Digitehnoloogiate instituut

Tarkvara ehitust automatiseeriva tööriista Gradle kohta õppematerjali loomine Java veebirakenduse põhjal

Seminaritöö

Autor: Sven- Kristjan Kompus

Juhendaja: Jaagup Kippar

Autor: .....“.....“.....2016

Juhendaja: .....“.....“.....2016

Instituudi direktor: .....“.....“.....2016

Tallinn 2016

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Sisukord

Sissejuhatus

Tarkvaraarenduses on tänapäeval küllaltki oluline erinevate protsesside automatiseerimine - olgu selleks rakenduse testimine, ehitamine, või hoopiski midagi muud. Käesoleva seminaritöö teemaks ongi õppematerjali loomine Gradle'i – tööriista, mis loob võimaluse automatiseerida rakenduse kompileerimist, testimist ning ehitamist, kasutamise kohta.

Teema valiku peamiseks põhjuseks on autori enda mõningane kokkupuude ning huvi selle teema vastu. Samuti ei ole saadaval eestikeelset õppematerjali Gradle'i tööriista kohta, mis hõlmab endas Java veebirakenduste ehitamist.

Siinse seminaritöö eesmärgiks on luua eestikeelne õppematerjal Gradle'i tööriista kasutamise kohta Java veebirakenduse põhjal, mis esiteks tekitab lugejal huvi Gradle'i tööriista kohta. Teiseks annab käesolev õppematerjal lugejale võimaluse iseseisvalt õppida kasutama Gradle'it.

Esimeses peatükis antakse lugejale ülevaaade Gradle'i tööriistast. Juttu tuleb lühidalt nii ajaloost kui ka tööriista tunnusjoontest.

Teises peatükis tutvustatakse lugejale Gradle'i tööriista ehitusskripti ning erinevaid ehitusskriptis kasutatavaid komponente.

Kolmandas peatükis tutvustatakse lugejale paari peamist alternatiivi Gradle'i tööriistale.

Neljandas peatükis antakse lugejale ülevaade olemasolevatest õppematerjalidest.

Viiendas peatükis tuleb juttu käesolevast õppematerjalist. Räägitakse nii selle eesmärgist, vajalikkusest, sihtgrupist ning samuti õppematerjali testimisest ning selle tulemustest.

Kuuendaks peatükiks ongi õppematerjal, mis annab lugejale ülevaate, algteadmised ning oskuse iseseisvalt kasutada Gradle'i tööriista.

Mõisted

DSL (*Domain-Specific Language*) – Valdkonnapõhine programmeerimiskeel.

Task – Ülesanne, ehk mingi töö, mida Gradle täidab ehituse käigus.

Plugin – komponent, mis Gradle'i puhul annab projektile lisafunktsionaalsust.

Ehitusskript – build.gradle fail.

Dependency – teiste projektide poolt loodud failid, mida vastav projekt töötamiseks, ehitamiseks ja muuks vajab.

Remote repository – kaugarvutis paiknev repositoorium.

1 Ülevaade Gradle'ist

Gradle on avatud lähtekoodiga, Hans Dockter'i (Gradle Inc., varasemalt Gradleware Inc., looja) poolt loodud tarkvara ehitamist ning teisi protsesse automatiseeriv tööriist, mis jõudis avalikkuse ette 2008. aasta aprillis. Tänaseks päevaks on kasutajatele saadaval juba versioon 2.9. (*Gradle Inc.*)

Gradle'it on võimalik kasutada nii Windows'i, Linux'i ning samuti OS X'i operatsioonisüsteemidel.

1.1 Tunnusjooned

Gradle'il on mitmeid tunnusjooni, mis muudavad tarkvara ehitamise mõnevõrra lihtsamaks ning samuti pakub Gradle'i tööriist palju erinevaid funktsionaalsusi.

Üheks peamiseks tunnusjooneks on see, et erinevalt paljudest teistest sarnastest tööriistadest, kasutab Gradle ehitusskripti loomisel valdkonnapõhist programmeerimiskeelt (DSL), mis põhineb Groovy'l ning ehitusskript võib sisalda mis-tahes Groovy programmeerimiskeele elemente. Groovy on objekt-orienteeritud dünaamiline programmeerimiskeel, mis on süntaksilt väga sarnane Java programmeerimiskeelele. Kasutust leiabki see skriptimiskeelena Java platvomil.

Teiseks tunnusjooneks on see, et Gradle teeb kasutajale lihtsaks luua hästi struktureeritud ehitusskript, varustades kasutajat deklaratiivse programmeerimiskeele elementidega, mida kasutaja saab kokku panna vastavalt enda soovidele ning vajadusele.

Lisaks on Gradle väga hästi kasutatav olenemata projekti ehituse ulatusest – olgu tegu siis väikese projektiga, või väga ulatusliku projektiga, mille alla kuulub mitmeid alamprojekte.

Gradle teeb kasutajale lihtsaks ka sõltuvuste haldamise, olenemata sellest, mis viisil kasutaja sõltuvusi hallata soovib. Sõltuvuste haldamiseks on kasutajal võimalik kasutada nii Maven'i ning Ivy'i repositooriume, kuid samuti ka samas failisüsteemis asuvaid katalooge.

Samuti teeb Gradle võimalikuks ning väga lihtsaks olemas olevate Ant'i projektide importimise.

Lisaks on võimalik Gradle'i ehitusi käivitada arvutitest, kus Gradle ei ole installeeritud. Seda võimaldab Gradle Wrapper, ehk siis *shell*'i või *batch*'i skript. See teeb projekti ehitamise väga lihtsaks, sest Wrapper on projektis sees ning seda on võimalik jooksutada olenemata sellest, kas vastavas arvutis on Gradle installeeritud, või mitte.

(*Gradle Inc.*)

2 Gradle'i ehitusskript ja selles kasutatavad komponendid

Gradle'i töös on väga tähtsal kohal ehitusskript, mille täpsemaks definitsiooniks on ehituse konfiguratsiooni skript. Gradle'i ehitusskriptiks on build.gradle fail ning kasutaja saab seda peaaegu, et täielikkult üles ehitada vastavalt oma soovidele ning vajadustele. Ehitusskripti kirjutamisel kasutatakse valdkonnapõhist programmeerimiskeelt, mis põhineb Groovy keelel. Lisaks sellele on veel teatud täiendusi, mis teevad ehituse kirjeldamise kasutajale lihtsamaks. Ehitus toetub peamiselt kahele põhikontseptile: projekt ning *task*. Iga ehitus sisaldab vähemalt ühte projekti ning iga projekti alla kuulub vähemalt üks *task*. (*Gradle Inc.*)

2.1 Projektid ning *task*'id

See, mida vastav Gradle'i projekt endast kujutab, oleneb sellest, mida kasutaja Gradle'i abiga teeb. Projekt ei kujuta endast tingimata seda, mida ehitatakse. Ehk siis lisaks sellele, et projektiks võib olla veebirakendus, võib projektiks olla ka näiteks veebirakenduse juurutamine test-keskkonda. Iga projekt on üles ehitatud vähemalt ühest *task*'ist. *Task* kujutab endast mingit ehituse poolt tehtavat tööd – selleks võib olla nii klasside kompileerimine, kui ka vaja minevate failide kopeerimine ühest kaustast teise. (*Gradle Inc.*)

2.2 Pluginad

Pluginad on Gradle'i töös vägagi tähtsal kohal. Gradle ise, kasutamata pluginaid, pakub sihilikult küllaltki vähesel määral automatiseerimist. Ehk siis näiteks koodi kompileerimine lisandub tänu pluginatele. Pluginate lisamine projektile tõstab selle võrra ka projekti võimekust, ehk pluginad võivad teha näiteks järgmisi asju:

- Laiendada Gradle'i mudelit, mistõttu on võimalik näiteks konfigureerida uusi valdkonnapõhise programmeerimiskeele elemente.
- Konfigureerida projekte, ehk lisada näiteks uusi *task*'e.
- Kohaldada teatud konfiguratsioone.

Gradle'is on kahte tüüpi pluginaid: skripti pluginad ning binaar pluginad. Skripti pluginateks on täiendavad ehitusskriptid. Binaar pluginateks loetakse klasse, mis implementeerivad Plugin liidest. (*Gradle Inc.*)

2.3 Sõltuvuste haldamine

Sõltuvusteks ehk *dependency*'teks loetakse teiste projektide poolt loodud faile, mida vastav projekt kasutab. Et projekt korrektselt kompileeruks, ehituks ning töötaks, on vaja sõltuvused ka Gradle'ile leitavaks teha. Gradle teeb kasutajale küllaltki lihtsaks projekti töötamiseks vajalike sõltuvuste kirjeldamise. Sõltuvuste allalaadimiseks on võimalik kasutada nii Maveni ning Ivy repositooriume, kui ka samas failisüsteemis asuvaid katalooge. Samuti on võimalik sõltuvustena kasutada näiteks samas multiprojektis oleva projekti loodavat JAR faili. (*Gradle Inc.*)

2.4 Ehituse tööprotsessi logimine

Ehituse tööprotsessi logimine on küllaltki tähtsal kohal, eriti kui tööprotsessis esineb vigu, või midagi on valesti läinud. Põhjalik ning hea logimine võib osutada vägagi kasulikuks. Gradle'is on ära defineerinud 6 erinevat logitaset: (*Gradle Inc.*)

- ERROR (veateated)
- QUIET (tähtsad infoteated)
- WARNING (hoiatusteated)
- LIFECYCLE (progressi infoteated)
- INFO (infoteated)
- DEBUG (*debug* teated)

2.5 Testimine

Kasutajal on võimalik ka oma projekte testida kasutades selleks JUnit või TestNG teste. Gradle tuvastatab ja käivitab testid automaatselt ning samuti genereeerib testiraporti, kui testid on töö lõpetanud. (*Gradle Inc.*)

3 Alternatiive Gradle'ile

Lisaks Gradle'ile on veel mitmeid teisi, sageli tuntuimaid ja suuremat kasutajaskonda omavaid, rakendusi, mis pakuvad rakenduste ehituse automatiseerimist. Peamisteks alternatiivideks Gradle'ile on momendil kaks tööriista: Maven ning Ant. Ant'i puhul pakub ka Gradle väga mugavat võimalust Ant'i *target*'eid Gradle'i ehitusskriptis implementeerida ning jooksutada neid Gradle'i *task*'idena, mistõttu on kasutajale väga lihtsaks tehtud olemasoleva Ant'i projekti kasutada Gradle'is.

3.1 Apache Maven

Apache Maven on Apache Software Foundation'i poolt välja arendatud tarkvara ehitust automatiseeriv tööriist, mis on kasutuseks mõeldud Java-põhistele projektidele. Erinevalt Gradle'ist, mis kasutab ehitusskriptis Groovy programmeerimiskeelel põhinevat valdkonnapõhist programmeerimiskeelt, asuvad Maven'i projektid pom.xml failis ning on kirjutatud XML märgistuskeeles. Projektide ehitamiseks kasutab Maven projekti objektmudelit (POM). (*Apache Software Foundation*)

3.2 Apache Ant

Apache Ant on Apache Software Foundation'i poolt välja arendatud käsureal töötav tööriist, mis leiab peamist kasutust Java rakenduste ehitamisel. Ant võimaldab jooksutada protsesse, mis on XML failis ära defineeritud Ant *target*'itena. Samuti varustab Ant kasutajat mitmete sisseehitatud *task*'idega, mis võimaldavad Java rakendusi kompileerida, testida ja jooksutada. Lisaks Java rakendustele võimaldab Ant ehitada ka näiteks C või C++ programmeerimiskeeles kirjutatud rakendusi. (*Apache Software Foundation*)

4 Olemasolevad õppematerjalid

Materjale ning juhendeid, mis tutvustavad Gradle' tööriista ja selle kasutamist täielikkult ning läbinisti, ei ole palju. Peamiseks Gradle'i tööriista õppematerjaliks on selle dokumentatsioonis asuv õppematerjal, kus põhiliselt vaja minev on kasutajale küllaltki hästi ära seletatud. Lisaks sellele on siinses peatükis välja toodud veel mõningad hästi loodud õppematerjale.

4.1 Gradle'i dokumentatsioon

Veebiaadress:
https://docs.gradle.org/current/userguide/userguide.html

Gradle'i dokumentatsioonis on väga hästi ära dokumenteeritud Gradle'i tööriista viimane reliis – välja on toodud kõik uuendused ning muudatused võrreldes eelmise reliisiga. Lisaks on dokumentatsioonis olemas ka Gradle'i tööriista õppematerjal, mis on tänu heale vormistusele ning täielikkusele vägagi kasulik, eriti algajale kasutajale. Õppematerjal muutub peatükkide kaupa keerulisemaks, mis tähendab et algajal kasutajal on hea alustada algusest ehk kõige lihtsamast, kuid näiteks kasutaja, kellel on juba algteadmised olemas, võib liikuda keerulistemate peatükkide juurde.

Käesoleva seminaritöö autor kasutab Gradle'i dokumentatsioonis asuvat õppematerjali ka alusena eestikeelse õppematerjali loomisel.

4.2 Õppematerjal IRomin'i veebilehel

Veebiaadress:
http://rominirani.com/2014/07/28/gradle-tutorial-series-an-overview/

IRomin'i veebilehel leiab mitme-peatükilise ning küllaltki laiapõhjalise juhendi, mis katab ära mitmed Gradle'i kasutusomadused- ja võimalused. Räägitakse nii Gradle'i esialgsest

installatsioonist, esimese Java projekti ehitamisest kasutades Gradle'i tööriista, Gradle'i tööriista kasutamisest Android Studios ja veel muust. Õppematerjal on ülesehitatud peatükkidena, alustades Gradle tutvustamise ja installatsiooniga ning jätkates Gradle'i kasutamisega erinevat tüüpi projektide puhul.

4.3 Õppematerjal Petri Kainulainen'i veebilehel

Petri Kainulainen veebilehel on Gradle'i kohta õppematerjal, mis taaskord koosneb peatükkidest, millest igaüks hõlmab mingit kindlat tööriista kasutuse võimalust. Olenevalt peatükist, on need mingil määral küll erineva raskustasemega, kuid personaalse arvamusena sobivad need just algajatele ning kesktaseme kasutajatele. Õppematerjal on hästi ülesehitatud ning autor on näited hästi ära seletanud, seetõttu on seda õppematerjali väga mugav kasutada.

Veebiaadress:

http://www.petrikainulainen.net/getting-started-with-gradle/

4.4 Õppematerjal Vogella veebilehel

Vogella veebilehel asub üheleheline õppematerjal, mis on küllaltki hästi jagatud peatükkideks ning seetõttu on seda õppematerjali väga mugav lugeda. Kokkuvõttes on see mahult võibolla küll lühem, kui eelnevalt välja toodud õppematerjalid, kuid ülesehituse lihtsuse, kasutusmugavuse ning küllaltki hästi tehtud näidete tõttu sobib see algajale kasutajale ideaalselt.

Veebiaadress:

http://www.vogella.com/tutorials/Gradle/article.html

5 Õppematerjali analüüs

Õppematerjal koosneb peamiselt mitmest väiksemast osast ning ühest suuremast osast. Väiksemad osad on sissejuhatavad, kus lugejale tutvustatakse Gradle'i tööriista kasutuseks vajalikke komponente ning samuti testitakse need komponendid läbi. Suuremas osas luuakse mitmikprojekt, mis sisaldab juurprojektina Java veebirakendust ning alamprojektina väiksemat sorti tavalist Java rakendust.

5.1 Eesmärk

Selle seminaritöö eesmärgiks on lugejale tutvustada Gradle'i tööriista ning võimaldada lugejal saada baasteadmised Gradle'i tööriista kasutamise kohta Java veebirakenduste ehitamisel. Õppematerjali läbides peaks kasutaja suutma iseseisvalt luua ehitusskripti, luua ning käivitada *task*'e, hallata rakenduse tööks vajalikke sõltuvusi, lisada projektile pluginaid ja veel mõndagi muud, mida õppematerjali käigus läbi tehakse.

5.2 Vajalikkus

Õppematerjal on vajalik just seetõttu, et hetkel puudub eestikeelne materjal, mis esiteks tutvustab Gradle'i tööriista ning selle kasutamist ning teiseks tutvustab lugejale Java veebirakenduste kompileerimist, testimist ning ehitamist kasutades Gradle'i ehitusskripti.

5.3 Sihtgrupp

Käesolev töö on suunatud kõigile õpihimulistele inimestele, keda huvitab tarkvara arendus ning erinevate protsesside automatiseerimine. Samuti võiks lugejal olemas olla algteadmised Java arenduses.

5.4 Testimine

Testimise käigus töötasid käesoleva õppematerjali läbi kaks testijat. Mõlema testimise korra puhul oli õppematerjali autor testija kõrval ning abistas, kui vaja.

Esimene testimine

Esimene testija on Tallinna Ülikooli, Digitehnoloogiate instituudi kolmanda kursuse õpilane. Testimine võttis aega ligikaudu poolteist tundi. Testimise käigus esines õppematerjalis üksikuid vigu ning probleeme, mille õppematerjali looja õppematerjalis ära parandas. Üheks põhiprobleemiks oli see, et osade koodinäidete puhul võis arusaamatuks jääda, millisesse faili tuleb koodiosa lisada. Testija andis ka omapoolseid soovitusi, mida võiks õppematerjali juures veel parandada ning mitut nõuannet õppematerjali autor ka kasutas.

Teine testimine

Esimene testija on Tallinna Ülikooli, Digitehnoloogiate instituudi teise kursuse õpilane. Testimine võttis aega veidikene vähem kui poolteist tundi, kuid testimine oli põhjalikum, kui esimese testimine. Seetõttu tulid testimise käigus välja ka üksikud vead näiteks sõnastuses. Üldiselt läks teine testimine ladusamalt, kui esimene, sest õppematerjali näidetes enam vigu ei esinenud. Testija andis testimise lõpus ka omapoolseid soovitusi, mida õppematerjali autor ka kasutas.

6 Õppematerjal

Õppematerjali loomise käigus on kasutatud Gradle'i versiooni 2.8 ning Java versiooni 7. Operatsioonisüsteemina on kasutatud Ubuntu 14.04 LTS.

Õppematerjalis olevaid koodinäiteid kasutades tuleks veenduda, et koodi struktuur on täpselt sama, mis näidetes.

6.1 Gradle'i installeerimine

Gradle'i installeerimine ei ole keeruline, aga võrreldes tavapärase installeerimisfaili käivitamisega, on Gradle'i installeerimine siiski veidikene keerukam. Esiteks tuleb kindlaks teha, kas arvutisse on installeeritud Java JDK või JRE (6. verioon või uuem). (*Gradle Inc.*) Selle kontrollimiseks on kõige lihtsam Command Prompt'is (Windows) või Terminal'is (Linux ning OS X) käivitada järgmine käsk:

```
java -version
```

Seejärel tuleb minna Gradle'i kodulehele ning allalaadida Gradle'i tööriista ZIP fail. Olenevalt kasutaja vajadustest on allalaadimiseks saadaval mitu erinevat varianti. Esiteks on variant, mis sisaldab ainult tööriista kasutuseks vajalikke faile. Teiseks on saadaval variant, mis sisaldab lisaks kasutuseks vajalikele failidele veel lähtekoodi ning *offline'*is kasutatavat dokumentatsiooni. Viimaseks on variant, mis sisaldab vaid Gradle'i tööriista lähtekoodi.

Kui ZIP fail on allalaaditud, siis tuleb see lahti pakkida ning paigutada kasutajale sobivasse kausta. Seejärel tuleb kasutajal lisada PATH keskkonna muutujale (*Environment variable*) Gradle'i tööriista kodukausta alamkaust **bin**. Protsess selleks olenevalt operatsioonisüsteemist on järgmine:

Windows

- Avada **Start menüü**
- Parem hiireklikk **My Computer** peal ning valikust valida **Properties**

- **Advanced system settings → Advanced → Environment Variables**
- **User variables** alla luua uus keskkonna muutuja vajutades nuppu **New**
  - **Variable name** – GRADLE_HOME
  - **Variable value** – Gradle'i tööriista kodukausta asukoht failisüsteemis
- Muuta **PATH** keskkonna muutujat, vajutades sellel ning vajutades nuppu **Edit**
  - **Variable value** lõppu lisada %GRADLE_HOME%\bin;

Linux ja OS X

OS X'i ning Linuxi puhul on installeerimiseks vaja lisada samad keskkonna muutujad. Selleks tuleb avada kõige pealt ***profile*** fail järgmise käsuga:

```
nano ~/.profile
```

Pärast seda tuleb faili lõppu lisada järgmised read:

```
export GRADLE_HOME=GRADLE_KODUKAUST
export PATH=$PATH:$GRADLE_HOME/bin

//GRADLE_KODUKAUST tuleb asendada Gradle'i tööriista kodukaustaga
```

6.1.1 Pärast installatsiooni

Et installatsiooni käigus tehtud muudatused salvestuks, tuleks kasutajast välja ning siis uuesti sisse logida.

Kui installatsioon on tehtud, siis avame taaskord Command Prompt'i või Terminali.

Seejärel käivitame järgmise käsu `gradle -v`.

Väljundiks kuvab Gradle tööriista versiooni ning muu vajaliku info. Kui väljundiks on veateade, siis tuleks installeerimise käik uuesti üle vaadata.

Pärast installeerimist tuleb kasutajale sobivasse kausta luua **build.gradle** fail. Seejärel tuleb avada olenevalt operatsioonisüsteemist Command Prompt või Terminal ning liikuda kausta, kuhu see fail loodi.

6.2 *Task*'ide põhitõed

Selles peatükis tuleb juttu Gradle'i ehituse põhikomponentide – *task*'ide põhitõdedest. *Task* kujutab endast projektis tehtavad tööd, mida ehituse käigus täidetakse. Olemas on nii erinevat sorti *task*'e ning samuti erinevaid viise *task*'ide deklareerimiseks.

Task'ide struktuur on küllaltki lihtne ning arusaadav. Kõige lihtsamat sorti *task* on võimalik **build.gradle** failis (ehitusskriptis) defineerida järgmiselt:

```
task naide
```

Sellel *task*'il ei ole hetkel küll mingisugust sisu, kuid seda on võimalik ilma probleemideta käsurealt käivitada järgmise käsuga:

```
gradle naide

//gradle + task'i nimetus

//lisades käsule -q või --quiet flag, varjatakse Gradle'i logi sõnumid
```

Järgmiseks võib luua task'i, millel on juba väikene tööülesanne:

```
task naide2 << {
     println 'Tere!'
}
```

Käivitades käsu `gradle naide2` on väljundiks järgmine:

```
:naide2
Tere!

BUILD SUCCESSFUL
```

Eelnevast näitest on näha, et pärast *task*'i nimetust on lisatud '<<' sümbolid. Need sümbolid tähistavad lühendit ***doLast*** meetodile. Gradle'i *task*'e on võimalik deklareerida kasutades erinevaid variatsioone, mida on vaja kasutada olenevalt situatsioonist. Järgmises näites kasutamegi lisaks ***doLast*** meetodile veel ***doFirst*** meetodit.

```
task naide3{
     doLast{
          println 'Olen viimane'
     }
     doFirst{
          println 'Olen esimene'
     }
}
```

Käivitades käsu `gradle naide3` on väljundist näha, et *doLast* meetod käivitatakse hiljem, olenemata sellest, et see on defineeritud esimesena:

```
:naide3
Olen esimene
Olen viimane

BUILD SUCCESSFUL
```

Samuti on võimalik muuta juba olemasolevaid *task*'e. Järgmiseks loome ehitusskriptis *task*'i, ning lisame sellele muudatuse *task*'i väliselt.

```
task naide4{
     String sonum = 'viimane';
     doLast{
          println 'Olen ' + sonum
     }
}

naide4.doFirst{
     println 'Olen esimene'
}
```

Task'e on võimalik deklareerida ka teisel kujul. Näiteks on võimalik *task*'i nimi määrata ära järgmiselt:

```
task (naide5) << {
     println 'Tere!'
}
```

Task'i nimi on võimalik ära määrata ka *string*'ina:

```
task ('naide6') << {
     println 'Tere!'
}
```

6.2.1 *Task*'ide sõltuvused

Task'idele saab anda ka sõltuvusi teiste *task*'ide näol. Deklareerime järgmiseks ära ühe *task*'i, mis sõltub teisest *task*'ist.

```
task naide1(dependsOn:'naide2') << {
     println 'naide1'
}

task naide2 << {
     println 'naide2'
}
```

Käivitades käsu `gradle naide1` on väljundist näha, et mõlemad *task*'id käivituvad ilma probleemideta. Samuti on näha ka see, et esmaselt käivituvad sõltuvad *task*'id :

```
:naide2
naide2
:naide1
naide1

BUILD SUCCESSFUL
```

Task'il võib sõltuvusi olla ka mitu:

```
task naide3(dependsOn:['naide4','naide5']) << {
     println 'naide3'
}
task naide4 << {
     println 'naide4'
}
task naide5 << {
     println 'naide5'
}
```

6.2.2 *Task*'ide loomine dünaamiliselt

Gradle'is on võimalik luua *task*'e ka dünaamiliselt. Järgmine näide kujutabki *task*'ide dünaamilist loomist:

```
3.times { id ->
     task "naide$id" << {
          if(id == 2){
               println 'Hea valik!'
          }
          println "Uus task, mille ID on $id"
     }
}
```

Käesoleva näite puhul luuakse kolm *task*'i (**naide0**, **naide1** ja **naide2**) ning näitena käivitame ühe neist järgmiselt:

```
gradle naide2
```

Väljundiks on järgmine:

```
:naide2
Hea valik!
Uus task, mille ID on 2

BUILD SUCCESSFUL
```

Dünaamiliselt on *task*'e võimalik luua ka näiteks järgmisel moel:

```
List arvud = new ArrayList()
arvud.push(1)
arvud.push(2)

arvud.each{
     arv ->
     task "arv$arv" << {
          println "See on task $arv"
     }
}
```

Selle näite käigus loodud ühe *task*'i võime käivitada järgmiselt:

```
gradle arv1
```

6.2.3 *Task*'ide atribuudid

Task'idele on võimalik ka lisada atribuute, mida on teistel *task*'idel võimalik oma töös kasutada. Käesoleva näite puhul tuleb mainida, et *task'i*, kus atribuut defineeritakse, peab initsialiseerima esimesena. Seetõttu eemaldame '<<' sümbolid selle *task*'i nimetuse tagant:

```
task naide {
     ext.naidis = 'Naide'
}

task naidePrint << {
     println naide.naidis
}
```

Käivitades käsu `gradle naidePrint` kuvatakse väljundis *task*'is **naide** defineeritud atribuut:

```
:naidePrint
Naide

BUILD SUCCESSFUL
```

6.2.4 Vaikimisi *task*'id

Vaikimisi *task*'ideks nimetatakse *task*'e, mis käivituvad juhul, kui ühtegi *task*'i ei ole etteantud. Järgmises näites määratakse ära kaks vaikimisi *task*'i:

```
defaultTasks 'naide1', 'naide2'

task naide1 << {
     println 'naide1'
}

task naide2 << {
     println 'naide2'
}

task naide3 << {
     println 'naide3'
}
```

Käivitades käsu `gradle` on väljundist näha, et käivitati *task*'id naide1 ning naide2:

```
:naide1
naide1
:naide2
naide2

BUILD SUCCESSFUL
```

6.2.5 Ülesanded

- Loo kokku kolm *task*'i järgmisel viisil: esimene *task* sõltub teisest ning teine *task* sõltub kolmandast. Seejärel käivita käsureal üks loodud *task* ning veendu, et kõik loodud *task*'id sõltuvuste kaudu käivituvad.

- Loo kaks *task*'i. Ühes *task*'is loo kaks atribuuti. Selles samas *task*'is prindi loodud atribuudid ka ühel real välja. Seejärel loo teine *task*. Kasutades *doFirst* ning *doLast* meetodit prindi *doFirst* meetodis välja eelnevalt loodud *task*'i üks atribuut ning *doLast* meetodis teine atribuut. Seejärel määra ära, et teisena loodud *task* on sõltuv esimesena loodud *task*'ist. Lõpuks käivita näide.

6.3 Mitmikprojektide ehitus

Gradle'i ehitusskriptis on võimalik defineerida ka mitut projekti, ehk siis ühekordse Gradle'i ehitusskripti käivitamisega on võimalik ehitada mitut projekti. Selleks loome

esiteks **settings.gradle** faili, kus me defineerime ära kõik ehitusse kuuluvad projektid:

```
include 'naide1', 'naide2'
```

Tehtud näites määrasime me ära kaks alamprojekti: naide1 ja naide2.

Järgmises näites loome ehitusskriptis *task*'i, mis kehtib kõigi alamprojektide kohta ning prindib välja alamprojekti nime:

```
subprojects{
     task alamprojektideNaide << {
          println "Projekt nimega $project.name"
     }
}
```

Käivitades käsu `gradle alamprojektideNaide` saame väljundiks järgmise:

```
:naide1:alamprojektideNaide
Projekt nimega naide1
:naide2:alamprojektideNaide
Projekt nimega naide2

BUILD SUCCESSFUL
```

Iga alamprojekt on võimalik seadistada vastavalt vajadusele. Järgmises näites loome mõlema projekti alla ühe *task*'i, kuid veidi erineva väljundiga:

```
project(':naide1'){
     task naide << {
          println 'See on esimene alamprojekt'
     }
}

project(':naide2'){
     task naide << {
          println 'See on teine alamprojekt'
     }
}
```

Sellisel puhul on võimalik käivitada *task*'i **naide** mitmel erineval viisil. Esiteks mõlema alamprojekti puhul käsuga `gradle naide` ning samuti on võimalik käivitada vaid vastava projekti alla kuuluvat *task*'i, andes lisaks *task*'i nimetusele kaasa ka projekti nimetus:

```
gradle naide1:naide
```

Viimase käsu puhul on väljundiks järgmine:

```
:naide1:naide
See on esimene alamprojekt

BUILD SUCCESSFUL
```

6.3.1 Ülesanne

- Lisa settings.gradle faili alla kolm alamprojekti. Seejärel loo igale alamprojektile sama nimega *task*, millest igaüks prindib väljundisse erineva sõnumi.

6.4 Pluginad

Plugin on lihtsalt öeldes Gradle'i laiendus, mis mingil moel seadistab kasutaja projekti. Peamiseks seadistuseks ongi projektile mingi hulga eelseadistatud *task'*ide lisamine, mis täidavad mingit ülesannet.

Kuna selle õppematerjali käigus luuakse ehitusskript, mis loob Java veebirakenduse projektist WAR faili, siis näitena proovime läbi kaks pluginat: Java ning WAR plugina. WAR plugin on põhimõtteliselt Java plugina laiendus, mis lisab toe WAR faili kokkupanekuks. Java plugina lisamine projektile käib ehitusskriptis järgmiselt:

```
apply plugin: 'java'
```

Java ning WAR plugin otsivad projekti lähtekoodi **src** kaustast, seetõttu tuleb luua src kaustale järgmine struktuur (tavalise Java rakenduse puhul **webapp** kausta ei looda):

```
src
 | main
 |    | java
 |    | resources
 |    | webapp
 |        | WEB-INF
 |             | web.xml
 |
 | test
      | java
```

Käivitades käsu `gradle build` pärast Java plugina projektile lisamist, on väljundis näha *task*'id mis Java plugina tõttu käivitusid:

```
:compileJava
:processResources
:classes
:jar
:assemble
:compileTestJava
:processTestResources
:testClasses
:test
:check
:build
```

Järgmiseks asendame ehitusskriptis projektile lisatud Java plugina WAR pluginaga. WAR plugina lisamine käib järgmiselt:

```
apply plugin: 'war'
```

Käivitame taas käsu `gradle build` on väljundist näha, et seekord luuakse JAR faili asemel hoopiski WAR fail:

```
:compileJava
:processResources
:classes
:war
:assemble
:compileTestJava
:processTestResources
:testClasses
:test
:check
:build
```

Loodud WAR ning JAR failid leiab **build/libs** kataloogist.

6.4.1 Pluginate loomine

Pluginaid on võimalik ka ise luua. Gradle teeb ise loodud pluginate implementeerimise küllaltki lihtsaks ning seetõttu on võimalik pluginaid luues ning kasutades ehituseloogikat taaskasutada.

Pluginate lähtekoodi hoiustamiseks on mitmeid asukohti: (*Gradle Inc.*)

- Ehitusskriptis

- buildSrc alamprojektis
    - JUURPROJEKT/buildSrc/main/groovy kataloogis
- Eraldi projektis, millest luuakse JAR fail. Loodud JAR faili, mis sisaldab loodud pluginat, on võimalik hiljem taaskasutada.

Näitena loome ehitusskripti sisse lihtsakujulise plugina:

```
apply plugin: PluginaNaide

class PluginaNaide implements Plugin<Project> {
    void apply(Project project) {
        project.task('naide') << {
            println 'Plugina naide!'
        }
    }
}
```

Kui plugin on projekti lisatud, siis Gradle kutsub välja **Plugin.apply()** meetodi ning parameetrina antakse kaasa ka **Project** tüüpi objekt.

Käivitades näites loodud *task*'i käsuga `gradle naide` on väljundiks järgmine:

```
:naide
Plugina naide!

BUILD SUCCESSFUL
```

6.4.2 Ülesanne

- Loo plugin, mis lisab projektile kaks *task*'i, millest üks sõltub teisest ning mis mõlemad prindivad väljundisse sõnumi. Seejärel lisa loodud plugin projekti külge. Lõpuks käivita näide.

6.5 Sõltuvuste haldamine

Sõltuvusteks nimetatakse erinevaid raamistikke, rakendusliideseid ja teisi rakendusi, mida projekt vajab, et töötada. Sõltuvused ja nende versioonid määratakse ära ehitusskriptis. Lisaks sõltuvustele tuleb ära määrata ka sõltuvuste repositoorium – ehk asukoht, kus kohast Gradle sõltuvusi otsib.

6.5.1 Sõltuvuste repositooriumid

Repositooriumiks nimetatakse failide kollektsiooni, mis on organiseeritud nime, grupi ning versiooni järgi. Enim kasutatavateks repositooriumiteks on Maven'i ning Ivy'i repositooriumid.

Repositooriumist failid kätte saamine on esiteks võimalik üle HTTP. Näiteks Maven'i keskrepositooriumi defineerimine ehitusskriptis käib järgmiselt:

```
repositories{
     mavenCentral()
}
```

Kaugarvutis paikneva Maven'i repositooriumi defineerimine käib järgmiselt:

```
repositories{
     maven{
          url 'REPOSITOORIUMI_AADRESS'
     }
}
```

Kaugarvutis paikneva Ivy'i repositooriumi defineerimine käib järgmiselt:

```
repositories{
     ivy{
          url 'REPOSITOORIUMI_AADRESS'
     }
}
```

Samuti on võimalik repositooriumina kasutada ka kohalikus failisüsteemis paiknevaid repositooriume. Sellisel juhul tuleb repositooriumi aadressi asemel viidata vastavale kataloogile.

6.5.2 Sõltuvuste defineerimine

Lisaks sõltuvuse repositooriumi määramisele tuleb ära määrata ka vajalikud sõltuvused. Sõltuvuste defineerimise käigus on vajalik ära määrata esiteks sõltuvuse seadistus, ehk mis protsessis on vastav sõltuvus projektis vajalik. Java plugin defineerib selleks ära mitmed standardid, millest tähtsamad on järgmised: (*Gradle Inc.*)

compile	Rakenduse kompileerimiseks vajalikud sõltuvused
runtime	Rakenduse käitusajal vajalikud sõltuvused
testCompile	Rakenduse testi osa kompileerimiseks vajalikud sõltuvused
testRuntime	Rakenduse testide käitusajal vajalikud sõltuvused

Lisaks on vajalikud veel sõltuvuse grupp, nimetus ning versioon.

Rakenduse käitusajal vajaliku sõltuvuse võib defineerida järgmiselt:

```
dependencies{
     compile group:'org.springframework', name:'spring-core',
     version:'4.2.3.RELEASE'
}

//group väärtuseks on sõltuvuse grupp
//name väärtuseks on sõltuvuse JAR faili nimetus ilma versiooninumbrita
//version väärtuseks on sõltuvuse versiooninumber
```

Lühendatult on võimalik ka eelnev sõltuvus defineerida järgmiselt:

```
dependencies{
     compile 'org.springframework:spring-core:4.2.3.RELEASE'
}
```

Mitme sõltuvuse defineerimine projektis käib järgmiselt:

```
dependencies{
     compile(
           [group:'org.springframework', name:'spring-core',
           version:'4.2.3.RELEASE'],
           [group:'org.springframework', name:'spring-web',
           version:'4.2.3.RELEASE']

     )
}
```

6.5.3 Ülesanded

- Defineeri ehitusskriptis ära tavakujul kaks *runtime* tüüpi sõltuvust.
- Defineeri ehitusskriptis lühendatud kujul ära kaks *testCompile* tüüpi sõltuvust.

6.6 Rohkem *task*'ide kohta

Lisaks tavapärastele *task*'idele, on Gradle'is lisaks ka näiteks täiustatud *task*'id ning samuti on *task*'idele võimalik määrata ka reegleid.

6.6.1 Täiustatud *task*'id

Täiustatud *task*'ideks nimetatakse *task*'e, millesse on käitumine juba sisse kirjutatud. *Task*'i käitumine on defineeritud *task*'i klassis ning kui kasutaja deklareerib täiustatud *task*'i, siis tuleb ka ära määrata vastav tüüp, ehk klass. See käib järgmiselt:

```
task kopeerimisNaide(type:Copy)
```

Selles näites on ära deklareeritud ***copy task***, ehk siis *task*, kus on juba ära defineeritud kopeerimiseks vajalik. Teeme läbi näited mõndade *task*'i klasside kohta.

Copy task

```
task kopeerimisNaide(type:Copy){
     from 'kopeeritav_kaust'
     into 'sihtkaust'
     include '*.txt'
}

//from väärtuseks on kaust, mida kopeeritakse
//into väärtuseks on kaust, kuhu kopeeritakse
//include(mittekohustuslik väli) väärtuseks on failid, mida soovitakse
kopeerida
```

Käivitades siinse näite puhul käsu `gradle kopeerimisNaide`, kopeeritakse juurkaustas asuvast kaustast **kopeeritav_kaust** kõik **.txt** laiendiga failid kausta **sihtkaust**.

Delete task

```
task kustutamisNaide(type:Delete){
     delete 'naite-kaust1/naide.txt', 'naite-kaust2'
}

//delete väärtuseks on kustutava faili – või kaustanimega
```

Käivitades käesoleva näite puhul käsu `gradle kustutamisNaide`, kustutatakse esiteks juurkaustas asuvast kaustast **naite-kaust1** fail nimega **naide.txt** ning teiseks kustutatakse kaust nimega **naite-kaust2**.

ZIP task

```
task zipNaide(type:Zip){
     from 'src'
     destinationDir file('zipid')
     archiveName 'zipNaide.zip'
}

//from väärtuseks on kaust, millest ZIP fail luuakse
//destinationDir file väärtuseks on sihtkaust, kuhu ZIP fail luuakse
//archivename väärtuseks on tulevase ZIP faili nimetus
```

Käivitades siinse näite puhul käsu `gradle zipNaide`, luuakse juurkaustas asuvast kaustast **src** ZIP fail nimega **zipNaide.zip** ning paigutatakse kausta **zipid**.

JavaExec task

Seda tüüpi *task*'i jaoks on vaja eelnevalt teatud protsessid läbi teha ning seetõttu on vaja ehitusskriptis lisada projektile ka Java plugin. Selle näite jaoks loome esiteks **src/main/java** kausta uue *package*'i **test**. Seejärel loome sinna uue klassi **Test**.

Test.java:

```
package test;
public class Test{
     public static void main (String[]args){
          String arg = args.length > 0 ? args[0] : "Argumendid puuduvad!";
          System.out.println("JavaExec näide! " + arg);
     }
}
```

Ehitusskript:

```
apply plugin: 'java'

task javaExecNaide(type:JavaExec){
     classpath = sourceSets.main.runtimeClasspath
     main = 'test.Test'
     if(project.hasProperty('argument')){
          args argument
     }
}

//main väärtuseks on package'i nimi + klassi nimetus
//args väärtuseks on Java rakendusele edasi antavad argumendid
```

Käivitades loodud *task*'i käsuga `gradle javaExecNaide -Pargument='Naide'`, on väljundist näha, et esimesena läheb Java plugina tõttu tööle mitu vajalikku *task*'i ning seejärel loodud javaExecNaide *task*:

```
:compileJava
:processResources
:classes
:javaExecNaide
JavaExec näide! Naide

BUILD SUCCESSFUL
```

6.6.1.1 *Task*'i klasside loomine

Uusi task'i klasse on võimalik kasutajal ka ise luua. Järgmises näites loome ehitusskriptis uue klassi ning seejärel kasutame seda *task*'i loomiseks:

```
class UusTask extends DefaultTask {

    @TaskAction
    def naide() {
        println 'Tere!'
    }
}

task customTaskiNaide(type: UusTask)
```

Käivitades loodud *task*'i käsuga `gradle customTaskiNaide`, on väljundist näha järgmine:

```
:customTaskiNaide
Tere!

BUILD SUCCESSFUL
```

Väljundist on näha, et käivitus ***@TaskAction*** annotatsiooniga meetod **naide**. See annotatsioon märgib seda, et vastav meetod peab käivituma, kui *task* käivitatakse.

6.6.2 Ant'i kasutamine Gradle'is

Gradle toetab ka Ant'i task′ite ehitusskripti importimise. Loome ühe Ant *task*'i, selleks loome kõigepealt projekti juurkausta **build.xml** faili. Seejärel avame selle ning loome järgmise *task*'i:

```
<project>
     <target name='antNaide'>
          <echo>See on Ant'i naide!</echo>
     </target>
</project>
```

Nüüd avame Gradle'i ehitusskripti ning impordime build.xml'is loodu:

```
ant.importBuild 'build.xml'
```

Nüüd on võimalik meil build.xml'is loodud *task* käivitada käsuga `gradle antNaide` ning väljundis näha järgmist:

```
:antNaide
[ant:echo] See on Ant'i naide!

BUILD SUCCESSFUL
```

6.6.3 *Task*'ide reeglid

Task'idele on võimalik määrata ka reegleid. Reeglite määramine võib kasulik olla, kui *task* peaks mingitel tingimustel tegutsema teatud viisil. Loome ehitusskripti järgmiseks ühe näite, mis vastavalt *task*'i nimele tagastab väljundina vastava sõnumi:

```
tasks.addRule('Pattern: inimene-<nimi>'){
     String nimi ->
     if(nimi - 'inimene-' == 'Kalle'){
          task(nimi) << {
               println 'Nimi on ' + (nimi - 'inimene-')
          }
     }
     else{
          task(nimi) << {
               println 'Nimi ei ole Kalle'
          }
     }
}
```

Käivitades loodud *task*'i käsuga `gradle inimene-Kalle`, on väljundist näha järgmine:

```
:inimene-Kalle
Nimi on Kalle

BUILD SUCCESSFUL
```

Ükskõik millise muu *task*'i nimega saame järgmise väljundi:

```
Nimi ei ole Kalle
```

6.6.4 Ülesanded

- Loo ZIP tüüpi task, mis juurkaustas asuvast kaustast **failid/dokumendid** loob ZIP faili nimega **dokumendid.zip** kausta **failid/zipid**.
- Loo JavaExec tüüpi task, mis annab Java klassile kaasa kaks argumenti, mis väljundis Java rakenduse poolt välja prinditakse.
- Loo ehitusskriptis uus *task*'i klass ning defineeri selles ära kaks lihtsat meetodit, mis mõlevad prindivad välja lihtsa sõnumi.

6.7 Ehituse tööprotsessi logimine

Ehituse tööprotsessi logimine on Gradle'is küllaltki lihtsalt tehtav. Gradle'i poolt on juba loodud instants **Logger'i** klassist. Samas võime luua ehitusskriptis ka uue instantsi järgmiselt:

```
Logger uuslogger = Logging.getLogger('uus-logger')
```

Loome ehitusskripti uue *task*'i, kus meie äsja loodud Logger'i instantsi juba kasutatakse:

```
task zipLogiNaide(type:Zip){
     def kaust = file('src2')
     String kaustanimi = kaust.getName()
     Logger uuslogger = Logging.getLogger('uus-logger')
     if(!kaust.exists()){
          uuslogger.warn("$kaustanimi kausta ei ole olemas! Zipi ei looda")
          return
     }
     else{
          uuslogger.info("$kaustanimi kaustast luuakse ZIP")
          from kaust
     }
     uuslogger.info("ZIP'i sihtkaustaks on zipid")
     destinationDir file('zipid')

     uuslogger.info("Loodud ZIP faili nimeks on zipNaide.zip")
     archiveName 'zipNaide.zip'
}
```

Selle näite puhul on *task*'i lisatud kahte sorti logisõnumeid: hoiatus- ja infosõnum. Hoiatusõnumid on nähtavad igaljuhul, et aga infosõnumeid näha, tuleb *task*'ile anda kaasa ka **-i** *flag*:

```
gradle -i zipLogiNaide
```

Väljund on seekord küll tunduvalt kirjum, aga kui siinse näite puhul on src kaust juurkaustas olemas, peaks väljundis järgmine olema üles leitav:

```
src kaustast luuakse ZIP
ZIP'i sihtkaustaks on zipid
Loodud ZIP faili nimeks on zipNaide.zip
```

6.7.1 Ülesanne

- Loo lihtne *copy* tüüpi *task* ning täiusta seda vajalikuks olevate logisõnumitega.

6.8 Testide jooksutamine

Juhul, kui Java projektis on loodud teste, siis tavapäraselt Gradle'i *build task* käivitab need. Testide jooksutamise näiteks loome projekti **JUURKAUST/src/test/java** kausta uue *package*'i **naide.testid**, kuhu sisse me loome **TestiNaide** klassi. Sinna loome omakorda lihtsa *unit* testi:

```
package naide.testid;

import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class TestiNaide {

    int arv1 = 3;
    int arv2 = 2;

    @Test
    public void liitmiseNaide(){
        assertEquals(5, arv1 + arv2);
    }
}
```

Järgmiseks lisame ehitusskripti järgmise:

```
apply plugin: 'war'

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
}
```

Käivitades käsu `gradle test`, on väljundist näha, et test käivitus, aga puudub täpsem info. Et väljundist oleks ka näha, milline test käivitus, lisame ehitusskripti lisaks ***beforeTest***

meetodi, mis käivitub eelnevalt testi kävitumisele. Samuti laseme Gradle'il luua **testid** kausta testiraporti:

```
test{
    beforeTest {
        test ->
            logger.lifecycle('Käivitub test: ' + test.getName())
    }

    reports {
           html.enabled = true
           html.destination = file('testid')
    }
}
```

Käivitades taaskord käsu `gradle test`, on väljundist seekord näha järgmine:

```
:compileJava
:processResources
:classes
:compileTestJava
:processTestResources
:testClasses
:test
Käivitub test: Test liitmiseNaide(naide.testid.TestiNaide)
```

Test task'i uuesti käivitamisel see sama test rohkem ei käivitu, kuna task on *up-do-date* seisundis. Et test uuesti käivituks, tuleb teha minimaalselt muudatus vähemalt testi parameetrite väärtuses (**arv1** või **arv2** väärtus).

6.8.1 Ülesanne

- Loo mitu lihtsat *unit* testi ning määra ehitusskriptis ka ära, et käivitunud testi nimi ka logis väljenduks. Lisaks lase testide tulemustest luua ka testiraport ning uuri seda.

6.9 Projekti ning ehitusskripti loomine

Selle peatüki käigus loome ehitusskripti Java veebirakenduse projektile, millel on lisaks üks väiksem alamprojekt. Projektil on ka mitmeid sõltuvusi, mis on rakenduse tööks vajalikud. Projektide loomiseks kasutatakse NetBeans'i programmeerimiskeskkonda. Alustame NetBeansi avamisest. Seejärel installeerime **Gradle Support** plugina:

- Avame pluginate akna (Tools → Plugins)
- Avame vahelehe **Available Plugins**
- Installeerimise **Gradle Support** plugina

Kui plugin on installeeritud, teeme taaskäivituse NetBeans'ile. Seejärel loome uue Gradle'i projekti nimega **NaideProjekt**:

- File → New Project
- Gradle → Gradle Root Project

Kui uus projekt on loodud, siis esimese asjana teeme **build.gradle** ning **common.gradle** failid tühjaks.

Seejärel avame **Files** vaheakna ning loome **src/main/java** alla uue *package*'i. Siinse näite puhul on *package*'i nimeks **ee.naide.mudelid**.

Äsja loodud *package*'i alla loome nüüd uue klassi nimega **Naide**:

```
package ee.naide.mudelid;

public class Naide {

    private int id;
    private String nimi;
    private int number;

    public Naide(int id, String nimi, int number){
        this.id = id;
        this.nimi = nimi;
        this.number = number;
    }

    public int getId(){
        return this.id;
    }

    public void setId(int id){
        this.id = id;
    }

    public String getNimi(){
        return this.nimi;
    }

    public void setNimi(String nimi){
        this.nimi = nimi;
    }

    public int getNumber(){
        return this.number;
    }

    public void setNumber(int number){
        this.number = number;
    }

}
```

Seejärel avame projekti juurkaustas asuva **build.gradle** faili ning lisame sinna järgmise (pärast ehitusskripti uuendamist tee parem hiireklikk projektil ja vali *Reload Project*):

```
apply plugin: 'war'

repositories {
    mavenCentral()
}

dependencies {
    compile group: 'org.springframework', name: 'spring-webmvc', version:
'4.2.3.RELEASE'
    runtime group: 'javax.servlet', name: 'jstl', version: '1.2'
}
```

Järgmiseks loome uue package'i: **ee.naide.kontrollerid**. Seejärel loome uue klassi nimega

NaideKontroller, milles me teeme vajalikud kaardistused:

```
package ee.naide.kontrollerid;

import ee.naide.mudelid.Naide;
import java.util.ArrayList;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class NaideKontroller {

    @RequestMapping("/naide")
    public String showNaited(Model model){

        List<Naide> naited = new ArrayList();

        naited.add(new Naide(0, "Kalle", 5));
        naited.add(new Naide(0, "Malle", 25));
        naited.add(new Naide(0, "Juku", 125));

        model.addAttribute("naited", naited);

        return "naide";
    }

}
```

Seejärel loome projekti **src/main** kausta uue kausta nimega **webapp**. Sellesse kausta loome omakorda uue kausta nimega **WEB-INF**. WEB-INF kausta loome uue faili nimega **web.xml**, millesse me lisame järgmise:

```
<web-app>
    <display-name>Näidisrakendus</display-name>
    <servlet>
        <servlet-name>naidis</servlet-name>
          <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
          </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>naidis</servlet-name>
        <url-pattern>/naide</url-pattern>
    </servlet-mapping>
</web-app>
```

Nüüd on vaja luua uus XML fail, kus me deklareerime ära *viewResolver*'i. Selle faili loomisel tuleb tähele panna, et faili nime struktuur peab olema järgmine:

```
<servlet-name>-servlet.xml
```

Meie tehtud näite puhul on XML faili nimeks **naidis-servlet.xml**. Faili 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"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.2.xsd">

    <context:component-scan base-package="ee.naide.kontrollerid"/>

    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
      <property name="prefix" value="/WEB-INF/pages/"/>
      <property name="suffix" value=".jsp"/>
    </bean>
</beans>
```

Järgmiseks loome WEB-INF kausta uue kausta: **pages**. Seejärel loome WEB-INF/pages kausta **naide.jsp** faili, mille ülesandeks on kontrollerist saadud info välja kuvada:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
    <head>
        <title>Näidisrakendus</title>
    </head>
    <body>
        <c:forEach var = 'naide' items = '${naited}'>
            ${naide.id}, ${naide.nimi}, ${naide.number} </br>
        </c:forEach>
    </body>
</html>
```

Täiendame ka veel **build.gradle** faili, kus me lisame *war task*'i alla projekti versiooni ning samuti lisame manifesti faili paar atribuuti.

```
war {
    version = '1.0'
    manifest {
        attributes (
            'Implementation-Title' : 'Näidisrakendus',
            'Implementation-Version' : version
        )
    }
}
```

Järgmisena loome meie loodud juurprojektile alamprojekti. Võtame esiteks juurprojekti valikusse ning seejärel loome uue Gradle'i projekti nimega **AlamProjekt**:

- File → New Project
- Gradle → Gradle Subproject

Kui projekt on loodud, teeme alamprojekti alla loodud **build.gradle** faili tühjaks ning asendame **settings.gradle** failis oleva järgmisega:

```
include 'AlamProjekt'
```

Järgmisena loome **src/main/java** kausta alla uue *package*'i nimega **ee.alamnaide**. Järgmisena loome uue klassi nimega **Alamnaide**, mis sisaldab näitena vaid ühte lihtsalt meetodit:

```
package ee.alamnaide;

public class Alamnaide {

    public String alamNaideMeetod(int arv){
        return arv > 15 ? "See arv on suurem kui 15" :
        "See arv on võrdne, või väiksem kui 15";
    }
}
```

Järgmisena lisame juurprojektis asuvale **build.gradle** failile järgmise:

```
subprojects {
    apply plugin: 'java'

    jar {
        version = '1.0'
        manifest {
            attributes (
                'Implementation-Title' : 'Alamrakendus',
                'Implementation-Version' : version
            )
        }
    }
}
```

Nagu eelnevast näitest on näha, luuakse kõigist juurprojekti alamprojektidest JAR fail. Järgmisena käivitame ehitusskripti järgmise käsuga:

```
gradle build
```

Tulemusena luuakse juur – ning alamprojekti **/build/libs** kausta vastavalt WAR ning JAR fail. Loome lisaks veel juur – ja alamprojektile *task*'i, mis kopeerib ehitatud JAR ning WAR faili ühisesse kausta nimega **warid**. Ehitusskriptis juurprojekti alla asetuv *task* on järgmine:

```
task copyNaiderakendus(type:Copy){
    from 'build/libs'
    into 'buildid'
    rename {
        name ->
        name.replace(name, 'naiderakendus.war')
    }
}
```

Alamprojekti *task* asetub ehitusskriptis *subprojects* alla ning on järgmine:

```
task copyAlamrakendus(type:Copy){
     from 'build/libs'
    into '../buildid'
    rename {
           name ->
        name.replace(name, 'alamrakendus.jar')
    }
}
```

Seejärel anname ehitusskriptis *build task*'ile külge meetodi, mis käivitab eelnevalt loodud *task*'id pärast seda, kui *build task* on töö lõpetanud.

```
build.finalizedBy(copyNaiderakendus) // juurprojekti alla

build.finalizedBy(copyAlamrakendus) // subprojects alla
```

Täielikul kujul on loodud ehitusskript järgmine:

```
apply plugin: 'war'

repositories {
    mavenCentral()
}

dependencies {
    compile group: 'org.springframework', name: 'spring-webmvc', version:
'4.2.3.RELEASE'
    runtime group: 'javax.servlet', name: 'jstl', version: '1.2'
}

war {
    version = '1.0'
    manifest {
        attributes (
            'Implementation-Title' : 'Näidisrakendus',
            'Implementation-Version' : version
        )
    }
}

task copyNaiderakendus(type:Copy){
    from 'build/libs'
    into 'buildid'
    rename {
        name ->
        name.replace(name, 'naiderakendus.war')
    }
}

subprojects {
    apply plugin: 'java'

    jar {
        version = '1.0'
        manifest {
            attributes (
                'Implementation-Title' : 'Alamrakendus',
                'Implementation-Version' : version
            )
        }
    }

    task copyAlamrakendus(type:Copy){
        from 'build/libs'
        into '../buildid'
        rename {
            name ->
            name.replace(name, 'alamrakendus.jar')
        }
    }
    build.finalizedBy(copyAlamrakendus)
}
build.finalizedBy(copyNaiderakendus)
```

Viimaks käivitame *build task*'i käsuga `gradle build`.

Väljundist on näha, et lisaks rakenduse kompileerimisele, pakendamisele ja teistele *task*'idele, käivitus nii juur – ning alamprojekti puhul ka meie loodud kopeerimise *task*.

```
:compileJava
:processResources
:classes
:war
:assemble
:compileTestJava
:processTestResources
:testClasses
:test
:check
:build
:copyNaiderakendus
:AlamProjekt:compileJava
:AlamProjekt:processResources
:AlamProjekt:classes
:AlamProjekt:jar
:AlamProjekt:assemble
:AlamProjekt:compileTestJava
:AlamProjekt:processTestResources
:AlamProjekt:testClasses
:AlamProjekt:test
:AlamProjekt:check
:AlamProjekt:build
:AlamProjekt:copyAlamrakendus

BUILD SUCCESSFUL
```

Kokkuvõte

Käesoleva seminartiöö eesmärgiks oli luua eestikeelne õppematerjal Gradle'i tööriista kohta, mis esiteks annab lugejale käesoleva tööriista kohta algteadmised ning samuti annab lugejale võimaluse iseseisvalt õppida tööriista kasutama.

Seminaritöö esimestes osades räägiti Gradle'i tööriistast, selle alternatiividest ning samuti tutvustati lugejale juba olemasolevaid õppematerjale. Õppematerjali osas tehti läbi erinevaid näiteid ning samuti anti kasutajale võimalus täita autori poolt loodud ülesandeid. Õppematerjali viimases osas loodi ehitusskript veebirakenduse projektile, millel oli samuti üks alamprojekt.

Köesolevat õppematerjali testisid ka kaks testijat. Testimise tulemusena said testijad paremat aimu Gradle'i tööriistast ning ehitusskripti loomisest. Samuti oli õppematerjali testimine suureks abiks autorile, sest testimise käigus avastatud probleemid said lahendatud ning testijate nõuandeid sai õppematerjalis kasutatud.

Lisaks eelnevale, õppis autor õppematerjali luues ka ise palju juurde - esiteks Gradle'it kasutades ehitusskripti loomise kohta, kuid samuti õppematerjali üldise vormistamise kohta.

Kasutatud materjal

Gradle Inc. *About Gradle Inc*. Loetud 1. november 2015 aadressil https://gradle.org/company/

Gradle Inc. Overview. Loetud 2. november 2015 aadressil https://docs.gradle.org/current/userguide/overview.html

Gradle Inc. Build Script Basics. Loetud 23. november 2015 aadressil https://docs.gradle.org/current/userguide/tutorial_using_tasks.html

Gradle Inc. Gradle Plugins. Loetud 23. november 2015 aadressil https://docs.gradle.org/current/userguide/plugins.html

Gradle Inc. Dependency Management Basics. Loetud 16. detsember 2015 aadressil https://docs.gradle.org/current/userguide/artifact_dependencies_tutorial.html

Gradle Inc. *Logging*. Loetud 23. november 2015 aadressil https://docs.gradle.org/current/userguide/logging.html

Gradle Inc. The Java Plugin. Loetud 16. detsember 2015 aadressil https://docs.gradle.org/current/userguide/java_plugin.html

Apache Software Foundation. What is Maven? Loetud 24. november 2015 aadressil https://maven.apache.org/what-is-maven.html

Apache Software Foundation. Loetud 24. november 2015 aadressil http://ant.apache.org/

Gradle Inc. Installing Gradle. Loetud 16. detsember 2015 aadressil https://docs.gradle.org/current/userguide/installation.html

Gradle Inc. Writing Custom Plugins. Loetud 18. detsember 2015 aadressil https://docs.gradle.org/current/userguide/custom_plugins.html

Gradle Inc. Gradle User Guide. Loetud 16. detsember 2015 aadressil https://docs.gradle.org/current/userguide/userguide.html

Irani R. (2014, 28. juuli). Announcing .. Gradle Tutorial Series. Loetud 25. november 2015 aadressil http://rominirani.com/2014/07/28/gradle-tutorial-series-an-overview/

Kainulainen P. *Getting Started With Gradle*. Loetud 20. detsember 2015 aadressil http://www.petrikainulainen.net/getting-started-with-gradle/

Vogel L., Scholz S. (2015, 2. oktoober). *The Gradle build system- Tutorial*. Loetud 18. detsember 2015 aadressil http://www.vogella.com/tutorials/Gradle/article.html

Lisad

Lisa 1. Õppematerjalis loodud näited

Õppematerjali käigus loodud näited on kättesaadavad järgmiselt veebiaadressilt:

http://www.tlu.ee/~svenkris/gradle_naited