Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Sari Hildén

TAULUKKONÄKYMÄ MULTIFUNKTIONAALISEN DATAN ESITTÄMISEEN

Työn ohjaaja lehtori Erkki Hietalahti
Työn teettäjä DokuMentori Oy, valvojana insinööri (AMK) Erno Nieminen
Tampere 2007

Tietotekniikka
Ohjelmistotekniikka
Hildén, Sari Taulukkonäkymä multifunktionaalisen datan esittämiseen
Tutkintotyö 43 sivua
Työn ohjaaja lehtori Erkki Hietalahti
Työn teettäjä Insinööri (AMK) Erno Nieminen
Tammikuu 2007
Hakusanat JavaBean, ohjelmointi, Java, JTable

TIIVISTELMÄ

Insinöörityön tarkoituksena oli määritellä, suunnitella ja toteuttaa taulukkokomponentti DokuMentori Oy:lle ennalta tunnettuihin käyttökohteisiin ja tuleviin tarpeisiin. Suunnittelussa ja toteutuksessa tuli ottaa huomioon totetettavalle komponentille asetetut toiminnalliset ja laadulliset vaatimukset. Työssä keskityttiin komponentin määrittelyyn, suunnitteluun ja toteutukseen.

Työn tavoitteena oli laajentaa Java-ohjelmointikielen JTable-luokan toteuttamaa taulukkoa lisäämällä siihen toiminnallisuuksia, kuten esimerkiksi lajittelu, sekä parantamalla sen luettavuutta. Tavoitteena oli muuntautumiskykyinen taulukko, joka soveltuu käytettäväksi niin sisällön esittämiseen kuin muokkaamiseenkin ja kykenee esittämään aktiivikomponentteja, kuten painonappeja, valintaruutuja ja alasvetolistoja, soluissaan. Toteutettavaa taulukkokomponenttia tulee voida käyttää useassa ohjelmistotuotteessa monin eri tavoin. Komponentin tulee myös olla helposti laajennettavissa.

Työn toteutus tuli ajankohtaiseksi, koska useissa DokuMentori Oy:n DokuPort™-sovelluksen käyttöliittymissä taulukoilta kaivattiin parempaa luettavuutta sekä enemmän toiminnallisuuksia. Lisäksi monissa suunnitteilla olevissa käyttöliittymäkomponenteissa taulukoilta vaadittiin toiminnallisuuksia, joita Javan JTable-luokka ei tarjoa. Tarjolla ei myöskään ollut valmista Javalla toteutettua taulukkoa vapaalla lisenssillä, jota olisi voitu käyttää. Työn aloittamista kiirehdittiin kiusallisten piirto-ongelmien vuoksi, joita DokuPort™-sovelluksessa käytetyssä taulukossa tiedettiin olevan.

Suunnittelussa käytettiin laajalti hyväksi MVC-ratkaisumallia. Komponentti toteutettiin Java-ohjelmointikielellä Java-ajoaikaympäristön tarjoamia kehityskomponentteja hyväksi käyttäen sekä JavaBean-arkkitehtuurin mukaisesti.

Toteutusvaiheessa kyettiin usean toiminnallisuuden yhteydessä hyödyntämään monia opittuja algoritmeja, tietorakenteita ja säiliöiden käyttötapoja. Lisäksi koulutuksen myötä tutuksi tullut riskien hallinta mahdollisti aikataulujen tekemisen yllättävällä tarkkuudella.

Työn tulos on monipuolinen ja muuntautumiskykyinen taulukkokomponentti, jossa kaikki vaaditut ominaisuudet toimivat toivotulla tavalla. Komponenttia käytetään DokuMentori Oy:n DokuPort™-sovelluksessa useissa käyttöliittymäkomponenteissa. Komponentille on myös jatkokehitysideoita, joten sen kehitystä tullaan jatkamaan edelleen.

TAMPERE POLYTECHNIC
Comuter Systems Engineering
Software engineering
Hildén, Sari
Tableview for presenting multifunctional data
Engineering Thesis 43 pages
Thesis Supervisor lector Erkki Hietalahti
Commissioning Company DokuMentori Oy. Supervisor: Erno Nieminen (BSc)
January 2007
Keywords JavaBean, programming, Java, JTable

ABSTRACT

The goal of this bachelor’s thesis was to specify, design and implement table-component for joint stock company DokuMentori for known applications and future needs. Both functional and qualitative requirements were taken into account while designing and implementing this table-component. Batchelor's thesis focuses mainly on designing and implementation of TableView.

Thesis aims to improve and expand a table implemented by JTable-class found in Java language by adding functionalities like for example sorting and improving it's readability. Expected result was a versitile table which is suitable for presenting data and for editing it. Table-component had to be able to present active components, like push buttons, check boxes and drop-down lists, in it's cells. Implemented table-component was also expected to be easily extendable if needed and ofcourse easy to use.

A need for an extended table rose from the needs of various user interfaces in DokuMentori Oy's DokuPort™-application. In many cases requirements were related to better readability and additional functionalities. Also many planned user interfaces required tables with functionalities that Java's JTable-class doesn't offer. None of the free lisences tables that were accessable at the time filled the needs either. To begin this project was rushed because of embarrassing drawing problems known to accure in table that was used in DokuPort™-appilaction at the time.

The MVC design pattern was used widely in the component’s design. The component was programmed using the Java language and implementing some of the Java Development/Runtime Environment’s development components. The component was also programmed according to the JavaBean-architecture.

When implementing the TableView learned algorithms datastructures and container were widely used. Risk management skills were also found wery usefull and also made it possible to make time estimations with suprising accuracy.

The result of this thesis is a versatile and protean table-component in where all required features work in a desired way. Implemented component is used in DokuMentori Oy's DokuPort™-application in multiple user interfaces. There are also some ideas for continuation development and therefore the developement will be carried on in the future.

TERMIT JA LYHENTEET

MVC-malli Ohjelmistokehitysmalli, jossa toteutettavasta komponentista eritellään sisällön tietomalli (Model), fyysinen näkymä (View) sekä käyttäytyminen ja tapahtumat (Controller) omiksi yksiköikseen.

IDE Integroitu kehitysympäristö (Integrated Developement Environment)

Implementoija Tässä: ensisijaisesti ohjelmoija, joka käyttää Taulukkonäkymää osana omaa käyttöliittymäkomponenttiansa. Myös mahdollinen jatkokehittäjä.

Applet Javalla toteutettu ohjelmakokonaisuus, jota ajetaan internetselaimessa. Varsinaiset suoritustiedostot sijaitsevat palvelimella, mutta suoritettaessa ne latautuvat paikalliskoneen muistiin rasittaen suorituksen yhteydessä pelkästään paikalliskonetta.

JavaBean Sun Microsystemsin kehittämä arkkitehtuuri, jota toteuttava komponentti on käytettävissä kehitysympäristöissä Javan omien komponenttien tapaan.

renderer Tässä: piirtämistä eli renderöintiä hoitava luokka

editor Tässä: solussa esitettävä komponentti ja/tai luokka, kun käyttäjä muokka ko. solun sisältöä.

Tietotekniikka, Ohjelmistotekniikka
Sari Hildén

SISÄLLYSLUETTELO

1 JOHDANTO 1
2 ESITUTKIMUS 3
2.1 Kieli, kehitysympäristö ja arkkitehtuuri 3
2.1.1 JavaBean-arkkitehtuuri ja sen toteuttaminen 3
2.2 Käytetyt ohjelmistotuotannon menetelmät 4
2.3 Pääasiallinen käyttökohde - DokuPortTM 5
2.4 Vaatimukset 6
2.4.1 Käyttökohteiden vaatimukset 6
2.4.1.1 Perinteinen taulukko 6
2.4.1.2 Taulukko valintakomponenttina 8
2.4.1.3 Taulukko editorina 9
2.4.1.4 Toiminnallinen, dataa esittävä, taulukko 10
2.4.2 Implementoijan vaatimukset 11
3 SUUNNITTELU 13
3.1 Rakenne 13
3.1.1 Model – Tietomalli 13
3.1.2 View – Näkymä 14
3.1.3 Controller – Hallinta 16
3.1.4 Pääluokka ja omistussuhteet 17
3.2 Ongelmien arviointi 19
4 TOTEUTUS 21
4.1 Toiminnallisuuksien toteuttaminen 21
4.1.1 Aktiivikomponentit Taulukkonäkymään 21
4.1.2 Luettavuus paranee automaattisilla korostuksilla 23
4.1.3 Editoitavuuksien muokkaaminen 25
4.1.4 Lajittelun rakentaminen 26
4.1.5 Luettavuus paranee implementoijan tekemillä korostuksilla 31
4.1.6 Suodatusrivin automatisointi 32
4.1.7 Editoitujen rivien korostaminen käyttäjälle 33
4.2 Riskien toteutuminen ja ilmenneet ongelmat 35
4.3 Testaus 35
5 LOPPUTUOTTEEN ARVIOINTIA 37
5.1 Taulukkonäkymän käyttöönotto 37
5.2 Vaatimusten täyttyminen 37
5.3 Soveltuminen käyttökohteisiin 38
5.4 Jatkokehitysideoita 42
6 YHTEENVETO 42
LÄHDELUETTELO 43

KUVAT

Taulukkonäkymän model-osan rakenne ja toiminnallisuus 14
Taulukkonäkymän view-osan rakenne ja toiminnallisuus 15
Talukkonäkymän controller-osan rakenne ja toiminnallisuus 16
Rakenne ja luokkien omistussuhteet 18
Aktiivikomponenttien rendererit 23
Tietorakenteen funktio lajittelussa 29

Taulukkonäkymä valintakomponenttina....................................38
Automaattiset korostukset Taulukkonäkymässä....................................39
Tekstiin verrattavan sisällön lajittelu....................................39
Numerotiedon lajittelu....................................40
Suodatusrivi Taulukkonäkymässä....................................47
Suodatuksen tehokkuus Taulukkonäkymässä....................................41

1 JOHDANTO

Tässä dokumentissa keskitytään esittelemään DokuMentori Oy:n tarpeisiin toteutetun taulukkokomponentin eri työvaiheet. DokuMentori Oy on joulukuussa 1998 perustettu dokumentointituotantoon erikoistunut asiantuntijayritys, joka tarjoaa asiakkailleen dokumentoinnin sisällöntuotantoa, palveluiden tuotantoa ja ohjelmistotuotantoa /7/. DokuMentori Oy:n kehitysosasto tuottaa sekä räätälöityjä että tuotteistettuja ohjelmistoja.

Työn aloittamista kiirehdittiin, koska useissa käyttöliittymäkomponenteissa DokuMentori Oy:n tuottamassa DokuPort™-sovelluksessa taulukoilta vaadittiin toiminnallisuuksia ja parempaa luettavuutta kuin mitä Java-ohjelmointikielen taulukko JTable kykenee tarjoamaan. Myöskään mikään valmisratkaisu ei tarjonnut vaadittuja ominaisuuksia.

Tärkein vaatimus tälle Taulukkonäkymäksi ristitylle taulukkokomponentille oli, että sen tulisi kyetä esittämään mahdollisimman monenlaista sisältöä: sekä toiminnallista että ei-toiminnallista. Toiminnallisella sisällöllä tarkoitetaan tässä aktiivikomponentteja, kuten valintalistoja, valintaruutuja ja painikkeita. Ei-toiminnallisella puolestaan tarkoitetaan taulukoille tyypillistä tekstiin rinnastettavaa sisältöä.Komponentin tulisi olla myös helposti laajennettavissa tulevaisuuden tarpeita silmällä pitäen.

Työ käsittää esitutkimuksen, määrittelyn, suunnittelun ja toteutusvaiheen. Myös käyttöönotto ja testaus on osittain sisällytetty dokumenttiin. Dokumenttiin sisällytetyt ohjelmakoodilistaukset on erotettu muusta sisällöstä courier-fontilla ja niiden tarkoitus on kerrottu ennen listausta. Dokumentissa ei esitellä toteutettuja rajapintoja, ellei se ole ymmärrettävyyden kannalta välttämätöntä. Komponentin ohjelmointi on toteutettu Sun Microsystemsin Java-kielellä käyttäen JavaBean-tekniikkaa.

Taulukko 1 Kaavioissa käytetty itse kehitetty merkintätapa

Merkki	Selitys
	Suhde, jossa nuolen alkukohteen luokka toteuttaa osoitetun rajapinnan (implements)
	Suhde, jossa nuolen alkukohteen luokka perii osoitetun luokan tai rajapinnan (extends)
	Toiminnallisen kokonaisuuden yhteys, jossa luokka, johon toiminnallisuus yhdistetään, tulee toteuttamaan sen.
0...n	Suhde, jossa alkukohteen luokalla on olemassa muuttujana instanssi kohdeluokasta. Nuolen alla oleva määre määrittää, kuinka monta instanssia ko. kohdeluokasta voi olla olemassa. Esimerkiksi kuvan 0...n tarkoittaa, että instansseja on nolla tai enemmän.
	Suhde, jossa alkukohteen luokka käyttää instanssia osoitetusta luokasta
	Luokka eli määritykseltään public class
	Abstrakti luokka eli määritykseltään public abstract class
	Rajapinta eli määritykseltään public interface
	Sininen taustaväri kertoo, että luokka kuuluu Taulukkonäkymä-komponenttiin
	Valkoinen taustaväri kertoo, että luokka on osa Javan omia kirjastoja
	Ulospäin näkyvä pääluokka Taulukkonäkymä-komponentista, jossa myös kaikki rajapintametodit ovat
	Toiminnallisuuden kuvaus
	Rajapintametodi, jolla kuvassa (ellipsin alla) esitetty kontrolleri voidaan lisätä

2 ESITUTKIMUS

Työn alussa keskityttiin vaatimusten määrittelyyn ja pohdittiin, mitkä ominaisuudet olisi mahdollista toteuttaa ja mitkä eivät. Luvussa esitellään vaatimuksia eri näkökulmista ja käydään lyhyesti läpi kieleen ja kehitysympäristöön liittyvät seikat. Lisäksi tehdään tiivis katsaus menetelmiin, joita projektissa tullaan käyttämään.

2.1 Kieli, kehitysympäristö ja arkkitehtuuri

DokuPort™-sovellus, jossa Taulukkonäkymä-komponenttia tullaan pääasiallisesti käyttämään, on ohjelmoitu Javalla. Tämän vuoksi työn alkaessa olikin selvää, että ohjelmointikielen tulisi olla Java. Lisäksi kyseinen ohjelmisto on Applet, eli muu ohjelmointikieli kuin Java ei tulisi kysymykseenkään. Javan versioksi valittiin Sun Microsystemsin Java SE 5.0:n /6/ päivitys 6, koska se oli aloitushetkellä määritelty koko ohjelmiston minimivaatimukseksi.

DokuMentori Oy:ssä on jo jonkin aikaa käytetty IDE:nä NetBeans-ohjelmistoa Java-sovellusten ja komponenttien kehittämiseen. Tämä oli painavin syy, miksi NetBeans 5.0 valittiin kehitysympäristöksi. Tuttuuden lisäksi juuri tämän kehitysympäristön valintaa tuki vaatimus, että komponentin on oltava JavaBean.

2.1.1 JavaBean-arkkitehtuuri ja sen toteuttaminen

JavaBean-arkkitehtuuri on Javan komponenttimalli, joka mahdollistaa Javan komponenttikehitysmallin. Tämän kehitysmallin avulla voidaan rakentaa ohjelmia komponenttiolioista, joita voi asettaa paikalleen joko graafisesti tai ohjelmallisesti. JavaBean-arkkitehtuuri tarkoittaa ohjelmistoteknisesti monia asioita, joita ovat muun muassa:

- tuki visuaaliselle ohjelmoinnille

- yhteensopivuus muiden käyttöliittymäkomponenttien kanssa
- olio-ohjelmoinnin periaatteiden mukaisesti luotu ohjelmakoodi
- tietoturvariskien hallinta
- olion yksilöitävyys sekä visuaalisesti että ohjelmallisesti
- vuorovaikutteisuus eli yhteensopivuus Javan tapahtumankäsittelymallin kanssa

/1/

Vaatimukset JavaBean-arkkitehtuuria noudattavalle komponentille ovat:

- pääluokassa oltava oletusrakentaja, eli rakentaja, joka ei ota yhtään parametria
- pääluokan periytyminen graafisesta säiliöluokasta, kuten esimerkiksi JPanel-luokasta

Tällainen komponentti voidaan esimerkiksi NetBeans IDE:ssä lisätä graafisten komponenttien joukkoon. Tämän jälkeen komponentti voidaan lisätä käyttöliittymiin raahaamalla se suunnittelunäkymässä haluttuun osaan suunniteltavaa käyttöliittymää.

2.2 Käytetyt ohjelmistotuotannon menetelmät

DokuMentori Oy:ssä työskentelee ohjelmistokehityksen parissa muutama työntekijä. Tilanne on ollut DokuMentori Oy:n ohjelmistokehityksen historiassa aina tältä osin sama. Tämän vuoksi ei ole koettu tarpeelliseksi dokumentoida käytössä olevia ohjelmistotuotannon metelmiä.

DokuMentori Oy:ssä on vakiintuneitä käytäntöjä ohjelmistotuotannossa, joita tässäkin projektissa tullaan noudattamaan. Näitä toimintatapoja ovat esimerkiksi jokaisesta kehitysprojektista selkeästi löydettävät ohjelmistokehityksen vaiheet: märittely-, suunnittelu-, toteutus- ja testausvaihe. Toteutuksen iteratiivisuus on myös yksi vakiintunut käytäntö, joka juontaa juurensa sekä tiimin pienuudesta että erillisen testaajan puutteesta. Yrityksessä suositaan iteratiivista kehitystä, koska sen on todettu parantavan kehitettävien ohjelmistojen ja komponenttien laatua sekä nopeuttavan itse kehitystyötä. Yleensä kehitys- ja suunniteluvastuu ovat kussakin projektissa yhdellä kehittäjällä märittelyvastuun jakautuessa koko tiimille.

Käytettävät ohjelmistotuotannon menetelmät eivät suoranaisesti vastaa ketterän ohjelmistotuotannon määritelmiä vaikka niissä on paljon yhteneviä tekijöitä. Tämä johtuu siitä, että DokuMentori Oy:ssä on kehittäjän päätettävissä, mitä menetelmiä projektissa tullaan käyttämään. Tässä projektissa tullaan käyttämään rinnakkain sekä vesiputousmallia että iteratiivista kehitystä tilanteesta riippuen. Projektissa tullaan myös huomioimaan riskien hallinta, koska toteutettava komponentti on laaja ja koska komponentti tulee laajentamaan luokka, jonka sisäinen toiminnallisuus on kehittäjille tuntematonta. Riskien hallinnalla pyritään ensisijaisesti vähentämään projektiin kuluvaa aikaa sekä tekemään aikatauluista luotettavampia.

2.3 Pääasiallinen käyttökohde - DokuPort™

DokuMentori Oy:n DokuPort™ on ensisijaisesti rakenteellisen dokumentaation hallintajärjestelmä /2/. DokuPort™-sovelluksella voi esimerkiksi luoda kirjan lukuineen. Kutakin kirjan lukua edustaa moduuli, jolle voi luoda sisältöä erikseen. Sisällön luomisen sykli etenee siten, että moduuli kuitataan ulos sisällöntuotantoon, jonka jälkeen siihen voidaan lisätä rakenteellista sisältöä halutulla ohjelmalla. Tämän jälkeen moduuli kuitataan takaisin sisälle sovellukseen. Sisään kuitattaessa moduulista luodaan uusi versio ja kunkin version tiedosto tallennetaan. DokuPort™-sovellus mahdollistaa sisällön uudelleenkäytön, koska samaa moduulia voidaan käyttää useammassa kirjassa.

Taulukkonäkymää tullaan käyttämään DokuPort™-sovelluksessa erilaisen tiedon, kuten esimerkiksi hakutulosten tai käyttäjien esittämiseen. Taulukkonäkymän odotetaan tekevän DokuPort™-sovelluksesta näyttävämmän sekä toimivamman monilta osin.

2.4 Vaatimukset

Johdannossa esiteltiinkin jo Taulukkonäkymän tärkein vaatimus: kyky esittää mahdollisimman monimuotoista sisältöä. Tämä vaatimus toi useita lisävaatimuksia niin rajapinnan kuin toiminnallisuuksienkin osalta. Lisäksi suunnittelun edetessä tuli esiin muitakin vaatimuksia. Edellä on esitelty kaikki Taulukkonäkymän vaatimukset sekä käyttökohteiden että implementoijan näkökulmasta.

2.4.1 Käyttökohteiden vaatimukset

Käyttökohteiden vaatimukset liittyvät tiiviisti toiminnallisuuksiin ja siihen, kuinka esitettävän sisällön tulisi näkyä. Alla on esitelty vaatimukset käyttökohteittain.

2.4.1.1 Perinteinen taulukko

Perinteisellä taulukolla tarkoitetaan tässä taulukkoa, joka sisältää pelkästään tekstiin verrattavaa sisältöä ja joka on tarkoitettu pelkästään tiedon esittämiseen. Tämän käyttökohteen alle listatut vaatimukset ovat olemassa tietenkin kaikille käyttökohteille, mutta vaatimusten luonne kohdistaa ne erityisesti perinteiselle taulukolle.

Vaatimus 1: Automaattinen rivien korostaminen
Yksi olennaisimmista vaatimuksista on, että Taulukkonäkymän tulee voida esittää perinteistä taulukkoon tarkoitettua sisältöä. Tavallinen Javan JTable kyllä pystyy siihen, mutta lisävaatimuksena tulee mahdollisuus asettaa Taulukkonäkymän joka toinen rivi automaattisesti maalatuksi, mikä lisäisi luettavuutta huomattavasti.

Vaatimus 2: Solujen, rivien ja sarakkeiden korostaminen

Taulukon ollessa iso on luettavuus usein melko huono. Joissain tilanteissa on tarpeen korostaa jotain solua syystä tai toisesta. Ratkaisuksi kaavailtiin mahdollisuutta vaihtaa solujen väriä halutuksi. Tästä tulikin yksi Taulukkonäkymän vaatimuksista sillä lisäyksellä, että yksittäisten solujen lisäksi värejä pitäisi voida asettaa riveille ja sarakkeillekin kuitenkin siten, että se on erotettu automaattisesta rivien korostamisesta (ks: Vaatimus 1).

Vaatimus 3: Editoitavuuden hallinta

Oleellinen osa sisällön esittämistä taulukoissa on, että solujen editoitavuutta voidaan muuttaa halutuksi. Käytettäessä Javan DefaultTableModel-luokkaa (tietomalli) kaikki taulukon solut ovat editoitavissa, mikä ei välttämättä ole joka tilanteessa toivottua. Yhdeksi vaatimukseksi kirjattiinkin mahdollisuus muuttaa koko Taulukkonäkymän, yksittäisen rivin, sarakkeen tai jopa solun editoitavuus halutuksi.

Vaatimus 4: Drag 'n Drop

Jotta Taulukkonäkymä olisi mahdollisimman monikäyttöinen ja sitä voitaisiin hyödyntää mahdollisimman hyvin sovelluksessa, jossa sitä pääasiassa tullaan käyttämään, päätettiin siihen lisätä Drag 'n Drop (DND) eli ns. ”vedä ja pudota”-toiminnallisuus. Tämä ominaisuus haluttiin myös siksi, että sovelluksessa on myös toinen komponentti, joka tukee DND:tä. Tämä toinen komponentti eli TreeView/3/ laajentaa Javan JTree-komponenttia. Lisäämällä DND toteutettavaan taulukkokomponenttiin, sitä voitaisiin hyödyntää täydessä mittakaavassa.

Käytännössä vaatimukseksi kirjattiin, että TreeView-komponenttiin tulee voida raahata rivejä Taulukkonäkymästä. Raahattavalta tiedolta vaadittiin ehdotonta yksiselitteisyyttä. Tämä tarkoittaa siis sitä, että DND:ssä siirtyvän tiedon täytyy voida olla tunnistettavissa (esim. onko kyseessä käyttäjä vai onko kyse sisällöstä, jonka tyyppiä ei ole määritelty).

DND:ssä tässä kyseisessä sovelluksessa tulee olemaan niin paljon erityispiirteitä ja erikoisuuksia, jotka rajoittuvat pelkästään tähän sovellukseen, että ominaisuuden

spesifisyyden ja laajuuden vuoksi sen suunnittelu- ja toteutusvaiheen esittely jätetään tämän dokumentin ulkopuolelle. Myöskään suoranaisesti DND:hen liittyviä luokkia ei esitellä.

2.4.1.2 Taulukko valintakomponenttina

Valintakomponentin taulukosta tekee se, että esitettävä sisältö ei varsinaisesti listaa mitään, vaan se on puhtaasti tarkoitettu esittämään ja/tai asettamaan vaihtoehtoja eri asioille. Hyvänä esimerkkinä taulukosta valintakomponenttina olisi esimerkiksi asetusten hallinta, jossa on tyypillisesti asetuksen nimi ja vieressä kenttä, johon arvo asetetaan tai komponentti, josta se valitaan. Vaatimukset, jotka tämän käyttökohteen alle on listattu, koskevat yhtä lailla kaikkia käyttökohteita, mutta erityisesti aktiivikomponentteja, kuten esimerkiksi tekstikenttiä tai alasvetolistoja, sisältävää taulukkoa.

Vaatimus 5: Toiminnalliset komponentit

Vaatimus toiminnallisten komponenttien esittämisestä asetti aivan erilaisia vaatimuksia. Alustavasti Taulukkonäkymältä vaaditaan kykyä esittää viittä eri komponenttia tekstiin rinnastettavan sisällön lisäksi: JLabel, JTextField, JCheckBox, JComboBox ja JButton. Eikä pelkästään esittäminen riittäisi, vaan näiden komponenttien tulisi myös pystyä toimimaan Taulukkonäkymän sisällä samoin kuin ne toimisivat missä tahansa käyttöliittymässä. Java SE5.0 esittelee kyllä JTable-luokan esimerkeissä mahdollisuuden esittää mitä tahansa swing-komponenttia taulukon soluissa. Esimerkkejä kokeiltaessa kävi kuitenkin ilmi, että komponenttien esittämiseen liittyi valtavasti piirto-ongelmia (ks: Vaatimus 6) eikä valmista ratkaisua voitu näin käyttää.

Vaatimus 6: Aktiivikomponenttien piirto-ongelmien poisto

Niillä paikoilla, joihin Taulukkonäkymää tullaan käyttämään on tähän saakka käytetty toista JTable-luokan ilmentymää. Tätä kautta on komponenttien toiminnasta Javan JTable-luokan ilmentymässä saatu paljon tietoa. Tätä kautta tuli myös esille paljon ongelmia komponenttien toiminnassa JTable-luokassa. Alla on

esitelty ongelmat, jotka Taulukkonäkymän tulee korjata. Kaikki alla esitellyt ongelmat ovat käyttäjän kannalta erityisen harmillisia ja antavat ohjelmistosta erittäin epäammattimaisen kuvan.

Käytettäessä JComboBox-komponenttia eli alasvetolistaa komponentti ei piirry oikein. Jos alasvetolistasta halutaan valita jokin vaihtoehto, lista välkähtää ensimmäisellä hiiren klikkauksella ja vasta toisella klikkauksella jää näkyville mahdollistaen valinnan tekemisen. Taulukkonäkymän tulee kyetä esittämään alasvetolista ilman ylimääräistä välkyntää.

Käytettäessä JButton-komponenttia eli painonappia ei sen painaminen vaikuta ulkonäköön millään lailla. Painonappia painettaessa se näyttää, kuin sitä ei olisi painettukaan sen sijaan, että sen ulkonäkö muuttuisi.

Näiden erityisten piirto-ongelmien lisäksi kaikille komponenteille tyypillinen ongelma, joka liittyy nimenomaan niiden esiintymiseen taulukossa, on fokukseen eli kohdistukseen liittyvä. Komponenttien tai oikeammin solun, jossa ne olivat, ympärille jää tavallisesti fokusta kuvaava keltainen reunus, joka ei poistu vaikka fokus siirtyykin toiselle solulle.

2.4.1.3 Taulukko editorina

Editorin taulukosta tekee se, että taulukon soluissa on sisältöä, jota käyttäjän on tarkoitus muokata. Tämän käyttökohteen alle on listattu Taulukkonäkymään kohdistuvat vaatimukset, jotka erityisesti koskevat Taulukkonäkymän käyttöä tiedon muokkaamisessa.

Vaatimus 7: Editoitujen rivien automaattinen ilmaiseminen
Jos taulukkoa käytetään siten, että siinä olevaa sisältöä on tarkoitus muokata, on joissain tapauksissa tarkoituksellista antaa käyttäjälle välitöntä palautetta siitä, mitä rivejä on muokattu. Tähän keksittiin ratkaisuksi automaattinen editoitujen rivien korostaminen jollain värillä kuitenkin siten, että se pidetään erillään solujen, rivien ja sarakkeiden korostamisesta (ks. 2.4.1.1 Perinteinen taulukko - Vaatimus 2).

Tämä automaattinen korostaminen listattiin Taulukkonäkymän vaatimuksiin sillä lisäyksellä, että ominaisuus pitää voida asettaa päälle ja pois. Lisäksi implementoijalle pitää pystyä tuomaan tieto, kun jollain rivillä olevaa sisältöä on muokattu.

2.4.1.4 Toiminnallinen, dataa esittävä, taulukko

Tämän käyttökohteen alle on listattu vaatimukset, jotka on nimenomaan kohdistettu toimintaa sisältävälle taulukolle, jossa on sisältönä sekä aktiivikomponentteja että tekstiin verrattavaa sisältöä ja joka lisäksi listaa jotain. Käyttökohteena tämä poikkeaa edellisistä vaatimusten osalta siten, että vaatimukset liittyvät tiiviisti sekä aktiivikomponenttien toimintaan että sisällön esittämiseen.

Vaatimus 8: Suodatusrivi ja sen automatisointi
Tähän saakka suodatusrivi ja sen toiminnallisuus on aina rakennettu taulukkoon tarvittaessa. Kävi ilmi, että suodatusrivillä varustettua taulukkoa tultaisiin tarvitsemaan useassa käyttöliittymässä, joten sen automatisointi olisi tarpeen. Käytännössä automatisointi tarkoittaa sitä, että suodatusrivi ei millään lailla saisi vaikuttaa taulukon käytettävyyteen, se olisi helppo lisätä ja poistaa ja että suodatuskomponenttien arvot olisivat helposti saatavissa. Automatisointia ei kuitenkaan vietäisi sille tasolle, että suodatus tehtäisiin suoraan taulukon sisällölle, vaan implementoijan tulisi huolehtia suodatetun sisällön asettamisesta sen mukaan, mitä suodatuskomponenteissa on. Suodatusrivin automatisointi tarkoittaa käytännössä myös sitä, ettei itse suodatusriville pitäisi voida päästä käsiksi muuten kuin nimenomaan sille tarkoitetuilla metodeilla. Suodatusrivi ei esimerkiksi saisi näkyä rivi-indekseissä.

Vaatimus 9: Sisällön lajittelu sarakkeen mukaan
Lajittelu suodatuksen rinnalla ja erikseen helpottaa huomattavasti halutun tiedon löytämistä. Lajittelu haluttiin ominaisuudeksi Taulukkonäkymään nimenomaan tiedon etsintää helpottamaan. Lajittelussa tulee ottaa huomioon sisällön muuttumisen mahdollisuus, mikä käytännössä tarkoittaa, että uudelleenlajittelu tulee tehdä tarvittaessa automaattisesti. Lajittelussa tulee myös ottaa huomioon ero numeroiden

ja tekstin lajittelussa (vertaa tekstinä: 1, 11, 111, 2, 36, 7 ja numeroina: 1, 2, 7, 11, 36, 111). Lisäksi aktiivikomponenteista tulee lajittelussa ottaa huomioon niissä näkyvissä oleva tieto tai valinta. Oleellisinta huomioon ottamisessa on määrittää, miten tämä näkyvissä oleva tieto tai valinta määritellään esimerkiksi valintaruuduissa, jotta se olisi verrattavissa tekstitietoon samassa sarakkeessa.

2.4.2 Implementoijan vaatimukset

Implementoijan vaatimukset liittyivät enimmäkseen rajapintaan ja käytettävyyteen.

Vaatimus 10: Rajapintojen selkeys
Vaatimuksena on rajapinnan ehdoton selkeys ja se, että kaikkien metodien tulee olla asianmukaisesti dokumentoitu ohjelmakoodiin, jotta niiden toiminta ei jää epäselväksi. Lisäksi vaaditaan, että metodien nimet tulee aloittaa Javalle tyypillisesti pienellä alkukirjaimella ja että koodissa käytettynä kielenä tulisi luettavuuden vuoksi olla englanti. Ohjelmakoodissa tuli myös noudattaa DokuMentori Oy:n ohjelmistotuotannon laatukäsikirjassa esiteltyjä koodaustapoja, mikä helpottaa mahdollisen jatkokehittäjän työtä huomattavasti.

Vaatimuksena kaikille metodeille on, että niiden nimien tuli olla kuvaavia ja että kaikille oleellisille toiminnallisuuksille tulee olla tarvittavat get- ja set-metodit (esimerkiksi getData(row, column) ja setData(data, row, column)). Lisäksi metodien nimien tulisi seurata Javan omissa luokissa käytettyjä metodeita niiltä osin kuin toiminnallisuus on verrattain sama. Rajapinta ei saa tarjota metodeita, jotka saattavat rikkoa Taulukkonäkymän rakenteen (ks. 3.1.4 Pääluokka ja omistussuhteet).

Vaatimus 11: Hyvä käytettävyys
Tärkeimpänä käytettävyyteen liittyvänä vaatimuksena on, että Taulukkonäkymän tulee olla JavaBean. Käytännössä tämä asettaa rakenteelle tietyn vaatimuksen yhdessä rajapintavaatimusten kanssa: Taulukkonäkymässä tulee olla ulospäin ensimmäisenä JPanel-luokan ilmentymä, jossa kaikki mahdolliset rajapintametodit

sijaitsisivat.

Vaatimus 12: Edeltäjään liittyvien ongelmien poistaminen
Kuten mainitsin vaatimus 4:n yhteydessä, Taulukkonäkymä tulee korvaamaan toisen komponentin. Tämä korvattava JTable-luokan ilmentymä sisältää useita implementoijalle kiusallisia piirteitä, joista kaikki vaadittiin implementoijien puolesta korjattaviksi Taulukkonäkymään. Osa näistä piirteistä johtuu JTable-luokan rakenteesta. Osa taas on suoria syy-seuraus-suhteita huonosta rakenteesta ja ohjelmakoodista edeltäjässä. Alle on listattu ne piirteet, jotka ehdottomasti vaativat korjauksia.

- sarakkeen lisääminen johonkin tiettyyn indeksiin näkyy visuaalisessa taulukossa oikeassa indeksissään, mutta silti sarakkeeseen joutuu aina viittaamaan, kuin se olisi viimeinen sarake

- rivien poistamisen taulukon keskeltäkin tulee onnistua ongelmitta

- solujen lukitsemisen täytyy onnistua ilman, että käytetään komponenttia, jota ei voi editoida

- rivin lisääminen on voitava tehdä yhdellä käskyllä: edeltäjässä on ensin luotava uusi tyhjä rivi, jonka jälkeen erikseen jokaiseen uuteen soluun asetetaan sisältö

- uuden sisällön asettaminen taulukkoon ei saa vaatia implementoijaa ottamaan huomioon uudelleenmaalaamista, vaan Taulukkonäkymän on huolehdittava itse automaattisesti sisältönsä maalaamisesta tarpeen vaatiessa

Vaatimus 13: Näkyvän tiedon ja solun komponentin erottaminen
Javan JTable- ja DefaultTableModel-luokkien rajapinnat ja toiminnallisuus on luotu siten, että taulukon solusta voidaan hakea vain sen sisällä oleva luokka (Object getValueAt(row, column)). Jos solu sisältää tekstiin verrattavaa sisältöä,

palautuu tällöin näkyvä teksti, mutta jos solussa on komponentti ei palaudukaan teksti, vaan komponentti itse. Toivomuksena implementoijilta tulikin, että rinnalle saataisiin metodit, joilla voitaisiin hallita myös solun näkyvää sisältöä komponentin sijaan.

3 SUUNNITTELU

Tässä luvussa esitellään Taulukkonäkymän suunnitteluvaihe.

3.1 Rakenne

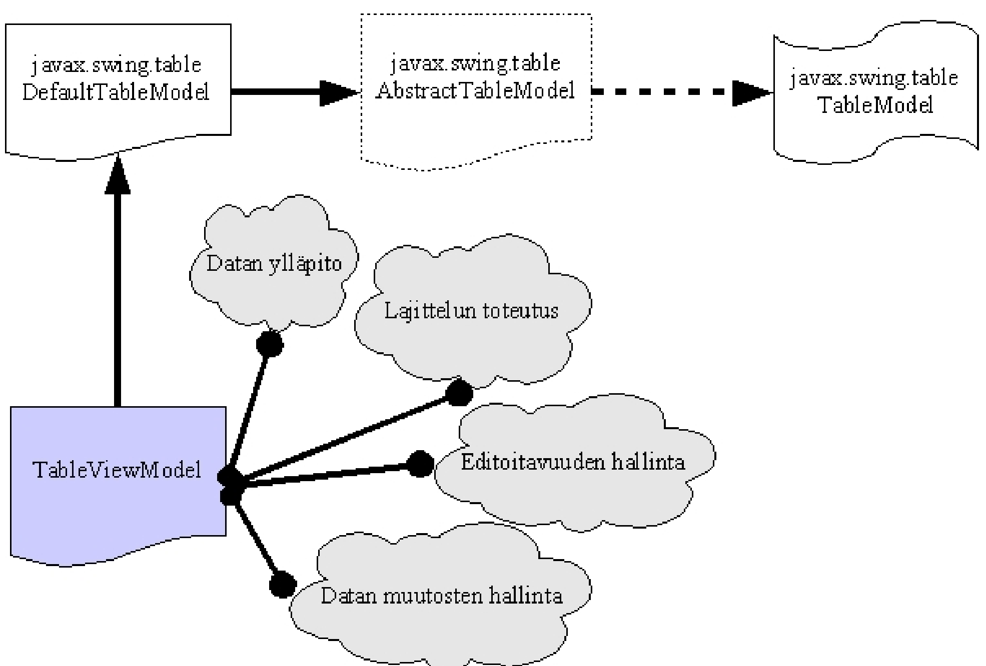
Toteutusta aloitettaessa oli selvää, että Taulukkonäkymän rakenne tulisi olemaan MVC-mallin mukainen. Käytännössä tämä edellyttäisi, että jokainen luokka tulee sijoittaa omaan tiedostoonsa ja että toiminnallisuudet tulisi tarkasti rajata kullekin MVC-mallin osalle.

3.1.1 Model – Tietomalli

Taulukon tietomallin tehtävä on esitettävän sisällön hallinta ja säilytys. Tietomallin tulee sisältää kaikki käsitteelliseen sisältöön liittyvä toiminnallisuus kuten tallentaminen, lajittelusta huolehtiminen ja editoitavuuksien käsittely. Tässä vaiheessa tiedetään, että tietomalli tulee sisältämään vain yhden luokan, jonka sisälle kaikki vaadittava toiminnallisuus on rakennettava.

TableViewModel-luokka periytetään Javan javax.swing.table.DefaultTableModel-luokasta, koska tämä luokka tarjoaa siihen parhaat edellytykset. Jos käytettäisiin tietomallin pohjana suoraan javax.swing.table.AbstractTableModel-luokkaa täytyisi toteutusta tehdä paljon enemmän ja kuitenkin toteutettavien osien toiminnallisuus olisi aivan samanlainen kuin javax.swing.table.DefaultTableModel-

luokassa. Esitelty javax.swing.table-paketti pitää sisällään Javan taulukon tarvitsemat luokat.



Kuva 1 Taulukkonäkymän model-osan rakenne ja toiminnallisuus

3.1.2 View – Näkymä

Näkymän tehtävä on huolehtia kaikesta, mikä liittyy sisällön esittämiseen, kuten esimerkiksi vaadituista korostuksista, taulukon sisällön piirtämisestä ja lajitteluikoneista.

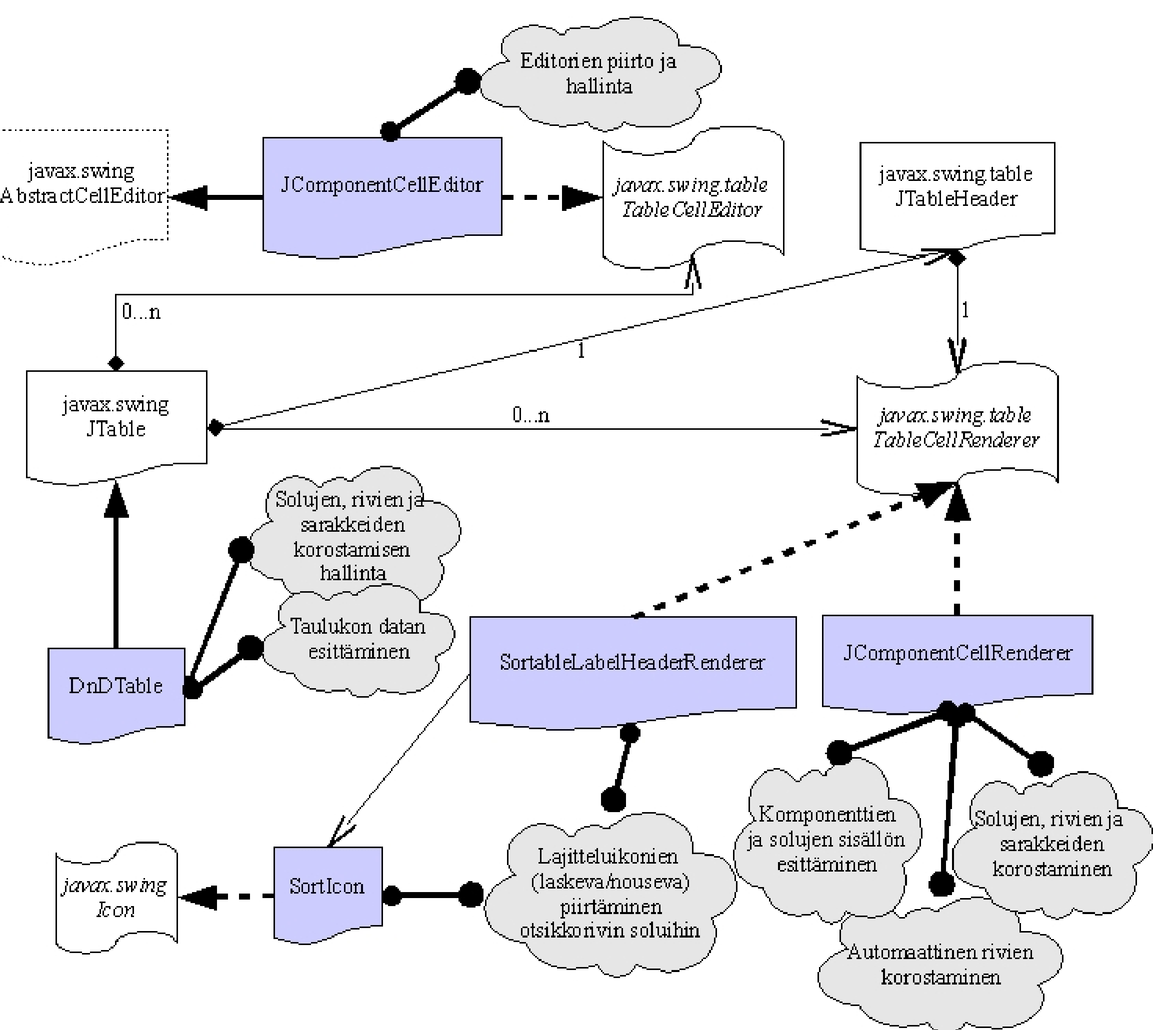
JComponentCellEditor-luokka

Kuvassa 2 esitelty JComponentCellEditor on luokka, jonka tehtävä on huolehtia kaikkien aktiivikomponenttien editorin piirtämisestä ja toiminnasta. Editorit toimivat taulukossa käytännössä siten, että JTable-luokalla on olemassa tietorakenne, johon on tallennettu kunkin luokan editori. Oletuksena se on kaikille luokille JTablen oma oletuseditori, mutta se voidaan halutessa vaihtaa. Tarkoituksena onkin asettaa kaikille aktiivikomponenttiluokille oletuseditoriksi

JComponentCellEditor, mikä toivottavasti poistaa kaikki piirto-ongelmat (ks. 2.4.1.2 Taulukko valintakomponenttina – Vaatimus 6).

JComponentCellRenderer-luokka

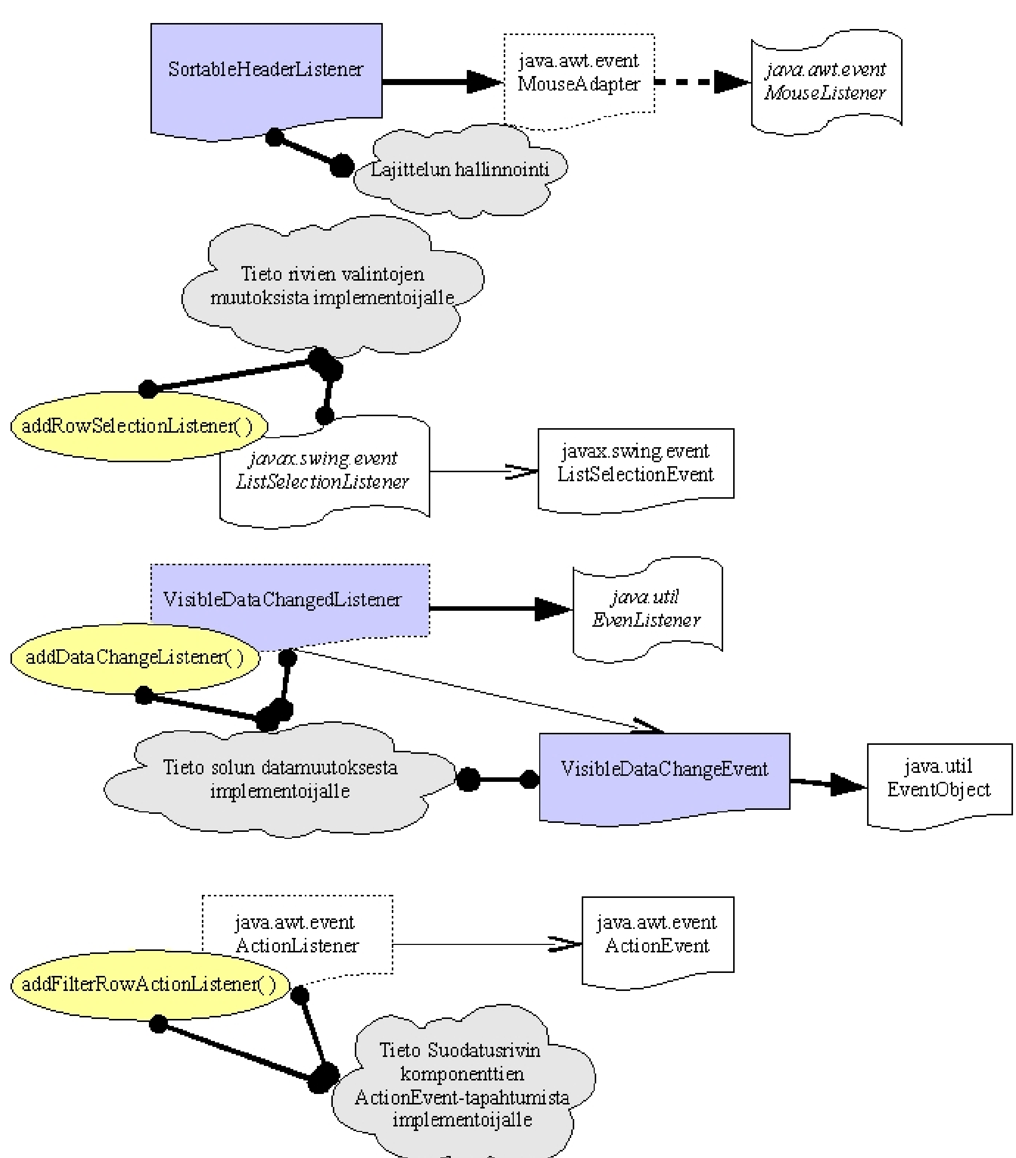
Samoin kuin editoreille JTable-luokalla on olemassa rendereillekin tietorakenne, josta jokaiselle luokalle löytyy oletusrenderer. JComponentCellRenderer-luokka tullaan asettamaan rendereriksi kaikille aktiivikomponenteille, jolloin se huolehtii niiden piirtämisestä kaikilta osin silloin, kun ne eivät ole editointitilassa.



Kuva 2 Taulukkonäkymän view-osan rakenne ja toiminnallisuus

3.1.3 Controller – Hallinta

Hallinta koostuu luokista, jotka ottavat vastaan käyttäjän syötteitä, kuten esimerkiksi hiiren klikkauksia.



Kuva 3 Talukkonäkymän controller-osan rakenne ja toiminnallisuus

ListSelectionListener-luokka

Kuvassa 3 esitelty ListSelectionListener on kuuntelija, jonka tehtävä on huolehtia rivin valinnan muutoksien kuuntelemisesta oli sitten kyse taulukoista tai listoista.

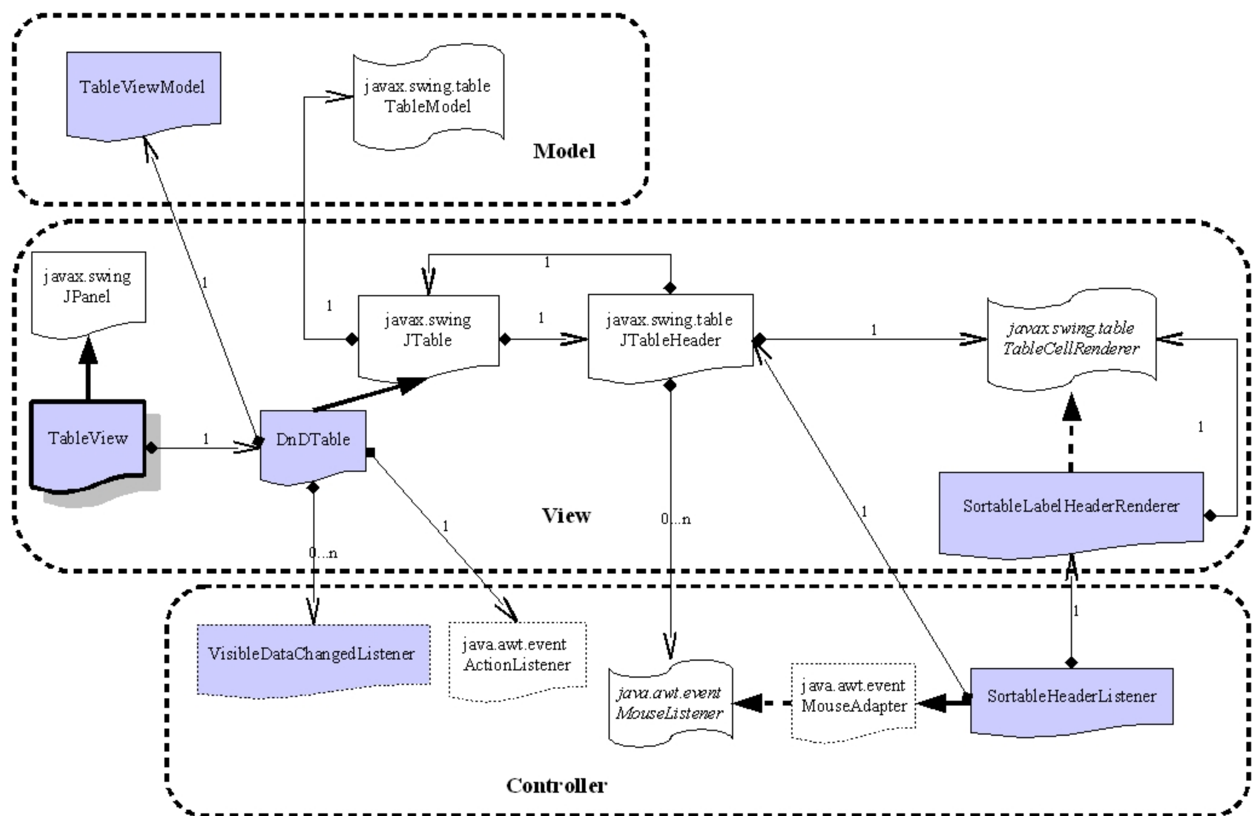
JTable-luokalle voi halutessaan lisätä yhden tai useamman tällaisen kuuntelijan, mikä tarkoittaa, että se on mahdollista myös periytymissuhteen myötä DNDTable-luokalle (ks. 3.1.2 View - Näkymä). Kuten kuuntelijoiden yleensä, ListSelectionListenerin toteutuksesta huolehtii implementoija.

3.1.4 Pääluokka ja omistussuhteet

Monien toiminnallisuuksien toteuttamisen kannalta on oleellista, ettei implementoija pääse Taulukkonäkymää käyttäessään luomaan tilannetta, jossa jokin osa hajoaisi. Näin saattaisi tapahtua jos implementoijalle näkyy ulospäin metodeita, jotka antavat pääsyn taulukon oleellisiin muuttujiin, kuten esimerkiksi tietomalliin. Tietomalliin käsiksi pääseminen mahdollistaisi esimerkiksi rivien lisäämisen lajittelun ohi, näkyvän sisällön täydellisen sekoittamisen tai editoitavuuksien rikkomisen.

Tähän ongelmaan on onneksi olemassa hyvin yksinkertainen ratkaisu, joka on myös helppo toteuttaa. Perimällä pääluokka JPanel-luokasta sen sijaan, että se olisi suoraan JTable-luokasta peritty, voidaan implementoijalta estää suora pääsy JTable-luokan metodeihin. Tätä ratkaisua tukee myös vaatimus, että Taulukkonäkymän tulisi olla JavaBean (ks. 2.4.2 Implementoijan vaatimukset – Vaatimus 10). Käytännössä tämä siis tarkoittaa, että pääluokka omistaa instanssin taulukkoluokasta ja implementoijan tarvitsemat rajapintametodit kirjoitetaan pääluokkaan.

MVC-mallin osien välillä täytyy olla yhteyksiä, jotta tiedon kulku toimisi. Esimerkiksi hallinnoijan täytyy olla tietoinen hallittavistaan ja näkymän tietoinen tietomallistaan. Kuvassa 4 on esitelty oleelliset suhteet MVC-mallin eri osissa sijaitsevien luokkien välillä. Lisäksi on esitelty omistussuhteet, jotka oleellisesti liittyvät toimintamekanismeihin ja jotka eivät ole tulleet esille aikaisemmin.



Kuva 4 Rakenne ja luokkien omistussuhteet

TableViewModel -luokka

Kuten kuvassa 4 on esitetty, JTable-luokka omistaa instanssin TableModel-rajapinnasta, jonka implementoijan on tarkoitus asettaa. Käytännössä tämä tarkoittaa sitä, että DNDTable-luokka asettaa omistamansa TableViewModel-luokan täksi instanssiksi. Yleensä tietomallin tarkoitus on, että implementoija voi sen itse toteuttaa ja asettaa sen sitten taulukolle. Tässä tapauksessa, kuten edellä on tullut ilmi, on kuitenkin tarpeen suojata tietomalli. Tämän vuoksi DNDTable omistaa instanssin omasta tietomallistaan, eikä sitä näytetä ulospäin.

ActionListener-luokka

Kuvassa 4 esitelty ActionListener, jonka DNDTable omistaa, on suodatusrivin komponenttien kuuntelija, joka asetetaan kaikille suodatusriville asetettaville komponenteille. Tämä mahdollistaa suodatusrivillä tapahtuvan ActionEventin (esim. valinta alasvetolistassa, painikkeen painaminen tai Enter-näppäimen painallus tekstikentässä) kulkeutumisen implementoijalle.

SortableLabelHeaderRenderer-luokka

SortableLabelHeaderRenderer on luokka, joka huolehtii lajitteluikoneiden piirtämisestä taulukon otsikon soluihin, kun se on tarpeen. Jotta SortableLabelHeaderRenderer osaisi piirtää oikeaan paikkaan, omistaa se instanssin DNDTable-luokan otsikon (JTableHeader) rendereristä (JTableCellRenderer).

SortableHeaderListener-luokka

SortableHeaderListener on kuuntelijaluokka, joka kuuntelee hiiren klikkauksia. Tämä kuuntelija lisätään DNDTable-luokan otsikolle (JTableHeader). Instanssi otsikosta välitetään myös SortableHeaderListener-luokalle. Kuuntelijan täytyy olla tietoinen otsikostaan, jotta se voisi välittää lajittelukomennon oikealle tietomallille. Tämän mahdollistaa se, että JTableHeader-luokka omistaa instanssin JTable-luokasta, jonka otsikkona se esiintyy. JTable-luokan instanssin kautta päästään puolestaan käsiksi tietomalliin, jonka sisältö pitäisi lajitella. Lisäksi kuuntelijalla on instanssi SortableLabelHeaderRender-luokasta, jotta kuuntelija voisi komentaa sitä piirtämään vaaditun lajitteluikonin oikeaan otsikon soluun.

3.2 Ongelmien arviointi

Projektin alussa pyrittiin myös kartoittamaan mahdollisia ongelmatilanteita, jotka saattaisivat vaikuttaa toteutukseen. Edellä on esitelty riskit ja ongelmatilanteet, jotka alkuvaiheessa tiedostettiin.

DokuMentori Oy:n ohjelmistokehittäjistä yhdellä oli aikaisempaa kokemusta mittavan, Javan omaa komponenttia laajentavan komponentin toteuttamisesta. Tästä aiemmasta projektista oli opittu se tosiasia, että komponentin laajentaminen vaatii perinpohjaisen tutustumisen laajennettavan komponentin toimintamekanismeihin ja rakenteeseen, mikä puolestaan pidentää suunnittelujaksoa, mutta lyhentää toteutusvaihetta huomattavasti.

Rakenteen syvyys

Rakenteen ollessa syvä, mikä on aina tilanne Javan monimutkaisimmissa komponenteissa, joihin taulukkokin kuuluu, on kaiken toiminnan kattava kartoittaminen mahdotonta. Mahdotonta siitä tekee se, että kartoittaminen vaatisi kaikkien komponenttiin kuuluvien luokkien ohjelmakoodin läpikäynnin, mikä saattaisi viedä viikkoja. Kompromissina alussa päätettiin käydä läpi niihin toiminnallisuuksiin liittyvät mekanismit, joihin vaadittiin parannuksia. Tämä suhteellisen pintapuolinen tarkastelu todennäköisesti vähentää projektiin kuluvaa aikaa. Pintapuolinen tarkastelu saattaa myös kostautua aikatauluissa siten, että toteutusvaiheessa voi tulla vastaan ongelma, joka vaatii lisää komponentin tarkastelua ja vie näin lisää aikaa. Tämä riski päätettiin kuitenkin ottaa ja huomioida se aikataulutuksessa: neljän viikon jakson sijaan aikataulua venytettiin viiteen viikkoon.

Jumiutuminen

Projektin jumiutuminen toteutusvaiheessa on myös varteenotettava riski. Jumiutuminen saattaa seurata esimerkiksi, jos jokin toiminnallisuus ei käyttäydy halutulla tavalla eikä mitään näkyvää syytä löydy. Tällainen tilanne saattaisi toteutua jos esimerkiksi jokin toimintamekanismi on toteutettu niin syvällä, ettei sen toimintaa voi paikallistaa kuin laajennettavan komponentin lähdekoodin sekä Java-ohjelmointikielen mekanismien täydellisellä ymmärtämisellä, mikä on suhteellisen mahdotonta. Tällöin ainoa ratkaisu on päätellä todennäköisimmät vaihtoehdot ja testata ne. Tämä on äärimmäisen aikaavievää, mutta silti nopeampaa, kuin etsiä vikaa lähdekoodista, jota ei tunne. On myös hyvin todennäköistä, että tämä keino tuottaa tulosta, koska yleensä ongelmat mekanismeissa voidaan rajata jollekin alueelle tai liittää ne johonkin luokkaan tai rakenteeseen.

4 TOTEUTUS

Tässä luvussa esitellään Taulukkonäkymän toteutukseen liittyvät seikat ja käsitellään lyhyesti myös Taulukkonäkymän testausta. Toteuttaminen lähti liikkeelle pääluokan toteuttamisesta ja rajapintojen suunnittelusta, minkä jälkeen pääluokan taakse rakennettiin varsinainen taulukko eli DNDTable-luokka sekä siihen liittyvä tietomalli TableViewModel-luokka. Tämän jälkeen ryhdytiin rakentamaan toiminnallisuuksia ja liittämään tarvittavia luokkia kokoonpanoon.

4.1 Toiminnallisuuksien toteuttaminen

Toteutusvaihe eteni toteutettavalta komponentilta vaadittujen toiminnallisuuksien mukaan siten, että ensin keskityttiin aktiivikomponentteihin liittyviin vaatimuksiin, jotta voitaisiin heti alussa olla varmoja, että niitä voitaisiin käyttää Taulukkonäkymässä. Seuraava etappi oli toteuttaa vaatimukset, jotka liittyivät editoitavuuksiin, lajitteluun ja käyttäjän tekemiin solujen maalaamisiin. Tämän jälkeen jäljelle jäivät suodatusrivi, editoitujen rivien näyttäminen sekä DND.

4.1.1 Aktiivikomponentit Taulukkonäkymään

Koska koko Taulukkonäkymän käyttökohteiden kannalta tärkein toiminnallisuus on kyky esittää aktiivikomponentteja siten, että ne toimivat odotetusti otettiin tämä toiminnallisuus ensimmäisenä toteutettavaksi. Toiminnallisuutta lähdettiin rakentamaan ensimmäisenä myös siitä syystä, että sen ennakoitiin tuottavan eniten mahdollisia viivästyksiä, koska toiminnallinen koodi komponenttien piirtämiseen on todella syvällä Javan omien komponenttien hierarkkisessa rakenteessa.

Toteutuksen yhteydessä luotiin ensin rendererluokka JComponentCellRenderer ja editorluokka JComponentCellEditor (ks. 3.1.2 View – Näkymä). DNDTable-luokan sisällä määrättiin kaikille aktiivikomponenttiluokille, jotka vaatimusten mukaan tuli kyetä esittämään, oletusrendereriksi JComponentCellRenderer ja

oletuseditoriksi JComponentCellEditor. Hyvin nopeasti kuitenkin huomattiin, että tämä ratkaisu, jonka ennakoitiin poistavan kaikki aktiivikomponentteihin liittyvät ongelmat, ei poistanut niistä ainuttakaan (ks. 2.4.1.2 Taulukko valintakomponenttina – Vaatimus 5, Vaatimus 6).

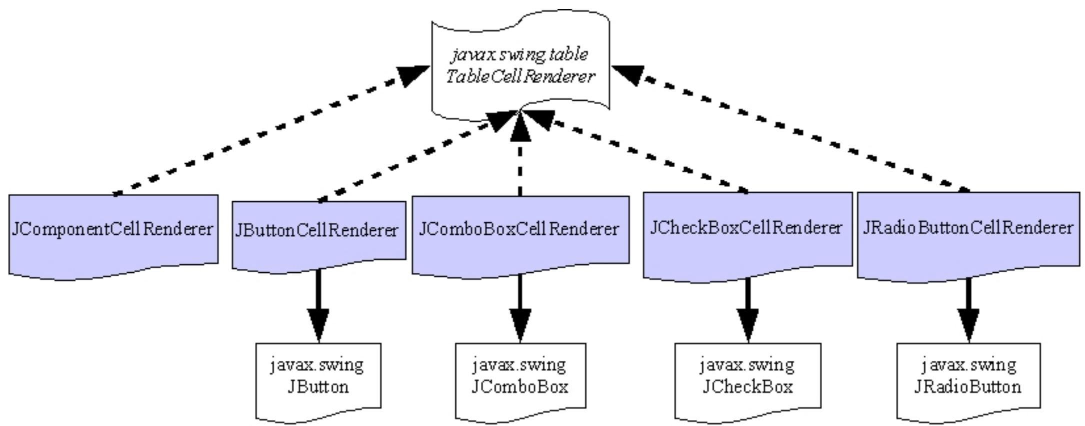
Koska mistään lähteestä ei löytynyt esimerkkitapausta aktiivikomponenttien onnistuneesta käytöstä Javan taulukossa saati ratkaisua tunnettuihin ongelmiin, päätettiin tätä ilmennyttä ongelmaa lähestyä tutkimalla komponenttien yleisiä piirtomekanismeja. Tämä lähestymistapa ei kuitenkaan tuonut minkäänlaista tulosta, vaan vielä muutaman päivän kuluttua oltiin lähtöpisteessä. Näihin erikoisiin ja selvästikin piirtoon liittyviin ongelmiin ei kyetty löytämään ratkaisua ohjelmakoodia tutkimalla toivotussa ajassa, koska se vaatisi täydellistä ohjelmakoodin tuntemista kaikkien mekanismiin osallistuvien luokkien osalta.

Seuraava looginen askel oli pyrkiä rajaamaan aktiivikomponenttien mahdollinen häiriökäyttäytyminen Taulukkonäkymässä johonkin tiettyyn osa-alueeseen ja näin rajata etsittävää aluetta eli luokkien määrää. Selvää oli, että vian täytyi olla jossain Javan taulukon ja aktiivikomponentin välissä, koska aktiivikomponenttien piirto-ongelmat ilmenivät vain, kun niitä käytettiin taulukon soluissa. Tämän vuoksi ongelma voitiin rajata kolmeen luokkaan: omaan editoriin, omaan rendereriin ja JTable-luokkaan. Ensin ongelman syytä etsittiin luokkien välisestä toiminnasta, mikä ei tuottanut tulosta usean päivän etsinnöistä huolimatta. Aluksi näytti, että vika olisi pohjimmiltaan editorissa, koska Taulukkonäkymässä alasvetolista välkähtää editorin ollessa piirtovuorossa, joten kaikille aktiivikomponenttiluokille tehtiin omat editorit: JLabelCellEditor, JButtonCellEditor, JComboBoxCellEditor, JCheckBoxCellEditor ja JRadioButtonCellEditor. Päivän intensiivinen testaaminen kuitenkin osoitti, ettei tästä ollut mitään hyötyä.

Viimein päädyttiin hakuammuntaan. Editorin lisäksi alasvetolistalle luotiin myös oma renderer: JComboBoxCellRenderer, joka perittiin javan JComboBox-luokasta ja joka toteutti TableCellRenderer-rajapinnan. Tämä ratkaisu näytti poistavan alasvetolistan välkkymisen ja fokus siirtyi pois alasvetolistalta silloin, kun sen pitikin.

Testauksen perusteella voitiin päätellä, että aktiivikomponenteilla, joihin käyttäjän syötteet vaikuttavat, täytyy olla omasta luokasta peritty renderer. Ratkaisun varmistuttua kaikille aktiivikomponenttiluokille tehtiin omat rendererit ja täysin tarpeettomiksi osoittautuneet komponenttikohtaiset editor-luokat poistettiin. Perimmäinen syy, miksi oman renderien lisäys kullekin aktiivikomponentille ratkaisi piirto-ongelmat, ei selvinnyt. Koska mekanismin löytäminen Java-kielen komponenttien ohjelmakoodista ei kuulunut projektin kuvaan ja koska mekanismin ymmärtäminen ei ole edellytys ratkaisun löytämiselle, ei syytä ratkaisun toimivuuteen odotettukaan eriteltävän.

Kuvassa 5 on esitetty luokkahierarkia renderereille, jotka lopulta toteutettiin. JLabel-luokalle ei toteutettu rendereriä, koska se on komponentti, joka ei reagoi käyttäjän syötteisiin. Tämän vuoksi tälle komponentille voidaan käyttää rendererinä JComponentCellRenderiä ilman pelkoa piirto-ongelmista.



Kuva 5 Aktiivikomponenttien rendererit

4.1.2 Luettavuus paranee automaattisilla korostuksilla

Luettavuutta parannettiin usealla ominaisuudella, joista yksinkertaisin oli joka toisen rivin automaattinen korostaminen. Tämän ominaisuuden toteutus oli todella yksinkertainen: tarkistetaan vain rivi-indeksin parillisuus ja parillisen rivin solujen taustat värjätään toisen väriseksi. Rajapintaan tehtiin metodit, jotka mahdollistavat tämän automaattisen värin vaihtamisen ja ominaisuuden hallinnoinnin. Lisäksi

toteutettiin rakentajia, joilla ominaisuus voidaan asettaa päälle tai pois. Oletuksena ominaisuus on päällä ja parilliset rivit maalataan värillä, joka on määritelty TableViewConstants-luokassa erillisellä vakiolla.

Koska tässä vaiheessa Taulukkonäkymässä oli useampi renderer, ei ollut järkevää kirjoittaa taustavärin asettavaa koodia erikseen kaikkiin rendererеihin. Tämän vuoksi luotiin erillinen abstrakti luokka, jolla on staattisia metodeita, huolehtimaan taulukon rivien taustaväreistä. Samaan luokkaan tullaan myöhemmin sijoittamaan myös käyttäjän tekemät solujen, rivien ja/tai sarakkeiden korostukset (ks. 2.4.1.1 Perinteinen taulukko – Vaatimus 2 ; 4.1.5 Luettavuus paranee implementoijan tekemillä korostuksilla). Jokaisessa renderereissä kutsutaan tämän luokan staattista piirtometodia. Alla on esimerkkinä JComboBoxCellRenderer-luokan ohjelmakoodi, josta selviää tämän abstraktin luokan käyttö.

```
public class JComboBoxCellRenderer extends javax.swing.JComboBox implements
javax.swing.table.TableCellRenderer {
    /** Creates a new instance of JComboBoxCellRenderer */
    public JComboBoxCellRenderer() {
        super();
        setOpaque( true );
    }

    /* Called from parent when cell needs to be rendered */
    public java.awt.Component getTableCellRendererComponent( javax.swing.JTable table,
                                              Object value, boolean isSelected,
                                              boolean hasFocus, int row, int column) {
        TableViewCellRenderer.renderRow(  row, column, this, isSelected, hasFocus,
                                          (DNDTable) table );
        this.removeAllItems();
        this.addItem(  ( (javax.swing.JComboBox) value).getSelectedItem() );
        this.setSelectedItem( ( (javax.swing.JComboBox) value).getSelectedItem() );
        this.setEnabled( ( (javax.swing.JComboBox) value ).isEnabled() );
        return (java.awt.Component) this;
    }
}
```

TableViewCellRenderer-luokan tarkoitus on siis huolehtia kaikilta osin taustavärien piirtämisestä soluihin, joka tehdään luokan staattisessa renderRow()-metodissa.

4.1.3 Editoitavuuksien muokkaaminen

Taulukon sisällä solun editoitavuus ei tule riippua suoraan solussa olevan komponentin editoitavuudesta, vaan solulle itselleen asetetusta editoitavuudesta. MVC-mallissa on tietomallin tehtävä huolehtia sisällön säilyttämisen ohessa myös sen editoitavuuden hallinnoinnista. Taulukkonäkymän tietomalli TableViewModel-luokka on peritty Javan DefaultTableModel-luokasta, jossa on toteutettu TableModel-rajapinnassa määritelty boolean isCellEditable(row, column)-metodi, joka palauttaa aina oletuksena true. Ylikirjoittamalla tämä metodi, voidaan solun editoitavuutta säädellä halutulla tavalla.

Ensimmäinen haaste ennen metodin ylikirjoittamista oli kuitenkin valita sopiva tietorakenne editoitavuuksien säilyttämiseen. Javassa on toteutettu STL-tietorakenteita ja assosiaatiivisiä säiliöitä vastaavat luokat Java Collections:ssa /4/, joita tässä käytettiin. Editoitavuudesta päätettiin säilöä aina yksittäisen solun tieto, koska se on helpoin tarkistaa. Kuitenkin tiedon säilöminen jokaisesta solusta erikseen (true/false) todettiin liian raskaaksi ja myös tarpeettomaksi, sillä säilömällä esimerkiksi editoitavia soluja säilöttävän tiedon määrä on huomattavasti pienempi ja lisäksi tietorakenteen ei tarvitse olla niin monimutkainen.

Useiden testien ja käyttötilanteiden perusteella päädyttiin käyttämään map-tyyppistä säiliötä, jossa avaimena on rivi-indeksi ja sisältönä arraylist-tyyppinen tietorakenne, joka sisältää niiden solujen sarake-indeksit ko. rivillä, jotka ovat lukittuja. Map-tietorakenne ehkäisee myös tiedon duplikaatteja (useita samoja rivi-indeksejä) ja mahdollistaa niiden rivien poisjättämisen, joilla ei ole lukittuja soluja (rivi-indeksillä ei löydy arvoa map-tietorakenteesta).

Suurin haitta toteutetusta tietorakenteesta koitui tilanteesta, jossa taulukkoon lisätään rivi haluttuun indeksiin. Map-tietorakenteessa ei voi lisätä avainten arvoa automaattisesti, vaan koko tietorakenne joudutaan käymään indeksin, johon rivi lisättiin, alapuolelta läpi ja lisäämään avainarvoa ohjelmakoodissa. Koska yleensä taulukkoa käytetään lisäämällä rivejä taulukon loppuun, tämän ikävän piirteen ei katsottu aiheuttavan Taulukkonäkymässä merkittäviä viiveitä. Lisäksi taulukossa

olevien lukittujen solujen määrän täytyisi olla merkittävä, jotta siitä seuraisi havaittavia vaikutuksia, jotka ilmenisivät piirto-ongelmina.

Vaatimuksen mukaan (ks. 2.4.1.1 Perinteinen taulukko – Vaatimus 3) myös riveille ja sarakkeille tulee voida määrätä editoitavuus. Uusia rivejä lisättäessä täytyykin lukita ne solut, jotka osuvat lukittujen sarakkeiden kohdalle. Samoin uutta saraketta lisättäessä täytyy tietää, onko uusi solu rivin lukituksen myötä lukittu vai ei. Tätä tarkoitusta varten TableViewModel-luokkaan luotiin erilliset arraylist-tietorakenteet, sekä riveille että sarakkeille, joissa säilytetään arvoina lukituksi asetettujen rivien/sarakkeiden indeksejä.

Taulukossa editoitavuusrajoituksia halutaan yleensä tehdä sarakkeille tai yksittäisille soluille. Toinen hyvin yleinen tapaus on, että koko taulukon halutaan olevan lukittu. Tämän vuoksi TableViewModel-luokkaan sijoitettiin tietorakenteen lisäksi myös yksi muuttuja, jonka avulla voidaan määritellä koko taulukko lukituksi. Tällöin yksittäisen solun editoitavuus voidaan pitää erillään koko taulukon editoitavuudesta ja lukittaessa koko taulukko, tietorakennetta ei tarvitse täyttää.

4.1.4 Lajittelun rakentaminen

Tätä ominaisuutta toteutettaessa lähdettiin liikkeelle helpoimmasta päästä eli piirtämiseen liittyvistä luokista ja toiminnallisuuksista. Ensimmäisenä toteutettiin sarakkeiden otsikkoriville renderer eli SortableLabelHeaderRenderer, jota käytettäisiin rendererinä lajittelun ollessa päällä, sekä rendererin tarvitsema ikoniluokka SortIcon, joka piirtää varsinaiset lajitteluikonit.

SortIcon-luokka on rakenteeltaan hyvin yksinkertainen. Se sisältää rakentajan, jolla voi asettaa ikonityypin (ei nuolta, alaspäin-nuoli tai ylöspäin-nuoli) ja ylikirjoitetun paintIcon()-metodin, jossa piirretään ikonityyppiä vastaava nuoli.

SortableLabelHeaderRenderer-luokka huolehtii varsinaisesta ikonin piirtämisestä.

Tämän vuoksi sillä on rajapinnassa metodit klikatun sarakkeen tallentamiseen ja seuraavan tilan asettamiseen. Luokka toteuttaa getTableCellRendererComponent()-metodin, jossa varsinainen otsikon solun ulkoasun luominen tapahtuu. Metodissa asetetaan lajittelun tilaa vastaava ikoni otsikon solun JLabel-komponenttiin. Komponentti haetaan taulukon otsikkorivin oletusrendereriltä, jonka instanssin luokalle on rakentajassa välitetty. Alla on getTableCellRendererComponent()-metodin ohjelmakoodi, josta käy ilmi lajitteluikonien käyttö ja Taulukkonäkymän otsikkorivin oletusrendererin käyttötarkoitus.

```
/** Hakee headerin render-komponentin ja lisää siihen oikean ikonin tilasta riippuen */
public Component getTableCellRendererComponent(JTable table, Object value, boolean
                                isSelected, boolean hasFocus, int row, int column ) {
     // getting default renderer component
     Component c = tableCellRenderer.getTableCellRendererComponent( table, value,
                                                      isSelected, hasFocus,
                                                      row, column );
     // if JLabel was found
     if( c instanceof javax.swing.JLabel ) {
         javax.swing.JLabel l = (javax.swing.JLabel) c;
         Object columnState = mapState.get( column );

         // if there is a state for columnheader to be rendered:
         if( columnState != null ) {
              if( ( (Integer) columnState ) == TableViewConstants.SORT_DESCENDING )
                   l.setIcon( new SortIcon(SortIcon.DOWN_ICON) );
              if( ( (Integer) columnState ) == TableViewConstants.SORT_ASCENDING )
                   l.setIcon( new SortIcon(SortIcon.UP_ICON) );
          } else
              l.setIcon( new SortIcon(SortIcon.BLANK_ICON) );
          l.setHorizontalTextPosition( javax.swing.JLabel.LEFT );
          int modelColumn = table.convertColumnIndexToModel(column);
     }
     return c;
 }
```

DNDTable-luokan otsikolle asetetaan kuuntelijaksi SortableHeaderListener-kuuntelijaluokka. SortableHeaderListener-luokalle on puolestaan rakentajassa välitetty DNDTable-luokan omistama instanssi SortableLabelHeaderRenderer-luokasta. Tämä mahdollistaa SortableLabelHeaderRenderer-luokan käskyttämisen SortableHeaderListener-luokan mouseClicked()-metodista, jota kutsutaan automaattisesti, kun jossain otsikon solussa klikataan hiiren nappia.

Tässä vaiheessa piirron toimivuutta testattiin ja sen todettiin toimivan erittäin

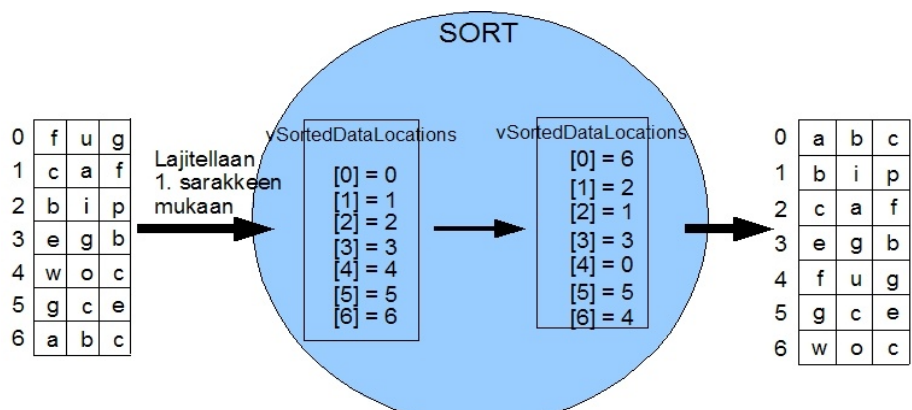
hyvin. Ominaisuudesta puuttui enää itse lajittelu. Lajittelualgoritmi päätettiin sijoittaa kokonaisuudessaan TableViewModel-luokkaan. Tässä yhteydessä luokka uudelleennimettiin SortableTableViewModel-luokaksi, jotta se kuvaisi mahdollisimman kattavasti luokan toimintaa.

Jotta lajittelualgoritmi voitaisiin suorittaa oikeassa vaiheessa ja oikealle sarakkeelle välitettiin SortableHeaderListener-luokalle rakentajassa myös DNDTable-luokan omistama instanssi TableHeader-luokasta, jolloin lajittelumetodia voidaan kutsua mouseClicked()-metodissa. Alla on esitelty ko. metodi kokonaisuudessaan, mistä käy ilmi välitettyjen SortableLabelHeaderRenderer- ja TableHeader-luokan käyttö.

```
public void mouseClicked(MouseEvent e) {
     int col = header.columnAtPoint(e.getPoint());
     int sortCol = header.getTable().convertColumnIndexToModel(col);
     renderer.setPressedColumn(col);
     renderer.setSelectedColumn(col);
     header.repaint();

     int iSortType = renderer.getState(col);
     // sorting...
     ((SortableTableViewModel) header.getTable().getModel()).sortDataByColumn( sortCol,
                                                                   iSortType );
     renderer.setPressedColumn(-1); // clear
     header.repaint();
 }
```

Lajittelu-ominaisuus kuulostaa ensialkuun valtavan yksinkertaiselta ja mitä piirtoon tulee se onkin. Itse lajittelun toteutus ei sitten enää ollutkaan niin yksioikoista. Ei ainoastaan lajitellun sisällön tule näkyä oikein, vaan myös editoitavuuksien tulee seurata mukana kuten myös tulevien implementoijan tekemien korostusten. Useimmissa lähteissä lajittelussa oli käytetty apuna kahta tietomallia, joista toinen näyttää lajiteltua sisältöä ja toinen sisältää lajittelemattoman sisällön. Taulukkonäkymän osalta tällainen ratkaisu ei ole kannattava, koska synkronointi esimerkiksi editoitavuuksien kannalta olisi melko raskasta. Lopulta päädyttiin yksinkertaiseen ratkaisuun: säilytetään lajiteltu rivijärjestys vektorissa, jolloin voidaan pitää tietomallin sisältö aina samassa järjestyksessä ja rajapintametodeissa käyttää tietorakennetta välikätenä rivi-indeksin muuntamisessa.



Kuva 6 Tietorakenteen funktio lajittelussa

Vektorin käyttäminen rivi-indeksien säilyttämisessä mahdollistaa oikeaan riviin viittamisen silloinkin, kun sisältö on lajiteltu, käyttämällä rivi-indeksinä vektorista annetun rivi-indeksin perusteella haettua arvoa. Kuvassa 6 on esitetty, kuinka indeksit tallennetaan vektoriin lajittelun yhteydessä. Alla on esimerkkinä SortableTableViewModel-luokan toteuttama javax.swing.table.TableModel-rajapinnassa esitelty metodi, jota käytetään aina, kun halutaan hakea solussa oleva komponentti.

```
/**
 *  Hakee tietyssä solussa sijaitsevan komponentin/objektin.
 *  @param  row     Rivi, jolla solu sijaitsee
 *  @param  column  Sarake, jossa solu sijaitsee
 *  @return Palauttaa perusluokille (String, Long, Integer...) solussa olevan datan,
 *          mutta jos solu sisältää komponentin, palautetaan ko. komponentti.
 */
public Object getValueAt( int row, int column ) {
    return super.getValueAt( (Integer) vSortedDataLocations.get(row), column );
}
```

Koska lajittelulta vaaditaan ehdotonta nopeutta, valittiin lajittelualgoritmiksi tunnetusti nopea quicksort /5/. Erona perinteiseen quicksort-algoritmiin verrattuna SortableTableViewModel-luokassa toteutetussa lajittelualgoritmissa on mahdollisuus määrittää sarakkeelle tietotyyppi: tekstityyppi tai numerotyyppi. Tietotyypin määrittämisen tarkoitus oli mahdollistaa numeroiden lajittelu oikeaan järjestykseen (vertaa: tekstinä 1, 23, 9, 19, 2, 245 lajiteltaisiin 1, 19, 2, 23, 245, 9, kun ne numeroina tulisi lajitella 1, 2, 9, 19, 23, 245).

Numerotyypin lisääminen lajitteluehtoihin aiheutti hieman lisätöitä, sillä lajittelua tehdessä täytyy olla varma, ettei sarakkeessa, jossa tietotyyppinä on numero, ole muun tyyppistä sisältöä, kuin numeroita tai lajittelu keskeytyy virheeseen. Ongelma ratkaistiin tekemällä muutoksia metodeihin, joilla voidaan asettaa soluihin sisältöä ja joilla rivejä voidaan lisätä: jos sarakkeen tietotyyppi on numerotyyppi ja siihen yritetään lisätä tietoa tai komponenttia, jossa sisältö ei vastaa numeroa, heitetään virhe ja lisäys jätetään tekemättä. Tällä mekanismilla voidaan informoida implementoijaa virheestä ja estää mahdolliset ongelmat lajittelussa. Myös silloin, kun sarakkeelle asetetaan tietotyypiksi numero, käydään sarakkeen sisältö läpi. Jos yhdenkin solun sisältö ei vastaa numerotyypin määritelmää, jätetään lisäys tekemättä ja heitetään virhe.

Taulukkonäkymän normaalissa käytössä on hyvin mahdollista, että vastaan tulee tilanne, jossa käyttäjä muuttaa näkyvää sisältöä solussa, joka sijaitsee sarakkeessa, jonka mukaan lajittelu on tehty. Tällaisessa tilanteessa Taulukkonäkymän tulisi reagoida löytämällä muuttuneelle riville uusi paikka sen uuden sisällön perusteella. Tarvittava toiminnallisuus uuden rivin etsintä mukaan luettuna sijoitettiin SortableTableViewModel-luokan setValueAt()-metodiin, jota kutsutaan automaattisesti, kun solun sisällön editointi päättyy.

Toiminnallisuutta testattaessa todettiin, että on ristiriitaista sallia rivin lisääminen implementoijan toimesta taulukon keskelle, jos taulukko on lajiteltuna. Tämän vuoksi rivin lisääminen taulukon keskelle estettiin tässä tilanteessa. Estäminen toteutettiin lisäämällä rivin lisäyksen suorittavaan metodiin tarkistus siitä, onko jokin sarake lajiteltu vai ei. Sen lisäksi, että lisäys jätetään tekemättä heitetään myös virhe, johon implementoija voi halutessaan reagoida.

Käyttöönotossa huomattiin, että implementoijan lisätessä riviä taulukon loppuun taulukon ollessa lajiteltuna, ei Taulukkonäkymän automaattilajittelu toiminut. Tällä hetkellä rivejä lisätään kerralla niin paljon jokaisessa käyttökohteessa, että lajittelu kannattaa joka tapauksessa ottaa pois päältä rivejä lisättäessä suoritusajan minimoimiseksi. Tämän vuoksi ongelman ei katsottu haittaavan käyttöä.

4.1.5 Luettavuus paranee implementoijan tekemillä korostuksilla

Alusta saakka oli selvää, että korostusten tallentamiseen tultaisiin käyttämään tietorakenteita. DNDTable-luokkaan luotiin tallentamista varten kolme map-tyyppistä säiliötä: mapUserPaintCells, mapUserPaintCols ja mapUserPaintRows. mapUserPaintCells on tietorakenne, joka pitää kirjaa yksittäisen solun korostuksista kun taas mapUserPaintRows ja mapUserPaintCols ovat aputietorakenteita. mapUserPaintRows pitää kirjaa korostetuista riveistä, jolloin uutta saraketta lisättäessä voidaan uusille soluille, jotka ovat korostetuilla riveillä, asettaa oikea väri. Sama funktio on mapUserPaintCols-rakenteella, mutta rivien lisäämisen yhteydessä. Tietorakenteiden käyttö on hyvin saman tyyppinen kuin editoitavuuksissa. Niiden käyttöön sisältyy myös sama ongelma: rivejä lisättäessä map-tietorakenteen avainarvoja täytyy kasvattaa ohjelmakoodissa lisätyn rivi-indeksin alapuolelta, kun uusi rivi lisätään muualle kuin taulukon loppuun. Tämän piirteen ei kuitenkaan katsottu tässäkään tilanteessa aiheuttavan merkittävää haittaa.

Kaikissa edellä esitellyissä tietorakenteissa rivi-indeksit on talletettu sen mukaan, mikä rivi korostettiin tietomallista ei lajitellusta sisällöstä. SortableTableViewModel-luokkaan luotiin tähän tarkoitukseen rivi-indeksin hakemiseen metodi, joka palauttaa tietomallin rivi-indeksin, kun sille välitetään näkyvän taulukon rivi-indeksi. Kaikissa metodeissa, jotka liittyvät korostuksen hakemiseen tai asettamiseen, rivi-indeksi muunnetaan aina sisäisesti tietomallin rivi-indeksiksi, jotta oikea solu pysyy aina korostettuna lajittelustakin huolimatta. Edellä on esimerkkinä DNDTable-luokassa toteutettu metodi, jolla haetaan tieto siitä, onko solu korostettu.

```
public boolean isCellUserPainted( int row, int column ) {
    int iRelativeIndex = objModel.getRelativeIndex(row);
    // if row has zero ephasized cells
    if( !mapUserPaintCells.containsKey( iRelativeIndex ) )
        return false;
    // if given column was not emphasized in given row
    if( !((java.util.HashMap)mapUserPaintCells.get(iRelativeIndex)).containsKey(column) )
        return false;
    return true;
}
```

objModel on DNDTable-luokan omistama SortableTableViewModel-luokan instanssi. Alla on esitelty käytetty getRelativeIndex()-metodi.

```
public int getRelativeIndex( int iVisibleIndex ) {
    return (Integer) vSortedDataLocations.get( iVisibleIndex );
}
```

Itse korostuksen saattaminen näkyviin taulukossa oli todella yksinkertainen toimenpide. Aikaisemmin luvussa 4.1.2 Luettavuus paranee automaattisilla korostuksilla esitellyn abstraktin TableViewCellRenderer-luokan renderRow()-metodiin lisättiin tarkistus siitä onko solu korostettu sekä solun taustavärin asettaminen mahdollisen korostuksen mukaiseksi.

4.1.6 Suodatusrivin automatisointi

Ensimmäinen askel suodatusrivin toteuttamiseen oli suunnitella hyvä rajapinta pääluokkaan, koska rajapinta vaikuttaa paljon siihen, miten varsinainen toiminnallisuus toteutetaan. Varsinkin suodatuskomponenttien asettamiseen sekä arvojen hakemiseen täytyi kiinnittää huomiota, jotta rajapinnat pysyisivät mahdollisimman helppokäyttöisinä. Lopulta päädyttiin kahteen vaihtoehtoiseen tapaan lisätä komponentit: käyttämällä komponenttityyppejä tai käyttämällä varsinaisia komponentteja.

Komponenttityyppejä varten TableViewConstants-luokkaan määriteltiin vakioiksi mahdolliset komponenttityypit, joita ovat: ei komponenttia, tekstikenttä ja alasvetolista, josta on mahdollisuus valita 'true', 'false' tai tyhjää. Rajapinnasta löytyy lisäksi metodit suodatusrivin piilottamiselle sekä kuuntelijan asettamiselle suodatusrivikomponenteille. Kuuntelijana toimii ActionListener-rajapinnan toteuttava luokka, jonka implementoija toteuttaa haluamallaan tavalla. Taulukkonäkymä ei suorita suodatusta automaattisesti, vaan suodatusriville on lisättävissä kuuntelija nimenomaan siksi, että implementoija voisi suodatukomponentin arvon muuttuessa huolehtia sisällön esittämisestä kuten tahtoo. Rajapintaan on myös toteutettu metodi, jolla voidaan hakea suodatusrivissä

olevien komponenttien tekstiin verrattavat arvot.

Suodatusrivitoiminnallisuuden vaatimuksen yhteyteen (ks. 2.4.1.4 Toiminnallinen, dataa esittävä, taulukko – Vaatimus 8) kirjattiin myös vaatimus siitä, että suodatusrivi ei saisi näkyä implementoijalle rivi-indekseissä. Ratkaisu tähän oli todella yksinkertainen. Lisätään rajapinnan kaikissa metodeissa, joissa asetetaan jotain rivi-indeksin perusteella, rivi-indekseihin arvoa yhdellä suodatusrivin ollessa näkyvillä. Tällöin kun implementoija käyttää indeksiä 0, on se tällöin taulukon ensimmäinen eli suodatusrivistä seuraava rivi. Samoin kaikissa rajapintametodeissa, joissa haetaan rivi-indeksejä, vähennetään saatuja rivi-indeksien arvoja yhdellä suodatusrivin ollessa näkyvillä. Tällöin ulospäin näyttää siltä, että suodatusriviä ei oteta huomioon lainkaan.

4.1.7 Editoitujen rivien korostaminen käyttäjälle

Editoitujen rivien tallentamisessa tuli ottaa huomioon kolme asiaa: onko rivi jo editoitujen joukossa, onko asetettava sisältö muuttunut ja onko se peräisin käyttäjältä vai asetettu ohjelmallisesti. Jos uusi sisältö ei ole peräisin käyttäjältä, vaan implementoija on sen ohjelmallisesti asettanut, ei riviä saa lisätä editoitujen joukkoon. Kaikki nämä edellä mainitut asiat voidaan ottaa huomioon yhdessä metodissa: DNDTable-luokan setValueAt()-metodissa.

setValueAt()-metodia kutsutaan automaattisesti kun solun editointi lopetetaan. Tässä metodissa voidaan verrata edellistä arvoa asetettavaan sekä tarkistaa ettei rivi ole valmiiksi editoitujen joukossa. Koska metodi on implementoijankin käytössä rajapinnan myötä, täytyy ensin tarkistaa, että arvoa ollaan muuttamassa käyttäjän puolesta eikä ohjelmallisesti. Jos arvoa muutetaan käyttäjän puolesta, käytetään muuttamiseen käytetään tällöin editoria, joka haetaan käyttöön automaattisesti kutsuttavalla getCellEditor()-metodilla. Tämän metodin sisällä voidaan ottaa talteen edellinen arvo sekä asettaa päälle lippu (bAddedThroughEditor), jolla voidaan myöhemmin varmistaa, että arvoa ollaan muuttamassa käyttäjän toimesta. Edellä on esitelty setValueAt()-metodissa sijaitseva ohjelmapätkä, jolla

tallennetaan editoitujen rivien indeksejä. Metodissa käytetty aValue muuttuja on setValueAt()-metodille kutsussä välitetty solun sisältö.

```
...
//Checking that the edited row isn't filter row:
if( !( row == 0 && this.isFilterRowVisible() ) ) {
     // if row isn't amongst edited...
     if( bAddedThroughEditor && !alEditedRows.contains( objModel.getRelativeIndex(row) ) ) {
         String newValue = "";
         // getting new value
         if( aValue instanceof javax.swing.JComponent ) {
              newValue = String.valueOf( this.getVisibleValueAt(row, column) );
              // A selected index is also saved from JComboBox: appending selected index too
              if( aValue instanceof javax.swing.JComboBox )
                   newValue += "_" + ((javax.swing.JComboBox)aValue).getSelectedIndex();
         }
         else
              newValue = (String) aValue;

         // if value is a new compared to old one
         if( !newValue.equals(strOriginalValue) ) {
              alEditedRows.add( objModel.getRelativeIndex(row) );
              fireVisibleDataChanged(this, row, column );
         }
     }
}
...
```

Implementoijalla on mahdollisuus määrittää korostetaanko Taulukkonäkymässä editoidut rivit vai ei. Huolimatta siitä onko korostus asetettu päälle vai pois editoituja rivejä tallennetaan aina. Korostuksen päälle asettaminen vaan aiheuttaa editoitujen rivien maalaamisen.

Implementoijan on myös mahdollista halutessaan reagoida rivin lisäämiseen editoitujen joukkoon. Editoidun rivin lisäämisen yhteydessä kutsuttava fireVisibleDataChanged()-metodi laukaisee erillisen tapahtuman: VisibleDataChangedEventin. Lisäämällä VisibleDataChangedListener-kuuntelijan Taulukkonäkymälle ja toteuttamalla sen implementoija voi reagoida tähän tapahtumaan haluamallaan tavalla. Rajapinnasta löytyy kuuntelijan asettamisen lisäksi metodit ominaisuuden hallintaan, värin asettamiseen sekä kerättyjen editoitujen rivien tyhjentämiseen.

Dokumentin kirjoittamisen tässä vaiheessa editoitujen rivien korostaminen on

käytössä yhdessä käyttöliittymäkomponentissa, joka sisältää useita Taulukkonäkymiä. Käytössä on tullut esille piirre, johon puututaan heti kun aikaa löytyy. Rivi ei nimittäin kirjaudu editoitujen joukkoon lennosta vaan solun, jonka sisältöä on editoitu, valinta täytyy poistua ennen kuin muutos voidaan havaita.

4.2 Riskien toteutuminen ja ilmenneet ongelmat

Aikaisemmin esitellyistä toteutetuista ominaisuuksista ainoastaan aktiivikomponenttien lisääminen Taulukonäkymään aiheutti suuria viivästyksiä. Kuten luvussa 4.1.1 Aktiivikomponentit Taulukkonäkymään kävi ilmi, aktiivikomponenttien piirto-ongelmia ratkaistaessa toteutui kummatkin mahdollisiksi arvioidut riskit: jumiutuminen ja rakenteen syvyys. Ellei riskejä olisi otettu huomioon aikataulujen suunnittelussa olisi projektia odotettu valmiiksi lähes kahta viikkoa liian aikaisin.

Toteutuksen edetessä eteen tuli muitakin tosin paljon vähäpätöisempiä ongelmia. Näistä hyvänä esimerkkinä editoitavuuksia toteutettaessa ohjelmalliset avainarvojen lisäämiset, joita ei ensin ymmärretty toteuttaa jokaiseen tarvittavaan ohjelmakoodin osaan. Tästä aiheutui käsittämättömiä ongelmia, kun ominaisuutta päästiin kattavasti testaamaan. Samoin implementoijan tekemät korostukset aiheuttivat samankaltaisen toteutuksen vuoksi hyvin samantyyppisiä ongelmia. Korostuksia testattaessa esimerkiksi uutta saraketta lisättäessä, eivät korostukset seuranneet alkuperäisiä soluja. Tämän tyyppisiä ajattelemattomuuteen liittyviä ongelmia oli todella paljon, mutta niihin ratkaisu oli onneksi helppo löytää, eikä niillä ollut huomattavaa vaikutusta projektin aikatauluihin.

4.3 Testaus

Toteutuksen edetessä testaaminen eteni toteutuksen rinnalla ja kehitys oli suhteellisen iteratiivistä. Toiminnallisuudet toteutettiin yksi kerrallaan. Toteuttaja testasi kunkin toiminnallisuuden sen valmistuttua kattavasti tarkoitusta varten rakennetussa testipenkissä. Tämän jälkeen toteuttaja korjasi toiminnallisuudessa

mahdollisesti ilmenneitä virheitä kunnes se vastasi vaatimuksia. Toteuttaja testasi toiminnallisuuksia myös toteutuksen lomassa, jotta voitiin olla varmoja, että ollaan menossa oikeaan suuntaan. Lopussa, kun Taulukkonäkymän koettiin olevan valmis, se otettiin käyttöön yhdessä käyttöliittymässä. Käyttöönotossa päästiin testaamaan Taulukkonäkymää todellisessa käyttökohteessa ja tässäkin tapauksessa ilmenneet virheet korjattiin vaatimuksia vastaaviksi. Tässä testausvaiheessa implementoijan vaatimukset olivat huomattavasti suuremmassa roolissa, kuin toiminnallisuuksia testattaessa, koska yksittäiset toiminnallisuudet testattiin lähinnä vain toiminnallisia vaatimuksia vastaan.

Taulukkonäkymälle ei toteutusvaiheessa tehty yksikkötestausta, koska se katsottiin kokonaisuuden kannalta merkityksettömäksi sekä liian aikaavieväksi. Kriittisiä rajapintoja testattiin ennen käyttöönottoa, jotta voitiin varmistua niiden toimivuudesta. Kriittisiksi rajapinnoiksi luokiteltiin valittujen rivien hakeminen, näkyvän sisällön hakeminen ja asettaminen solusta, solun komponentin hakeminen ja asettaminen, tapahtumien lähettäminen, rivien ja sarakkeiden lisääminen sekä kaikki rakentajat. Kriittisiä näistä mainituista rajapinnoista tekee lajittelussa tapahtuva indeksien muuntaminen, suodatusrivin käyttö sekä piirron onnistumisen varmistaminen siten, että käyttäjä saa aina oikeaa tietoa.

Java-ajoympäristön tulee vaatimustensa mukaan toimia samalla tavalla riippumatta käyttöjärjestelmästä, joten käyttöjärjestelmästä riippuvia testejä ei katsottu tarpeelliseksi. Sen sijaan Taulukkonäkymää testattiin kattavasti niissä käyttökohteissa käyttöliittymäkomponenttina, joihin se oli ensisijaisesti suunniteltu. Testaus tapahtui käyttöönoton yhteydessä ja tulokset olivat hyviä lähinnä, koska pahimmat virheet oli korjattu ensimmäisen käyttöönoton yhteydessä. Oli myös yllättävää huomata, kuinka paljon vähemmän koodirivejä Taulukkonäkymän käyttö vaati verrattuna edeltäjäänsä. Joissain tapauksissa 20 ohjelmariviä voitiin korvata kahdella tai kolmella rivillä. Luvussa 5 esitellään Taulukkonäkymää käyttökohteissaan hieman tarkemmin.

5 LOPPUTUOTTEEN ARVIOINTIA

Tämä luku tarjoaa tiiviin katsauksen valmiiseen Taulukkonäkymään ja tähän mennessä syntyneisiin jatkokehitysideoihin.

5.1 Taulukkonäkymän käyttöönotto

Koska DokuMentori Oy:ssä käytetään IDE:nä NetBeans 5.0:aa ja koska Taulukkonäkymä toteuttaa onnistuneesti JavaBean-arkkitehtuuria, voidaan Taulukkonäkymä lisätä graafisiin komponentteihin sen pääluokan (TableView) työkaluvalikon kautta. Tämän jälkeen Taulukkonäkymä on lisättävissä mihin tahansa käyttöliittymään raahaamalla se haluttuun paikkaan IDE:n suunnittelunäkymässä. Tämän jälkeen komponentin rajapinta on nähtävissä IDE:ssä käyttöliittymän ohjelmointinäkymässä pikakomennoilla. Myös rajapintojen kommentit näkyvät samalla. Tämä tekee komponentin käyttöönoton erittäin helpoksi.

Ensimmäistä kertaa Taulukkonäkymää käyttävä implementoija voi halutessaan tutustua tarkemmin sen käyttöön tarkoitusta varten rakennetussa demossa, jonka ohjelmakoodi on kaikkien halukkaiden saatavilla. Yleisesti tällaisessa tilanteessa implementoija kuitenkin pyytää komponentin kehittäjää kertomaan ne seikat, jotka hän haluaa tietää.

5.2 Vaatimusten täyttyminen

Käyttöönoton yhteydessä oli nähtävissä, kuinka muuntautumiskykyinen Taulukkonäkymästä tuli. Kahdessa eri käyttöliittymässä Taulukkonäkymä saattoi olla aivan eri näköinen. Samalla käyttö implementoijan näkökulmasta on todella helppoa ja joustavaa. Pitkäaikainen käyttö tuo varmasti esille uusia tarpeita ja ongelmia, mutta ainakin alkumetreillä Taulukkonäkymä vaikuttaa lunastaneen vaatimuksensa. Implementoijat ovat olleen tyytyväisiä rajapintojen selkeyteen, vaadittujen toiminnallisuuksien toimivuuteen ja ulkoasun monipuolisuuteen. Toimivuutensa

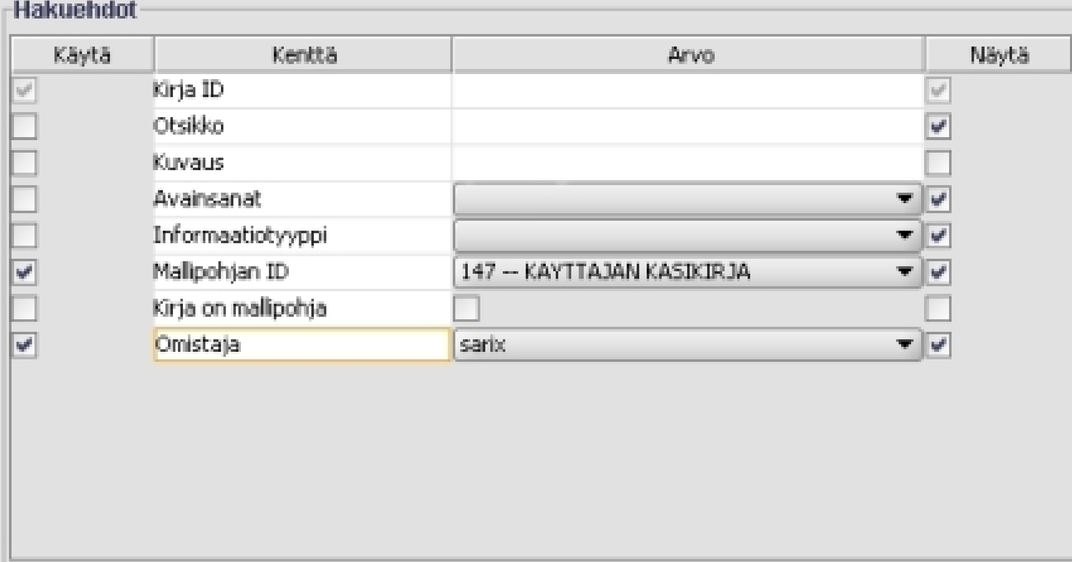
lisäksi Taulukkonäkymä tuo myös väriä käyttöliittymiin ja parantaa taulukoiden luettavuutta huomattavasti.

5.3 Soveltuminen käyttökohteisiin

Vaatimukset jaettiin luvun 2.4.1 alla käyttökohteittain, joista kuhunkin Taulukkonäkymän odotetaan taipuvan. Tässä luvussa on esitelty Taulukkonäkymää sen kolmessa käyttökohteessa käyttöliittymäkomponenttina DokuPort™-sovelluksessa. Nämä kolme esiteltävää käyttökohdetta valittiin niiden havainnollisuuden vuoksi. Niistä käy ilmi Taulukkonäkymän monipuolisuus sekä muuntautumiskyky.

Hakuehtojen asettaminen

Taulukkonäkymä otettiin käyttöön useassa hakukäyttöliittymässä esittämään hakuehtoja. Alla on kuvankaappaus Taulukkonäkymästä komponenttina, jolla voi asettaa hakuehdot haettaessa kirjoja DokuPort™-sovelluksessa.



Kuva 7 Taulukkonäkymä valintakomponenttina

Hakutuloksen esittäminen

Edellä on esitetty kolme kuvankaappausta samasta hakutuloksesta. Haussa on etsitty kaikkia moduuleita, joihin käyttäjällä on oikeus. Kuvankaappauksista käy ilmi lajittelun näkyvyys sekä se, kuinka automaattiset korostukset todella tekevät taulukosta luettavamman.

Haku 1

Moduuli ID	Otsikko	Kuvaus	Avainsanat	.
43	Johdanto	DokuPortin käyttöohje: johdanto		...
44	Käytetyt termit	DokuPortin käyttöohje: käytetyt termit		...
45	Käyttöliittymä	DokuPortin käyttöohje: käyttöliittymä	Käyttöliittymä	...
46	Sisäänkirjautuminen	DokuPortin käyttöohje: sisäänkirjautuminen	Käyttöliittymä	...
47	Kirjan rakenne	DokuPortin käyttöohje: kirjan rakenne	Käyttöliittymä	...
48	käännöstesti			...
49	käännöstesti 2			...
50	Käyttöliittymän kuvaus	DokuPortin käyttöohje: käyttöliittymän kuvaus	Käyttöliittymä	...
51	Työkalupalkki	DokuPortin käyttöohje: työkalupalkki	Käyttöliittymä	...
52	Sisällönhallinta	DokuPortin käyttöohje: sisällönhallinta		...
53	Tehtävälista	DokuPortin käyttöohje: tehtävälista	Käyttöliittymä	...
54	Perushaku	DokuPortin käyttöohje: perushaku	Käyttöliittymä	...
55	Moduulin haku	DokuPortin käyttöohje: moduulin haku	Käyttöliittymä	...
56	Kirjan haku	DokuPortin käyttöohje: kirjan haku	Käyttöliittymä	...
57	Uloskuittaus	DokuPortin käyttöohje: uloskuittaus	Käyttöliittymä	...
58	Sisäänkuittaus	DokuPortin käyttöohje: sisäänkuittaus	Käyttöliittymä	...
59	Informaatiokortti	DokuPortin käyttöohje: informaatiokortti	Käyttöliittymä	...
60	Versiohistoria	DokuPortin käyttöohje: versiohistoria	Käyttöliittymä	...
61	Metatietopuut	DokuPortin käyttöohje: metatietopuut	Käyttöliittymä	...
62	Moduulin tiedostot	DokuPortin käyttöohje: moduulin tiedostot	Käyttöliittymä	...
63	Versiointi	DokuPortin käyttöohje: versiointi	Käyttöliittymä	...
64	Käyttäjäasetusten hallinta	DokuPortin käyttöohje: käyttäjäasetusten hallinta		...
65	Käyttäjäasetusten hallinta	DokuPortin käyttöohje: käyttäjäasetusten hallinta	Käyttöliittymä	...
66	Editorin Avaajan hallinta	DokuPortin käyttöohje: editorin avaajan hallinta	Käyttöliittymä	...
67	Järjestelmänvalvojan toiminnot	DokuPortin käyttöohje: järjestelmänvalvojan toiminnot		...

Kuva 8 Automaattiset korostukset Taulukkonäkymässä

Haku 1

Moduuli ID	Otsikko ▲	Kuvaus	Avainsanat	.
117				...
146			Avainsana,Käyttöliittymä	...
147			Avainsana,Sisällön hallinta	...
148			Avainsana,Käyttöliittymä	...
207	Attribuutit	DokuPortin käyttöohje: attribuutit	Käyttöliittymä	...
206	Avainsanat	DokuPortin käyttöohje: avainsanat	Käyttöliittymä	...
115	DP_admin_kayttajaasetusten_hallinta	DokuPortin käyttöohjeen kuva DP_admin_kayttajaasetu...	Käyttöliittymä	...
116	DP_admin_kayttajienhallinta	DokuPortin käyttöohjeen kuva DP_admin_kayttajienhalli...	Käyttöliittymä	...
198	DP_avaa_kirja	DokuPortin käyttöohjeen kuva DP_avaa_kirja.jpg	Käyttöliittymä	...
118	DP_edistynyt_haku	DokuPortin käyttöohjeen kuva DP_edistynyt_haku.jpg	Käyttöliittymä	...
120	DP_editorin_avaajan_hallinta	DokuPortin käyttöohjeen kuva DP_editorin_avaajan_halli...	Käyttöliittymä	...
105	DP_informaatiokortti_lisää_attribuutin_arvo	DokuPortin käyttöohjeen kuva DP_informaatiokortti_lisä...	Käyttöliittymä	...
104	DP_informaatiokortti_muokkaa_attribuutin_arvoa	DokuPortin käyttöohjeen kuva DP_informaatiokortti_muo...	Käyttöliittymä	...
89	DP_kayttajaasetusten_hallinta	DokuPortin käyttöohjeen kuva DP_kayttajaasetusten_h...	Käyttöliittymä	...
95	DP_kayttoliittyma_alussa_fin	DokuPortin käyttöohjeen kuva DP_kayttoliittyma_alussa...	Sisällön hallinta	...
81	DP_kirjan_haku	DokuPortin käyttöohjeen kuva DP_kirjan_haku.jpg	Käyttöliittymä	...
90	DP_kirjan_rakenne	DokuPortin käyttöohjeen kuva DP_kirjan_rakenne.jpg	Käyttöliittymä	...
113	DP_kirjan_rakenne_muokkaaminen	DokuPortin käyttöohjeen kuva DP_kirjan_rakenne_muok...	Käyttöliittymä	...
114	DP_kirjan_rakenne_muokkaaminen_2	DokuPortin käyttöohjeen kuva DP_kirjan_rakenne_muok...	Käyttöliittymä	...
193	DP_lisaa_kirjaan	DokuPortin käyttöohjeen kuva DP_lisaa_kirjaan.jpg	Käyttöliittymä	...
191	DP_lisaa_uloskuittaukseen	DokuPortin käyttöohjeen kuva DP_lisaa_uloskuittauksee...	Käyttöliittymä	...
197	DP_lisaa_versiointiin	Doku Portin käyttöohjeen kuva DP_lisaa_versiointiin.jpg	Käyttöliittymä	...
86	DP_metatietopuut	DokuPortin käyttöohjeen kuva DP_metatietopuut.jpg	Käyttöliittymä	...
201	DP_moduuli_avattu_rakenne	DokuPortin käyttöohjeen kuva DP_moduuli_avattu_rake...	Käyttöliittymä	...
203	DP_moduuli_ei_rakennetta	DokuPortin käyttöohjeen kuva DP_moduuli_ei_rakennett...	Käyttöliittymä	...

Kuva 9 Tekstiin verrattavan sisällön lajittelu

Haku 1

Moduuli ID ▼	Otsikko	Kuvaus	Avainsanat	.
218	DP_tietokannanhallinta_attribuutti	DokuPortin käyttöohjeen kuva DP_tietokannanhallinta_a...	Käyttöliittymä	...
217	DP_tietokannanhallinta_kohdemedia	DokuPortin käyttöohjeen kuva DP_tietokannanhallinta_k...	Käyttöliittymä	...
216	DP_tietokannanhallinta_sisältötyyppi	DokuPortin käyttöohjeen kuva DP_tietokannanhallinta_si...	Käyttöliittymä	...
215	DP_tietokannanhallinta_kirjaluokka	DokuPortin käyttöohjeen kuva DP_tietokannanhallinta_ki...	Käyttöliittymä	...
214	DP_tietokannanhallinta_informaatiotyyppi	DokuPortin käyttöohjeen kuva DP_tietokannanhallinta_in...	Käyttöliittymä	...
212	DP_tietokannanhallinta_avainsanat	DokuPortin käyttöohjeen kuva DP_tietokannanhallinta_a...	Käyttöliittymä	...
211	Kohdemedia	DokuPortin käyttöohje: kohdemedia	Käyttöliittymä	...
210	Sisältötyyppi	DokuPortin käyttöohje: sisältötyyppi	Käyttöliittymä	...
209	Kirjaluokka	DokuPortin käyttöohje: kirjaluokka	Käyttöliittymä	...
208	Informaatiotyyppi	DokuPortin käyttöohje: informaatiotyyppi	Käyttöliittymä	...
207	Attribuutit	DokuPortin käyttöohje: attribuutit	Käyttöliittymä	...
206	Avainsanat	DokuPortin käyttöohje: avainsanat	Käyttöliittymä	...
205	Tietokannanhallinta	DokuPortin käyttöohje: tietokannanhallinta		...
203	DP_moduuli_ei_rakennetta	DokuPortin käyttöohjeen kuva DP_moduuli_ei_rakennett...	Käyttöliittymä	...
202	DP_moduuli_suljettu_rakenne	DokuPortin käyttöohjeen kuva DP_moduuli_suljettu_rake...	Käyttöliittymä	...
201	DP_moduuli_avattu_rakenne	DokuPortin käyttöohjeen kuva DP_moduuli_avattu_rake...	Käyttöliittymä	...
200	DP_nayta_metatiedot_kirja	DokuPortin käyttöohjeen kuva DP_nayta_metatiedot_kir...	Käyttöliittymä	...
199	DP_uusi_kirja	DokuPortin käyttöohjeen kuva DP_uusi_kirja.jpg	Käyttöliittymä	...
198	DP_avaa_kirja	DokuPortin käyttöohjeen kuva DP_avaa_kirja.jpg	Käyttöliittymä	...
197	DP_lisaa_versiointiin	Doku Portin käyttöohjeen kuva DP_lisaa_versiointiin.jpg	Käyttöliittymä	...
196	DP_vapauta	Doku Portin käyttöohjeen kuva DP_vapauta.jpg	Käyttöliittymä	...
195	DP_nayta_metatiedot_moduuli	DokuPortin käyttöohjeen kuva DP_nayta_metatiedot_m...	Käyttöliittymä	...
194	DP_poista_kirjasta	DokuPortin käyttöohjeen kuva DP_poista_kirjasta.jpg	Käyttöliittymä	...
193	DP_lisaa_kirjaan	DokuPortin käyttöohjeen kuva DP_lisaa_kirjaan.jpg	Käyttöliittymä	...
192	DP_uusi_moduuli	DokuPortin käyttöohjeen kuva DP_uusi_moduuli.jpg	Käyttöliittymä	...

Kuva 10 Numerotiedon lajittelu

Käyttäjien listaaminen ja valitseminen

Seuraavassa esitellään Taulukkonäkymää käyttökohteessa, jossa on listattu käyttäjiä. Kyseisessä käyttökohteessa käyttäjiä on tarkoitus valita taulukosta, joten suodatusrivi koettiin paremmaksi vaihtoehdoksi kuin lajittelu, jolla ei voi rajata etsittävää joukkoa. Kuvankaappauksilla on pyritty havainnollistamaan suodatuksen tehokkuus, kun rivejä on paljon. Käyttäjien nimiä on muutettu kuvankaappausta varten, mutta volyymi vastaa todellisten käyttäjien määrää eräässä DokuPort™-sovelluksessa.

ID	Käyttäjätunnus	Nimi	Aktiivinen	Käyttäjäryhmä	tyyppi
					KAYTTAJA
0	default		☐		KÄYTTÄJÄ
1	mattix	Matti Meikäläinen	☑	Administrators	KÄYTTÄJÄ
2	maijax	Maija Meikäläinen	☑		KÄYTTÄJÄ
3	eskox	Esko Pajunen	☑	Administrators,Group 1	KÄYTTÄJÄ
5	teemux	Teemu Koivu	☑	Writers	KÄYTTÄJÄ
6	keijox	Keijo Kotipeli	☑	Writers	KÄYTTÄJÄ
7	maaretx	Maaret Hukkapää	☐	Writers	KÄYTTÄJÄ
9	reijox	Reijo Rupunen	☑		KÄYTTÄJÄ
10	koox	Kapteeni Kosmos	☐	Group 1,Group 4	KÄYTTÄJÄ
11	uunox	Uuno Turmanen	☑		KÄYTTÄJÄ
12	peelox	Peelo Haamu	☐		KÄYTTÄJÄ
13	tuomox	Tuomo Teippaaja	☑	Writers,Administrators,Group 2	KÄYTTÄJÄ
14	aamux	Aamu Saraste	☐	Subcontractors,Administrators,Group 1	KÄYTTÄJÄ
15	veikkox	Veikko Virtanen	☐	Group 3	KÄYTTÄJÄ
16	namex	Name Here	☑	Customers,Group 2	KÄYTTÄJÄ
17	sailax	Saila Suokukko	☑		KÄYTTÄJÄ
18	ollix	Olli Osaaja	☑	Customers,Administrators	KÄYTTÄJÄ
19	jarix	Jari Jorma	☐	Customers,Group 1	KÄYTTÄJÄ
20	newx	New User	☐	Group 1	KÄYTTÄJÄ
21	johnx	John Doe	☑	Customers,Group 1	KÄYTTÄJÄ
22	riittax	Riitta Herkku	☐	Customers	KÄYTTÄJÄ
23	arix	Ari Armoton	☐		KÄYTTÄJÄ
24	testx	Tests Again	☐	Customers	KÄYTTÄJÄ

Kuva 11 Suodatusrivi Taulukkonäkymässä

ID	Käyttäjätunnus	Nimi	Aktiivinen	Käyttäjäryhmä	tyyppi
		ma	Kyllä		KAYTTAJA
1	mattix	Matti Meikäläinen	☑	Administrators	KÄYTTÄJÄ
2	maijax	Maija Meikäläinen	☑		KÄYTTÄJÄ
11	uunox	Uuno Turmanen	☑		KÄYTTÄJÄ
32	markox	Marko Moilanen	☑		KÄYTTÄJÄ
34	raimox	Raimo Reima	☑		KÄYTTÄJÄ
35	katselmax	Katselma Testi	☑	Writers,Group 1	KÄYTTÄJÄ
40	terox	Tero Tormaaja	☑		KÄYTTÄJÄ
43	tuomax	Tuomas Teuva	☑		KÄYTTÄJÄ
52	markkux	Markku Korva	☑		KÄYTTÄJÄ

Kuva 12 Suodatuksen tehokkuus Taulukkonäkymässä

5.4 Jatkokehitysideoita

Koska näyttävyys ja helppolukuisuus ovat tärkeitä ominaisuuksia mille tahansa ohjelmistolle ei ole ihme, että ensimmäinen jatkokehitysidea tukee näistä piirteistä kumpaakin. Taulukkonäkymään on toivottu otsikkosaraketta, jossa voidaan esittää sisältöä kuvaavaa ikonia. Tämä helpottaisi esimerkiksi hakutulosten hahmottamista.

Muut kehitysideat painottavat kosmeettisuuteen. Jos esimerkiksi teksti ei mahdu näkymään solussa kokonaisuudessaan näytetään tekstistä pätkä ja loppu korvataan kolmella pisteellä. Tämä on Javan sisäistä toimintaa. Paljon on toivottu, että esimerkiksi vietäessä kursori solun päälle näytettäisiin koko sisältö pienessä ponnahdusikkunassa solun päällä silloin, kun koko teksti ei mahdu soluun. Samaan kategoriaan kuuluu ajatus, että alasvetolistojen ponnahdusvalikkoa laajennettaisiin siten, että se olisi leveydeltään aina pisimmän vaihtoehdon levyinen.

Jatkokehitykseen menee myös kahden ominaisuuden toiminnan parantaminen: editoitujen rivien ilmaiseminen sekä automaattinen lajittelu riviä lisättäessä. Kummatkin näistä tarpeista tulivat esille ominaisuuksien toteutusta esiteltäessä.

6 YHTEENVETO

Projektina Taulukkonäkymä oli todella onnistunut. Yhtenä onnistumisen mittareista voidaan pitää aikataulujen hyvää paikkaansapitävyyttä koko projektin kestossa. Toinen mittari, jolla voidaan todeta projektin onnistuneen, on lopputuotteen erinomainen vastaavuus määriteltyihin vaatimuksiin. Käyttöönottovaiheessa Taulukkonäkymän todettiin olevan myös laadullisesti hyvä: rajapinnat olivat hyvin kommentoitu ja yksiselitteisiä. Myös ohjelmakoodi ja rakenne olivat selkeät. Tulevaisuudessa Taulukkonäkymää tullaan hyvin todennäköisesti laajentamaan usealla ominaisuudella, joista tällä hetkellä varmoja ovat luvussa 5.4 Jatkokehitysideoita esitellyt.

LÄHDELUETTELO

Painetut lähteet

1 Englander, Robert, Developing Java Beans. O'Reilly 1997. 1. luku: Introduction

3 Loippo, Eki , Rakenteisen informaation hallintakomponentti. Tutkintotyö. Tampereen ammattikorkeakoulu. Ohjelmistotekniikka. Tampere 2006. 39 sivua + 1 liitesivu.

Painamattomat lähteet

7 Koskentausta, Kivioja, DokuMentorin työhönotto-opas 2005. Sivu 2.

Sähköiset lähteet

2 DokuMentori Oy [www-sivu]. [viitattu 10.12.2006] Saatavissa: http://www.dokumentori.fi/?page=tuotteet&id=0

4 Trail: Collections (The Java™ Tutorials) [www-sivu] [viitattu 10.12.2006] Saatavissa: http://java.sun.com/docs/books/tutorial/collections/index.html

5 QuickSort – Wikipedia, the free encyclopedia [www-sivu] [viitattu 10.12.2006] Saatavissa: http://en.wikipedia.org/wiki/Quicksort

6 JDK 5 Documentation [www-sivu] [viitattu 10.12.2006] Saatavissa: http://java.sun.com/j2se/1.5.0/docs/index.html