LAPIN AMK
Lapland University of Applied Sciences

GRAILS-OPAS

Joni Nikkeri

Opinnäytetyö
Tietotekniikan koulutusohjelma
Insinööri (AMK)

2014

LAPIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN ALA

Tietotekniikan koulutusohjelma

Opinnäytetyö

GRAILS-OPAS

Joni Nikkeri

2014

Toimeksiantaja Lapin ammattikorkeakoulu

Ohjaaja Maisa Mielikäinen

Hyväksytty 25.11.2014

Työ on luettavissa Theseus-verkkokirjastossa

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön
tiivistelmä

Tekniikan koulutusala
Tietotekniikan koulutusohjelma

Tekijä	Joni Nikkeri	Vuosi	2014

Toimeksiantaja	Lapin AMK
Työn nimi	Grails-opas
Sivu- ja liitemäärä	34

Opasta kirjoitettaessa on pyritty ottamaan huomioon käyttäjän osaamistaso. Tutkimusaineistona ja lähteenä on käytetty Grailsin verkkosivuja ja Groovyn dokumentointia. Oppaan avulla aloitteleva ohjelmoija voi opiskella Grailsin alkeet ja suunnitella ensimmäisen ohjelmansa kyseisellä frameworkilla.

Opas on laadittu niin että, siinä on käytetty ohjelmoinnin perustietoja karsien mahdolliset Grailsin lisäosat. Näin päädyttiin oppaaseen, joka lähtee perustasolta, käy läpi asennuksen, käyttöönoton ja ensimmäisen ohjelman luonnin. Opas sisältää Grailsissä käytettävät perusmenetelmät ja lisäharjoituksen tietokantojen käytöstä.

Avainsanat Grails, Groovy

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Technology, Communication and Transport
Degree Programme in Information Technology

Author Joni Nikkeri Year 2014

Commissioned by Lapland University of Applied Sciences
Subject of thesis Grails Guide
Number of pages 34

The aim of the thesis was to create a beginner's guide to the Grails framework. With the thesis a beginner programmer can learn the basics of Grails framework and how to create their first software on this basis.

The Guide was written from a personal view of the framework, while taking the possible readers and their experience on frameworks in consideration. The research materials used in the guide were the Grails webpage and Groovy documentation.

Finally the guide was constructed by using only the basic information about Grails and the possible add-ons were cut from the final draft, thus creating a basic guide which starts with the setup and guides the reader trough installation, start-up and creating their first software on Grails. The software exercise includes the basic methods of Grails and an extra exercise based on the databases.

Key words Grails, Groovy

SISÄLTÖ

KUVIOLUETTELO

1 JOHDANTO

Tähän oppaaseen olen sisällyttänyt tarvittavat tiedot aloittelevalle Grails-käyttäjälle ohjelmien asennuksesta Groovy-kielen perusteisiin. Vuonna 2006 julkaistu Grails on lyhenne frameworkin alkuperäisestä nimestä Groovy on Rails, joka muutettiin Ruby on Rails kehittäjän David Heinemeier Hanssonin pyynnöstä (Rocher 2006). Vuonna 2008 Springsource osti G2One-nimisen yhtiön, jolla oli omistusoikeus Grailsiin ja Groovy-kieleen (Delap 2008). Vuonna 2009 VMWare osti Springsourcen, joka omisti G2Onen (Harrington 2009). Näin monen välikäden kautta Grails on päätynyt nykyiseen paikkaansa suosittuna frameworkina, joka hyödyntää dynaamista Groovy-kieltä pohjakielenä.

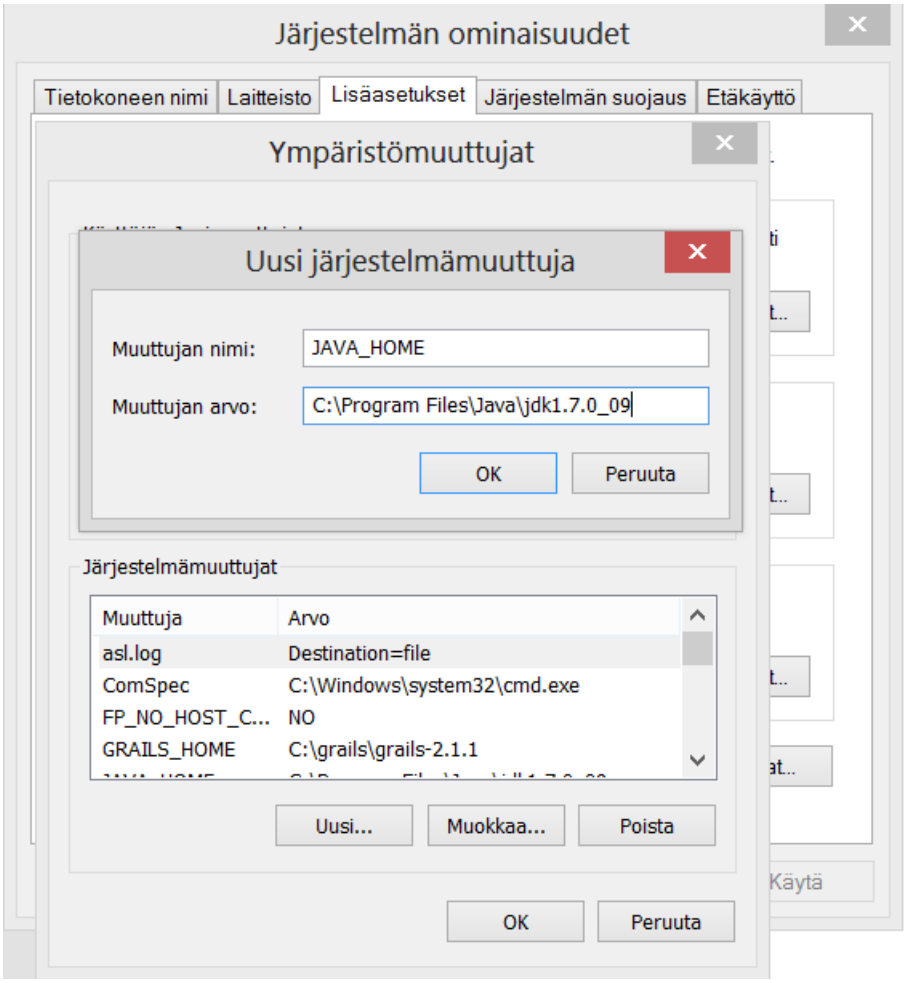
Oppaan tarve opetuskäyttöön tälle frameworkille on mielestäni todella suuri. Uudet opiskelijat voidaan tutustuttaa Javan ohella dynaamiseen kieleen ja tehokkaaseen framework-työskentelyyn. Tavoitteenani oli saada Grails-frameworkin pääasiat ja Groovyn pohjustaminen tähän oppaaseen, josta jokainen voi kehittää omia ohjelmiaan oppaan tietojen pohjalta.

2 ASENNUS JA KÄYTTÖÖNOTTO

2.1 Java sdk-asennus

Java on Grailsin toiminnallinen pohja. Kielenä Grailsissa on Groovy, jonka pohjakieli on Java. Ensimmäisenä tarvitaan Java -kehitystyökalu, eli java SE jdk:n (java development kit), jonka voi ladata eri käyttöjärjestelmille osoitteesta http://www.oracle.com/technetwork/java/javase/downloads/index.html. Lataamisen ja asentamisen jälkeen Java SE jdk täytyy määritellä Järjestelmämuuttujana nimellä JAVA_HOME, joka osoittaa asennusosoitteen.

Ympäristömuuttujien määrittäminen löytyy Windows 7- ja Windows 8 -käyttöjärjestelmissä kohdasta järjestelmä -> järjestelmän lisäasetukset -> Ympäristömuuttujat. Muuttujan nimeksi asetetaan (kuvio 1) JAVA_HOME ja arvoksi asetetaan asennuskansio.

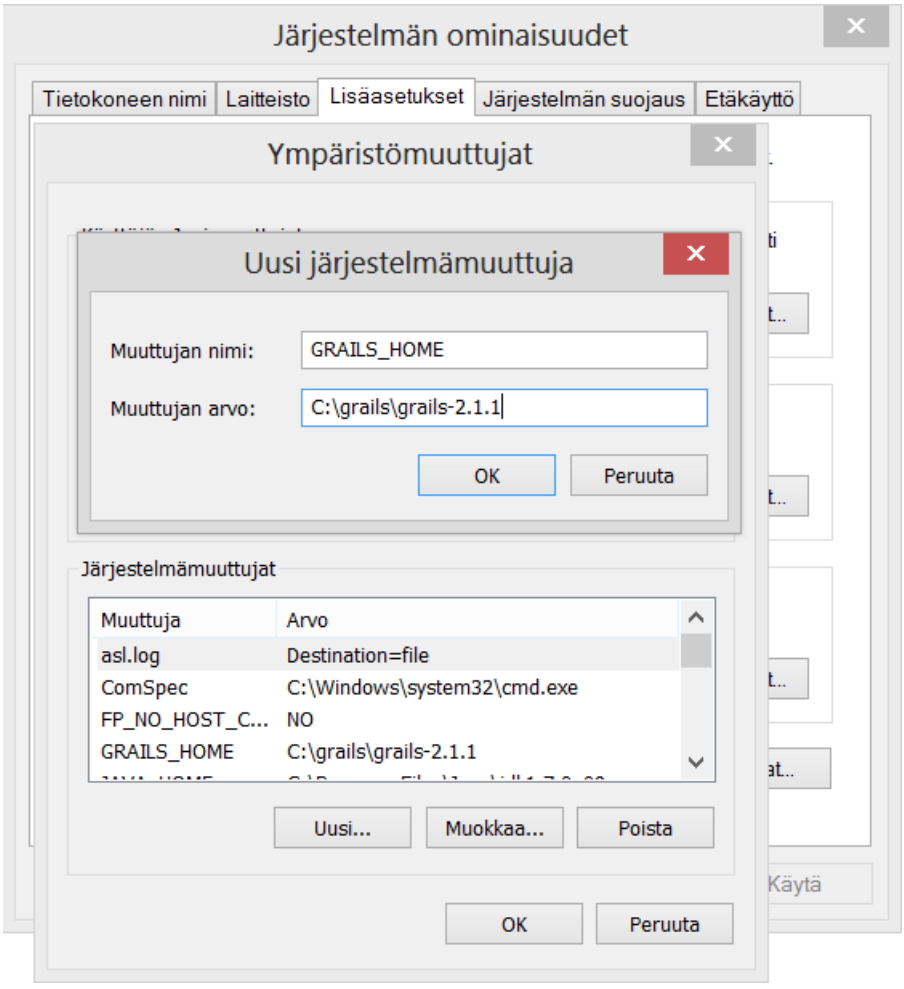


Kuvio 1. Java-järjestelmämuuttujien määrittäminen

2.2 Grailsin ja Groovyn asennus

Grails-verkoston asennuspaketti on ladattavissa osoitteesta http://grails.org/download. (asennuspaketin uusin versio oli kirjoittaessa 2.1.1). Lataamisen jälkeen on hyvä luoda uusi kansio c:\ -aseman juureen nimeltä Grails, jonne asennustiedostot voidaan purkaa. Kansio on sijoitettu c:\ -aseman juureen toiminnallisuuden parantamiseksi.

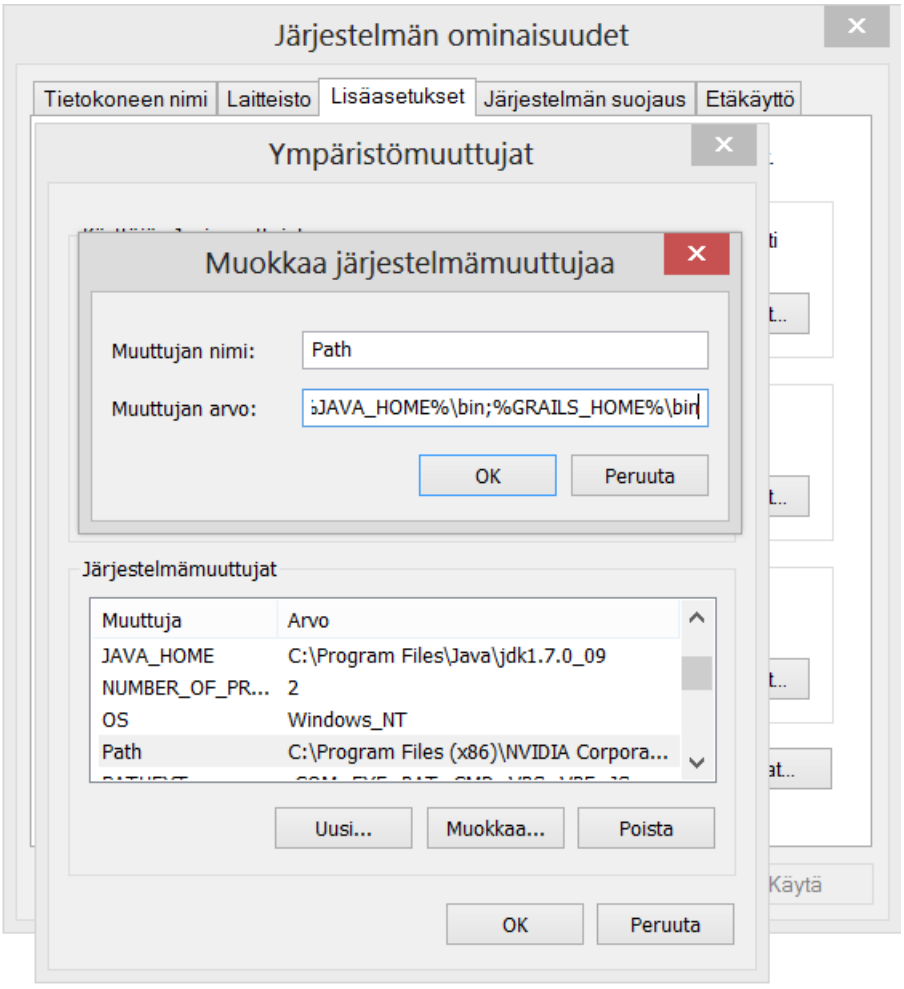
Tämän jälkeen täytyy luoda uusi järjestelmämuuttuja nimeltä GRAILS_HOME ympäristömuuttujiin, kuten edellisessä kappaleessa luotiin Java SE jdk:lle.



Kuvio 2. Grails-järjestelmämuuttujien määrittäminen

Grailsin muuttujan nimeksi asetetaan (kuvio 2) GRAILS_HOME ja arvoksi Grailsin asennuskansio. Näiden uusien järjestelmämuuttujien jälkeen täytyy vielä muokkata yhtä jo olemassaolevaa järjestelmämuuttujaa nimeltä Path, joka löytyy järjestelmämuuttujat -listasta. Muokkaa painikkeella voi muokata Path-muuttujan arvoa. Muuttujan arvoon tulee lisätä loppuun seuraava arvo:

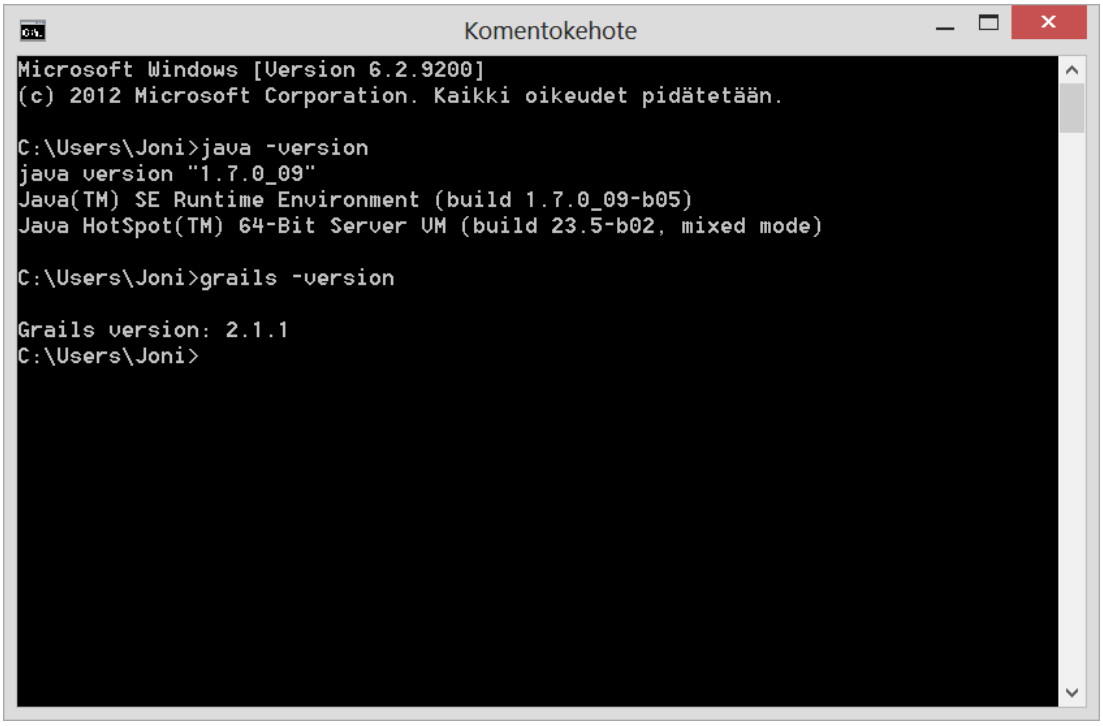
%JAVA_HOME%\bin;%GRAILS_HOME%\bin. Kun kaikki nämä muutokset on hyväksytty, voi navigoida ok-painikkeilla pois järjestelmän ominaisuuksista (kuvio 3) ja alkaa kokeilemaan Java SE jdk:n ja Grailsin toimivuutta.



Kuvio 3. Path-järjestelmämuuttujan määrittäminen

2.3 Java ja Grails asennettuna

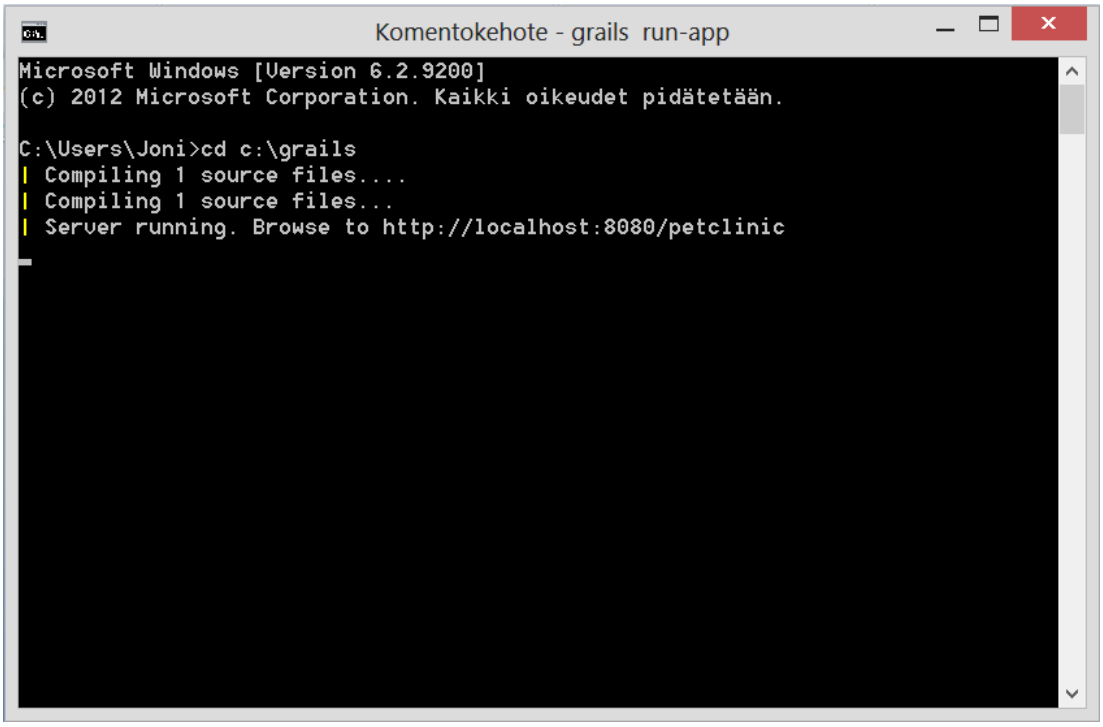
Sen jälkeen kun Java SE jdk ja Grails on asennettu, voi kokeilla niiden toimivuutta komentokehoitteella (cmd.exe). Javan toimivuuden voi varmistaa kirjoittamalla komentokehoitteeseen komennon *java -version*, jolloin JAVA_HOME ilmoittaa Javan asennuskohteen ja version. Grailsin toimivuuden voi myös tarkistaa samalla komentokehoitteella. Syötettäessä komento *grails -version*, tulostuu Grailsin versionumero komentokehoitteesta (kuvio 4). Komento *java -version* antaa Javan asennetun version, joka on 1.7.0_09 ja *grails -version* tulostaa Grailsin version, joka on asennettuna eli 2.1.1



Kuvio 4. Grails-versio komentokehoitteella

Grailsin ensimmäisen koesovelluksen voi ajaa tässä vaiheessa. Esimerkkisovellus löytyy osoitteesta http://grails.org/learn kohdasta 2 ”Try a sample application”. Lataamalla tiedoston Grails-petclinic-master.zip saa täysin valmiin sovelluksen kokeiltavaksi Grails-ympäristössä. Sovelluksen voi purkaa Grailsin asennuskansioon, joka on asennettuna kohteessa c:\grails.

Purkamisen jälkeen tarvitaan taas komentokehoitetta, sillä IDE-työkalua kuten Eclipseä, ei ole vielä asennettu. Komentokehoitteessa Grails-ohjelmien ajaminen suoritetaan käskyllä *grails run-app*. Aluksi joutuu navigoimaan oletuskansiosta c:\Users\Joni\ kansioon, jonne esimerkkisovellus on purettu, eli kansioon c:\grails. Kansioon siirtyminen tapahtuu komennolla cd c:\grails. Kansioon päästyä voi ajaa esimerkkisovelluksen komennolla *grails run-app*.



Kuvio 5. Esimerkkisovelluksen käynnistäminen

Tämä komento käynnistää Grailsin ja kokoaa projektin valmiiksi sekä antaa osoitteen http://localhost:8080/petclinic (kuvio 5), josta sovelluksen toimivuuden voi käydä tarkistamassa selaimen avulla. Seuraavaksi käsitellään IDE-ympäristöä ja sen asennusta, jolla saa luotua uusia Grails-projekteja ja visuaalisen ympäristön sovellusten kehittämiseen.

2.4 Eclipsen asennus

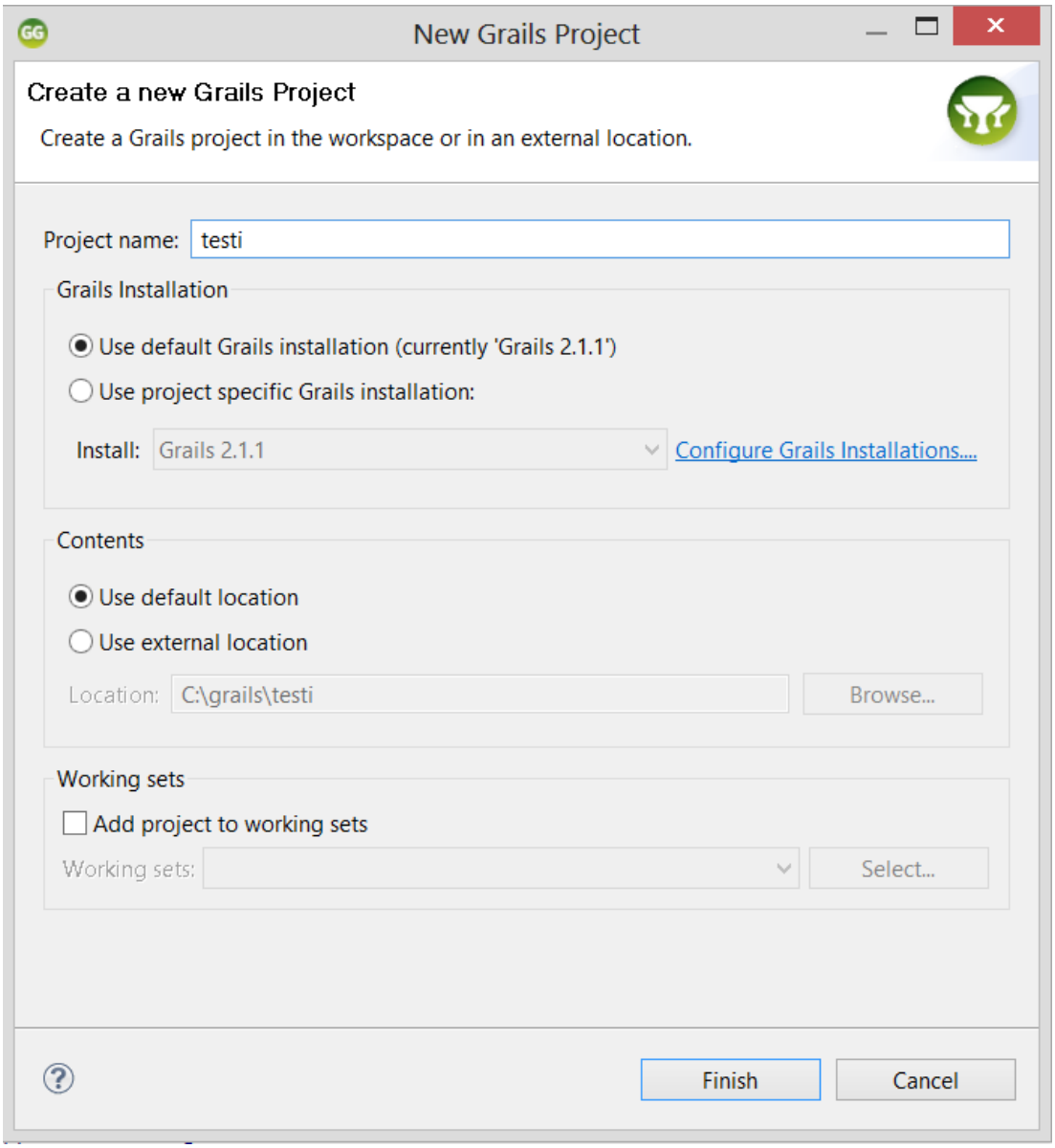
Grailsille löytyy useita IDE-ympäristöjä, kuten Eclipse, NetBeans ja TextMate, joilla voidaan luoda sovelluksia. Tulevissa esimerkeissä käytetään Eclipse-pohjaista Groovy/Grails Tools Suite -ohjelmaa, jonka saa ladattua osoitteesta http://www.springsource.org/downloads/sts-ggts. Uusin versio opinnäytetyötä kirjoittaessa oli 3.2.0.

Ohjelma pyytää lisäämään oletuksena kohdassa 4/8 asennukseen uusimman version Grailsista. Valinnan voi poistaa asennuksesta, sillä kappaleessa 2.2 asennettu Grails on jo tietokoneella, joten uutta asennusta ei tarvita.
Kohdassa 5/8 asennus kysyy Java SE jdk:n asennuskansiota, jonka toimivuus tarkistetaan asennuksen kuluessa. Kansiopoluksi asetetaan C:\Program

Files\Java\jdk1.7.0_09. Asennuksen viimeisissä kahdessa kohdassa voi luoda pikakuvakkeen ja käynnistää ohjelman.

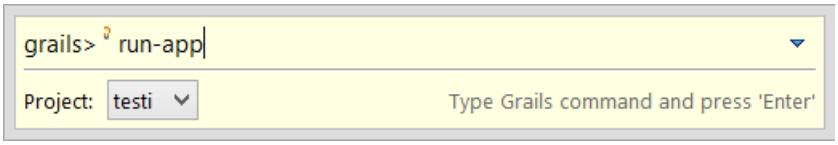
Ohjelman käynnistyessä ensimmäistä kertaa tulee kysymys työtilan sijainnista. Yksinkertaisuuden vuoksi työtilan voi asettaa samaan kansioon, kuin Grailsin asennuksen eli c:\grails. Tässä vaiheessa on myös hyvä asettaa tämä työtila oletustyötilaksi, jolloin seuraavalla kerralla käynnistäessä ohjelma tietää polun muistista.

Seuraavaksi on mahdollista luoda uusi ohjelma, jolla varmistetaan asennuksen toimivuus. Uuden Grails-projektin saa luotua valitsemalla File->New->Grails Project, joka avaa uuden projektin luomisikkunan (kuvio 6).



Kuvio 6. Uuden projektin luominen

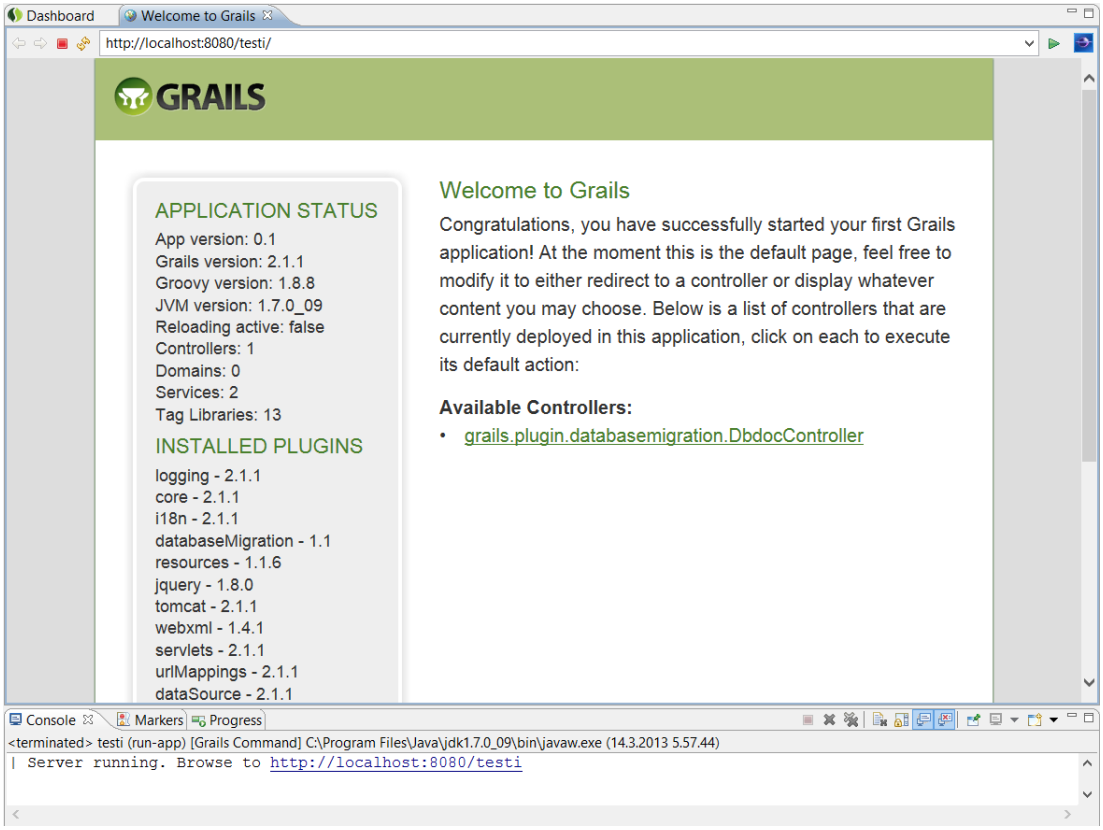
Olen antanut projektille nimeksi "testi" sillä tällä kokeillaan ohjelman toimivuutta. Jos ohjelman on asentanut eri kohteeseen kuin Grailsin, täytyy määritellä Grailsin olinpaikka kohdassa Grails installation, jos ohjelman asennuskansioksi valitsi saman kansion kuin missä Grails sijaitsee, tunnistaa ohjelma asennuksen itsestään, kuten yllä kuviossa 6. Kun kaikki asetukset projektille ovat valmiit, voi painaa Finish-painiketta. Uutta projektia luodessa Eclipse käynnistää Grailsin ja tarkistaa Groovyn ja Grailsin yhteensopivuuden. Kun käynnistys on valmis, voi käynnistää ensimmäisen itse luodun sovelluksen painamalla ohjelmassa ylhäällä olevaa Grails Command History -painiketta ja syöttämällä komento *run-app* (kuvio 7).



Kuvio 7. Komento *run-app*

Komennolla on sama vaikutus kuin aiemmin komentokehoitteessa syötetyllä komennolla.

Kun sovellus on käynnistynyt, tulee ohjelmassa alhaalla sijaitsevaan konsoliin näkyviin ilmoitus: ”Server running. Browse to http://localhost:8080/testi”. Klikkaamalla osoitetta aukeaa Eclipsen omalla selaimella uusi ikkuna, jossa voi tarkastella juuri luotua ohjelmaa (kuvio 8).



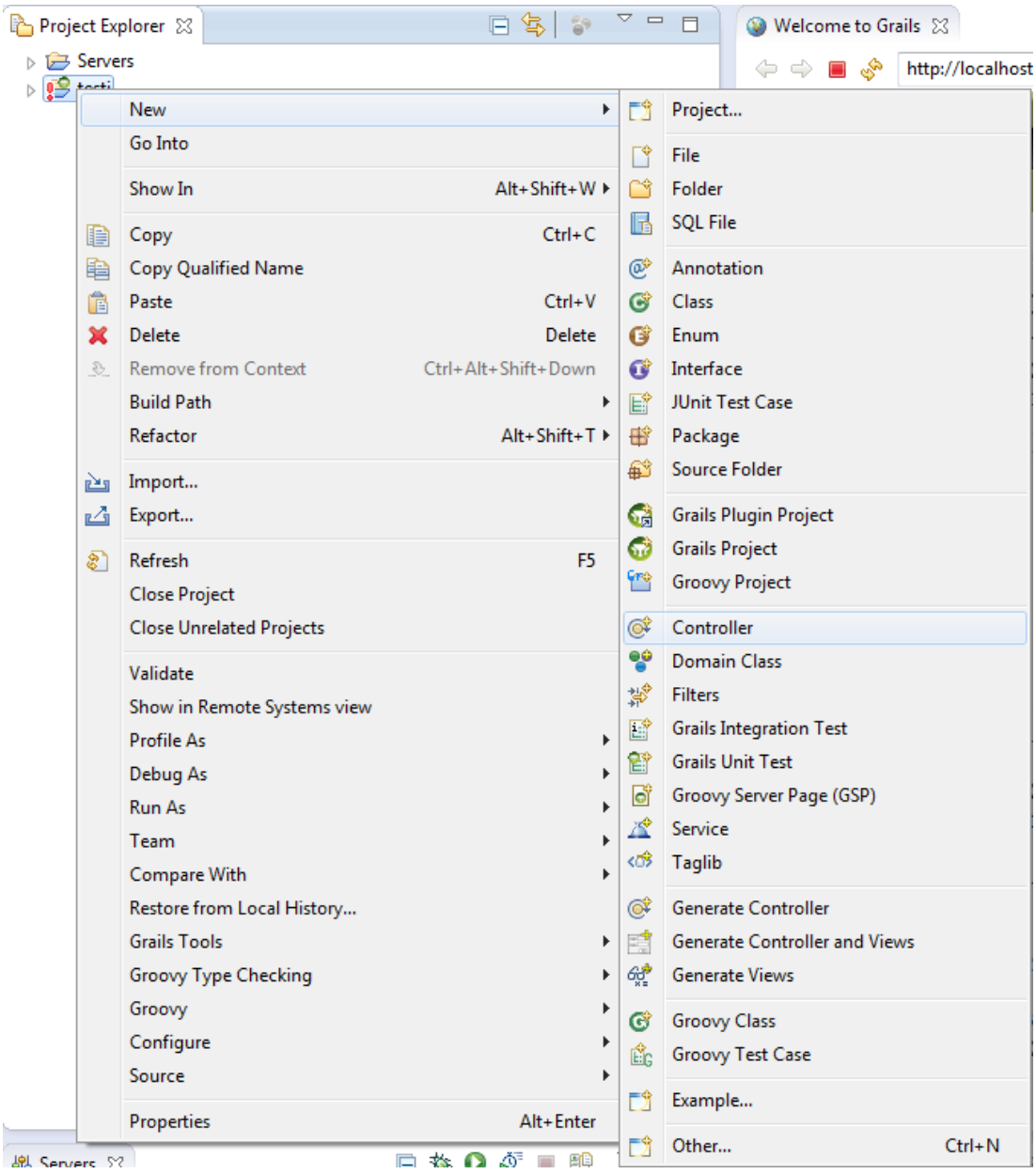
Kuvio 8. Tervetuloa Grailsiin!

Tämä kertoo Grailsin ja Eclipsen toimivan hyvin yhdessä, joten seuraavaksi voi aloittaa luomaan ensimmäistä toimintoja sisältävää ohjelmaa.

3 ENSIMMÄINEN OHJELMA

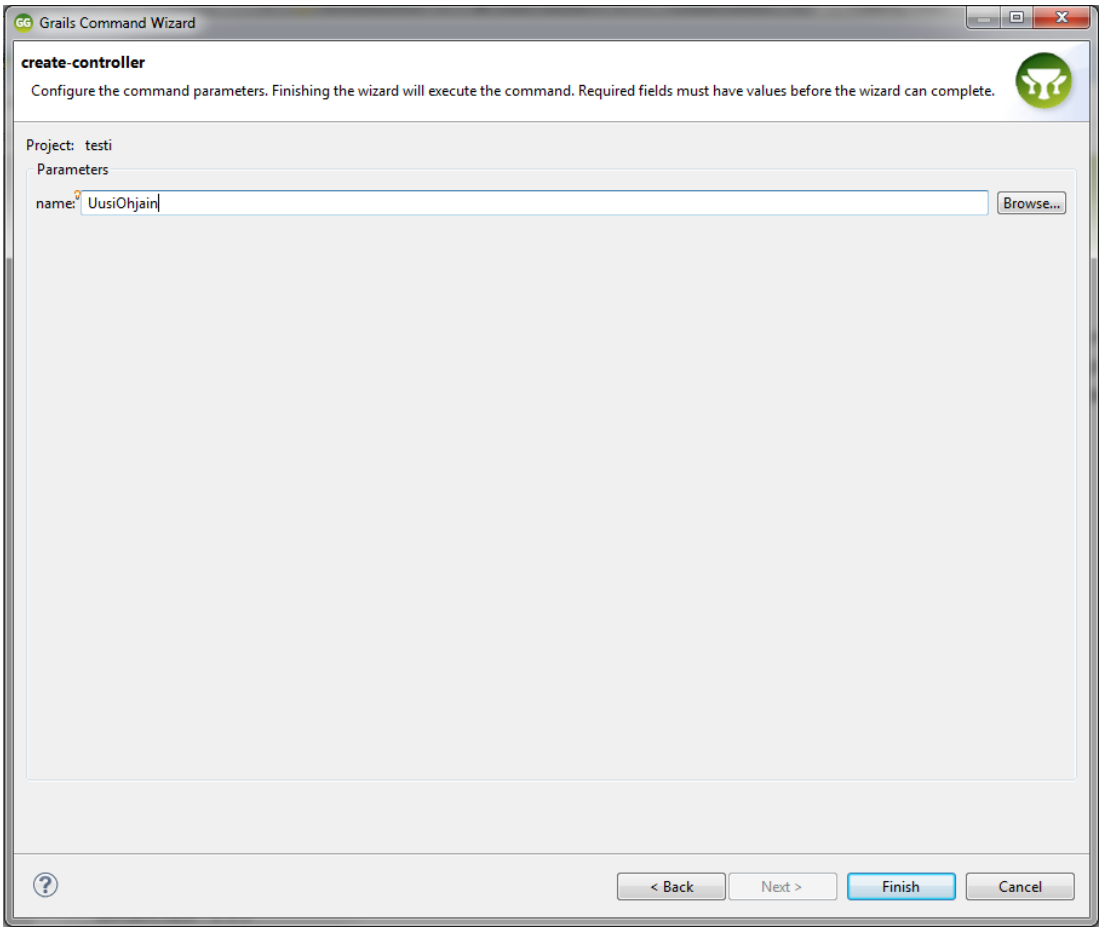
3.1 Ohjaimen luominen

Ohjain eli controller on tarkoitettu Grails-ohjelman url-määrittelyihin. Seuraavassa esimerkissä näkyy, kuinka tämä ohjain toimii käytännössä. Esimerkissä luodaan kaksi erillistä sivua pääsivun alle ja myös toiminto, kuinka pääsivulta voidaan osoittaa tiettyyn sivuun.



Kuvio 9. Ohjaimen luonti

Uusi ohjain luodaan klikkaamalla oikeanpuoleista hiiren painiketta projektiin päällä ja valitsemalla *new* ja *controller* (kuvio 9).



Kuvio 10. UusiOhjain

Tämän jälkeen määritellään ohjaimelle nimi, jonka nimeksi tulee UusiOhjain (kuvio 10).

```
package testi

class UusiOhjainController {

    def index() { }
}
```

Kuvio 11. Oletusmäärittely

Ohjain sisältää tiedon, mihin pakettiin ohjain kuuluu. Grails luo myös uuden luokan ohjaimelle ja oletusmäärittelyn index-metodille (Kuvio 11). Seuraavaksi metodiin lisätään hieman logiikkaa.

```
class UusiOhjainController {

    def index = {
        render "Moi lukija"
    }
}
```

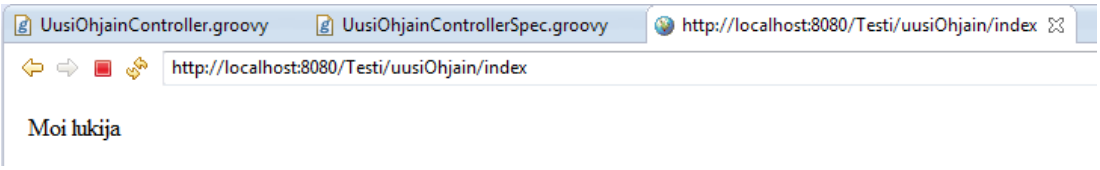
Kuvio 12. *Render*-komento

Render-komento (kuvio 12) on lähes samanlainen kuin *print*. Komento tulostaa sulkujen sisällä olevan tekstin string-muodossa.

```
.................................
|Packaging Grails application
.........................................
|Running Grails application
|Server running. Browse to http://localhost:8080/Testi
......
|Compiling 1 source files
```

Kuvio 13. Frameworkin käynnistys

Käyttämällä *run-app*-komentoa saa palvelimen päälle. Jos palvelin on jo toiminnassa, voi vain tallettaa ohjain-tiedoston (kuvio 13).



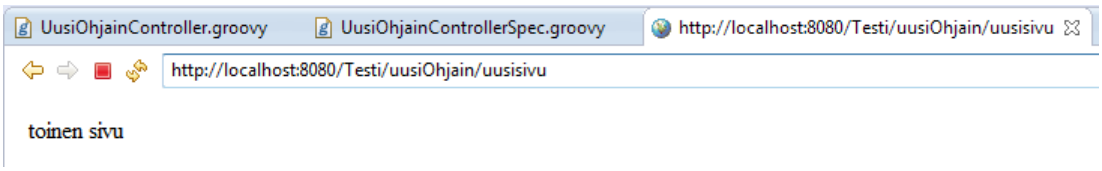
Kuvio 14. Uusi ohjain toiminnassa

Toiminnallisuuden voi kokeilla selaimella osoitteesta http://localhost:8080/Testi/uusiOhjain/ (kuvio 14). Seuraavaksi lisätään ohjaimeen uusi metodi nimeltään ”uusisivu” ja siihen logiikkaa, eli uuden *render* string -tyypin tekstin, esimerkkinä ”toinen sivu” (kuvio 15).

```
class UusiOhjainController {

    def index = {
        render "Moi lukija"
    }

    def uusisivu = {
        render "toinen sivu"
    }
}
```

Kuvio 15. Toinen sivu

Tiedoston tallentamisen jälkeen voi kokeilla luotua metodia.



Kuvio 16. Toinen sivu toiminnassa

http://localhost:8080/Testi/uusiOhjain/uusisivu avaa uuden metodin sisältämän tiedon (kuvio 16). Index-metodi avautuu aina oletuksena ohjainta kutsuttaessa. Seuraavaksi näkyy, kuinka metodin sisällä voidaan avata toinen metodi. Tämä on käytännöllistä, jos katsoo Ohjain-näkymä-malli -suunnittelupohjaa. Jos ohjain sisältäisi suurimman osan teksti tiedosta, tulisi tietojen siirtelystä vaikeaa. Oletuksena tekstisisältö tulisi sijoittaa tietokantaan, eli malli tiedostoon tai näkymätiedostoon. Esimerkkinä näytän kuinka kutsutaan index-metodilla uusisivu-metodia.

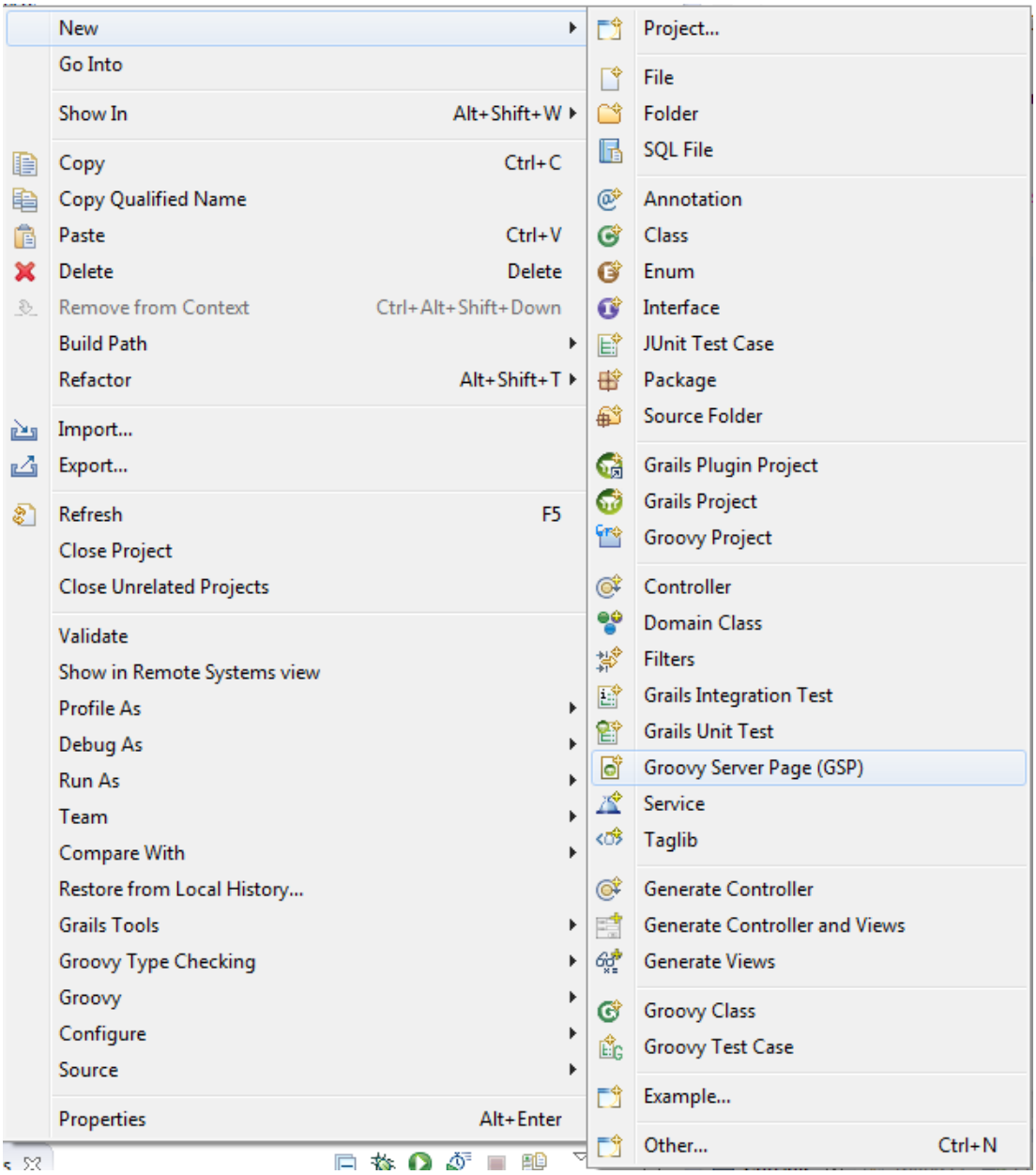
```
class UusiOhjainController {

    def index = {
        redirect (action: uusisivu)
    }

    def uusisivu = {
        render "toinen sivu"
    }
}
```

Kuvio 17. *Redirect*-komento

Sijoittamalla index-metodin logiikkaan kuvion 17 mukaisen komennon *redirect* (action: uusisivu) määrää metodin logiikka nyt ohjaamaan index-metodista uusisivu-metodiin. Nyt on luotu ensimmäinen ohjain ja kokeiltu metodeja. Seuraava kappale kertoo näkymä-tiedostosta ja sen käyttötarkoituksesta.

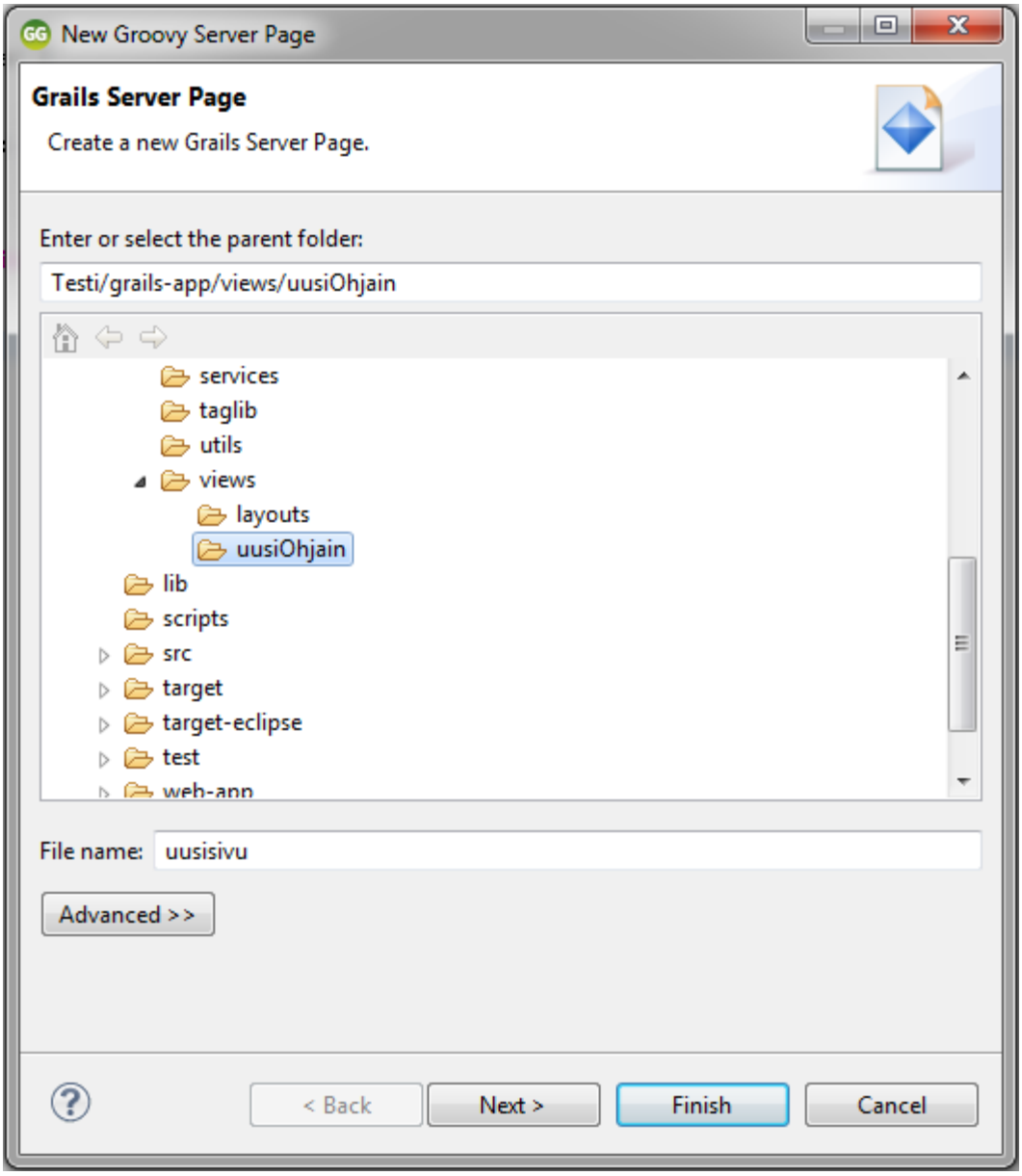
3.2 Näkymän luominen ja käyttötarkoitus

Näkymän luominen projektiin on tärkeää sen sisältämän html-asettelun vuoksi. Jos suurin osa näytettävästä sisällöstä ja asettelusta asetettaisiin ohjaimeen, tulisi koodista heikkomuotoista ja vaikeata muokata. Seuraavaksi luodaan uusi näkymä ja osoitetaan sille kaksi muuttujaa ohjaimessa.



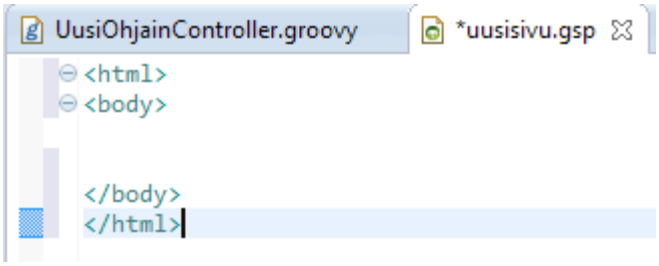
Kuvio 18. Groovy Server Page

Näkymää ei löydy Eclipsestä nimellä view, kuten olettaisi, vaan nimellä Groovy Server Page (GSP). Tämän takia luodaan uusi gsp (kuvio 18).



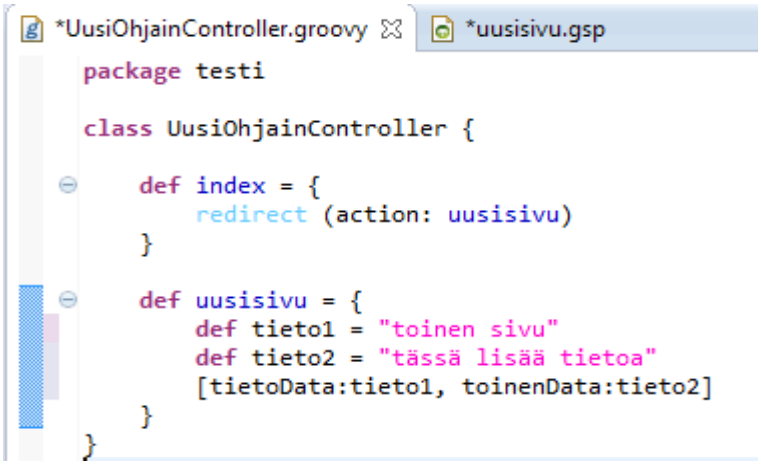
Kuvio 19. Kansion valinta

Eclipse haluaa oletuksena asettaa luodun gsp-tiedoston views-kansioon. Toiminnan yksinkertaistamiseksi se kannattaa sijoittaa view:in alikansioon, jolla on sama nimi kuin käytetyllä ohjaimella, eli uusiOhjain (kuvio 19). Tiedosto tulee nimetä viittauksen vuoksi samalle nimelle kuin ohjaimessa käytetty metodi, eli uusisivu. Näkymiä voi luoda yhden jokaiselle käytetylle metodille, mutta tähän esimerkkiin tarvitaan vain yksi näkymä. Nimeämisen ja kansion valitsemisen jälkeen voi finish-painikkeella siirtyä katsomaan uutta näkymää.



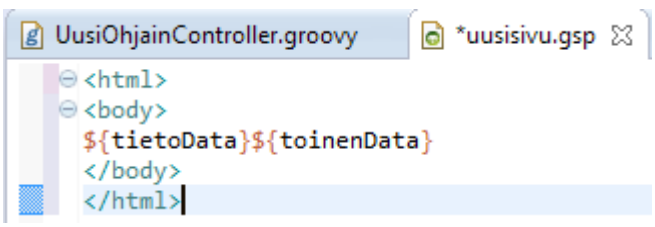
Kuvio 20. Html-koodi

Näkymä avautuu tyhjänä, joten siihen tarvitsee lisätä hieman html-koodia (kuvio 20). Koodin lisäämisen jälkeen täytyy muokata ohjaimessa sijaitsevaa Uusisivu-metodia, eli lisätä siihen kaksi muuttujaa, joita tullaan kutsumaan myöhemmin näkymän html-koodin sisällä.



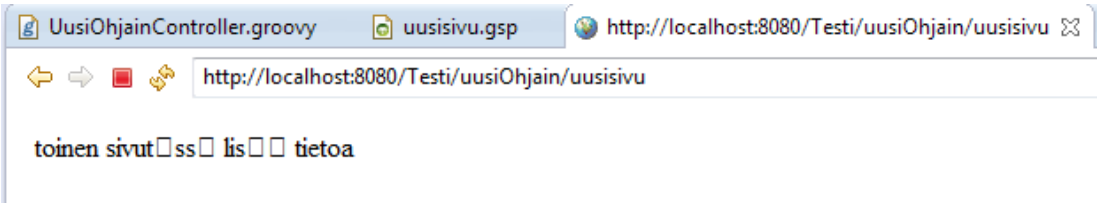
Kuvio 21. Uudet muuttujat

Sen jälkeen kun on määritelty kaksi muuttujaa (tieto1 ja tieto2), asetetaan niille String-tyypin arvot ”toinen sivu” ja ”tässä lisää tietoa” (kuvio 21). Hakasulkeiden käyttötarkoitus on Groovy-kartoitus, joka asettaa muuttujille osoittimet, joita voi käyttää näkymässä. Osoittimet ovat myös hyödyllisiä, sillä niiden osoittamaa muuttujaa voi muuttaa asettamalla kaksoispisteen jälkeen uuden muuttujan nimi. Näin voidaan muuttaa jo valmiin html-asettelun sisältöä tarvitsematta tehdä muutoksia näkymään.



Kuvio 22. Osoittimet näkymässä

Luodut tietoData- ja toinenData -osoittimet voidaan sijoittaa näkymään Groovy-merkillä eli $-merkillä (kuvio 22), jonka jälkeen osoittimet laitetaan aaltosulkujen sisään. Nyt voi tehdyt muutokset tallentaa ja katsoa kuinka näkymä toimii.

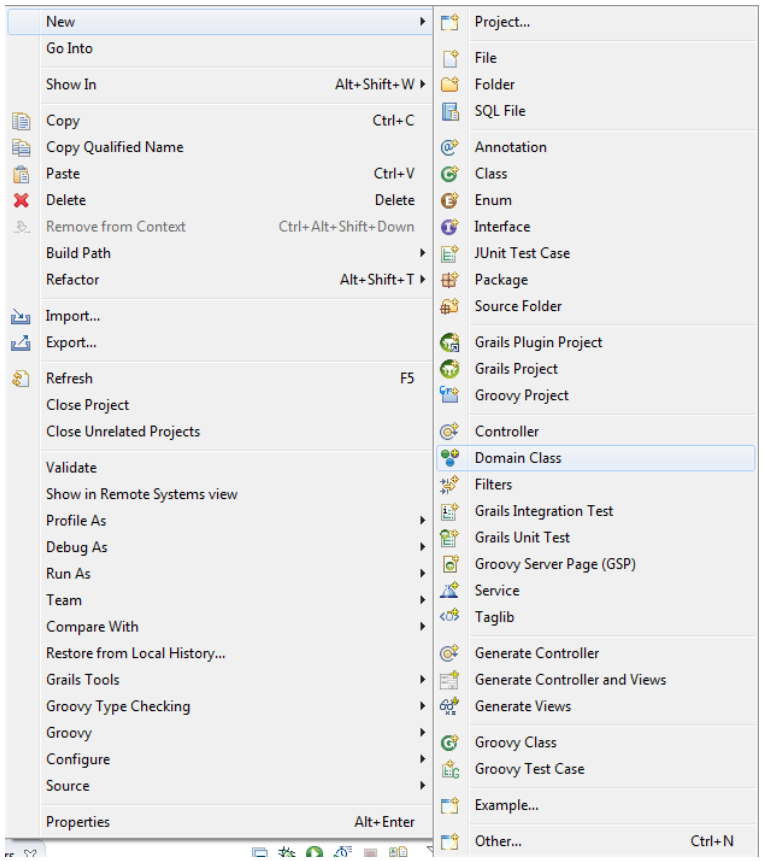


Kuvio 23. Puuttuvat entiteetit

Uusisivu-metodia kutsutaan näkymässä ja muuttujien arvot näkyvät peräkkäin html-asettelun vuoksi. Ä-kirjaimet näkyvät neliöinä, sillä niitä ei ole asetettu määritellyiksi entiteeteiksi (kuvio 23). Seuraava kappale sisältää tietokantoja käsittelevän malli-tiedoston luomisen ja sen liittämisen ohjaimeen ja näkymään sopivaksi.

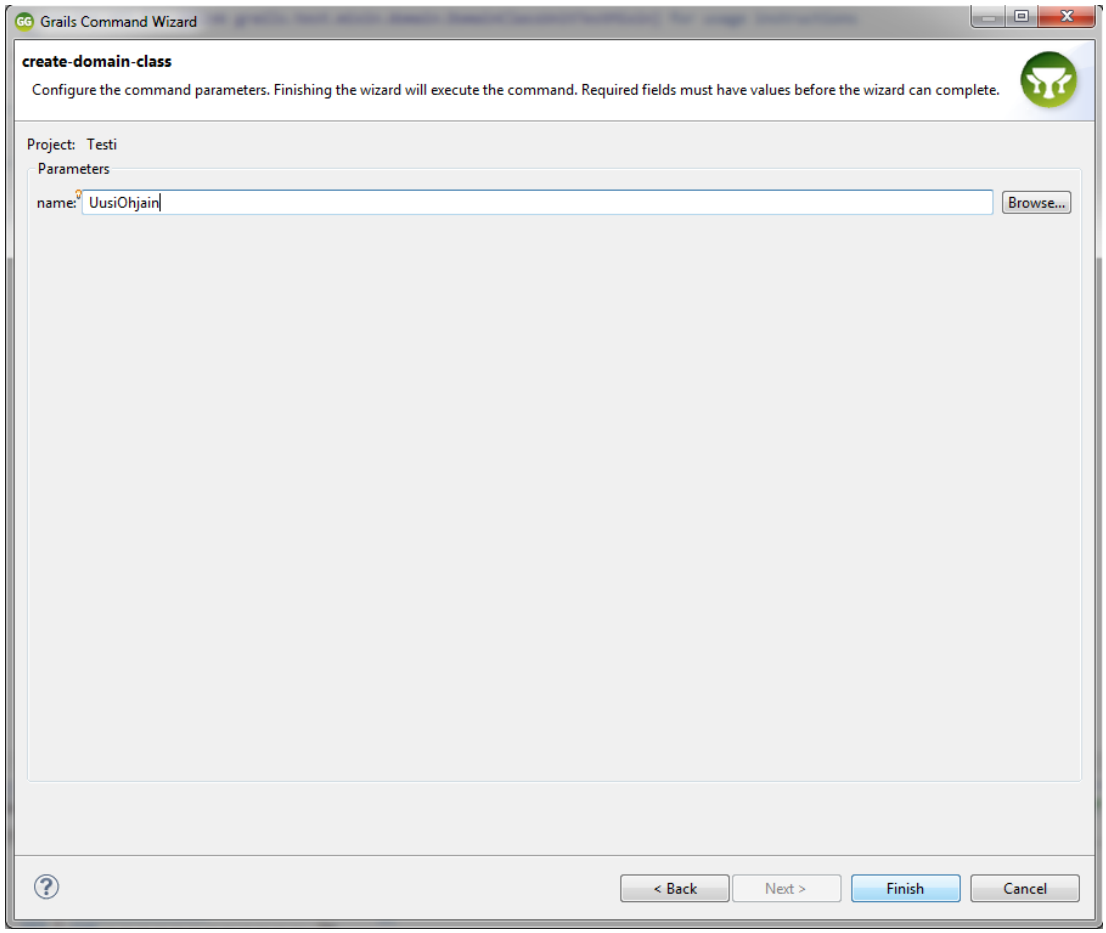
3.3 Malli eli ohjain

Edellisessä esimerkissä syötettiin tieto ohjaimesta näkymään, mutta tässä kappaleessa syvennytään siihen, miten tiedon voi hakea näkymään ja ohjaimeen tietokannasta malli-tiedoston kautta. Ensin luodaan uusi malli.



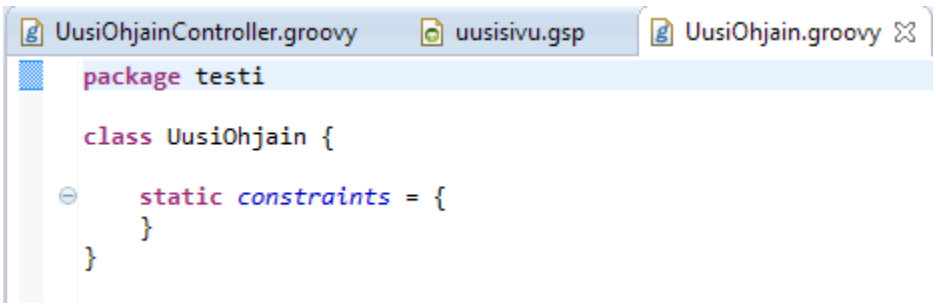
Kuvio 24. Domain Class

Uuden mallin voi luoda klikkaamalla oikealla hiirenpainikkeella projektissa ja valitsemalla New -> Domain Class (kuvio 24).



Kuvio 25. Ohjaimen nimeäminen

Helpoin tapa yhdistää malli ohjaimeen on antaa sille sama nimi kuin ohjaimelle, kuten esimerkissä UusiOhjain (Kuvio 25). Nimeämisen jälkeen voi painaa finish-painiketta.



Kuvio 26. Testi-paketti

Nyt on luotu uusi ohjain joka kuuluu testipakettiin (kuvio 26). Tähän ohjaimeen voi sisällyttää tietomuotoja, jotka tallentuvat h2-tietokantaan. On mahdollista luoda esimerkiksi tietokanta, joka sisältää otsikon, sisällön sekä päivämäärän.

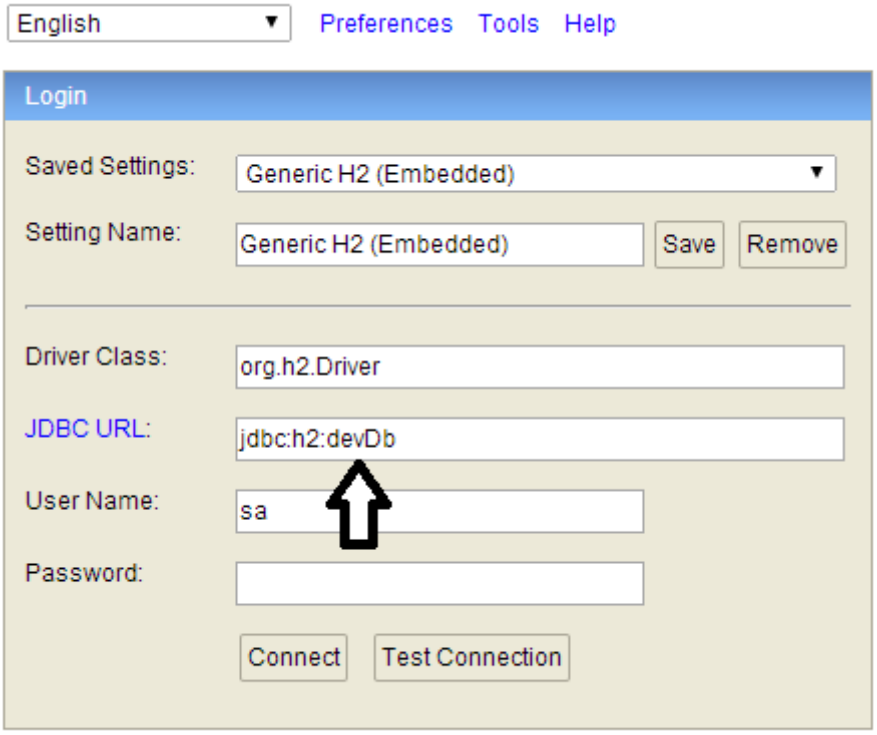
```
package testi

class UusiOhjain {

    String otsikko
    String teksti
    Date paiva

    static constraints = {
    }
}
```

Kuvio 27. Kolme muuttujaa

Kuviossa 27 näkyy kolme muuttujaa (otsikko, teksti ja päivämäärä), joiden luonnin jälkeen voi tiedoston tallentaa ja alkaa lisätä tietoa muuttujien sisään.

3.4 Tietokannan hallinta

Tietokantaan kirjautuminen onnistuu kirjoittamalla selaimen osoiteriville: ”http://localhost:8080/ testi / dbconsole”, eli esimerkissä: http://localhost:8080/Testi/dbconsole/.



Kuvio 28. Tietokannan asiakassovellus

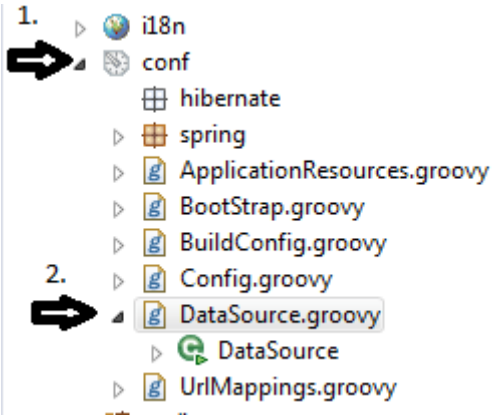
JDBC URL eli Java Database Connection on rajapinta, jolla määritellään tietokannan asiakassovellus (kuvio 28). Oletuksena url:n loppuna on ~testi

käyttäessä development-kehitysympäristöä, joka ilmoitetaan Grailsin käynnistyessä (kuvio 29)

```
|Loading Grails 2.3.5
|Configuring classpath
.
|Environment set to development
```

Kuvio 29. Kehitysympäristönä toimii development

Tietokannan nimi on etsittävä Eclipsen DataSource-tiedostosta (kohdat 1 ja 2 kuviossa 30).



Kuvio 30. DataSource-tiedosto

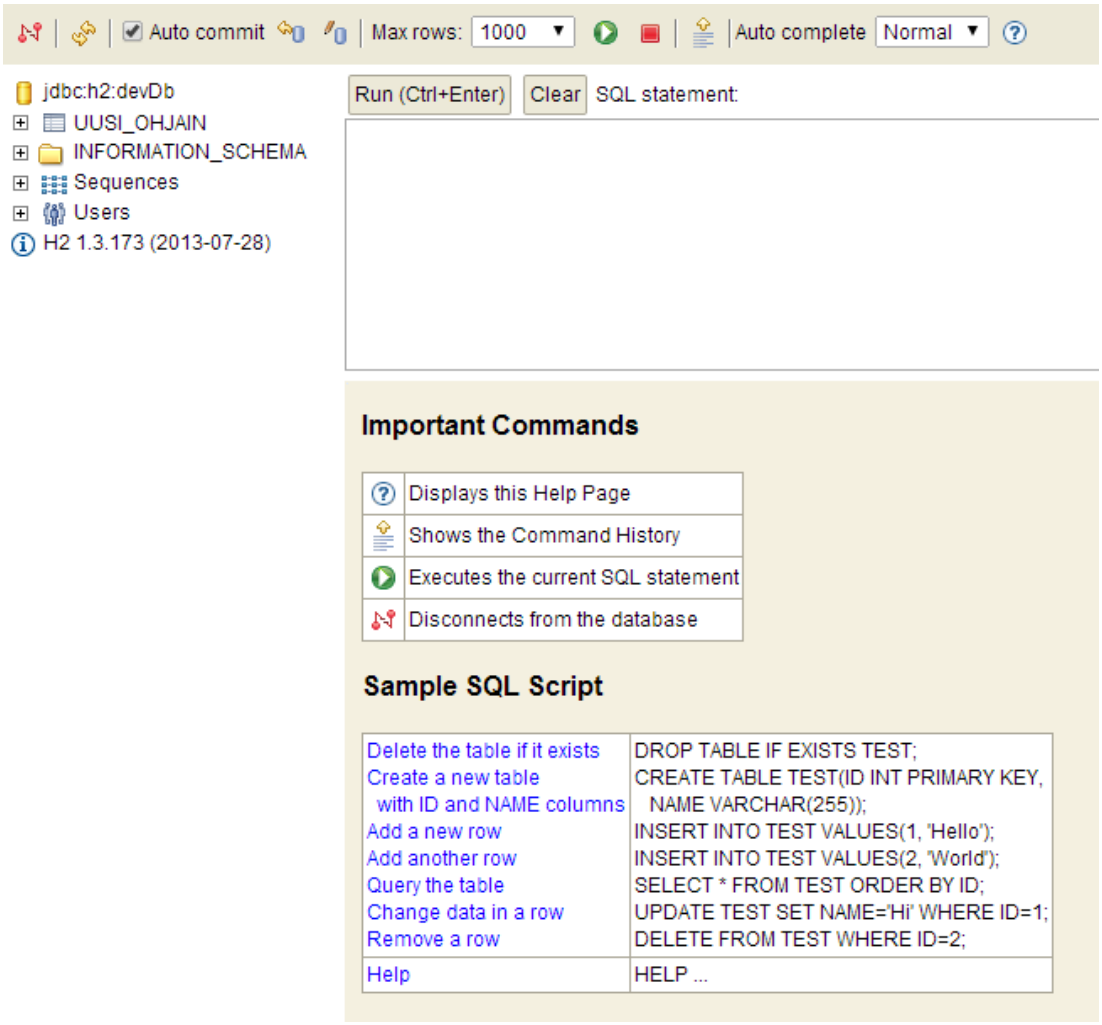
Avaamalla DataSource-tiedoston näkee erilaisten kehitysympäristöjen tietokantayhteydet.

```
environments {
    development {
        dataSource {
            dbCreate = "create-drop" // one of 'create', 'create-drop', 'update', 'validate', ''
            url = "jdbc:h2:mem:devDb;MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE"
        }
    }
    test {
        dataSource {
            dbCreate = "update"
            url = "jdbc:h2:mem:testDb;MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE"
        }
    }
```

Kuvio 31. Tietokannan tiedot

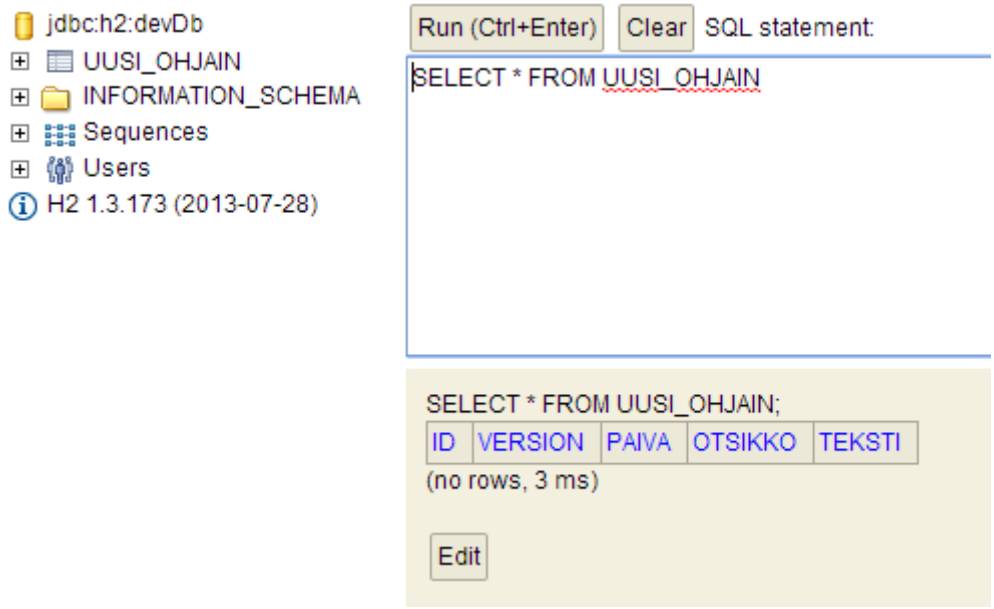
Kuviossa 31 Development-ympäristössä, url-kohdassa näkyy tietokannan tyyppi (h2), jonne tieto tallennetaan (mem) ja tietokannan nimi (devDb). Kun kehitysympäristön tietokannan nimi on tiedossa, voi tietokantaan kirjautua antamalla Java Database Connection-rajapinnalle tiedon käytetyn tietokan-

nan nimestä (kuvio 28). Nimen kirjoittamisen jälkeen tietokantaan voi yhdistää tietokannan asiakassovelluksella painamalla Connect-painiketta.



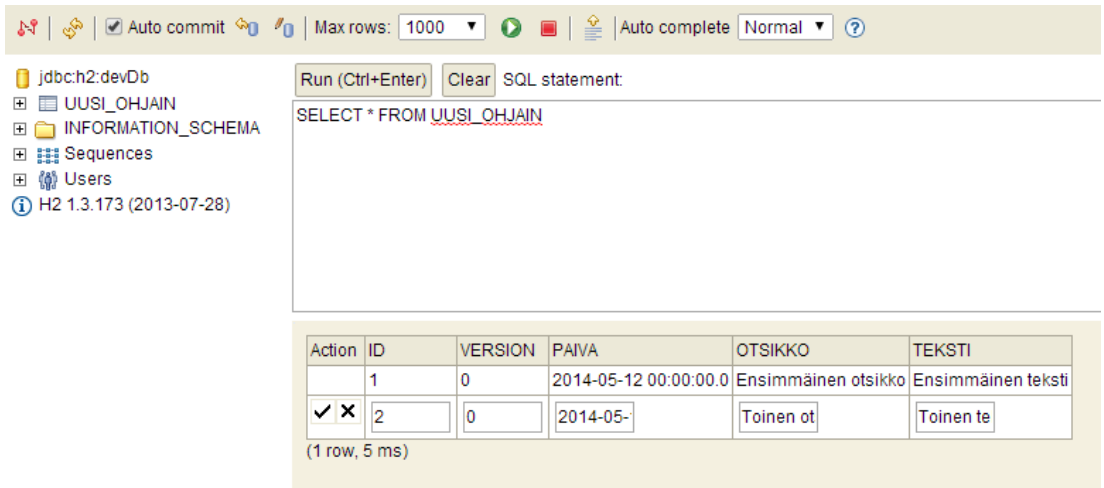
Kuvio 32. Tietokannan sisältämät muuttujat

Tietokannasta löytyy aikaisemmin luotu uusi_ohjain -malli ja sen sisältämät muuttujat (kuvio 32).



Kuvio 33. SQL-Lauseke

Valitsemalla uusi_ohjain –mallin, tulee SQL statement -kohtaan sql-lauseke SELECT * FROM UUSI_OHJAIN ja ajamalla lausekkeen voi huomata tietokannan lisänneen malliin kaksi uutta muuttujaa, ID, sekä VERSION (kuvio 33). Nämä muuttujat lisäävät jokaiselle riville oman tunnisteen (ID) sekä versionumeron.



Kuvio 34. Tietojen syöttäminen tietokantaan

Edit-painikkeella voi lisätä tietoa muuttujille. ID-kohdassa helpoin tapa lisätä tietoa on aloittaa numerosta 1 ja jatkaa eteenpäin tiedon lisääntyessä. VERSION -kohtaan voi asettaa halutun numeron. Esimerkissä käytetään numeroa 0. PAIVA on date-tyypin muuttuja, jossa tieto lisätään muodossa vuosi-

kuukausi-päivä. OTSIKKO -kohtaan voi lisätä halutun tekstin otsikolle ja TEKSTI -kohtaan haluttu teksti sisällölle (kuvio 34).

Seuraavassa kappaleessa tarkastellaan, kuinka juuri luotu tieto voidaan noutaa tietokannasta.

3.5 Tiedon noutaminen tietokannasta

Tiedon noutamiseen tietokannasta ja sen näkyviin saamiseksi näkymässä on muokattava aiemmin luotua ohjainta sekä näkymää.

```
package testi

class UusiOhjainController {

    def index = {
        redirect (action: uusisivu)
    }

    def uusisivu = {
        def kaikkiTieto = UusiOhjain.list()
        [kaikkiTieto:kaikkiTieto]
    }
}
```

Kuvio 35. KaikkiTieto -muuttuja

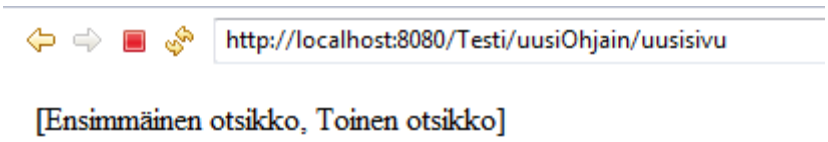
Määritetään uusi muuttuja nimeltään kaikkiTieto, joka hakee kaiken tiedon mallista. Tämä tehdään kirjoittamalla mallin nimi ja lisäämällä perään komento *.list()* (kuvio 35). Alla olevaan Groovy-kartoitukseen on lisätty kaikkiTieto-muuttujalle kaikkiTieto-osoitin, joka noutaa muuttujan tiedot näkymään (kuvio 36).

UusiOhjainController.groovy | uusisivu.gsp

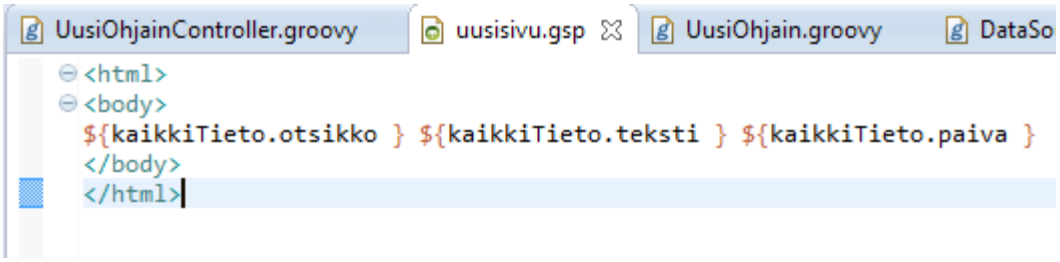
```
<html>
<body>
${kaikkiTieto.otsikko }
</body>
</html>
```

Kuvio 36. KaikkiTieto-osoitin

Asettamalla näkymässä osoittimen perään haettavan tiedon muuttuja, saadaan sivulle tulostumaan muuttujan sisältämä tieto. Muutoksien tallentamisen jälkeen voi mennä osoitteeseen http://localhost:8080/Testi/uusiOhjain/uusisivu ja katsoa saako tietokannasta noudettua halutut rivit. Kuviossa 37 näkyy noudetut rivit.

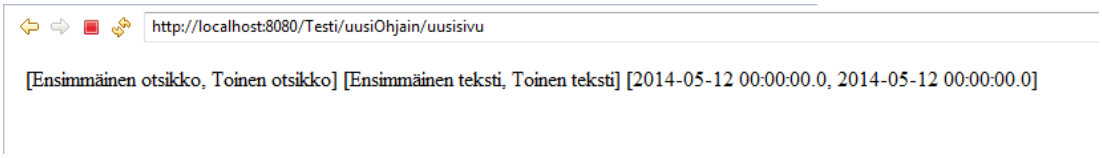


Kuvio 37. Tietokannasta noudetut rivit



Kuvio 38. Kaikki muuttujat

Lisää tietoa voi noutaa lisäämällä loput muuttujat näkymään (kuvio 38).



Kuvio 39. Uudet tietokannasta noudetut rivit

Muuttujat tulevat näkymään sivulla (kuvio 39). Malleja eli ohjaimia voi lisätä sivuston tarpeiden mukaan.

4 GRAILS JA GROOVY

4.1 Groovyn ja Javan erilaisuudet

Groovy on Grailsin käyttämä kieli, josta kerrotaan hieman lisää tässä kappaleessa. Ennen Grails-projektin aloittamista on hyvä opetella muutama erot ja samankaltaisuudet Javan ja Groovyn välillä esimerkkien avulla ja antamalla kielistä tietoa. Groovy on Javasta muokattu laajennettu kieli. Luodessa Grails-sovelluksia on hyväksyttävää käyttää myös Java-kieltä Groovyn sijasta. Groovy on kuitenkin dynaaminen kieli, joten eri komennot ovat helpompia käyttää kuin perinteisessä Javassa. Tämä käy ilmi seuraavan kappaleen Groovy-esimerkistä. Groovy käännetään Java-bittikoodiksi Java-vituaalikoneessa. Tämä mahdollistaa samojen palvelinalustojen käytön kuin mikä tahansa Java-sovellus vaatisi. (Tomcat, Glassfish yms.).

4.2 Groovy Grailsin pohjana

Grails käyttää Groovyä pohjanaan, joten on hyvä opetella muutama esimerkki Groovyn käyttämistä dynaamisista komennoista ja toiminnoista.

Koodiesimerkki 1. String muuttujien käyttö.

```
def etuNimi = 'Joni'
		def sukuNimi = 'Virtanen'
		def a = 5
		def b = 6

		println "${sukuNimi}, ${etuNimi}"
		println "a + b = ${a + b}"
		println '${sukuNimi}, ${fetuNimi}'
		println 'a + b = ${a + b}'
```

Kaikki muuttujat, jotka osoitetaan ilman tyyppiä, ovat itsestään string-tyypin muuttujia (koodiesimerkki 1). Ero Javaan ilmenee muuttujien määrittelyssä. Lainausmerkein osoitetut muuttujat ovat korostettuja muuttujia. Niiden toiminta antaa eri tuloksen kuin heittomerkein osoitetut muuttujat. Tämä komento antaa tulosteen (Koodiesimerkki 2). Optimoidumpi koodi saadaan käytettäessä korostettuja muuttujia, asettamalla String-tyypin muuttujat lainausmerkkeihin sekä asettamalla ne lausekkeen ${} sisään.

Koodiesimerkki 2. Muuttujien tuloste

```
Virtanen, Joni
a + b = 11
${sukuNimi}, ${etuNimi}
a + b = ${a + b}
```

Datamanipulaatio Groovyssa, kuten listojen luominen ja *map*-toiminto, ovat helpompia kuin perinteisessä Javassa. Seuraavissa esimerkeissä näkyy Groovyn eri toimintoja *list*-komennolle. Listaa luodessa Javassa käytetään komentoa *list* (koodiesimerkki 3).

Koodiesimerkki 3. List-komento.

```
List javaList = new ArrayList()
```

List-toiminto Javassa vaatii listan määrittelyn ja muuttuvan implementoidun listatyypin kuten *ArrayList()*, kun taas Groovyssa listan luominen on helpompaa (koodiesimerkki 4).

Koodiesimerkki 4. Listan luominen Groovyssa

```
def groovyList = []
```

Pelkkä nimeäminen ja hakasulut riittävät ilmoittamaan tyypiksi dynaamisen listan (koodiesimerkki 5).

Koodiesimerkki 5. Dynaaminen lista Groovyssa

```
def groovyList = [6, ”Joni”, new Integer (3), new Double(2.6) ]
```

Listan sisälle voi asettaa eri tietotyyppejä (Int, String, Double jne.) Listaan tiedon lisääminen tapahtuu antamalla listan nimi sekä merkit << (koodiesimerkki 6).

Koodiesimerkki 6. Listaan lisääminen

```
println groovyList<<3
```

Yksittäiseen tietoon osoittamisen voi tehdä indeksinumeroa osoittamalla (koodiesimerkki 7).

Koodiesimerkki 7. Yksittäiseen tietoon osoittaminen

```
println groovyList[1]
```

Listan elementtien numeromäärän saa näkyviin *size()*-metodilla (koodiesimerkki 8).

Koodiesimerkki 8. Listan elementtien lukumäärän näyttäminen

```
println groovyList.size()
```

Eritellyn tiedon listan muuttujista, jotka täyttävät halutut ehdot saadaan komennolla *findAll* (koodiesimerkki 9).

Koodiesimerkki 9. Komento *findAll*.

```
println grovyList.findAll{it  }
```

Aaltosulkeiden sisälle asetetun ehdon täyttämien muuttujien arvot tulostetaan. Uuden listan luominen edellisen listan muuttujilla onnistuu komennolla *collect{}*. Jos uuteen listaan halutaan muokata muuttujia, tulee aaltosulkeiden sisään asettaa haluttu ehto. Listan sisällön muuttujien merkkijonon pituuden saa tulostettua komennolla listanNimi.*length()*. *Map*-toiminto on tarkoitettu parien arvojen tallettamiseen, esim. Koira on eläin. Ero listan ja ”kartan” välillä on kaksoispiste (koodiesimerkki 10).

Koodiesimerkki 10. Parien arvojen tallettaminen *map*-toiminnolla

```
def uusityhjäMap = [:]
def uusiMap = [eläin:'Koira',  nisäkäs:'Kissa']
```

Map-toiminto käyttää rajapintaa java.util.*Map*. Kaikki muuttujat ovat oletukselta String-tyyppisiä, myös ilman erikseen heittomerkein määritellyt muuttujat ovat String-tyyppiä, kuten ylemmässä esimerkissä eläin. Tietojen lisäämisessä listaan on kolme eri metodia. Näillä menetelmillä voi luoda uusia muuttujapareja (koodiesimerkki 11).

Koodiesimerkki 11. Kolme metodia listaan lisäämisessä

```
uusiMap['auto'] = 'Porrasperä'
uusiMap.put('esineet', 'Lamppu')
uusiMap << [eläin:'Hevonen']
```

Silmukoiden käyttö Groovy-kielessä on tehty myös erittäin helpoksi. Seuraavaksi esitellään esimerkkejä silmukoista. Komennon ajaminen useaan kertaan onnistuu antamalla kertojen määrä (8), jonka jälkeen asetetaan komento *.times* (koodiesimerkki 12).

Koodiesimerkki 12. Tulostussilmukka

```
8.times {
        println "Moi"
}
```

Listan yksittäiset muuttujat voidaan tulostaa komennolla *.each* (koodiesimerkki 13).

Koodiesimerkki 13. Listan yksittäisten muuttujien tulostus

```
['kissa', 'koira', 'hevonen'].each { nimi ->
        println nimi
```

Tulostaessa numeroita järjestyksessä voi käyttää samaa silmukkakomentoa kuin edellisessä esimerkissä (koodiesimerkki 14).

Koodiesimerkki 14. Numeroiden tulostus silmukkakomennolla

```
(1..10).each { number ->
        println number
}
```

Sulkujen sisällä olevat numerot sekä niiden väliin jäävät numerot tulostuvat asettamalla kaksi pistettä lukujen väliin. *Map*-tyypin listan muuttujien tulostuksessa käytetään samaa silmukkakomentoa kuin edellisissä esimerkeissä. Tulostaessa niihin tulee viitata korostettuina arvoina asettamalla ne lainausmerkkeihin, sekä lisäämällä viittaajat ${} symbolin sisään (koodiesimerkki 15).

Koodiesimerkki 15. *Map*-tyypin listan muuttujien korostus silmukassa

```
[hedelmä:'Omena', kasvis:'Kurkku', auto:'Opel'].each { key, value ->
        println "${key} = ${value}"
}
```

5 YHTEENVETO

Opinnäytetyössä on käsitelty perusasiat Groovysta Grailsin pohjana sekä perusteet Grails-ohjelmien luomiseen. Aikani opinnäytetyön tekemiseen on rajallinen, ja suuri osa Grails-dokumentaatiosta on yhä läpikäymättä tässä opinnäytetyössä. Opinnäytetyötä voisi vielä laajentaa, mutta tällä hetkellä vastuu laajennuksesta jää mahdollisille jatkokehittäjille. Grails ohjelmien luontiin liittyvien asioiden selostaminen vaatii tarkkuutta ja on aikaa vievää.

Opinnäytetyön sisältö ei aukea helposti lukijalle, jos hänellä ei ole tarvittavaa tietopohjaa. Opinnäytetyötä kirjoittaessani minun piti usein pohtia lukijan osaamisen tasoa tästä aiheesta. Toivottavasti tämän opinnäytetyön lukija innostuu luomaan omia ohjelmia Grails-ympäristössä, oli hän millä tasolla tahansa ohjelmien luomisessa.

LÄHTEET

Delap, S 2008. SpringSource Embraces Groovy and Grails with Acquisition of G2One. Viitattu 14.5.2014. http://www.infoq.com/news/2008/11/springsource-g2one

Harrington, C 2009. VMWare acquiring SpringSource which Acquired G2One. Viitattu 14.5.2014. http://colinharrington.net/blog/2009/08/vmware-acquiring-springsource-which-acquired-g2one/

Rocher, G 2006. Groovy on Rails is no more (kind of). Viitattu 14.5.2014 http://grails.1312388.n4.nabble.com/Groovy-on-Rails-is-no-more-kind-of-td1313422.html.