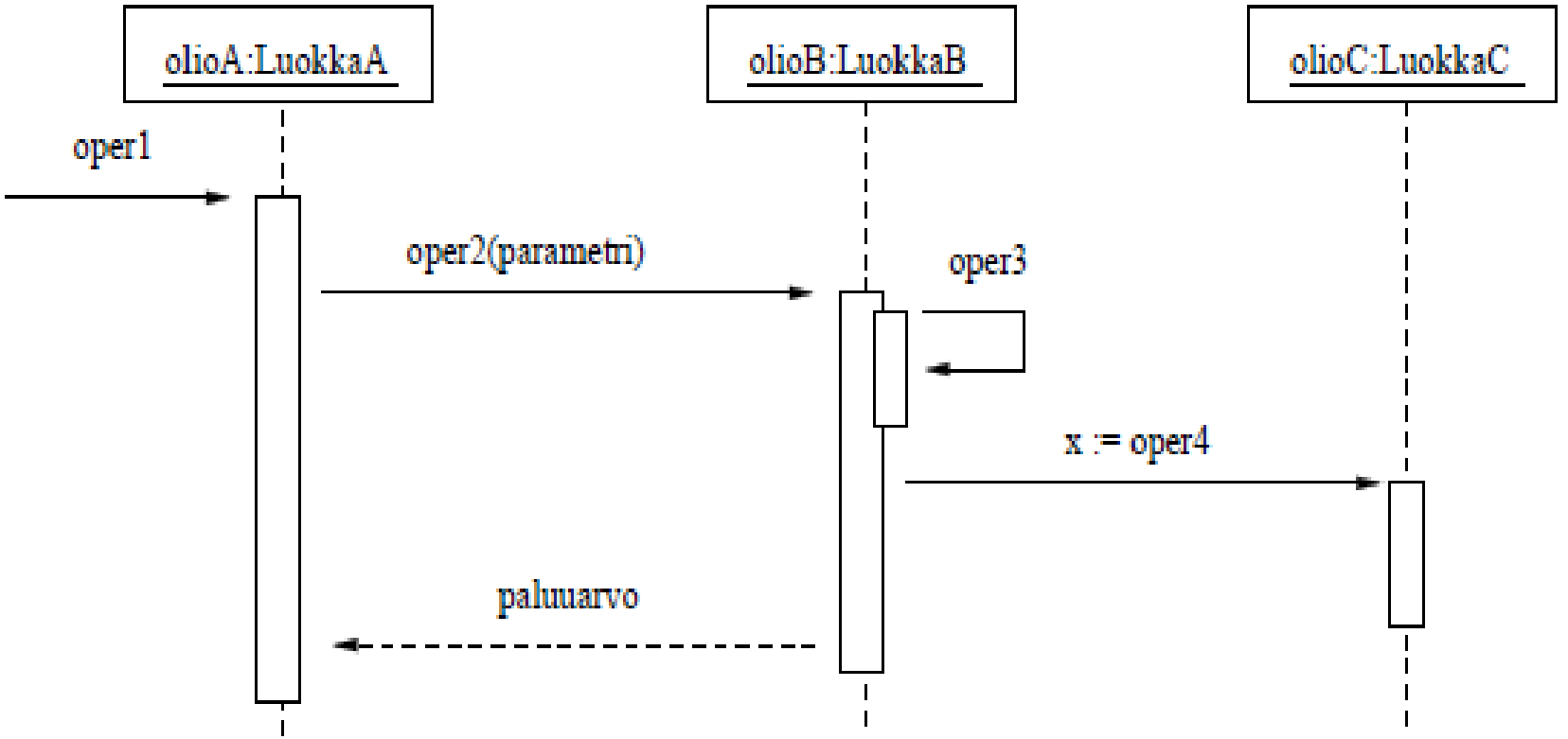
- Sekvenssikaaviossa kuvataan tarkasteltavan skenaarion aikana tapahtuva olioiden vuorovaikutus
- Oliot esitetään kuten oliokaaviossa, eli laatikkoina, joissa alleviivattuna olion nimi ja tyyppi
- Sekvenssikaaviossa oliot ovat (yleensä) ylhäällä rivissä
- Aika etenee kaaviossa alaspäin
- Jokaiseen olioon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

Edullisen lounaan oston sekvenssikaavio

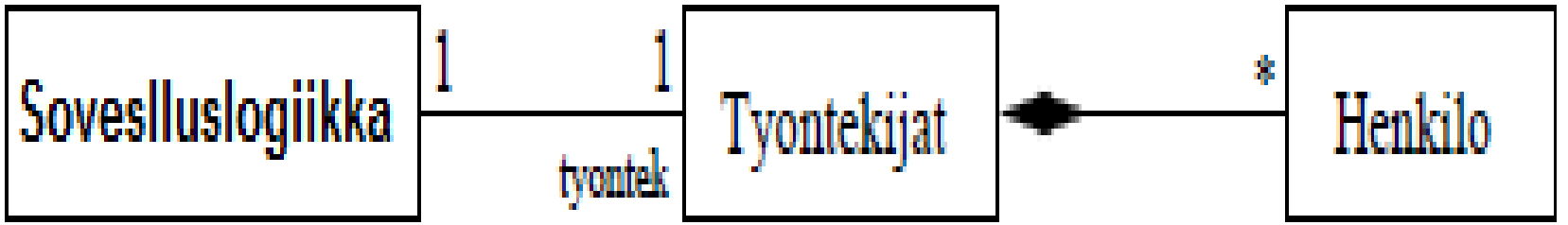
- Esimerkissä toiminta alkaa sillä, että joku (esim. pääohjelma) kutsuu Kassapaate-olion metodia syoEdullisesti
  - Parametrina on Lyyrakorttiolio k
- Metodikutsun seurauksena kassapääte kutsuu lyyrakortin metodia saldo, joka palauttaa kortilla olevan rahamäärän
  - Kortin palauttama saldo on merkitty katkoviivalla
- Tämän jälkeen kassapääte kutsuu kortin metodia otaRahaa, parametrilla 2.5 eli velottaa kortilta edullisen lounaan hinta
- Kun hinta on veloitettu, Kassapääte palauttaa operaation onnistumisen merkiksi true metodin syoEdullisesti kutsujalle
  - Metodin paluuarvo on jälleen merkitty katkoviivalla

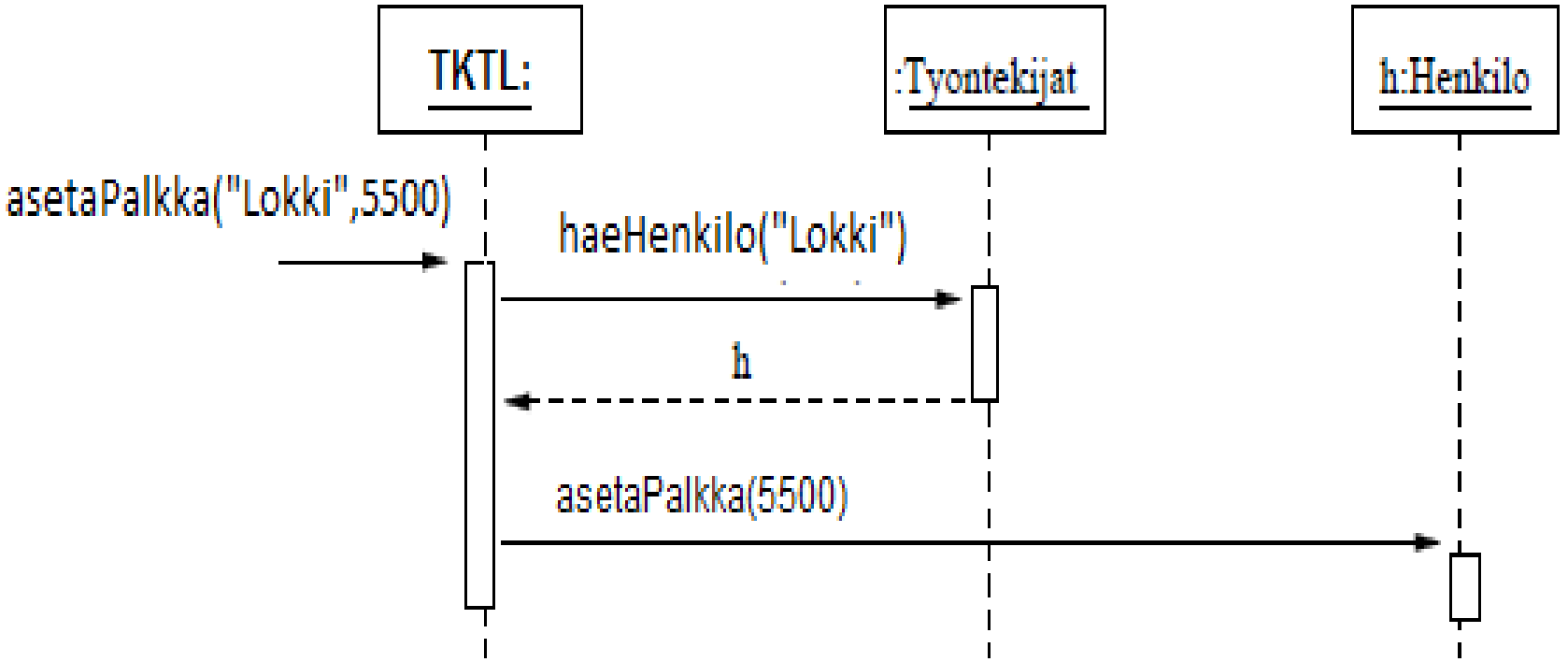
Lisää syntaksia

- 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






```
public 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 (Lokin tiedot sisältävän) olion nimi!

Käyttötapaus ja sekvenssikaavio

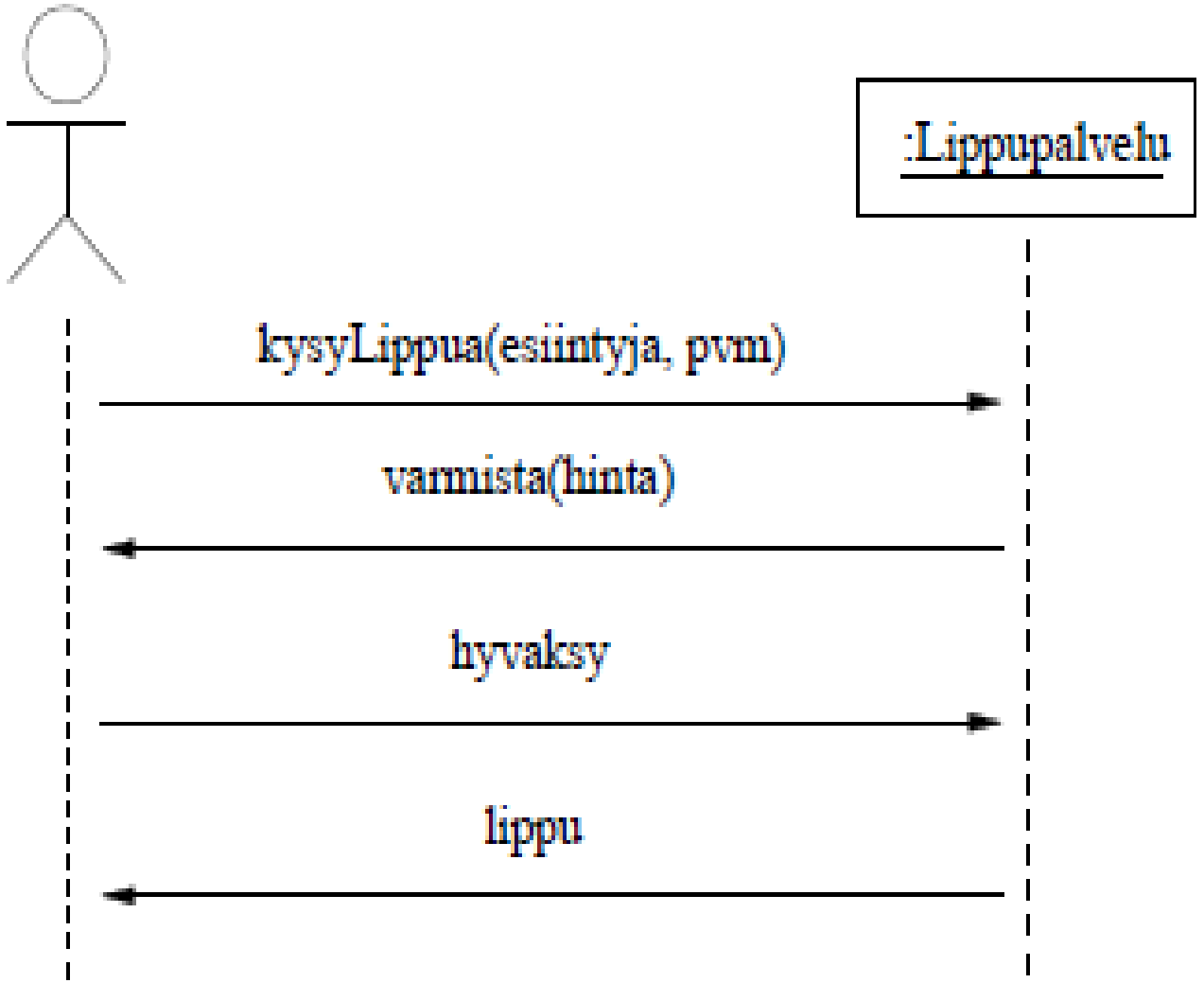
- Tarkastellaan alkeellista lippupalvelun tietojärjestelmää ja sen käyttötapausta

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*

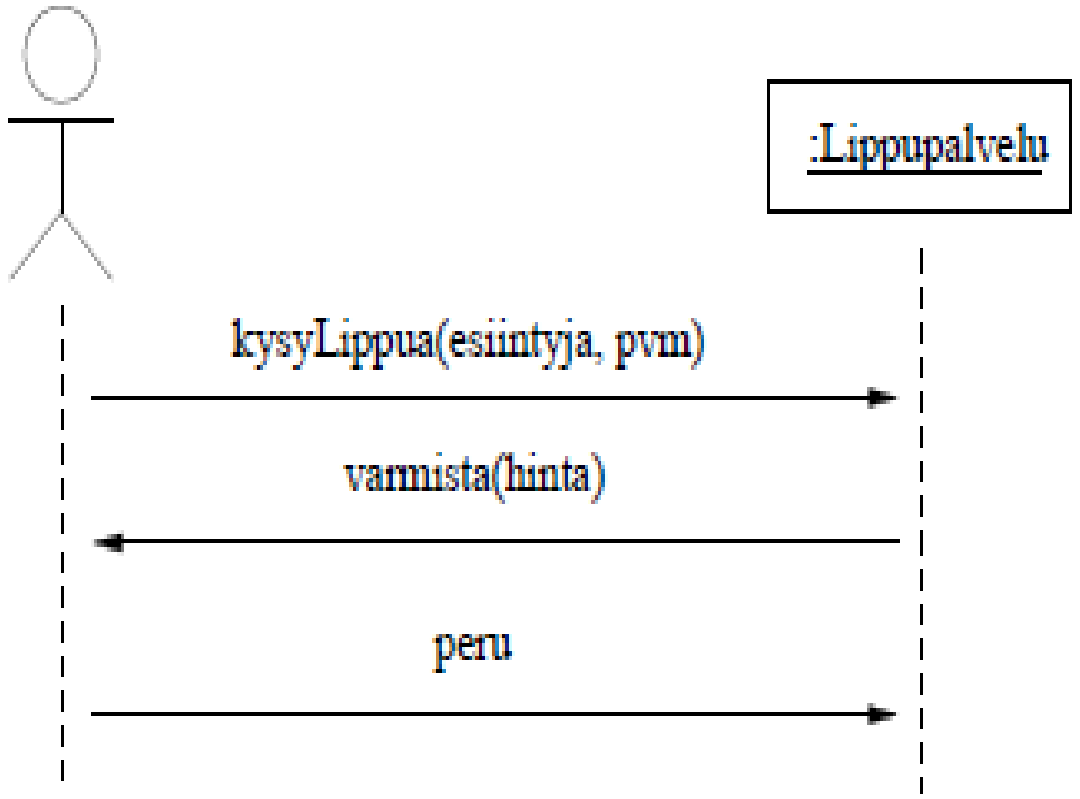
Käyttötapauksen *Lipun varaus* kuvaava sekvenssikaavio



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

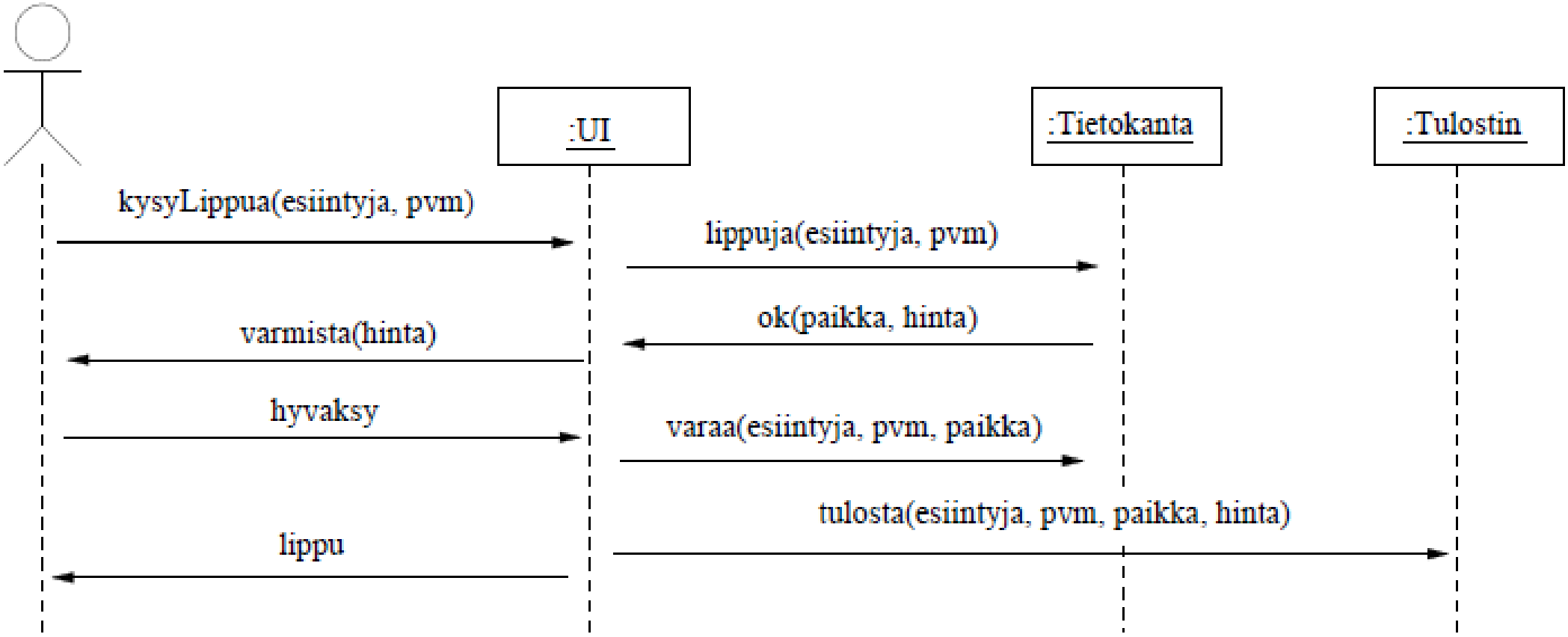
Vaihtoehtoinen skenaario

- 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



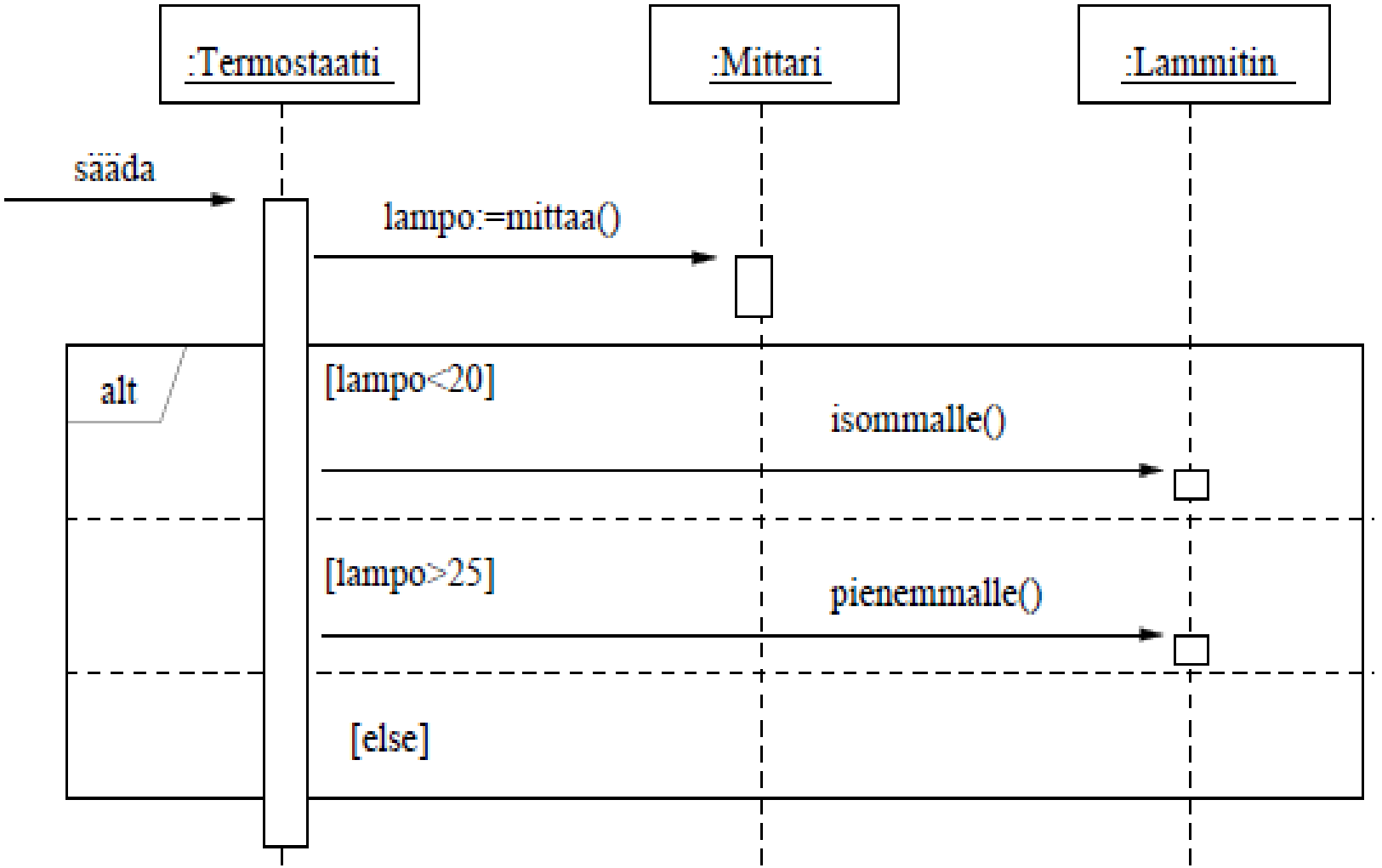
Järjestelmätasolta suunnittelutasolle

- 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:



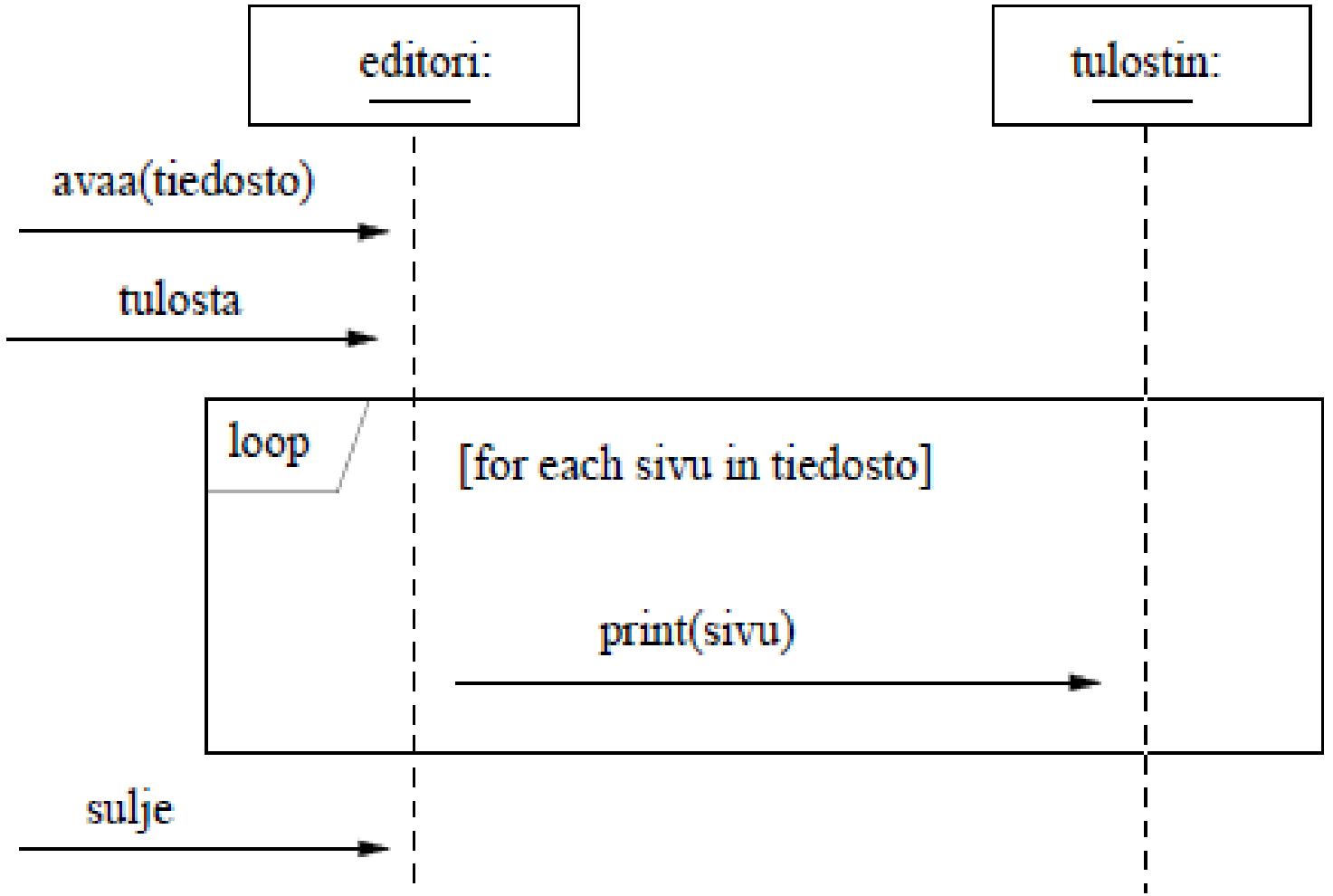
Valinnaisuus sekvenssikaaviossa

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



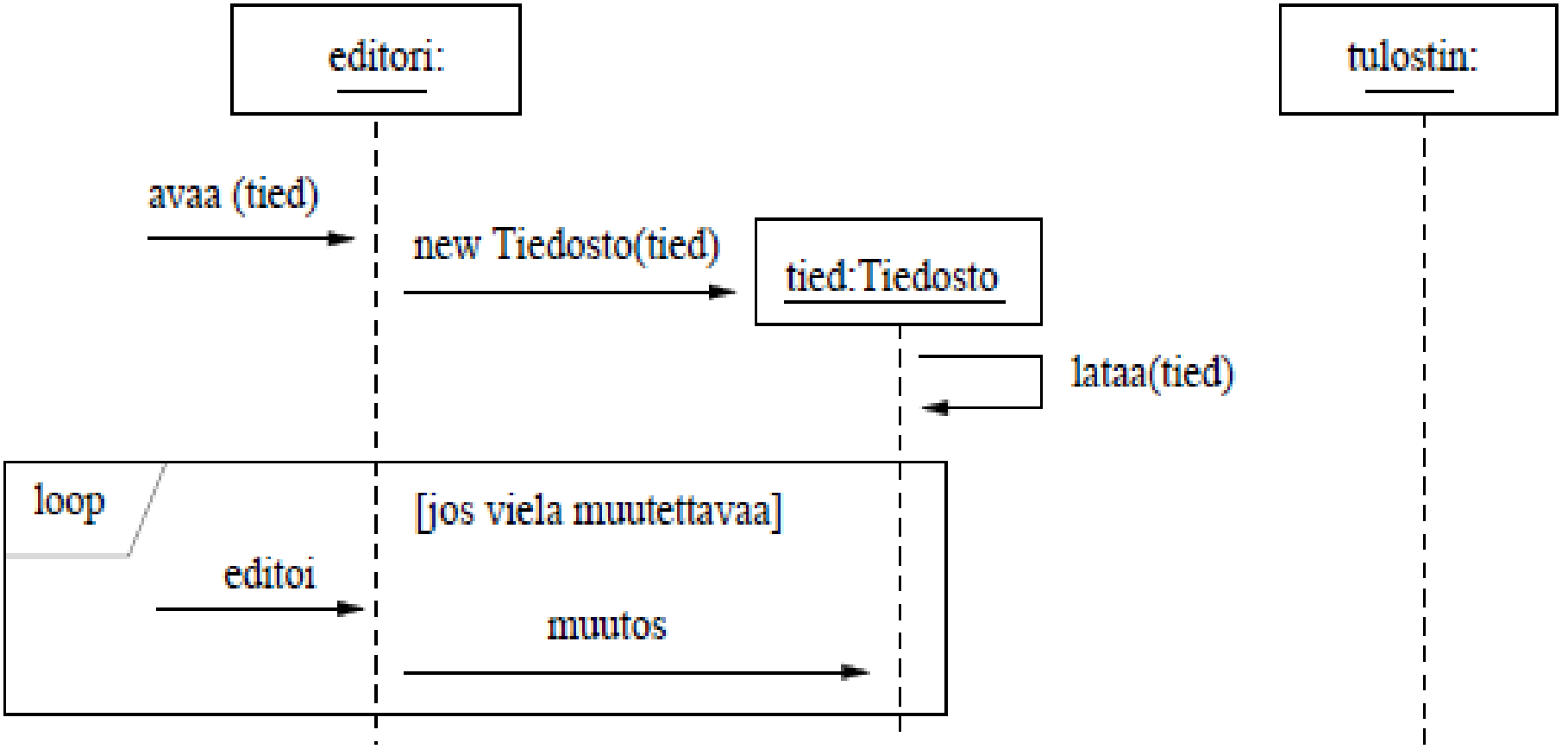
Toisto

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

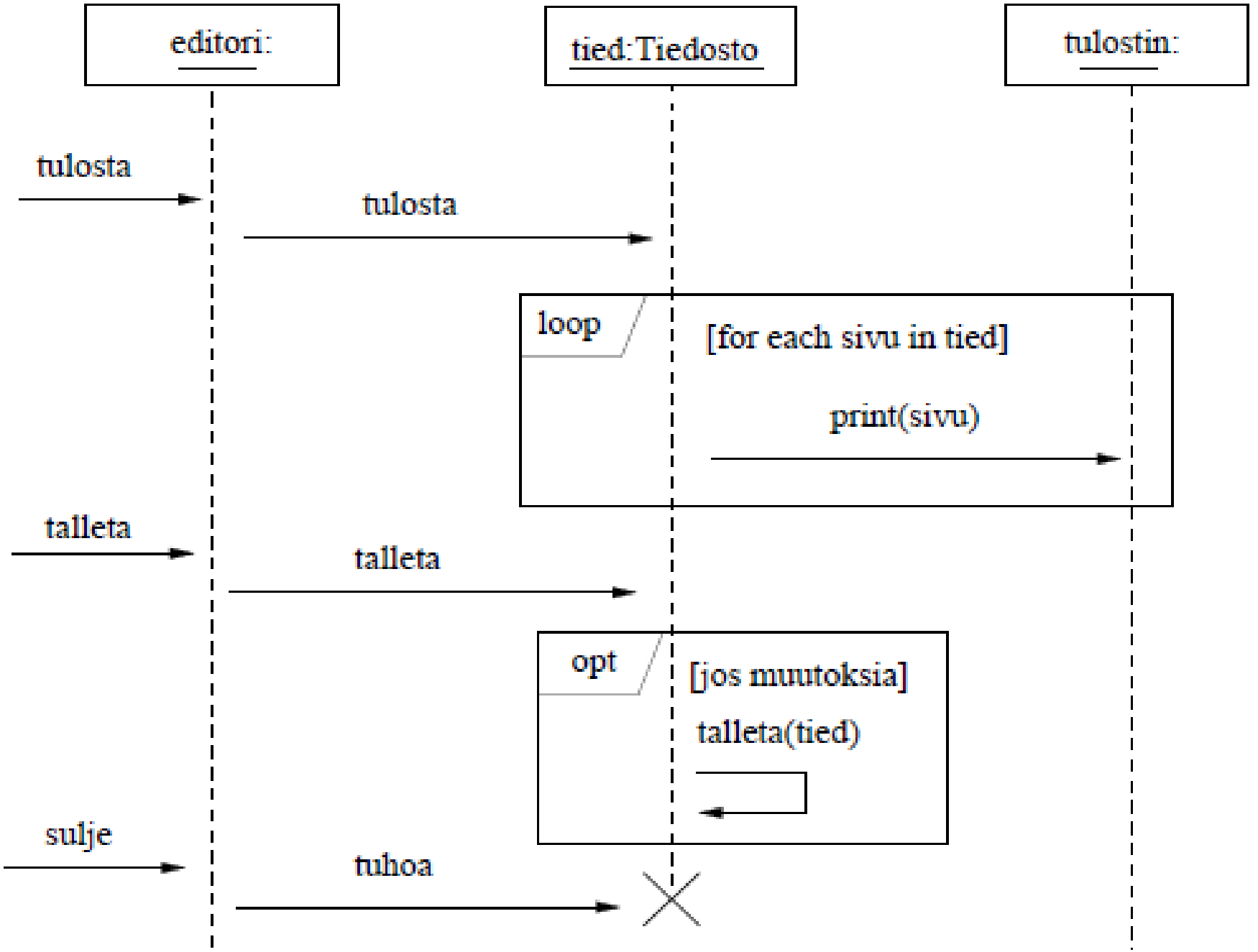


Olioiden luominen ja tuhoaminen

- 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

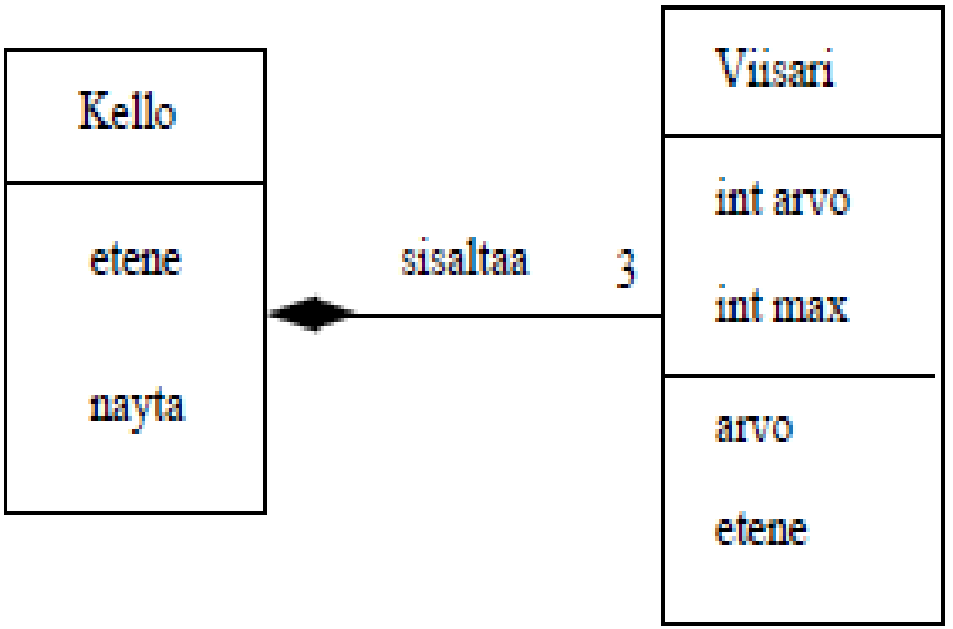


- Editoriesimerkki jatkuu
- Mukana myös *valinnainen (opt) lohko*, joka suoritetaan jos ehto tosi

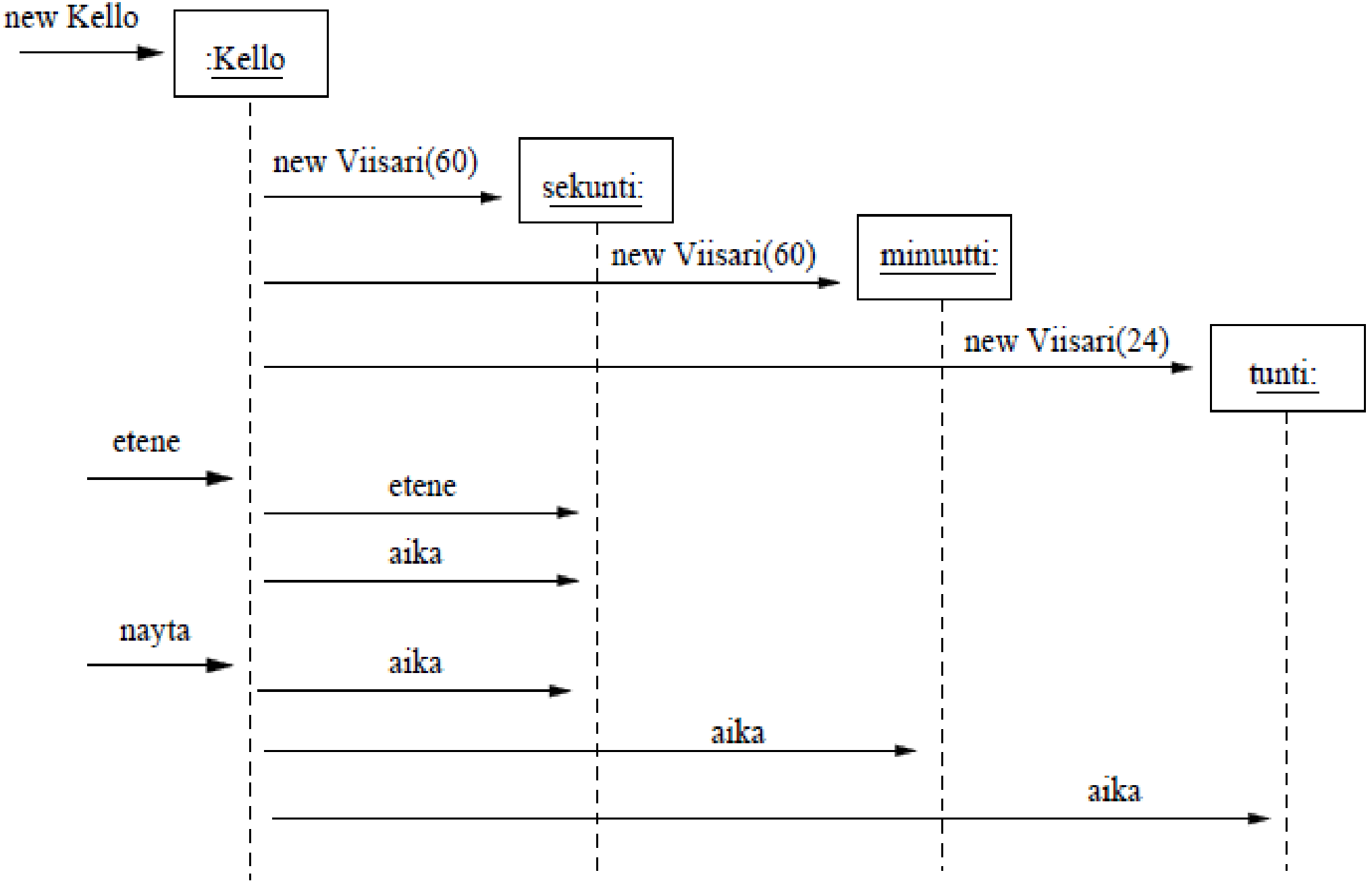


Takaisinmallinnus

- *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

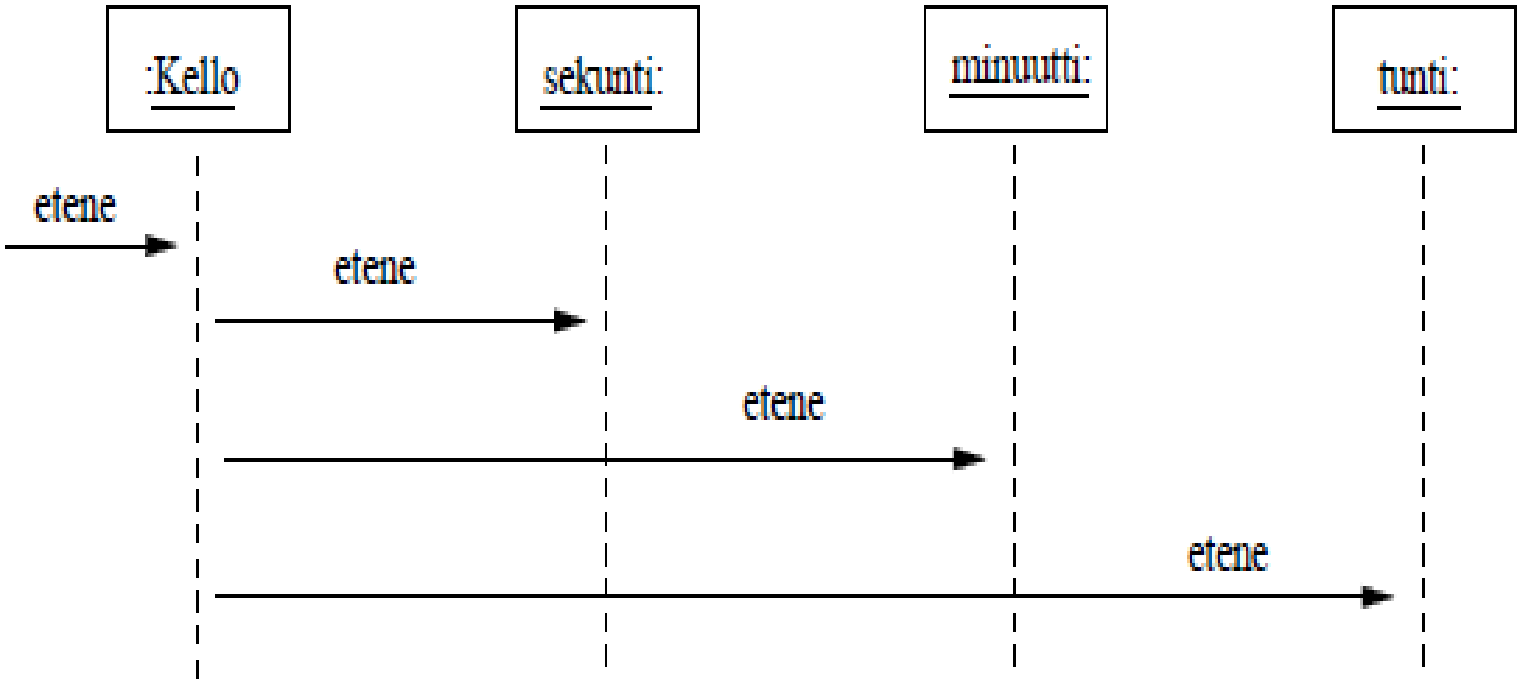


Kellon syntyy ja lähtee käymään



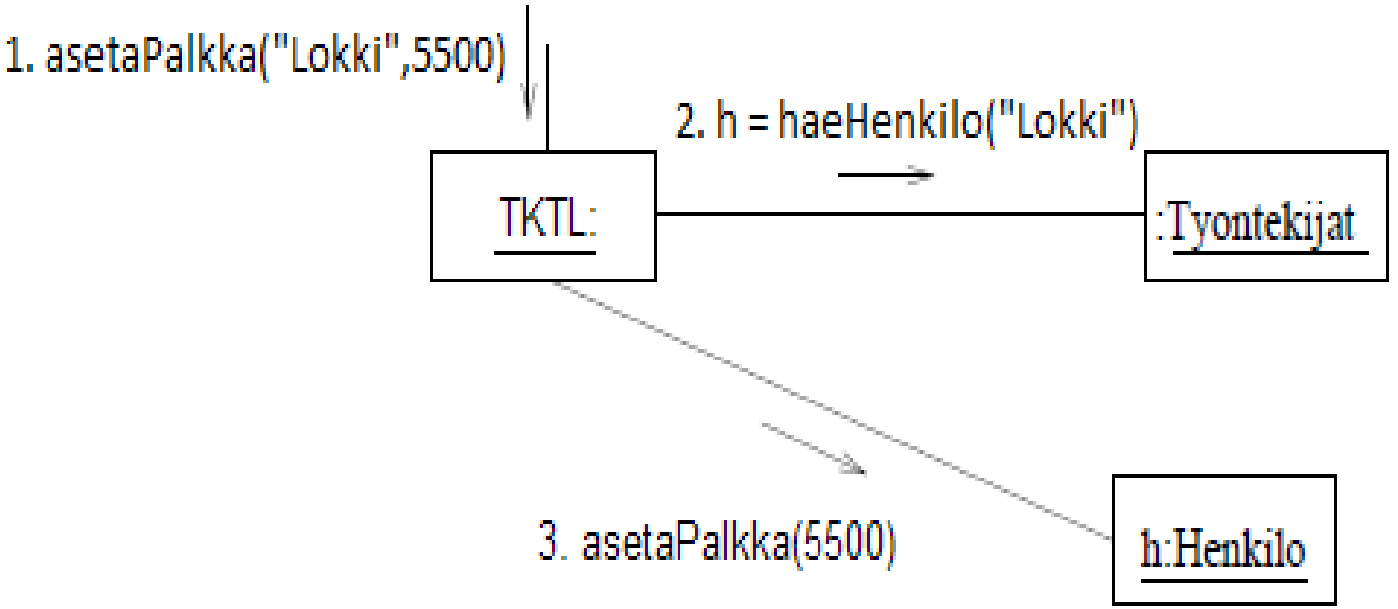
Kellon eteneminen tasatunnilla

- 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

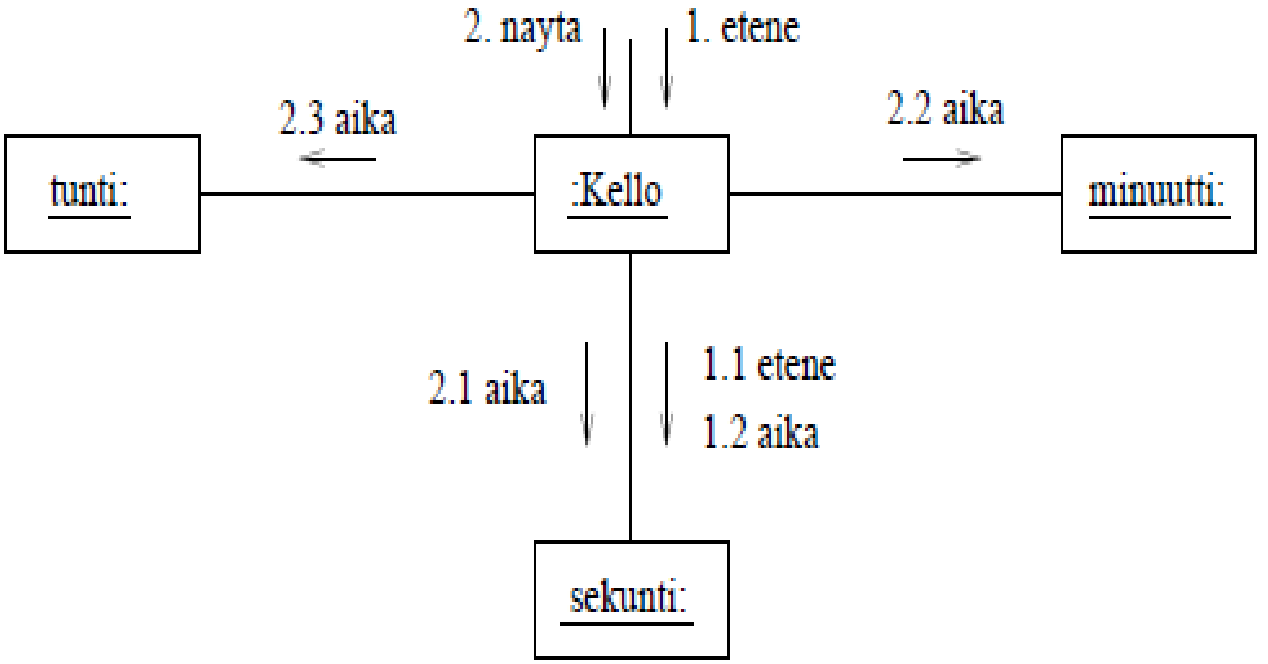


Kommunikaatiokaavio

- 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