TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Teemu Kähkönen

Hakujärjestelmän toteutus Codewitz-projektille

Tutkintotyö, joka on jätetty opinnäytteenä tarkastettavaksi insinöörin tutkintoa varten Tampereella huhtikuussa 2005.

Kähkönen, Teemu	Hakujärjestelmän toteutus Codewitz-projektille
Tutkintotyö	31 sivua
Työn teettäjä	Esa Kujansuu
Huhtikuu 2005	
Hakusanat	haku, codewitz, struts, java

Työn ohjaaja — Ohjelmistotekniikan lehtori Kaj Sundström

Tiivistelmä

Codewitz on Euroopan Unionin Minerva-ohjelman rahoittama projekti, jonka tarkoituksena on tuottaa visuaalista ja vuorovaikutteista sähköistä oppimateriaalia ohjelmistotekniikan opettajille ja opiskelijoille. Tutkintotyön aiheena oli hakujärjestelmän toteuttaminen projektille. Toteutus tehtiin erillisenä moduulina ServerSide Java -tekniikoita ja PostgreSQL-tietokantaa käyttäen.

Kähkönen, Teemu	Material search for Codewitz project
Engineering thesis	31 pages
Commissioning party	Esa Kujansuu
April 2005	
Keywords	search, codewitz, struts, java

Thesis supervisor	Kaj Sundström

Abstract of the theses

Purpose of Codewitz project is to produce interactive and visual learning aid for teachers and students in the field of software engineering. It is a European Union Minerva programme funded project. Target of this thesis was to produce a search function to be used for finding these learning aid files from Codewitz website material bank section. The search function was developed as a separate module using ServerSide Java techniques and PostgreSQL database.

Alkusanat

Olen jakanut tämän tutkintotyöraportin kolmeen pääalueeseen. Näistä ensimmäisessä käsittelen erilaisia työssä käytettyjä työkaluja ja tekniikoita. Ennen työn aloittamista minulla ei ollut kokemusta palvelinpuolen Java-ohjelmoinnista, joten useimmat näistä olivat minulle uusia tuttavuuksia ja erilaisten tekniikoiden runsaus aiheutti päänvaivaa. Työn edetessä näiden ohjelmien ja tekniikoiden sijoittuminen järjestelmään tuli kuitenkin selkeäksi ja kokonaisuus valkeni pala palalta.

Saatuani selvitettyä itselleni Struts-sovelluksen toimintaperiaatteen, oli aika suunnitella hakujärjestelmän rakenne. Aluksi niin monimutkaiselta tuntunut Struts tarjosikin yhtäkkiä loogisen kehyksen sovelluksen suunnittelemiselle. Haun suunnittelu -kappaleessa olen käsitellyt tätä vaihetta. Suunnitteluvaiheessa olen myös pyrkinyt ottamaan huomioon muutamia julkiseen käyttöön tulevan ohjelman riskejä.

Hakujärjestelmän toteutus onnistui pääosin Strutsin luokkia pohjana käyttäen. Toteutus-kappaleessa olen käsitellyt suunnitteluvaiheen asioita käytännön kannalta ja toteutusratkaisuja lähdekoodin avulla.

Kokonaisuutena projekti oli opettavainen kokemus palvelinpuolen Javaan tutustumisessa, mutta ennen kaikkea ohjelmistoprojektin eri vaiheiden tuntemisessa.

_____

29.4.2005 Teemu Kähkönen

Sisällys

1 KÄYTETYT TERMIT JA LYHENTEET

API	Application Programming Interface. Ohjelmointirajapinta.
ASCII	American Standard for Information Interchange. 127 merkkiä käsittävä merkistö.
HTML	HyperText Markup Language. Eräs WWW:ssä käytetyistä sivunkuvauskielistä.
HTTP	HyperText Transfer Protocol. WWW:ssä käytetty tiedonsiirtoprotokolla.
JavaBean	Sun Microsystemsin tekemä määrittely, joka käsittelee olioiden välistä vuorovaikutusta.
JDBC	Java DataBase Connectivity. Yhtenäinen ohjelmointirajapinta Javassa tietokantojen käsittelyyn.
Metodi	Javassa käytetty nimitys luokan jäsenfunktiolle.
SQL	Structured Query Language. Tietokannoista tiedonhakuun käytetty kieli.
Unicode	Unicode-konsortion ylläpitämä merkistö ASCIIn tavoin, käsittäen kuitenkin huomattavasti suuremman määrän merkkejä. Unicode sisältää osajoukkonaan myös ASCII-merkistön. http://www.unicode.org/
URL	Universal Resource Locator. Standardi tapa ilmoittaa resurssin osoite. Esimerkiksi WWW-osoitteet ovat tällaisia.
WWW	World Wide Web. Internetin päällä toimiva maailmanlaajuinen järjestelmä, joka käsittää erilaisia tietolähteitä ja resursseja.

2 JOHDANTO

Tutkintotyöni aiheena oli Codewitz-projektin hakujärjestelmän kehittäminen. Projekti toimii pääasiassa WWW-sivustonsa kautta ja tarjoaa oppimateriaalia maailmanlaajuisesti suoraan Internetin yli. Hakujärjestelmä on tarkoitettu kyseisen materiaalin hakuun.

Hakujärjestelmän toteutin erillisenä moduulina, joka on mahdollista lisätä Codewitz-projektin web-sivuston osaksi.

Codewitz on Tampereen ammattikorkeakoulusta alkunsa saanut ohjelmistotekniikan sähköistä oppimateriaalia opettajille ja opiskelijoille tuottava projekti. Projekti saa rahoitusta EU:n etäopetusta tukevalta Minerva-ohjelmalta.

Oppimateriaali on toteutettu Macromedia Flash- ja Shockwave-tekniikoita käyttäen ja se on siten käytettävissä WWW-selaimen avulla suoraan Internetistä tai paikallisesti opettajan tai opiskelijan koneelta.

Oppimateriaali on usein vuorovaikutteista eli opiskelija pääsee vaikuttamaan opetuksen etenemiseen. Myös harjoituskysymyksiä on useassa oppimateriaalissa. /9/

3 KÄYTETYT TYÖKALUT JA TEKNIIKAT

Kuten aiemmin on mainittu, Codewitz-projekti toimii pääasiassa WWW:ssä. Projektin monimutkaisuuden vuoksi pelkkä HTML-tekniikka ei jousta sivuston mittoihin vaan on tarvittu yhtenäinen käyttöliittymä jolla käyttäjät voivat hakea materiaalia, tekijät voivat lisätä sitä ja ylläpitäjä voi huolehtia kokonaisuuden toiminnasta.

Tämä toiminnallisuus on toteutettu Javalla käyttäen palvelinpuolen Java- eli

ServerSide Java -tekniikoita. Tähän tarkoitukseen on valittu Jakarta projektin Struts, joka on riittävä monimutkaisten WWW-projektien toteuttamiseen. Struts on ohjelmistokehys, joka käyttää pohjanaan Servlet- ja JSP-tekniikoita.

Codewitz-projektin toteuttamiseen on tarvittu myös jonkinlainen tapa säilöä muuttuvaa tietoa. Tämän tiedon ja sen käyttäjien määrän vuoksi ei yksinkertainen teksti- tai XML-tiedosto enää riitä, vaan on tarvittu tietokantapalvelin. Tähän tarkoitukseen on valittu PostgreSQL, joka mahdollistaa tiedon pysymisen järjestyksessä monen samanaikaisen käyttäjän ympäristössä. Tietokanta on yhdistetty ohjelmistoon JDBC-rajapinnan kautta, joka on lähes standardi tapa tietokannan käsittelyyn Javalla.

Työssäni käyttämieni työkalujen ja tekniikoiden valinta pohjautui Codewitz-projektin tekemiin valintoihin. Seuraavissa kappaleissa on käsitelty tarkemmin näitä työkaluja ja tekniikoita.

3.1 ServerSide Java

ServerSide Java on yleisnimitys palvelimella ajettaville sovelmille eli servleteille. Tämä käsittää myös JavaServer Pages -tekniikan.

3.1.1 Servlet

Puhtaasti HTML:llä toteutetut sivut näkyvät jokaiselle käyttäjälle samanlaisina. Servleteillä on mahdollista lisätä web-sivuihin käyttäjittäisiä muutoksia ja erilaista toiminnallisuutta eli yleisesti käyttäjän toimien mukaan vaihtuvaa sisältöä. Käyttäjältä servlet-tekniikka ei vaadi mitään normaalin web-selaimen lisäksi, sillä servlet-ohjelmat ajetaan palvelinkoneella ja palvelimen sekä käyttäjän välinen rajapinta ei muutu.

Servletit ovat Javalla kirjoitettuja ja Javan luokkatiedostoiksi käännettyjä ohjelmia. Mikä tahansa Java-ohjelma ei servletiksi kuitenkaan käy, vaan ohjelmasta pitää löytyä servlet-rajapinnan toteuttava luokka. Tässä luokassa oleva julkinen service-metodi toimii koko servlet-ohjelman aloituspisteenä. Tämä service-metodi tutkii saapuneen kyselyn ja ohjaa sen protected service -metodille, joka jäsentelee saapuneen kyselyn ja ohjaa sen vuorostaan vastaavalle metodille. Nämä do-alkuiset metodit käsittelevät pyynnön ja vastaavat siihen kyselyn vaatimalla tavalla.

Käytännössä tämä tarkoittaa HttpServlet-luokasta periytyvän luokan kirjoittamista. HttpServlet-luokassa on oletustoteutukset service-metodille ja do-alkuisille metodeille, joten luokan käyttäjän ei tarvitse toteuttaa kaikkea, vaan ainoastaan tarvitsemansa metodit alla olevan malliesimerkin mukaisesti.

```
import javax.servlet.http.*;

public class OmaServlet extends HttpServlet {

   protected void doGet (HttpServletRequest request,
                         HttpServletResponse response){
      // Täällä käsitellään tulevat GET-pyynnöt.
   }


   protected void doPost (HttpServletRequest request,
                          HttpServletResponse response){
      //Täällä käsitellään tulevat POST-pyynnöt.
   }

   // ...
}
```

Mallitoteutus ottaa vastaan ja käsittelee tulevat GET- ja POST-tyyppiset pyynnöt.

Kaikki tulevien pyyntöjen käsittelijät ovat samantyyppisiä. Ne eivät palauta mitään arvoa ja ottavat kaksi parametria: requestin ja responsen.

Request on HttpServletRequest-tyyppinen Interface, joka sisältää saapuneen HTTP-pyynnön kaikkine saatavilla olevine tietoineen. Nämä tiedot käsittävät pyynnön otsikkotiedot ja kyselylauseen parametrit. Otsikkotiedoista löytyvät esimerkiksi päivämäärä, pyynnön lähettäjän ip-osoite ja istuntotiedot. Kyselylauseen parametrit ovat suoraan url:stä &-merkillä toisistaan erotettuja parametreja, joilla välitetään käyttäjältä tietoa palvelimelle. HttpServletRequest sisältää myös metodit, joilla pyynnön tietoja pääsee hyödyntämään.

Response on HttpServletResponse-tyyppinen rajapinta, joka sisältää pyynnön tekijälle lähtevän vastauksen. Samoin kuin request, response sisältää HTTP-otsikkotiedot. Kyselylauseen parametreja vastaus ei sen sijaan sisällä. Pyyntöön lähtevä vastaus voi olla pyydetty sisältö, virheilmoitus tai uudelleenohjaus.

Mikäli vastaus on virheilmoitus tai uudelleenohjaus, ei siihen tarvitse kirjoittaa sisältöä. Sen sijaan voidaan käyttää HTTP-standardin mukaisia virhekoodeja, joita käyttäjän web-selain ymmärtää. Esimerkiksi asettamalla vastauksen virhekoodiksi 404, näkee käyttäjä selaimensa tuottaman virheilmoituksen, joka ilmaisee, ettei pyydettyä tiedostoa löydy palvelimelta. Uudelleenohjaus sisältää URL:n, jonne käyttäjän selain lähettää uuden pyynnön.

Vastauksen sisältö kirjoitetaan PrintWriter- tai ServletOutputStream-oliolla, joka saadaan responsen getWriter- tai getOutputStream-metodilla. /8/ /4/

3.1.2 JavaServer Pages

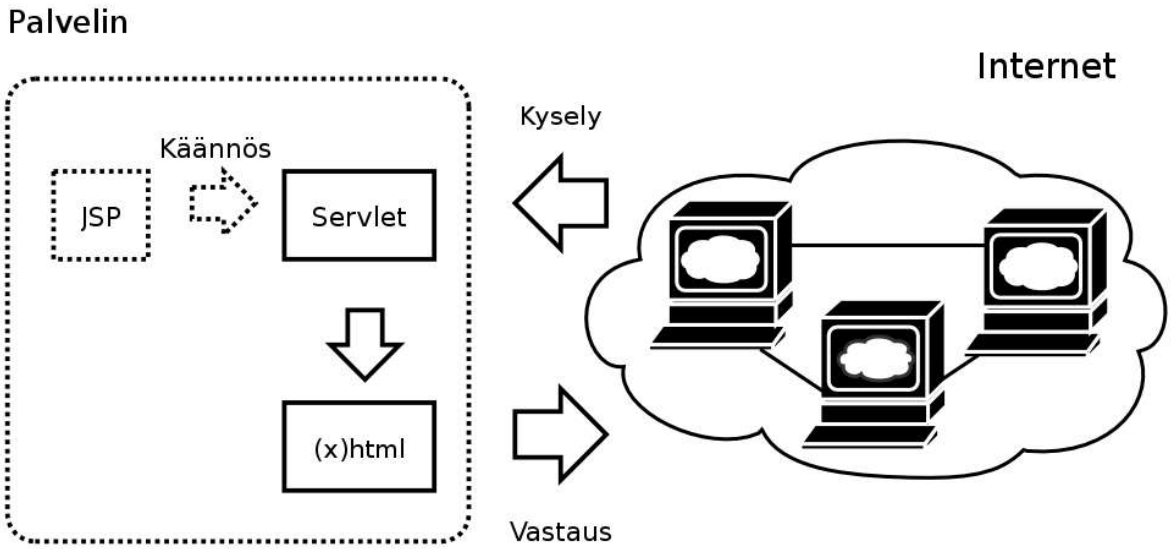
JSP-sivut ovat palvelimella ajettavia Java-ohjelmia servletien tapaan. Loppukäyttäjän kannalta ei ole merkitystä kummalla tekniikalla palvelimella oleva ohjelma on toteutettu. Itse asiassa JSP-sivut käännetään servleteiksi ennen niiden ajoa. Erona näiden kahden tekniikan välillä on siis lähdekoodin

toteutus. Siinä, missä servletit kirjoitetaan suoraan Javan luokiksi ja loppukäyttäjälle näkyvä HTML tulostetaan Java-ohjelmalla, JSP:t kirjoitetaan staattisen HTML:n sisään. Java-koodi erotellaan JSP-sivuissa HTML-koodista kirjoittamalla se omien tagien sisään alla olevan esimerkkilähdekoodin tavoin.

```
<%@ page import="java.util.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html40/strict.dtd">
<html>
<title>Esimerkki</title>
<body>

<p>

<!-- Tasta alkaa dynaaminen osa -->
<%

// Luodaan uusi Date-olio.

Date pvm = new Date();
%>
Nyt on <%= pvm %>

</p>
</body>
</html>
```

Esimerkki tuottaa yksinkertaisen HTML-sivun jossa on tulostettuna teksti "Nyt on" sekä senhetkinen päivämäärä ja kellonaika. Teksti on staattista HTML:ää ja kellonaika sekä päivämäärä saadaan käyttämällä Javan Date-luokkaa. Kaikki "<%"- ja "%>"-merkkien välissä oleva on Java-kieltä.

JSP-sivut käännetään servleteiksi ensimmäisellä pyyntökerralla. Seuraavilla pyynnöillä tätä käännöstä ei enää tarvitse tehdä, vaan voidaan käyttää jo muodostettua servletiä kuvan 2 mukaisesti. Käännöksessä staattinen HTML-osuus kääntyy Javan tulostuskomennoiksi. Näin kaikki JSP-sivun osat ovat Javan lähdekoodina ja ne voidaan siten kääntää Javan luokkatiedostoksi. /7/ /2/

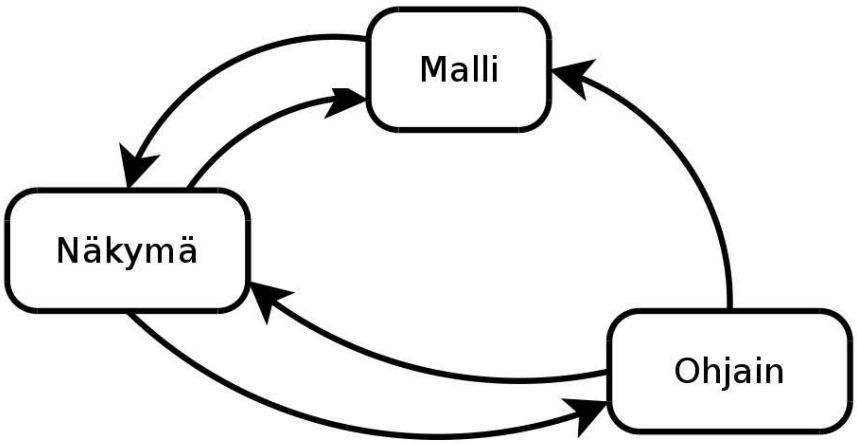


Kuva 1 ServerSide Java -toimintakaavio. Käännös JSP:stä Servletiksi tehdään vain kerran. /2/

3.1.3 Jakarta Struts

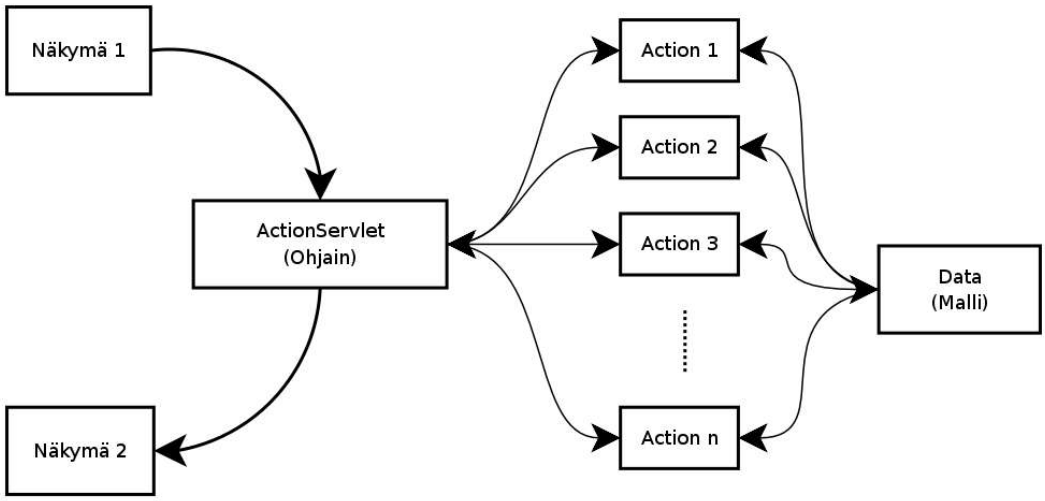
Jakarta-projekti tuottaa ja ylläpitää avoimen lähdekoodin sovelluksia. Projekti koostuu useista aliprojekteista, jotka ovat pääasiassa ohjelmistojen kehittäjille tarkoitettuja apuohjelmia. Projekteista löytyy niin Java-ohjelmien testaukseen kuin suunnitteluunkin tarkoitettuja ohjelmia ja kirjastoja. Palvelinpuolen Java-sovelluskehitys on hyvin edustettuna. Struts-aliprojekti on yksi näistä. Se on ServerSide Java -sovelluksiin tarkoitettu sovelluskehys, joka toteuttaa MVC-suunnitelumallin.

MVC-suunnittelumallin mukaisesti tehtävä sovellus jaetaan kolmeen osaan: Modeliin eli malliin, view:hun eli näkymään sekä Controlleriin eli ohjaimeen. Malli kuvaa ohjelman käsittelemää tietoa. Ohjaimella tätä tietoa muutetaan ja näkymän kautta puolestaan voidaan ohjata ohjainta. Alla oleva kuva 3 selventää MVC-suunnitelumallin toimintaperiaatetta.



Kuva 2 MVC-suunnittelumalli

Struts-projektin MVC-suunnittelumallin toteutus on hiukan yllä olevasta kuvasta poikkeava, mutta samat elementit löytyvät siitäkin. Kuvassa 4 on esitetty tämä versio.



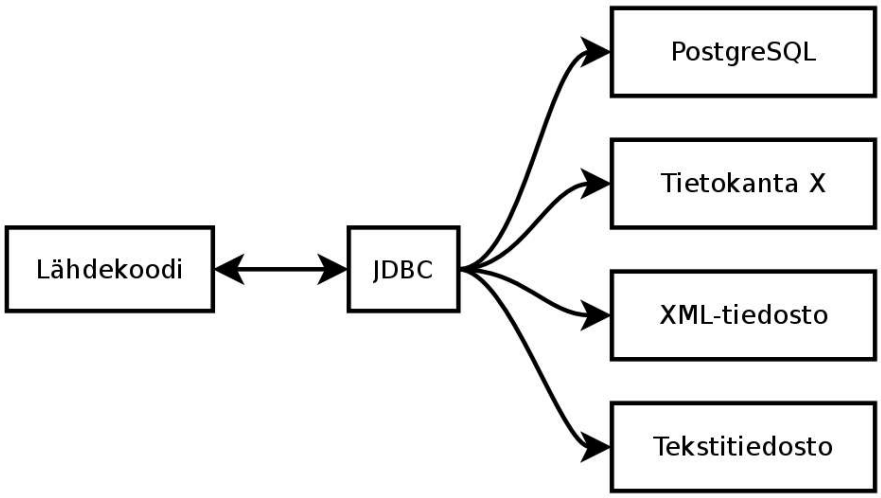
Kuva 3 Jakarta Strutsin tapa toteuttaa MVC-suunnittelumalli. /2/

Näkymät ovat kuvassa vasemmalla ja mallit puolestaan oikealla. Ohjaimena niiden välissä toimii ActionServlet. Toiminta lähtee liikkeelle ensimmäisestä näkymästä, josta annetaan ohjauskäsky ohjaimelle. ActionServlet ajaa ohjauskäskyn mukaisen toiminnon mallille eli ohjelman datalle. Tämä toiminta voi olla esimerkiksi tiedon muuttaminen tai hakeminen. Toiminnon valmistuttua tulokset palaavat ActionServletille, joka näyttää tehdyn toiminnon mukaisen näkymän käyttäjälle. Toinen näkymä voi olla sama tai eri kuin ensimmäinen.

Käytännössä Strutsia pohjanaan käyttävä sovellus toteutetaan periyttämällä oman ohjelman luokat Strutsin luokista. /2/

3.2 JDBC

JDBC on Javassa oleva yhtenäinen ohjelmointirajapinta tietokantojen käsittelyyn. Suurimpana etuna yhtenäisestä rajapinnasta on riippumattomuus jostakin tietystä tietokantajärjestelmästä. Tämä mahdollistaa käytetyn tietokantajärjestelmän vaihtamisen mahdollisimman pienillä lähdekoodin muutoksilla toisen valmistajan tietokantaan tai kokonaan erilaiseen tietolähteeseen. JDBC-toteutuksia löytyy esimerkiksi teksti- ja XML-tiedostoille.



Kuva 4 JDBC:n toimintaperiaate

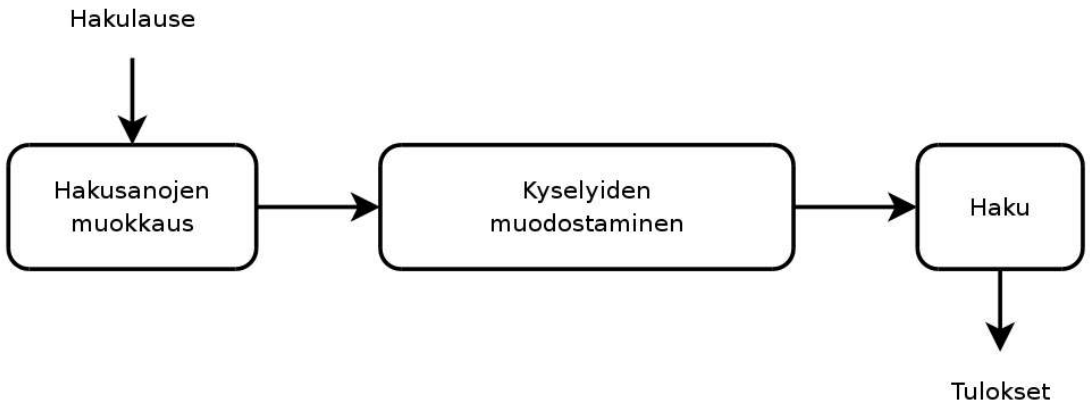
Ohjelmointirajapinnan toteutus on tietokantakohtaista ja niinpä tietokannan valmistaja toimittaa yleensä myös JDBC-toteutuksen tietokannalleen, mutta toteutuksia löytyy markkinoilta myös kolmannen osapuolen valmistamina. Tässä työssä on käytetty PostgreSQL-projektin omaa toteutusta. /5/

3.3 PostgreSQL-tietokanta

PostgtreSQL on pitkän historian omaava relaatiotietokanta. Sen juuret ovat Berkeleyn yliopistossa vuonna 1977 alkunsa saaneessa Ingres-tietokannassa, jonka kehitystä jatkoi Ingres-yhtiö. Vuonna 1986 alkuperäisen Ingresin kehitystä jatkettiin Berkeleyn yliopistossa ja projektin nimeksi annettiin Postgres. Kymmenen vuotta myöhemmin nimi vaihtui nykyiseen muotoonsa, PostgreSQL:ksi, ja projektin kehitys muuttui maailmanlaajuiseksi avoimen lähdekoodin projektiksi. Myös alkuperäinen Ingres-tietokanta on nykyisin avoimen lisenssin alla.

4 HAUN SUUNNITTELU

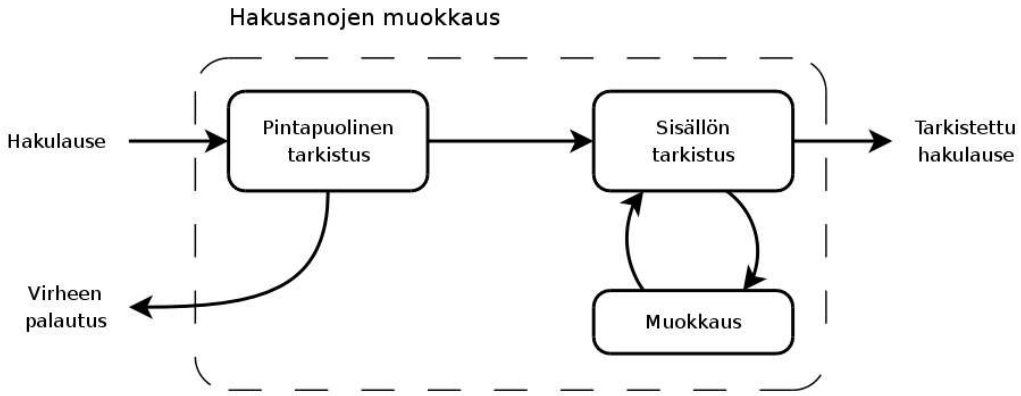
Hakujärjestelmän suunnittelu on jaettu kolmeen vaiheeseen kuvan 6 mukaisesti. Ensimmäisessä vaiheessa käyttäjän syöttämä hakulause tarkistetaan ja muokataan pois vaaralliset merkit. Tämän jälkeen hakulauseesta erotellaan hakusanat ja muodostetaan kyselyt tietokantaa varten. Kolmannessa vaiheessa suoritetaan itse haku ja palautetaan tulokset.



Kuva 5 Haun vaiheet

4.1 Hakusanojen muokkaus

Ennen kyselyiden tekemistä tietokantaan, pitää varmistaa, että käyttäjän syöttämät hakusanat ovat järkeviä eivätkä sisällä mitään hakujärjestelmälle haitallista. Hakusanojen tarkistus ja muokkaus tehdään kahdessa vaiheessa kuvan mukaisesti.



Kuva 6 Hakusanojen muokkausvaiheet.

Hakusanojen saapuessa käyttäjältä palvelimelle tehdään niille pikainen pintapuolinen tarkistus. Tämän vaiheen tarkoituksena on varmistaa että hakumerkkijono täyttää tietyt ulkoiset kriteerit ja että se voidaan lähettää eteenpäin sisällön tarkasteluun. Tarkistettavat asiat ovat hakumerkkijonon pituus ja varmistus ettei ole vastaanotettu tyhjää merkkijonoa. Hakumerkkijonon pituuden tulee olla vähintään yksi tavu, mutta kuitenkin alle yhden kilotavun. Kilotavu on arvioitu riittäväksi käsin syötetylle merkkijonolle vaikka myöhemmin siirryttäisiin käyttämään jotain Unicode-koodausta. Pahimmassa tapauksessa, eli neljä tavua merkkiä kohden, olisi käytettävissä vielä noin 250 merkkiä. Pintapuolisessa tarkistuksessa ei oteta kantaa hakumerkkijonon sisältöön. Mikäli kriteerit eivät täyty, palautetaan virheilmoitus.

Pintapuolisen tarkistuksen jälkeen hakumerkkijono tarkistetaan sisältönsä osalta. Tällöin varmistetaan, ettei hakumerkkijono sisällä mitään haitallisia merkkejä. Tässä vaiheessa ei enää viallisen syötteen sattuessa palauteta virhettä, vaan hakumerkkijono korjataan. Käytännössä mahdolliset

vaaralliset merkit korvataan. Hakumerkkijonon käsittelyn periaatteet on kuvattu tietoturvaa käsittelevässä kappaleessa 3.4.

4.2 SQL-kyselyiden muodostaminen

Kun hakulause on tarkistettu ja hakusanat eroteltu, muodostetaan tietokannan ymmärtämät SQL-lauseet. Hakusanat voisi lisätä yhteen lauseeseen, mutta tällöin tuloslistaan tulisi helposti käyttäjää kiinnostamattomia tuloksia. Kun hakusanat jaetaan ryhmiin ja pudotetaan jokaisella hakukierroksella yksi sana pois, saadaan tarkimmat tulokset ja oleellisimmat tulokset listan kärkeen.

4.3 Haku

Muodostetut kyselyt ajetaan tietokannassa luontijärjestyksessä eli tarkin haku ensimmäisenä. Koska lopuissa hauissa on samoja hakusanoja tarkimpien kanssa, tietokannasta tulee samoja tuloksia useaan kertaan. Lisättäessä näitä tuloslistaan pitää ensin tarkistaa tuloksen olemassaolo, ja mikäli sitä ei löydy, lisätä se listan loppuun.

4.4 Tietoturva

Hakujärjestelmälle tuleva syöte (hakusanat) tulee Internetin yli salaamattomana ja tuntemattomasta lähteestä, joten turvallisuuden vuoksi on hyvä olettaa sen olevan vihamielistä. Hakujärjestelmää pääsee käyttämään vasta kirjauduttuaan Codewitzin sivulle, joten käytännössä vihamielinen syöte lienee harvinaista. Suurin vaara on kirjautumissyötteen (käyttäjätunnus ja salasana) käsittelyssä.

Suurimmat syötteen käsittelyyn liittyvät riskit tässä tapauksessa voidaan

jakaa kahteen osaan: Palvelunestohyökkäyksiin ja ns. injection-hyökkäyksiin. Näistä ensimmäinen tarkoittaa syötteen lähettämistä palvelimelle yli sen suorituskykyrajan, jolloin palvelin ei enää pysty käsittelemään hyötyliikennettäkään. Palvelunestohyökkäysten torjuminen ei onnistu palvelimen suorittamalla ohjelmalla, vaan se pitää hoitaa palomuurin ja erilaisten suodattimien avulla. Tästä syystä palvelunestohyökkäyksiä ei käsitellä tässä dokumentissa.

Injection-hyökkäyksillä tarkoitetaan oikean syötteen sekaan lisättyä syötettä, jolla saadaan syötettä käyttävät ohjelmat ajamaan se. Käytännössä nämä hyökkäykset voidaan jakaa kahteen osaan: SQL-injection ja HTML-injection.

4.4.1 SQL injection

SQL-injection tarkoittaa SQL-kielen lisäämistä syötteeseen sopivasti muotoiltuna, jotta syötteen käsittelijä päästäisi sen läpi tietokantaan asti ja tietokanta suorittaisi sen. Tämä on hyvin mahdollista sen tavan vuoksi jolla tietokannan ymmärtämät SQL-hakulauseet muodostetaan hakusanoista. Hakusanat sijoitetaan SQL-kyselyn where-lauseen sisällöksi.

```
String kysely=new String ("SELECT nimi, osoite FROM Taulu WHERE
nimi='" + <syöte> + "')";
```

Where-lauseen parametri on hakusanojen tapauksessa merkkijono, joten se pitää erottaa '-merkein. Jos syötteessä on kyseinen merkki, päästään "karkaamaan" merkkijonon ulkopuolelle ja voidaan kirjoittaa suoraan SQL-komentoja. Esimerkiksi syötteellä "' or 1=1;" saadaan edellisen esimerkin pohjalta muodostettua seuraavanlainen SQL-kysely. Tällä kyselyllä saadaan koko taulun sisältö, koska ehdot täyttyvät aina.

```
SELECT nimi, osoite FROM Taulu WHERE nimi='' or 1=1;
```

Toinen vaarallinen merkkiyhdistelmä syötteessä on "--", joka SQL:ssä merkitsee kommentin alkua. Esimerkiksi syötteellä " '; DELETE FROM Taulu; -- " saadaan seuraavanlainen SQL-kysely.

```
SELECT nimi, osoite FROM Taulu WHERE nimi=' '; DELETE FROM Taulu; --
';
```

Tässä kyselyssä hakukyselyn tuloksella ei ole merkitystä. Mikäli tietokannan käyttäjällä on kirjoitusoikeudet käytettyyn tauluun, on se tämän jälkeen tyhjä.

Ongelma voidaan korjata joko korvaamalla puolilainaus- ja väliviivamerkit joillakin vastaavilla tai poistamalla ne kokonaan syötteestä. Työssäni päädyin korvaamaan vaaralliset merkit vastaavilla ASCII-koodeilla. Näin tietokanta ei ymmärrä niitä ohjausmerkeikseen, mutta lopullisessa HTML-tuotoksessa selain näyttää ne oikein. /10/

4.4.2 HTML injection

SQL-komentojen lisäksi syöte voi sisältää myös HTML:ää ja selainten tukemia skriptikieliä, esimerkiksi JavaScriptiä. Tällöin vaarassa ei ole tietokantapalvelin, mutta lopullista tuloslistasivua voidaan näin muuttaa, lisätä selaimella suoritettavia skriptejä tai ohjata kokonaan toisaalle. Hakujärjestelmän tapauksessa riskit ovat pienet, sillä tietokantaan ei tallenneta mitään toisille käyttäjille näkyvää. Riskiä vähentää myös se, että syöte välitetään palvelimelle POST-parametrina eli omana parametrinaan HTTP-paketissa. Mikäli syöte välitettäisiin GET-parametrina eli selaimen

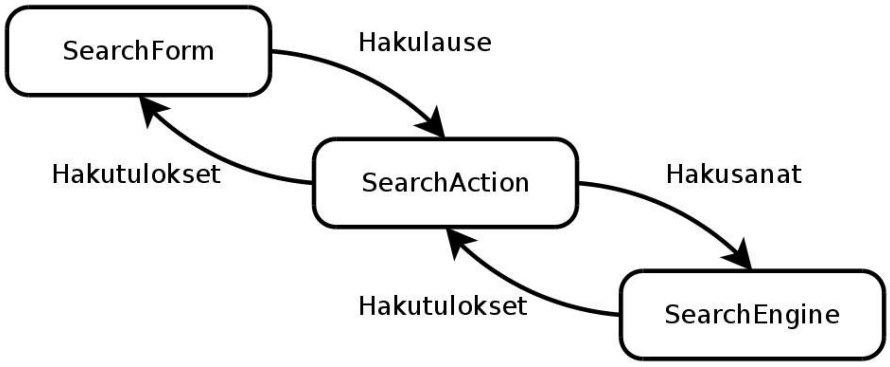
osoiterivillä näkyvänä URL-parametrina lisäisi tämä riskiä näyttää muotoiltu tulossivu myös muiden kuin syötteen kirjoittajan selaimessa.

Ongelmaan on samanlainen ratkaisu kuin SQL:n tapauksessa eli tietyt syötteessä olevat merkit pitää poistaa tai korvata jollakin toisella merkillä. HTML:n tapauksessa vaarallisia merkkejä ovat ns. tagierottimina käytetyt pienempi kuin ja suurempi kuin -merkit sekä HTML:ssä käytetyt kommenttimerkkijonot "<!--" ja "-->".

Mahdollinen ongelmien aiheuttaja saattavat olla myös nk. ohjausmerkit eli ASCII-merkit arvoltaan alle 32. Nämä ovat merkkejä, joilla ei välttämättä ole näkyvää kirjainta tai muuta merkkiä, mutta joiden avulla muotoillaan tekstiä tai joilla on jokin erikoismerkitys. Esimerkiksi rivinvaihto löytyy tältä alueelta samoin kuin NUL-merkki (ASCII-arvo 0), jota käytetään lopetusmerkkinä joissain tapauksissa ja joka useasti aiheuttaa ongelmia. Käytännössä niille ei ole mitään tarvetta hakusanoissa, joten ne voi huoletta poistaa kokonaan. /10/

5 TOTEUTUS

Kappaleessa 3 esitellyt vaiheet on toteutettu pääosin kolmella eri luokalla. Kuvassa 8 on hakujärjestelmän periaattellinen toimintakaavio. Seuraavissa kappaleissa on selostettu näiden luokkien toiminta.



Kuva 7 Eri luokkien sijoittuminen hakujärjestälmässä

5.1 SearchForm

SearchForm on Jakarta Strutsin ActionForm-luokasta periytetty JavaBean. Se sisältää haussa tarvittavat tiedot ja se on tarkoitettu pääasiassa tiedon välitykseen eri komponenttien välillä. Se on myös ensimmäinen komponentti, joka saa käyttäjän antaman syötteen, ja niinpä siellä myös suoritetaan syötteelle pintapuolinen tarkistus ennen sen välittämistä eteenpäin.

Vaikka SearchForm-luokan tarkoitus on pääasiassa tiedonvälitys, sisältää se myös toiminnallisuutta jäsenmuuttujien käsittelyyn. Tämä julkinen rajapinta voidaan jakaa kahteen osaan: ActionForm-luokasta perittyihin metodeihin ja luokan omiin, jäsenmuuttujia käsitteleviin metodeihin.

Näistä jälkimmäiset ovat get-alkuiset metodit jäsenmuuttujien saantiin ja set-alkuiset puolestaan jäsenmuuttujien asettamiseen.

```
public String getSearchstring(){
   return this.searchstring;
}

public void setSearchstring(String searchstring){
   this.searchstring=searchstring;
}

// Jäsenmuuttujat
   private String searchstring=null;
```

Yllä olevan esimerkin mukaisesti getSearchstring-metodi palauttaa hakulauseen sisältävän searchstring-jäsenmuuttujan, kun puolestaan setSearchstring asettaa sille arvon. Myös Struts-ympäristö käyttää näitä metodeja asettaessaan lomakkeelta tulleet arvot.

SearchForm-luokka on perinyt ActionForm-luokasta joitakin metodeja pääasiassa luokasta luodun olion tilan muuttamiseen. Nämä metodit

voidaan jättää SearchForm-luokassa määrittelemättä, jolloin käytetään alkuperäisiä metodeja tai tehdään oma toteutus SearchForm-luokalle. Tässä tapauksessa ainoa uuden toteutuksen saava metodi on validate, jota Struts kutsuu automaattisesti tarkistaakseen jäsenmuuttujien kelpoisuuden. Tätä metodia käytetään siis pikaisen tarkistuksen suorittamiseen.

```
public ActionErrors validate(ActionMapping mapping,
                     javax.servlet.http.HttpServletRequest request){
   ActionErrors errors=new ActionErrors();

   // tarkistetaan hakulause
   if(searchstring==null){
      errors.add(new String("searchstring"),
                 new ActionError("errors.keywords_empty"));
   }
   else if (searchstring.trim().length()<1){
      errors.add(new String("searchstring"),
                 new ActionError("errors.keywords_empty"));
   }
   else if(searchstring.trim().length()>Constants.SEARCHSTR_MAXLEN){
      errors.add(new String("searchstring"),
                 new ActionError("errors.keywords_too_long"));
   }

   // tuleeko tieto perus- vai monipuolisesta hausta
   if(searchtype==null ||
   !(searchtype.equals(Constants.SEARCHTYPE_BASIC) ||
   searchtype.equals(Constants.SEARCHTYPE_ADV))){
      errors.add(new String("searchtype"),
                 new ActionError("errors.invalid_searchtype"));
   }

   // tarkistetaan monipuoliset hakukriteerit
   if(searchtype.equals(Constants.SEARCHTYPE_ADV)){
      [...]
   }
   return errors;
}
```

Metodissa luodaan uusi tyhjä ActionErrors-olio, jonne virheellisen syötteen tapauksessa lisätään ActionError-olioita. Lopussa validate-metodi palauttaa luodun ActionErrors-olion. Mikäli palautunut olio on tyhjä ActionErrors-olio tai arvoltaan null, on tarkistettu syöte kunnossa. Struts ohjaa käyttäjän virhesivulle mikäli syötteestä löytyi virheitä.

5.2 SearchAction

SearchAction on Strutsin Action-luokasta periytetty luokka, jonka tehtävänä on käsitellä käyttäjän syöttämä hakulause ja erotella siitä haussa käytettävät hakusanat. SearchAction toimii myös eräänlaisena rajapintana HTTP-kutsujen ja varsinaisen toimintalogiikan välillä.

SearchAction-luokan julkinen rajapinta on yksinkertainen ja sisältää ainoastaan yhden metodin, execute. Se on Strutsin Action-luokasta peritty ja sille on tehty SearchAction-luokkaan oma toteutus. Kyseistä metodia ei tarvitse missään vaiheessa kutsua itse, vaan Struts kutsuu sitä tarvittaessa. Alla execute metodin toteutus olennaisimmilta osiltaan.

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             javax.servlet.http.HttpServletRequest
                             request,
                             javax.servlet.http.HttpServletResponse
                             response)
        throws SQLException, ClassNotFoundException, Exception{
   [...]
   SearchForm searchForm=(SearchForm)form;
   SearchEngine engine=new SearchEngine();
   LinkedList res=null;
   adv_search=searchForm.getSearchtype().equals
              (Constants.SEARCHTYPE_ADV);

   // tarkistetaan hakulauseen sisältö ja erotellaan se hakusanoiksi
   parsed_searchstr=parseString(searchForm.getSearchstring().trim());
   keywords=parsed_searchstr.split(" ");

   // Tarkistetaan monipuolisen haun sisältö tarvittaessa
   if(adv_search==true){
      [...]
   }
   // suoritetaan haku
   try{
      res=engine.createResults(keywords, adv_search, parsed_author);
   }
   catch(SQLException e){
      [...]
   }
   catch(ClassNotFoundException e){
      [...]
   }
   catch(Exception e){
      [...]
   }
```

```
      // asetetaan tulokset searchFormiin ja ohjataan tulossivulle
      int entry_idx, list_idx=0;
      if(res.size()>0){
         ListIterator iter=res.listIterator(0);
         ResultsEntry entry=(ResultsEntry)iter.next();
         for(; iter.hasNext(); entry=(ResultsEntry)iter.next()){
            entry_idx=entry.getIndex();
            searchForm.addHeader(entry_idx, entry.getHeader());
            searchForm.addExplanation(entry_idx, entry.getExplanation());
            searchForm.addTech_description(entry_idx,
                                           entry.getTechnical_dscr());
            searchForm.addAuthor(entry_idx, entry.getAuthor());
            searchForm.addDate(entry_idx, entry.getDate());
         }
      }
      searchForm.setResultcount(res.size());
      request.setAttribute("results", searchForm);
      return mapping.findForward("results");
   }
```

Metodi saa parametrinaan SearchForm-luokan instanssin, jonka Struts on luonut ja joka sisältää käyttäjän syöttämän hakumerkkijonon. Metodi luo myös SearchEngine-olion ja String-tyyppisen taulukon hakusanoja varten. Tämän jälkeen hakumerkkijono tarkistetaan sisältönsä osalta ja tehdään taulukko hakusanoista. SearchEnginen createResults-metodia kutsumalla suoritetaan haku ja saadaan tulokset paluuarvona. SearchAction käsittelee myös mahdolliset toiminnallisuudesta tulleet poikkeukset. Nämä liittyvät tietokantayhteyksissä oleviin ongelmiin.

ParseString-metodin tehtävänä on siis tarkistaa ja korjata SearchForm:lta tuleva hakumerkkijono. Hakusanataulukko muodostetaan String-luokan split-metodilla.

```
   private String parseString(String searchstring)
      int i, j;
      StringBuffer parsed=new StringBuffer("");

      for(i=0; i<searchstring.length(); i++){
         // Korvaa ASCII-ohjainmerkit
         if(searchstring.charAt(i)<0x20){
            parsed.append('.');
            continue;
         }

         // Korvaa vaaralliset merkit. Lista merkeistä
         // löytyy Constants-luokasta
         for(j=0; j<Constants.SEARCHSTR_INVALID.length; j++){
```

```
            if(searchstring.charAt(i)==Constants.SEARCHSTR_INVALID[j]){
               break;
            }
         }

      if(j==Constants.SEARCHSTR_INVALID.length){
         parsed.append(searchstring.charAt(i));
      }
      else{
         parsed.append(Constants.SEARCHSTR_REPLACE[j]);
      }
      return parsed.toString();
   }
```

Suuri osa metodin toiminnasta koostuu syötteen sisällön käsittelystä. Se voidaan jakaa kahteen osaan: Ohjainmerkkien tarkistukseen ja vaarallisten merkkien tarkistukseen.

Hakumerkkijono käydään silmukassa läpi tavu kerrallaan ja mikäli merkki ei ole ASCII-ohjainmerkki eli sen arvo on suurempi kuin 31 (0x20), tarkistetaan löytyykö sitä kiellettyjen merkkien listalta. Jos merkki on ohjainmerkki, korvataan se pisteellä.

Jos merkki löytyy Constants-luokasta löytyvästä kiellettyjen merkkien taulukosta SEARCHSTR_INVALID, korvataan se SEARCHSTR_REPLACE-taulukosta löytyvällä vastineella.

```
public final static char[] SEARCHSTR_INVALID = {'\'', '-', '<', '>'};
public final static String[] SEARCHSTR_REPLACE = {"&#39;", "&#45;",
"&lt;", "&gt;"};
```

Yllä on molemmat taulukot esiteltynä. Korvaustaulukosta löytyvät &-alkuiset merkkijonot ovat selainten ymmärtämiä HTML-entiteettejä. Molempien taulukoiden pitää olla samankokoisia, sillä korvattava sisältö valitaan suoraan ensimmäisen taulukon indeksin perusteella.

Koska Javan String-muotoiset merkkijonot ovat staattisia, on muokkaukseen käytetty StringBuffer-luokkaa. Näin merkkijonon muokkaus on tehokkaampaa kuin String-luokalla toteutettuna.

5.3 SearchEngine

SearchEngine-luokan tehtävänä on luoda käytettyyn tietokantaan sopivat hakulauseet saamistaan hakusanoista, suorittaa haku ja palauttaa tulokset.

Vaikka SearchAction vaikuttaa loogiselta valinnalta itse hakuprosessin toteutukselle, katsoin työssäni parhaaksi eriyttää sen omaksi luokakseen. Näin itse hakuprosessi saadaan riippumattomaksi Jakarta Strutsista tai palvelinpuolen toteutuksista yleensäkin.

Luokan julkinen rajapinta on yksinkertainen, sillä ainoa metodi on alla esitelty createResults.

```
public LinkedList createResults(String[] keywords,
                                boolean advanced_search,
                                String author)
                  throws SQLException, ClassNotFoundException
{
   String[] searchQueries=new String[keywords.length];
   int i, j, keywordCount;

   // tehdään sql-kyselyt hakusanoista
   keywordCount=keywords.length;
   for(i=keywordCount-1; i>0; i--){
      searchQueries[i]="select id, kunta, kieli, maakunta
                        from kunnat where lower(kunta) like '%";
      for(j=0; j<=i; j++){
         searchQueries[i]=searchQueries[i] + keywords[j] + " ";
      }
      searchQueries[i]=searchQueries[i].trim() + "%' ";

      // lisätään monipuolisen haun kriteerit tarvittaessa
      if(advanced_search==true){
         searchQueries[i]=searchQueries[i] +
                          "and lower(author) like '%" + author + "%'";
      }
      searchQueries[i]=searchQueries[i].trim() + ";";
   }
```

```
      // tehdään viimeinen sql-kysely
      searchQueries[0]="select id, kunta, kieli, maakunta from kunnat
                        where";
      for(i=0; i<keywordCount; i++){
         searchQueries[0]=searchQueries[0] + " lower(kunta) like '%" +
                          keywords[i] + "%'";
         if((i-keywordCount)<(-1)){
            searchQueries[0]=searchQueries[0] + " and";
         }
      }

      // lisätään tarvittaessa monipuolinen haku viimeiseen kyselyyn
      if(advanced_search==true){
         searchQueries[0]=searchQueries[0] + " and lower(author) like '%"
                          + author + "%'";
      }
      searchQueries[0]=searchQueries[0] + ";";

   /* ------------------------------------------------------------------ */

      SqlConn con=null;
      Statement st=null;
      ResultSet rs=null;
      LinkedList results=new LinkedList();
      ResultsEntry temp_entry=null;
      try{
         con=new SqlConn(Constants.URL, Constants.USER,
                         Constants.PASSWD);
         st=con.getConn().createStatement();
         for(i=(keywordCount-1); i>=0; i--){
            rs=st.executeQuery(searchQueries[i]);
            while(rs.next()){
               temp_entry=new ResultsEntry(rs.getInt("ID"),
                                           rs.getString("kunta"),
                                           rs.getString("kieli"),
                                           rs.getString("maakunta"),
                                           new String("author"), new Date
                                           ());
               if(!results.contains(temp_entry)){
                  results.addLast(temp_entry);
               }
               temp_entry=null;
            }
            rs.close();
            rs=null;
         }
         con.closeConn();
      }
      catch(SQLException e){
         [...]
      }
      catch(ClassNotFoundException e){
         [...]
      }
      return results;
   }
```

Metodi saa parametrinaan hakusanataulukon ja palauttaa tulokset

LinkedList-muodossa, joka sisältää ResultsEntry-olioita.

Metodin toiminta on kaksivaiheinen. Ensin luodaan SQL-hakulauseet saaduista hakusanoista ja toisessa vaiheessa ne suoritetaan tietokannassa. Yllä nämä ovat erotettuna toisistaan kommenttiviivalla.

SQL-kyselyitä luodaan saman verran kuin hakusanoja on taulukossa. Niitä ei kuitenkaan haeta yksitellen vaan ryhmissä. Esimerkiksi hakusanoilla ”aa bee cee dee” saadaan seuraavanlaiset kyselyt.

```
select id, description from Material where lower(description) like '%
aa bee cee dee%';

select id, description from Material where lower(description) like '%
aa bee cee%';

select id, description from Material where lower(description) like '%
aa bee%';

select id, description from Material where lower(description) like '%
aa%' and lower(description) like '%bee%' and lower(description) like
'%cee%' and lower(description) like '%dee%';
```

Näistä ensimmäinen antaa tarkimmat tulokset ja vähiten osumia tarkkuuden huonontuessa ja osumien määrän lisääntyessä kahdessa seuraavassa kyselyssä. Viimeinen kysely on erikoistapaus, jossa haetaan kaikkia hakusanoja riippumatta niiden sijainnista tekstissä. Kolme ensimmäistä kyselyä luodaan kahden sisäkkäisen for-silmukan avulla ja viimeinen luodaan omassa silmukassaan.

Kyselyt suoritetaan yllä olevassa järjestyksessä ja hakutulokset lisätään results-listaan. Ennen tuloksen lisäämistä tarkistetaan sen olemassaolo. Näin varmistetaan ettei tuloksia tule useaan kertaan ja tulokset saadaan tarkkuusjärjestykseen.

5.4 Monipuolinen hakujärjestelmä

Monipuolinen haku käyttää samoja luokkia kuin perinteinen hakukin. Tiedon välitys komponenttien välillä ja pintapuolinen tarkistus tehdään normaaliin tapaan SearchForm-JavaBeanilla, hakulauseen käsittely tehdään SearchAction-oliolla ja itse haku puolestaan SearchEngine-luokasta luodulla oliolla. Monipuolinen haku käsitellään erikseen jokaisessa edellä mainitussa vaiheessa.

6 Lähdeluettelo

1 Eckel, Bruce
Thinking in Enterprise Java

2 Goodwill, James 2002
Mastering Jakarta Struts
Wiley Publishing, inc.
0-471-21302

3 Momjian, Bruce
PostgreSQL: Introduction and Concepts

4 Moss, Karl 1998
Java servlets
McGraw-Hill
0-07-913779-2

5 Stinson, Barry 2001
PostgreSQL Essential reference
New Riders
0-7357-1121-6

6 Drake, Joshua D.; Worsley, John C. 2002
Practical PostgreSQL
O'Reilly
1-56592-846-6

7 http://java.sun.com/products/jsp/

8 http://java.sun.com/products/servlet/

9 http://www.codewitz.net/

10 http://www.spidynamics.com/support/whitepapers/WhitepaperSQLInjection.pdf