Tallinna Ülikool
Informaatika Instituut

SÕLTUVUSTE VÄHENDAMINE JA ASPEKT-ORIENTEERITUD PROGRAMMEERIMINE SPRING RAAMISTIKU ABIGA

Seminaritöö

Autor: Hannes Mäehans
Juhendaja: Jaagup Kippar

Tallinn 2011

Sisukord

Sissejuhatus

Spring raamistikku on võimalik arendada erinevates integreeritud programmeerimis-keskkondades (IDE- *Integrated Development Environment*), nagu Eclipse, Netbeans IDE ja SpringSource Tool Suite (STS). Käesoleva juhendi eesmärgiks on seletada, kuidas seadistada Eclipse keskkonda, kasutada Spring raamistiku sõltuvuste sisestust (DI – *Dependency Injection*) ning aspekt-orienteeritud programmeerimist (AOP – *Aspect Oriented Programming*). Juhendis teeme läbi erinevaid näiteid. Juhend on mõeldud iseseisvaks õppeks ning õpitu kinnistamiseks on lisatud ülesanded.

Juhendi esimeses osas tutvustame Springi raamistikku ja räägime lähemalt kasutatavatest moodulitest. Jätkame Eclipse keskkonna paigaldamise ja seadistamisega, millele järgneb „Game“ projekti loomine ja sõltuvuste sisestamine. Viimases osas vaatame lähemalt aspekt-orienteeritud lahenduste loomist.

Teema valikut mõjutas eelnev kokkupuude raamistikuga. Tekkis vajadus lahendada Spring ülesannet ning tol hetkel mõistsin, et eestikeelne materjal puudub. Aine „Rakenduste Programmeerimine“ raames tuli teemaga edasi tegeleda ning osutus mõistlikuks kirjutada õpijuhend kaasavastajatele.

1 Olemasolevate materjalide ülevaade

Materjale Spring raamistiku kohta leidub veebis küllaga. Iseseisvaks õppimiseks aga eesti keeles häid materjale ei leidu. Juhendis mainitud materjalid on ühed huvitavamad mida leidsin ning soovitan ka nendega tutvumist.

1.1 PowerPoint slaidiesitlus enos.itcollege.ee serveris

Eestikeelne slaidiesitlus, milles kasutatakse Spring Framework 2.5 versiooni. Kirjeldatakse sõltuvuste sisestamist ja servlette. Aspekt-orienteeritud näiteid kahjuks ei eksisteeri. Arenduskeskkonnast juttu ei tehta. Hea seletus skoobi kohta. Sobib pigem loengumaterjaliks kui iseseisvaks õppematerjaliks.

http://enos.itcollege.ee/~signe/Java%20baaskoolitus%202009/Koolitusmaterjalid/Spring%20raamistik.ppt

1.2 Spring 3 Hello World, Spring Hello World Example

Tutvustatakse kuidas Eclipse keskkonnas luua Spring Framework 3.0 sõltuvuste sisestamise projekti. Eclipsele Springi lisamooduleid ei paigaldata. Projekt luuakse tava Java projektina. Detailne näide.

http://www.roseindia.net/spring/spring3/spring-3-hello-world.shtml

1.3 Spring Framework Tutorial | JAVA9S.com

Videotutvustuse seeria, mis koosneb kaheksast videost. Kasutatakse Spring Framework 2.0 versiooni. Arendatakse Eclipse keskkonnas. Tutvustatakse sõltuvuste sisestamist ja aspekt-orienteeritud programmeerimist. Väga hea ja üheselt mõistetav õpetus.

http://www.java9s.com/spring-framework-tutorial

1.4 Spring Tutorial

Kogu juhenditest, kus tutvustatakse Eclipse seadistamist, sõltuvuste sisestamist ja teisi Springi mooduleid. Hea ja selge juhend.

http://www.vaannila.com/spring/spring-tutorial/spring-tutorial.html

1.5 Spring In-depth, In Context

Tegemist on veebiraamatuga, kus keskendutakse Spring Framework 2.5 versioonile. Sisaldab koodinäited koos pikemate seletustega. Sobilik detailse info saamiseks.
http://www.springindepth.com/

1.6 Spring Reference Documentation

Viimasena, kuid mitte tähtsuse järjekorras on SpringSource kodulehel asuv dokumentatsioon. Viimane sisaldab koodinäiteid ning muudatuste infot. Sobilik detailse info saamiseks.
http://static.springsource.org/spring/docs/3.1.0.RC1/spring-framework-reference/html/

2 Spring Framework

Spring on kasvav raamistik. Kui Springi raamistikuga välja tuldi, nõustusid vähesed, et Java EE esindab hetke parimat lahendust. Springi tutvustati uhkusega, kuna see otsis lahendust Java EE lihtsustamiseks. Iga järgnev Springi väljalase pakub uusi lahendusi ja lihtsustatud funktsioone. (Mak, Long, & Rubio, 2010)

Raamistik on kerge ja potentsiaalselt terviklik (*one-stop-shop*) äri-valmidusega rakenduse ehitamiseks. Spring on modulaarne, võimaldamaks kasutada ainult vahendeid, mida on tarvis. Spring integreerub hästi teiste raamistikega. Näiteks on võimalik kasutada Springi sõltuvuste vähendamist (DI – Dependency Injection) koos Struts raamistikuga, „Hibernate"i integratsiooniga või JDBC (Java Database Connenctivity) abstraktsioonikihiga. Springi raamistik toetab deklaratiivsete tehingute haldamist, kaughaldust läbi RMI (*Remote Method Invocation*) või veebiteenuse ja mitmeid vahendeid andmete salvestamiseks. Spring pakub veebile MVC (*Model-View Controller*) lahendust ja aitab tarkvarasse integreerida aspekt-orienteeritud programmeerimist (AOP – *Aspect Oriented Programming*). (Part I. Overview of Spring Framework)

Spring ei ole sissetungiv raamistik, selle all mõeldakse, et domeeni loogika kood ei sõltu Springi raamistikust. Integratsioonikihis (näiteks andmetega manipuleerimise kihis) võivad mõned sõltuvused siiski esineda, kuid neid klasse on lihtne isoleerida ülejäänud koodist. (Part I. Overview of Spring Framework)

Alates versioonist 2.0 võeti Springi raamistiku sihtmärgiks ka teised platvormid. Raamistik pakub endiselt lahendust olemasolevatele platvormidele, aga seal kus võimalik on platvormist sõltuvus eemaldatud. Java EE on põhiline suund, kuid ei ole enam ainus sihtmärk. Annotatsooni-keskse raamistiku ja XML skeemide tutvustusega on SpringSource loonud raamistiku, mis efektiivselt modelleerib probleemispetsiifilise domeeni. Springi peale ehitatavad raamistikud on loonud toe rakenduste integreerimisele (*application integration*), pakktöötlusele (*batch processing*), Flexi ja Flashi integreerimisele ja teistele. (Mak, Long, & Rubio, 2010)

Käesolevas juhendis vaatame lähemalt sõltuvuste vähendamist ja aspekt-orienteeritud programmeerimist Spring 3.1 raamistiku abiga.

2.1 Sõltuvuste vähendamine

Java rakendus on lai mõiste, mis hõlmab alates lihtsast Appletist kuni keeruliste n-astmeliste serverirakendusteni. Tüüpiliselt koosneb lahendus osadest, mis koos töötades loovad terviku. Sellises tervikus tekib objektide vahel sõltuvus.

Java platvorm pakub rakenduse arendamiseks rohkeid võimalusi. Kahjuks aga puuduvad vahendid põhiplokkide organiseerimiseks. See töö jääb arhitektide ja arendajate kanda. On olemas küll disainimustrid nagu tehas (*factory*), abstraktne tehas (*abstract factory*) jne kuid parimal juhul need annavad vaid nime, kirjelduse mida nad teevad, kus kasutada ja mis probleemi lahendada. Disainimustreid kasutades parima lahenduse saamiseks tuleb mustrid ise realiseerida.

Spring raamistiku sõltuvuste sisestamise (DI – *Dependency Injection*) komponent lahendab selle probleemi sobitades kokku erinevad komponendid ühtseks töötavaks rakenduseks. Raamistik süstematiseerib esimese klassi objektid (*first-class objects*) disainimustriteks, mida saab integreerida rakendusse. Rohke arv organisatsioone ja instituute kasutavad just sellepärast Springi raamistiku keeruliste ja hallatavate lahenduste loomiseks. (Part I. Overview of Spring Framework)

Nagu JavaBeansis nimetatakse ka Springis komponente „ubadeks“ (*beans*). Springi ubade lähenemine erineb Suni poolt loodud JavaBeansi omast. Springi konteineris deklareeritud oad ei pea olema JavaBeansid. Nad võivad olla tavalised Java objektid (POJO – *Plain Old Java Object*), mis ei pea realiseerima kindlat liidest või laiendama baasklassi. See termin eristab lihtsad Java komponendid keerulistest komponentidest. (Part I. Overview of Spring Framework)

2.2 Aspekt-orienteeritud programmeerimine

Aspekt-orienteeritud programmeerimine on üks huvitavamaid meetodeid abstraheerimiseks pärast objekt-orienteeritud programmeerimist. Tegu on küllaltki uue programmeerimismudeliga, mis aitab modelleerida korduvtegevusi nagu logimine, turvalisus, transaktsioonid ja objektide puhverdamine. Selliseid üle terve rakenduse käivitatavaid tegevusi nimetatakse ristlõikuvateks muredeks (*cross-cutting concerns*). Näiteks logimise lisamiseks rakendusse, leiad end korduvalt kirjutamas väga sarnast koodi terves projektis. Lähemalt uurima hakates ei ole sellel koodil mingit seost komponentide tegeliku funktsionaalsusega. AOP aitab eemaldada sellised ristlõikuvad mured ülejäänud rakendusest. AOP realiseerimiseks tuleb eristada äriloogilised komponendid ja neid toetavad komponendid. Viimaseid nimetatakse aspektideks. Järgmisena tuleb kirjeldada ühendpunktid (*join points*), milleks on töövoo punktid. Ühendpunktiks võib olla

väli, konstruktor või meetod, mida käivitades või muutes soovitakse rakendada aspekti. Spring toetab ainult meetodiga ühendpunkte. Aspektide rakendamiseks defineeritakse juhis (*advice*), mis on käivitushetk (enne, pärast, ümber, pärast-erindit ja pärast-naasmist) ühendpunkti suhtes. Juhisele määratakse lõikepunkt (*pointcuts*), milleks on ühendpunktide kogum või regulaaravaldis. (6. AOP)

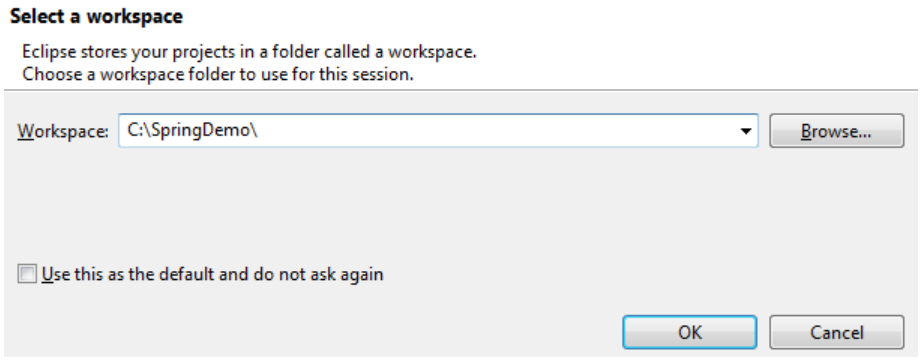
AOP on hea lähenemine kasutamaks korduvtegevusi, kuid ei ole ravim halvale või puudulikule disainile. Väga keeruline on realiseerida aspekte halvasti disainitud süsteemis. Süsteem peab endiselt olema korralikult disainitud kasutades traditsioonilisi meetodeid nagu OOP. (Laddadd, 2003)

3 Arenduskeskkonna seadistamine

Juhendis kasutame Eclipse Java SE Indigo versiooni. Eclipse on küll mitmeplatvormiline, kuid tuleb täheldada, et käesolevas juhendis on näited loodud Windowsi operatsioonisüsteemil ning failisüsteemi viited tuleb teistel platvormidel kasutades viia kooskõlla kasutatava süsteemiga.

3.1 Eclipse paigaldamine

- Laeme alla Eclipse viimase versiooni Indigo aadressilt (Kui Eclipse on juba paigaldatud jätka punktist 3.2
  http://mirror.switch.ch/eclipse/technology/epp/downloads/release/indigo/SR1/eclipse-java-indigo-SR1-win32.zip
- Pakime arhiivi lahti soovitud asukohta näiteks „C:/Program Files/eclipse“ ning käivitame „eclipse.exe“.
- Valime tööala (*workspace*) näiteks „C:/SpringDemo“ kuhu Eclipses loodud projekte hakatakse salvestama. Kui soovid et Eclipse kasutaks valitud tööala vaikeväärtusena märgi linnuke „Use this as the default and do not ask again“ ette.



Ekraanipilt 1. Tööala valimine

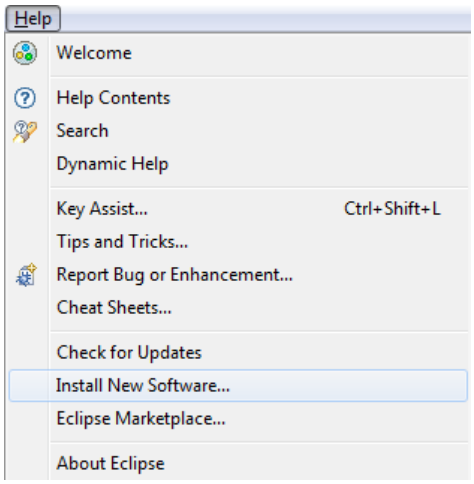
- Eclipse käivitub ja ilmub avakuva (*Welcome*). Mugavamaks jätkamiseks sulgeme avakuva vajutades avakuva sakil asuvale ristile.

Ekraanipilt 2. Avakuva sulgemine.

3.2 Spring IDE tarkvaramooduli paigaldamine

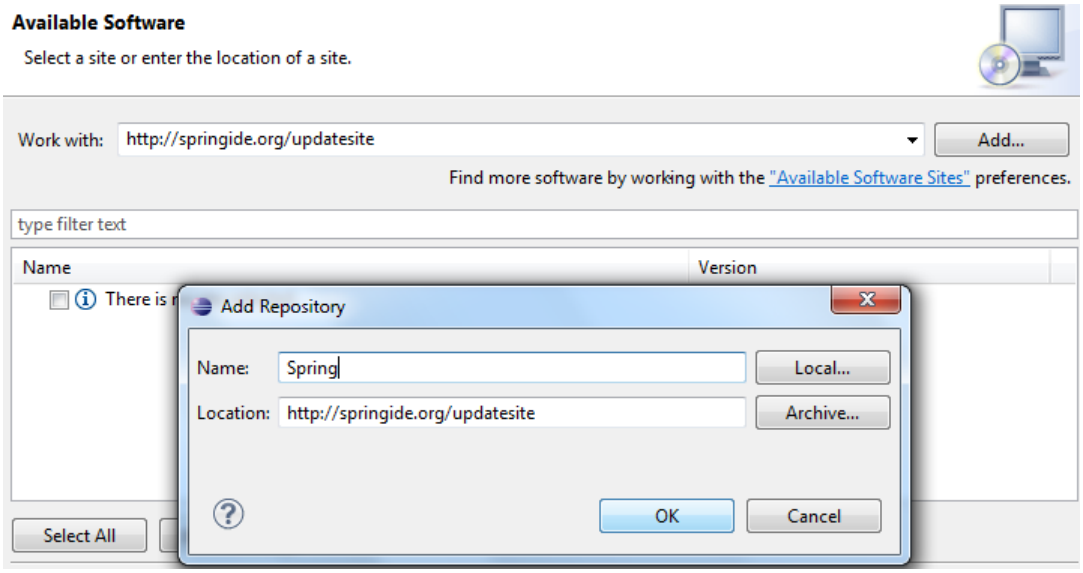
Spring projekti mugavamaks arendamiseks on Eclipse keskkonnale loodud Springi lisamoodulid. Moodulite abiga aitab Eclipse mugavamalt kirjutada Spring projekti. Näiteks luues konfiguratsioonifaile oskab keskkond ise defineerida nimeruumi ja XML skeemifaili. Aspekt-orienteeritud näidete juures illustreerib Eclipse aspekti ja klassidevahelised seosed vastavate noolekestega.

- Spring raamistiku paigaldamiseks Eclipse keskkonda valime „*Help*“ ja „*Install New Software…*“



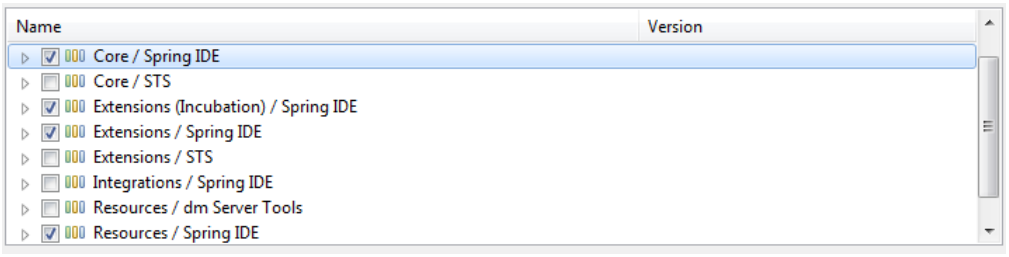
Ekraanipilt 3. Uue tarkvara installeerimine

- Kuvatavas aknas lisame „*Work with*:“ tekstilahtrisse Springi repositooriumi aadressi http://springide.org/updatesite ja vajutame „*Add*“. Hüpikaknas lisame repositooriumile nime ja vajutame „*OK*“.



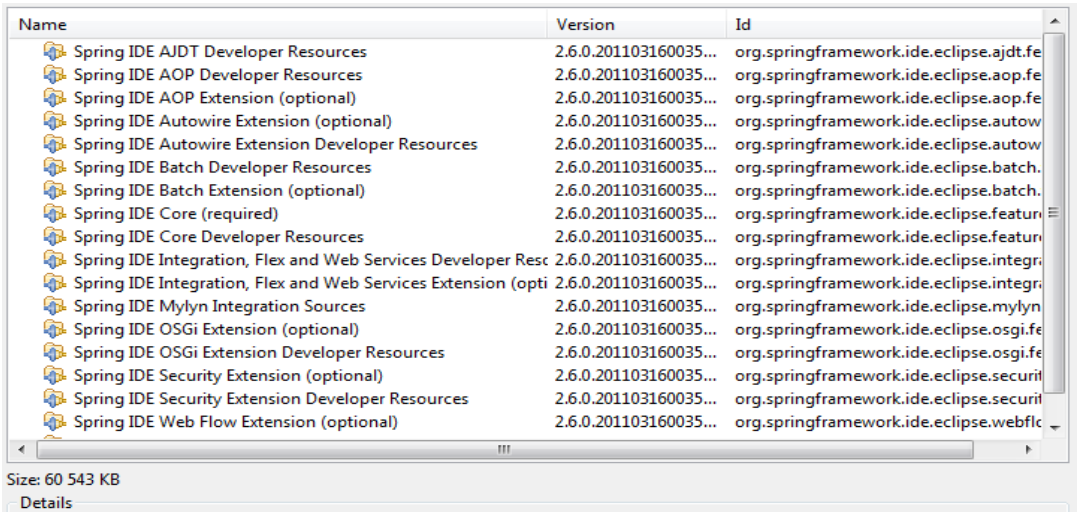
Ekraanipilt 4. Springi repositooriumi lisamine

- Paigaldamiseks valime Core / Spring IDE, Extensions(Incubtation) / Spring IDE, Extensions /Spring IDE, Resources / Spring IDE moodulid ja vajutame „*Next*“



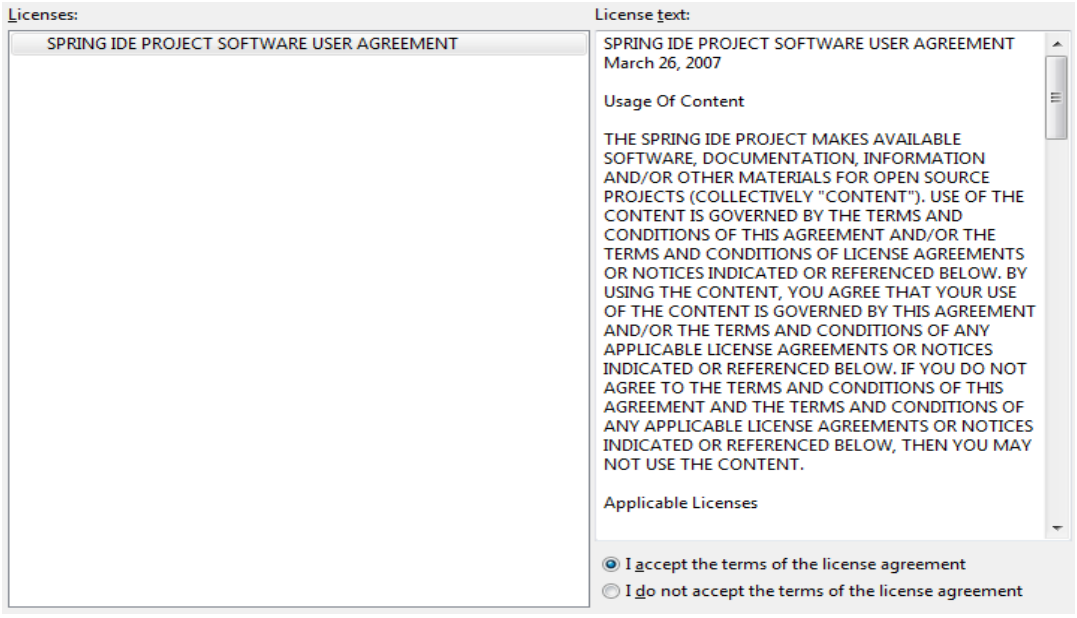
Ekraanipilt 5. Paigaldamiseks valitud moodulid

- Kuvatakse paigaldatavad moodulid ja nende sõltuvused. Jätkamiseks vajutame „*Next*“.



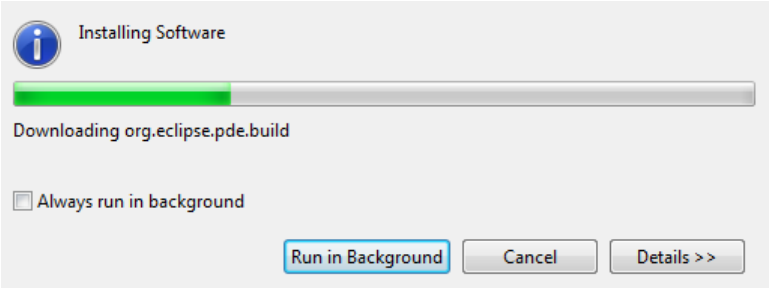
Ekraanipilt 6. Paigaldatavad moodulid

- Kuvatakse litsentsitingimused. Jätkamiseks märgime raadionupu valikuks „*I accept the terms of the license agreement*“ ja vajutame „*Finish*“.



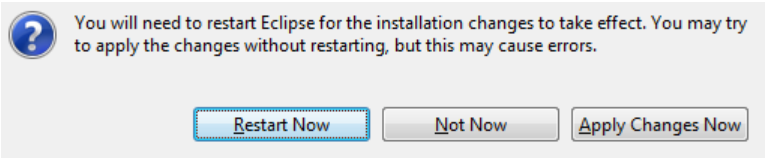
Ekraanipilt 7. Litsentsitingimustega nõustumine

- Algab paigaldus, mille protsessi saab jälgida, peita vajutades „*Run in Background*“, katkestada „*Cancel*“ või vaadata detailsemat infot „*Details*“.



Ekraanipilt 8. Paigalduse protsess

- Paigaldus on lõppenud. Muudatuste sisseviimiseks tuleb Eclipse taaskäivitada. Taaskäivitamiseks vajutame „*Restart Now*“ nuppu.



Ekraanipilt 9. Paigalduse lõpp

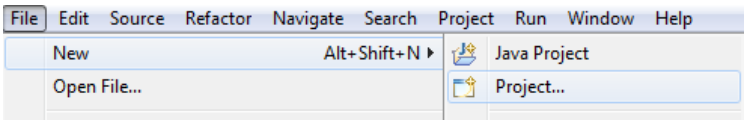
- Eclipse taaskäivitub. Nüüd on Eclipse IDE valmis Spring projekti loomiseks.

4 Sõltuvuste vähendamine Springiga

Selles peatükis vaatame lähemalt, kuidas vähendada sõltuvusi Springi abil. Eesmärgiks on luua mäng, mille klasside vahel ei ole otsest seost, vaid seos kirjeldatakse Spring raamistiku abil. Näiteks valime mängu. Kirjeldame ära meie projektis olevad olulised objektid. Esimene objekt on mängija, kellel on mingi hulk muutujaid, meetodeid ja objektina relv. Mängus on mitmeid eritüüpi mängijaid, kes kasutavad erinevaid relvi. Mängijale relva määramisega tekib tavakoodis sõltuvus mängija ja relva vahel. Kui esialgu määratud relva soovitakse muuta tuleb klassides teha muudatusi. Springi abiga me klasse ei muuda vaid muudame konfiguratsioonifaili. Näitrakenduse valmides on mitu mängija klassi, relva ning nende vahel kirjeldatud ja muudetud sõltuvusi. Sõltuvuste muutmist näeme protsessi käigus. Sellest piisab näitamaks sõltuvuste vähendamist.

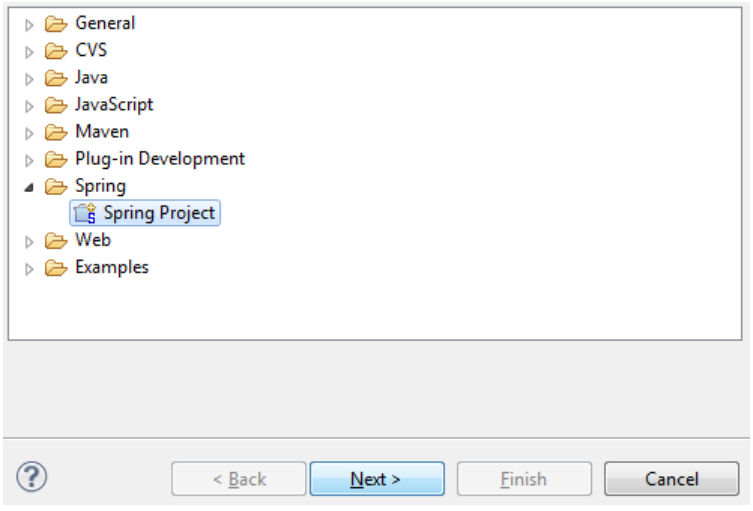
4.1 Projekti loomine

- Uue projekti loomiseks vajutame „*File*“, edasi „*New*“ ja viimaks „*Project*“.



Ekraanipilt 10. Uue projekti loomine

- Uue projekti viisardis valime Spring kataloogist „*Spring Project*“ ja vajutame „*Next*“ nuppu

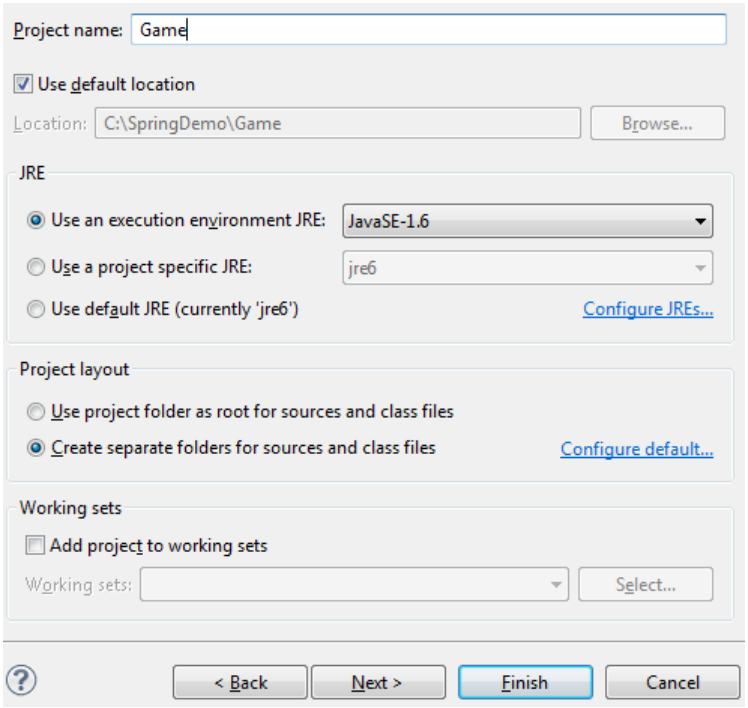


.

Ekraanipilt 11. Spring projekti valimine

- Projekti seaded. Määrame projekti nimeks „*Game*“. Lisaks on võimalik muuta Java Runtime Environmenti. Vaikeväärtusena kasutab süsteemi konfiguratsiooni. „*Project*

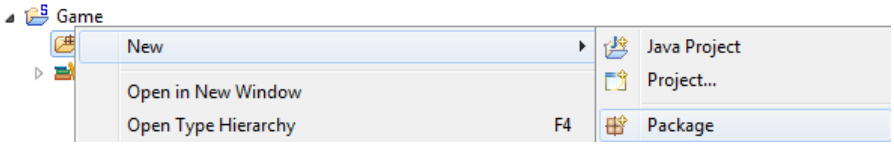
layout“ alt on võimalik muuta kuidas faile projekti kaustas hoitakse. Vaikeväärtusena luuakse lähtekoodi ja „*class*“ failidele eraldi kaustad.



Ekraanipilt 12. Projekti seaded

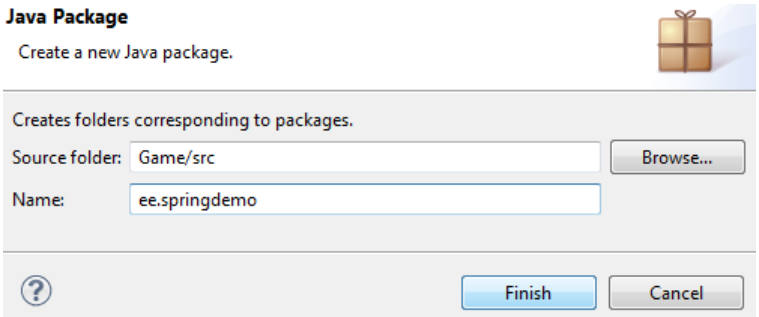
4.2 Paketi ja kausta loomine

- Teeme „src“ kausta peal parema kliki, valime „*New*“ ja „*Package*“.



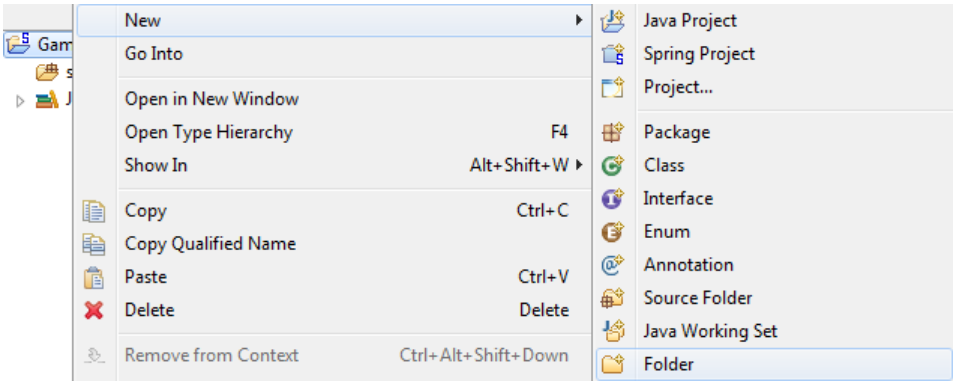
Ekraanipilt 13. Paketi loomine

- Määrame nime „ee.springdemo“ ja jätkamiseks „*Finish*“.



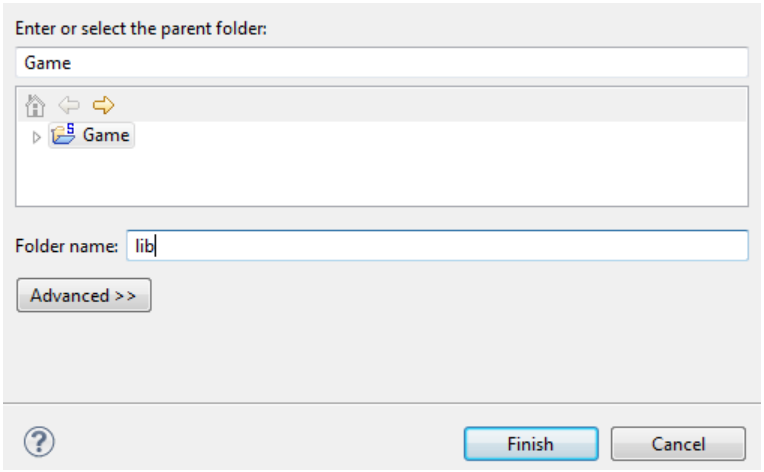
Ekraanipilt 14. Paketi nime määramine

- Loome projekti alla uue kausta. Teeme projektil parema kliki valime „*New*“ ja „*Folder*“.



Ekraanipilt 15. Kausta loomine.

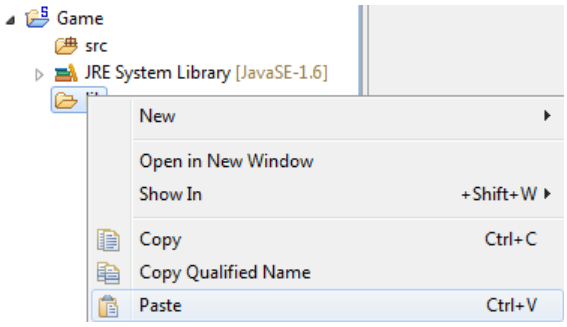
- Määrame ülemkaustaks projektikausta „*Game*“ ning nimeks „lib“ ja vajutame „*Finish*“ nuppu.



Ekraanipilt 16. Kaustale nime määramine.

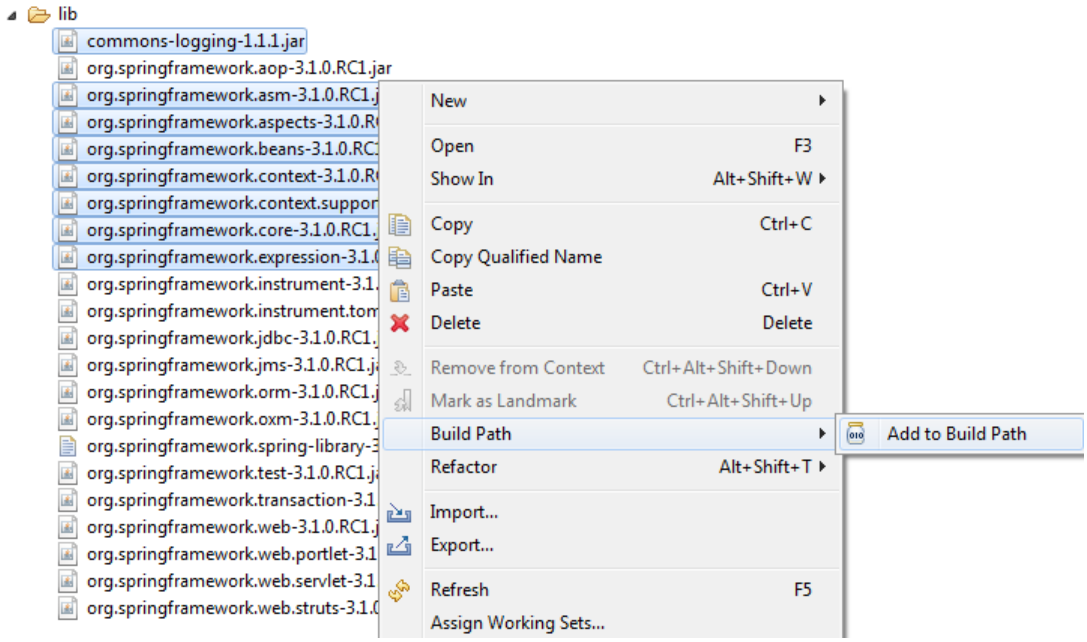
4.3 Spring raamistiku ja sõltuvuste haakimine

- Spring raamistiku kasutamiseks projektis vajame raamistikku sisaldavaid pakette. Laeme alla pakette sisaldava arhiivi:
  http://s3.amazonaws.com/dist.springframework.org/milestone/SPR/spring-framework-3.1.0.RC1.zip. Pakime arhiivi lahti ning kopeerime kausta „dist“ sisu Eclipse projekti äsja loodud „C:\SpringDemo\Game\lib“ kausta. Kopeerimiseks võime kasutada operatsioonisüsteemi failihaldurit või asetame (*paste*) Eclipse keskkonnas vastavasse kausta.



Ekraanipilt 17. Asetamine (*paste*) lib kausta

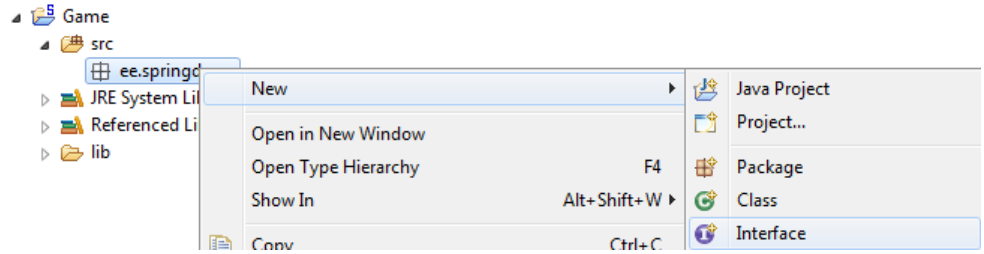
- Springi tuummooduli ainsaks sõltuvuseks on logimine. Spring peab seda väga oluliseks sõltuvuseks kuna alati peab näha olema kasutatava tööriista logid. Logimiseks kasutab Spring „Jakarta Commons Logging“ moodulit. Laeme alla arhiivi http://mirror.its.uidaho.edu/pub/apache//commons/logging/binaries/commons-logging-1.1.1-bin.zip
- Pakime lahti ning asetame „commons-logging-1.1.1.jar“ paketi samuti „lib“ kausta.
- Klassiteesse (*classpath*) lisamine. Avame „lib“ kausta Eclipses. Muudame aktiivseks järgmised paketid:
  commons-logging-1.1.1.jar
  org.springframework.asm-3.1.0.RC1.jar
  org.springframework.aspects-3.1.0.RC1.jar
  org.springframework.beans-3.1.0.RC1.jar
  org.springframework.context-3.1.0.RC1.jar
  org.springframework.context.support-3.1.0.RC1.jar
  org.springframework.core-3.1.0.RC1.jar
  org.springframework.expression-3.1.0.RC1.jar
  Teeme parema kliki aktiivsetel pakettidel ning valime „*Build Path*“ ja „*Add To Build Path*“.



Ekraanipilt 18. JAR arhiivide lisamine klassiteesse

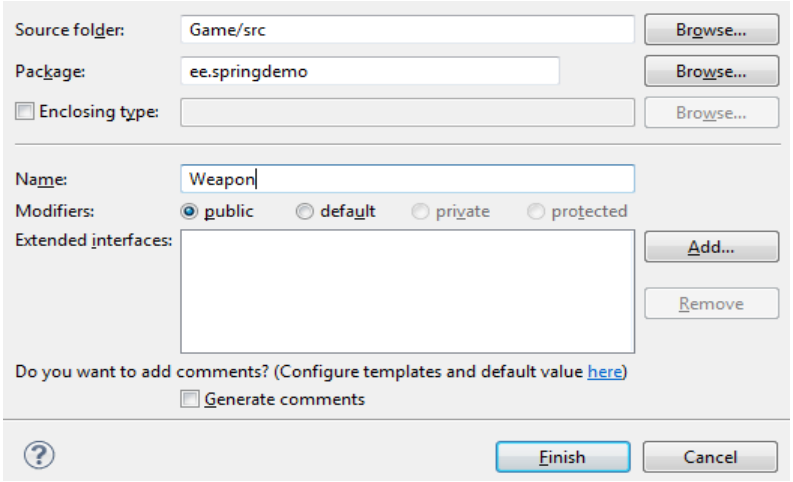
4.4 Liideste ja klasside loomine

- Loome „ee.springdemo“ paketti relva liidese. Selleks teeme nimetatud paketil parema kliki ning valime „*New*“ ja „*Interface*“.



Ekraanipilt 19. Liidese loomine

- Liideseseadete alt saab määrata, mis kausta ja paketti liides luuakse. Määrame liidese nimeks „*Weapon*“ ja vajutame „*Finish*“.



Ekraanipilt 20. Liidese seaded ning nime määramine

- Loome liidesele kolm meetodit: getName(), makeDamage() ja describe().

```
package ee.springdemo;

public interface Weapon {
	public String getName();
	public void makeDamage();
	public void describe();
}
```

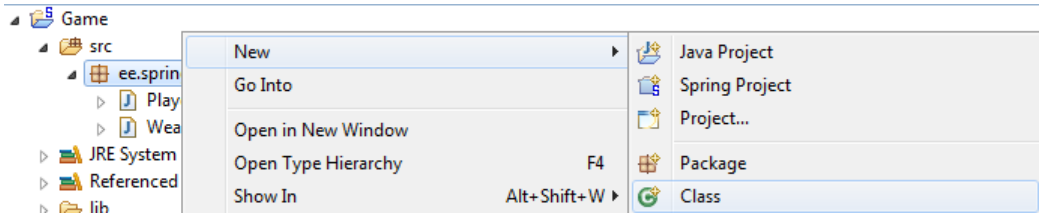
Koodinäide 1. Liides *Weapon*

- Loome teise liidese nimega „*Player*“, mis sisaldab mängija meetodeid: setName(String name), getName(), setWeapon(Weapon weapon), getWeapon(), beginGame(), endGame() ja move(int xChange, int yChange). Meetodite sisust räägime lähemalt vastavat liidest realiseerivate klasside juures.

```
package ee.springdemo;
public interface Player {
	public void setName(String name);
	public String getName();
	public void setWeapon(Weapon weapon);
	public Weapon getWeapon();
	public void beginGame();
	public void endGame();
	public void move(int xChange, int yChange);
}
```

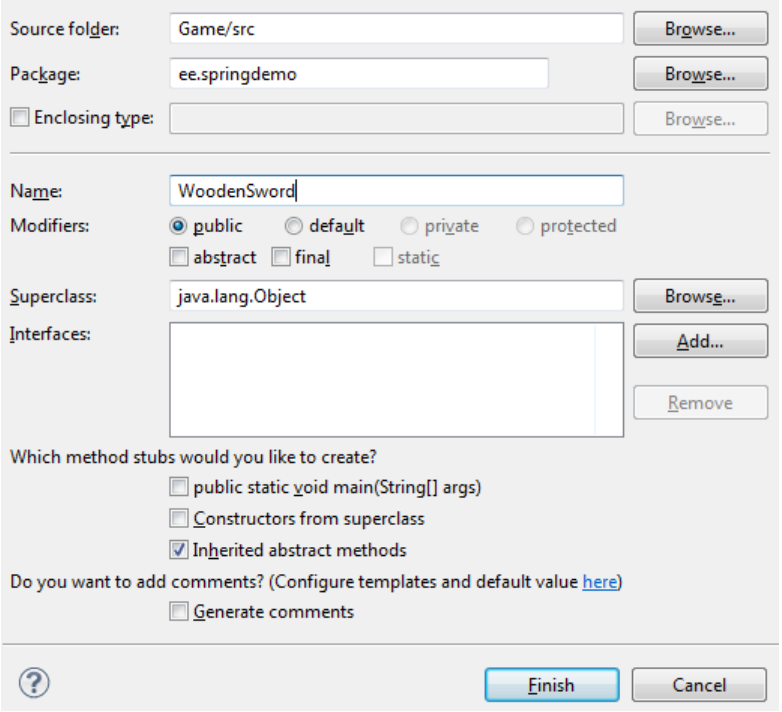
Koodinäide 2. Liides *Player*

- Jätkame esimese relva loomisega, milleks on puumõõk (*WoodenSword*). Teeme parema kliki „ee.springdemo“ paketil vajutame „*New“* ning „*Class*“.



Ekraanipilt 21. Klassi loomine

Klassiseadete alt saab määrata, mis kausta (*source folder*) ning paketti (*package)* klass luuakse. Lisaks saab määrata klassi muutmisõigused (*modifiers*) avalikuks (*public*), vaikeväärtuseks (*default*), abstraktseks (*abstract*) ja lõplikuks (*final*). Mugavalt on võimalik genereerida main meetod. Määrame klassinimeks *WoodenSword* ning vajutame „*Finish“*.



Ekraanipilt 22. Klassiseaded

- Määrame klassi realiseerima *Weapon* liidest. Viimase defineerimiseks kirjutame klassi kirjeldamise lõppu „implements Weapon“.

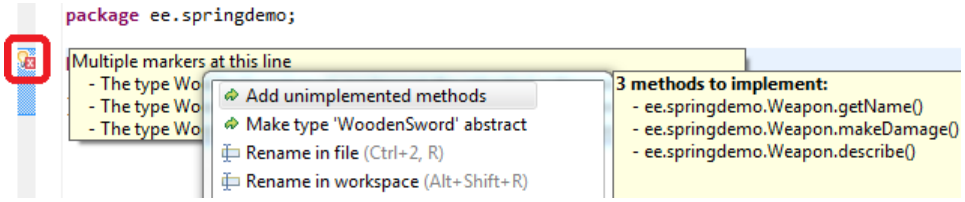
```
package ee.springdemo;
public class WoodenSword implements Weapon{

}
```

Koodinäide 3. Klass *WoodenSword* realiseerib *Weapon* liidest

- Rea „public class WoodenSword implements Weapon {„ juures on veamärk. Realiseerides liidest peab klass sisaldama kõiki vastava liidese meetodeid. Veamärgil

vajutades pakub Eclipse lahendusi ning vastavate meetodite lisamiseks vajutame „*Add unimplemented methods*“ nupul.



Ekraanipilt 23. Realiseerimata meetodite lisamine

- Eclipse genereerib üledefineeritud (*@Override*) meetodid .

```
@Override
public String getName() {
	// TODO Auto-generated method stub
	return null;
}
@Override
public void makeDamage() {
	// TODO Auto-generated method stub
}
@Override
public void describe() {
	// TODO Auto-generated method stub
}
```

Koodinäide 4. Genereeritud meetodid

- Defineerime meetodite sisu. Meetod getName() tagastab sõne „WoodenSword“, makeDamage() trükib konsooli „WoodenSword is for practice“ ning describe() trükib sõne „WoodenSword“ ning kutsub välja makeDamage(). Sellega on WoodenSword klassi loomine lõppenud.

```
package ee.springdemo;
public class WoodenSword implements Weapon{
	@Override
	public String getName(){
		return "WoodenSword";
	}
	@Override
	public void makeDamage() {
		System.out.println("WoodenSword is for practice");
	}
	@Override
	public void describe(){
		System.out.println("WoodenSword");
		makeDamage();
	}
}
```

Koodinäide 5. WoodenSword klass

- Järgmiseks loome algtaseme mängija klassi nimega „*Newbie*“. Newbie klass realiseerib Player liidest. Klassi deklaratsiooni juurde lisame liidese realiseerimise käsu „implements Player“.

```
package ee.springdemo;
public class Newbie implements Player {
}
```

Koodinäide 6. Klass Newbie realiseerib Player liidest

- Player liidest realiseerides tuleb lisada liidese meetodid. Kasutame jällegi Eclipse abi ja genereerime meetodid.
- Deklareerime muutujad String name, Weapon weapon, int x ja int y. Loome konstruktori public Newbie (Weapon weapon). Klassi ainus konstruktor, mis vajab sisendiks relva ehk Weapon liidest realiseerivat klassi.

```
String name;
Weapon weapon;
int x=0, y=0;
public Newbie(Weapon weapon){
  this.weapon = weapon;
}
```

Koodinäide 7. Klassi Newbie muutujad ja konstruktor

- Kirjutame genereeritud meetoditele sisu. Meetodi setName(String name) ülesandeks on omistada klassi muutujale *this.name* parameeter *name* väärtuse, getName() tagastab muutuja *name* väärtuse, getWeapon tagastab *Weapon* objekti, beginGame() trükib mängijat kirjeldava teksti ja info mängu alguse kohta, endGame() trükib mängu lõppemise info, move(int xChange, int yChange) liidab muutujatele x ja y sisendina saadud väärtused. Sellega on *Newbie* klass defineeritud.

```
package ee.springdemo;
public class Newbie implements Player {
	String name;
	Weapon weapon;
	int x=0, y=0;
	public Newbie(Weapon weapon){
		this.weapon = weapon;
	}
	@Override
	public void setName(String name) {
		this.name= name;
	}
	@Override
	public String getName() {
		return this.name;
	}
	@Override
	public void setWeapon(Weapon weapon) {
		this.weapon = weapon;
	}
	@Override
	public Weapon getWeapon() {
		return weapon;
	}
	@Override
	public void beginGame() {
		System.out.println("I'm just newbie. Learning to fight with "
				+ weapon.getName());
	}
	@Override
	public void endGame() {
		System.out.println("I'm still a newbie and I'm dying "
				+ "already...");
	}
	@Override
	public void move(int xChange, int yChange) {
		this.x+=xChange;
		this.y+=yChange;
	}

}
```

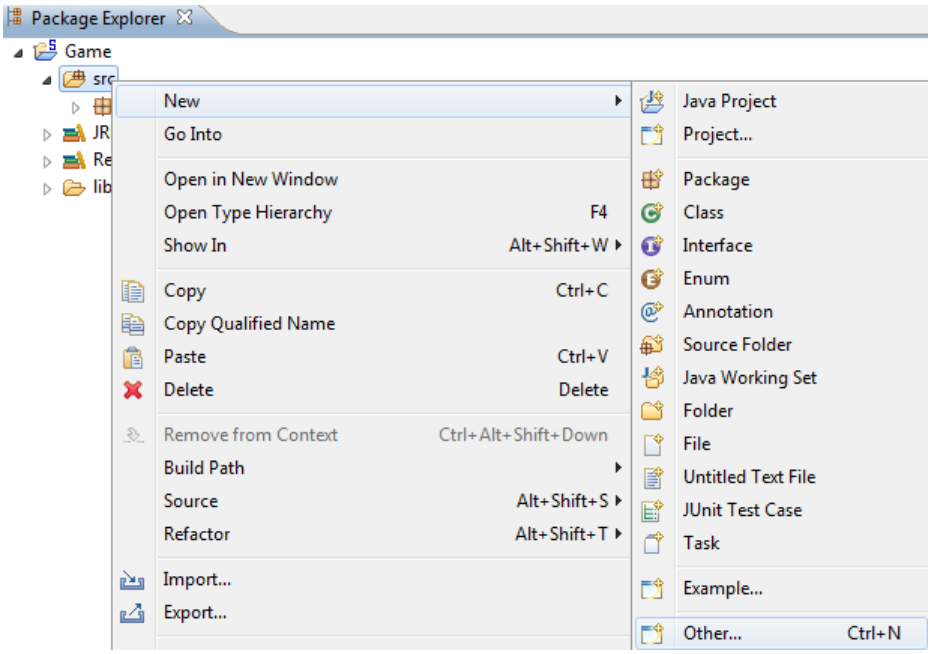
Koodinäide 8. Klass *Newbie*

4.5 Ubade kaabeldamine

Seoste loomist komponentide vahel nimetatakse tavaliselt kaabeldamiseks (*wiring*). Springis on mitmeid võimalusi kaabeldamiseks, aga harilikult tehakse seda XML teel (Walss, 2011).

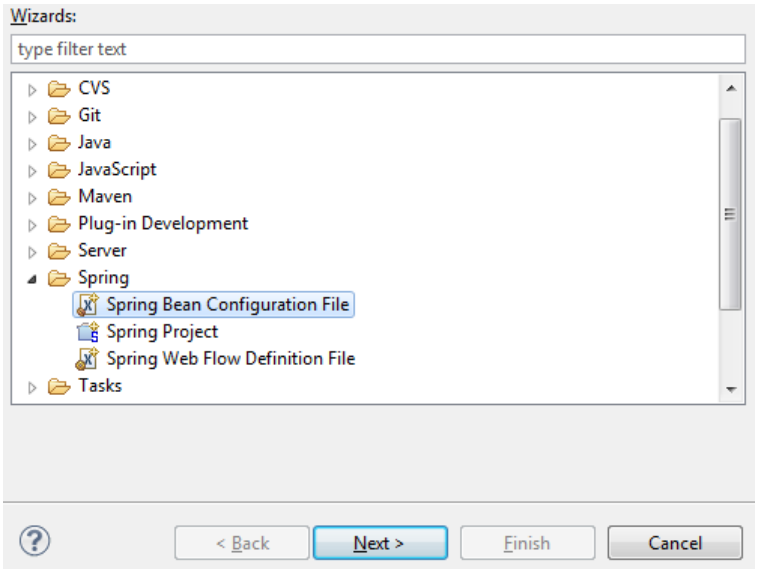
Näites kaabeldamise eesmärgiks on eemaldada sõltuvused mängija ja relva vahel. Kaabeldatud koodis on lihtsam muuta mängija ja relva sõltuvusi. Muutmiseks tuleb muudatused sisse viia ubade konfiguratsioonifailis, Java klassifailid jäävad aga muutmata. Näites loome kaabeldatud sõltuvuse *Newbie* klassi ja *WoodenSword* vahel. Peale esimese kaabelduse õnnestumist loome uue mängija ja relva klassi ning muudame kaabeldamist. Alustame esimese kaabeldamisega.

- Loome Springi konfiguratsioonifaili, teeme „src“ kaustal parema kliki, valime „*New*“ ja seejärel „*Other*“.



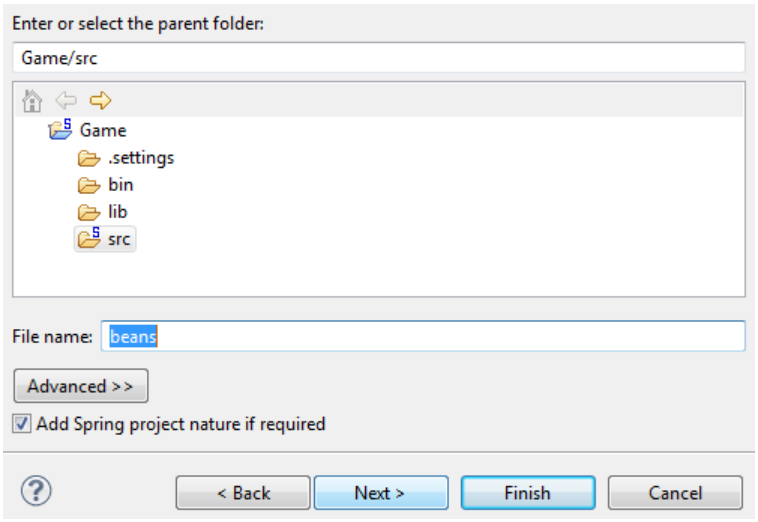
Ekraanipilt 24. Konfiguratsioonifaili loomine

- Hüpikaknast valime „*Spring*“ ja „*Spring Bean Configuration File*“, seejärel „*Next*“.



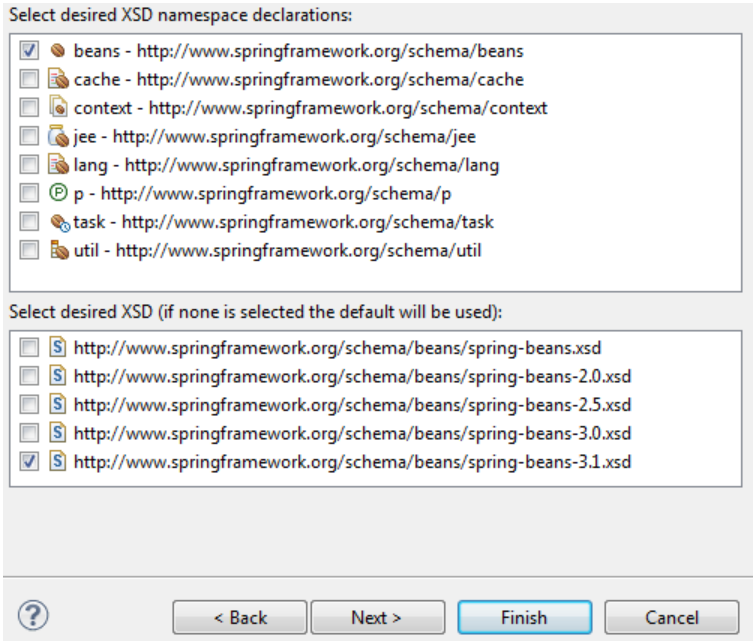
Ekraanipilt 25. Spring ubade konfiguratsiooni faili valimine

- Sisestame nimeks „*beans*”. Jätkamiseks „*Next*“.



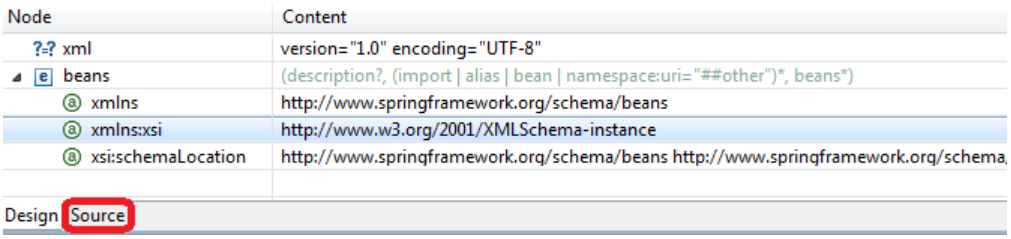
Ekraanipilt 26. Konfiguratsioonifaili nime määramine

- Ülemisel väljal kuvatakse võimalikud XSD (XML skeemi definitsioon) nimeruumid. Nende lisamiseks tuleb teha linnuke vastava nimeruumi ette. Märgime linnukese „beans – http://www.springframework.org/schema/beans“ ette. Seejärel alumisel väljal kuvatakse vastava nimeruumi XSD versioonid. Valime kõige uuema versiooni „http://www.springframework.org/schema/beans/spring-beans-3.1xsd“ ning vajutame „*Finish*“. Eclipse loob „beans.xml“ faili.



Ekraanipilt 27. XSD nimeruum ja nimeruumi versioon

- Avame loodud faili ning liigume lähtekoodi vaatele vajutades „*Source*“ sakil.



Ekraanipilt 28. Beans.xml disainivaade ja Source sakk

- Genereeritud sisu on järgmine:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.1.xsd"
</beans>
```

Koodinäide 9. Beans.xml sisu

- Hakkame kirjeldama ube. Ubade järgi võtame hiljem „*main*“ meetodit sisaldavas klassis kasutusele kaabeldatud objektid. Esimesena loome oa (*bean*) elemendi, mille identifikaatoriks (*id*) määrame „wsword“ ja klassiks (*class*) „ee.springdemo.WoodenSword“. Identifikaatori järgi pöördutakse vastava oa poole, klass määrab kasutatava klassi. Loome teise oa, mille identifikaatoriks määrame „noob“ ja klassiks „ee.springdemo.Newbie“. Lisaks defineerime skoobi (*scope*). Vaikeväärtusena on oa skoobiks üksikeksemplar (*singleton*). Üksikeksemplarina luuakse Springi konteineris alati ainult üks klassi isend. Kui soovime ühe konteineri piires luua rohkem kui ühe isendi siis tuleb skoobina kasutada prototüüpi (*prototype*). Lisaks loome noob oale konstruktori argumentide (*constructor-arg)* elemendi. Viimasega määrame noob oale konstruktori väljakutsudes kaasa wsword oa.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">
       <bean id="wsword" class="ee.springdemo.WoodenSword"/>
       <bean id="noob" class="ee.springdemo.Newbie" scope="prototype">
             <constructor-arg ref="wsword" />
       </bean>
</beans>
```

Koodinäide 10. beans.xml koos loodud ubadega

4.6 Main meetod ja käivitamine

Vajalikud oad on defineeritud. Nüüd järgmise tegevusena oleme valmis käivitama projekti. Selle jaoks peame looma „main“ meetodiga klassi.

- Loome klassi NewbieMain, mis sisaldab main meetodit. Impordime kaks Springi klassi. „*ApplicationContext*“ laeb ja ühendab ubade definitsioonid. Viimane vastutab täielikult objektide loomise ja kaabeldamise eest. Springis on mitmeid ApplicationContexti lahendusi, mis erinevad konfiguratsiooni laadimise poolelt. Kuna kasutame faili „beans.xml“, mis on XML kujul siis meile sobib „*ClassPathXmlApplicationContext*“. See lahendus laeb konfiguratsiooni ühest või mitmest rakenduse klassitees (*classpath*) asuvast XML failist. (Walss, 2011)

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext
```

Koodinäide 11 NewbieMain klassi importimised

- Loome ApplicationContexti isendi kasutades ClassPathXmlApplicationContexti ja beans.xml faili. Isend seob XML failis kirjeldatud oad. ApplicationContexti getBean() meetodi abil küsime „noob“ identifikaatorile vastava oa. Viimane on kaabeldatud Newbie klassiga, mis realiseerib Player liidest. Meetod aga tagastab „*Object*“ klassi isendi, mille teisendame Player eksemplariks.

```
ApplicationContext context =
                    new ClassPathXmlApplicationContext("beans.xml");
Player noob = (Player) context.getBean("noob");
```

Koodinäide 12. ApplicationContexti loomine ja oa küsimine

- Edasi saame kasutada noob objekti nagu tavalist Player liidesele vastavat isendit, teadmata NewbieMain koodis, millist klassi kasutatakse. Demonstreerimiseks käivitame järgmised meetodid: setName(), beginGame(), getWeapon().describe() ja endGame().

```
package ee.springdemo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class NewbieMain {
  public static void main(String[] args){
    ApplicationContext context =
                        new ClassPathXmlApplicationContext("beans.xml");
    Player noob = (Player) context.getBean("noob");
    noob.setName("New Noob");
    noob.beginGame();
    noob.getWeapon().describe();
    noob.endGame();
  }
}
```

Koodinäide 13. Klass NewbieMain

- Käivitame NewbieMain klassi vajutades CTRL + F11. Spring logib „Jakarta Commons Logging“ abil protsessi järgmised tegevused: ApplicationContexti värskenduseks valmistumine, värskendamine, ClassPathXmlApplicationContexti käivitamine, ubade

lugemine, üksikeksemplaride initsialiseerimine ja ubade defineerimine (loetletakse leitud oad). Rakenduse väljundis on Newbie klassi meetodite väljatrükid.

```
10.10.2011 21:25:08
org.springframework.context.support.AbstractApplicationContext
prepareRefresh
INFO: Refreshing
org.springframework.context.support.ClassPathXmlApplicationContext@b6
1d36b: startup date [Mon Oct 10 21:25:08 EEST 2011]; root of context
hierarchy
10.10.2011 21:25:08
org.springframework.beans.factory.xml.XmlBeanDefinitionReader
loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource
[beans.xml]
10.10.2011 21:25:08
org.springframework.beans.factory.support.DefaultListableBeanFactory
preInstantiateSingletons
INFO: Pre-instantiating singletons in
org.springframework.beans.factory.support.DefaultListableBeanFactory@
c5a67c9: defining beans [wsword,noob]; root of factory hierarchy
I'm just newbie. Learning to fight with WoodenSword
WoodenSword
WoodenSword is for practice
I'm still a newbie and I'm dying already...
```

Koodinäide 14. Logi ja rakenduse väljund

4.7 Ubade muutmine

- Loome puukirves klassi, mis realiseerib *Weapon* liidest. Nimeks määrame „*WoodenAxe*“. Lisame Eclipse abil realiseerimata meetodid. Kirjutame genereeritud meetoditele sisu.

```
package ee.springdemo;
public class WoodenAxe implements Weapon{
	@Override
	public String getName() {
		return "WoodenAxe";
	}
	@Override
	public void makeDamage() {
		System.out.println("Wooden Axe can kill bugs");
	}
	@Override
	public void describe() {
		System.out.println("WoodenAxe");
		makeDamage();
	}
}
```

Koodinäide 15. Klass *WoodenAxe*

- Loome uue edasijõudnud mängija „*AdvancedNewbie*“ klassi. Viimane realiseerib Player liidest. Laseme Eclipsel genereerida vastavad meetodid ja kirjutame nendele sisu. AdvancedNewbie sisu võiks erineda Newbie beginGame() ja endGame() meetodite poolest.

```
package ee.springdemo;
public class Newbie implements Player {
	String name;
	Weapon weapon;
	int x=0, y=0;
	public Newbie(Weapon weapon){
		this.weapon = weapon;
	}
	@Override
	public void setName(String name) {
		this.name= name;
	}
	@Override
	public String getName() {
		return this.name;
	}
	@Override
	public void setWeapon(Weapon weapon) {
		this.weapon = weapon;
	}
	@Override
	public Weapon getWeapon() {
		return weapon;
	}
	@Override
	public void beginGame() {
		System.out.println("I'm just newbie. Learning to fight "
				+ "with" + weapon.getName());
	}
	@Override
	public void endGame() {
		System.out.println("I'm still a newbie and I'm dying "
				+ "already...");
	}
	@Override
	public void move(int xChange, int yChange) {
		this.x+=xChange;
		this.y+=yChange;
	}
}
```

Koodinäide 16. Klass AdvancedNewbie

- Tõestamaks, et seotud klasside muutmine on loodud väga lihtsaks teeme muudatusi beans.xml faili. Loome uue oa mille id on woodenaxe ja klassiks „ee.springdemo.WoodenAxe“. Noob klassi asendame „ee.springdemo.AdvancedNoob“ klassiga, konstruktori argumendi wsord oa asendame woodenaxe oaga.

```
<bean id="noob" class="ee.springdemo.AdvancedNewbie"
                scope="prototype">
      <constructor-arg ref="woodenaxe" />
</bean>
<bean id="woodenaxe" class="ee.springdemo.WoodenAxe" />
```

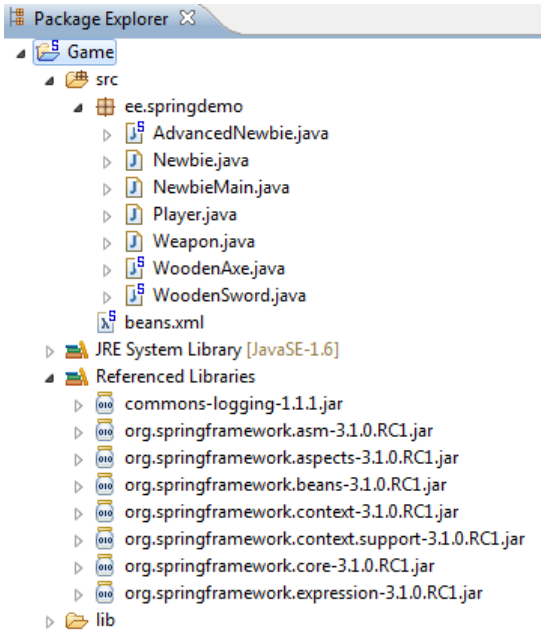
Koodinäide 17. Uus uba woodenaxe ja noob oa muudatused

- Käivitame projekti. Logi viimasest osast näeme, et seekord on laetud 3 uba (wsword, noob ja woodenaxe). Käivituse tulemis on nüüd *AdvancedNoob* ja *WoodenAxe* objektide väljatrükid.

```
INFO: Pre-instantiating singletons in
org.springframework.beans.factory.support.DefaultListableBeanFactory@7
0453807: defining beans [wsword,noob,woodenaxe]; root of factory
hierarchy
I can already fight with WoodenAxe
WoodenAxe
Wooden Axe can kill bugs
I'm advanced newbie and I want to fight little bit more...
```

Koodinäide 18. Osaline logi ja käivituse tulem

- Murede korral on võimalik võrrelda projekti failipuud ning lisaks on selle punkti lõpuks valminud projekt saadaval aadressil: http://www.tlu.ee/~maehans/Seminar/Game_DI.rar



Ekraanipilt 29. Projekti failipuu

4.8 Ülesanded

- Loo uus Player liidest realiseeriv klass Warrior.

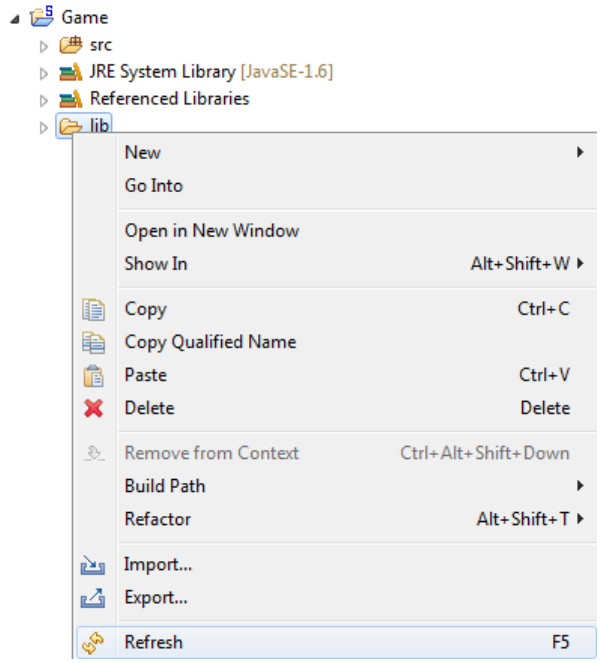
- Loo uus Weapon liidest realiseeriv klass SteelSword.
- Loo vastavad oad beans.xml faili.
- Katseta Warrior objekti loomist.

5 Aspekt-orienteeritud programmeerimine

Eelnevalt vaatasime, kuidas Spring aitab vähendada sõltuvusi komponentide vahel, nüüd jätkame sõltuvuste vähendamist korduvtegevuste juures. Jätkame valminud näite arendamist (siit punktist alustamiseks lae projekt http://www.tlu.ee/~maehans/Seminar/Game_DI.rar aadressilt).

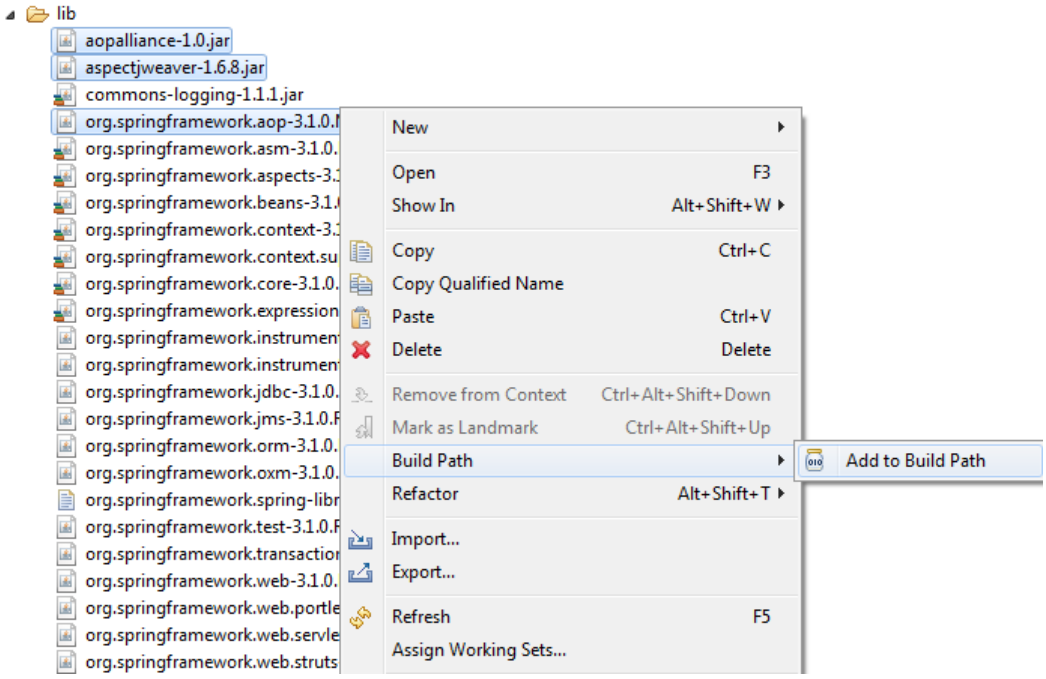
5.1 Spring AOP ja sõltuvuste haakimine

- AOP kasutamiseks peame projektile lisama kolm moodulit klassiteesse. Spring AOP moodulil on kaks sõltuvust.
- Laeme alla AOPAlliance mooduli salvestades projekti „lib“ kausta: http://repo1.maven.org/maven2/aopalliance/aopalliance/1.0/aopalliance-1.0.jar
- Laeme alla teise sõltuvuse AspectJWeaver mooduli salvestades samuti projekti „lib“ kausta: http://repo1.maven.org/maven2/org/aspectj/aspectjweaver/1.6.8/aspectjweaver-1.6.8.jar
- Kolmas moodul org.springframework.aop-3.1.0.M2.jar on juba „lib“ kaustas olemas, kuid klassiteesse lisamata.
- Värskendame kausta sisu selekteerides „lib“ kausta ja vajutades F5 või teeme parema kliki ja valime „*Refresh*“.



Ekraanipilt 30. Kausta värskendamine

- Selekteerime nimetatud arhiivid, teeme parema kliki ning valime „*Build Path*“ ja „*Add to Build Path*“.



Ekraanipilt 31. Pakettide klassiteesse lisamine

5.2 Logger klass ja kaabeldamine

- Loome klassi Logger, mis hakkab tegelema logimistega. Klassi loome privaatse meetodi getDateTime(). Meetodi ülesandeks on tagastada kuupäev ja kellaaeg soovitud vormingus. Soovitud tulemi saamiseks impordime java.util.Calendar klassi, mille eksemplarilt saame küsida kellaaja. Soovitud vormingu jaoks impordime java.text.SimpleDateFormat klassi, mille isend vormindab kuupäeva ja kellaaja etteantud kujule. Teeme teise public meetodi logIt(), mis trükib konsooli meetodi getDateTime() tulemi ning „Something happened“ teksti .

```
package ee.springdemo;
import java.text.SimpleDateFormat;
import java.util.Calendar;
public class Logger {
	private String getDateTime(){
		return new SimpleDateFormat("yyyy.MM.dd HH:mm:ss")
		.format(Calendar.getInstance().getTime());
	}
	public void logIt(){
		System.out.println(getDateTime() + " Something happened");
	}
}
```

Koodinäide 19. Klass Logger

- Me tahame logida kõiki „noob“ oa käivitusi. Lisame beans.xml faili AOP nimeruumi ja skeemi definitsiooni (kui luua uus Springi ubade konfiguratsiooni fail siis vastava koodi saab automaatselt genereerida).

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
      http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">
```

Koodinäide 20. AOP nimeruum ja skeemi definitsioon

- Kirjutame klassile Logger vastava oa, mille identifikaatoriks määrame logger ja klassiks ee.springdemo.Logger.

```
<bean id="logger" class="ee.springdemo.Logger" />
```

Koodinäide 21. Uba logger

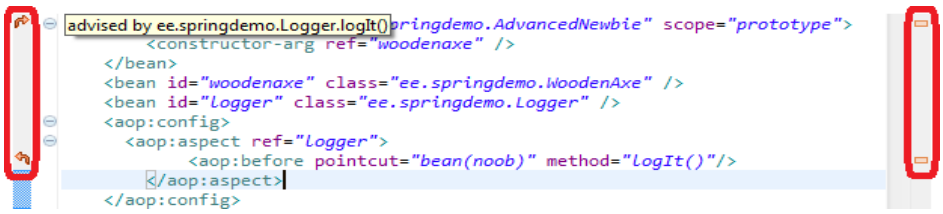
- Järgmisena kirjutame aspekt-orienteeritud oad. Kirjutades tuleb ära määrata aspekt (*aspect*), juhis (*advice*) ja lõikepunkt (*pointcut*). Aspekti all mõeldakse uba, mida soovime kasutada aspektina. Juhises määratakse lõikepunkt ja aspekti meetod. Juhiseid on viite tüüpi: enne (*before*), pärast (*after*), pärast-naasmist (*after-returning*), pärast-erindit (*after-throw*) ja ümber (*around*). Lõikepunkt määrab tingimuse, millal aspekti meetod käivitatakse. Lõikepunkti saab defineerida mitmel tingimusel: argumentide (*args*), oa (*bean*), käivitatava meetodi (*execution*), klassi (*within*), iga käivituse (*this*) ning sihtmärgi (*target*) alusel. Lisaks saab viimaseid loogikaoperaatoritega ühendada. (9. Spring AOP APIs)

  Esimeses näites kasutame lõikepunktina „noob“ uba. Aspektiks valime „logger“ oa „logIt“ meetodi. Viimase käivitame enne (*before*) lõikepunkti.

```
<aop:config>
      <aop:aspect ref="logger">
            <aop:before pointcut="bean(noob)" method="logIt()"/>
      </aop:aspect>
</aop:config>
```

Koodinäide 22. Esimese aspekti konfiguratsioon

- Kaabeldamise õnnestumist näitab Eclipse ubade konfiguratsiooni failis ja vastavate Java klasside failis pruunide noolekestega vasakul (noolekeste suund näitab, kas käivitatakse enne, pärast, ümber, pärast-naasmist või pärast-erindit) ja infokastidega paremal. Info noolele või kastile liikudes kuvatakse info vastava kaabelduse kohta.



Ekraanipilt 32. AOP kaabeldamise info.

5.3 AOP projekti käivitamine

- Nüüd oleme valmis käivitama aspekt-orienteeritud projekti. Käivitamiseks vajutame taaskord CTRL + F11. Tulemist näeme, et seekord on laetud 6 uba (wsword, noob, woodenaxe, logger ja kaks AOP klassi). Rakenduse tulemis on näha, et iga kord kui pöördutakse noob oa klassi poole käivitub eelnevalt logger oa logIt() meetod.

```
INFO: Pre-instantiating singletons in
org.springframework.beans.factory.support.DefaultListableBeanFactory@2
af081: defining beans
[wsword,noob,woodenaxe,logger,org.springframework.aop.config.internalA
utoProxyCreator,org.springframework.aop.aspectj.AspectJPointcutAdvisor
#0]; root of factory hierarchy
2011.10.11 20:52:30 Something happened
2011.10.11 20:52:30 Something happened
I can already fight with WoodenAxe
2011.10.11 20:52:30 Something happened
Wooden Axe can kill bugs
2011.10.11 20:52:30 Something happened
I'm advanced newbie and I want to fight little bit more...
```

Koodinäide 23. Käivituse tulem

5.4 Uus lõikepunkt, loogikaoperaator ja logimismeetod

- Loodud aspekt logib kõiki pöördumisi „noob“ oa poole ja seetõttu tuleb palju korduvat infot. Rakenduse tulemi loetavuse ja loogikaoperaatorite kasutamise eesmärgil kasutame „ja“ (*and*) ning „ei“ (*not*) operaatorit. Lisaks kasutame klassi (*within*) ühendpunkti. Klassimääramisel tuleb sisestada klassinimi koos paketiga. Kui paketi- või klassinime asemel märgime tärni (*) võetakse kõik vastavas paketis olevad paketid või klassid lõikepunktiks. Klassina kasutame *.*.AdvancedNewbie. Vastaval juhul võetakse kõik AdvancedNewbie klassid, mis asuvad teise astme paketis. Meie klassitees vastab vaid üks selline klass (ee.springdemo.AdvancedNewbie). Loogikaoperaatoreid kasutades saime lõikepunktiks tingimuse, kus oa ühendpunktina viidatakse noob oale ning ei (*not*)

operaator keelab AdvancedNewbie klassi puhul käivituse. Tekib loogika viga, aspekti ei käivitu kuna viitame ühele ja samale klassile kasutades nende vahel „and not“ loogikat. Hetkel sobib selline viga kuna soovime peatada logimist. Lisaks saame katsetada „ja“ (*and*) ning „ei“ (*not*) operaatorit.

```
<aop:aspect ref="logger">
      <aop:before pointcut="bean(noob)
                            and not within(*.*.AdvancedNewbie)"
                  method="logIt()"/>
</aop:aspect>
```

Koodinäide 24. Loogikaoperaatorid ja within võtmesõna

- Loome Logger klassi uue meetodi logItWithObject(). Muutujana võtab vastu *Object* tüüpi muutuja. Meetod trükib välja getDateTime() tulemi ning lisaks vastuvõetud muutuja toString() meetodi. Kui klassil ei ole üledefineeritud toString() meetodit siis trükitakse klassitäisnimi @- märk ja kuueteistkümnendarv, mis saadakse objekti räsi teisendades (Object (Java Platform SE 6), 2011).

```
public void logItWithObject(Object object){
      System.out.println(getDateTime() + " " + object.toString());
}
```

Koodinäide 25. logItWithObject(Object object) meetod

- Lõikepunktiks valime meetodi setString(String) käivituse (execution) ning käivitatavaks meetodiks logItWithObject(Object). Lõikepunktis tuleb loogikaoperaatoriga märkida ka objekti kaasaandmine. Lisaks tuleb märkida atribuut „arg-names“, mis on aspekti meetodi muutujanimi.

```
<aop:aspect ref="logger">
      <aop:before pointcut="execution(* *.setName(String))
                            and this(object)"
                  method="logItWithObject" arg-names="object"/>
</aop:aspect>
```

Koodinäide 26. Logger aspekt, käivitatakse logItWithObject

- Käivitame projekti. logIt() meetodit enam ei käivitata. Nüüd on näha aga logItWithObject(Object object). Kuvatakse getDateTime() ja objekt.toString() meetod.

```
2011.10.12 19:25:03 ee.springdemo.AdvancedNewbie@6f1f23e5
I can already fight with WoodenAxe
Wooden Axe can kill bugs
I'm advanced newbie and I want to fight little bit more...
```

Koodinäide 27. Rakenduse tulem

- Loome Logger klassi meetodi logItAndDetectObject(Object object, String retVal). Vastu võtab käivitava objekti ning käivitatud meetodi tagastuse. Objekti võrreldakse Player ja Weapon liidestega. Kui objekti klass pärineb vastavast objektist logitakse vastavale klasssile omaselt. Kui klassi ei tuvastata logitakse object.toString() meetodiga.

Kontrollimiseks kasutame tingimuslauset (*if*) ning klassist pärinemise (*instanceof)* meetodit.

```
public void logItAndDetectObject(Object object, String retVal){
      if(object instanceof Player){
            System.out.println(getDateTime()
                              + " Instanceof Player, Object: "
                              + object.toString() + " Name: "
                              + retVal);
            return;
      }
      if(object instanceof Weapon){
            System.out.println(getDateTime()
                             + " Instanceof Weapon, Object: "
                             + object.toString() + " Name: "
                             + retVal);
            return;
      }
      System.out.println(getDateTime()
                        + "Unknown instance, Object: "
                        + object.toString());
}
```

Koodinäide 28. Meetod logItAndDetectObject(Object object)

- Loome uue aspekti, mille juhendiks valime pärast-naasmist (after-returning) meetodi. Lõikepunktiks valime meetodi getName(). Aspektil käivitame logItAndDetectObject(Object object), seega lõikepunktis anname target meetodiga kaasa ka objekti.

```
<aop:aspect ref="logger">
      <aop:after-returning pointcut="execution(* *.getName())
                                    and target(object)"
                           returning="retVal"
                           method="logItAndDetectObject"
                           arg-names="object,retVal"/>
</aop:aspect>
```

Koodinäide 29. Aspekt logger, lõikepunkt getName(), meetod logitAndDetectObject

- NewbieMain klassi kirjutame viimase käivitusena väljatrüki noob.getName() meetodist.

```
System.out.println(noob.getName());
```

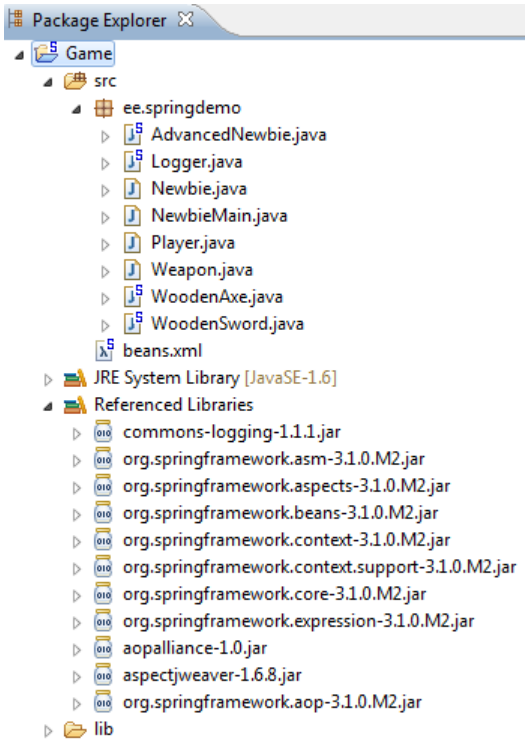
Koodinäide 30. Noobi nime väljatrükk

- Käivitame projekti CTRL + F11. Esimene logimine on logItWithObject tulem. Edasi järgneb kaks getName() meetodi käivitust, mida logitakse logItAndDetectObject abil. NewbieMain klassis käivitatakse Player „noob“ isendil beginGame() meetod, milles küsitakse relva nime getName() meetodi abil. Kolmas logimine tehakse eelmises punktis lisatud väljatrüki peale.

```
2011.10.12 21:23:14 ee.springdemo.AdvancedNewbie@5c6ed322
2011.10.12 21:23:14 Instanceof Weapon, Object:
ee.springdemo.WoodenAxe@6fe78c96 Name: WoodenAxe
I can already fight with WoodenAxe
Wooden Axe can kill bugs
I'm advanced newbie and I want to fight little bit more...
2011.10.12 21:23:14 Instanceof Player, Object:
ee.springdemo.AdvancedNewbie@5c6ed322 Name: New Noob
New Noob
```

Koodinäide 31. Rakenduse tulem

Sellega lõppevad aspekt-orienteeritud programmeerimise näited. Murede korral on võimalik võrrelda projekti failipuud ning lisaks on selle punkti lõpuks valminud projekt saadaval aadressil: http://www.tlu.ee/~maehans/Seminar/Game_DI_AOP.rar



Ekraanipilt 33. Projekti failipuu

5.5 Ülesanded

- Loo uus aspekt logimiseks.

- Juhise lõikepunktiks määra beginGame() meetod, klassiks AdvancedNewbie ja käivita meetod logItAndDetectObject().
- Katseta aspekti.

6 Lisaülesanne - töövoog

Töövoog koosneb üksteisest sõltuvatest tööetappidest. Automatiseeritud tööetapid võivad ennast ise lõpetada, inimkontrolli vajavad etappe aga peame kinnitama. Ülesandes kirjeldame materjali mõõtu lõikamise protsessi. Näiteks lõikame vineeritahvlit soovitud mõõtudesse. Esimese asjana määrame materjali, seejärel soovitud mõõdud, kontrollime materjali sobivust ja lõikame. Korraliku teenuse lõppu kuulub ka pakkimine.

Kujutame protsessikäiku ette lihtsal meetodil. Võtame materjali, märgime kirjutusvahendiga mõõdud peale ning lõikame kreis- või ketassaega materjali mõõtu. See on põhitegevus. Hiljem vaatame, kui kasutada automatiseeritud süsteeme, mis kontrollivad sisendmõõte ning materjali sobivust ja lõikavad automaatselt. Kirjeldatud ülesande lõplik lahendus on kättesaadaval aadressilt: http://www.tlu.ee/~maehans/Seminar/Spring_WorkFlow.rar. Juhin tähelepäanu, et kaabelduste paremaks mõistmiseks on lahenduses kaks kaabeldusfaili beans.xml ja beans2.xml.

6.1 Vajalikud klassid ja liidesed

- Klass materjal (*Material*). Klassil on 3 muutujat pikkus, laius ning kõrgus. Lisaks mõõtude määramise meetod ning iga dimensiooni küsimise meetodid.
- Liides märk (*Mark)*, põhimeetodiks on märkimine pikkuse, laiuse ning materjali alusel.
- Mark liidest realiseeriv klass *MarkByHand.* Kuvatakse materjali suurus ning küsitakse sobivust. Kasutaja peab kinnitama „Y“ või „N“ vajutades.
- Liides lõige (*Cut*). Põhimeetodiks on materjali lõikamine. Käivitades antakse kaasa Material tüüpi objekt.
- Cut liidest realiseeriv klass CutWithHexSaw ehk ketassaega lõikamine. Kasutajale kuvatakse juhtnöörid ning küsitakse kas materjal on väljalõigatud. Jällegi kasutaja vastab „Y“ või „N“ vajutades.
- Pakkimise liides (*Packing).* Põhimeetodiks on pakkimine.
- Klass käsitsi pakkimine (*PackingManually*). Küsib kasutajalt, kas on pakitud, vastatakse „Y“ või „N“ valides.

6.2 Kaabeldamine ja käivitus

Kaabeldame kõik loodud objektid. Pakkimise määrame aspektina, mis käivitatakse pärast lõikamist. Loome main meetodit sisaldava klassi. Springi abil küsime materjali klassi, määrame materjali mõõdud. Küsime märkimise klassi, märgime ja järgmisena lõikame. Pakkimise

määrasime aspektina, mis käivitub pärast lõikamist, seda main meetodis käivitama ei pea. Sellega oleme valmis projekti käivitama.

6.3 Muutmine

Töövoos võetakse kasutusele CNC (*computer numerical control*) pink ehk seade, mis oskab mõõtude järgi ise väljalõigata soovitud suuruse. Teeme järgmised muudatused:

- Loome CNC klassi, mis realiseerib Mark ja Cut liidest. Mark liidese põhimeetodi käivitusel, kontrollib materjali suurust. Sobivuse korral teavitab, vastandjuhul programmi töö seiskub. Cut meetod teatab lõikamisest ning viimase lõpetamisest.
- Muudame kaabeldamist. Cut ja mark oa klassiks saab CNC klass.
- Käivitame ja vaatame tulemust.

7 Spring raamistikule toetuvad lahendused

Ärirakenduste loomisel on Spring raamistik muutunud „*de facto*“ standardiks (Walss, 2011).
Sellest johtuvalt võiks näiteid leida palju, kahjuks aga ei leia rakenduse või veebilehe kirjelduses, et nende lahendus kasutab Spring raamistikku. Näiteid leiab veebilehelt „springsource.com“, kus on juhtumite uurimused. Toon välja kaks näidet. Esimene LinkedIn rohke kasutajate arvuga sotsaalvõrgustik, teine telekommunikatsiooni teenuseid pakkuv Telecom Italia.

7.1 LinkedIn

LinkedIn on mailma suurim professionaalidele orienteeritud sotsiaalvõrgustik, mis käivitus 2003 aasta mais. Tänaseks on üle 120 miljoni kasutaja. (About Us)
2005 aastal võeti kasutusele Spring eesmärgiga vähendada sõltuvusi. Sooviti vabaneda staatilistest sõltuvustest, parandada konfiguratsiooni, elutsükleid ja raskesti muudetavat kaabeldust. Kasutatakse sõltuvuste vähendamist, millele lisati LinkedIn laiendused. Viimane sisaldab „lin“-nimeruumi, kohandatud atribuutide muutjaid (custom property editors, timespan, memorysize) ja *ApplicationContext*. LinkedIn leiab, et raamistikuga kaasnevad plussid ja minused kuid üldjoones muutus rakendus paremini hallatavaks. (Pujante, 2008)

7.2 Telecom Italia

Telekom Italia on üks suurim Itaalia telekommunikatsioonifirma, pakkudes tava ja mobiiltelefoni, interneti, meedia ja uudiste teenuseid.
Telekom Italias leiti, et kasutatav kood oli väikese efektiivsusega ning nõudis rohkeid Java servereid, mille käivitamine võttis palju aega ning testimiseks tuli seda alati korrata. Testide rakendamine ning bugide leidmine oli keeruline. Rakendust oli raske hallata
Spring raamistikuga säästavad 30 protsenti silumis- (*debug)* ning testimisaega. Raamistiku abil muutus arendus lihtsamaks ning saavutati suurem tootlikkus. Lisaks Spring tagab kvaliteetsema koodi, pakkudes katsetatud disainimustreid. (Case Study at Telecom Italia).

8 Juhendi tutvustamine ja täiendamine

Nagu eelnevalt mainitud käsitletakse Spring raamistikku „Rakenduste programmeerimine“ aine raames. Spring teemat käsitledes kasutasime käesolevat juhendit, mis andis hea võimaluse katsetamaks juhendi arusaadavust ning jälgitavust. Juhendit läbi töötades ilmnesid mõningad vead, mis käesolevas versioonis on parandatud. Toon välja läbivad probleemid ning loodud lahendused.

8.1 Liideste loomine

Esimeseks segadusallikaks oli liideste loomine. Esialgses juhendis lõime liidesed klassidena ning seejärel muutsime klassi (*class*) tüübi liideseks (*interface*). Viimane oli käsitsi ümber defineerimine. Eclipse võimaldab liidese loomist ning parandatud versioonis luuakse liidesed kohe „*Interface*“ tüüpi objektidena. Selle tulemusena tuli luua uued ekraanipildid liidese loomiseks ning esimese klassi juurde seletus.

8.2 Lähtekoodi vaatele lülitamine

Punktis 4.5 peale Spring ubade konfiguratsioonifaili genereerimist lülitame ümber lähtekoodi vaatele. Lülitamiseks mõeldud koht jäi osa juhendikasutajatele arusaamatuks. Kuna tegemist ei ole nupuga vaid pigem vahelehega siis oli seda raske märgata. Täiustasin ekraanipilti lisades „*Source*“ saki ümber pilkupüüdva raami.

8.3 Trüki ja kujundusvead

Töös esines mõningaid trüki ning kujundusvigu. Viimase kõige olulisem näide on Java koodi murdumine. Selle tagajärjel ei oleks kood olnud enam kopeeritav ning kohe käivitatav. Parandusena lõin reavahetused sobivatesse kohtadesse.

Koodinäidete kirjastiilina kasutasin Eclipse keskkonnas kasutatavat „*Consolas*“ stiili. Viimane tekitas segadust eriti XML failide juures, kus kippus segamini minema suur ja väike L täht. Asendasin stiili „*CourierNew*“ga.

8.4 Beans.xml ja bean.xml

Punktis 4.5 määrame konfiguratsioonifaili nimeks beans.xmli. Punktis 5.2 kasutasin failinimena bean.xml’i. Segadust lisas seletus võimalusest genereerida AOP nimeruum ja skeemiversioon uue ubadefaili loomisel. Viimase kombinatsioonina hakati looma uut ubade konfiguratsioonifaili.

Näites oli mõeldud jätkamine esialgse failiga. Bean.xml nime asendasin beans.xml’ga ning lisaks sai parandatud sõnastust.

Kokkuvõte

Käesolevas juhendis tutvustasin Spring raamistikku. Lähemalt rääkisin sõltuvuste sisestamisest ning aspekt-orienteeritud programmeerimisest Spring raamistikuga. Praktilisemalt poolelt õpetasin, kuidas paigaldada Eclipse Indigot ning lisada Springi arendamiseks mõeldud tarkvaramooduleid. Kõige olulisema osana vaatasime sõltuvuste vähendamist. Viimase katsetamiseks sai loodud mitu erinevat näidet. Lisaks vaatasime aspekt-orienteeritud programmeerimise rakendamist olemasolevas Spring projektis.

Juhendi tutvustamise ja paranduse tulemusena olen veendunud, et käesoleva materjali läbitöötanu on võimeline looma Spring rakendusi kasutades DI ja AOP põhimõtteid.

Kasutatud kirjandus

1. *Object (Java Platform SE 6)*. (10. 8 2011. a.). Kasutamise kuupäev: 12. 10 2011. a., allikas Oracle Software Downloads: http://download.oracle.com/javase/6/docs/api/java/lang/Object.html#toString()
2. *6. AOP*. (kuupäev puudub). Kasutamise kuupäev: 13. 10 2011. a., allikas springindepth.com: http://springindepth.com/book/aop.html
3. *9. Spring AOP APIs*. (kuupäev puudub). Kasutamise kuupäev: 13. 10 2011. a., allikas SpringSource.org | : http://static.springsource.org/spring/docs/3.1.0.M2/spring-framework-reference/html/aop-api.html#aop-api-advice-types
4. *About Us*. (kuupäev puudub). Kasutamise kuupäev: 25. 10 2011. a., allikas LinkedIn Press Center: http://press.linkedin.com/about
5. *Case Study at Telecom Italia.* (kuupäev puudub). Kasutamise kuupäev: 30. 10 2011. a., allikas SpringSource Case Studies: http://www.springsource.com/files/uploads/all/pdf_files/customer/S2_CaseStudy_TelecomItalia_USLET_EN.pdf
6. Laddadd, R. (2003). *AspectJ In Action: Practical Aspect-Oriented Programming.* Greenwich: Manning Publications Co.
7. Mak, G., Long, J., & Rubio, D. (2010). *Spring Recepies: A Problem-Solution Approach (2. trükk).* New York: Apress.
8. *Part I. Overview of Spring Framework*. (kuupäev puudub). Allikas: SpringSource.org |: http://static.springsource.org/spring/docs/3.1.0.RC1/spring-framework-reference/html/spring-introduction.html
9. Pujante, Y. (12 2008. a.). *Case Study Spring at LinkedIn.* Kasutamise kuupäev: 15. 10 2011. a., allikas SpringSource Case Studies | SpringSource: http://www.springsource.com/files/SpringAtLinkedIn.pdf
10. Walss, C. (2011). *Spring in Action (3. trükk).* New York: Manning Publications CO.