Ohjelmistotekniikan menetelemät

Matti Luukkainen, Olli-Pekka Mehtonen

Helsingin Yliopisto, TKTL

Kesällä 2016

Wikipedia:

Ohjelmistotuotanto on yhteisnimitys niille työnteon ja työnjohdon menetelmille, joita käytetään, kun tuotetaan tietokoneohjelmia sekä monista tietokoneohjelmista koostuvia tietokoneohjelmistoja.

Ohjelmistotuotanto laajasti ymmärrettynä kattaa kaiken tietokoneohjelmistojen valmistukseen liittyvän prosessinhallinnan sekä kaikki erilaiset tietokoneohjelmien valmistamisen menetelmät.

Ohjelmistotuotanto kattaa siis kaikki aktiviteetti, jotka tähtää tietokoneohjelmien tai -ohjelmistojen valmistukseen.

Mallintaminen

- Tietokoneohjelman valmistamisen menetelmiin liittyy **mallintaminen**, eli kyky tuottaa erilaisia kuvauksia, joita tarvitaan ohjelmiston kehittämisen yhteydessä
- Mallit toimivat kommunikoinnin välineinä
- *Mitä ollaan tekemässä, miten ollaan tekemässä, mitä tehtiin?*

Kurssista

- **Aikataulu ja kurssimateriaali:**
  - `https://github.com/Ooppa/OTM-kesa16`
- **Laskarit:**
  - Aloitetaan jo ensimmäisellä viikolla
  - Yhteensä 7kpl, 3h per tilaisuus
- **Arvostelu:**
  - Kurssin kokonaispistemäärä on 36p
  - Kurssikoe 22p
  - Laskareista 14p, jotka koostuu...
    - Paikanpäällä tehtävistä 7p (1p per kerta)
    - Etukäteen tehtävistä 7p (90% $\rightarrow$ 7p)
  - Noin 32p $\rightarrow$ arvosana 5
  - Läpipääsy vaatii puolet kurssikokeen pisteistä ja puolet laskaripisteistä, eli ainakin 18p

Laskarit

- **Etukäteen tehtävät:**
  - Keskimäärin 6 tehtävää viikossa
  - Tehdään etukäteen niin, että vastauksia voidaan tarkastella ryhmissä, eli tulostettuna, läppärillä tai verkossa
  - Pisteitä saa vaikka kaikki ei ole oikein
  - Yrityksestä jää kuitenkin aina jälki!
- **Paikanpäällä tehtävät:**
  - Ryhmätyöskentelyä
  - Tehdään niin paljon kuin ehtii, mutta työskennellään aktiivisesti
  - Laskarit eivät ole paja, paikalla on oltava alusta loppuun!
- **Laskariajat**
  - TODO

Ohjelmistotuotantoprosessi

Miksi prosessi kun voi vain tehdä?

Ohjelmistotuotantoprosessi

Miksi prosessi kun voi vain tehdä?

- Pienissä itselle tehtävissä projekteissa voidaan tuottaa sovellusta noudattamatta systematiikkaa
- Voidaan helposti väsätä kasaan sovellus, joka *"toimii"*
- Tämä menetelmä ei toimi isommille, monen hengen projekteissa asiakasta varten tuotetuille ohjelmille (Toimiiko sovellus niin kuin alunperin haluttiin?)
- Rakenne epämääräinen $\rightarrow$ Ylläpidettävyys huono
- Ratkaisu: Kehitelty erilaisia menetelmiä ohjelmistotuotantoprosessin systematisoimiseksi [1]
- Mitä menetelmää tulisi käyttää? Hyvä kysymys!

[1] https://en.wikipedia.org/wiki/List_of_software_development_philosophies

Ohjelmistotuotantoprosessin vaiheet

Yhteenveto vaiheista

1. **Vaatimusanalyysi- ja määrittely**
   - Mitä halutaan?
2. **Suunnittelu**
   - Miten tehdään?
3. **Toteutus**
   - Ohjelmoidaan
4. **Testaus**
   - Varmistetaan että toimii niin kuin halutaan
5. **Ylläpito**
   - Korjataan bugeja ja laajennetaan ohjelmistoa

Ohjelmistotuotantoprosessin vaiheet

Vaatimusanalyysi ja -määrittely

Kartoitetaan ja dokumentoidaan **mitä asiakas haluaa**

- Ei vielä puututa siihen miten järjestelmä tulisi toteuttaa
- Ei oteta kantaa ohjelman sisäisiin teknisiin ratkaisuihin, ainoastaan siihen miten toiminta näkyy käyttäjälle
- Sovelluksen **toiminnalliset vaatimukset**
  - Miten ohjelman tulisi toimia?
  - Toimintaympäristön asettamat rajoitteet
  - Toteutusympäristö
  - Suorituskykyvaatimukset
  - Luotettavuusvaatimukset
  - Käytettävyys

Ohjelmistotuotantoprosessin vaiheet

Vaatimusanalyysi ja -määrittely

Esim: Yliopiston kurssinhallintajärjestelmä

- Toiminnallisia vaatimuksia:
  - Opetushallinto voi syöttää kurssin tiedot järjestelmään
  - Opiskelija voi ilmoittautua valitsemalleen kurssille
  - Opettaja voi syöttää opiskelijan suoritustiedot
  - Opettaja voi tulostaa kurssin tulokset
- Toimintaympäristön rajoitteita:
  - Kurssien tiedot talletetaan jo olemassa olevaan tietokantaan
  - Järjestelmää käytetään www-selaimella
  - Toteutus Javalla
  - Kyettävä käsittelemään vähintään 100 ilmoittautumista minuutissa

Ohjelmistotuotantoprosessin vaiheet

Vaatimusanalyysi ja -määrittely

- Jotta toteuttajat ymmärtäisivät mitä pitää tehdä, joudutaan ongelma-aluetta analysoimaan
  - Esimerkiksi jäsennetään ongelma-alueen käsitteistöä
  - Tehdään ongelma-alueesta malli eli yksinkertaistettu kuvaus
- Vaatimusmäärittelyn päätteeksi yleensä tuotetaan **määrittelydokumentti**
  - Kirjaa sen mitä ohjelmalta halutaan
  - Käytetään ohjeena suunnitteluun ja toteutukseen
- Määrittelydokumentin sijaan määrittely (tai ainakin sen osa) voidaan myös ilmaista ns. hyväksymistesteinä. Tällöin ohjelma toimii ”määritelmänsä mukaisesti” jos se läpäisee kaikki määritellyt hyväksymistestit

Ohjelmistotuotantoprosessin vaiheet

Ohjelmiston suunnittelu

Miten saadaan toteutettua määrittelydokumentissa vaaditulla tavalla toimiva ohjelma?

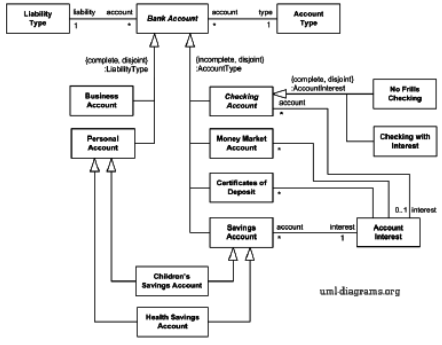
1. **Arkkitehtuurisuunnittelu**
   - Ohjelman rakenne karkealla tasolla
   - Mistä suuremmista rakennekomponenteista ohjelma koostuu?
   - Miten komponentit yhdistetään, eli komponenttien väliset rajapinnat
2. **Oliosuunnittelu**
   - yksittäisten komponenttien suunnittelu

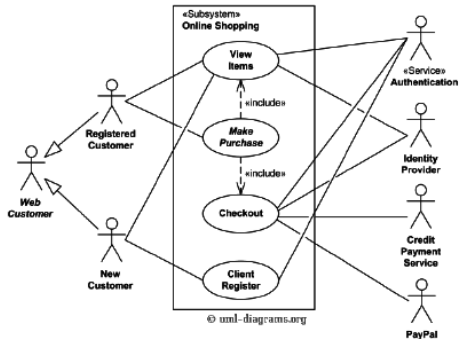
$\longrightarrow$ Lopputuloksena suunnitteludokumentti

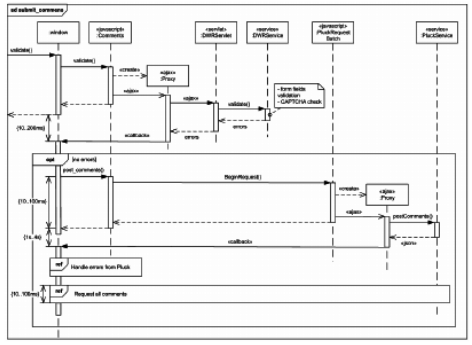
Ohjelmistotuotantoprosessin vaiheet

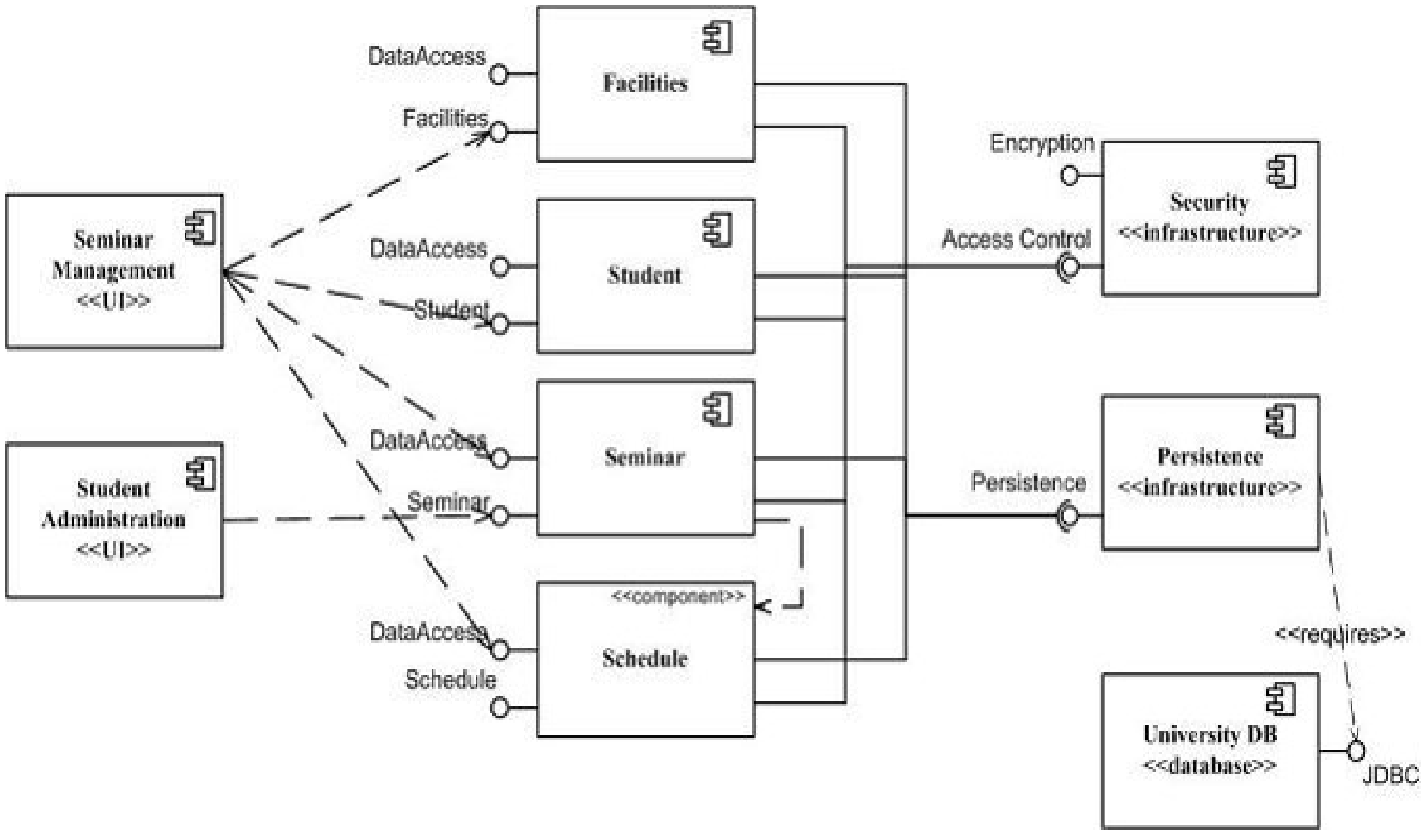
Ohjelmiston suunnittelu

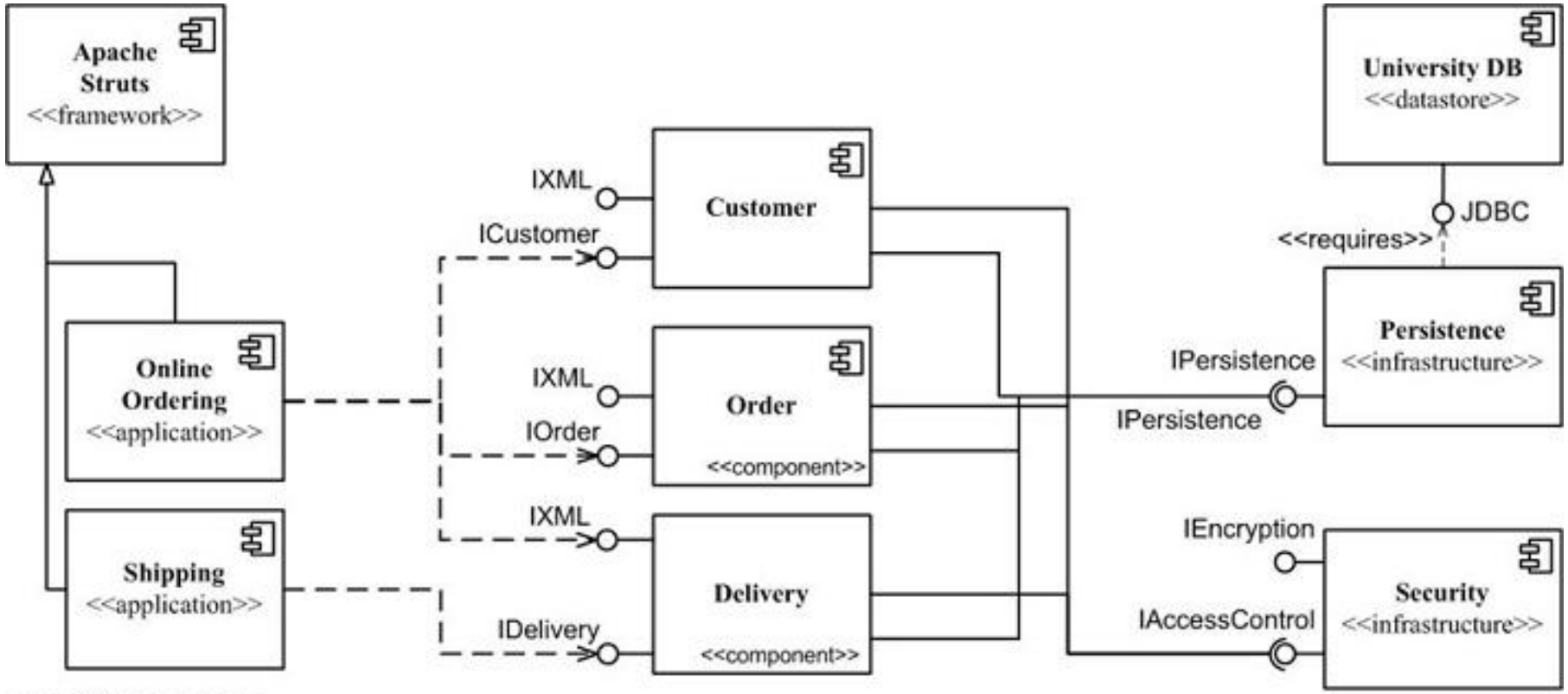
- **Suunnitteludokumentti**
  - Ohje toteuttajille
  - Joskus/usein suunnittelu- ja ohjelmointivaihe ovat niin kiinteästi sidottuna toisiinsa, että tarkkaa suunnitteludokumenttia ei tehdä
  - Joskus koodi toimii dokumenttina
- **Mallit liittyvät vahvasti suunnitteluun!**
  - Arkkitehtuurikuvaus
    - Järjestelmän alikomponentit
    - Komponenttien väliset rajapinnat
    - muista: suunnittelumallit (*design patterns*)








Facilities
DataAccess
Facilities
Seminar Management
<<UI>>
Student
DataAccess
Student
Seminar
DataAccess
Seminar
Student Administration
<<UI>>
<<component>>
Schedule
DataAccess
Schedule
Security
<<infrastructure>>
Encryption
Access Control
Persistence
<<infrastructure>>
Persistence
<<requires>>
University DB
<<database>>
JDBC


Apache
Struts
<<framework>>
Online
Ordering
<<application>>
Shipping
<<application>>
IXML
ICustomer
Customer
IXML
IOrder
Order
<<component>>
IXML
IDelivery
Delivery
<<component>>
University DB
<<datastore>>
JDBC
<<requires>>
Persistence
<<infrastructure>>
IPersistence
IPersistence
IEncryption
IAccessControl
Security
<<infrastructure>>
Copyright 2004 Scott W. Ambler

Ohjelmistotuotantoprosessin vaiheet

Toteutus, testaus ja ylläpito

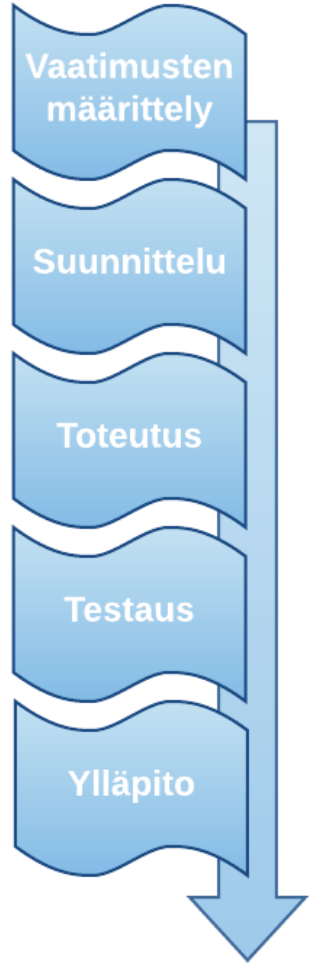
- Suunnittelun mukainen järjestelmä toteutetaan valittuja tekniikoita käyttäen
- Toteutuksen yhteydessä ja jälkeen testataan:
  - **Yksikkötestaus**
    - Toimivatko yksittäiset metodit ja luokat?
  - **Integraatiotitestaus**
    - Varmistetaan komponenttien yhteentoimivuus
  - **Järjestelmätestaus**
    - Toimiiko kokonaisuus niin kuin vaatimusdokumentissa sanotaan (huom: hyväksymistestaus)?
- Valmiissakin järjestelmässä virheitä ja tarvetta laajennuksiin
- **Tämän kurssin painopiste on vaatimusmäärittelyssä ja suunnittelussa, mutta osin myös testaamisessa**

Ohjelmistotuotantoprosessi käytännössä

Vesiputousmalli

Vesiputousmalli on vaiheellinen ohjelmistotuotantoprosessi, jossa suunnittelu- ja toteutusprosessi etenee vaihe vaiheelta alaspäin kuin vesiputouksessa.

- **Perinteinen tapa tehdä ohjelmistoja**
- Tuotantoprosessin vaiheet etenevät peräkkäin
- Vaiheen valmistuttua seuraavaan vaiheeseen
- Jokaisen vaiheen lopputulos dokumentoidaan tyypillisesti erittäin tarkasti



Ohjelmistotuotantoprosessi käytännössä

Vesiputousmalli

- **Vesiputousmallissa kuitenkin ongelmia:**
  - Järjestelmää testataan kokonaisuudessaan vasta kun "kaikki" on valmiina
    - Suunnitteluvaiheen virheet saattavat paljastua vasta testauksessa
  - Perustuu oletukselle, että käyttäjä pystyy määrittelemään ohjelmalta halutun toiminnallisuuden heti projektin alussa
    - Näin ei usein kuitenkaan tapahdu, asiakas ei osaa sanoa mitä haluaa, lisäksi projekti voi kestää pitkään, jolloin vaatimuksien muutokset ovat todennäköisempiä

Ohjelmistotuotannon yksi perustavanlaatuisimmista ongelmista on asiakkaan ja toteuttajien välinen kommunikointi!

Ohjelmistotuotantoprosessi käytännössä

Ketterä ohjelmistokehitys

- Lähdetään olettamuksesta, että asiakkaan vaatimukset muuttuvat ja tarkentuvat projektin kuluessa
  - Ei siis yritetäkään kirjoittaa alussa määrittelydokumenttia, joss kirjattuna tyhjentävästi järjestelmältä haluttu toiminnallisuus
- Tuotetaan järjestelmä **iteratiivisesti**, eli pienissä paloissa!
  - Ensimmäisen iteraation aikana tuotetaan pieni osa järjestelmän toiminnallisuutta
    - määritellään vähän, suunnitellaan vähän ja toteutetaan ne
  - Lyhyitä iteraatioita - tyypillisesti muutaman viikon
  - Asiakas antaa palautetta iteraation päätteeksi $\rightarrow$ Korjausliike!
  - Seuraavassa iteraatiossa toteutetaan taas hiukan uutta toiminnallisuutta asiakkaan toiveiden mukaan

Jokaisen iteraation päätteeksi lopputuloksena ohjelmisto, jossa on mukana toiminnallisuutta.

Ohjelmistotuotantoprosessi käytännössä

Ketterä ohjelmistokehitys

Asiakkaan kanssa jatkuva komunikaatio on ketteryyden suola.

- Asiakkaan palaute on välitön
  - Vaatimuksia voidaan tarkentaa ja muuttaa
- Asiakas valitsee jokaisen iteraation aikana toteutettavat lisäominaisuudet
- $\rightarrow$ Todennäköisempää että aikaansaannos toiveiden mukainen
- Iteraation sisällä määrittely, suunnittelu, toteutus ja testaus eivät välttämättä etene peräkkäin (usein jatkuvaa!)
  - Ketterissä menetelmissä dokumentoinnin rooli on yleensä kevyempi kun vesiputousmallissa

Ohjelmistotuotantoprosessi käytännössä

Ketterä ohjelmistokehitys

Virheellinen johtopäätös on ajatella, että kaikki ei-perinteinen tapa tuottaa ohjelmistoja on ketterien menetelmien mukainen!

- Häkkerointi siis ei ole ketterä menetelmä!
- Ketteryys ei tarkoita ettei ole sääntöjä!
- Monissa ketterissä menetelmissä (kuten eXtreme Programming) on päinvastoin erittäin tarkasti määritelty miten ohjelmien laatua hallitaan
- Pariohjelmointi, jatkuva integraatio, automatisoitu testaus, Testaus ensin -lähestymistapa (TDD), ...
- Eli myös ketteryys (voi) vaatii kurinalaisuutta, joskus jopa enemmän kuin perinteinen vesiputousmalli

Mallintaminen

Eli mihin kurssilla keskitytään?

Mallintaminen

Eli mihin kurssilla keskitytään?

Malli on abstrakti kuvaus mielenkiinnon alla olevasta kohteesta

- Perinteiset insinöörialat perustuvat malleihin
  - Esim. siltaa rakentaessa tarkat lujuuslaskelmat (=malli)
  - Näihin perustuen tehdään piirustukset, eli malli siitä miten silta pitää toteuttaa (=edellistä hieman tarkempi malli)
- Kuvaa olennaisen ja VAIN olennaisen
- Käyttötarkoitusta varten liian tarkat tai liian ylimalkaiset mallit epäoptimaalisia
- Mitä on olennaista, riippuu mallin käyttötarkoituksesta
  - Metron linjakartta on hyvä malli julkisen liikenteen käyttäjälle
  - Autoilija taas tarvitsee tarkemman mallin eli tiekartan
  - Helsingin keskustassa kävelija taas tarvitsee tarkempaa kartta

Mallintaminen

Mallin näkökulma ja abstraktiotaso

- Mallien **abstraktiotaso** vaihtelee:
  - Abstraktimpi malli käyttää korkeamman tason käsitteitä
  - Konkreettisempi malli taas on yksityiskohtaisempi ja käyttää "matalamman" tason käsitteitä ja kuvaa kohdetta tarkemmin
- Mallien **näkökulma** vaihtelee:
  - Jos kaikki yritetään mahduttaa samaan malliin, ei lopputulos ole selkeä
  - Malli kuvaa usein korostetusti tiettyä näkökulmaa
  - Eri näkökulmat yhdistämällä saadaan idea kokonaisuudesta
- Käytännössä siis asunnon ostaja tarvitsee *pohjapiirrustuksen* ja sähköasentaja *sähkösuunnitelman* (näkökulmia talosta)

Oikea malli oikeaan tarkoitukseen!

Mallintaminen

Ohjelmistojen mallintaminen

Entä ohjelmistojen mallintaminen?

- Vaatimusdokumentissa mallinnetaan mitä järjestelmän toiminnallisuudelta halutaan
- Suunnitteludokumentissa mallinnetaan...
  - Järjestelmän arkkitehtuuri eli jakautuminen tarkempiin komponentteihin
  - Yksittäisten komponenttien toiminta
- Toteuttaja käyttää näitä malleja ja luo konkreettisen tuotteen
- Vaatimuksien mallit yleensä korkeammalla abstraktiotasolla kuin suunnitelman mallit
  - Vaatimus ei puhu ohjelman sisäisestä rakenteesta toisin kuin suunnitelma

Mallintaminen

Ohjelmistojen mallintaminen

- Myös ohjelmistojen malleilla on erilaisia näkökulmia
- Jotkut mallit kuvaavat rakennetta...
  - Mitä komponentteja järjestelmässä on?
- Jotkut taas toimintaa...
  - Miten komponentit kommunikoivat?

Eri näkökulmat yhdistämällä saadaan idea kokonaisuudesta

Mallintaminen

Mallinnuksen kaksi suuntaa

- Usein mallit toimivat apuna kun ollaan rakentamassa jotain uutta, eli
  - Ensin tehdään malli, sitten rakennetaan esim. silta
- Toisaalta esim. fyysikot tutkivat erilaisia fyysisen maailman ilmiöitä rakentamalla niistä malleja ymmärryksen helpottamiseksi
  - Ensin on olemassa jotain todellista josta sitten luodaan malli
- Ohjelmistojen mallinnuksessa myös olemassa nämä **kaksi mallinnussuuntaa**
  - Ohje toteuttamiselle: malli $\rightarrow$ ohjelma
  - Apuna asiakkaan ymmärtämiseen: ohjelma $\rightarrow$ malli
    - ns. takaisinmallinnus

Ohjelmistojen mallinnuskäytännöt

Oliomallinnus ja UML

Ohjelmistojen mallinnuskäytännöt

Oliomallinnus ja UML

Miten mallintaa ohjelmistoja?

- Pitkään tilanne oli sekava ja on sitä osin edelleen
- Suosituimmaksi tavaksi on noussut **oliomallinnus**
- *Minkä tahansa järjestelmän katsotaan voivan muodostua olioista, jotka yhteistyössä toimien ja toistensa palveluja hyödyntäen tuottavat järjestelmän tarjoamat palvelut*
- Eli järjestelmä tarjoaa joukon palveluja, osa-järjestelmiä, jotka toteuttavat asiakkaan vaatimuksia/toiminnallisuuksia
- Mallinnetaan siis niitä olioita!

Ohjelmistojen mallinnuskäytännöt

Oliomallinnus ja UML

- Unified Modelling Language eli UML
- Oliomallinnusta varten kehitetty standardoitu kuvaustekniikka
  - 1990 luvulta, nykyinen versio 2.5 (vuonna 2015)
- UML:ssä nykyään 13 esityyppistä kaaviota
  - Eri näkökulmille omat kaavionsa
- UML standardi ei määrittele miten ja missä tilanteissa kaavioita tulisi käyttää
  - Tätä varten olemassa useita oliomenetelmiä

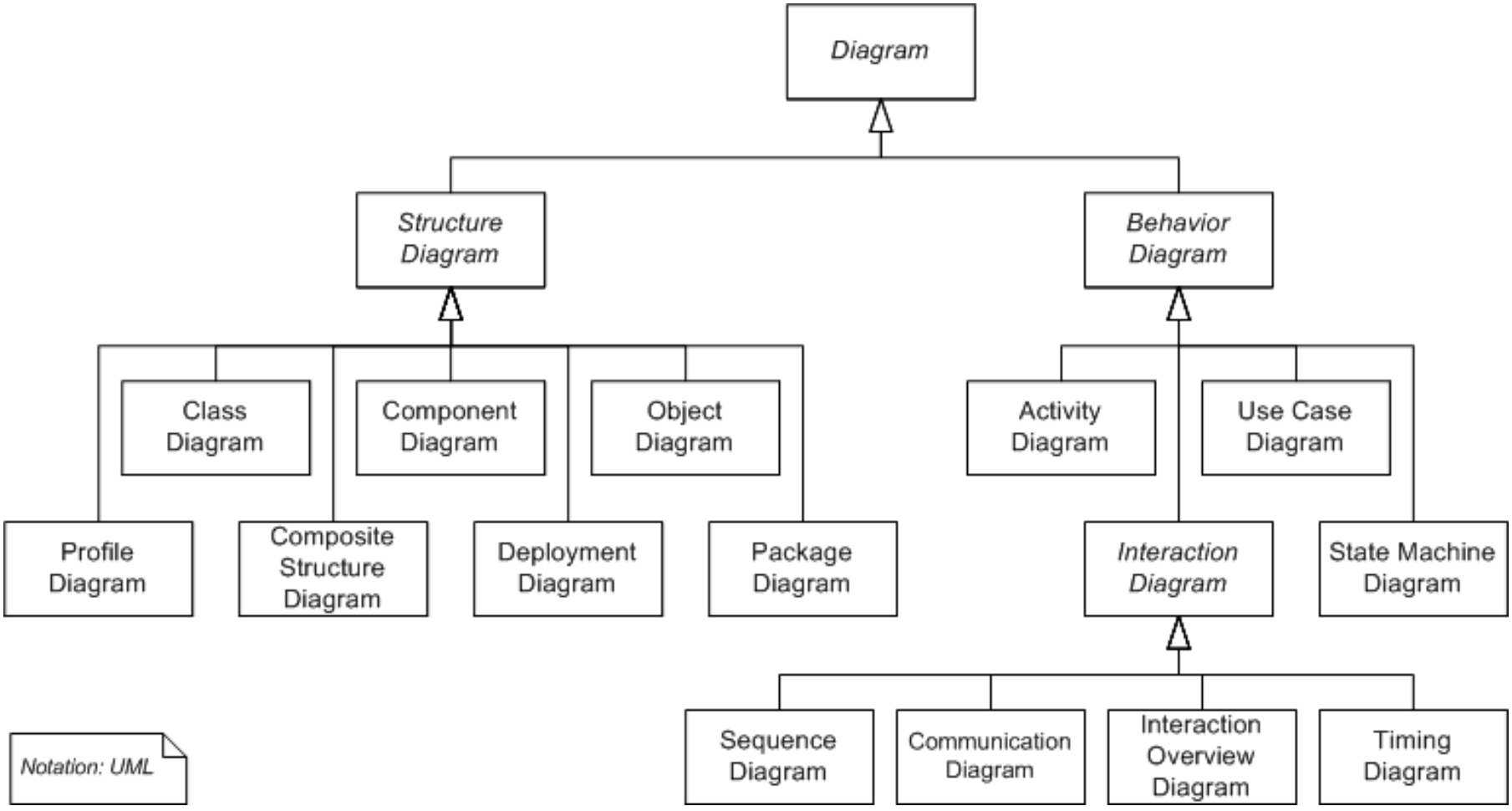
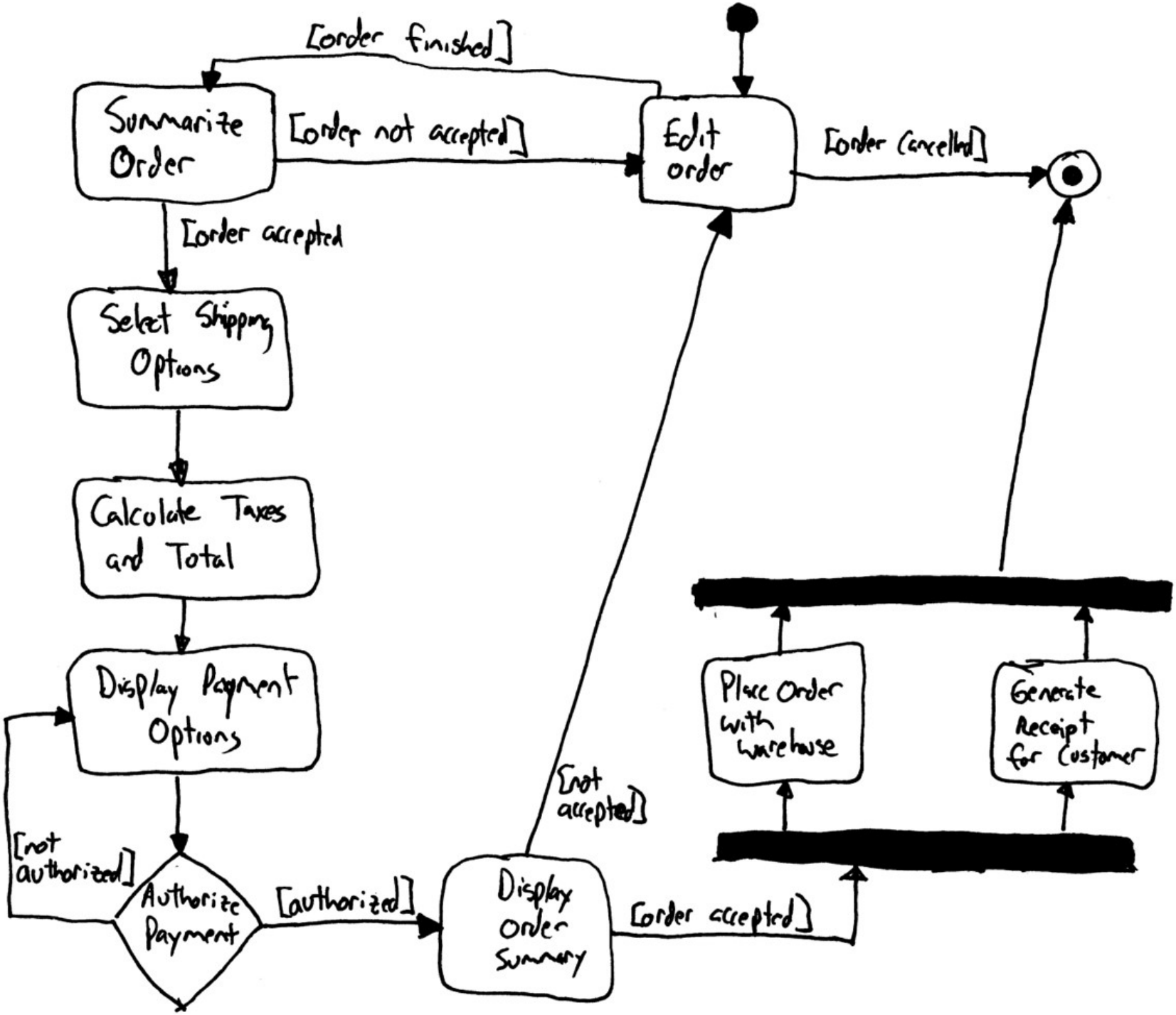


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
Notation: UML

Ohjelmistojen mallinnuskäytännöt

UML:n käyttötapa

- UML-standardi määrittelee kaavioiden syntaksin eli oikeaoppisen piirtotavan suhteellisen tarkasti
  - Eri versioiden välillä pieniä muutoksia
  - Vanhojen standardien mukaisia kaavioita näkyy yhä
- Jotkut suosivat UML:n käyttöä tarkasti syntaksia noudattaen
  - Kaaviot piirretään tällöin usein tietokoneavusteisella suunnitteluvälineellä
  - Nykyään tämä suuntaus alkaa olla jo melko harvinaista
- On myös UML:n luonnosmaisemman käytön puolestapuhujia
  - ns. ketterä mallinnus
  - Kaaviot ennenkaikkia kommunikoinnin apuväline
  - Tärkeimmät kuvat (ehkä) siirretään sähköiseen muotoon
  - Digikuva tai uudelleenpiirto editorilla


[order finished]
Summarize Order
[order not accepted]
Edit order
[order cancelled]
[order accepted]
Select Shipping Options
Calculate Taxes and Total
Display Payment Options
[not authorized]
Authorize Payment
[authorized]
Display order summary
[not accepted]
[order accepted]
Place Order with warehouse
Generate Receipt for Customer

Käyttötapausmalli

Vaatimusanalyysi ja -määrittely

Käyttötapausmalli

Vaatimusanalyysi ja -määrittely

Sovelluksen vaatimukset:

- **Toiminnalliset vaatimukset**
  - Miten ohjelmsito vaikuttaa ympäristöönsä, eli mitä toimintoja ohjelmassa on?
  - *Opetushallinto voi syöttää kurssin tiedot järjestelmään*
  - *Opiskelija voi ilmoittautua valitsemalleen kurssille*
  - *Opettaja voi syöttää opiskelijan suoritustiedot*
- **Ei-toiminnalliset vaatimukset** (eli ympäristön rajoitteet)
- Miten ohjelmiston tulee täyttää toiminnalliset vaatimukset
- Toteutusympäristö, suorituskykyvaatimukset, ...

Ennenkuin voidaan toteuttaa ohjelmisto, joka ratkaisee ongelman on tiedettävä mikä ratkaistava ongelma on!

Käyttötapausmalli

Vaatimusanalyysi ja -määrittely

- **Vaatimusmäärittelyssä** ei oteta kantaa ohjelman sisäisiin teknisiin ratkaisuihin, ainoastaan siihen miten toiminta näkyy käyttäjälle
- Miten toiminnalliset vaatimukset tulisi ilmaista?
- Ratkaisuna **käyttötapausmallit**:
  - Tapa ohjelman toiminnallisten vaatimusten ilmaisemiseen
  - Ei-toiminnallisten vaatimusten ilmaisemiseen käyttötapausmalli ei juuri ota kantaa, vaan ne on ilmaistava muuten
  - On olemassa muitakin tapoja toiminnallisten vaatimusten ilmaisuun esim. kurssilla Ohjelmistotuotanto esiteltävät User storyt eli käyttäjätarintat

Käyttötapausmalli

Vaatimusanalyysi ja -määrittely

- Ohjelmisto tarjoaa käyttäjälleen **palveluita**
  - Ohjelmiston toiminta voidaan kuvata määrittelemällä sen tarjoamat palvelut
- Palveluilla on **käyttäjä**
  - Henkilö, toinen järjestelmä, laite yms. taho, joka on järjestelmän ulkopuolella, mutta tekemisissä järjestelmän kanssa
  - Järjestelmän tiedon hyväksikäyttäjä tai tietojen lähde

Käyttötapausmalli

Käyttäjien tunnistaminen

Hyvä tapa aloittaa vaatimusmäärittely on tunnistaa/etsiä rakennettavan järjestelmän käyttäjät.

- Kysymyksiä jotka auttavat:
  - Kuka/mikä saa tulosteita järjestelmästä?
  - Kuka/mikä toimittaa tietoa järjestelmään?
  - Kuka käyttää järjestelmää?
  - Mihin muihin järjestelmiin kehitettävä järjestelmä on yhteydessä?
- Käyttäjä on oikeastaan **rooli**
  - Missä roolissa toimitaan järjestelmän suhteen
  - Yksi ihminen voi toimia monessa roolissa...

Käyttötapausmalli

Käyttäjien tunnistaminen

TKTL:n kurssi-ilmoittautumisjärjestelmä

- Käyttäjärooleja
  - Opiskelija
  - Opettaja
  - Opetushallinto
  - Suunnittelija
  - Laitoksen johtoryhmä
  - Tilahallintojärjestelmä
  - Henkilöstöhallintajärjestelmä

Osa käyttäjistä yhteydessä järjestelmään vain epäsuorasti.

- Osa "*käyttäjistä*" on muita järjestelmiä
  - Sana käyttäjä ei ole terminä tässä tilanteessa paras mahdollinen
  - Englanninkielinen termi **actor** onkin hieman suomenkielistä termiä kuvaavampi

Käyttötapausmalli

Käyttötapaus

- **Käyttötapaus** (engl. use case) kuvaa käyttäjän ohjelman avulla suorittaman tehtävän.
  - *Miten käyttäjä kommunikoi järjestelmän kanssa tietyssä käyttötilanteessa?*
- Käyttötilanteet liittyvät käyttäjän **tarpeeseen** tehdä järjestelmällä jotain
  - Kurssi-ilmoittautumisjärjestelmä: Opiskelijan ilmoittautuminen
  - Mitä vuorovaikutusta käyttäjän ja järjestelmän välillä tapahtuu kun opiskelija ilmoittautuu kurssille?
- Yksi käyttötapaus on looginen, “isompi” kokonaisuus
  - Käyttötapauksella lähtökohta
  - Ja merkityksen omaava lopputulos
  - Eli pienet asiat, kuten “syötä salasana” eivät ole käyttötapauksia
    - Kyseessä pikemminkin yksittäinen operaatio, joka voi sisältyä käyttötapaukseen

Käyttötapausmalli

Käyttötapauksen kuvaaminen

- **Kuvataan tekstinä**
- Ei ole olemassa täysin vakiintunutta tapaa kuvaukseen (esim. UML ei ota asiaan kantaa)
- Kuvauksessa mukana usein tietyt osat:
  - Käyttötapauksen nimi
  - Käyttäjät
  - Laukaisija
  - Esiehto
  - Jälkiehto
  - Käyttötapauksen kulku
  - Poikkeuksellinen toiminta

- *Käyttäjä:* opiskelija
- *Tavoite:* saada kurssipaikka
- *Laukaisija:* opiskelijan tarve
- *Esiehto:* opiskelija on ilmoittautunut kuluvalla lukukaudella läsnäolevaksi
- *Jälkiehto:* opiskelija on lisätty haluamansa ryhmän ilmoittautujien listalle
- *Käyttötapauksen kulku:*

    1. Opiskelija aloittaa kurssi-ilmoittautumistoiminnon
    2. Järjestelmä näyttää kurssitarjonnan
    3. Opiskelija tutkii kurssitarjontaa
    4. Opiskeljia valitsee ohjelmiston esittämästä tarjonnasta kurssin ja ryhmän
    5. Järjestelmä pyytää opiskelijaa tunnistautumaan
    6. Opiskelija tunnistautuu ja aktivoi ilmoittautumistoiminnon
    7. Järjestelmä ilmoittaa opiskelijalle ilmoittautumisen onnistumisesta.

- *Poikkeuksellinen toiminta:*

    4a. Opiskelija ei voi valita ryhmää, joka on täynnä

    6a. Opiskelija ei voi ilmoittautua, jos hänelle on kirjattu osallistumiseste.

Käyttötapausmalli

Käyttötapauksen kuvaaminen

- **Esiehto**
  - Asioiden tila joka on vallittava, jotta käyttötapaus pystyy käynnistymään
- **Jälkiehto**
  - Kuvaa mikä on tilanne käyttötapauksen onnistuneen suorituksen jälkeen
- **Laukuaisija**
  - Mikä aiheuttaa käyttötapauksen käynnistymisen, voi olla myös ajan kuluminen
- **Käyttötapauksen kulku**
  - Kuvaa onnistuneen suorituksen, usein edellisen sivun tapaan käyttäjän ja koneen välisenä dialogina
- **Poikkeuksellinen toimita**
  - Mitä tapahtuu jos tapahtumat eivät etene onnistuneen suorituksen kuvauksen mukaan
  - Viittaa onnistuneen suorituksen dialogin numeroihin, esim. jos kohdassa 4 voi tapahtua poikkeus normaaliin kulkuun, kuvataan se askeleena 4a

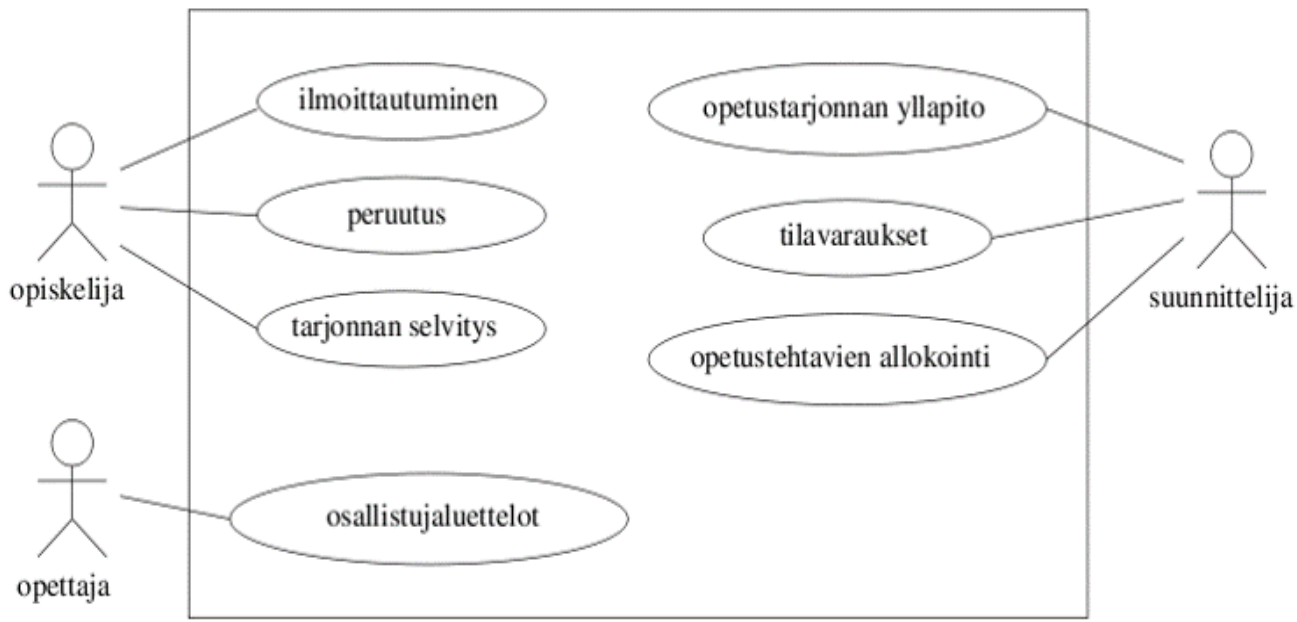
- *Käyttäjä:* opiskelija
- *Tavoite:* perua ilmoittautuminen, välttää sanktiot
- *Laukaisija:* opiskelijan tarve poistaa ilmoittautuminen
- *Esiehto:* opiskelija on ilmoittautunut tietylle kurssille
- *Jälkiehto:* opiskelijan ilmoittautuminen kurssille on poistettu
- *Käyttötapauksen kulku:*

    1. Opiskelija valitsee toiminnon "omat ilmoittautumiset"
    2. Järjestelmä pyytää opiskelijaa tunnistautumaan
    3. Opiskelija tunnistautuu
    4. Järjestelmä näyttää opiskelijan ilmoittautumiset
    5. Opiskeljia valitsee tietyn ilmoittatumisensa ja peruu sen
    6. Järjestelmä ilmoittaa opiskelijalle ilmoittautumisen peruuntumisesta

Käyttötapausmalli

Käyttötapauskaavio

- UML:n käyttötapauskaavion avulla voidaan kuvata käyttötapausten ja käyttäjien (engl. actor) keskinäisiä suhteita
- Kurssi-ilmoittautumisjärjestelmän “korkean tason” käyttötapauskaavio:



Käyttötapausmalli

Käyttötapauskaavio

- Käyttäjät kuvataan tikku-ukkoina
  - Olemassa myös vaihtoehtoinen symboli, joka esitellään pian
- Käyttötapaukset taas kuvataan järjestelmää kuvaavan nelilön sisällä olevina ellipseinä
  - Ellipsin sisällä käyttötapauksen nimi
- Käyttötapausellipsiin yhdistetään viivalla kaikki sen käyttäjät
  - Kuvaan ei siis piirretä nuolia!

HUOM: Käyttötapauskaaviossa ei kuvata mitään järjestelmän sisäisestä rakenteesta

- Esim. vaikka tiedettäisiin että järjestelmä sisältää tietokannan, ei sitä tule kuvata käyttötapausmallissa

Käyttötapausmalli

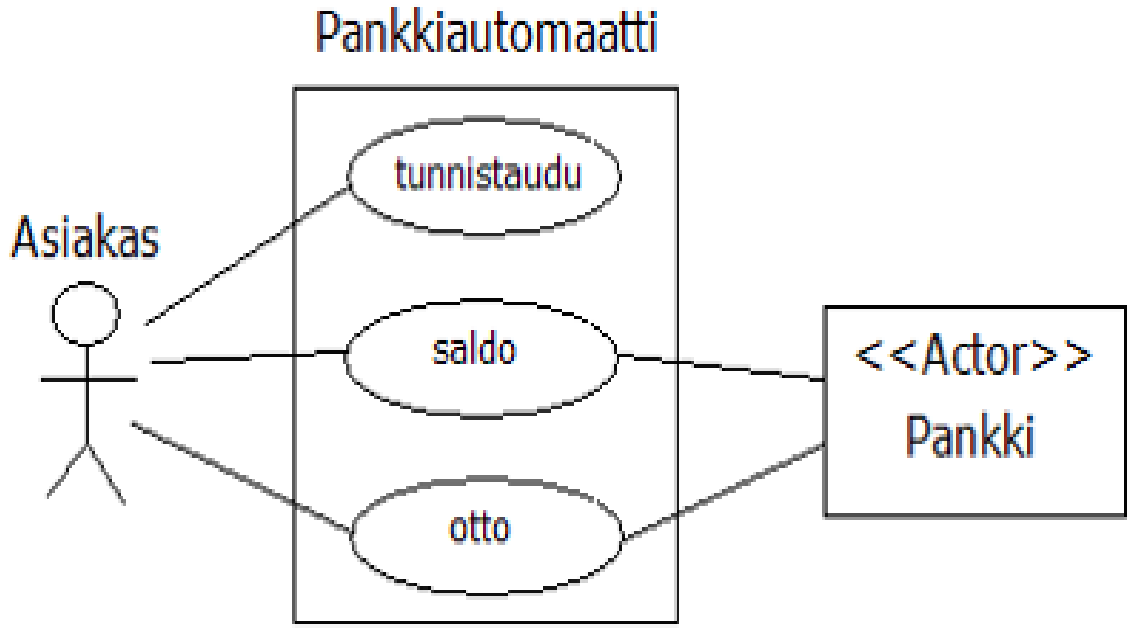
Käyttötapauskaavion käyttö

- Kaaviossa siis käyttötapauksista ainoastaan nimi
  - Käyttötapauksen sisältö kuvataan aina tekstuaalisena esityksenä
- Kaavio tarjoaa hyvän yleiskuvan järjestelmän käyttäjistä ja palveluista
  - Määrittelydokumentin alussa kannattaakin olla käyttötapauskaavio “sisällysluettelona”
- Jokainen käyttötapaus tulee sitten kirjata tekstuaalisesti tarvittavalla tarkkuudella
  - Ei siis ole olemassa standardoitua tapaa käyttötapauksen kirjaamiseen
  - Ohjelmistoprojektissa tulee kuitenkin määritellä käyttötapauspohja, eli sopia joku yhteinen muoto, jota kaikkien käyttötapausten dokumentoinnissa noudatetaan

Käyttötapausmalli

Toinen esimerkki: pankkiautomaatin käyttötapaukset

- Käyttötapaukset ovat tunnistaudu, saldo ja otto
- Käyttötapausten käyttäjät eli toimintaan osallistuvat tahot ovat Asiakas ja Pankki
  - Alla on esitelty tikku-ukolle vaihtoehtoinen tapa merkitä käyttäjä eli laatikko, jossa merkintä «actor»



Käyttötapaus 1: otto

Tavoite Asiakas nostaa tililtään haluamansa määrän rahaa

Käyttäjät Asiakas, Pankki

Esiehto Kortti syötetty ja asiakas tunnistautunut

Jälkiehto käyttäjä saa tililtään haluamansa määrän rahaa.
Jos saldo ei riitä, tiliä ei veloiteta

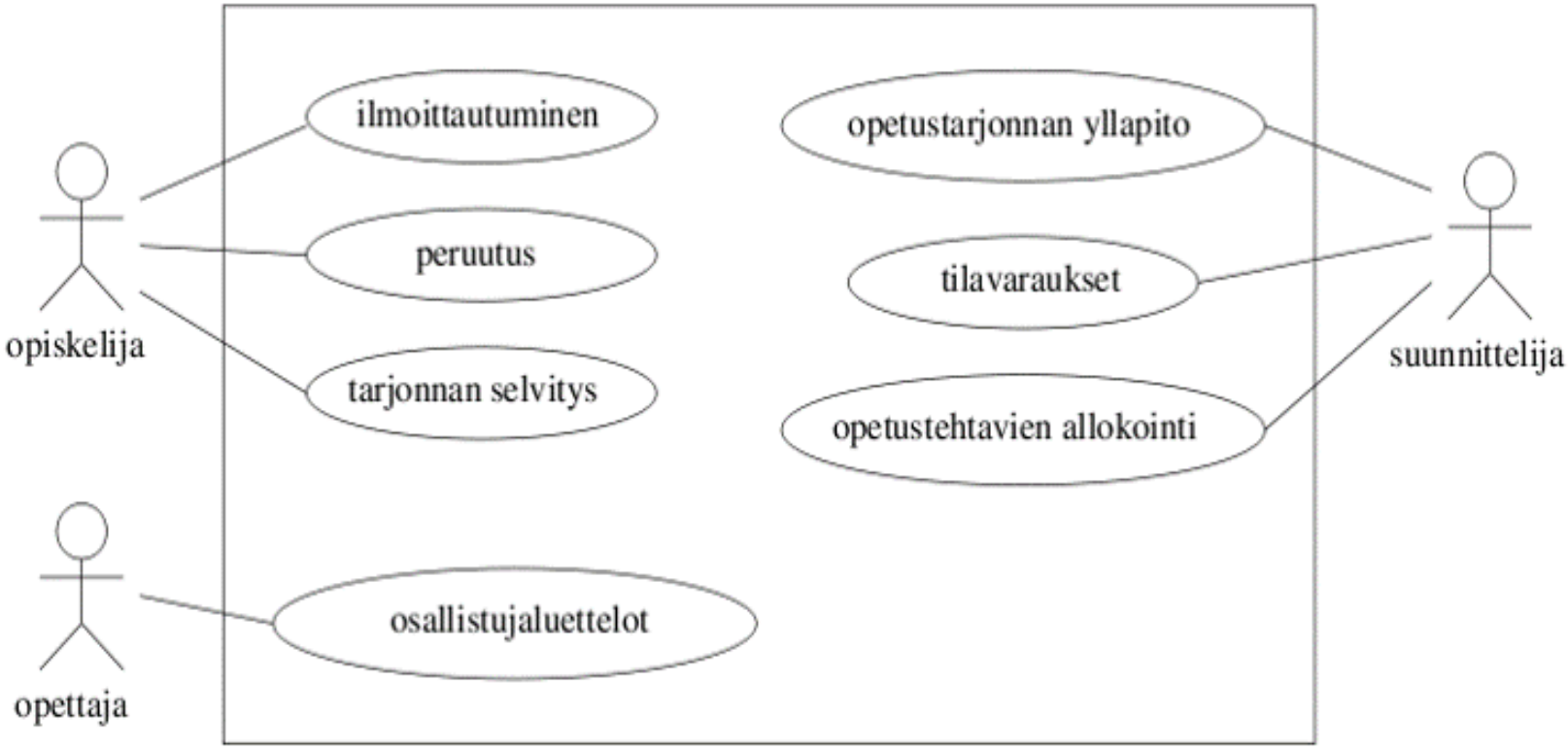
Käyttötapauksen kulku:

1. asiakas valitsee otto-toiminnon
2. automaatti kysyy nostettavaa summaa
3. asiakas syöttää haluamansa summan
4. pankilta tarkistetaan riittääkö asiakkaan saldo
5. summa veloitetaan asiakkaan tililtä
6. kuitti tulostetaan ja annetaan asiakkaalle
7. rahat annetaan asiakkaalle
8. pankkikortti palautetaan asiakkaalle

Käyttötapauksen kulku:

4a asiakkaan tilillä ei tarpeeksi rahaa, palautetaan kortti

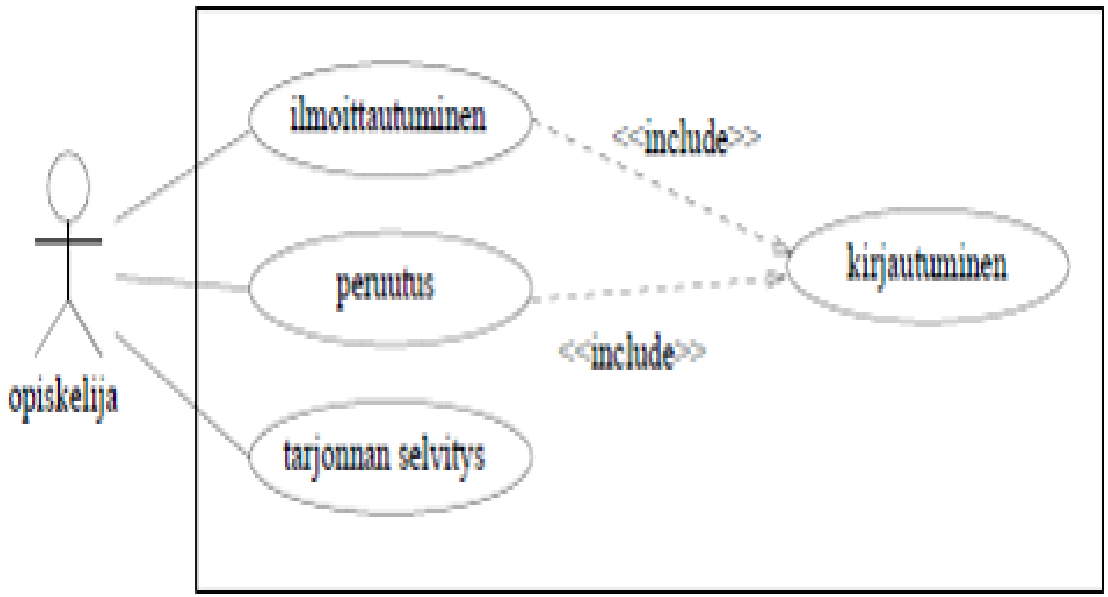
Palataan takaisin kurssi-ilmoittautumisjärjestelmään...



Käyttötapausmalli

Yhteiset osat

- Moneen käyttötapaukseen saattaa liittyä yhteinen osa
- Yhteisestä osasta voidaan tehdä “alikäyttötapaus”, joka sisällytetään (include) pääkäyttötapaukseen
- Käyttötapauskaaviossa tätä varten merkintä «include»
  - katkoviivanuoli pääkäyttötapauksesta apukäyttötapaukseen
- Esim. käyttötapaus kirjautuminen suoritetaan aina kun tehdään ilmoittautuminen tai peruutus



- *Käyttäjä:* opiskelija
- *Tavoite:* saada kurssipaikka
- *Laukaisija:* opiskelijan tarve
- *Esiehto:* opiskelija on ilmoittautunut kuluvalla lukukaudella läsnäolevaksi
- *Jälkiehto:* opiskelija on lisätty haluamansa ryhmän ilmoittautujien listalle
- *Käyttötapauksen kulku:*

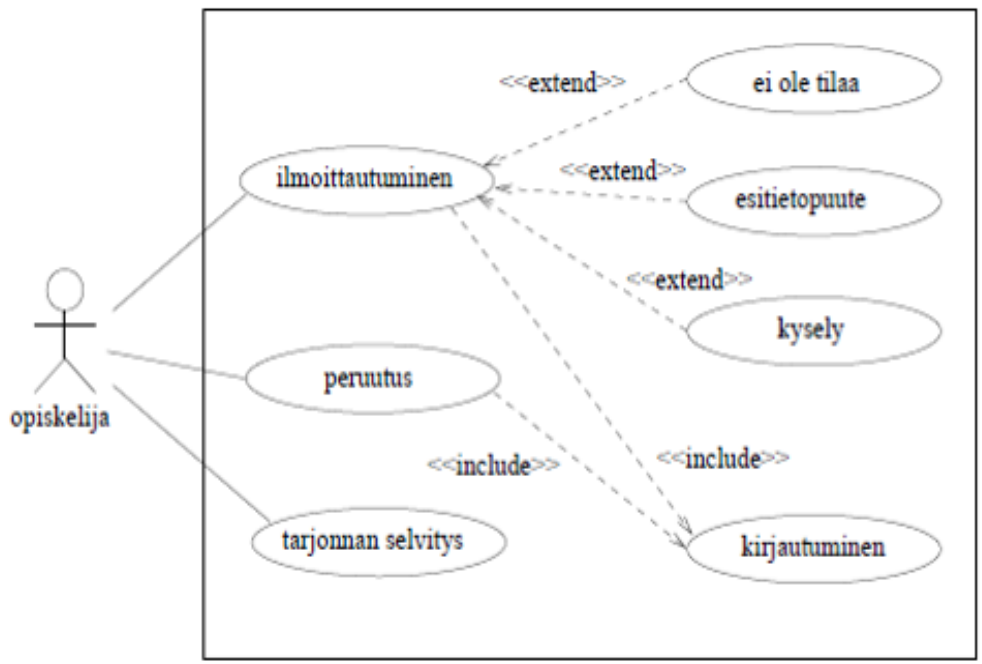
    1. Opiskelija aloittaa kurssi-ilmoittautumistoiminnon
    2. Järjestelmä näyttää kurssitarjonnan
    3. Opiskelija tutkii kurssitarjontaa
    4. Opiskeljia valitsee ohjelmiston esittämästä tarjonnasta kurssin ja ryhmän
    5. **Suoritetaan käyttötapaus kirjautuminen**
    6. Järjestelmä ilmoittaa opiskelijalle ilmoittautumisen onnistumisesta.

- *Poikkeuksellinen toiminta:*

Käyttötapausmalli

Poikkeustilanteet ja laajennukset

- Sisällytettävä (eli **include**) käyttötapaus suoritetaan aina pääkäyttötapauksen suorituksen yhteydessä
- Myös tarvittaessa suoritettava laajennus tai poikkeustilanne voidaan kuvata apukäyttötapauksena, joka laajentaa (**extend**) pääkäyttötapausta
  - Laajennus suoritetaan siis vaan tarvittaessa
- Esim. Ilmoittautuessa saatetaan huomata esitietopuute, jonka käsittely on oma käyttötapauksensa



Käyttötapausmalli

Poikkeustilanteet ja laajennukset

Huomaa, että laajennuksessa nuolensuunta on apukäyttötapauksesta pääkäyttötapaukseen päin (toisin kuin sisällytyksessä)

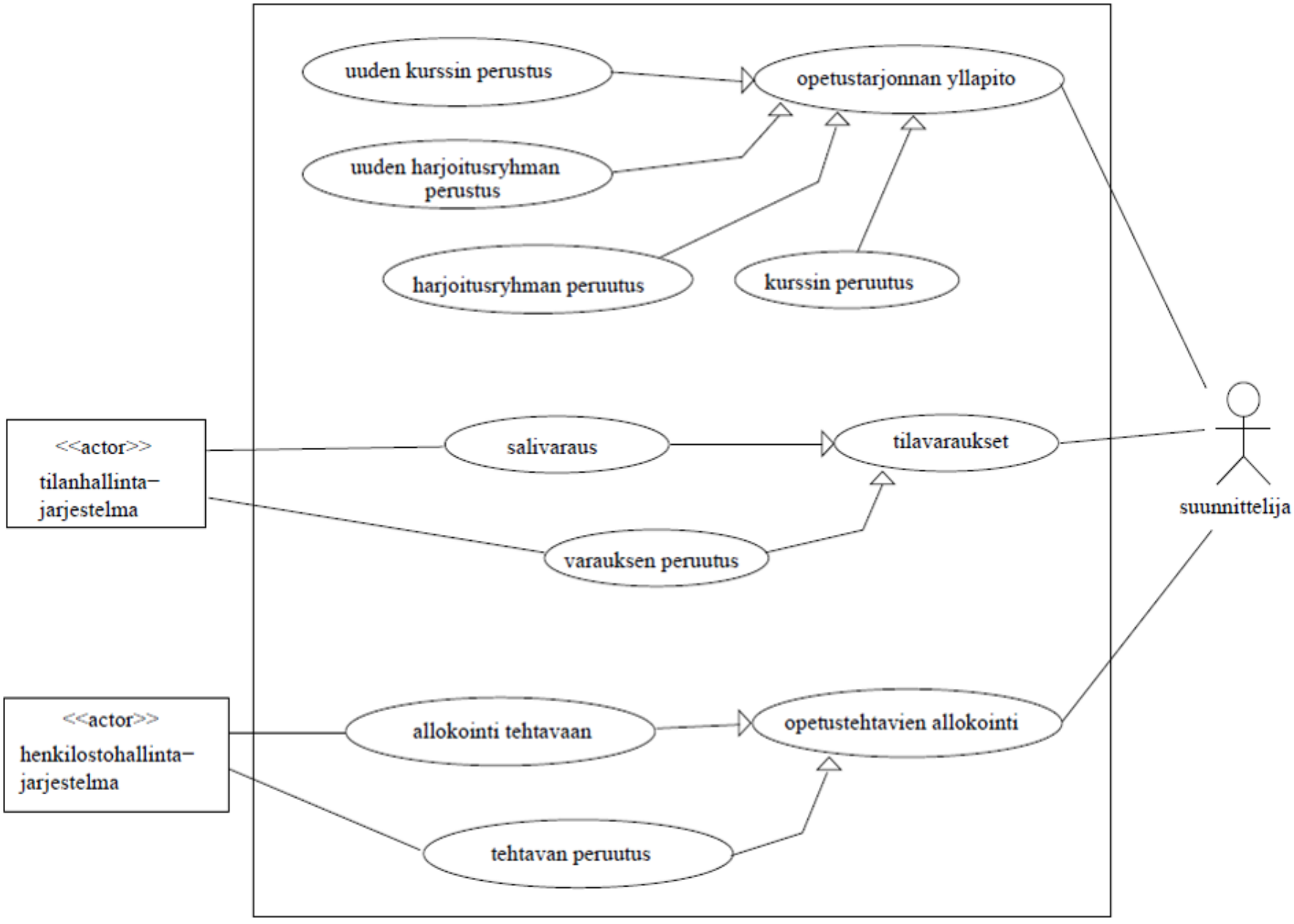
- Laajennus tulee ehdottomasti merkitä käyttötapauksen tekstuaaliseen kuvaukseen
- Edellisen dian laajennusesimerkki ei ole erityisen onnistunut
  - Laajennuksienkin pitäisi olla kunnollisia käyttötapauksia (eli asioita joilla on selkeä tavoite), ei metodikutsumaisia kyselyjä tai ilmoituksia (kuten ei tilaa - tai esitietopuute-ilmoitus)
- Poikkeustilanteet on parempi kuvata tekstuaalisessa esityksessä ja jättää ne kokonaan pois käyttötapauskaavioista
- Koko laajennuskäsitteen tarve käyttötapauskaavioissa on hieman kyseenalainen

Käyttötapausmalli

Yleistetty ja erikoistettu käyttötapaus

- Suunnittelijan käyttötapauksista erityisesi opetustarjonnan ylläpito on hyvin laaja tehtäväkokonaisuus
- Voidaankin ajatella, että kyseessä on *yleistetty käyttötapaus*, joka oikeasti pitääkin sisällään useita konkreettisia käyttötapauksia, kuten
- Esim. voidaan ajatella, että kurssin peruutus on osa yleisettyä käyttötapausta, jota voimme kutsua opetustarjonnan ylläpidoksi

Yleisetyt käyttötapaukset merkitään nuolella, jossa on nuolenkärki osoittamaan siihen suuntaan, johon käyttötapaus on yleistetty

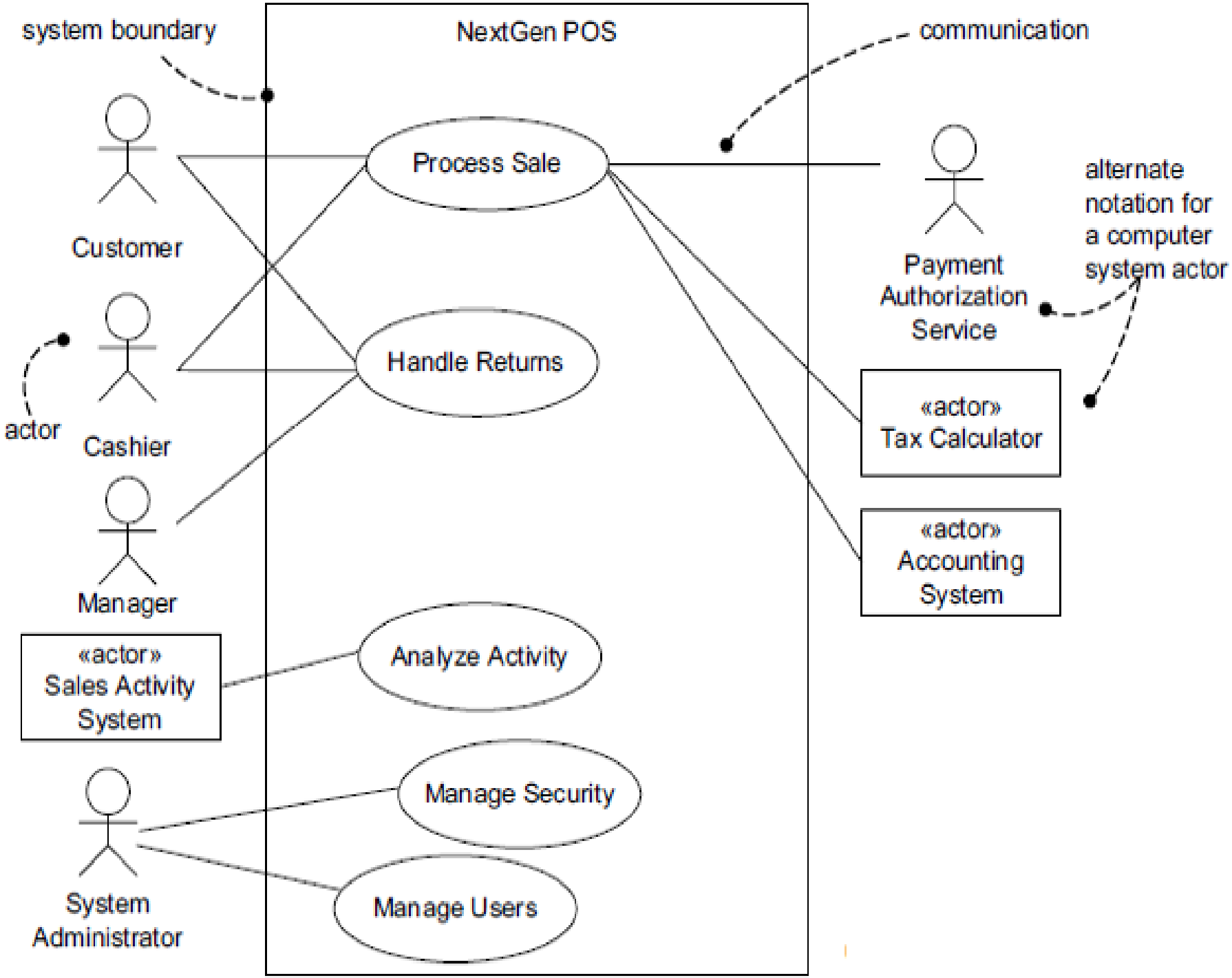

uuden kurssin perustus
opetustarjonnan yllapito
uuden harjoitusryhman perustus
harjoitusryhman peruutus
kurssin peruutus
<<actor>>
tilanhallinta–jarjestelma
salivaraus
tilavaraukset
varauksen peruutus
suunnittelija
<<actor>>
henkilostohallinta–jarjestelma
allokointi tehtavaan
opetustehtavien allokointi
tehtavan peruutus

Käyttötapausmalli

Yleistetty ja erikoistettu käyttötapaus

- Craig Larmanin kirjasta *Applying UML and Patterns* [2]
- Aluksi etsitään järjestelmän käyttäjät
- Mietitään käyttäjien tavoitteita: mitä käyttäjä haluaa saada järjestelmällä tehtyä
- Käyttäjän tavoitteellisista toiminnoista (esim. käsittele ostos) tulee tyypillisesti käyttötapauksia
- Samalla saatetaan löytää uusia käyttäjiä (erityisesti ulkoisia järjestelmiä joihin järjestelmä yhteydessä)
- Hahmotellaan alustava käyttötapausdiagrammi $\longrightarrow$

[2] Kirjan käyttötapausluku löytyy verkosta:
http://www.craiglarman.com/wiki/index.php?title=Articles


system boundary
NextGen POS
communication
Process Sale
Customer
Payment Authorization Service
alternate notation for a computer system actor
Handle Returns
actor
Cashier
«actor» Tax Calculator
«actor» Accounting System
Manager
«actor» Sales Activity System
Analyze Activity
Manage Security
System Administrator
Manage Users

Käyttötapausmalli

Yleistetty ja erikoistettu käyttötapaus

- Otetaan aluksi tarkasteluun järjestelmän toiminnan kannalta kriittisimmät käyttötapaukset
- Ensin kannattanee tehdä vapaamuotoinen kuvaus käyttötapauksista ("brief use case")

Process Sale: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.

Kuva: POS = point of sales terminal eli kassapääte

- Tarkempi käyttötapaus kirjoitetaan projektin sopiman käyttötapauspohjan määräämässä muodossa

Use Case UC1: Process Sale

Primary Actor: Cashier
Preconditions: Cashier is identified and authenticated.
Success Guarantee (Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

Main Success Scenario (or Basic Flow):

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

*a. At any time, System fails:

To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.

1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.
   2a. System detects anomalies preventing recovery:
      1. System signals error to the Cashier, records the error, and enters a clean state.
      2. Cashier starts a new sale.

3a. Invalid identifier:

1. System signals error and rejects entry.

3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers):

1. Cashier can enter item category identifier and the quantity.

3-6a: Customer asks Cashier to remove an item from the purchase:

1. Cashier enters item identifier for removal from sale.
2. System displays updated running total.

3-6b. Customer tells Cashier to cancel sale:

1. Cashier cancels sale on System.

3-6c. Cashier suspends the sale:

1. System records sale so that it is available for retrieval on any POS terminal.

7a. Paying by cash:

1. Cashier enters the cash amount tendered.
2. System presents the balance due, and releases the cash drawer.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

7b. Paying by credit:

1. Customer enters their credit account information.
2. System sends payment authorization request to an external Payment Authorization Service System, and requests payment approval.
   2a. System detects failure to collaborate with external system:
      1. System signals error to Cashier.
      2. Cashier asks Customer for alternate payment.
3. System receives payment approval and signals approval to Cashier.
   3a. System receives payment denial:
      1. System signals denial to Cashier.
      2. Cashier asks Customer for alternate payment.
4. System records the credit payment, which includes the payment approval.
5. System presents credit payment signature input mechanism.
6. Cashier asks Customer for a credit payment signature. Customer enters signature.

Käyttötapausmalli

Tarkkaan kuvattu käyttötapaus

- Esimerkin mallin mukaan käyttötapauksen pääkulku kannattaa kuvata tiiviisti
  - Eri askeleiden sisältöä voi tarvittaessa tarkentaa (askel 7)
- Huomioi tapa, miten poikkeusten ja laajennusten sijainti pääkulussa merkitään
  - 7a $\longrightarrow$ laajentaa/tarkentaa pääkulun kohtaa 7

Osa jossa laajennukset, tarkennukset ja poikkeukset dokumentoidaan, on usein paljon pidempi kuin normaali kulku

- Koska kyse vaatimusmäärittelystä, kuvaus toteutetaan abstraktilla tasolla eli...
  - Ei oteta kantaa toteutusyksityiskohtiin
  - eikä käyttöliittymään
  - Esim. tunnistetaanko ostos viivakoodin perusteella...

Käyttötapausmalli

Yhteenveto

- Käyttötapaukset ovat yksi tapa kuvata ohjelmiston toiminnallisia vaatimuksia
- **Käyttötapauksen tekstuaalinen esitys oleellinen**
- Ohjelmistoprojektissa pitää sopia yhteinen tapa (*käyttötapauspohja*) käyttötapausten tekstuaaliseen esitykseen
- Käyttötapauskaavion merkitys lähinnä yleiskuvan antaja
- Jos huomaat käyttäväsi paljon aikaa “oikeaoppisen” käyttötapauskaavion piirtämiseen, ryhdy välittömästi tekemään jotakin hyödyllisempää (esim. käyttötapausten tekstuaalisia esityksiä)

Ohjelmistotekniikan menetelemät
Luento 2

Testaus ja oliokaaviot

Testaus

Testaustasot

- **Yksikkötestaus**
  - Toimivatko yksittäiset metodit ja luokat kuten halutaan?
- **Integraatiotestaus**
  - Toimivatko yksittäiset *moduulit* yhdessä halutulla tavalla?
- **Järjestelmä/hyväksymistestaus**
  - Toimiiko kokonaisuus niin kuin vaatimusdokumentissa sanotaan?
- **Regressiotestaus**
  - *regressio* $\approx$ palautuminen, taantuminen, takautuminen
  - järjestelmälle muutosten ja bugikorjausten jälkeen ajettavia testejä jotka varmistavat että muutokset eivät riko mitään
  - Regressiotestit koostuvat yleensä yksikkö-, integraatio- ja järjestelmä/hyväksymätesteistä

Nykyinen trendi on tehdä testeistä automaattisesti suoritettavia

Testaus

Yksikkötestit

```
public class LyyraKorttiTest {

    private LyyraKortti kortti;

    @Before
    public void setUp() {
        kortti = new LyyraKortti(10);
    }

    @Test
    public void konstruktoriAsettaaSaldonOikein() {
        assertEquals("Kortilla on rahaa 10.0 euroa", kortti.toString());
    }

    @Test
    public void syoEdullisestiVahentaaSaldoaOikein() {
        kortti.syoEdullisesti();
        assertEquals("Kortilla on rahaa 7.5 euroa", kortti.toString());
    }
}
```

Testaus

Yksikkötestit

- Yksikkötesteillä testataan yksittäistä "*asiaa*", esim. toimiiko metodi oikein tietyntyylisellä syötteellä
- Yhtä kokonaisuutta (esim. luokkaa) testaavat testitapaukset sijoitetaan yhden testiluokan sisälle
- Hyvät testit ovat kattavat eli testaavat kaiken koodin monipuolisesti
- Normaalien syötteiden lisäksi on testeissä erityisen tärkeää tutkia testattavien metodien toimintaa virheellisillä syötteillä ja erilaisilla raja-arvoilla

$x = y$	$x > y$	x instaceof y
$x < y$	$x \neq y$	...

Testaus

JUnit

- Testaamisen harjoitteluun käytetään kurssilla **JUnit** kirjastoa
- Testitapaukset (metodit) annotoidaan @Test avainasanalla
- Ennen jokaista testiä suoritetaan @Before -metodi
- Jokaisen testin jälkeen suoritetaan @After
- Ennen jokaista testiluokkaa suoritetaan @BeforeClass -metodi
- Testiluokan suorituksen jälkeen ajetaan @AfterClass -metodi

Testausta oppii parhaiten tekemällä, joten jätetään yksikkötestaus seuraaviin laskuharjoituksiin ja siirrytään eteenpäin!

Oliot ja luokat

Jokainen olio kuuluu johonkin luokkaan

Oliot ja luokat

Jokainen olio kuuluu johonkin luokkaan

Olio...

- on ohjelman tai sen sovellusalueen kannalta *mielenkiintoinen* asia tai käsite, tai ohjelman osa
- yleensä yhdistää tietoa ja toiminnallisuutta
- omaa identiteetin, eli erottuu muista olioista omaksi yksilökseen
- kuuluu johonkin luokkaan

Minkä tahansa järjestelmän katsotaan voivan muodostua olioista, jotka yhteistyössä toimien ja toistensa palveluja hyödyntäen tuottavat järjestelmän tarjoamat palvelut

Oliot ja luokat

Suunnittelu ja toteutusvaiheen oliot ja luokat

Henkilö-luokka:

```
public class Henkilo {
    private String nimi;
    private int ika;

    public Henkilo(String n) {
        nimi = n;
    }

    public void vanhene() {
        ika++;
    }
}
```

Henkilö-olioita:

```
Henkilo arto =
new Henkilo("Arto");

Henkilo heikki =
new Henkilo("Heikki");

arto.vanhene();
heikki.vanhene();
```

Oliot ja luokat

Suunnittelu ja toteutusvaiheen oliot ja luokat

- *Luokka* kuvaa minkälaisia siihen kuuluvat oliot ovat tietosisällön ja toiminnallisuuden suhteen
- Oliota ja luokkia ajatellaan usein ohjelmointitason käsitteinä
  - Ohjelma muodostuu olioista
  - Oliot elävät koneen muistissa
  - Ohjelman toiminnallisuus muodostuu olioiden toiminnallisuudesta
- Ohjelmiston suunnitteluvaiheessa suunnitellaan mistä oliosta ohjelma koostuu ja miten oliot kommunikoivat
  - Nämä oliot sitten toteutetaan ohjelmointikielellä toteutusvaiheessa

Mistä suunnitteluvaiheen oliot tulevat? Miten ne keksitään?

Oliot ja luokat

Vaatimusanalyysivaiheen oliot ja luokat

- Vaatimusmäärittelyn yhteydessä tehdään usein *vaatimusanalyysi*
  - Kartoitetaan ohjelmiston sovellusalueen (eli sovelluksen kohdealueen) kannalta **tärkeitä käsitteitä ja niiden suhteita**
- Mietitään mitä tärkeitä asioita sovellusalueella on olemassa
  - Esim. kurssihallintojärjestelmän käsitteitä ovat...
    - Kurssi
    - Laskariryhmä
    - Ilmoittautuminen
    - Opettaja
    - Opiskelija
    - Sali
    - Salivaraus
  - Nämä käsitteet voidaan ajatella luokkina!

Oliot ja luokat

Vaatimusanalyysivaiheen oliot ja luokat

- **Vaatimusanalyysivaiheen luokat ovat vastineita reaalimaailman käsitteille**
- Kun edetään vaatimuksista ohjelmiston suunnitteluun, monet vaatimusanalyysivaiheen luokista saavat vastineensa ”ohjelmointitason” luokkina, eli luokkina, jotka on tarkoitus ohjelmoida esim. Javalla
- Eli riippuen katsantokulmasta, luokka voi olla joko
  - reaalimaailman käsitteen vastine, tai
  - suunnittelu- ja ohjelmointitason ”tekninen” asia
- Tyypillisesti ohjelmatason olio on vastine jollekin todellisuudessa olevalle ”oliolle” (*simuloidaan todellisuutta*)
- Ohjelmissa on myös paljon luokkia ja olioita, joille ei ole vastinetta todellisuudessa (Esim. käyttöliittymän oliot)

Oliot ja luokat

Oliomallinnus ja olioperustainen ohjelmistokehitys

- Olioperustainen ohjelmistokehitys etenee yleensä seuraavasti:
  1. Luodaan **määrittelyvaiheen oliomalli** sovelluksen käsitteistöstä
     - Mallin oliot ja luokat ovat rakennettavan sovelluksen kohdealueen käsitteiden vastineita
  2. Suunnitteluvaiheessa tarkennetaan edellisen vaiheen oliomalli **suunnitteluvaiheen oliomalliksi**
     - Oliot muuttuvat yleiskäsitteistä teknisen tason olioiksi
     - Mukaan tulee olioita, joilla ei suoraa vastinetta reaalimaailmassa
     - Osa olioista on luonteeltaan *pysyviä* ja niitä tulee vastaamaan jokin rakenne ohjelman tietokannassa
  3. Toteutetaan suunnitteluvaiheen oliomalli jollakin **olio-ohjelmointikielellä**
- Voidaankin ajatella, että malli tarkentuu muuttuen koko ajan ohjelmointikieliläheisemmäksi/teknisemmäksi siirryttäessä määrittelystä suunnitteluun ja toteutukseen

Luokka- ja oliokaaviot

Olioden ja luokkien kuvaus UML:ssä

Luokka- ja oliokaaviot

Olioden ja luokkien kuvaus UML:ssä

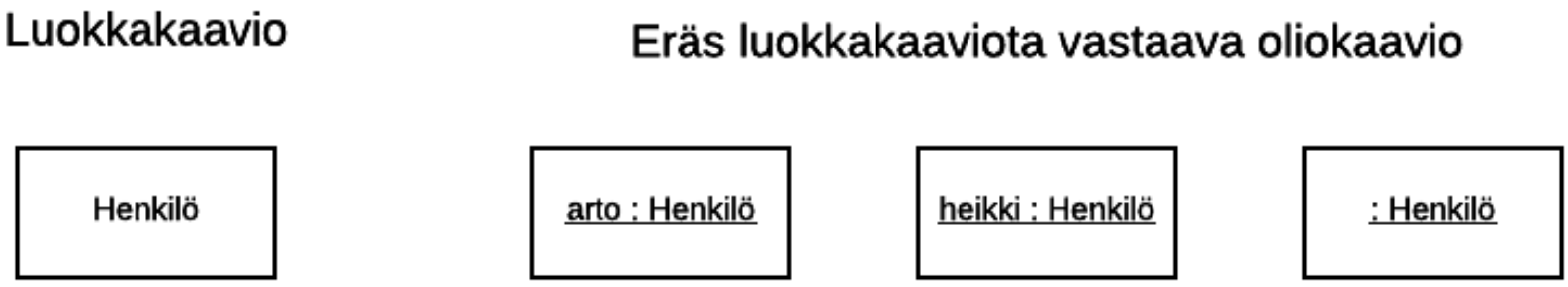
- Miten olioita ja luokkia voidaan kuvata UML:ssä[3]?
- Järjestelmän luokkarakennetta kuvaa **luokkakaavio** (engl. *class diagram*)
  - Mitä luokkia olemassa
  - Minkälaisia luokat ovat
  - Luokkien ja niiden olioiden suhteet toisiinsa
- Luokkakaavio on UML:n eniten käytetty kaaviotyyppi
- Luokkakaavio kuvaa ikäänkuin kaikkia mahdollisia olioita, joita järjestelmässä on mahdollista olla olemassa
- **Oliokaavio** (engl. *object diagram*) taas kuvaa mitä olioita järjestelmässä on tietyllä hetkellä

[3] Mikä on UML? Luennon 1 kalvoissa on kaikki mitä tarvitset.

Luokka- ja oliokaaviot

Olioden ja luokkien kuvaus UML:ssä

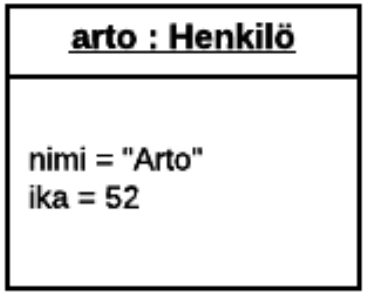
- Luokkaa kuvataan laatikolla, jonka sisällä on luokan nimi
- Luokasta luotuja olioita kuvataan myös laatikolla, erona on nimen merkintätapa
  - Nimi alleviivattuna, sisältäen mahdollisesti myös olion nimen
- Kuvassa Henkilö-luokka ja kolme Henkilö-olioa
- Luokkia ja olioita ei sotketa samaan kuvaan, kyseessä onkin kaksi kuvaa: vasemmalla luokkakaavio ja oikealla oliokaavio
  - Oliokaavio kuvaa tietyn hetken tilanteen, olemassa 3 henkilöä, joista yksi on nimetön

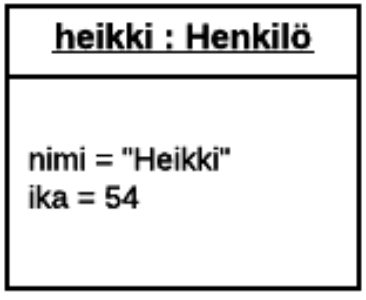


Luokka- ja oliokaaviot

Attribuutit

- Luokan olioilla on attribuutteja eli oliomuuttujia ja operaatiota eli metodeja
- Nämä määritellään luokkamäärittelyn yhteydessä
  - Aivan kuten Javassa kirjoitettaessa class Henkilö ... määritellään kaikkien Henkilö:n attribuutit ja metodit luokan määrittelyn yhteydessä
- Luokkakaaviossa attribuutit määritellään luokan nimen alla omassa osassaan laatikkoa
  - Attribuutista on ilmaistu nimi ja tyyppi (voi myös puuttua)
- Oliokaaviossa voidaan ilmaista myös attribuutin arvo





Luokka- ja oliokaaviot

Metodit

- Luokan olioiden metodit merkitään laatikon kolmanteen osaan
- Luokkiin on **pakko merkitä ainoastaan nimi**
  - Attribuutit ja metodit merkitään jos tarvetta
  - Usein metodeista merkitään ainoastaan nimi, joskus myös parametrien ja paluuarvon tyyppi
- Attribuuttien ja operaatioiden parametrien ja paluuarvon tyyppeinä voidaan käyttää valmiita tietotyyppejä (int, double, String, ...) tai rakenteisia tietotyyppejä (esim. taulukko, ArrayList).
- Tyyppi voi olla myös luokka, joko itse määritelty tai asiayhteydestä "itsestäänselvä"(alla Väri ja Piste)

Luokka- ja oliokaaviot

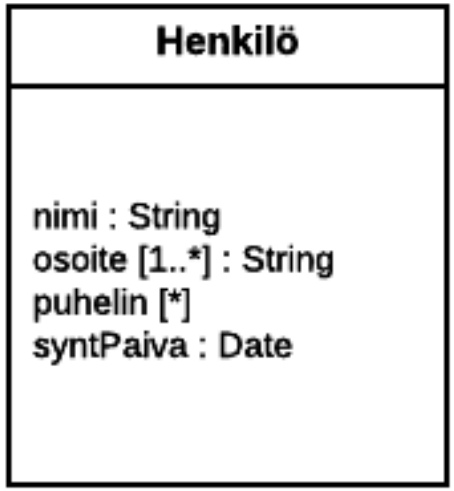
Attribuuttien ja operaatioiden näkyvyys

- Ohjelmointikielissä voidaan attribuuttien ja metodien näkyvyyttä muiden luokkien olioille säädellä
  - Javassa private, public, protected
- UML:ssa näkyvyys merkitään attribuutin tai metodin eteen: public $+$, private $-$, protected $\#$, package $\sim$
  - Jos näkyvyyttä ei ole merkitty, sitä ei ole määritelty (Kovin usein näkyvyyttä ei viitsitä merkitä)
- Esim. alla kaikki attribuutit ovat private eli eivät näy muiden luokkien olioille, metodit taas public eli kaikille julkisia

Luokka- ja oliokaaviot

Attribuutin moniarvoisuus

- Jos attribuutti on kokoelma samanlaisia arvoja (esim. taulukko), voidaan se merkitä luokkakaavioon
- Kirjoitetaan attribuutin perään hakasulkeissa kuinka monesta "asiasta"attribuutti toistuu ($*$ = tuntematon)
- Henkilöllä on vähintään 1 osoite, mutta osoitteita voi olla useita
- Puhelinnumeron kohdalle on kirjoitettu [$*$], joka tarkoittaa, että numeroa ei välttämättä ole tai numeroita voi olla useita
- **Moniarvoisuus on asia, joka merkitään kaavioon harvoin!**



Luokka- ja oliokaaviot

VahenevaLaskuri esimerkki

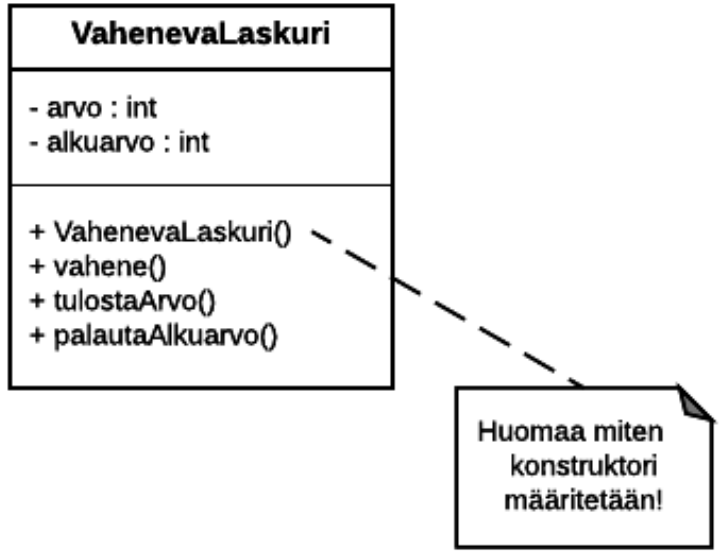
```
public class VahenevaLaskuri {
    private int arvo;
    private int alkuarvo;

    public VahenevaLaskuri(int arvo) {
        this.arvo = arvo;s
        this.alkuarvo = arvo;
    }

    public void vahene() {
        if ( this.arvo>0 ) {
            this.arvo--;
        }
    }

    public void tulostaArvo() {
        System.out.prntln(this.arvo);
    }

    public void palautaAlkuarvo() {
        this.arvo = this.alkuarvo;
    }
}
```

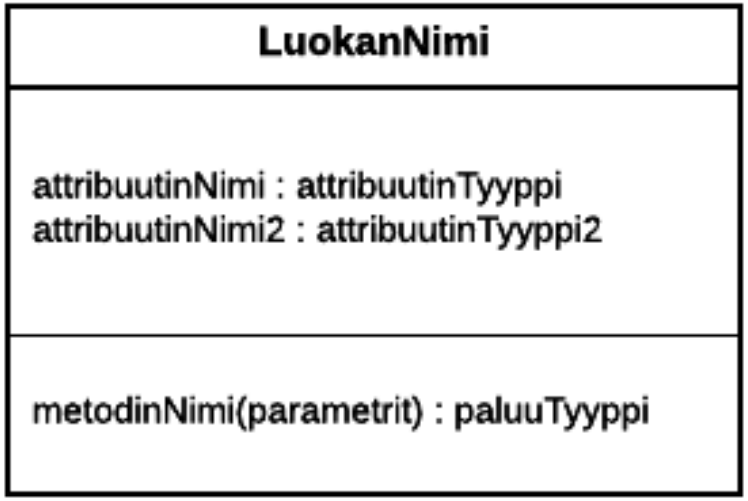


Kuvassa mukana UML-kommenttisymboli!

Luokka- ja oliokaaviot

Yhteenveto

- Eli luokka on laatikko, jossa luokan nimi ja tarvittaessa attribuutit sekä metodit
- Attribuuttien ja metodien parametrien ja paluuarvon tyyppi ilmaistaan tarvittaessa
  - Näkyvyysmääreet ilmaistaan tarvittaessa
- Parametrin tyyppi voidaan myös merkitä "Javamaisesti", esim. `Date : aika` tai `int : ikä`
  - myös kaksoispisteen voi jättää kokonaan pois...
- Jos esim. metodeja ei haluta näyttää, jätetään metodiosa pois, vastaavasti voidaan menetellä attribuuttien suhteen



Olioiden väliset yhteydet

Yleisesti yhteyksistä

- **Olioiden välillä on yhteyksiä:**
  - Työntekijä *työskentelee* Yrityksessä
  - Henkilö *omistaa* Auton
  - Henkilö *ajaa* Autolla
  - Auto *sisältää* Renkaat
  - Henkilö *asuu* Osoitteessa
  - Henkilö *omistaa* Osakkeita
  - Työntekijä *on* Johtajan alainen
  - Johtaja *johtaa* Työntekijöitä
  - Johtaja *erottaa* Työntekijän
  - Opiskelija *on* ilmoittautunut Kurssille
  - Kello *sisältää* kolme Viisaria
- Olioiden välinen yhteys voi olla pysyvämpiluontoinen (rakenteinen) tai hetkellinen
  - Aluksi keskitytään pysyvämpiluontoisiin yhteyksiin

Olioiden väliset yhteydet

Yleisesti yhteyksistä

- Ohjelmakoodissa pysyvä yhteys ilmenee yleensä luokassa olevana olioviitteenä, eli oliomuuttujana jonka tyyppinä on luokka
- Esimerkiksi **Henkilö** omistaa **Auton**:

```
public class Auto {
    public void aja() {
        System.out.println("liikkuu");
    }
}

public class Henkilö {
    private Auto omaAuto;

    public Henkilö(Auto auto) {
        omaAuto = auto
    }

    public void meneTöihin() {
        omaAuto.aja();
      }
}
```

Olioiden väliset yhteydet

Assosiaatio

Olioiden väliset yhteydet

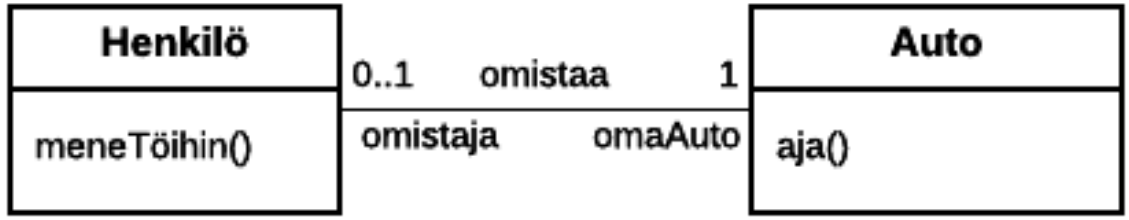
Assosiaatio

- Olioviite voitaisiin periaatteessa merkitä luokkakaavioon attribuuttina, kyseessähän on teknisessä mielessä attribuutti
- Parempi tapa on kuvata olioiden välinen yhteys luokkakaaviossa
  - Jos Henkilö- ja Auto-olion välillä voi olla yhteys, yhdistetään Henkilö- ja Auto-luokat viivalla
- Tilanne kuvattu alla
  - Yhteydelle on annettu nimi omistaa, eli Henkilö-olio omistaa Auto-olion

Olioiden väliset yhteydet

Kytkentärajoitteet ja roolit

- Ohjelmakoodissa jokaisella henkilöllä on täsmälleen yksi auto ja auto liittyy korkeintaan yhteen henkilöön
- Tämä kuvataan kytkentärajoitteina (engl. *multiplicity*)
  - Alla yhteyden oikeassa päässä on numero 1, joka tarkoittaa, että yhteen Henkilö-olioon liittyy täsmälleen yksi Auto-olio
  - Yhteyden vasemmassa päässä 0..1, joka tarkoittaa, että yhteen Auto-olioon liittyy 0 tai 1 Henkilö-olioa
- Auton *rooli* yhteydessä on olla henkilön omaAuto, rooli on merkitty Auton viereen
  - Huom: roolin nimi on sama kun luokan Henkilö oliomuuttuja, jonka tyyppinä Auto
- Henkilön rooli yhteydessä on olla omistaja



Olioiden väliset yhteydet

Kytkentärajoitteet ja roolit

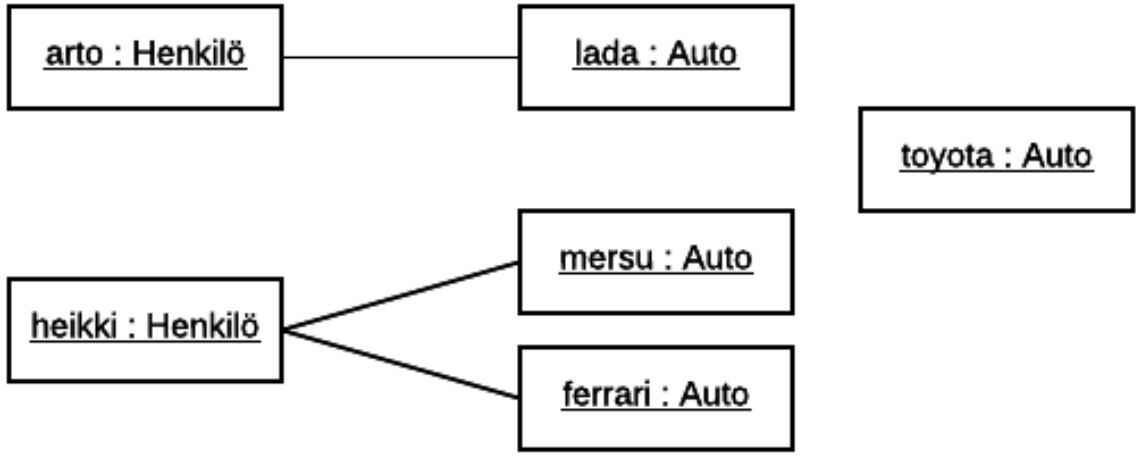
- Edellisessä esimerkissä yhdellä Henkilö-oliolla on yhteys täsmälleen yhteen Auto-olioon
  - Eli yhteyden Auto-päässä on kytkentärajoitteena 1
- Jos halutaan mallintaa tilanne, jossa kullakin Henkilö-oliolla voi olla mielivaltainen määrä autoja (nolla tai useampi), niin kytkentärajoitteeksi merkitään $*$

0	Ei yhtään (harvinainen!)
0..1	Ei yhtään tai yksi
1	Tasan yksi
0..*	Ei yhtään tai enemmän
*	Sama kuin 0..*
1..*	Yksi tai enemmän

Olioiden väliset yhteydet

Yhteydet oliokaaviossa

- Luokkakaavio kuvaa luokkien olioiden kaikkia mahdollisia suhteita
  - Edellisessä sivulla sanotaan vaan, että tietyllä henkilöllä voi olla useita autoja ja tietyllä autolla on ehkä omistaja
- Jos halutaan ilmaista asioiden tila jollain ajanhetkellä, käytetään oliokaaviota
  - Mitä olioita on tietyllä hetkellä olemassa ja miten ne yhdistyvät?
- Alla tilanne, jossa Artolla on 1 auto ja Heikillä 3 autoa, yhdellä autolla ei ole omistajaa



Olioiden väliset yhteydet

Yhden suhde moneen -yhteyden toteuttaminen Javassa

- Jos henkilöllä on korkeintaan yksi auto, on Henkilö-luokalla siis attribuutti, jonka tyyppi on Auto
  - `private Auto omaAuto;`
- Jos henkilöllä on monta autoa, on Javassa yleinen ratkaisu lisätä Henkilö-luokalle attribuutiksi listallinen (esim. ArrayList) autoja:
  - `private ArrayList<Auto> omatAutot;`
- ArrayLististä tarkemmin Ohjelmoinnin perusteiden materiaalissa

Olioiden väliset yhteydet

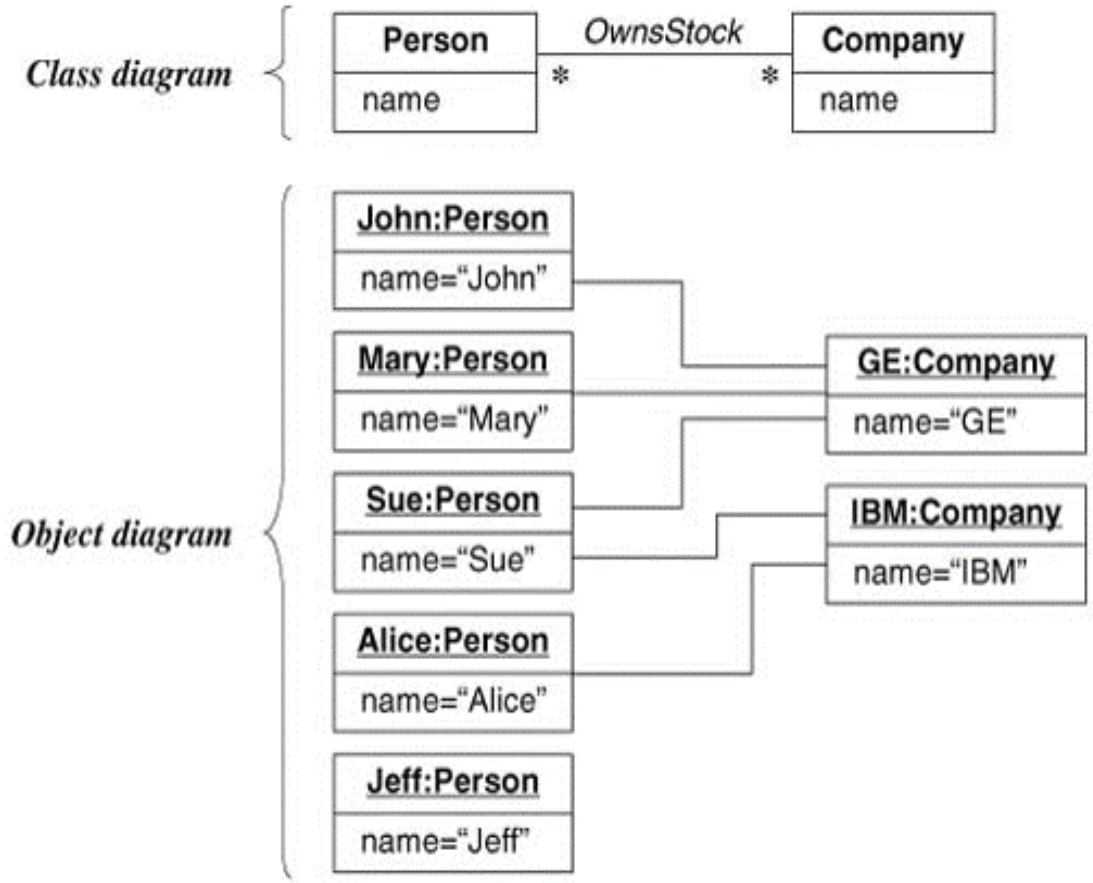
Helppoja yhteysesimerkkejä

- Kuten niin moni asia UML:ssä, on myös yhteyden nimen ja roolinimien merkintä vapaaehtoista
- Jos kytkentärajoite jätetään merkitsemättä, niin silloin yhteydessä olevien olioiden lukumäärä on määrittelemätön
- **Seuraavaksi joukko esimerkkejä** $\longrightarrow$

Olioiden väliset yhteydet

Helppoja yhteysesimerkkejä

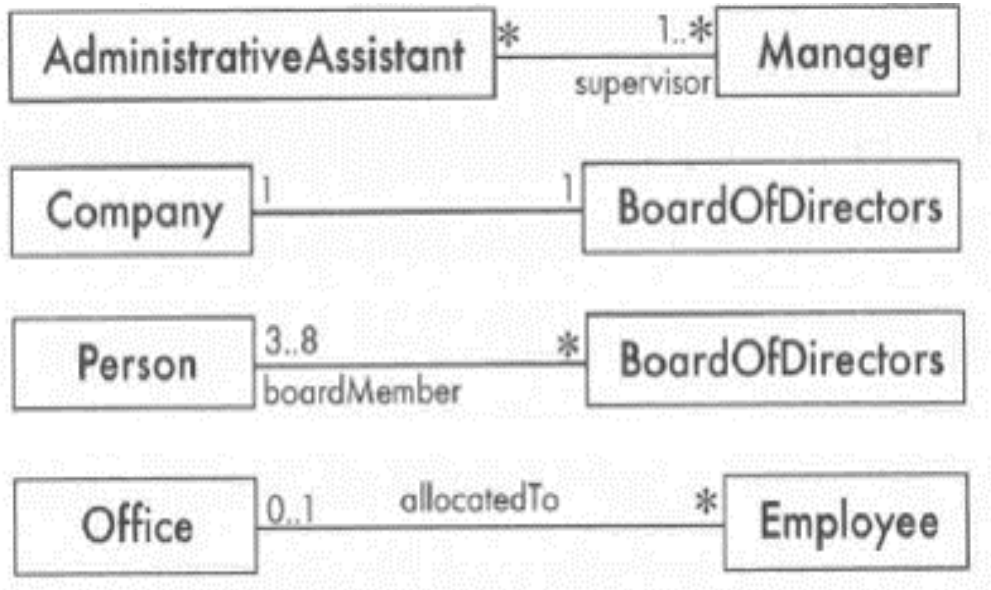
- Henkilö voi omistaa usean yhtiön osakkeita
- Yhtiöllä on monia osakkeenomistajia
- Eli yhteen Henkilö-olioon voi liittyä monta Yhtiö-olioa
- Ja yhteen Yhtiö-olioon voi liittyä monta Henkilö-olioa



Olioiden väliset yhteydet

Helppoja yhteysesimerkkejä

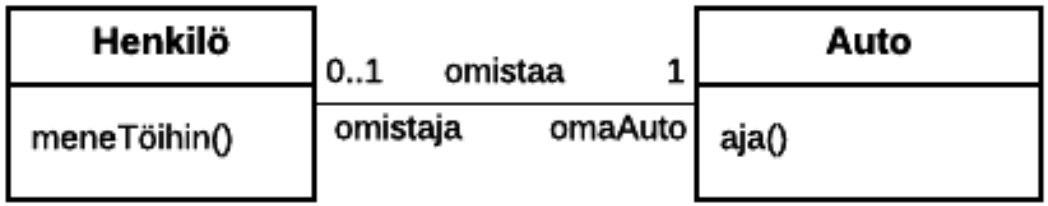
- Manageria kohti on useita assistentteja, assistentin johtajana (supervisor) toimii vähintään yksi manageri
- Yhtiöllä on yksi johtokunta, joka johtaa tasan yhtä yhtiötä
- Johtokuntaan kuuluu kolmesta kahdeksaan henkeä. Yksi henkilö voi kuulua useisiin johtokuntiin, muttei välttämättä yhteenkään.
- Toimistoon on sijoitettu (allocatedTo) useita työntekijöitä. Työntekijällä on paikka yhdessä toimisto tai ei missään



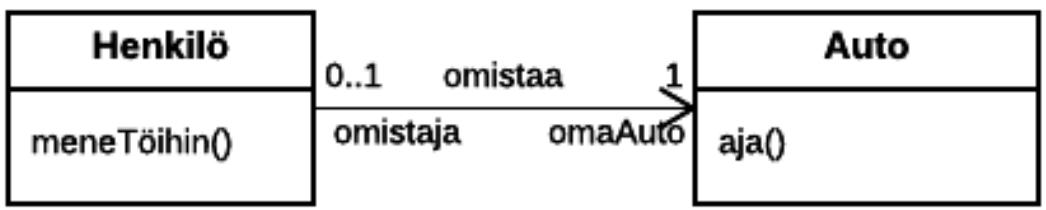
Olioiden väliset yhteydet

Yhteyden navigointisuunta

- **Palautetaan mieleen Auto ja Henkilö -esimerkki**[4]



- Auto-luokan koodista huomaamme, että auto-oliot eivät tunne omistajaansa
  - Henkilö-oliot taas tuntevat omistamansa autot Auto-tyyppisen attribuutin omaAuto ansiosta
- Yhteys siis on oikeastaan yksisuuntainen, henkilöstä autoon
- Asia voidaan ilmaista kaaviossa tekemällä yhteysviivasta nuoli
  - Nuolen kärki sinne suuntaan, johon on pääsy oliomuuttujan avulla (nyrkkisääntö!)



[4] Katso *Kytkentärajoitteet ja roolit* kalvolta numero 91.

Olioiden väliset yhteydet

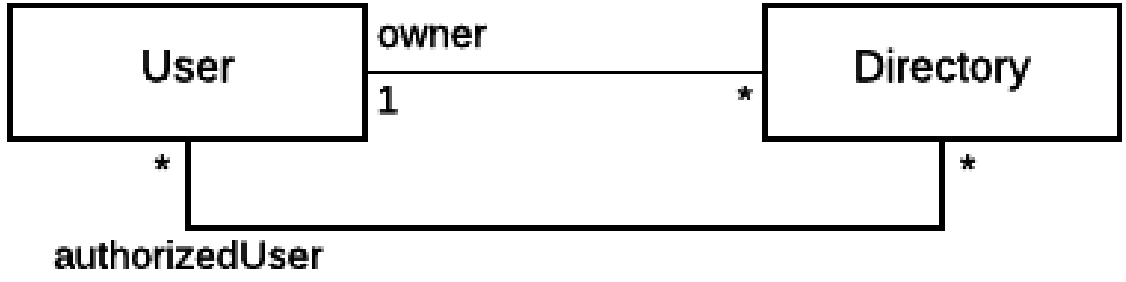
Yhteyden navigointisuunta

- Yhteyden navigointisuunnalla merkitystä lähinnä suunnittelu- ja toteutustason kaavioissa
  - Merkitään vain jos suunta tärkeä tietää
  - Joskus kaksisuuntaisuus merkitään nuolella molempiin suuntiin
  - Joskus taas nuoleton tarkoittaa kaksisuuntaista
- Määrittelytason luokkakaavioissa yhteyden suuntia ei yleensä merkitä ollenkaan
- Yhteyden suunnalla on aika suuri merkitys sille, kuinka yhteys toteutetaan kooditasolla

Olioiden väliset yhteydet

Useampia yhteyksiä olioiden välillä

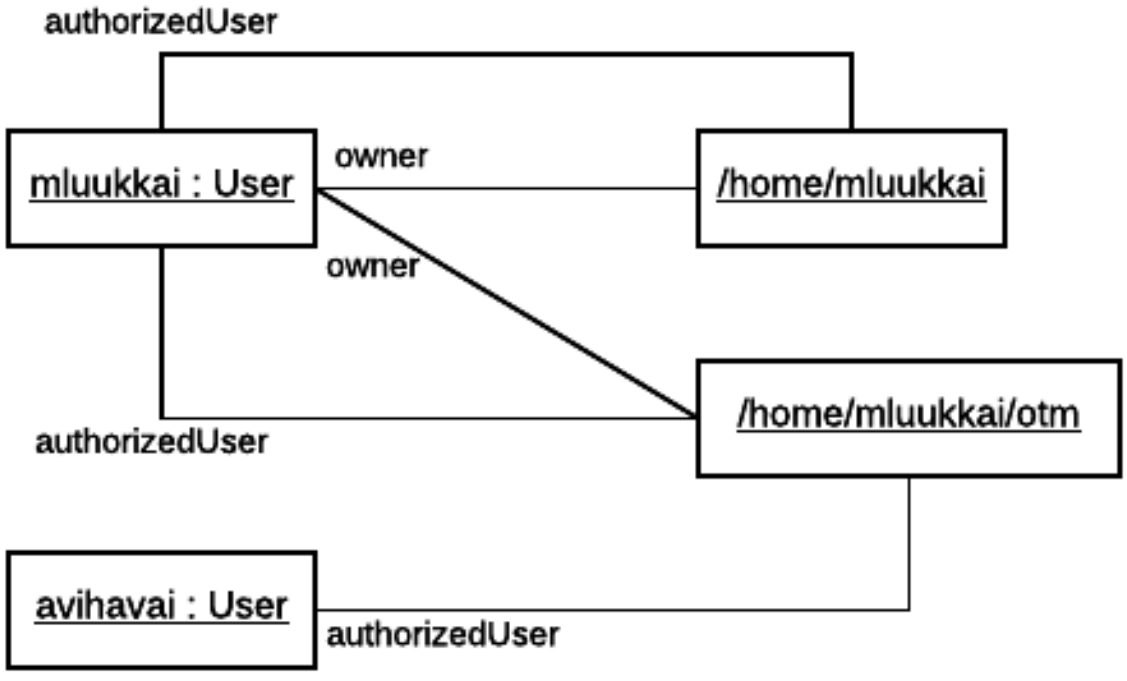
- Esim. Linuxissa jokaisella hakemistolla on tasan yksi omistaja
- Eli yhteen hakemisto-olioon liittyy roolissa owner tasan yksi käyttäjä-olio
- Jokaisella hakemistolla voi olla lisäksi useita käyttäjiä
  - Yhteen hakemistoon liittyy useita käyttäjiä roolissa authorizedUser
- Yksi käyttäjä voi omistaa useita hakemistoja
- Yhdellä käyttäjällä voi olla käyttöoikeus useisiin hakemistoihin
- Yhdellä käyttäjällä voi olla samaan hakemistoon sekä omistus- että käyttöoikeus



Olioiden väliset yhteydet

Useampia yhteyksiä olioiden välillä

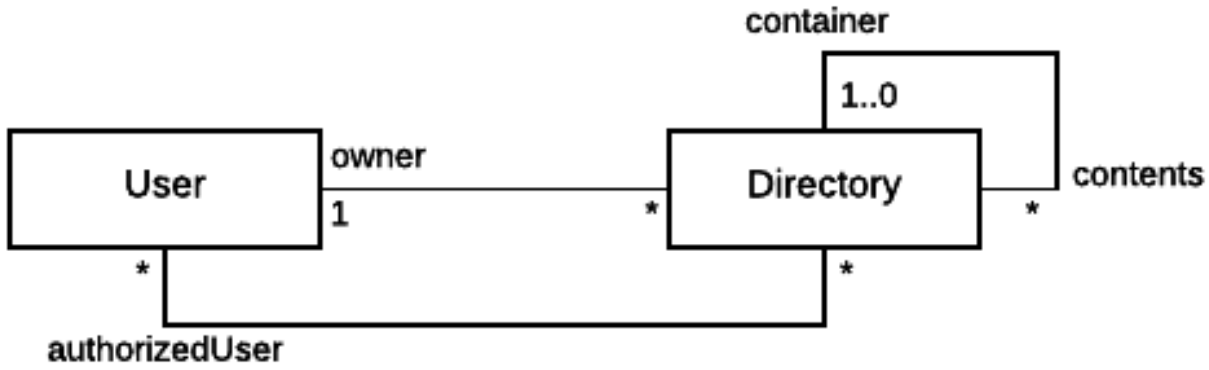
- Ohessa oliokaavio, joka kuvaa erään edellisen luokkakaavion mukaisen tilanteen
  - Käyttäjät: mluukkai ja avihavai
  - mluukkai omistaa kaksi hakemistoa
  - mluukkai omistaa kaksi hakemistoa
    - Samojen olioiden välillä kaksi eri yhteyttä!
  - avihavai:lla käyttöoikeus hakemistoon /home/mluukkai/otm



Olioiden väliset yhteydet

Yhteys kahden saman luokan olion välillä

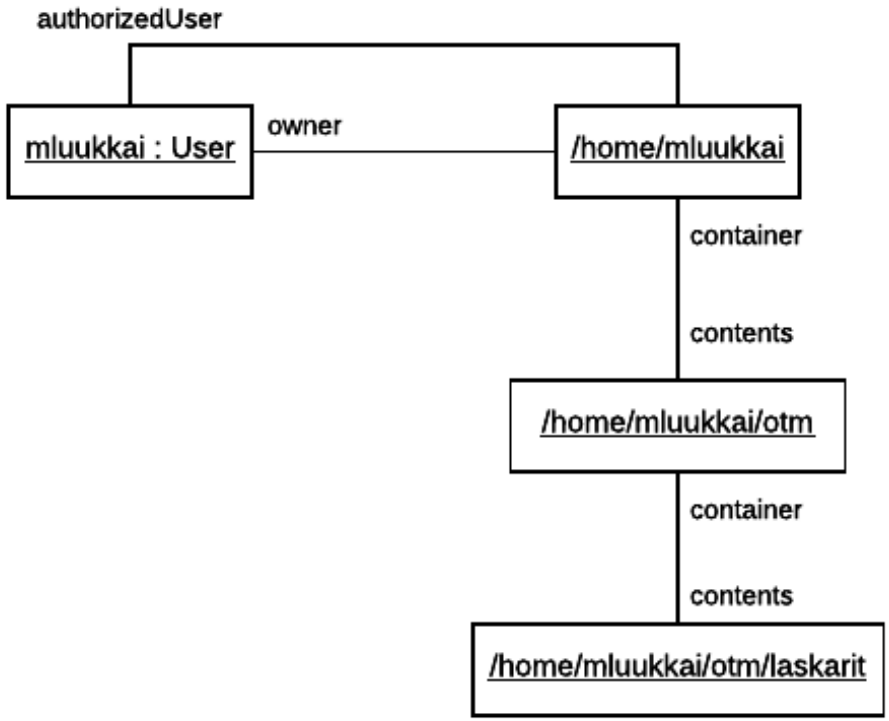
- Miten mallinnetaan se, että hakemistolla on alihakemistoja?
  - Hakemisto sisältää alihakemistoja
  - Hakemisto sisältyy johonkin toiseen hakemistoon
- Yhteen hakemisto-olioon voi liittyä 0 tai 1 hakemisto-olioa roolissa *container* (=sisältäjä), eli hakemisto voi olla jonkun toisen hakemiston alla
- Yhteen hakemisto-olioon voi liittyä mielivaltainen määrä (*) hakemisto-olioita roolissa *contents* (=sisältö), eli hakemisto voi sisältää muita hakemistoja



Olioiden väliset yhteydet

Yhteys kahden saman luokan olion välillä

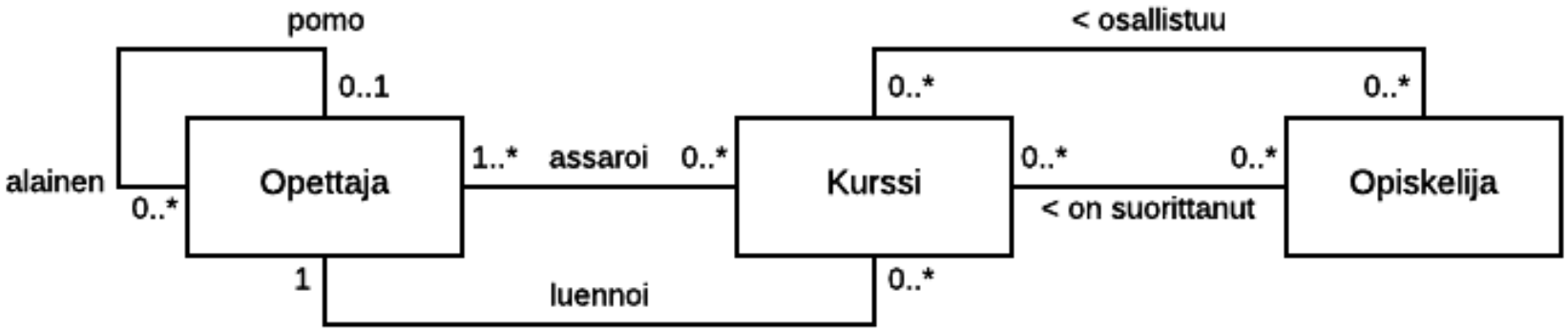
- Tilanne vaikuttaa sekavalta, selvennetään oliokaavion avulla
- /home/mluukkai hakemiston
  - Alihakemistojen rooli yhteydessä on *contents* eli sisältö
  - Päähakemiston rooli yhteydessä on *container* eli sisältäjä
- /home/mluukkai/otm on edellisen alihakemisto, mutta sisältää itse alihakemiston



Olioiden väliset yhteydet

Monimutkaisempi esimerkki assosiaatioista

- **Mallinnetaan seuraava tilanne:**
  - Kurssilla on luennoijana 1 opettaja ja assarina useita opettajia
  - Opettaja voi olla useiden kurssien assarina ja luennoijana
  - Opettajalla voi olla pomo ja useita alaisia
  - Opettajat johtavat toisiaan
  - Opiskelija voi osallistua useille kursseille
  - Opiskelijalla voi olla suorituksia useista kursseista



Olioiden väliset yhteydet

Riippuvuus

Olioiden väliset yhteydet

Riippuvuus

- **Muistellaan taas Auto-esimerkkiä:**
- On olemassa henkilöitä, jotka eivät omista autoa
- Autottomat henkilöt kuitenkin välillä lainaavat jonkun muun autoa
- **Koodissa asia voitaisiin ilmaista seuraavasti:**

```
public class AutotonHenkilo {
    public void lainaaJaAja( Auto lainaAuto ) {
        lainaAuto.aja();
    }
}
```

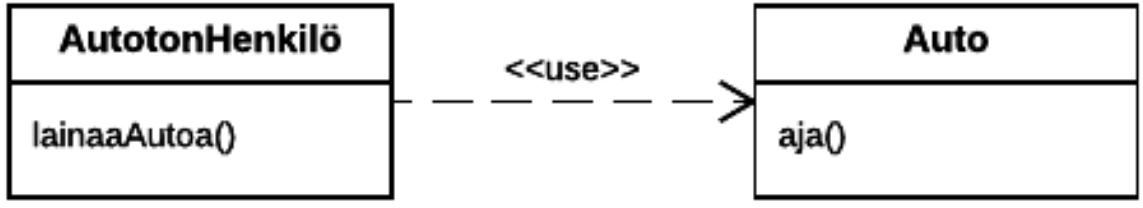
- Eli autottoman henkilön metodi `lainaaJaAja` saa parametrikseen auton (lainaAuto), jolla henkilö ajaa
- Auto on lainassa ainoastaan metodin suoritusajan
  - AutotonHenkilo ei siis omaa pysyvää suhdetta autoon

Olioiden väliset yhteydet

Riippuvuus

Riippuvuus on tavallaan myös yhteys, mutta "normaalia"yhteyttä "heikompi"(koska se ei kestä yhtä kauaa)

- Koska kyseessä ei ole pysyvämpiluontoinen yhteys, on parempi käyttää luokkakaaviossa riippuvuussuhdetta (engl. dependency)
- Riippuvuus merkitään katkoviivanuolena, joka osoittaa siihen luokkaan josta ollaan riippuvaisia
- Alla on ilmaistu vielä riippuvuuden laatu
  - Tarkennin (eli *stereotyyppi*) «use» kertoo että kyseessä on käyttöriippuvuus, eli `AutotonHenkilö` kutsuu `Auto`:n metodia



Olioiden väliset yhteydet

Riippuvuus

- Joskus riippuvuus määritellään siten, että luokka A on riippuvainen luokasta B jos muutos B:hen saa aikaan mahdollisesti muutostarpeen A:ssa
  - Näin on edellisessä esimerkissä: jos Auto-luokka muuttuu (esim. metodi aja muuttuu siten että se tarvitsee parametrin), joudutaan AutotonHenkilö-luokkaa muuttamaan
- Riippuvuus on siis jotain *heikompaa* kun tavallinen luokkien välinen yhteys
  - Jos luokkien välillä on yhteys, on niiden välillä myös riippuvuus, sitä ei vaan ole tapana merkitä (Henkilö joka omistaa Auton on riippuvainen autosta)

Huomaathan, että toisin kuin yhteyksiin, **riippuvuuksiin ei merkitä kytkentärajoitteita**

Olioiden väliset yhteydet

Kompositio

Olioiden väliset yhteydet

Kompositio

- Autossa on 4 pyörää, joten tilannehan voitaisiin mallintaa tekemällä autosta yhteys Rengas-olioon ja laittamalla kytkentärajoitteeksi 4
- Renkaat ovat kuitenkin siinä mielessä erityisessä asemassa, että voidaan ajatella, että ne ovat auton komponentteja
  - Renkaat sisältyvät autoon
- Kun auto luodaan, luodaan renkaat samalla
  - Koodissa auto luo renkaat
- Renkaat ovat private, eli niihin ei pääse ulkopuolelta käsiksi
- Kun roskienkerääjä tuhoaa auton, tuhoutuvat myös renkaat
- **Eli ohjelman renkaat sisältyvät autoon ja niiden elinikä on sidottu auton elinikään** (oikeat renkaat eivät tietenkään käyttäydy näin vaan ovat vaihdettavissa)

Olioiden väliset yhteydet

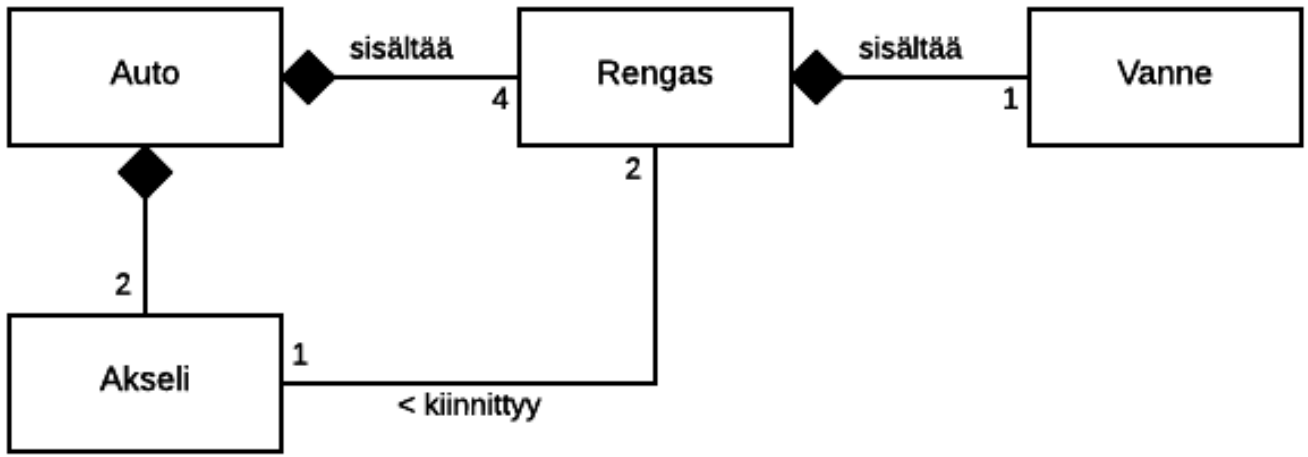
Kompositio

- Edellisen dian tilannetta nimitetään **kompositioksi** (engl. *composition*)
- Komposition symboli on ”musta salmiakkikuvio”, joka liitetään yhteyden siihen päähän, johon osat sisältyvät (nyrkkisääntö!)
- Kompositiota käytetään kun seuraavat ehdot toteutuvat:
  - Osat ovat olemassaoloriippuvaisia kokonaisuudesta
  - Osa voi kuulua vaan yhteen kompositioon
    - Rengasta ei voi siirtää toiseen autoon
  - Osa on koko elinaikansa kytketty samaan kompositioon
- Koska Rengas-olio voi liittyä nyt vain yhteen Auto-olioon, ei salmiakin puoleiseen päähän tarvita osallistumisrajoitetta koska se on joka tapauksessa 1

Olioiden väliset yhteydet

Monimutkaisempi esimerkki kompositiosta

- Tarkennettu Auto sisältää 4 rengasta ja 2 akselia
- Komposition osa voi myös sisältää oliota
  - Rengas sisältää vanteen
- Komposition osilla voi olla ”normaaleja” yhteyksiä
  - Akseli kiinnittyy kahteen renkaaseen
  - Rengas on kiinnittynyt yhteen akseliin



Olioiden väliset yhteydet

Monimutkaisempi esimerkki kompositiosta

- Onko kompositiomerkkiä pakko käyttää?
  - Ei, mutta usein sen käyttö selkiyttää tilannetta
- Kompositiota ei kannata laittaa sinne minne se ei kuulu!
  - Kompositio on erittäin rajoittava suhde olioiden välillä, toisin kuin ”normaali” yhteys, käytä kompositiota vasta kun olet tarkistanut onko kaikki sen ehdot täyttyneet
- **HUOM:** auton ja renkaat sisältävä esimerkki kuvaa vaan esimerkkikoodi tilannetta mutta ei tietenkään ole realistinen kuvaamaan reaalimaailman autojen ja renkaiden suhdetta sillä normaalistihan renkaat eivät ole autoista olemassaoloriippuvaisia

Olioiden väliset yhteydet

Kooste

Olioiden väliset yhteydet

Kooste

- **Koosteella** (engl. aggregation) tarkoitetaan koostumussuhdetta, joka ei ole yhtä komposition tapaan ”ikuinen”
- Koosteella (engl. agregation) tarkoitetaan koostumussuhdetta, joka ei ole yhtä komposition tapaan ”ikuinen”
  - **HUOM:** suomenkieliset termit kooste ja kompositio ovat huonot ja jopa harhaanjohtavat
- Koostetta merkitään ”valkoisella salmiakilla” joka tulee siihen päähän yhteyttä, johon osat kuuluvat
- Esimerkki: Joukkue koostuu pelaajista (jotka ovat ihmisiä)
  - Ihminen ei kuitenkaan kuulu joukkueeseen ikuisesti
  - Joukkue ei syynnytä eikä tapa pelaajaa
  - Ihminen voi kuulua yhtäaikaa useampaan joukkueeseen

Olioiden väliset yhteydet

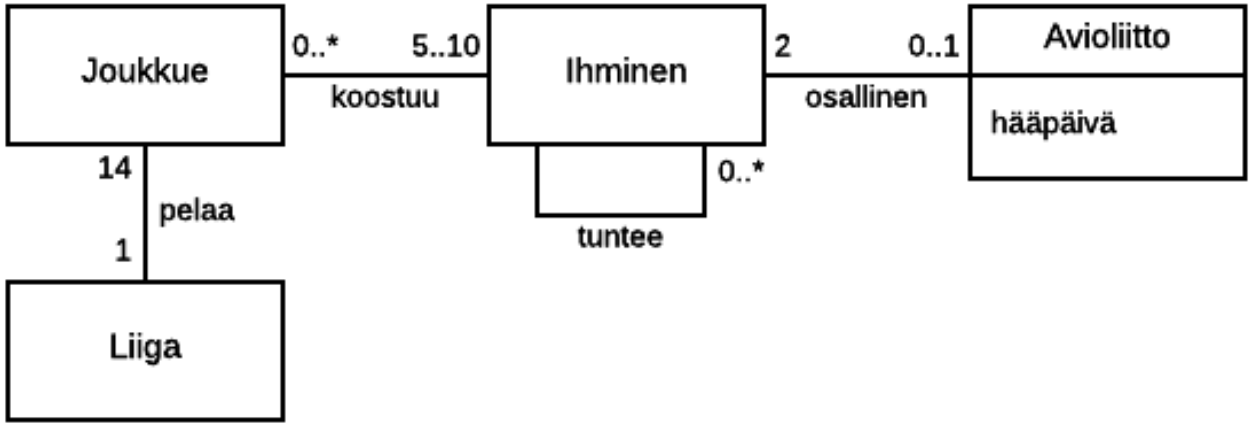
Kooste

- Komposition (eli mustan salmiakin) merkitys on selkeä, kyseessä on olemassaoloriippuvuus
- On sensijaan epäselvää millon koostetta (eli valkoista salmiakkia) tulisi käyttää normaalin yhteyden sijaan
- Monet asiantuntevat oliomallintajat ovat sitä mieltä että koostetta ei edes tulisi käyttää
- Koostesuhde on poistunut UML:n standardista versiosta 2.0 lähtien
- Koostesuhde on kuitenkin edelleen erittäin paljon käytetty joten on hyvä tuntea symbooli passiivisesti
- **Tällä kurssilla koostetta ei käytetä eikä sitä tarvitse osata!**
- Joukkueen ja pelaajien välinen suhde voidaankin ilmaista normaalina yhteytenä

Olioiden väliset yhteydet

Monimutkaisempi esimerkki Koosteen kierrosta

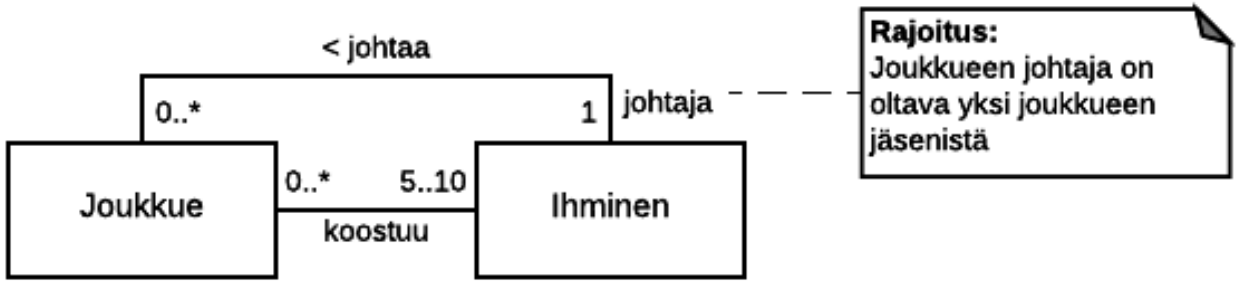
- Joukkue pelaa liigassa, jossa on 14 joukkuetta
- Ihminen voi kuulua mielivaltaisen moneen joukkueeseen
- Joukkueeseen kuuluu 5-10 ihmistä
- Ihminen tuntee useita ihmisiä
- Ihminen voi olla avioliitossa, mutta vain yhdessä avioliitossa kerrallaan
- Avioliitto koostuu kahdesta ihmisestä



Olioiden väliset yhteydet

Rajoitukset

- Jos haluttaisiin mallintaa tilanne, että joku joukkueen jäsenistä on joukkueen johtaja, pelkkä luokkakaavio (siinä määrin kun tällä kurssilla UML:ää opitaan) ei riitä
- Tilanne voitaisiin mallintaa alla esitetyllä tavalla
  - Eli lisätään normaali yhteys *johtaa* joukkueen ja ihmisen välille
  - Määritellään kytkentärajoite: joukkueella on tasan 1 johtaja
  - Ilmaistaan UML-kommenttina, että joukkueen johtajan on oltava joku joukkueen jäsenistä



Olioiden väliset yhteydet

Yhteydessä olevien luokkien järjestys

- Esim. edellisessä esimerkissä joukkue koostuu pelaajista
  - Pelaajilla ei ole kuitenkaan mitään erityistä järjestystä yhteyden kannalta
- Joskus osien järjestys voi olla tärkeä
- Esim. dokumentti sisältää sivuja ja kukin sivu sisältää tekstirivejä
  - Sivujen ja rivien on oltava tietyssä järjestyksessä, muuten dokumentissa ei ole järkeä
- Se seikka, että osat ovat järjestyksessä voidaan ilmaista lisämääreenä {ordered}, joka laitetaan niiden olioiden päähän yhteyttä joilla on järjestys

Työkaluja piirtoon

Ohjelmia ja nettisivuja

Työkaluja piirtoon

Ohjelmia ja nettisivuja

Millä kaaviot kannattaa piirtää?

- Kynä ja paperia tai valkotaulu
- Ilmaisia työkaluja
  - Dia (win+linux)
  - Umbrello
  - ArgoUML
  - Openoffice
- Maksullisia työkaluja:
  - Visual Paradigm
  - Magic Draw
  - Microsoft Visio
  - Omnigraffle (Mac)

Työkaluja piirtoon

Ohjelmia ja nettisivuja

Millä kaaviot kannattaa piirtää?

- Mahdollisuuksia myös netissä:
  - **http://yuml.me/** luokka- ja käyttötapauskaavioihin
  - **https://www.websequencediagrams.com/** Sekvenssikaavioihin
  - **https://www.draw.io/** Jokapaikan höylä

Ei siis ole olemassa selkeää vastausta mitä työkalua kannattaa käyttää. Tämän kurssin tarpeisiin kynä ja paperia riittää hyvin

Ohjelmistotekniikan menetelemät
Luento 3

Määrittelyvaiheen luokkakaavio, sekvenssikaavio ja kommunikaatiokaavio

Määrittelyvaiheen luokkakaavion laatiminen

Alustavan luokkamallin muodostaminen

Määrittelyvaiheen luokkakaavion laatiminen

Alustavan luokkamallin muodostaminen

- 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
  - Tunnistetaan sovellusalueen käsitteet ja niiden väliset suhteet
  - Eli karkeasti ottaen tehtävänä on etsiä todellisuutta simuloiva luokkarakenne
- Menetelmästä käytetään nimitystä **käsiteanalyysi** (engl. *conceptual modeling*)
  - Järjestelmän sovellusalueen käsitteistöä kuvaavaa luokkamallia kutsutaan **kohdealueen luokkamalliksi** (engl. problem domain model)

Määrittelyvaiheen luokkakaavion laatiminen

Alustavan luokkamallin muodostaminen

- Menetelmän voi ajatella etenevän seuraavien vaiheiden kautta
  1. Kartoita luokkaehdokkaat
  2. Karsi luokkaehdokkaita
  3. Tunnista olioiden väliset yhteydet
  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 harvinaista, että ensimiettimältä päädytään *lopulliseen* ratkaisuun

Määrittelyvaiheen luokkakaavion laatiminen

Luokkaehdokkaiden kartoitus

- **(1)** 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**

Määrittelyvaiheen luokkakaavion laatiminen

Luokkaehdokkaiden kartoitus

Etsitään käytettävissä olevista kuvauksista kaikki substantiivit ja otetaan ne alustaviksi luokkaehdokkaiksi:

- 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.

Määrittelyvaiheen luokkakaavion laatiminen

Luokkaehdokkaiden kartoitus

Löydettiin seuraavat substantiivit:

- elokuvalippu
- varaaminen
- lippu
- paikka
- näytös
- elokuva
- esittäminen
- teatteri
- aika
- asiakas
- varaus

Määrittelyvaiheen luokkakaavion laatiminen

Ehdokkaiden karsiminen

- **Näin pitkällä listalla on varmasti ylimääräisiä luokkakandidaatteja**
- **(2)** Karsitaan sellaiset jotka eivät vaikuta potentiaalisilta luokilta
- Kysymyksiä, joita karsiessa voi miettiä
  - Onko käsitteellä merkitystä järjestelmän kannalta
  - Onko kyseessä jonkin muun termin synonyymi
  - Onko kyseessä jonkin toisen käsitteen attribuutti
  - Liittyykö käsitteeseen tietosisältöä?
  - On tosin olemassa myös tietosisällöttömiä luokkia...
  - Onko käsitteeseen liittyvä tieto sellaista, että järjestelmän on *muistettava* tieto pitkiä aikoja
- Huomattavaa on, että kaikki luokat eivät yleensä edes esiinny sanallisissa kuvauksissa, vaan ne löytyvät vasta jossain myöhemmässä vaiheessa

Määrittelyvaiheen luokkakaavion laatiminen

Ehdokkaiden karsiminen

Löydettiin seuraavat substantiivit:

- ~~elokuvalippu~~ ← Termin lippu synonyymi
- ~~varaaminen~~ ← Tekemistä
- lippu
- paikka
- näytös
- elokuva
- ~~esittaminen~~ ← Tekemistä
- teatteri
- ~~aika~~ ← Attribuutti (näytöksen)
- asiakas
- varaus

Määrittelyvaiheen luokkakaavion laatiminen

Alustava yhteyksien tunnistaminen

- Kun luokat on alustavasti tunnistettu, kannattaa ottaa paperia ja kynä ja piirtää alustava luokkakaavio, joka koostuu vasta luokkalaatikoista
- **(3)** 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

Määrittelyvaiheen luokkakaavion laatiminen

Alustava yhteyksien tunnistaminen

Huom: Kaikki verbit eivät ole yhteyksiä! Yhteydellä tarkoitetaan pysyvää suhdetta, usein verbit ilmentävät ohimeneviä asioita

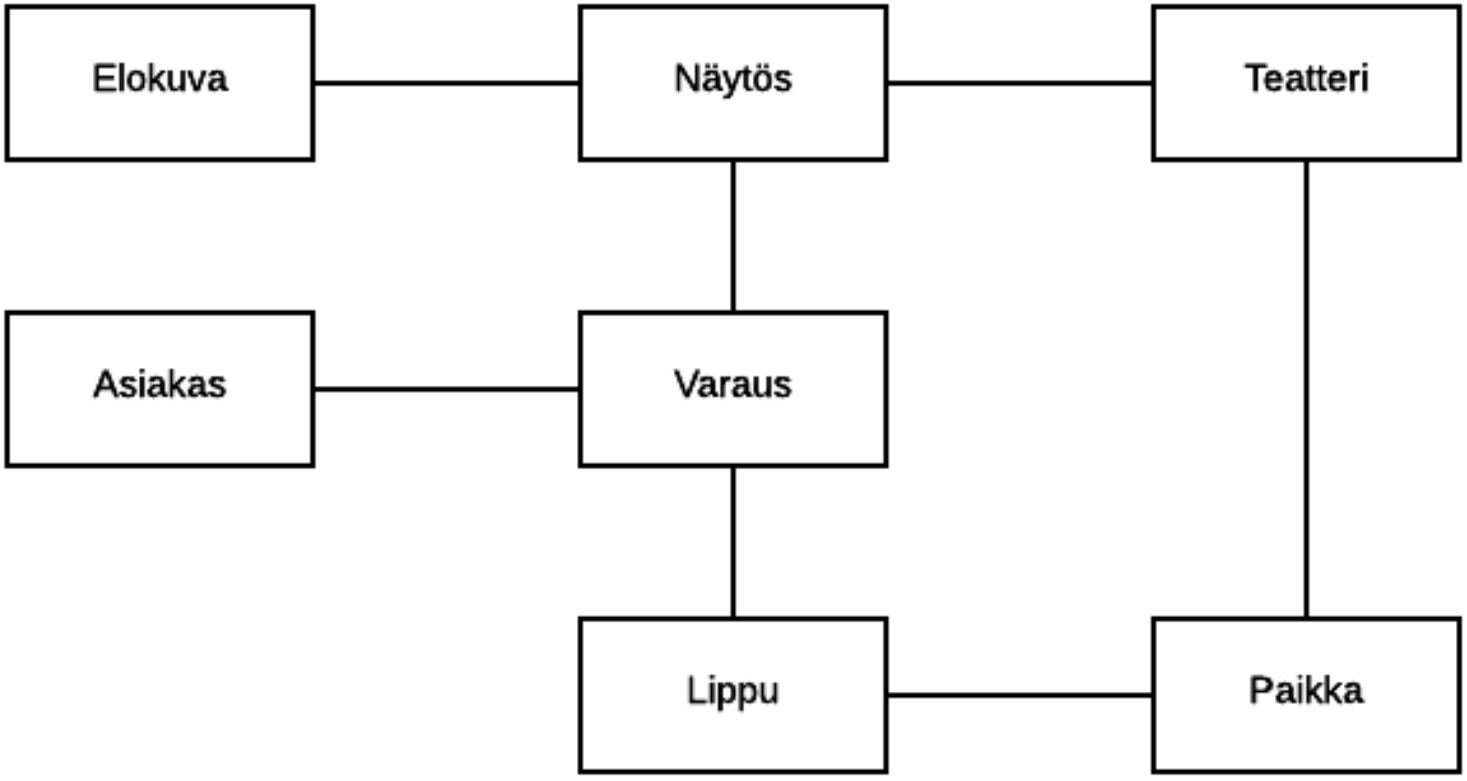
Löydettiin seuraavat yhteydet:

- Lippu – paikka
- Näytös – elokuva
- Näytös – teatteri
- Elokuva – teatteri
- Asiakas – varaus
- Asiakas – lippu
- Lippu – varaus

Määrittelyvaiheen luokkakaavion laatiminen

Alustava yhteyksien tunnistaminen

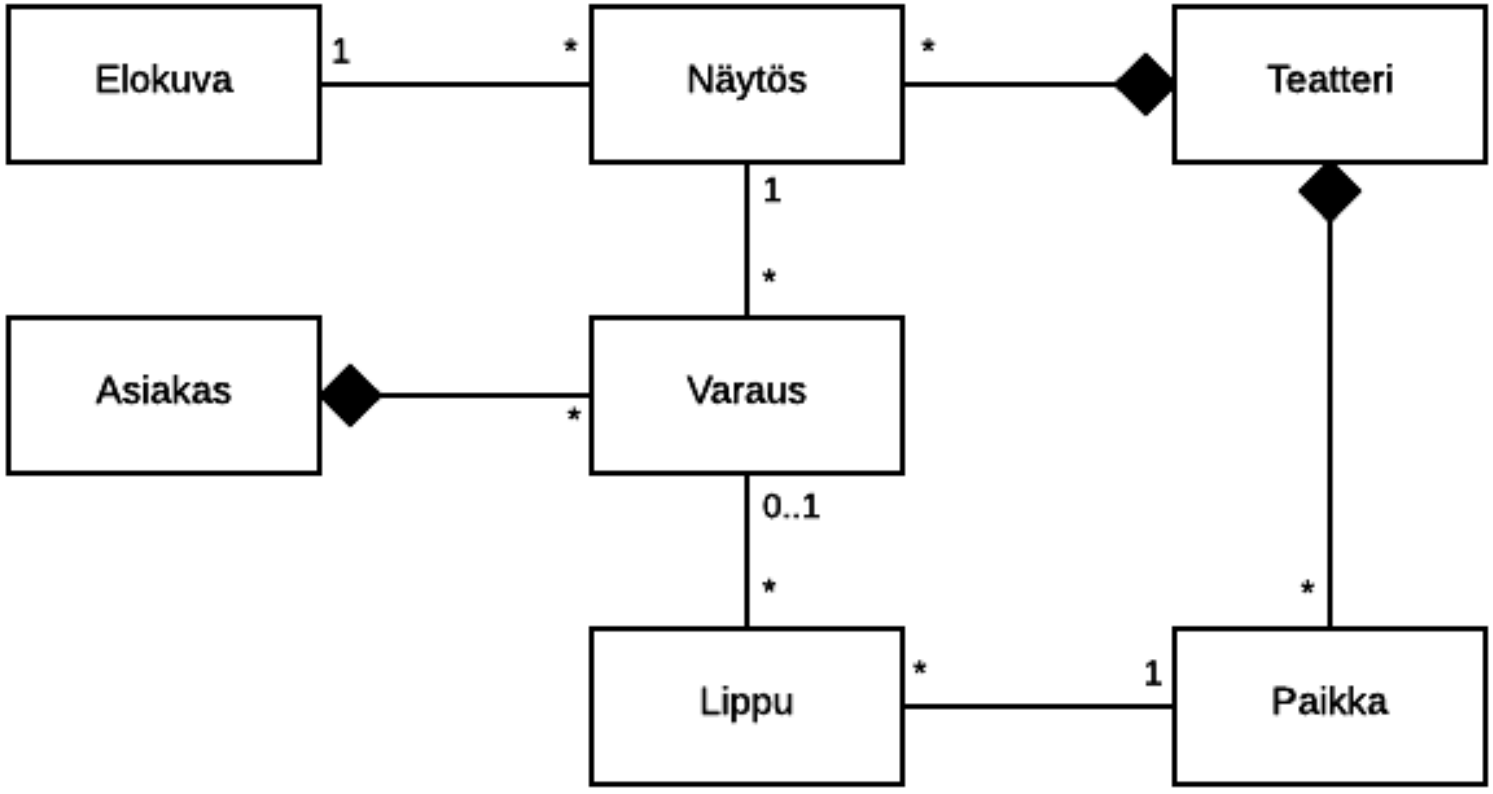
- **Seuraavaksi jonkinlainen hahmotelma.** Osa yhteyksistä siis edellisen listan ulkopuolelta, ”maalaisjärellä” pääteltyjä:



Määrittelyvaiheen luokkakaavion laatiminen

Alustava yhteyksien tunnistaminen

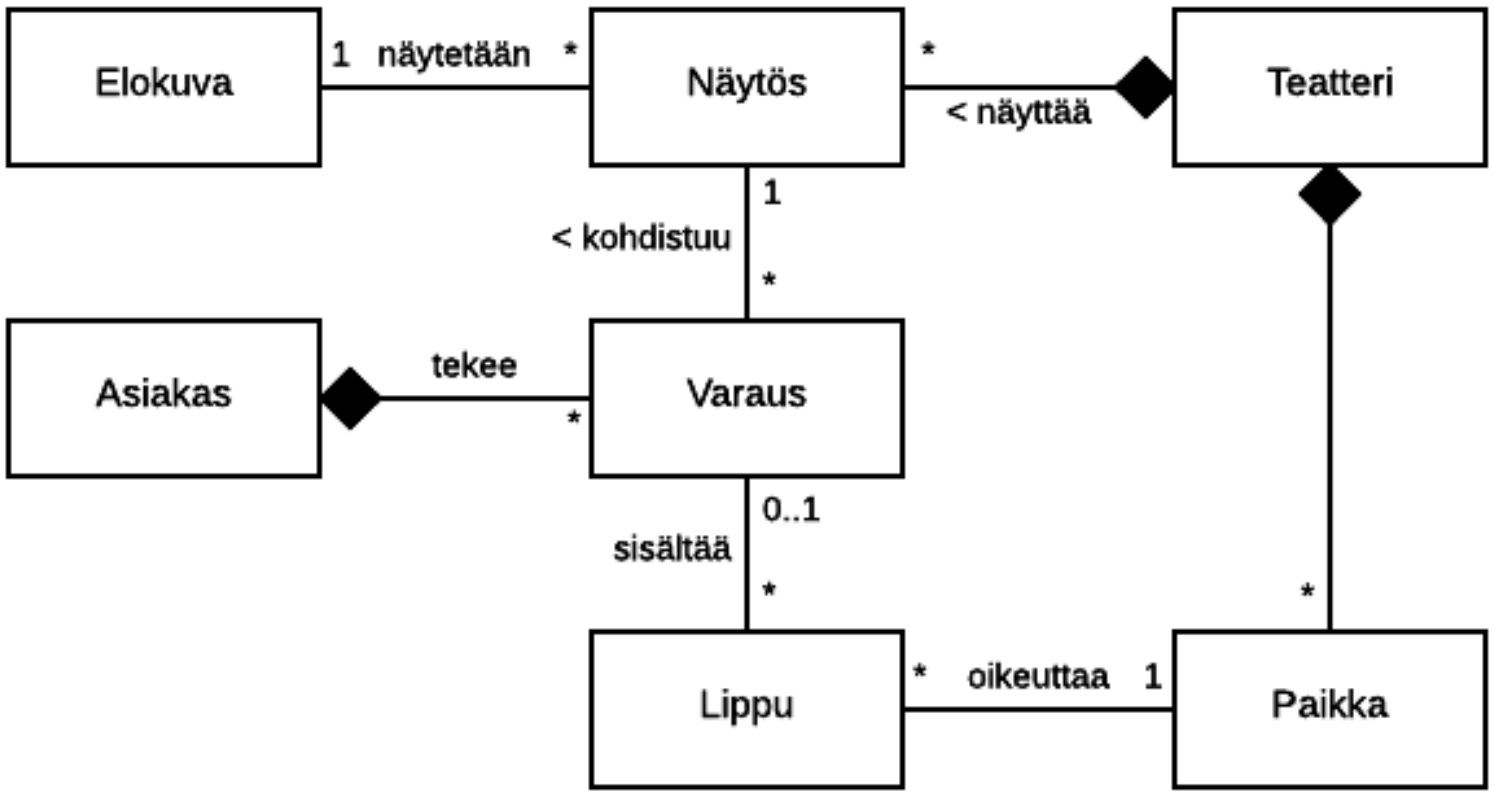
- Kun yhteys tunnistetaan ja vaikuttaa tarpeelliselta, tarkennetaan yhteyden laatua ja kytkentärajoitetta
- Ei ole olemassa oikeaa etenemisstrategiaa...



Määrittelyvaiheen luokkakaavion laatiminen

Yhteyksien tarkentaminen ja attribuuttien etsiminen

- **(4)** Attribuuttien löytäminen edellyttää yleensä lisätietoa, esim. asiakkaan haastatteluista
- Määrittelyvaiheen aikana tehtävää kohdealueen luokkamallia ei ole välttämättä tarkoituksenmukaista tehdä kaikin osin tarkaksi
- **(5)** Tarkennetaan myös yhteyksiä



Määrittelyvaiheen luokkakaavion laatiminen

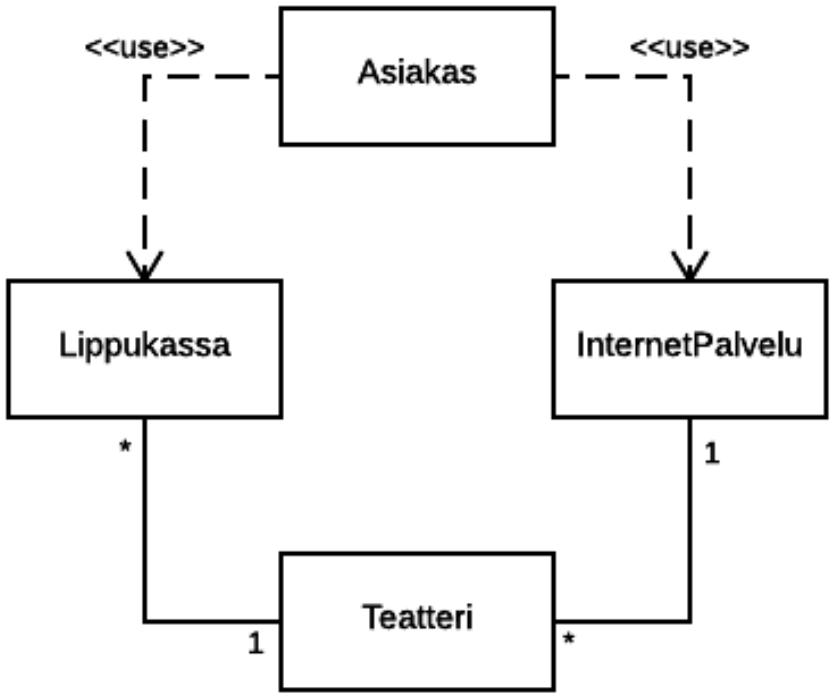
Mikä ei ole yhteys ja mikä ei?

- 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
- Ovatko nämä *rakenteellisia* yhteyksiä, jotka voidaan merkata myös luokkakaavioon?

Määrittelyvaiheen luokkakaavion laatiminen

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
- Jos se, että asiakas käyttää lippukassan palvelua halutaan merkata luokkakaavioon, tulee yhteyden sijasta käyttää riippuvuutta



Määrittelyvaiheen luokkakaavion laatiminen

Mikä ei ole yhteys ja mikä ei?

- Edellisen dian kaltaisessa tilanteessa ei olisi mielekästä merkitä Asiakkaasta riippuvuutta Lippukassaan tai Internetpalveluun
- Luokka Asiakas on järjestelmässä oikean elokuvateatterin asiakkaan representaatio, joka ei kuitenkaan itsessään ”suorita” mitään toimenpiteitä
- 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):

Suurin osa varsinaisen toiminnallisuuden suorittavista olioista tulee vasta suunnittelutason luokkamalleihin.

Määrittelyvaiheen luokkakaavion laatiminen

Yläkäsitteiden etsiminen

- Seuraavaksi täytyy **(7)** etsiä yläkäsitteet
- Tämä liittyy periytymiseen ja palaamme asiaan seuraavalla luennolla
- **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

Määrittelyvaiheen luokkakaavion laatiminen

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

Sekvenssikaavio

Olioiden yhteistyön mallintaminen

Sekvenssikaavio

Olioiden yhteistyön mallintaminen

- Luokkakaaviosta käy hyvin esille ohjelman *rakenne*
- 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”

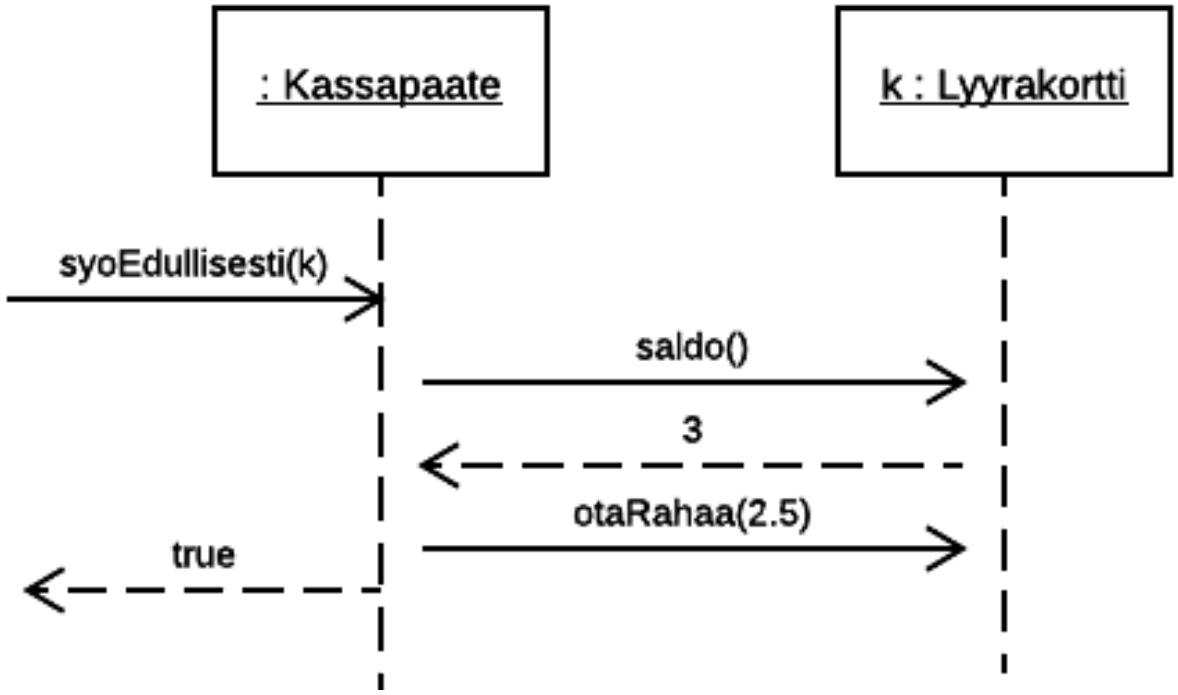
Koska järjestelmän toiminnan kulmakivenä on järjestelmän sisältämien olioiden yhteistyö, tarvitaan menetelmä yhteistyön kuvaamiseen

- UML tarjoaa kaksi menetelmää, joita kohta tarkastelemme:
  - sekvenssikaavio ja kommunikaatiokaavio

Sekvenssikaavio

Olioiden yhteistyön mallintaminen

- "*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:



Sekvenssikaavio

Olioiden yhteistyön mallintaminen

- 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
- Metodikutsu piirretään nuolena, joka lähtee kutsuvasta oliosta ja kohdistuu kutsuttavan olion elämänlankaan
- Tyypillisesti yksi sekvenssikaavio kuvaa järjestelmän yksittäisen toimintaskenaarion

Sekvenssikaavio

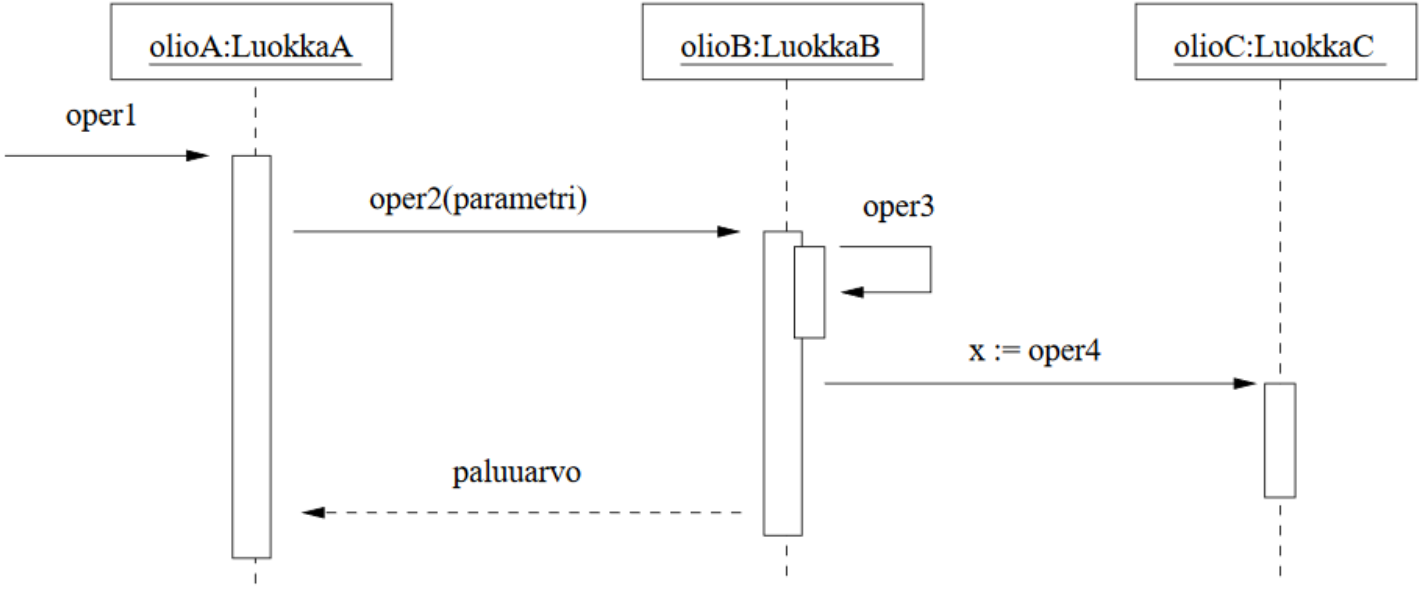
Olioiden yhteistyön mallintaminen

- Esimerkissä toiminta alkaa sillä, että joku (esim. pääohjelma, tässä tapauksessa nuoli on merkitty tulevan tyhjästä) kutsuu Kassapaate-olion metodia `syoEdullisesti`
- 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

Sekvenssikaavio

Olioiden yhteistyön mallintaminen

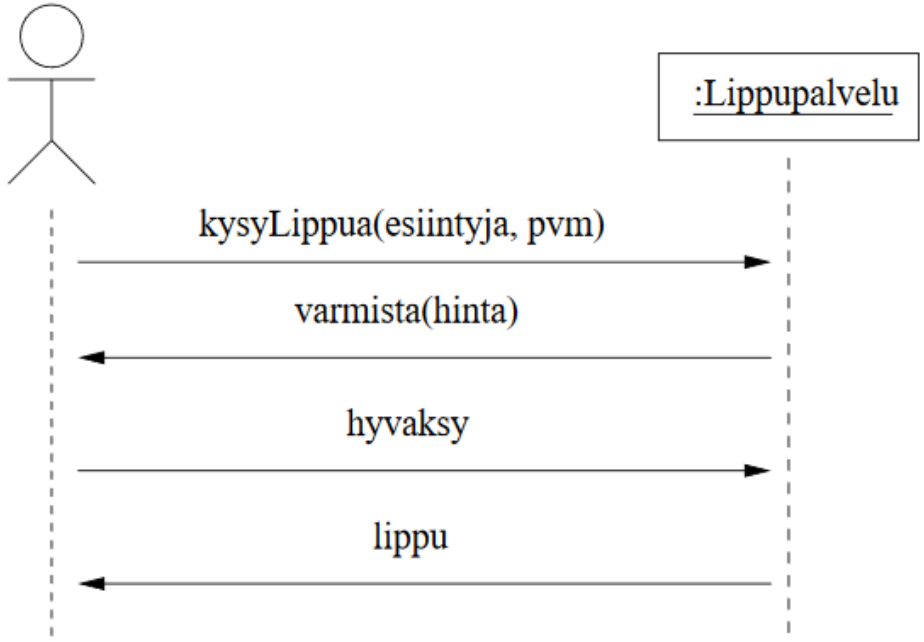
- Joskus on hyödyllistä piirtää **aktivaatiopalkki**, joka merkitsee ajan jolloin olio on aktiivisena, eli sen suoritus on kesken
- Aktivaatiopalkkia harvemmin jaksetaan piirtää paperille
- Merkinnällä `x := oper4` tarkoitetaan, että metodia `oper4()` kutsutaan ja sen palautusarvo "napataan"muuttujaan `x`



Sekvenssikaavio

Lippuvaraus esimerkki

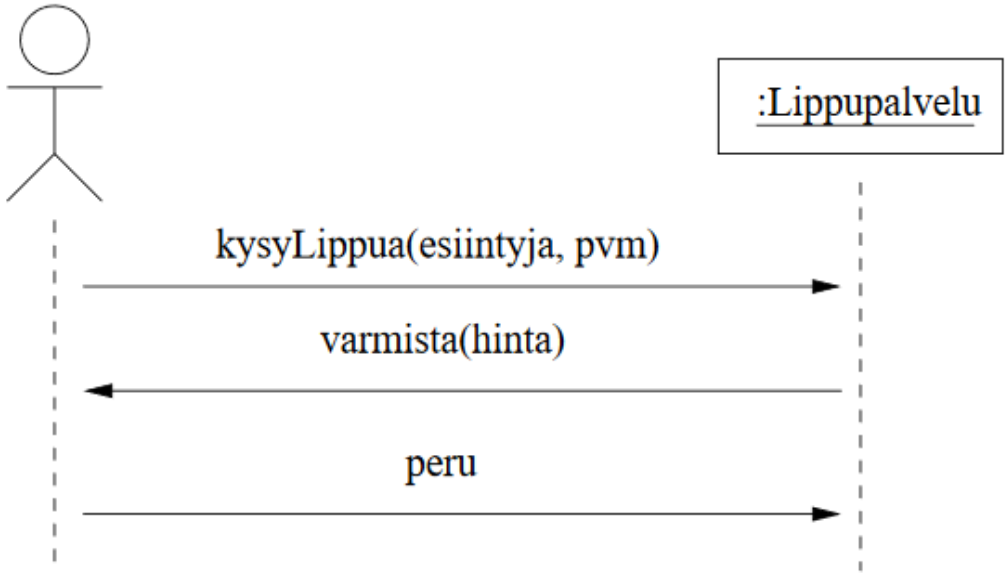
- Tarkastellaan alkeellista lippupalvelun tietojärjestelmää ja sen käyttötapausta
- Käyttötapauksen kulku:
  1. Käyttäjä kertoo tilaisuuden nimen ja päivämäärän
  2. Järjestelmä kertoo, minkä hintainen lippu on mahdollista ostaa
  3. Käyttäjä hyväksyy lipun
  4. Käyttäjälle annetaan tulostettu lippu



Sekvenssikaavio

Lippuvaraus esimerkki

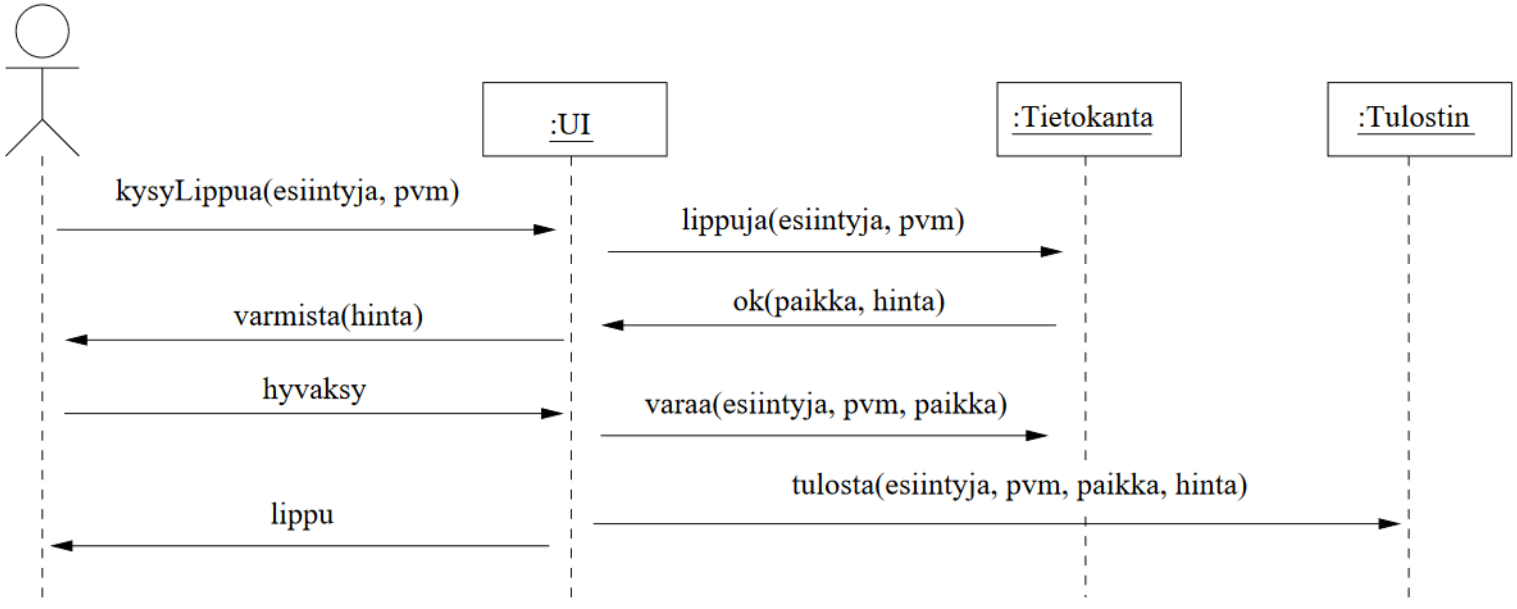
- Kuten näkyy voidaan sekvenssikaavioita käyttää myös oliosuunnittelussa!
  - Tästä voidaan saada helposti aikaan koodirunko
- Sekvenssikaaviot ovatkin usein käytössä oliosuunnittelun yhteydessä
- Jatketaan esimerkkiä, seuraavaksi järjestelmätason sekvenssikaaviona tilanne, jossa asiakas hylkää tarjotun lipun



Sekvenssikaavio

Lippuvaraus esimerkki

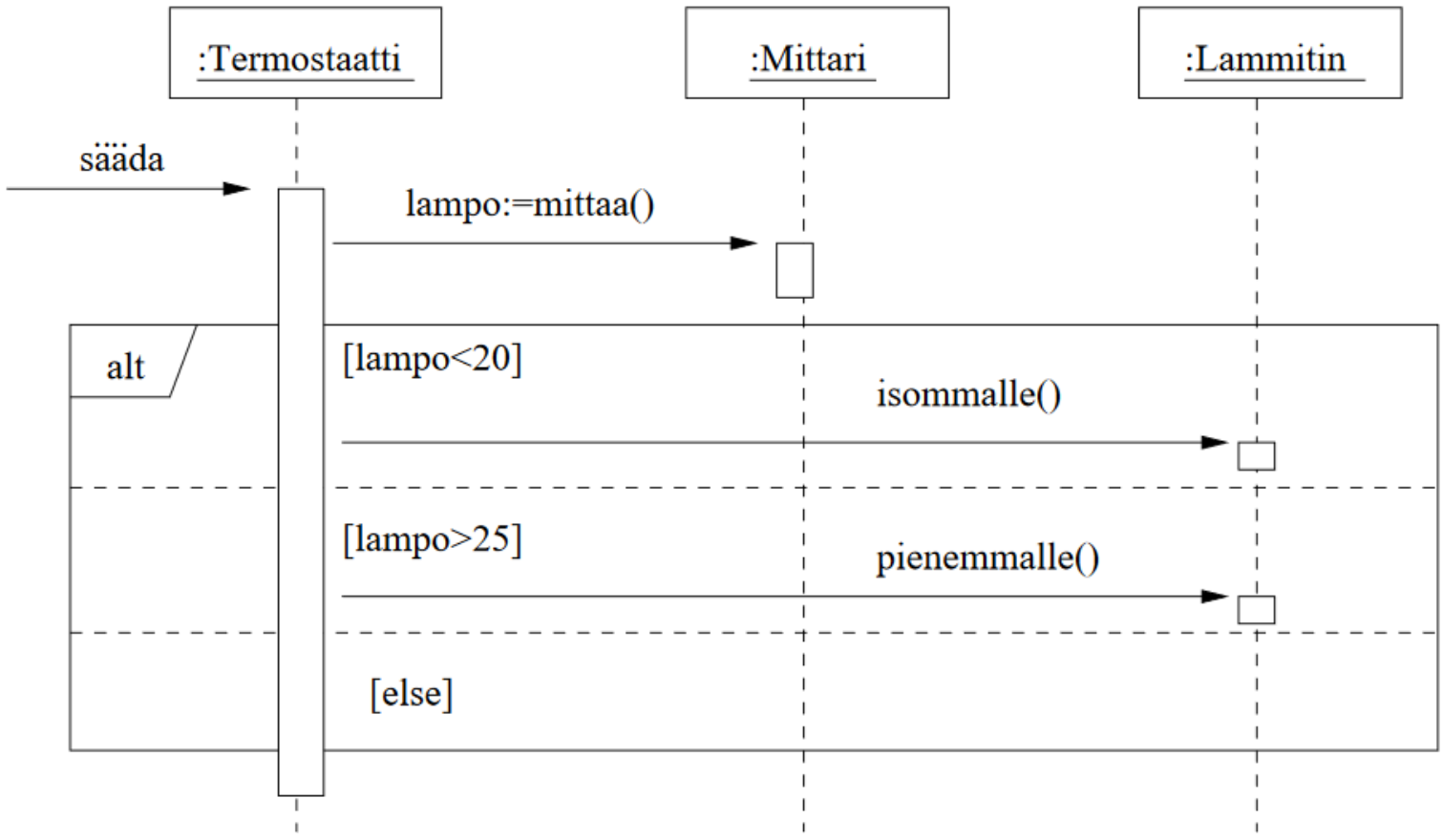
- Järjestelmätason sekvenssikaaviosta käy selkeästi ilmi käyttäjän ja järjestelmän interaktio
- Järjestelmän sisälle ei vielä katsota
- Seuraava askel on siirtyä suunnitteluun ja tarkentaa miten käyttötapauksen skenaario toteutetaan suunniteltujen olioiden yhteistyönä



Sekvenssikaavio

Valinnaisuus

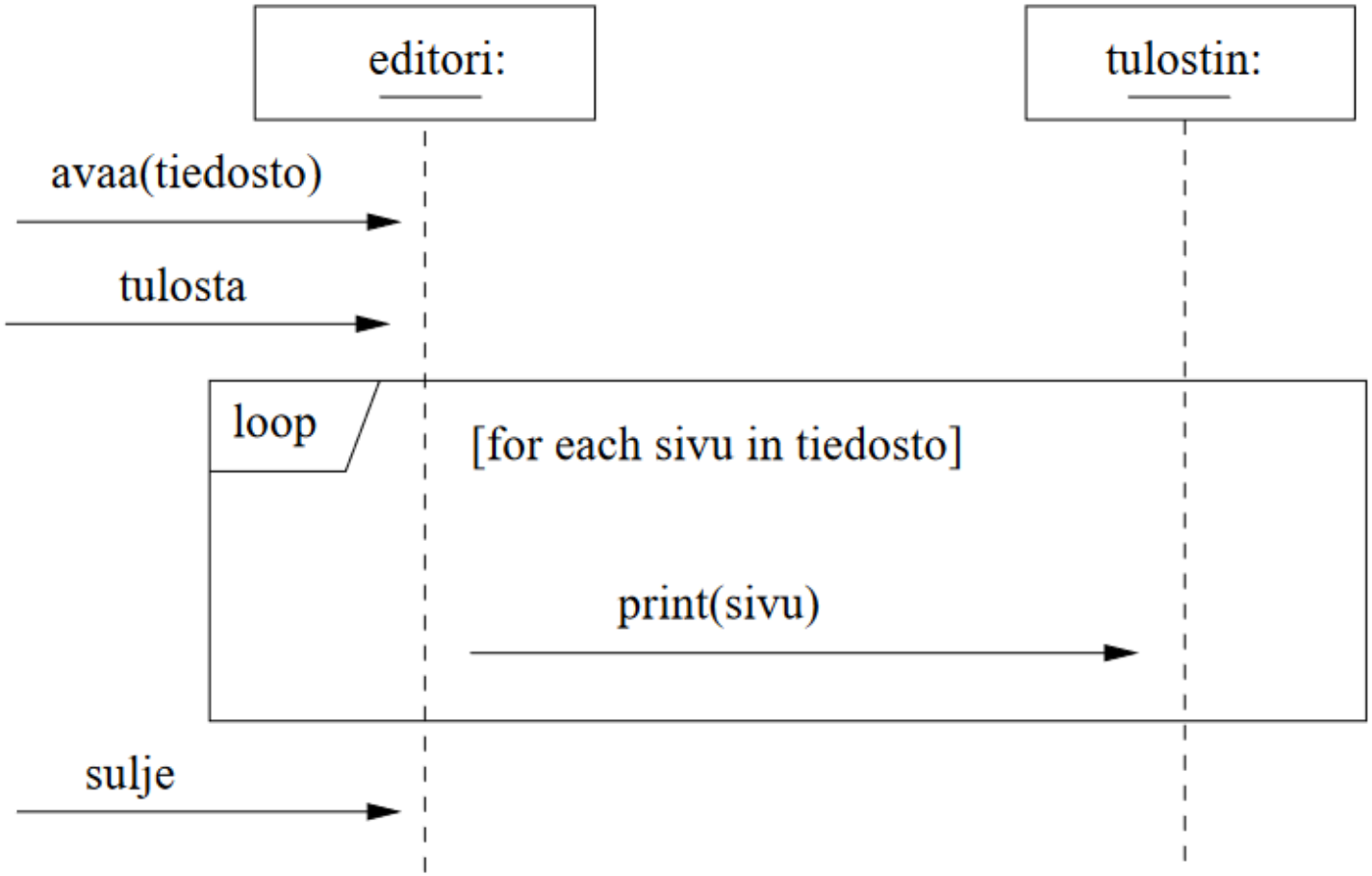
- Kaavioihin voidaan liittää lohko, jolla kuvataan valinnaisuutta
  - Vähän kuin if-else
  - Eli parametrina saadun arvon perusteella valitaan jokin kolmesta katkoviivan erottamasta alueesta



Sekvenssikaavio

Toisto

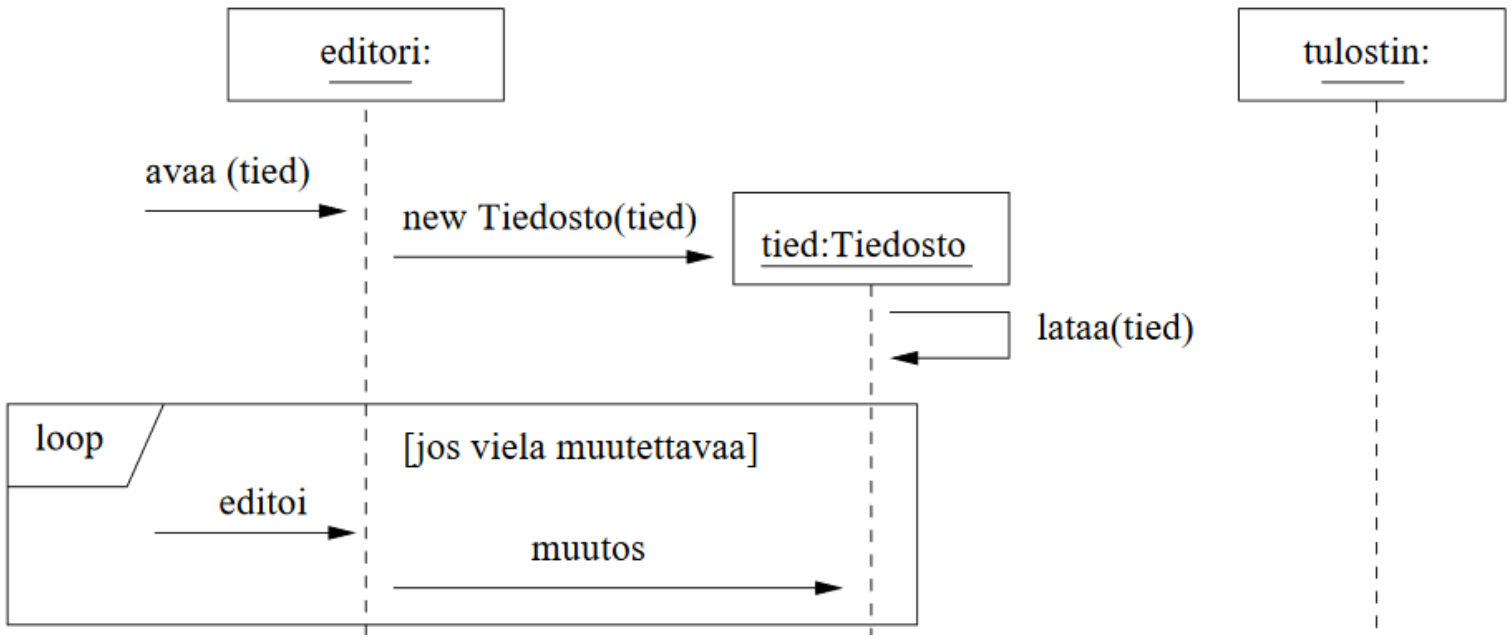
- Myös toistolohko mahdollinen (vrt. for tai while)
  - Huomaa miten toiston määrä on ilmaistu [ja] -merkkien sisällä
  - Voidaan käyttää myös vapaamuotoisempaa ilmausta, kuten "tulostetaan kaikki sivut erikseen"



Sekvenssikaavio

Olioiden luominen

- Esimerkki luentomonisteesta [5]
- **Huom:** Uusi olio ei aloita ylhäältä vaan vasta siitä kohtaa milloin se luodaan

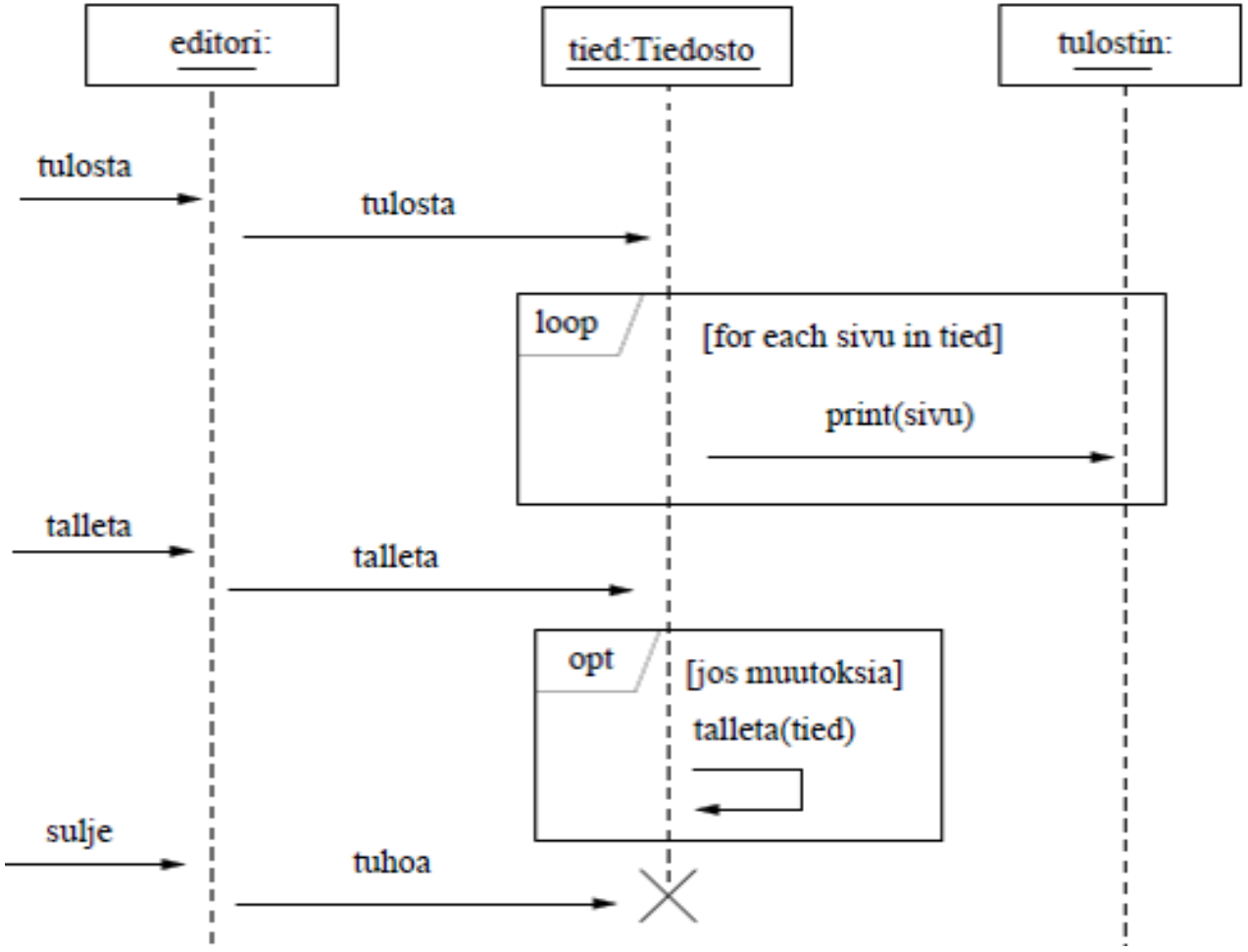


[5] Katso koko esimerkki sivulta 65

Sekvenssikaavio

Olioiden tuhoaminen

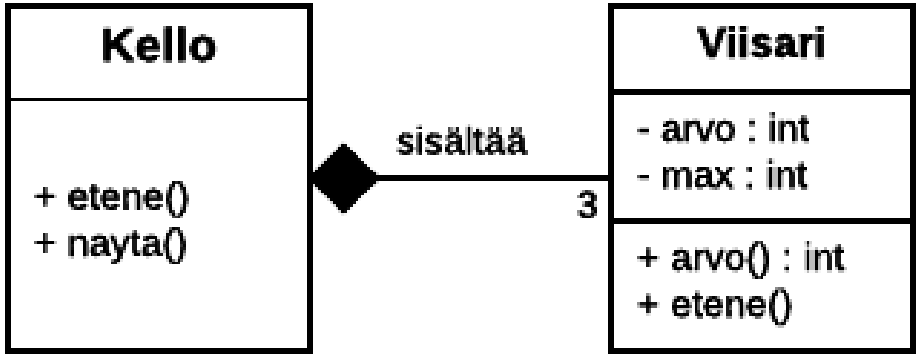
- Esimerkistä nähdään miten olion tuhoutuminen merkitään
- Mukana myös valinnainen (opt) lohko, joka suoritetaan jos ehto on tosi



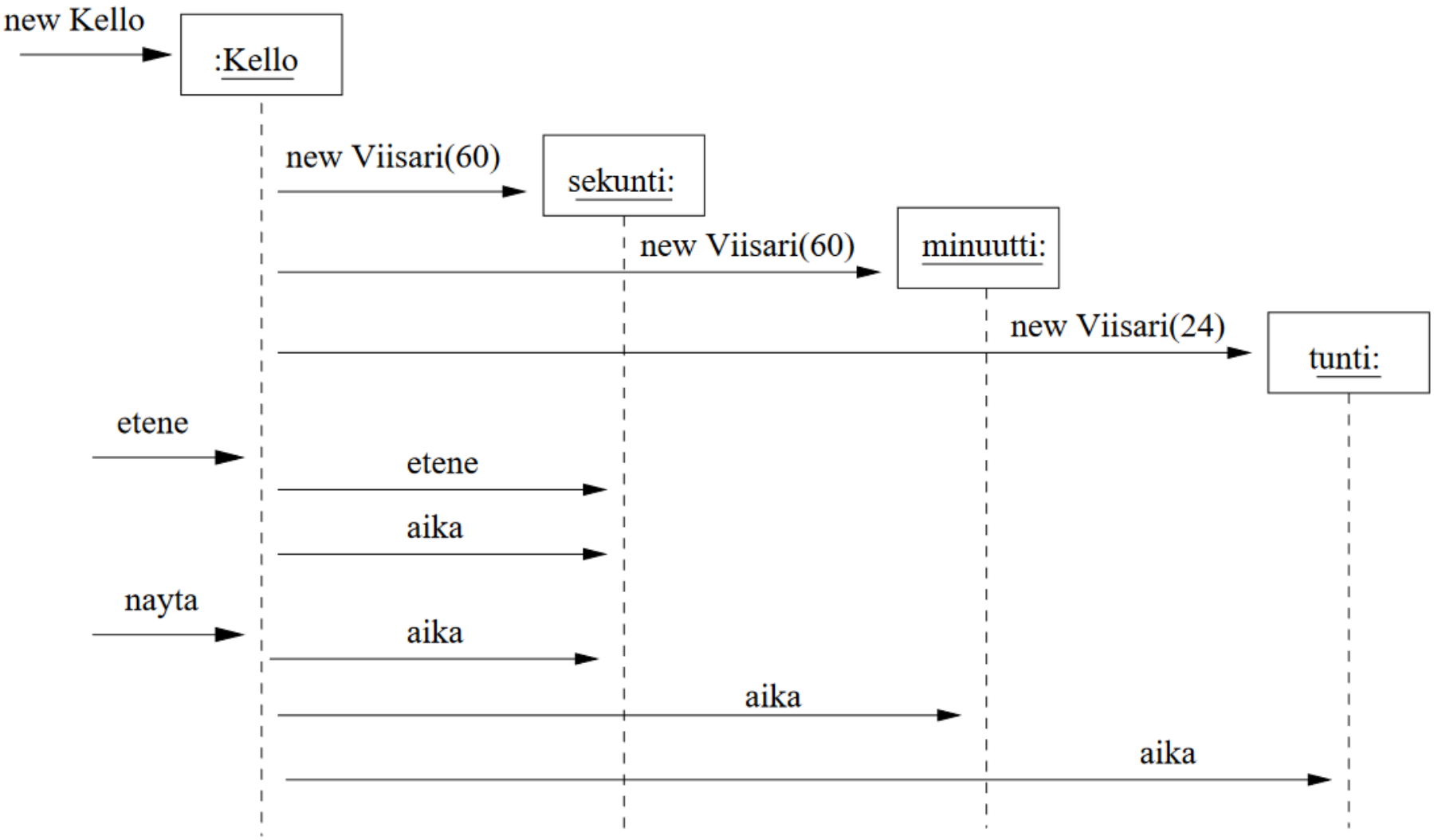
Sekvenssikaavio

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:**



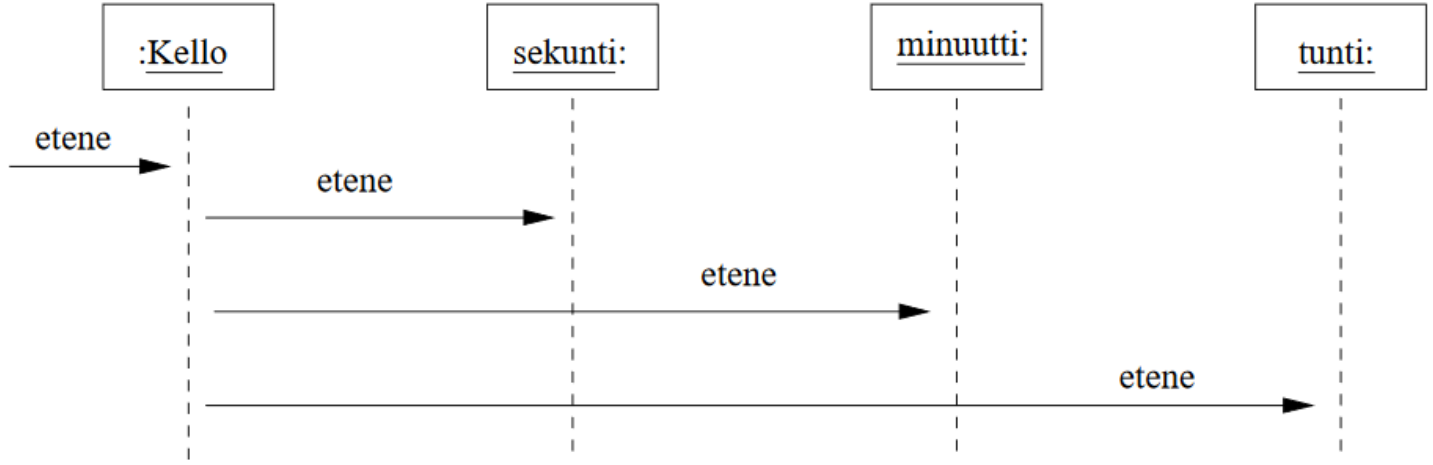
- Luokkakaaviosta ei vielä saa kuvaa kellon toimintalogiikasta joten tarvitaan sekvenssikaavioita $\rightarrow$


new Kello
:Kello
new Viisari(60)
sekunti:
new Viisari(60)
minuutti:
new Viisari(24)
tunti:
etene
etene
aika
nayta
aika
aika
aika

Sekvenssikaavio

Takaisinmallinnus

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



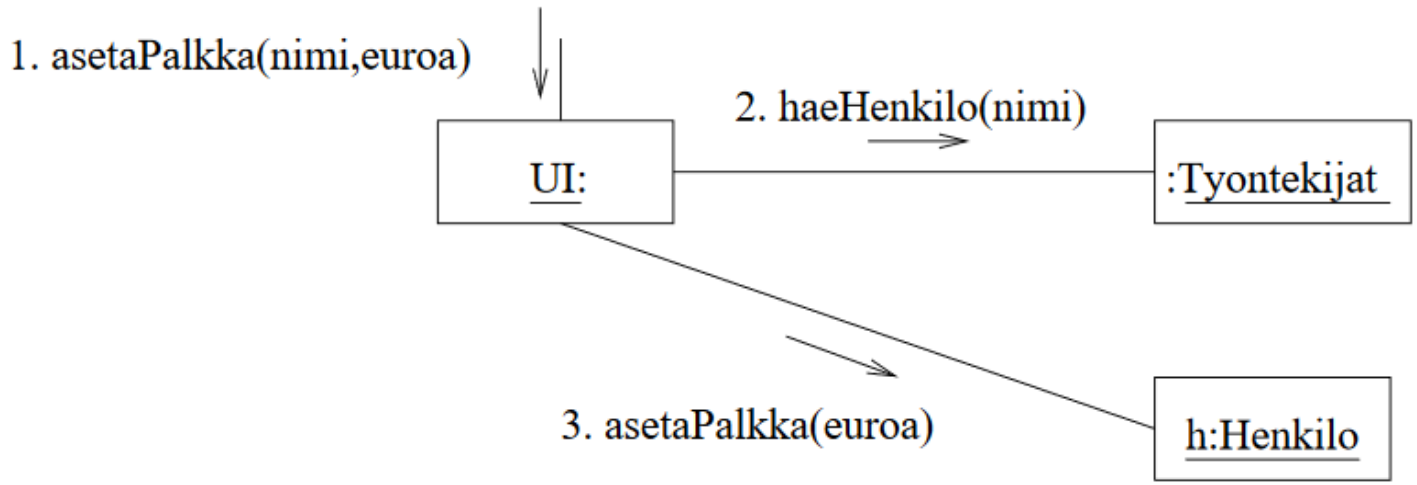
Kommunikaatiokaavio

Vaihtoehtoinen tapa kuvata yhteistoimintaa

Kommunikaatiokaavio

Vaihtoehtoinen tapa kuvata yhteistoimintaa

- Toinen tapa olioiden yhteistyön kuvaamiseen on kommunikaatiokaavio (engl. *communication diagram*) [6]
- Ohessa kalvon 150 Lippuvaraus -esimerkki
- Metodien suoritusjärjestys ilmenee numeroinnista, sijoittelu on vapaa

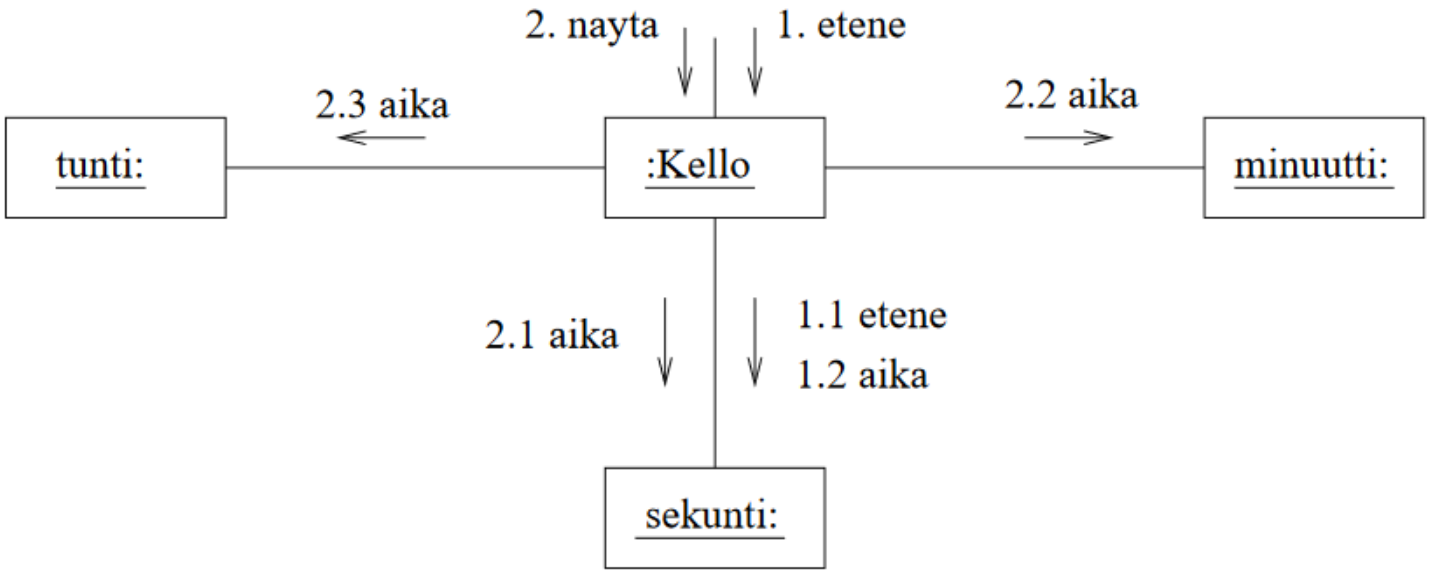


[6] Vanhalta nimitykseltä yhteistoimintakaavio (engl. *collaboration diagram*)

Kommunikaatiokaavio

Vaihtoehtoinen tapa kuvata yhteistoimintaa

- Viestien järjestyksen voi numeroida juoksevasti: 1, 2, 3, ...
- Tai allaolevan esimerkin[7] 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, ...



[7] Kello ja viisarit esimerkki

Kommunikaatiokaavio

Yhteenveto olioiden yhteistoiminnan kuvaamisesta

- Sekvenssikaavioita käytetään useammin kun kommunikaatiokaavioita
  - Sekvenssikavio lienee luokkakaavioiden jälkeen eniten käytetty UML-kaaviotyyppi
- Sekä sekvenssi- että kommunikaatiokaavioilla 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

Ohjelmistotekniikan menetelemät
Luento 4

Yhteyteen liittyvät tiedot, perintä ja rajapinnat

Ohjelmistotekniikan menetelemät
Luento 5

Ohjelmiston arkkitehtuuri, Oliosuunnittelu ja Koodin refaktorointi

Ohjelmistotekniikan menetelemät
Luento 6

TDD, Järjestelmätestaus ja kertausta