Ohjelmistotekniikan menetelmät

luento 7, 13.12.2016

Kertausta ja uutta asiaa testauksesta

Ohjelmistotuotantoprosessin vaiheita

- Kurssilla tutustuttu **ohjelmistotuotantoon**
  - yhteisnimitys niille työnteon ja työnjohdon menetelmille, joita käytetään, kun tuotetaan tietokoneohjelmia sekä monista tietokoneohjelmista koostuvia tietokoneohjelmistoja.
- *Ohjelmistotuotantoprosessiin* liittyy erilaisia vaiheita:
  1. Vaatimusmäärittely
  2. Suunnittelu
  3. Toteutus
  4. Testaus
  5. Ylläpito

Kertausta ja uutta asiaa testauksesta

Ohjelmistotuotantoprosessin vaiheita

- Kurssilla tutustuttu **ohjelmistotuotantoon**
  - yhteisnimitys niille työnteon ja työnjohdon menetelmille, joita käytetään, kun tuotetaan tietokoneohjelmia sekä monista tietokoneohjelmista koostuvia tietokoneohjelmistoja.
- *Ohjelmistotuotantoprosessiin* liittyy erilaisia vaiheita:
  1. Vaatimusmäärittely
  2. Suunnittelu
  3. Toteutus
  4. Testaus
  5. Ylläpito
- Perinteisesti vaiheet tehdään peräkkäin (*vesiputousmalli*)
- Vaihtoehtoinen tapa on työskennellä *iteratiivisesti*, eli toistaa vaiheita useaan kertaan peräkkäin ja limittäin (*Ketterä ohjelmistotuotanto*)

Kertausta ja uutta asiaa testauksesta

Ohjelmistotuotantoprosessin vaiheita

- Kurssilla tutustuttu **ohjelmistotuotantoon**
  - yhteisnimitys niille työnteon ja työnjohdon menetelmille, joita käytetään, kun tuotetaan tietokoneohjelmia sekä monista tietokoneohjelmista koostuvia tietokoneohjelmistoja.
- *Ohjelmistotuotantoprosessiin* liittyy erilaisia vaiheita:
  1. Vaatimusmäärittely
  2. Suunnittelu
  3. Toteutus
  4. Testaus
  5. Ylläpito
- Perinteisesti vaiheet tehdään peräkkäin (*vesiputousmalli*)
- Vaihtoehtoinen tapa on työskennellä *iteratiivisesti*, eli toistaa vaiheita useaan kertaan peräkkäin ja limittäin (*Ketterä ohjelmistotuotanto*)
- Ennen kun jatkamme kurssin tärkeimpien teemojen kertaamista, puhutaan hieman testaamisesta

Kertausta testauksesta

- Tarkoitus on varmistaa, että ohjelmisto on riittävän laadukas käytettäväksi

Kertausta testauksesta

- Tarkoitus on varmistaa, että ohjelmisto on riittävän laadukas käytettäväksi
- Jakautuu vaiheisiin, joita kutsutaan myös *testaustasoiksi*
  - **Yksikkötestaus**
    - Toimivatko yksittäiset metodit ja luokat kuten halutaan?
  - **Integraatiotestaus**
    - Toimivatko yksittäiset *moduulit* yhdessä halutulla tavalla?
  - **Järjestelmä/hyväksymistestaus**
    - Toimiiko kokonaisuus niin kuin vaatimusdokumentissa sanotaan?
  - **Regressiotestaus**
    - järjestelmälle muutosten ja bugikorjausten jälkeen ajettavia testejä jotka varmistavat että muutokset eivät riko mitään
    - Regressiotestit koostuvat yleensä yksikkö-, integraatio- ja järjestelmä/hyväksymätesteistä

Kertausta testauksesta

- Tarkoitus on varmistaa, että ohjelmisto on riittävän laadukas käytettäväksi
- Jakautuu vaiheisiin, joita kutsutaan myös *testaustasoiksi*
  - **Yksikkötestaus**
    - Toimivatko yksittäiset metodit ja luokat kuten halutaan?
  - **Integraatiotestaus**
    - Toimivatko yksittäiset *moduulit* yhdessä halutulla tavalla?
  - **Järjestelmä/hyväksymistestaus**
    - Toimiiko kokonaisuus niin kuin vaatimusdokumentissa sanotaan?
  - **Regressiotestaus**
    - järjestelmälle muutosten ja bugikorjausten jälkeen ajettavia testejä jotka varmistavat että muutokset eivät riko mitään
    - Regressiotestit koostuvat yleensä yksikkö-, integraatio- ja järjestelmä/hyväksymätesteistä

Lähes kaikki kurssilla tähän mennessä kirjoitetut testit ovat olleet yksikkötestejä, jotka on automatisoitu JUnitin avulla

Kertausta testauksesta

Testien laatu

- Normaalien syötteiden lisäksi on testeissä erityisen tärkeää tutkia testattavien metodien toimintaa *virheellisillä* syötteillä ja erilaisilla *raja-arvoilla*

Kertausta testauksesta

Testien laatu

- Normaalien syötteiden lisäksi on testeissä erityisen tärkeää tutkia testattavien metodien toimintaa *virheellisillä* syötteillä ja erilaisilla *raja-arvoilla*
- Testien laatua on kurssin aikana mitattu kolmella tavalla:
  - **Rivikattavuus** mittaa miten montaa prosenttia koodiriveistä testit suorittavat
  - **Haarautumakattavuus** taas mittaa miten monta prosenttia koodin haarautumakohdista (if:in ja toistolauseiden ehdot) on suoritettu

Kertausta testauksesta

Testien laatu

- Normaalien syötteiden lisäksi on testeissä erityisen tärkeää tutkia testattavien metodien toimintaa *virheellisillä* syötteillä ja erilaisilla *raja-arvoilla*
- Testien laatua on kurssin aikana mitattu kolmella tavalla:
  - **Rivikattavuus** mittaa miten montaa prosenttia koodiriveistä testit suorittavat
  - **Haarautumakattavuus** taas mittaa miten monta prosenttia koodin haarautumakohdista (if:in ja toistolauseiden ehdot) on suoritettu
  - **Mutaatiotestauksella** tarkasteltiin huomaavatko testit koodiin asetettuja bugeja
    - Mutaatiotestauskirjasto tekee koodiin pieniä muutoksia, eli mutantteja, esim. muuttaa ehdossa <:n <=:ksi
    - Jos testit eivät mene mutanttien takia rikki, on epäilys että testit eivät huomaisi koodissa olevaa virhettä ja testejä tulee parantaa
- Kun käytimme Maven-käännösympäristöä, oli testien laatumittareiden käyttö helppoa...

Kertausta testauksesta

Testaus ja riippuvuuksien eliminointi

- Testaus muuttuu joskus tarpeettoman hankalaksi, jos ohjelmissa on luokkien välillä tarpeettomia riippuvuuksia

Kertausta testauksesta

Testaus ja riippuvuuksien eliminointi

- Testaus muuttuu joskus tarpeettoman hankalaksi, jos ohjelmissa on luokkien välillä tarpeettomia riippuvuuksia
  - Eräs viime viikon laskareissa kokeillun Test Driven Development (TDD) -menetelmän hyviä puolia on se, että koodista tulee ”automaattisesti” hyvin testattavissa olevaa ja turhia riippuvuuksia ei koodissa yleensä esiinny

Kertausta testauksesta

Testaus ja riippuvuuksien eliminointi

- Testaus muuttuu joskus tarpeettoman hankalaksi, jos ohjelmissa on luokkien välillä tarpeettomia riippuvuuksia
  - Eräs viime viikon laskareissa kokeillun Test Driven Development (TDD) -menetelmän hyviä puolia on se, että koodista tulee ”automaattisesti” hyvin testattavissa olevaa ja turhia riippuvuuksia ei koodissa yleensä esiinny
- Seuraavalla kalvolla Ohjelmoinnin perusteiden viikon 3 tehtävän *Lottoarvonta* ratkaisu
  - Lottorivi-olion konstruktori luo Random-olion, jonka avulla lottonumerot arvotaan
- Luokan testaaminen on nyt erittäin vaikeaa, testeissähän ei pystytä kontrolloimaan Random-olion arpomia lukuja
- Lottorivistä tulisi testata ainakin seuraavat asiat
  - lottorivi ei voi sisältää samaa arvoa useampaan kertaan
  - lottorivi voi saada arvoja väliltä 1-39

Kertausta testauksesta

Lottorivi ja riippuvuuden injektointi

```
public class Lottorivi {
    private ArrayList<integer> numerot;
    private Random random;

    public Lottorivi() {
        random = new Random();
        arvoNumerot();
    }

    public ArrayList<integer> numerot() {
        return numerot;
    }

    public void arvoNumerot() {
        numerot = new ArrayList<>();
        while (numerot.size() < 7) {
            int numero = random.nextInt(39) + 1;
            if (!numerot.contains(numero)) {
                numerot.add(numero);
            }
        }
    }
}
```

Kertausta testauksesta

Lottorivi ja riippuvuuden injektointi

- Ongelmana Lottorivin testaamisessa ei sinänsä ole riippuvuus Random-olioon, vaan se, että lottorivin ulkopuolelta ei ole tapaa päästä käsiksi Lottorivin luomaan Random-olioon

Kertausta testauksesta

Lottorivi ja riippuvuuden injektointi

- Ongelmana Lottorivin testaamisessa ei sinänsä ole riippuvuus Random-olioon, vaan se, että lottorivin ulkopuolelta ei ole tapaa päästä käsiksi Lottorivin luomaan Random-olioon
- Muutetaan lottorivin konstruktoria seuraavasti:

```
public Lottorivi(Random random)
```

Kertausta testauksesta

Lottorivi ja riippuvuuden injektointi

- Ongelmana Lottorivin testaamisessa ei sinänsä ole riippuvuus Random-olioon, vaan se, että lottorivin ulkopuolelta ei ole tapaa päästä käsiksi Lottorivin luomaan Random-olioon
- Muutetaan lottorivin konstruktoria seuraavasti:

```
public Lottorivi(Random random)
```

- Eli lottorivi muodostetaan nyt luomalla Random-olio, joka annetaan lottoriville konstruktorin parametrina

```
Random random = new Random();
Lottorivi lotto = new Lottorivi( random );
```

Kertausta testauksesta

Lottorivi ja riippuvuuden injektointi

- Ongelmana Lottorivin testaamisessa ei sinänsä ole riippuvuus Random-olioon, vaan se, että lottorivin ulkopuolelta ei ole tapaa päästä käsiksi Lottorivin luomaan Random-olioon
- Muutetaan lottorivin konstruktoria seuraavasti:

```
public Lottorivi(Random random)
```

- Eli lottorivi muodostetaan nyt luomalla Random-olio, joka annetaan lottoriville konstruktorin parametrina

```
Random random = new Random();
Lottorivi lotto = new Lottorivi( random );
```

- Tekniikasta käytetään nimitystä **riippuvuuksien injektointi** (engl. *dependency injection*)
  - riippuvuutena olevaa olioa ei luoda konstruktorissa tai luokan sisällä
  - konstruktorille annetaan valmiiksi luotu riippuvuus, konstruktori laittaa riippuvuutena olevan olion talteen oliomuuttujaan

Kertausta testauksesta

Lottorivi ja riippuvuuden injektointi

```
public class Lottorivi {
        private ArrayList<Integer> numerot;
        private Random random;

        public Lottorivi(Random random) {
                this.random = random;
                arvoNumerot();
        }

        public Lottorivi() {  // olio mahdollista luoda myös vanhaan tapaan
                this(new Random());
        }

        public ArrayList<Integer> numerot() {
                return numerot;
        }

        public void arvoNumerot() {
                numerot = new ArrayList<>();
                while (numerot.size() < 7) {
                        int numero = random.nextInt(39) + 1;
                        if (!numerot.contains(numero)) {
                                numerot.add(numero);
                        }
                }
        }
}
```

Kertausta testauksesta

Lottorivi ja riippuvuuden injektointi

- Koska lottorivi saa käyttämänsä Random-olion konstruktorin parametrina, voimme helposti luoda *omia versioitamme Randomista* ja käyttää niitä apuna testaamisessa
- Seuraavassa luokka *RandomStub*, joka "arpoo" sille konstruktorin parametrina annetut luvut

```
public class RandomStub extends Random {
        List<Integer> numbers;
        public RandomStub(Integer ... luvut) {
            numbers = new ArrayList<>();
            numbers.addAll(Arrays.asList(luvut));
        }

        @Override // korvataan peritty toiminnallisuus
        public int nextInt(int bound) {
            return numbers.remove(0);
        }
}
```

Kertausta testauksesta

Lottorivi ja riippuvuuden injektointi

- Koska lottorivi saa käyttämänsä Random-olion konstruktorin parametrina, voimme helposti luoda *omia versioitamme Randomista* ja käyttää niitä apuna testaamisessa
- Seuraavassa luokka *RandomStub*, joka "arpoo" sille konstruktorin parametrina annetut luvut

```
public class RandomStub extends Random {
        List<Integer> numbers;
        public RandomStub(Integer ... luvut) {
            numbers = new ArrayList<>();
            numbers.addAll(Arrays.asList(luvut));
        }

        @Override // korvataan peritty toiminnallisuus
        public int nextInt(int bound) {
            return numbers.remove(0);
        }
}
```

- Koska RandomStub *perii* luokan Random, voidaan sitä käyttää kaikkialla randomin paikalla

Kertausta testauksesta

Lottorivi ja riippuvuuden injektointi

- Testaaminen muuttuu nyt helpoksi
- Seuraavassa testi, joka varmistaa, että sama luku ei esiinny useaan kertaan lottonumerossa, vaikka Random-olio palauttaakin saman luvun monta kertaa

```
public class LottoriviTest {
    @Test
    public void eiSamojaNumeroita() {
        Random randomStubi = new RandomStub(1,1,1,2,3,4,5,6,7);
        Lottorivi rivi = new Lottorivi(randomStubi);
        List<Integer> odotettu = Arrays.asList(2,3,4,5,6,7,8);
        assertSamatNumerot(odotettu, rivi.numerot());
    }

    private void assertSamatNumerot(List<Integer> odotettu, List<Integer> tulos) {
        assertEquals(odotettu.size(), tulos.size());
        for (Integer luku : tulos) {
            assertTrue(odotettu.contains(luku));
       }
    }
}
```

Integraatiotestaus

Integraatiotestaus

- Yksikkötestaus on käsitteenä suhteellisen hyvin ymmärretty
  - yksikkötestit kohdistuvat yhteen luokkaan
- Järjestelmä/hyväksymätestauksessa taas testataan ohjelmiston tarjoamaa toiminnallisuutta käyttöliittymän läpi samaan tapaan kuin loppukäyttäjä tulee järjestelmää käyttämään

Integraatiotestaus

- Yksikkötestaus on käsitteenä suhteellisen hyvin ymmärretty
  - yksikkötestit kohdistuvat yhteen luokkaan
- Järjestelmä/hyväksymätestauksessa taas testataan ohjelmiston tarjoamaa toiminnallisuutta käyttöliittymän läpi samaan tapaan kuin loppukäyttäjä tulee järjestelmää käyttämään
- Kaikki näiden väliin jäävät testauksen muodot ovat *integraatiotestausta*
- Kurssilla on jo nähty muutamia esimerkkejä integraatiotesteistä
- Viikon 6 laskareissa olleen palkanlaskennan testit ovat oikeastaan integraatiotestejä https://github.com/mluukkai/OTM2016/tree/master/koodi/Palkanl
- Vaikka testit kohdistuvatkin luokan Palkanlaskenta metodeihin, ne kuitenkin oleellisesti testaavat palkanlaskennasta, henkilöistä ja muutamasta muusta luokasta muodostuvan kokonaisuuden toimintaa

Integraatiotestaus

Integraatiotestaus ja ”hankalat” riippuvuudet

- Testien kannalta on hieman ikävää, jos testattavan järjestelmän jokin komponentti keskustelee tietokannan tai jonkin verkossa olevan komponentin kanssa
  - Testien suoritus hidastuu
  - Testien tulos saattaa olla riippuvainen tietokannan sisällöstä
  - Verkon yhteysongelmat voivat vaikuttaa testien toimivuuteen

Integraatiotestaus

Integraatiotestaus ja ”hankalat” riippuvuudet

- Testien kannalta on hieman ikävää, jos testattavan järjestelmän jokin komponentti keskustelee tietokannan tai jonkin verkossa olevan komponentin kanssa
  - Testien suoritus hidastuu
  - Testien tulos saattaa olla riippuvainen tietokannan sisällöstä
  - Verkon yhteysongelmat voivat vaikuttaa testien toimivuuteen
- Näissä tilanteissa on järkevää toteuttaa hankalista luokista testausta varten valekomponentti eli *stub* ja injektoida se testattavalle järjestelmälle hankalan riippuvuuden sijaan
  - Stubin tulee toimia testattavan järjestelmän kannalta kuten oikea komponentti
- Tämä tietysti edellyttää, että riippuvuudet pystytään injektoimaan testattaville luokille samoin kuin lottoesimerkissä

Integraatiotestaus

Integraatiotestaus ja ”hankalat” riippuvuudet

- Testien kannalta on hieman ikävää, jos testattavan järjestelmän jokin komponentti keskustelee tietokannan tai jonkin verkossa olevan komponentin kanssa
  - Testien suoritus hidastuu
  - Testien tulos saattaa olla riippuvainen tietokannan sisällöstä
  - Verkon yhteysongelmat voivat vaikuttaa testien toimivuuteen
- Näissä tilanteissa on järkevää toteuttaa hankalista luokista testausta varten valekomponentti eli *stub* ja injektoida se testattavalle järjestelmälle hankalan riippuvuuden sijaan
  - Stubin tulee toimia testattavan järjestelmän kannalta kuten oikea komponentti
- Tämä tietysti edellyttää, että riippuvuudet pystytään injektoimaan testattaville luokille samoin kuin lottoesimerkissä
- Stub-olio on helppo luoda perimällä alkuperäinen riippuvuus ja *korvata* kokonaan siinä testien kannalta merkityksellisten metodien toiminnallisuus

Integraatiotestaus

Integraatiotestaus ja ”hankalat” riippuvuudet

- Paalkkojen maksun yhteydessä suoritetaan tilisiirto luokan MaksupalveluRajapinta metodilla *suoritaTilisiirto*
- Konkreettisen tilisiirron tekeminen on tietenkin testien kannalta täysin tarpeetonta
- Palkanlaskennan mallivastauksen testeissä käytetäänkin todellisen maksupalvelurajapinnan sijaan stub-olioa:

Integraatiotestaus

Integraatiotestaus ja ”hankalat” riippuvuudet

- Paalkkojen maksun yhteydessä suoritetaan tilisiirto luokan MaksupalveluRajapinta metodilla *suoritaTilisiirto*
- Konkreettisen tilisiirron tekeminen on tietenkin testien kannalta täysin tarpeetonta
- Palkanlaskennan mallivastauksen testeissä käytetäänkin todellisen maksupalvelurajapinnan sijaan stub-olioa:

```
public class MaksupalveluStub extends MaksupalveluRajapinta {
    @Override
    public void suoritaTilisiirto(Maksusuoritus maksu) {
        // stubin ei tarvitse tehdä mitään sillä Palkanlaskenta ainoastaan
    }
}

public class PalkanlaskentaTest {
    Palkanlaskenta p;
    @Before
    public void setUp() {
        Map<String, OutputBuilder> outputBuilders = new HashMap<>();
        outputBuilders.put("csv", new CsvBuilder());
        p = new Palkanlaskenta(new MaksupalveluStub(), outputBuilders);
    }
```

Järjestelmätestaus

Järjestelmätestaus

Järjestelmätestauksessa tulee varmistaa toimiiko ohjelmisto sille vaatimusmäärittelyssä asetettujen vaatimusten mukaan

- Testauksen tulee tapahtua saman rajapinnan läpi, miten loppukäyttäjä järjestelmää käyttää, eli sovelluksesta riippuen joko komentoriviltä, graafisesta käyttöliittymästä tai webselaimesta
- Järjestelmätestit kannattaa muodostaa ohjelmiston käyttötapausten mukaan

Järjestelmätestaus

Järjestelmätestauksessa tulee varmistaa toimiiko ohjelmisto sille vaatimusmäärittelyssä asetettujen vaatimusten mukaan

- Testauksen tulee tapahtua saman rajapinnan läpi, miten loppukäyttäjä järjestelmää käyttää, eli sovelluksesta riippuen joko komentoriviltä, graafisesta käyttöliittymästä tai webselaimesta
- Järjestelmätestit kannattaa muodostaa ohjelmiston käyttötapausten mukaan
- Tarkastellaan Ohjelmoinnin jatkokurssin ensimmäisen viikon tehtävää *Laskin*
- Käyttäjän syötteiden simuloiminen on mahdollista korvaamalla ohjelman käyttämä syötevirta *System.in* omalla oliolla
- Vastaavasti voimme korvata tulostusvirran *System.out*

Järjestelmätestaus

Laskimen testi, joka korvaa syöte- ja tulostevirran

- Järjestelmätestauksen voi hoitaa myös JUnit-kirjaston avulla.
  - JUnit ei ole tarkoitukseen optimaalinen, mutta ”riittävän hyvä”, ja Javalle ei kauheasti parempiakaan vaihtoehtoja ole tarjolla

Järjestelmätestaus

Laskimen testi, joka korvaa syöte- ja tulostevirran

- Järjestelmätestauksen voi hoitaa myös JUnit-kirjaston avulla.
  - JUnit ei ole tarkoitukseen optimaalinen, mutta ”riittävän hyvä”, ja Javalle ei kauheasti parempiakaan vaihtoehtoja ole tarjolla

```
public class LaskinTest {
    private ByteArrayOutputStream tulosvirta;

    @Beforepublic void setUp() {
        tulosvirta = new ByteArrayOutputStream();
        System.setOut(new PrintStream(tulosvirta));
    }

    @Test
    public void summaJaLopetus() {
        String syote = "summa\n"+"1\n"+"2\n"+"lopetus\n";
        System.setIn(new ByteArrayInputStream(syote.getBytes()));
        Laskin laskin = new Laskin();laskin.kaynnista();
        String tulostus = tulosvirta.toString();

        assertTrue(tulostus.contains("summa 3"));
        assertTrue(tulostus.contains("laskuja laskettiin 1"));
    }
}
```

Järjestelmätestaus

Laskimen testi, joka korvaa syöte- ja tulostevirran

- Testin simuloitu syöte on merkkijono, jossa yksittäiset syöterivit päättyvät rivinvaihtoon
- Merkkijonosta luodaan bittivirta, joka asetetaan *ohjelman syötevirran* arvoksi
- setUp-metodissa testi asettaa uuden ByteArrayOutputStream-olion ohjelman tulosvirraksi metodin System.setOut avulla
- Laskimen suorituksen jälkeen tulosvirtaolio muutetaan toString-metodilla merkkijonoksi ja tarkastetaan, että ohjelman tulostus on odotetun kaltainen
- Testit olettavat, että ohjelmassa luodaan ainoastaan yksi Scanner-olio
- Vaihtoehtoinen ratkaisu syötevirran korvaamiselle olisi ollut muuttaa laskinta siten, että lukija-olio oltaisiin injektoitu konstruktorin parametrina

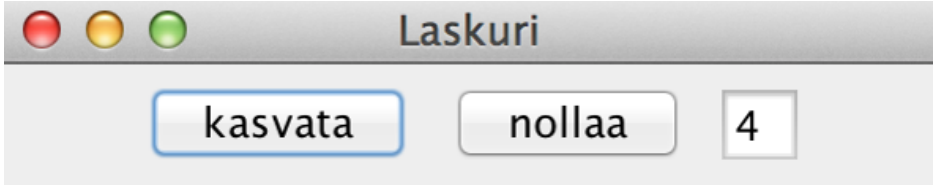
Graafisen käyttöliittymän testaus

Java-Swing käyttöliittymän testausta

Graafisen käyttöliittymän testaus

Java-Swing käyttöliittymän testausta

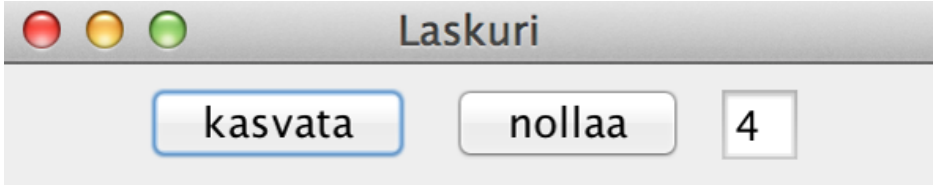
- Tarkastellaan yksinkertaista graafista ohjelmaa, jonka avulla laskurin arvoa voidaan kasvattaa tai laskuri voidaan nollata



Graafisen käyttöliittymän testaus

Java-Swing käyttöliittymän testausta

- Tarkastellaan yksinkertaista graafista ohjelmaa, jonka avulla laskurin arvoa voidaan kasvattaa tai laskuri voidaan nollata



- Javan Swing-kirjastolla tehtyjen graafisten sovellusten testaaminen ei loppujenlopuksi ole kovin vaikeaa
- Asetetaan komponenteille nimet:

```
private void luoKomponentit(Container container) {
    add = new Jbutton("kasvata");
    add.setName("kasvata");
    reset = new Jbutton("nollaa");
    reset.setName("nollaa");
    reset.setEnabled(false);
    show = new JTextField("  "+value);
    show.setName("tulos");
}
```

Graafisen käyttöliittymän testaus

Java-Swing käyttöliittymän testausta

- AssertJ [1] on johtava työkalu Swing-sovellusten testaamiseen:

```
public class LaskuriTest extends AssertJSwingJUnitTestCase {
    private FrameFixture window;

    @Test
    public void laskuriAlussaNolla() {
        window.textBox("tulos").requireText("0");
    }

    @Test
    public void laskuriKasvaa() {
        window.button("kasvata").click();
        window.textBox("tulos").requireText("1");
        window.button("kasvata").click();
        window.button("kasvata").click();
        window.textBox("tulos").requireText("3");
    }
}
```

- window-olio tarjoaa pääsyn eri komponentteihin. Normaalisti käytettävien assert-lauseiden sijaan testien odotettu tulos määritellään require-määreiden avulla

[1] http://joel-costigliola.github.io/assertj/

Kurssikoe

Tietoa kurssikokeesta

Kurssikoe

Tietoa kurssikokeesta

- **Koe pidetään tiistaina 20.12. klo 17-20** salissa A111
  - Kokeessa vastausaikaa 2 tuntia 30 minuuttia
  - Jos et pääse kokeeseen, järjestetään korvaava koetilaisuus maanantaina 19.12. klo 12
  - maanantain kokeeseen ilmoittautuminen viimeistään torstaina 15.12.

Kurssikoe

Tietoa kurssikokeesta

- **Koe pidetään tiistaina 20.12. klo 17-20** salissa A111
  - Kokeessa vastausaikaa 2 tuntia 30 minuuttia
  - Jos et pääse kokeeseen, järjestetään korvaava koetilaisuus maanantaina 19.12. klo 12
  - maanantain kokeeseen ilmoittautuminen viimeistään torstaina 15.12.
- Nippelitason detaljien sijasta kokeessa arvostetaan enemmän sovellusosaamista
  - Tosin eräät detaljit (mainitaan myöhemmin) ovat tärkeitä
- Kokeeseen saa tuoda mukanaan 2-puoleisen, **itse tehdyn, käsin kirjoitetun** yhden A4-arkin lunttilapun.
- Lunttilapun teossa siis ei saa käyttää kopiokonetta, tietokonetta tai printteriä
  - Yksikään kokeen kysymys ei tosin tule olemaan sellainen että kukaan onnistuisi kirjoittamaan vastauksen etukäteen lunttilapulle

Kurssikoe

Mikä on tärkeintä kurssilla/kokeessa?

- Kokonaiskuva: *Mitä? Miksi? Milloin? Missä? Miten?*
- **Ohjelmistotuotantoprosessin** vaiheet on syytä osata
- **Testaamisen rooli** ohjelmistotuotantoprosessissa ymmärrettävä
  - testejäkin saattaa joutua kirjoittamaan
  - kannattaa kerrata laskareiden testaukseen liittyvät asiat

Kurssikoe

Mikä on tärkeintä kurssilla/kokeessa?

- Kokonaiskuva: *Mitä? Miksi? Milloin? Missä? Miten?*
- **Ohjelmistotuotantoprosessin** vaiheet on syytä osata
- **Testaamisen rooli** ohjelmistotuotantoprosessissa ymmärrettävä
  - testejäkin saattaa joutua kirjoittamaan
  - kannattaa kerrata laskareiden testaukseen liittyvät asiat
- UML:sta ylivoimaisesti tärkeimpiä ovat
  - **Luokkakaaviot**
  - **Sekvenssikaaviot**
  - **Käyttötapauskaaviot**

Kurssikoe

Mikä on tärkeintä kurssilla/kokeessa?

- **Käyttötapausmallinnus**
- **Käsiteanalyysi**, eli määrittelyvaiheen luokkamallin muodostaminen tekstistä
- **Takaisinmallinnus**, eli valmiista koodista luokka- ja sekvenssikaavioiden teko
- **Ohjelmiston suunnittelu**
  - jakautuminen *ohjelmiston arkkitehtuuriin*
  - ja oliosuunnitteluun

Kurssikoe

Mikä on tärkeintä kurssilla/kokeessa?

- **Käyttötapausmallinnus**
- **Käsiteanalyysi**, eli määrittelyvaiheen luokkamallin muodostaminen tekstistä
- **Takaisinmallinnus**, eli valmiista koodista luokka- ja sekvenssikaavioiden teko
- **Ohjelmiston suunnittelu**
  - jakautuminen *ohjelmiston arkkitehtuuriin*
  - ja oliosuunnitteluun
- kokeessa ei tarvitse osata
  - Komentorivin käyttöä
  - Mavenia
- Mitä gitistä tulee osata?
  - tba

Kurssikoe

Oliosuunnittelusta kokeessa

- Oliosuunnittelun tehtävä on löytää/keksiä oliot ja niiden operaatiot siten, että ohjelma toimii kuten vaatimuksissa halutaan
  - Erittäin haastavaa: enemmän "art" kuin "science"
- Kurssilla on tarkasteltu muutamia yleisiä oliosuunnittelun periaatteita
  - **Single responsibility** eli luokilla vain yksi vastuu
  - **Program to an interface, not to concrete implementation**, eli suosi rajapintoja
  - **Favor composition over inheritance**, eli älä väärinkäytä perintää
  - Tarpeettomien **riippuvuuksien minimointi**
- Merkki huonosta suunnittelusta: koodihaju (engl. *code smell*)
- Lääke huonoon suunnitteluun/koodihajuun: refaktorointi
  - Muutetaan koodin rakennetta parempaan suuntaan muuttamatta toiminnallisuutta

Kurssikoe

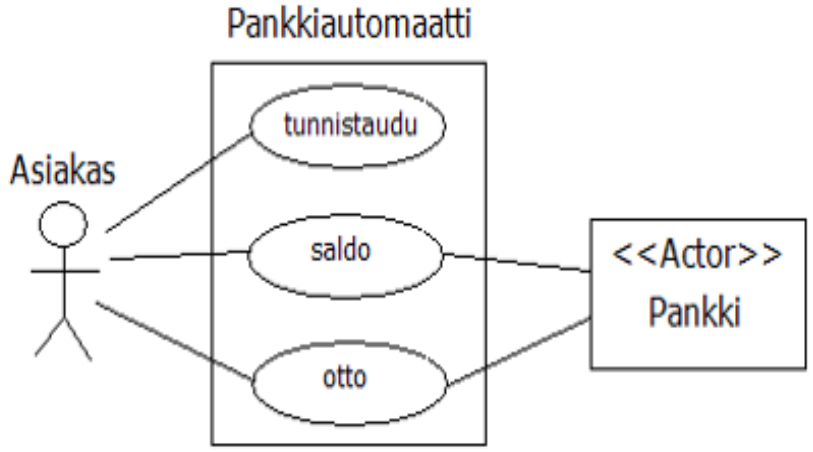
Oliosuunnittelusta kokeessa

- Edellisellä viikolla tarkastelimme Todo-sovelluksen yhteydessä oliosuunnittelua
- Laskareissa kokeiltiin **Test Driven Development** -menetelmää, joka ”yhdistää” testauksen, ohjelmoinnin ja suunnittelun
- **Mitä aiheesta pitää osata kokeessa?**
  - Yleiset periaatteet
  - Periaatteiden soveltaminen yksinkertaisessa tilanteessa
    - Esim. huonon koodin tunnistaminen ja refaktorointi paremmaksi
  - Ymmärrys, miksi periaatteet ovat olemassa ja milloin niitä saa rikkoa ja mitä käy jos niitä rikkoo (tekninen velka)

Käyttötapaukset ja käyttötapauskaaviot

Käyttötapaukset ja käyttötapauskaaviot

- Pankkiautomaatin käyttötapaukset ovat tunnistaudu, saldo ja otto
- Käyttötapausten käyttäjät (engl. actor) eli toimintaan osallistuvat tahot ovat Asiakas ja Pankki
- Tikku-ukolle vaihtoehtoinen tapa merkitä käyttäjä on laatikko, jossa tarkenne «actor»
- Käyttötapausellipsiin yhdistetään viivalla kaikki sen käyttäjät
- Kuvaan ei siis piirretä nuolia!



Käyttötapaukset ja käyttötapauskaaviot

- Käyttötapausten ja käyttäjien suhdetta kuvaa UML:n käyttötapauskaavio
- Huomattavaa on, että koska käyttötapausmallia käytetään vaatimusmäärittelyssä, ei ole syytä puuttua järjestelmän sisäisiin yksityiskohtiin
  - Pankkiautomaatin sisäiseen toimintaan ei oteta kantaa
  - Mitään automaatin sisäistä (esim. tietokantaa tai tiedostoja) ei merkitä kaavioon
  - Tarkastellaan ainoastaan, miten automaatin toiminta näkyy ulospäin
- Itse käyttötapauksen sisältö kuvataan *tekstuaalisesti*, käyttäen sopivaa *käyttötapauspohjaa*
- Käyttötapauskaavio toimii hyvänä yleiskuvana, mutta vasta tekstuaalinen kuvaus määrittelee toiminnan tarkemmin
  - Seuraavalla sivulla käyttötapauksen Otto tarkennus
- Seuraavana vaiheena saattaisi olla tarkemman, käyttöliittymäspesifisen käyttötapauksen kirjoittaminen

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. pankkikortti palautetaan asiakkaalle
8. rahat annetaan asiakkaalle

Poikkeustapaukset:

4a asiakkaan tilillä ei tarpeeksi rahaa, palautetaan kortti

Luokkakaaviot

Luokkakaaviot

mallinnuksen abstraktiotaso vaihtelee

- **Määrittelyvaiheen luokkamalli** kuvailee sovellusalueen käsitteitä ja niiden suhteita
  - Muodostetaan usein *käsiteanalyysin* avulla
  - Käytetään myös nimitystä **kohdealueen luokkamalli** (engl. domain model)

Luokkakaaviot

mallinnuksen abstraktiotaso vaihtelee

- **Määrittelyvaiheen luokkamalli** kuvailee sovellusalueen käsitteitä ja niiden suhteita
  - Muodostetaan usein *käsiteanalyysin* avulla
  - Käytetään myös nimitystä **kohdealueen luokkamalli** (engl. domain model)
  - Määrittelyvaiheen luokkamalliin saatetaan merkitä tärkeimmät attribuutit, *metodeja ei kuitenkaan merkitä*

Luokkakaaviot

mallinnuksen abstraktiotaso vaihtelee

- **Määrittelyvaiheen luokkamalli** kuvailee sovellusalueen käsitteitä ja niiden suhteita
  - Muodostetaan usein *käsiteanalyysin* avulla
  - Käytetään myös nimitystä **kohdealueen luokkamalli** (engl. domain model)
  - Määrittelyvaiheen luokkamalliin saatetaan merkitä tärkeimmät attribuutit, *metodeja ei kuitenkaan merkitä*
  - Ei toiminnallisuutta määrittelyvaiheen luokkakaavioon, ainoastaan olioiden pysyväislaatuiset yhteydet kannattaa merkitä

Luokkakaaviot

mallinnuksen abstraktiotaso vaihtelee

- **Määrittelyvaiheen luokkamalli** kuvailee sovellusalueen käsitteitä ja niiden suhteita
  - Muodostetaan usein *käsiteanalyysin* avulla
  - Käytetään myös nimitystä **kohdealueen luokkamalli** (engl. domain model)
  - Määrittelyvaiheen luokkamalliin saatetaan merkitä tärkeimmät attribuutit, *metodeja ei kuitenkaan merkitä*
  - Ei toiminnallisuutta määrittelyvaiheen luokkakaavioon, ainoastaan olioiden pysyväislaatuiset yhteydet kannattaa merkitä
- **Suunnittelutasolla** määrittelyvaiheen luokkamalli tarkentuu
  - Luokat ja yhteydet tarkentuvat
  - Mukaan tulee uusia teknisen tason luokkia

Luokkakaaviot

mallinnuksen abstraktiotaso vaihtelee

- **Määrittelyvaiheen luokkamalli** kuvailee sovellusalueen käsitteitä ja niiden suhteita
  - Muodostetaan usein *käsiteanalyysin* avulla
  - Käytetään myös nimitystä **kohdealueen luokkamalli** (engl. domain model)
  - Määrittelyvaiheen luokkamalliin saatetaan merkitä tärkeimmät attribuutit, *metodeja ei kuitenkaan merkitä*
  - Ei toiminnallisuutta määrittelyvaiheen luokkakaavioon, ainoastaan olioiden pysyväislaatuiset yhteydet kannattaa merkitä
- **Suunnittelutasolla** määrittelyvaiheen luokkamalli tarkentuu
  - Luokat ja yhteydet tarkentuvat
  - Mukaan tulee uusia teknisen tason luokkia
- **Suunnittelutason** luokkamalli muodostuu *oliosuunnittelun* myötä
  - Oliosuunnittelussa kannattaa noudattaa hyväksi havaittuja oliosuunnittelun periaatteita (mm. single responsibility...)
  - Luovuus, kokemus, tieto ja järki tärkeitä suunnittelussa

Luokkakaaviot

mallinnuksen abstraktiotaso vaihtelee

- Malleista tulee hieman erilaisia määrittely- ja suunnittelutasolla
- Eli se, minkälainen mallista tulee, riippuu mallinnuksen näkökulmasta
- Tämän takia **yleensä pelkkä kaavio ei riitä**: tarvitaan myös tekstiä (tai keskustelua), joka selvittää mallinnuksen taustaoletuksia
  - Tärkeintä, että kaikki asianosaiset tietävät taustaoletukset
  - Kun taustaoletukset tiedetään ja rajataan, löytyy myös helpommin ”oikea” tapa (tai joku oikeista tavoista) tehdä malli

Luokkakaaviot

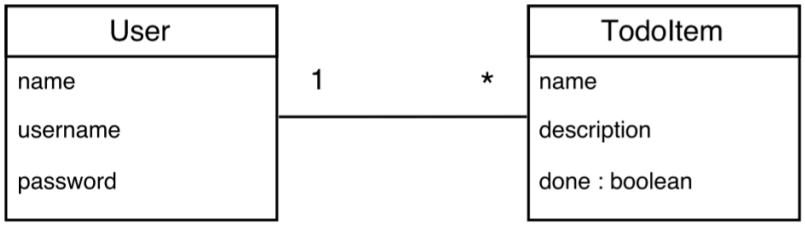
mallinnuksen abstraktiotaso vaihtelee

- Malleista tulee hieman erilaisia määrittely- ja suunnittelutasolla
- Eli se, minkälainen mallista tulee, riippuu mallinnuksen näkökulmasta
- Tämän takia **yleensä pelkkä kaavio ei riitä**: tarvitaan myös tekstiä (tai keskustelua), joka selvittää mallinnuksen taustaoletuksia
  - Tärkeintä, että kaikki asianosaiset tietävät taustaoletukset
  - Kun taustaoletukset tiedetään ja rajataan, löytyy myös helpommin "oikea" tapa (tai joku oikeista tavoista) tehdä malli
- Mallinnustapaan vaikuttaa myös mallintajan kokemus ja "orientaatio"
  - Kokenut ohjelmoija "näkee kaiken koodina" ja ajattelee väistämättä teknisesti myös abstraktissa mallinnuksessa
  - Tietokantaihminen taas näkee kaiken tietokannan tauluina jne.

Luokkakaaviot

Määrittely- vs suunnittelutaso

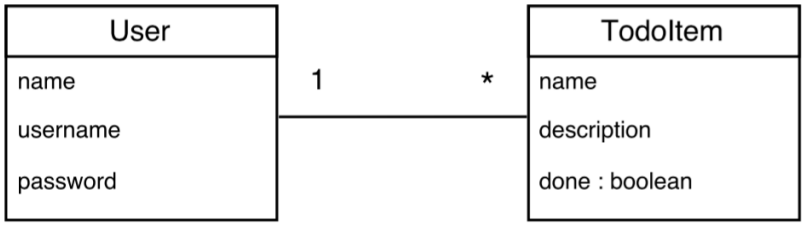
- Todo-sovelluksen määrittelyvaiheen luokkakaavio on yksinkertainen



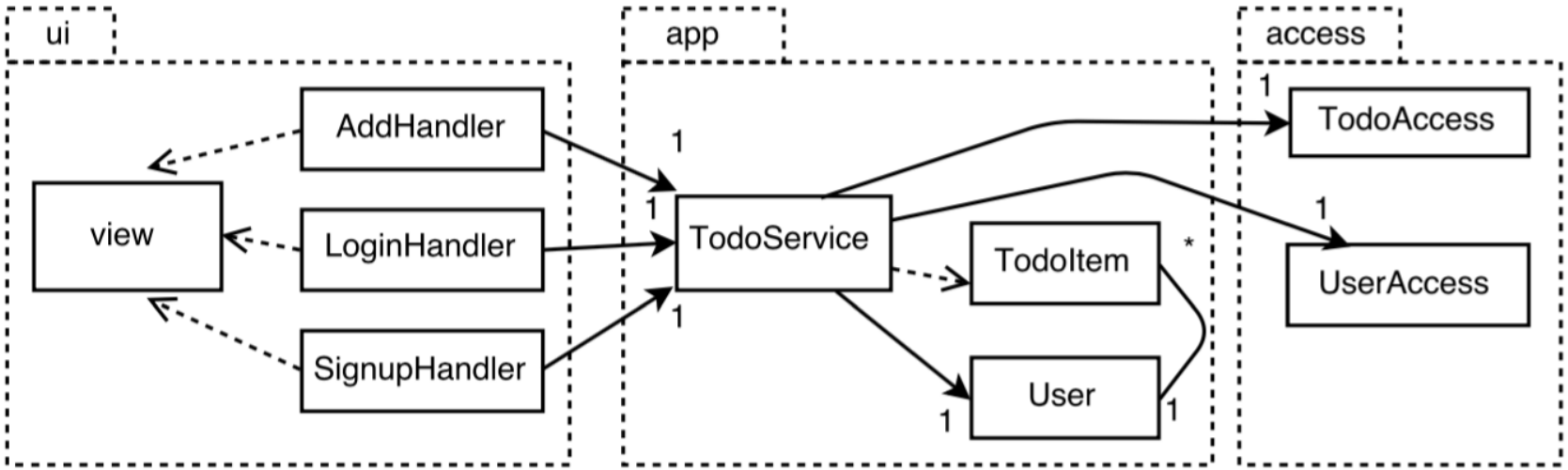
Luokkakaaviot

Määrittely- vs suunnittelutaso

- Todo-sovelluksen määrittelyvaiheen luokkakaavio on yksinkertainen



- oliosuunnittelun seurauksena päädyttiin suunnittelutason luokkakaavioon joka pitää sisällään määrittelyvaiheen luokat mutta sisältää myös paljon muuta



Luokkakaaviot

Normaali yhteys

- Erilaiset yhteystyypit ovat herättäneet ajoittain epäselvyyttä
- Yhteydellähän tarkoitetaan ”rakenteista” eli jollakin tavalla pitempiaikaista suhdetta kahden olion välillä
- Jos kahden luokan olioiden välillä on tällainen suhde, merkitään luokkakaavioon yhteys (ja yhteyteen liittyvät nimet, roolit, kytkentärajoitteet ym.)
- Toteutusläheisissä malleissa voidaan ajatella, että luokkien välillä on yhteys jos luokalla on olioviite tai viitteitä oliomuuttujana

Luokkakaaviot

Normaali yhteys

- Jokaisella Kurssi-oliolla on yhteys yhteen Henkilö-olioon, joka on Kurssin suhteen roolissa luennoija

```
public class Henkilo {
    ...
}

public class Kurssi {
    private Henkilo luennoija;
    ...
}
```

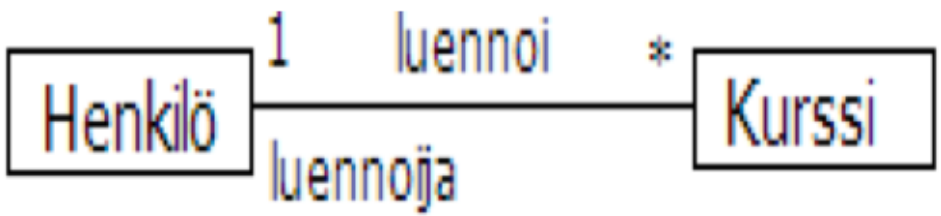
Luokkakaaviot

Normaali yhteys

- Jokaisella Kurssi-oliolla on yhteys yhteen Henkilö-olioon, joka on Kurssin suhteen roolissa luennoija

```
public class Henkilo {
    ...
}

public class Kurssi {
    private Henkilo luennoija;
    ...
}
```



Luokkakaaviot

Kompositio

- Normaali yhteys, eli viiva on varma valinta sillä se ei voi olla ”väärin”
- Usein määrittelyvaiheen luokkamalleissa käytetäänkin vain normaaleja yhteyksiä

Luokkakaaviot

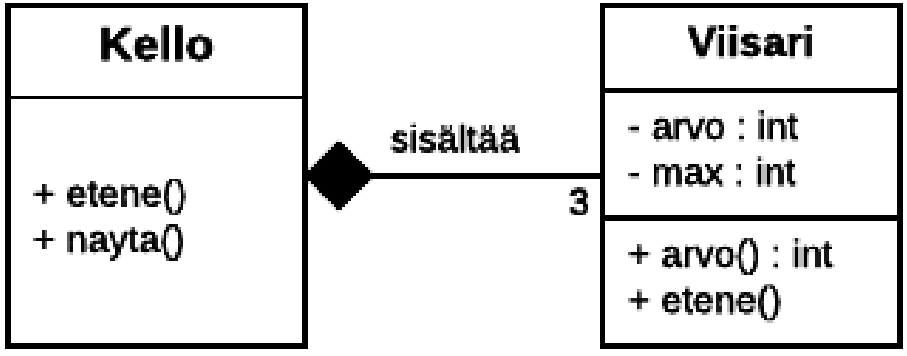
Kompositio

- Normaali yhteys, eli viiva on varma valinta sillä se ei voi olla ”väärin”
- Usein määrittelyvaiheen luokkamalleissa käytetäänkin vain normaaleja yhteyksiä
- Yhteyteen voidaan laittaa tarvittaessa navigointisuunta eli nuoli toiseen päähän, tällöin vain toinen pää tietää yhteydestä

Luokkakaaviot

Kompositio

- Normaali yhteys, eli viiva on varma valinta sillä se ei voi olla ”väärin”
- Usein määrittelyvaiheen luokkamalleissa käytetäänkin vain normaaleja yhteyksiä
- Yhteyteen voidaan laittaa tarvittaessa navigointisuunta eli nuoli toiseen päähän, tällöin vain toinen pää tietää yhteydestä
- Jos joku yhteyden osapuolista on *olemassaoloriippuvainen* yhteyden toisesta osapuolesta, voidaan käyttää *kompositioita* eli mustaa salmiakkia



Luokkakaaviot

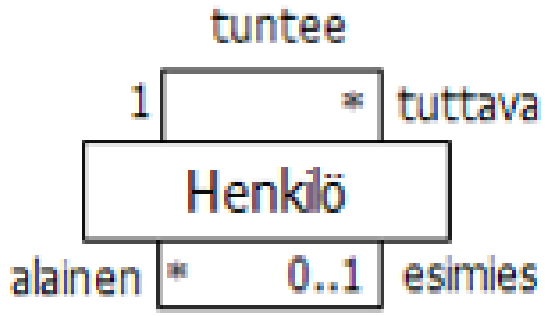
Rekursiivinen yhteys

- Saman luokan olioita yhdistävät yhteydet ovat joskus haasteellisia
- Henkilö tuntee useita henkilöitä
- Henkilöllä on mahdollisesti yksi esimies ja ehkä myös alaisia
- Yhteys *tuntee* liittää Henkilö-olioon useita Henkilö-olioita *roolissa tuttava*
- Henkilö-oliolla voi olla yhteys Henkilö-olioon, *roolissa esimies*

Luokkakaaviot

Rekursiivinen yhteys

- Saman luokan olioita yhdistävät yhteydet ovat joskus haasteellisia
- Henkilö tuntee useita henkilöitä
- Henkilöllä on mahdollisesti yksi esimies ja ehkä myös alaisia
- Yhteys *tuntee* liittää Henkilö-olioon useita Henkilö-olioita *roolissa tuttava*
- Henkilö-oliolla voi olla yhteys Henkilö-olioon, *roolissa esimies*
- Kuvan alempi yhteys siis sisältääkin kaksi yhteyttä: esimieheen ja alaisiin

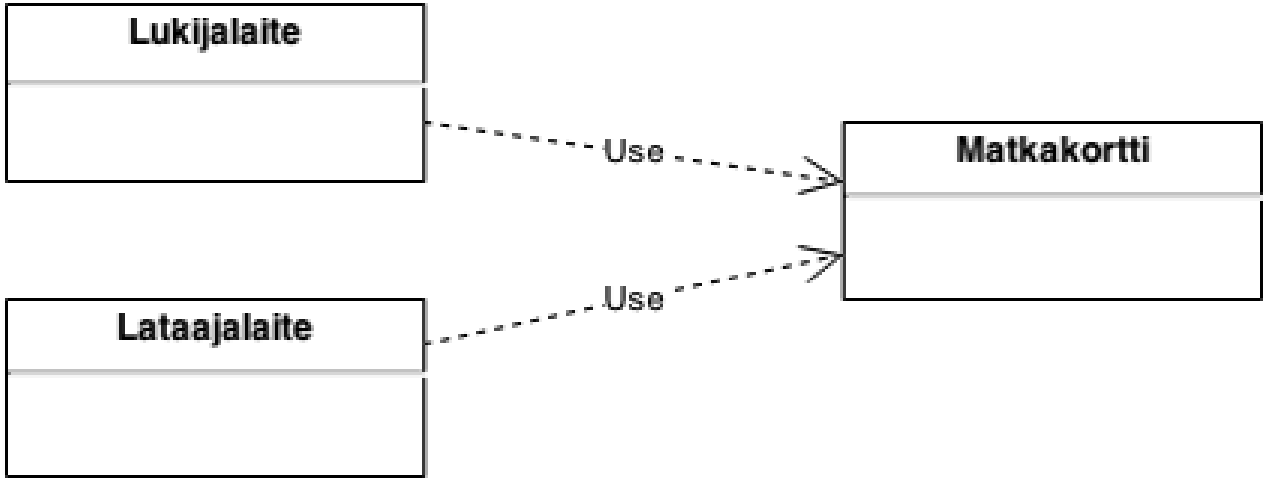


- Rekursiivisten yhteyksien kanssa on syytä käyttää roolinimiä

Luokkakaaviot

Yhteys vai riippuvuus?

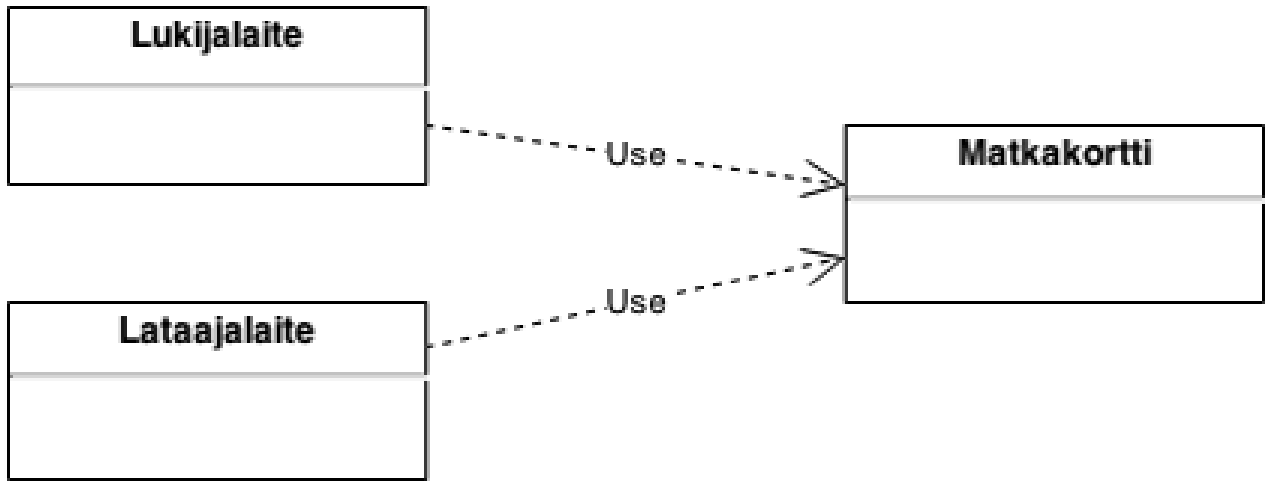
- **Riippuvuus on jotain normaalia yhteyttä heikompaa**
- Laskareista tuttujen Lataajalaitteen ja Lukijalaitteen suhden Matkakorttiin on riippuvuus
  - Esim. lataajalaite käyttää korttia vain ladatessan kortille lisää rahaa, laite ei muista kortteja pysyvästi



Luokkakaaviot

Yhteys vai riippuvuus?

- **Riippuvuus on jotain normaalia yhteyttä heikompaa**
- Laskareista tuttujen Lataajalaitteen ja Lukijalaitteen suhden Matkakorttiin on riippuvuus
  - Esim. lataajalaite käyttää korttia vain ladatessan kortille lisää rahaa, laite ei muista kortteja pysyvästi



- Riippuvuus ei siis ole rakenteinen vaan ajallisesti hetkellinen, esim. yhden metodikutsun ajan kestävä suhde

Luokkakaaviot

Yhteys vai riippuvuus?

- Riippuvuuden yhteyteen merkitään joskus riippuvuuden tyyppi, esim. edellisellä kalvolla «use», sillä lukija- ja lataajalaite käyttivät matkakorttia
- **Riippuvuuden yhteyteen ei merkitä osallistumisrajoitteita**
- **Riippuvuuden suunta on aina merkittävä!**
- Riippuvuuden ja yhteyden välinen valinta on joskus näkökulmakysymys

Luokkakaaviot

Yhteys vai riippuvuus?

- Riippuvuuden yhteyteen merkitään joskus riippuvuuden tyyppi, esim. edellisellä kalvolla «use», sillä lukija- ja lataajalaite käyttivät matkakorttia
- **Riippuvuuden yhteyteen ei merkitä osallistumisrajoitteita**
- **Riippuvuuden suunta on aina merkittävä!**
- Riippuvuuden ja yhteyden välinen valinta on joskus näkökulmakysymys
  - Miten kannattaisi esim. mallintaa Henkilön ja Vuokra-auton suhde?
  - Vaikka kyseessä voi olla lyhytaikainen suhde, lienee se järkevintä kuvata esim. autovuokraamon tietojärjestelmän kannalta normaalina yhteytenä

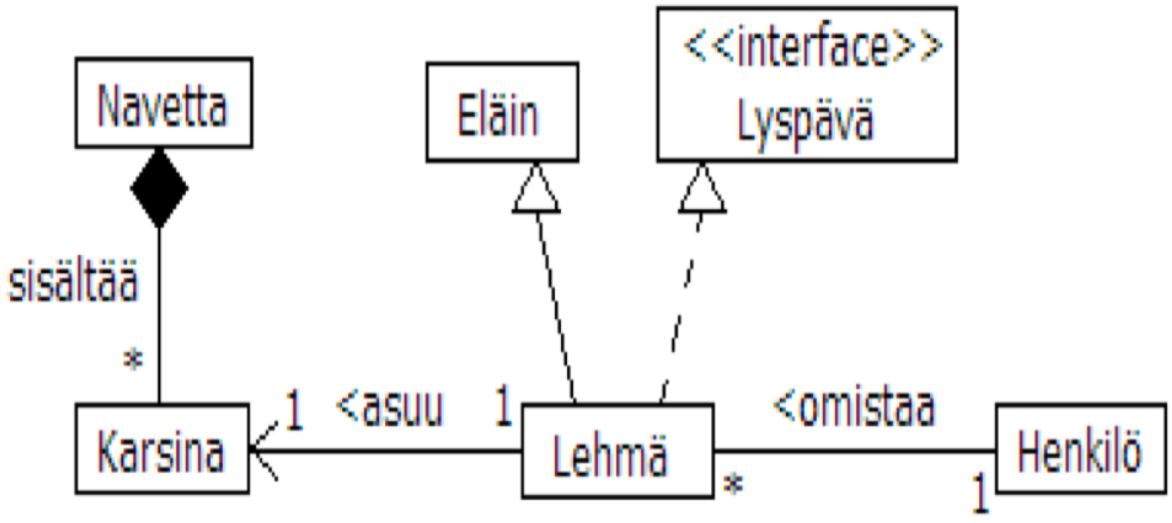
Luokkakaaviot

Yhteys vai riippuvuus?

- Riippuvuuden yhteyteen merkitään joskus riippuvuuden tyyppi, esim. edellisellä kalvolla «use», sillä lukija- ja lataajalaite käyttivät matkakorttia
- **Riippuvuuden yhteyteen ei merkitä osallistumisrajoitteita**
- **Riippuvuuden suunta on aina merkittävä!**
- Riippuvuuden ja yhteyden välinen valinta on joskus näkökulmakysymys
  - Miten kannattaisi esim. mallintaa Henkilön ja Vuokra-auton suhde?
  - Vaikka kyseessä voi olla lyhytaikainen suhde, lienee se järkevintä kuvata esim. autovuokraamon tietojärjestelmän kannalta normaalina yhteytenä
- Riippuvuuksia ei kannata välttämättä edes merkitä...
- Jotkut mallintajat eivät merkitse riippuvuuksia ollenkaan
- **Käytä siis riippuvuuksia hyvin harkiten**

Nuolenpäiden kanssa on oltava tarkkana

- Lehmä on Eläin-luokan aliluokka:
  - valkoinen kolmio yliluokan päässä, normaali viiva
- Lehmä toteuttaa Lypsävä-rajapinnan
  - valkoinen kolmio rajapinnan päässä, katkoviiva
- Navetta sisältää karsinoita, jotka olemassaoloriippuvaisia navetasta
  - musta salmiakki olemassaolon takaavan luokan päässä
- Lehmä asuu karsinassa, lehmä tuntee karsinansa, mutta karsina ei lehmää
  - yhteydessä navigointinuoli lehmästä karsinaan päin
- Lehmä ja omistaja (joka on Henkilö-olio) tuntevat toisensa
  - normaali yhteys ilman nuolia



Luokkakaaviot

Osallistumisrajoitteet

- Edellisen kalvon esimerkissä
  - Yhteen lehmäolioon *liittyy tasan yksi* Henkilöolio omistajana
  - Yhteen henkilöolioon taas voi *liittyä monta* Lehmäolioa yhteydessä omistaa
  - Mustan salmiakin yhteydessä ei tarvita osallistumisrajoitetta sillä salmiakki tarkoittaa käytännössä ykköstä, eli karsina sijaitsee yhdessä navetassa

Luokkakaaviot

Osallistumisrajoitteet

- Edellisen kalvon esimerkissä
  - Yhteen lehmäolioon *liittyy tasan yksi* Henkilöolio omistajana
  - Yhteen henkilöolioon taas voi *liittyä monta* Lehmäolioa yhteydessä omistaa
  - Mustan salmiakin yhteydessä ei tarvita osallistumisrajoitetta sillä salmiakki tarkoittaa käytännössä ykköstä, eli karsina sijaitsee yhdessä navetassa
- **Ei ole hyvä tapa jättää osallistumisrajoituksia merkitsemättä**, sillä ei ole selvää onko kyseessä unohdus vai tarkoitetaanko merkitsemättä jättämisellä sitä että yhteydessä olevien olioiden määrää ei osata määritellä

Luokkakaaviot

Osallistumisrajoitteet

- Yleisimmät osallistumisrajoitteet

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
5..9	Viidestä yhdeksään

Luokkakaaviot

Osallistumisrajoitteet

- Yleisimmät osallistumisrajoitteet

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
5..9	Viidestä yhdeksään

Riippuvuuksiin ja perintään ei koskaan merkitä osallistumisrajoitteita sillä kyseessä on luokkien ei olioiden välinen suhde

Sekvenssikaaviot

Sekvenssikaaviot

- Luokkakaavio ei kerro mitään olioiden välisestä vuorovaikutuksesta
- Vuorovaikutuksen kuvaamiseen paras väline on sekvenssikaavio
- Sekvenssikaavioiden peruskäyttö on melko suoraviivaista

Sekvenssikaaviot

- Luokkakaavio ei kerro mitään olioiden välisestä vuorovaikutuksesta
- Vuorovaikutuksen kuvaamiseen paras väline on sekvenssikaavio
- Sekvenssikaavioiden peruskäyttö on melko suoraviivaista
- **Kun teet sekvenssikaavioita ole erityisen tarkka, että:**
  - oliot syntyvät ”oikeassa kohtaa”, eli ylhäällä vain alusta asti olemassa olevat oliot
  - metodikutsu piirretään kutsujasta kutsuttavaan olioon
  - merkitset kaiken toiminnallisuuden kaavioon (myös metodikutsujen aiheuttamat metodikutsut)
  - parametrit ja paluuarvot on merkitty järkevällä tavalla

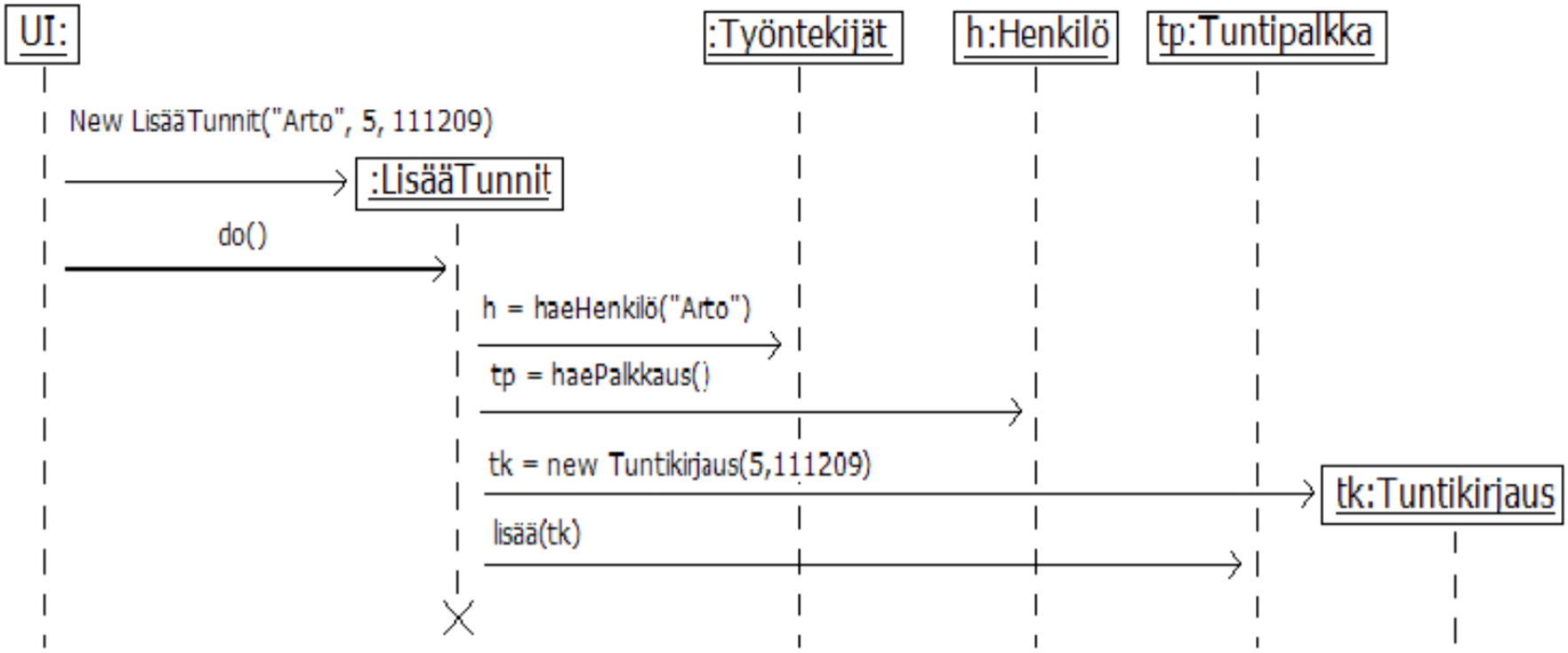
Sekvenssikaaviot

- Luokkakaavio ei kerro mitään olioiden välisestä vuorovaikutuksesta
- Vuorovaikutuksen kuvaamiseen paras väline on sekvenssikaavio
- Sekvenssikaavioiden peruskäyttö on melko suoraviivaista
- **Kun teet sekvenssikaavioita ole erityisen tarkka, että:**
  - oliot syntyvät "oikeassa kohtaa", eli ylhäällä vain alusta asti olemassa olevat oliot
  - metodikutsu piirretään kutsujasta kutsuttavaan olioon
  - merkitset kaiken toiminnallisuuden kaavioon (myös metodikutsujen aiheuttamat metodikutsut)
  - parametrit ja paluuarvot on merkitty järkevällä tavalla
- Sekvenssikaavioiden "vaikeudet" liittyvät toisto-, ehdollisuus- ja valinnaisuuslohkoihin
  - Näiden käyttökelpoisuus on rajallinen, ja niitä kannattaa käyttää harkiten, ei välttämättä ollenkaan

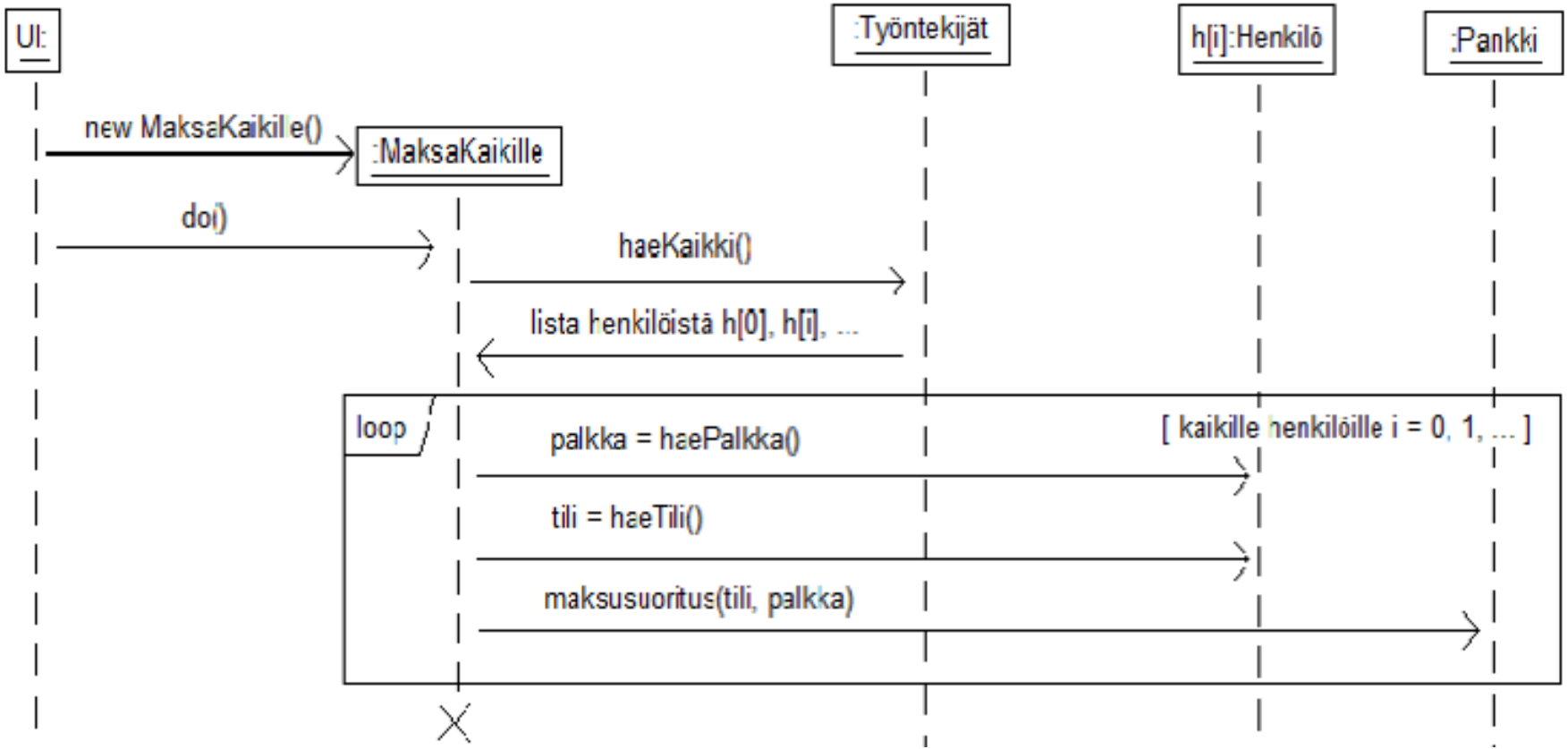
Sekvenssikaaviot

- Luokkakaavio ei kerro mitään olioiden välisestä vuorovaikutuksesta
- Vuorovaikutuksen kuvaamiseen paras väline on sekvenssikaavio
- Sekvenssikaavioiden peruskäyttö on melko suoraviivaista
- **Kun teet sekvenssikaavioita ole erityisen tarkka, että:**
  - oliot syntyvät "oikeassa kohtaa", eli ylhäällä vain alusta asti olemassa olevat oliot
  - metodikutsu piirretään kutsujasta kutsuttavaan olioon
  - merkitset kaiken toiminnallisuuden kaavioon (myös metodikutsujen aiheuttamat metodikutsut)
  - parametrit ja paluuarvot on merkitty järkevällä tavalla
- Sekvenssikaavioiden "vaikeudet" liittyvät toisto-, ehdollisuus- ja valinnaisuuslohkoihin
  - Näiden käyttökelpoisuus on rajallinen, ja niitä kannattaa käyttää harkiten, ei välttämättä ollenkaan
- Seuraavilla sivuilla 2 esimerkkiä kurssille kuulumattomasta osasta "*Oliosuunnitteluesimerkki: Yrityksen palkanlaskentajärjestelmä*"

- UI luo LisääTunnit-olion ja kutsuu sen *do*-metodia
- *do*:n suoritus kutsuu Työntekijät-olio metodia haeHenkilö
- Paluuarvona olevan Henkilö-olion *h* metodia *haePalkkaus* kutsutaan
- Luodaan uusi Tuntikirjaus *tk*, joka lisätään metodin *haePalkkaus* palauttamalle oliolle *tp* metodilla *lisää*
- Lopuksi LisääTunnit-luokan olio tuhoutuu



- UI luo luokan MaksaKaikille-olion ja kutsuu sen metodia *do*
- *do*:n suoritus kutsuu Työntekijät-olion metodia *haeKaikki*
  - Merkitään palautettua Henkilö-olioiden listaa h[0], h[1], ..., eli lyhyesti h[i], i=0,1,2, ...
- Loopissa jokaiselle Henkilö-oliolle *h[i]*
  - kutsutaan metodeja *haePalkka* ja *haeTili*
  - ja tehdään Pankki-oliolle metodikutsu *maksusuoritus*, jonka parametreina on henkilöltä kysytyt palkka ja tilinumero



Muut kaaviot

Muut kaaviot

- Näimme kurssilla myös
  - Kommuniaatiokaavioita
  - Tilakaavioita
  - Aktiviteettikaavioita
  - yhden komponenttikaavion
  - muutaman pakkauskaavion
- Täytyykö nämä osata kokeessa?

Muut kaaviot

- Näimme kurssilla myös
  - Kommuniaatiokaavioita
  - Tilakaavioita
  - Aktiviteettikaavioita
  - yhden komponenttikaavion
  - muutaman pakkauskaavion
- Täytyykö nämä osata kokeessa?
- Riittää jos osaa tulkita ym. tyyppisiä kaavioita
  - eli osaa selittää esim. yksinkertaisen tilakaavion mallintaman asian
  - ja tietää että kyseessä on tilakaavio
- Mitään tällä sivulla mainittua kaaviota ei tarvitse kokeessa osata piirtää

Loppu