Tampereen ammattikorkeakoulu

JAVAFX-sovelluksen toteutus

Antti Kaasalainen

OPINNÄYTETYÖ
Marraskuu 2021

Tietojenkäsittelyn tutkinto-ohjelma
Web-palvelut

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Web-palvelut

KAASALAINEN, ANTTI:
JAVAFX-sovelluksen toteutus

Opinnäytetyö 31 sivua
Marraskuu 2021

Tässä opinnäytetyössä esitellään JavaFX-käyttöliittymäkomponenttikirjastoa varteenotettavana teknologiana työpöytäsovelluksia toteutettaessa. JavaFX esitellään sen pääperiaatteiden kautta, mukaillen parhaaksi todettuja käytäntöjä, ja tutustutetaan lukija suurta suosiota saaneeseen MVC-arkkitehtuuriin.

Opinnäytetyössä kehitettiin esimerkkisovellus näyttämään konkreettisesti, miten JavaFX-sovellus rakentuu. Lukijalle esitellään UML:stä vaikutteita ottava suunnittelumalli ja tämän mallin mukainen toteutus. Sovellus on toteutettu niin, että se tuo esiin JavaFX-ydinominaisuuksia olematta kuitenkaan liian monimutkainen JavaFX-kehitykseen tutustuvalle.

Opinnäytetyö toimii oppaana JavaFX-sovelluskehityksestä kiinnostuneelle Java-ohjelmointia osaavalle henkilölle. Lukija osaa opinnäytetyöhön tutustuttuaan seurata parhaita käytäntöjä, sekä kehittää omaa JavaFX-osaamistaan.

Asiasanat: javafx, java, mvc

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Web Services

KAASALAINEN, ANTTI:
Implementation of JavaFX application

Bachelor's thesis 31 pages
November 2021

The purpose of this thesis is to introduce the JavaFX graphical user interface component library as a useful technology when implementing desktop applications. JavaFX is introduced with main focus in the MVC-architecture.

In addition to the theoretical part, an example application was created to show how to implement a JavaFX desktop application in practice. The example application shows the core features of JavaFX without being overly complicated to understand.

This thesis works as useful guide to anyone who is interested in implementing modern graphical user interface application with JavaFX. After reading this thesis, the reader should be able to implement a decent JavaFX application and is aware of JavaFX development best practices.

Key words: javafx, java, mvc

SISÄLLYS

ERITYISSANASTO tai LYHENTEET JA TERMIT (valitse jompikumpi)

API	Ohjelmointirajapinta
JavaFX	Javan käyttöliittymäkomponenttikirjasto
MVC	(Model, View, Controller) Malli, näkymä ja käsittelijä, sovellusarkkitehtuuri
UML	Unified modeling language, graafinen mallinnuskieli
XML	Extensible markup language, merkintäkielistandardi

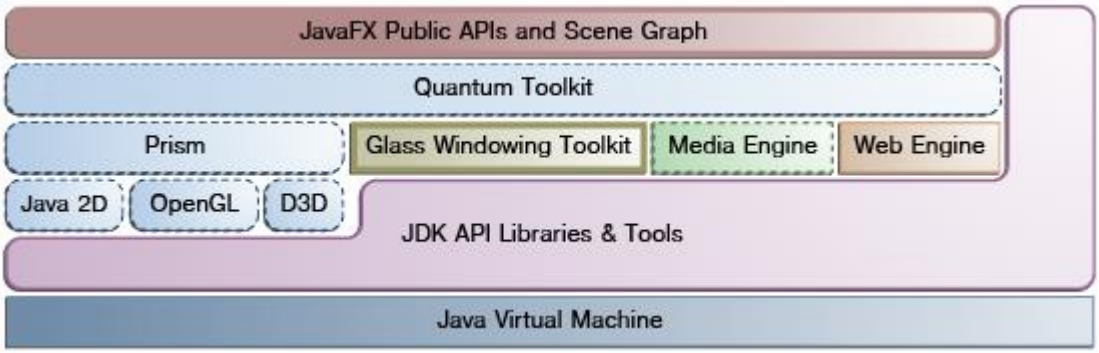
1 JOHDANTO

Tämä opinnäytetyö kokoaa yhteen kirjoittajan kokemuksia JavaFX-sovelluskehityksestä reilun puolenvuoden ajalta, tarkoituksena kerätä hajanainen tieto JavaFX:stä helposti lähestyttäväksi oppaaksi. Opinnäytetyö on kirjoitettu JavaFX-kehitystä opettelevan ja päivittäisessä työssään pääasiallisena teknologiana käyttävän sovellussuunnittelijan näkökulmasta. Vaikka JavaFX on menettänyt suosiotaan web-ohjelmoinnin vallatessa markkinoita, on se edelleen käyttökelpoinen ja monipuolinen teknologia toteutettaessa moderneja työpöytäsovelluksia.

Tämä opinnäytetyö haki alkuansa jo vuoden 2021 alusta, kun kirjoittaja löysi itsensä uudesta työtehtävästä JavaFX-sovelluskehityksen parista. Tarve opetella uusi teknologia ja samaan aikaan etsiä opinnäytetyön aihe törmäsivät sattumalta sopivasti yhteen. Puolen vuoden käytännön kokemuksen jälkeen opinnäytetyö alkoi muodostua vähitellen ja tarvittavien lähteiden analysointi helpottui merkittävästi jo opitun ansiosta. Tämä opinnäytetyö kävi prosessinsa aikana monta muokkausvaihetta ja se kirjoitettiin kerran jopa kokonaan uusiksi. Lopulta valmiina on opas, joka ehkä auttaa lukijaa löytämään apua oman JavaFX-sovelluskehityksensä synkillä ensiaskelilla.

2 JAVAFX

JavaFX on Javan kehittynein käyttöliittymäkomponenttikirjasto, joka on suunniteltu täyttämään modernien graafista käyttöliittymää hyödyntävien sovellusten tarpeet. Se on nostettu suositelluksi kirjastoksi graafisia käyttöliittymiä ohjelmoitaessa ja sitä pidetään vanhemman JFC:n (Java Foundation Classes) sisältämän swingin korvaajana. JavaFX-sovellus ohjelmoidaan pääosin valmiita käyttöliittymäkomponentteja hyväksi käyttäen. (Versterholm & Kyppö 2015, 412.) Sovelluksen voi toteuttaa täysin Javalla. Käyttöliittymän ulkoasun voi kuitenkin erottaa liiketoimintalogiikasta XML-pohjaista FXML-kieltä käyttäen. JavaFX-arkkitehtuuri on rakennettu useista komponenteista (Kuva 1), joihin sisältyy muun muassa grafiikkamoottorit ja Java virtuaalikone. Opinnäytetyössä tarkastellaan arkkitehtuurin ylintä tasoa, johon sijoittuu JavaFX:n julkinen ohjelmointirajapinta ja näkymäkaavio.



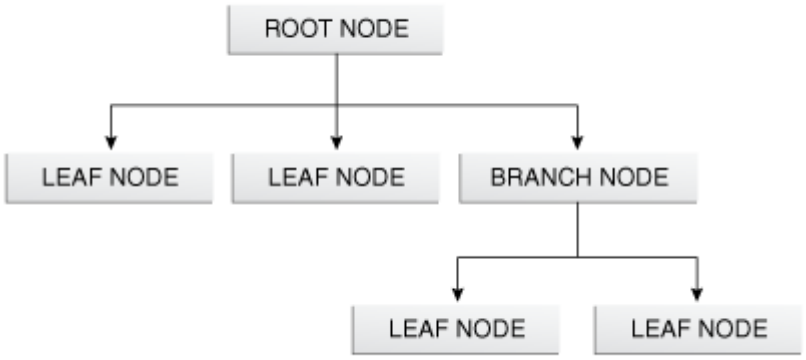
KUVA 1. JavaFX-arkkitehtuuri diagrammi (Castillo 2013).

2.1 SceneGraph-API ja näkymäkaavio

SceneGraph API on Javan ohjelmointirajapinta, joka vastaan käyttöliittymän renderöinnistä ja käyttäjän antaman syötteen vastaanottamisesta. Tämän ohjelmointirajapinnan hyödyntäminen käyttöliittymien ohjelmoinnissa vähentää huomattavasti tarvittavan koodin kirjoittamista, sillä se huolehtii automaattisesti kaikista käyttöliittymän piirtämiseen liittyvistä yksityiskohdista (Hommel, 2013).

JavaFX-sovelluksen käyttöliittymä muodostuu näkymäkaaviosta (scene graph). Se on joukko puumaisia tietorakenteita (Kuva 2), jotka sisältävät käyttöliittymän komponenttihierarkian. Jokainen komponentti on JavaFX käyttöliittymäkomponenttiluokka, joka periytyy suoraan tai välillisesti Node-yläluokasta. Jokaisesta käyttöliittymäkomponentista käytetään yleisesti nimitystä node tai tässä kirjoituksessa solmu.

Solmu on aina joko lehti- tai oksasolmu, sen mukaan sisältääkö komponentti muita käyttöliittymäkomponentteja vai ei. Oksasolmut ovat aina Parent-tyyppisiä. Näkymäkaaviossa on myös aina yksi solmu, joka ei sisälly toiseen solmuun. Tällaista solmua kutsutaan juureksi (root). (Hommel 2021.)



KUVA 2. Näkymäkaavio (Hommel, 2013)

2.2 JavaFX-komponentit

JavaFX-käyttöliittymäkomponenttikirjasto tarjoaa laajan kokoelman valmiita käyttöliittymäkomponentteja. Suurin osa komponenteista löytyy javafx.scene.control-pakkauksesta. Tyypillisesti komponentit ovat joko kontrolleja (controls) tai paneeleita (pane) ja sommitelmat (layout). (Vesterholm & Kyppö 2015, 420.) Kontrollit ovat selkeästi eroteltavia ja käyttöliittymässä olevia komponentteja, joihin liittyy yleisesti jokin toiminto. Kontrolleja ovat esimerkiksi painikkeet ja tekstikentät, joihin voi syöttää tietoa. Paneelit ja sommitelmat ovat taas kontrollien asettelua kuvaavia komponentteja. Ne määräävät, miten ja millä säännöillä kontrollit asettuvat käyttöliittymään. Paneelit voivat sisältää myös toisia paneeleita, jotka taas vuorostaan sisältävät toisia kontrolleja.

Kaikkia käyttöliittymäkomponentit sisältyvät jo aiemmin esiteltyyn näkymäkaavioon ja ovat joko lehti- tai oksasolmuja, sekä periytyvät kaikkien käyttöliittymäkomponenttien yläluokasta Node. Jokaiselle käyttöliittymäkomponentille voidaan asettaa oma id. Näkymäkaaviossa ei saa olla kahta samanlaista id:tä. Id on String-tyyppinen ja sitä voidaan käyttää ohjelmakoodissa viitatessa id:n osoittamaan komponenttiin.

2.3 CSS

CSS eli cascading style sheets, on tyylikieli, joka mahdollistaa käyttöliittymän ulkoasun muuttamisen. CSS-kieltä voidaan hyödyntää mihin tahansa käyttöliittymän solmuun. CSS:llä käyttöliittymän ulkoasua voidaan muuttaa dynaamisesti sovelluksen suorituksen aikana. Jokainen CSS-tyylin määrittely alkaa ”-fx-”-etuliitteellä.

3 FMXL

3.1 FXML

FXML on Oraclen luoma XML-pohjainen merkintäkieli, joka on Java-koodille vaihtoehtoinen tapa toteutettaessa graafista käyttöliittymää JavaFX-sovelluksissa (Oracle 01.2017). FXML-kielen hyödyntäminen sovelluksessa vähentää perinteisen koodin määrää ja tekee käyttöliittymästä selkeämmän ymmärtää ja helpoman ylläpitää. FXML-koodin rakenne on rinnastettavissa näkymäkaavion komponenttihierarkiaan, joten koodi itsessään antaa jo hyvän käsityksen siitä, miten käyttöliittymäkomponentit asettuvat suhteessa toisiinsa.

3.2 Elementit

FXML:ssä XML-elementti on joko luokan ilmentymä (class instance), luokan ilmentymän ominaisuus (property of class instance), staattinen ominaisuus (static property), määritelmälohko (define block) tai skriptilohko (block of script code). Kaikki elementit määritellään aloittavan ja lopettavan kulmasulkeen sisään (< />) tai vaihtoehtoisesti aloittavan ja lopettavan elementtimääritelmän sisään.

Luokan ilmentymää kuvaavan elementin nimi alkaa aina isolla alkukirjaimella (Oracle 01.2017). Kaikista JavaBeans-nimeämiskäytäntöä noudattelevista luokista voidaan luoda ilmentymä FXML:ssä. Luokan ilmentymän määrittelevä elementti voi olla esimerkiksi käyttöliittymässä näkyvää tekstiä ilmentävä Label-luokan elementti.

Luokan ilmentymän voi määritellä yksien kulmasulkeiden sisään tai vaihtoehtoisesti elementtiparina. Jos elementti on määritelty parina, voidaan sen sisässä hyödyntää ominaisuuselementtejä. Ominaisuuselementit sisältävät niiden vastaavalle luokalle oleellisia arvoja. Elementtejä voidaan käyttää joko arvojen asettamiseen tai vain niiden lukemiseen. Luokan ilmentymäelementille ”Label” voidaan asettaa teksti määrittelemällä sen sisään text-elementti.

Staattinen ominaisuus kirjoitetaan myös isolla alkukirjaimella, mutta erottuu tyypillisesti luokan ilmentymästä. Staattiset ominaisuudet ovat määritelty toisissa luokissa. Staattiset ominaisuudet alkavat luokan nimellä, jossa ne ovat määritelty.

3.3 Attribuutit

Attribuutit edustavat joko luokan ilmentymän ominaisuutta, staattista ominaisuutta tai tapahtumankäsittelijää. Attribuutit sijoitetaan luokkaelementin sisään. Ne toimivat vaihtoehtoisina toteutuksina muutamille elementeille, joilla määritellään luokan ominaisuuksia.

Attribuuttia voi käyttää tapahtumankäsittelijänä. Tällöin attribuuttiin voidaan merkitä tapahtuma, joka tulee suoritetuksi, kun tapahtumankäsittelijä saa suoritettavan tapahtuman. Esimerkiksi Luokkaelementille Button voidaan määritellä tapahtumankäsittelijäattribuutti onAction, jolle voidaan taas määritellä käsiteltävä tapahtuma. Attribuutin arvo voi olla metodin nimi, tai skriptikoodia. Metodin nimen asettamista käsittelijän arvoksi voidaan pitää suositeltavampana tapana, sillä se pitää FXML-koodin selkeämpänä. onAction-attribuutin arvo asetetaan kirjoittamalla ensin #-etuliite, jonka jälkeen metodin nimi. Metodin palautusarvon tyyppi on oltava void.

3.4 Käsittelijä

FXML-dokumentille voidaan asettaa vastaava käsittelijä, joka vastaa dokumentin sovelluslogiikasta. Käsittelijöissä toteutetaan toiminnot tapahtumankäsittelijöille. Käsittelijässä voidaan määritellä initialize-metodi, jota kutsutaan käsittelijän käynnistyessä. Metodilla voidaan alustaa tarpeelliset muuttujat, jotka ovat välttämättömiä.

Käsittelijässä voidaan tehdä viittauksia vastaavan FXML-dokumentin käyttöliittymäkomponentteihin. Luokasta javafx.fxml-FXML voidaan tuoda käyttöön FXML-notaatiot. Näiden notaatioiden avulla voidaan viitata FXML-dokumentin luokkaelementteihin.

3.5 FXMLLoader

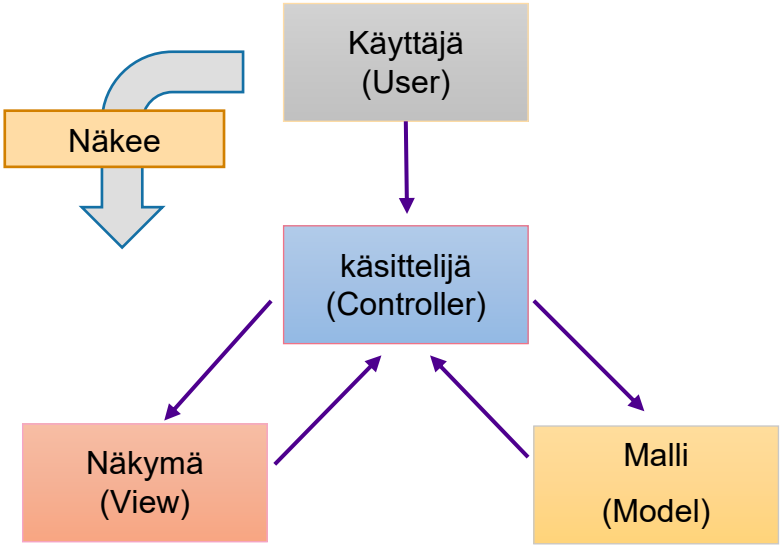
FXMLLoader on luokka, jonka avulla varsinainen FXML-dokumentin käyttöliittymäkomponenttihierarkia ladataan ja kootaan käyttöliittymäksi. FXMLLoader- luokan rakentajalle voidaan antaa parametreiksi sijainti ja lokalisoituja resursseja sisältävä resourceBundle. FXMLLoader-oliolta kutsutaan metodia load, joka palauttaa käyttöliittymäkomponentit juurielementin sisässä.

4 MVC-ARKKITEHTUURI

4.1 MVC yleisesti

MVC (Model, View, Controller) eli malli, näkymä sekä käsittelijä on norjalaisen tietojenkäsittelytieteilijä Trygve Reenskaug:n vuosina 1978-1979 kehittelemä sovellusarkkitehtuurimalli, jonka mukaan sovelluksen lähdekoodi jaetaan loogisiin kokonaisuuksiin. Arkkitehtuurin tarkoituksena on selkeyttää ohjelman toimintaa erottamalla sovelluksen käyttöliittymä ja toimintalogiikka toisistaan (Reenskaug 1979).

MVC on yleisesti käytetty arkkitehtuurimalli sovelluskehityksessä, kun tavoitteena on luoda graafista käyttöliittymää hyödyntävä sovellus. Tässä kappaleessa kuvataan MVC-arkkitehtuurin yleispiirteet työpöytäsovelluksen näkökulmasta tarkastelemalla jokaista aluetta omana kokonaisuutenaan. Tarkastelussa on Oraclen näkemys MVC-arkkitehtuurista (Kuva 3), jossa mallin ja näkymän suora yhteys on poistettu, ja kommunikointi näiden kahden komponentin välillä tapahtuu käsittelijän kautta (Eckstein, 2007).



KUVA 3. Kaavio MVC-arkkitehtuurista

4.2 Malli

Malli tai model edustaa ohjelman tietoa ja vastaa sen hakemisesta ja käsittelystä. Reenskaug mainitsee alkuperäisessä MVC:n kuvauksessa mallin olevan yksinkertaisimmillaan yksittäinen olio. Malli sisältää kaiken tiedon käsittelyyn vaadittavan logiikan, kuten tarvittavaan tietoon pohjautuvat algoritmit ja tiedon validointi.

Tiedon pääasiallisen käsittelyn lisäksi malli vastaa tietokantayhteyksistä. Malli suorittaa hakuja ja vientejä esimerkiksi tiedostoihin ja tietokantoihin. Käsitteeseen malli ei kuulu ainoastaan näitä operaatioita sisältävä luokka vaan myös domain-olioiksi kutsutut olioluokat, joilla mallinnetaan jotain reaalimaailmassa ilmenevää asiaa, joka halutaan esittää ohjelmassa. Malli kommunikoi käsittelijän (controller) kanssa, mutta ei koskaan suoraan näkymän kanssa.

4.3 Näkymä

Näkymä on nimensä mukaan sovelluksen alue, jossa esitetään visuaalisesti haluttu tieto. Näkymä koostuu käyttöliittymän määrittelevistä käyttöliittymäkomponenteista. JavaFX-työpöytäsovelluksessa näkymänä toimii luokka, jossa on määritelty kaikki näkymän komponentit tai FXML-tiedosto. Näkymä kommunikoi käsittelijän kanssa.

4.4 Käsittelijä

Mallin (model) ja näkymän (view) välissä toimii käsittelijä (controller). Käsittelijä vastaanottaa käyttäjän syötteen ja tekee haluttuja toimintoja tämän perusteella. Näkymä käyttää käyttäjän syötettä pyytääkseen mallilta tiettyjä toimintoja, kuten tietyn tiedoston haku. Malli tarjoaa logiikan, jonka käsittelijä siirtää näkymälle, joka taas esittää haetun ja käsitellyn tiedon käyttöliittymässä.

4.5 Tulevaisuus

MVC-arkkitehtuuri on opettelemisen arvoinen malli. Useita vuosikymmeniä ohjelmistokehityksessä käytetty arkkitehtuuri on edelleen suuressa käytössä. MVC-arkkitehtuuria sovelletaan nykyään myös web-kehityksessä. MVC ei ole kuitenkaan kovin helppo ymmärtää. Oppimista vaikeuttaa myös MVC-arkkitehtuurista olevat monet erilaiset tulkinnat, jotka eivät välttämättä edes toteuta MVC-periaatetta.

5 SUUNNITTELU

5.1 Suunnittelu

Kattavan suunnitelman tekeminen sovellukselle tekee varsinaisesta ohjelmointiprosessista helpompaa. Hyvin tehty suunnitelma selventää ohjelman rakennetta ja ratkaisee arkkitehtuurisia ongelmia. Suunnitelmassa voidaan hyödyntää UML-oliomallinnuskieltä (Unified Modeling Language) tekemällä esimerkiksi luokkakaavioita. Ohjelman toimintaa saadaan kuvattua käyttötapauskaavioilla ja sekvenssikaavioilla. Hyvä suunnitelma tekee ohjelmoinnista varsin mekaanista työtä (Vesterlholm & Kyppö, 2015, 48).

5.2 UML

UML-oliomallinnuskieli on laajasti käytetty standardoitu työkalu ohjelmistojen mallintamiseen. Se on kehitetty auttamaan ohjelmistokehittäjiä tarkentamaan, visualisoimaan, rakentamaan ja dokumentoimaan ohjelmiston artefakteja (UML. 2005). UML on tärkeässä osassa suunniteltaessa olioperustaisia ohjelmia.

5.3 Sovelluksen määrittely

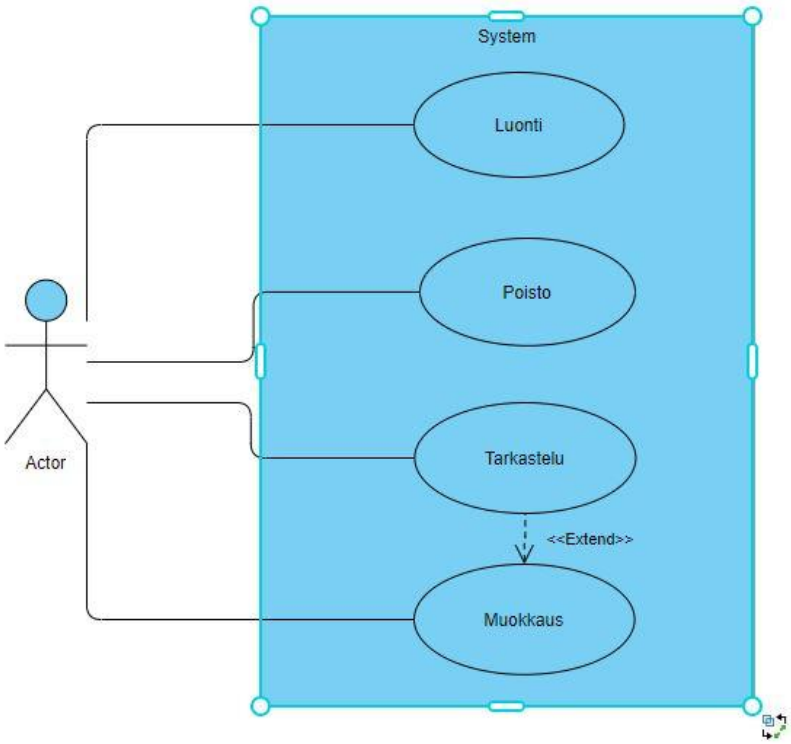
Tämän opinnäytetyön ohella tehdään esimerkkisovellus esittämään, kuinka JavaFX-sovellus rakentuu MVC-arkkitehtuurin mukaan. Esimerkkisovelluksena toimii muistiosovellus. Tässä sovellus esitellään yksinkertaisesti seuraavia kaavioita varten.

Ensimmäiseksi on syytä määritellä sovelluksen vaatimukset. Muistiosovelluksella tulee pystyä kirjoittamaan useita muistiinpanoja. Näitä muistiinpanoja tulee pystyä tallentamaan ja poistamaan tietokoneen keskusmuistista. Muistiinpanoja on pystyttävä lukemaan ja muokkaamaan sovelluksella. Sovellus on ikkunoitu sovellus, joka toimii Microsoft Windows-käyttöjärjestelmässä. Sovellus on toteutettu

kaiken kaikkiaan Java-kielellä, JavaFX-käyttöliittymäkomponenttikirjastoa käyttäen. Sovellusta käyttää yksi henkilö laitekohtaisesti.

5.4 Käyttötapauskaavio

Sovelluksen vaatimuksia voidaan tunnistaa ja kerätä käyttötapauskaavion avulla. Käyttötapauskaaviossa (Kuva 4) esitetään mahdollisia ja tyypillisiä käyttötapauksia, joita käyttäjän odotetaan tekevän sovelluksella. UML-käyttötapauskaaviota pidetään tehokkaana työkaluna määriteltäessä sovelluksen toimintaa. Kaavio luodaan tyypillisesti sovelluskehityksen varhaisessa vaiheessa.



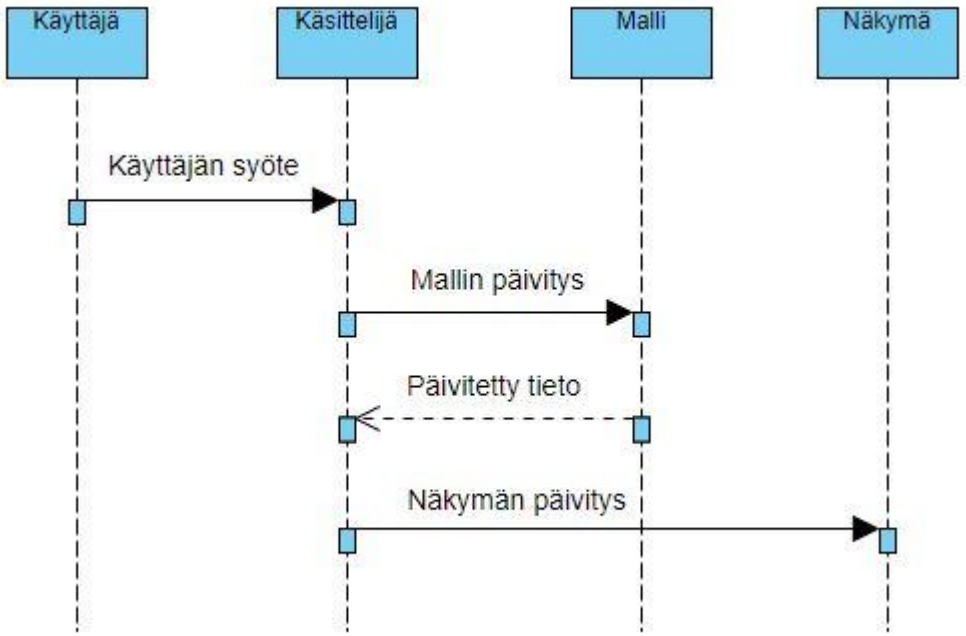
KUVA 4. Käyttötapauskaavio

Käyttötapauskaaviosta tehdään usein yksinkertainen, sillä siinä ei haluta esittää liian yksityiskohtaista toimintaa. Kaavio koostuu neljästä osasta: Näyttelijästä eli käyttäjästä, käyttötapauksesta, kommunikaatiolinkistä ja rajatusta systeemistä. Näyttelijä tai käyttäjä esitetään rajatun systeemin ulkopuolella tyypillisesti tikkuukkona. Käyttötapaus kuvataan soikiona muotona, jonka sisällä on kuvaus käyttötapauksesta. Käyttötapaukset on koottu rajatun systeemin sisään. Rajattu systeemi kuvaa jotain rajattavissa olevaa sovelluksen osaa tai mahdollisesti koko

sovellusta, johon käyttötapaukset liittyvät. Rajattu systeemi esitetään suorakaiteena. Kommunikaatiolinkit ovat viivoja, joilla kuvataan käyttäjän yhteyttä käyttötapauksiin.

5.5 Sekvenssikaavio

Sekvenssikaavioilla (Kuva 5) kuvataan käyttäjän ja sovelluksen välistä vuorovaikutusta. Sekvenssikaaviolla selvennetään käyttötapauskaaviossa määriteltyjä käyttötapauksia. Olio kuvataan kaaviossa suorakaiteena, josta piirtyy alaspäin olion elämää kuvaava viiva. Olioiden välille piirretään viivoja kuvaamaan viestiliikennettä olioiden välillä.



Kuva 5. Sekvenssikaavio

5.6 Luokkakaavio

UML-luokkakaaviossa sovelluksessa määritellyt luokat esitetään laatikkoina, joihin on kerätty yhteen niiden attribuutit ja metodit. Attribuuteista on mainittu kaaviossa attribuutin sisältävän arvon tyyppi, sekä attribuutin näkyvyys. Metodista taas mainitaan parametrien määrä ja tyypit, sekä palautusarvon tyyppi. Luokkien

välille piirretään viivoja, joilla kuvataan luokkien väliset suhteet. Luokkakaaviosta ei kannata tehdä liian yksityiskohtaista.

Luokkakaaviossa ylimpänä on luokan nimi. Tämän jälkeen tulee luokan attribuutit. Attribuuteista on tapana kirjoittaa niiden nimi ja tämän jälkeen kaksoispisteen jälkeen attribuutin tyyppi. Attribuuttien jälkeen kerrotaan, mitä metodeja luokka sisältää. Metodeista kerrotaan nimi ja niiden vaatimat parametrit. Metodien perään kirjoitetaan palautusarvon tyyppi. Attribuuttien sekä metodien eteen merkitään niiden näkyvyysmääre. Julkiset metodit ja attribuutit merkitään +-merkillä ja yksityiset käyttäen merkkiä -.

6 JAVAFX-SOVELLUS

6.1 Käynnistävä luokka

JavaFX-sovelluksen pääluokka periytetään Application-luokasta (Kuva 6). Application luokka toimii JavaFX-sovelluksen alkupisteenä. Sovellusta käynnistettäessä JavaFX-ajoympäristö muodostaa pääluokasta ilmentymän ja alustaa sovelluksen kutsuen Application-luokan init-metodia. Init-metodi voidaan ylikuormittaa (override) pääluokassa tekemään alustus ennen sovelluksen varsinaista käynnistämistä. Application-luokan init-metodi ei tee oletuksena mitään. Seuraavaksi kutsutaan Application-luokan abstraktia start-metodia, joka ylikuormitetaan pääluokassa.

```
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = BrowserController.createWindow();
        primaryStage.setTitle("Note");
        primaryStage.setScene(new Scene(root, width: 440, height: 640));
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Kuva 6. Sovelluksen pääluokka

6.2 Start-metodi, näyttämö ja kulissi

Start-metodilla on yksi Stage-tyyppinen parametri. Tätä parametria kutsutaan päänäyttämöksi (primary stage), sillä se on ensimmäinen näyttämö, joka aukeaa käyttäjän eteen. Näyttämö on korkean tason JavaFX-säiliö, joka voidaan nähdä työpöytäsovelluksen ikkunana. Start-metodin parametrinä oleva näyttämö, on

ympäristön muodostama. Muut sovelluksessa käytettävät näyttämöt muodostetaan sovelluksessa.

Start-metodissa määritellään ensimmäisen avautuvan ikkunan sisältö. Kaikki näkymäkaavion käyttöliittymäkomponentit sisältyvät start-metodissa määritellyn Scene-luokan muuttujaan. Scene-muuttuja määritellään antamalla sen rakentajalle näkymäkaavion juurisolmu (root node), sekä ikkunan leveys ja korkeus pikseleinä.

Näkymäkaavio on annettu kulissille, jolloin se voidaan asettaa päänäyttämölle setScene-metodilla. Näyttämölle voidaan myös asettaa otsikko setTitle-metodia käyttäen. Näyttämö on nyt valmis esitettäväksi show-metodilla.

6.3 Tapahtumapohjainen ohjelmointi

Graafisissa käyttöliittymissä hyödynnetään tapahtumapohjaista ohjelmointia (Vesterholm & Kyppö 2015, 412). Tällöin ohjelma odottaa käsittelyä vaativaa tapahtumaa perustilassa. Javassa tapahtumankäsittely on jo toteutettu valmiiksi, jolloin ohjelmoijan on ainoastaan määriteltävä tapahtumalle oikea toiminto.

Tapahtumat, esimerkiksi hiiren painallukset, tapahtuvat käyttöjärjestelmän ikkunointijärjestelmässä. Ikkunointijärjestelmä huomaa ja generoi tapahtumia, tietoina tapahtuman koordinaatit. Tämän jälkeen järjestelmä päättelee toimenpiteen tapahtumalle, etsii ikkunan, jossa tapahtuma tapahtui ja lisää sen kyseisen ikkunan tapahtumajonoon.

Käyttöliittymän komponenteille voidaan rekisteröidä tapahtumankäsittelijöitä. Esimerkiksi painikkeelle voidaan rekisteröidä käsittelijä, joka reagoi painikkeen painamiseen. Tapahtumankäsittelijä rekisteröidään käyttäen setOnAction-metodia tai vaihtoehtoisesti määrittelemällä FXML-tiedostossa käyttöliittymä-komponentille suoritettava metodi #onAction-muuttujaa käyttäen. setOnAction-metodille voidaan antaa suoritettava koodi funktionaalista ohjelmointia tukevaa lambdalausetta hyödyntämällä.

6.4 Ominaisuudet, pavut ja sitominen

Yksi JavaFX:n hienoisuuksiin kuuluva mekaniikka on luoda suoria yhteyksiä muuttujien välille käyttämällä sidontaa (binding). Tällä ominaisuudella JavaBeans-arkkitehtuuria mukailevilla ominaisuuksilla saadaan aikaiseksi automaattisesti tapahtuvia toimintoja muuttujien arvojen muuttuessa ohjelman suorituksen aikana. JavaFX-sovelluksessa sitomista voidaan hyödyntää esimerkiksi päivittämällä graafista käyttöliittymää vastaamaan sovelluksen datan muutoksia.

Sitominen voidaan toteuttaa kahdella eri tavalla, riippuen siitä miten sidottujen arvojen halutaan vaikuttavan toisiinsa. Yksisuuntaisessa sitomisessa (unidirectional binding) toisen sidotun arvon muuttaminen vaikuttaa sen pariin, mutta ei päinvastoin. Kaksisuuntaisessa sitomisessa (Bidirectional binding) kumman tahansa arvon muuttaminen vaikuttaa automaattisesti myös toiseen.

Sitomisessa käytetään käyttöliittymäkomponenttien ominaisuuksia (property). Jokaisella ominaisuudella on bind- ja bindBiDirectional-metodit. Yksisuuntaisessa sitomisessa käytetään bind-metodia. Metodin parametriksi annetaan käyttöliittymäkomponentit ominaisuus, joka halutaan sitoa. Kaksisuuntaisessa sitomisessa syntaksi on samanlainen, mutta bind-metodin sijasta käytetään bindBiDirectional-metodia.

JavaBeans on uudelleen käytettävien komponenttien luontiin tarkoitettu standardi (Vesterholm & Kyppö 2015, 134). Kaikki JavaBeans- ohjelmointiin liittyvä on koottu ohjelmointirajapinnaksi java.beans-paketin alle.

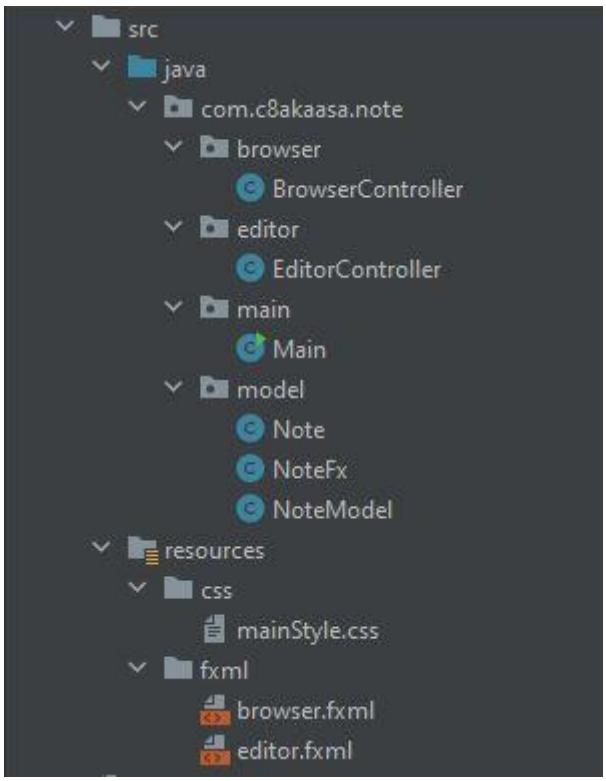
Bean tai papu on Java-luokka, joka noudattaa standardissa määriteltyä nimeämiskäytäntöä ja jolla on parametriton rakentaja. Pavulla on yksityisiä ominaisuuksia (property), joiden lukemiseen ja asettamiseen on standardin mukaiset julkiset metodit. Nimeämiskäytännöt tukevat koodin uudelleenkäyttöä, sillä standardin mukaisesti tehty koodi on helposti hyödynnettävissä. Tyypillisesti myös modernit Javaa tukevat integroidut kehitysympäristöt tunnistavat JavaBeans-nimeämiskäytännön.

JavaBeans-standardia noudattava asettajan metodin otsikko, esimerkiksi ominaisuudelle ”nimi” määritellään public void setNimi(String nimi). Lukevan metodin otsikko määritellään public String getNimi(). Jos lukevan metodin palautusarvo on tyypiltään boolean, on tapana metodin nimessä luettavan ominaisuuden eteen lisätä sana ”is” (on). Esimerkiksi luettaessa ravintolan aukiolon tilaa ilmoittavaa ominaisuutta open, metodin otsikko määriteltäisiin public boolean isOpen().

7 SOVELLUKSEN TOTEUTUS

7.1 Käynnistävä metodi

JavaFX-sovelluksen lähtöpiste on jo aiemmin mainittu Application-luokasta periytyvä luokka Main. Main toimii sovelluksen pääluokkana ja sisältää sovelluksen käynnistävän start-metodin. Start-metodissa määritellään toiminnot, joilla saadaan aikaiseksi ensimmäisen halutun ikkunan avautuminen käyttäjän nähtäville. Tässä sovelluksessa haluttu ikkuna on sovelluksen selainikkuna. Ainoana parametrinä olevalle Stage-luokan oliolle asetetaan selainnäkymän näkymäkaaviohierarkia Scene-luokan oliona, otsikko, sekä haluttu ikkunan koko pikseleinä. Tämän jälkeen voidaan luoda ikkuna kutsumalla Stage-olion show-metodia. Main-luokan toiminta on suoritettu. Sovelluksen rakenne (Kuva 6) noudattelee tyypillistä JavaFX-projektin rakennetta, jossa lähdekoodit on eritelty pakkauksiin omaan kansioonsa erilleen sovelluksen resursseista.



KUVA 7. Projektin rakenne

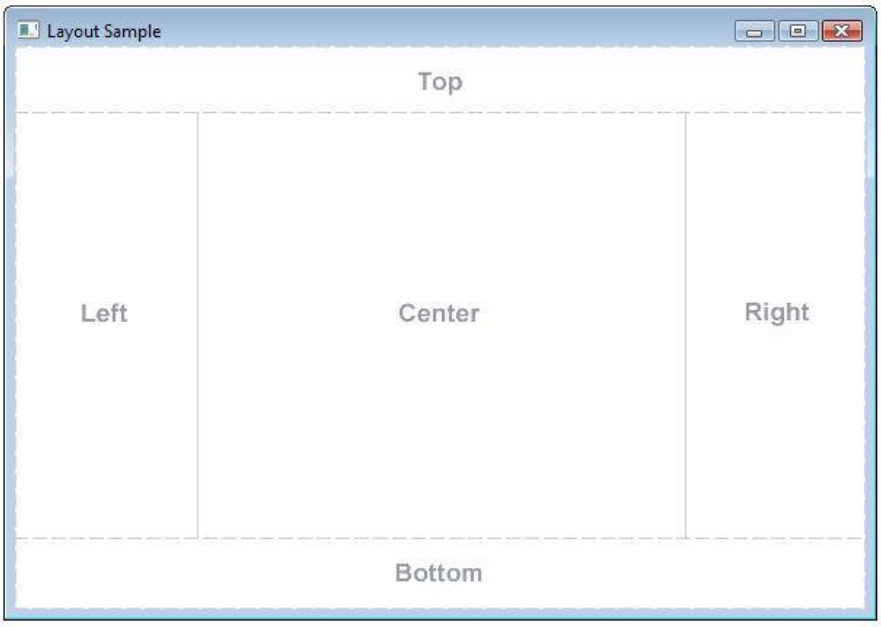
7.2 MVC-arkkitehtuuri

Muistiosovelluksessa hyödynnetään MVC-arkkitehtuuria, sillä kyseinen arkkitehtuuri nähdään yleisesti hyvänä käytäntönä JavaFX-sovellusta toteutettaessa. Toteutusjärjestys komponenteille on periaatteessa makuasia, mutta tätä sovellusta toteutettaessa todettiin, että helpointa on lähteä liikkeelle tekemällä näkymistä vähintään hahmotelmat, jolloin haluttujen ominaisuuksien kokeileminen on helppoa. Toteutusjärjestyksessä toisena komponenttina tulee malli. Mallin avulla saadaan selville, millaisia rakenteita käsittelijöille on tehtävä, jotta malli toimisi oikein. Viimeiseksi on selkeintä toteuttaa käsittelijä sitomaan näkymän ja mallin toiminta toisiinsa.

7.2.1 Näkymä

Sovellukseen toteutetaan kaksi näkymää. Näkymät toteutetaan parhaan käytännön mukaisesti FXML-kielellä.

Jokaisen näkymän määrittely alkaa juurielementillä, jonka sisään kaikki muut käyttöliittymäkomponentit määritellään. Juurielementti on Pane-luokasta periytyvä paneeli, tässä tapauksessa muistiosovelluksen näkyminen paneeleiksi on valittu BorderPane (Kuva 8).



Kuva 8. JavaFX BorderPane asettelu (Java Dokumentaatio)

Juurielementille voidaan asettaa id, sekä vastaava käsittelijä käyttämällä juurielementin muuttujia fx:id ja fx:controller. Jokaiselle käyttöliittymäkomponentille, johon halutaan päästä käsiksi käsittelijästä, on syytä määritellä id. Jos kuitenkin kyse on painikkeesta, id:n asettaminen ei ole välttämätöntä, sillä toiminto painikkeelle voidaan asettaa onAction- muuttujaan. Painike saa arvoksi etuliitteen #, jonka jälkeen tulee haluttu vastaavan ohjaimen metodi.

Näkymän ulkoasu tehdään CSS-kielellä omaan tiedostoonsa. CSS-tiedoston voi liittää näkymään monella tapaa, mutta selkeintä on liittää se osaksi FXML-dokumenttia stylesheets-elementin sisään.

7.2.2 Malli

Sovelluksen malli koostuu kahdesta osasta, tietoja käsittelevästä NoteModel-luokasta, sekä muistiinpanoa mallintavasta NoteFx-luokasta. NoteFx-luokan parina sovelluksessa on Note-luokka (Kuva 9), joka toimii tallennettavan olion muottina, sillä NoteFx-luokan olioita ei voi tallentaa sellaisenaan tiedostoon.

```
public class Note implements Serializable {

    private String title;
    private String note;
    private String ID;

    public Note(String title, String note, String ID) {
        this.title = title;
        this.note = note;
        this.ID = ID;
    }

    public String getTitle() { return title; }

    public String getNote() { return note; }

    public String getId() { return ID; }

    public void setTitle(String title) { this.title = title; }

    public void setNote(String note) { this.note = note; }

    public void setID(String id) { this.ID = id; }

    @Override
    public String toString() { return title + " " + note; }
}
```

Kuva 9. Note-luokka

Muistiinpanot mallinnetaan JavaBeans-käytäntöä mukailevan NoteFx-luokan avulla. NoteFx-luokassa on kaikki muistiinpanon tiedot StringProperty-tyyppisinä muuttujina. Property-muuttujiin on mahdollista hyödyntää edellä esiteltyä sidontaa, joten kaikki NoteFx-luokan oliokohtaiset muuttujat on hyvä määritellä haluttua tietotyyppiä vastaavaksi propertyksi.

NoteModel-luokka vastaa kaikista tietojen manipulointiin liittyvistä toimenpiteistä tässä sovelluksessa tallennuksesta, muokkaamisesta ja poistamisesta, sekä tietojen hakemisesta tiedostosta. Luokka pitää kirjaa olemassa olevista muistiinpanoista kahdessa eri listassa. Toinen listoista sisältää kaikki tallennetut muistiinpanot sellaisessa muodossa, että ne voidaan näyttää selainikkunan taulukossa. Toinen lista taas sisältää samat muistiinpanot, mutta Note-olioina, jolloin ne on mahdollista tallentaa tiedostoon. Yksi luokan attribuutti pitää kirjaa muistiinpanoista, jotka ovat avattuina editointi-ikkunoissa. Näin voidaan estää käyttäjää avaamasta samaa muistiinpanoa kahteen eri ikkunaan.

NoteModelissa on määritelty yksinkertainen oliontallennussysteemi. Käyttäjän luoma muistiinpano lisätään NoteFx-olioita sisältävään listaan. Listasta tehdään Note-olioita sisältävä lista ennen tallennusta. Tämä lista tallennetaan tiedostoon. Kun taas uusi olio halutaan tallentaa, haetaan ensin tiedostossa oleva lista, lisätään uusi olio listaan ja tallennetaan uusi lista.

7.2.3 Käsittelijä

Jokaiselle sovelluksen näkymälle on oma käsittelijänsä. Käsittelijä huolehtii sen oman näkymän ja ainoastaan tämän näkymän syötteen käsittelystä. Aiemmin mainitun mukaan käsittelijä toimii näkymän ja mallin välissä. Käsittelijässä on määritelty metodit kaikille näkymässä tarvittaville toiminnoille.

Muistiosovelluksen selaimen käsittelijä BrowserController toteuttaa Initializable-liittymän. Luokassa täytyy nyt ylikuormittaa initialize-metodi, jonka sisään voidaan kirjoittaa alustusta vaativaa koodia. Selainikkuna vaatii, että tallennetut muistiinpanot ovat näkyvillä taulukossa, kun sovellus avataan. Tämä tarkoittaa, että taulukon alustamiseen liittyvä logiikka kirjoitetaan initialize-metodiin. Taulukossa,

FXML-tiedostossa määritellyt sarakkeet (tässä tapauksessa muistiinpanon otsikkoon viittaava sarake), asetetaan vastaamaan NoteFx-olion kenttiä setCellValueFactory-metodilla. Taulukoon sidotaan NoteFx-oliot sisältävä ObservalbleList, jolloin tiedot päivittyvät tauluun, kun niitä lisätään, muokataan tai poistetaan mallissa.

8 LOPPUPÄÄTELMÄ

Vaikka JavaFX ei ole sovelluskehityksessä teknologiana kovinkaan suosittu nykypäivä, on se silti käyttökelpoinen työkalu moderneja työpöytäsovelluksia toteutettaessa. Kuitenkaan se ei näytä olevan opettelemisen arvoinen teknologia työmarkkinoiden näkökulmasta, mutta Javaa osaavalle sen oppiminen tarpeen vaatiessa ei ole suuri haaste.

JavaFX-sovelluskehityksessä nähdyt parhaat käytännöt osoittautuivat esimerkkisovelluksen toteutuksessa hyviksi. MVC-arkkitehtuurin noudattaminen jakaa sovelluksen helposti ymmärrettäviin ja ylläpidettäviin kokonaisuuksiin. MVC arkkitehtuurina on ansainnut paikkansa graafisten käyttöliittymien parissa, ei ainoastaan JavaFX-kehityksessä vaan myös web-kehityksessä.

FXML-kieli mukailee modernia tapaa luoda käyttöliittymä. XML-pohjaiset toteutukset ovat käytössä niin web-ohjelmoinnissa, kuin myös mobiilissa, Android-sovelluksissa. Käyttöliittymän ymmärtäminen yhdessä kehitysalueessa tukee toisen oppimista.

Java on ohjelmointikielenä saanut vahvan aseman ohjelmistokehityksessä. JavaFX kuitenkin poistetaan uusimmista Javan versiosta, mutta on edelleen ladattavissa tarvittaessa.

LÄHTEET

Eckstein, R. Java SE Application Design With MVC. Tekninen artikkeli. Luettu 14.06.2021. https://www.oracle.com/technical-resources/articles/javase/application-design-with-mvc.html

Hommel, S. Working with the JavaFX Scene Graph. Tekninen artikkeli. Luettu 20.06.2021 https://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm

Oracle. Introduction to FXML. Tekninen dokumentti. Luettu 18.09.2021. https://openjfx.io/javadoc/15/javafx.fxml/javafx/fxml/doc-files/introduction_to_fxml.html

Reenskaug, T. 1979. The original MVC reports https://core.ac.uk/download/pdf/30838874.pdf

UML. 2005 https://www.uml.org/what-is-uml.htm

Versterholm, M & Kyppö J. 2015. 9. painos. Java Ohjelmointi. Helsinki. Talentum Media Oy ja tekijät.