



samen sterk voor werk

Jenkins



Inhoudsopgave

1	INLEIDING.....	3
1.1	Doelstelling.....	3
1.2	Vereiste voorkennis.....	3
1.3	Nodige software	3
1.4	Continuous Integration (CI)	3
2	JENKINS DOWNLOADEN EN INSTALLEREN	4
2.1	Download	4
2.2	Installatie	4
2.3	Configuratie	4
2.3.1	JDK.....	4
2.3.2	Maven	5
2.3.3	E-mail	5
2.3.4	FindBugs plugin	5
2.3.5	Cobertura plugin	5
2.3.6	Deploy plugin	6
3	DE APPLICATIE	7
4	DE JENKINS JOB	8
4.1	De job definiëren.....	8
4.2	De job uitvoeren.....	8
4.3	Het resultaat van de uitvoering van de job	9
5	EEN UNIT TEST DIE FAALT	10
5.1	De falende test toevoegen	10
5.2	De Jenkins job uitvoeren	10
5.3	De correctie	11
5.4	De Jenkins job terug uitvoeren	11
6	CODE COVERAGE MET COBERTURA.....	12
6.1	Uitbreiding code	12

6.2	pom.xml.....	12
6.3	De Jenkins job.....	12
6.4	De Jenkins job uitvoeren	12
6.5	Het Cobertura rapport	13
6.6	Extra tests.....	13
7	STATIC CODE ANALYSIS MET FINDBUGS	15
7.1	pom.xml.....	15
7.2	De Jenkins job.....	15
7.3	De Jenkins job uitvoeren	15
8	JAVADOC	17
8.1	pom.xml.....	18
8.2	De Jenkins job.....	18
8.3	De Jenkins job uitvoeren	18
9	E-MAIL NOTIFICATIE.....	20
9.1	De Jenkins job.....	20
9.2	De Jenkins job uitvoeren	20
10	CONTINUOUS DELIVERY	21
10.1	Het war bestand ter beschikking stellen	21
10.2	Het war bestand deployen op een webserver	21
11	COLOFON.....	22

1 INLEIDING

1.1 Doelstelling

Je leert werken met Jenkins. Dit is een open source CI (continuous integration) tool.
Je leert straks wat continuous integration betekent.

1.2 Vereiste voorkennis

- Java
- JSP/Servlets
- Maven
- Git

1.3 Nodige software

- Een JDK (versie 7 of hoger)
- Eclipse for Java EE Developers (versie Luna)
- Maven
- Git

1.4 Continuous Integration (CI)

Je ontwikkelt een enterprise applicatie zelden op je eentje, maar meestal met een team.

Een versiebeheersysteem (Git, ...) op een gedeelde repository bevat de sources van de applicatie.

Om een wijziging aan te brengen aan de applicatie doe je volgende stappen:

1. Je pulst de sources van de gedeelde repository naar je private repository
2. Je doet de wijzigingen
3. Je voert de unit tests uit die bij die wijzigingen horen
4. Je voert de integration test uit die bij die wijzigingen horen
5. Je commit de wijzigingen naar je private repository
6. Je pusht je wijzigingen van je private repository naar de gedeelde repository

Na deze push moeten alle applicatie onderdelen (ook diegene die je niet wijzigde) correct werken.

Je controleert dit door alle unit tests en alle integration tests van de applicatie uit te voeren.

Dit vraagt veel tijd. Het zou onhandig zijn dit uit te voeren op de computer van een ontwikkelaar.

Een Continuous integration (CI) tool lost dit probleem op.

Je voert een CI tool uit op een test server.

Dit is een aparte computer, niet één van de computers van de ontwikkelaars.

De CI tool controleert regelmatig (bvb. om de minuut) of een ontwikkelaar een push deed naar de gedeelde repository. Als dit het geval is, compileert de tool de volledige applicatie en voert alle unit tests en integration tests uit. De tool publiceert daarna een rapport van deze taken.

Wanneer het past voor de ontwikkelaars, lezen ze dit rapport. Het bevat informatie over eventueel mislukte tests. De ontwikkelaars kunnen daarmee correcties aanbrengen.

Handig is dus:

- De CI tool compileert en test *automatisch* nadat één van de ontwikkelaars een push doet. Je moet dit werk dus niet handmatig in de CI tool starten.
- De CI tool werkt op een aparte test server en verbruikt geen processor of RAM van een computer van één van de ontwikkelaars.
- Terwijl de CI tool zijn werk doet, doen de ontwikkelaars ander werk. Ze wachten niet op de CI tool. Wanneer het hen past lezen ze het rapport van de CI tool. De CI tool kan dit rapport zelfs met een mail naar de ontwikkelaars sturen.

Een CI tool kan naast het compileren en testen van de applicatie extra taken uitvoeren, zoals

- De Javadoc documentatie van de applicatie maken
- Tools uitvoeren die de kwaliteit van de code van de applicatie controleren

Je ziet dit verder in de cursus.

2 JENKINS DOWNLOADEN EN INSTALLEREN

Jenkins draait normaal op een aparte test server. Je voert in deze cursus Jenkins uit op je computer die waarschijnlijk ook de computer is waarop je ontwikkelt.

2.1 Download

Je kan Jenkins downloaden op jenkins.io, met de knop Download Jenkins.

Je kiest daarna de LTS (Long-Term Support) Release.

Je krijgt dan een bestand `jenkins.war`

2.2 Installatie

De extensie van dit bestand is `war`. Dit betekent dat Jenkins een web applicatie is die je op een webserver (bijvoorbeeld Tomcat) kan installeren en uitvoeren.

Je kan daarna naar de Jenkins web applicatie surfen om taken op Jenkins uit te voeren.

Er bestaat echter ook een andere, eenvoudige manier om Jenkins uit te voeren.

`jenkins.war` bevat een embedded (ingebakken) webserver. Als je `jenkins.war` start via een console venster, start deze embedded webserver. Je kan daarna met een browser naar die webserver surfen om taken op Jenkins uit te voeren.

1. Je maakt in de root van de harde schijf een folder `jenkins`
2. Je kopieert `jenkins.war` in die folder.
3. Je start een Command Line venster
4. Je tikt `cd /jenkins` en je drukt Enter
Je bevindt je nu in de folder `jenkins`.
5. Je tikt `java -jar jenkins.war --httpPort=8081` en je drukt Enter
Je start daarmee Jenkins en je laat de embedded webserver draaien op TCP/IP poort 8081. Standaard draait die webserver op poort 8080 en kan zo in conflict komen met Tomcat die waarschijnlijk ook op je computer draait. Het starten kan even duren.
Op het einde van de startprocedure zie je `INFO: Jenkins is fully up and running`.
Je laat het Command Line venster open, anders zou je Jenkins sluiten.

Jenkins toont teksten in de taal (Engels, Nederlands, ...) die je in de browser opties configureerde. Deze cursus gebruikt Engels. We raden daarom aan in je browser ook Engels te kiezen.

Je surft met een browser naar `localhost:8081` en je ziet de startpagina van Jenkins. Je moet daar het paswoord van de Jenkins administrator tikken. Je krijgt een hint in welk bestand je dit paswoord vindt.

Je kiest in de volgende pagina `Install suggested plugins`.

Je ziet in de volgende pagina dat plugins gedownload en geïnstalleerd worden.

Je maakt in de volgende pagina een gebruiker aan.

Je komt daarna op de hoofdpagina van Jenkins.

2.3 Configuratie

2.3.1 JDK

Je geeft aan in welke directory de JDK (Java Development Kit) geïnstalleerd is

1. Je kiest `Manage Jenkins` (links in de hoofdpagina)
2. Je kiest `Global Tool Configuration`
3. Je kiest `Add JDK`
4. Je tikt een vrij te kiezen naam bij `Name`
5. Je verwijdert het vinkje bij `Install automatically`
6. Je tikt bij `JAVA_HOME` het pad naar directory waar je JDK geïnstalleerd is.
Je verlaat het veld en wacht enkele seconden. Als het pad verkeerd is, zie je een fout.

2.3.2 Maven

Je geeft aan in welke directory Maven geïnstalleerd is

1. Je kiest Add Maven
2. Je tikt een vrij te kiezen naam bij Name
3. Je verwijdert het vinkje bij Install automatically
4. Je tikt bij MAVEN_HOME het pad naar de directory waar Maven geïnstalleerd is.
Je verlaat het veld en wacht enkele seconden. Als het pad verkeerd is, zie je een fout.
5. Je kiest Save

2.3.3 E-mail

Je configureert de mailserver en de account waarmee Jenkins mails verstuurt.

Je gebruikt normaal de mailserver van je firma. Je zal hier de mailserver van GMail gebruiken.

1. Je kiest Manage Jenkins (links in de hoofdpagina)
2. Je kiest Configure System
3. Je tikt de gebruikersnaam van je GMail account bij System Admin e-mail address (onder Jenkins Location)
4. Je tikt smtp.gmail.com bij SMTP server (onder E-mail notification)
5. Je kiest Advanced
6. Je plaatst een vinkje bij Use SMTP Authentication
7. Je tikt de gebruikersnaam van je GMail account bij punt 3 bij User Name
8. Je tikt het bijbehorende paswoord bij Password
9. Je plaatst een vinkje bij Use SSL
10. Je tikt 465 bij SMTP Port
11. Je plaatst een vinkje bij Test configuration by sending test e-mail
12. Je tikt een e-mail adres bij Test e-mail recipient
13. Je kiest Test configuration
14. De boodschap Email was succesfully sent verschijnt en je kiest Save

De in-box van de gebruiker, vermeld bij puntje 9, bevat een mail met de titel Test email

Je installeert als volgende enkele Jenkins plugins die je zal nodig hebben in deze cursus

2.3.4 FindBugs plugin

Jenkins kan met deze plugin de kwaliteit van je code controleren (verder in de cursus)

1. Je kiest Manage Jenkins (links in de hoofdpagina)
2. Je kiest Manage Plugins
3. Je kiest Available
4. Je tikt findbugs naast Filter (rechts boven)
5. Je plaatst een vinkje bij FindBugs Plugin
6. Je kiest Install without restart

2.3.5 Cobertura plugin

Jenkins kan met deze plugin de kwaliteit van je tests controleren (verder in de cursus)

1. Je kiest Manage Jenkins (links in de hoofdpagina)
2. Je kiest Manage Plugins
3. Je kiest Available
4. Je tikt cobertura naast Filter (rechts boven)
5. Je plaatst een vinkje bij Cobertura Plugin
6. Je kiest Install without restart

2.3.6 Deploy plugin

Jenkins kan met deze plugin een war bestand deployen op een webserver

1. Je kiest Manage Jenkins (links in de hoofdpagina)
2. Je kiest Manage Plugins
3. Je kiest Available
4. Je tikt deploy to container naast Filter (rechts boven)
5. Je plaatst een vinkje bij Deploy to container Plugin
6. Je kiest Install without restart

3 DE APPLICATIE

De applicatie waarmee je Jenkins leert kennen is de Maven applicatie GoedeDoe1 in het materiaal bij deze cursus. Je importeert deze applicatie in Eclipse

1. Je kiest in het menu File de opdracht Import
2. Je kiest in de categorie Maven voor Existing Maven Projects en je kiest Next
3. Je kiest Browse, je kiest de folder GoedeDoe1 in het materiaal bij de cursus en je kiest OK
4. Je kiest Finish

Je maakt van dit project een Git project (zie Git cursus) en je doet een eerste commit.

Je pusht het project daarna naar GitHub.

4 DE JENKINS JOB

Je maakt in Jenkins één job per applicatie die je met Jenkins wil verwerken (compileren, testen, ...)

4.1 De job definiëren

Je maakt een job voor de applicatie GoedeDoel, met deze stappen in de startpagina van Jenkins

1. Je kiest links in de Jenkins hoofdpagina New Item (of je kiest create new jobs)
2. Je tikt bij Enter an item name een naam voor de job: Goede doel
Deze naam mag verschillen van de naam van de Maven applicatie.
3. Je kiest Maven project
4. Je kiest OK

Je komt op een pagina waarin je aangeeft welke taken deze job moet uitvoeren.

5. Je kiest bij Source Code Management voor Git
Je ziet dat Jenkins ook andere versiebeheersystemen ondersteunt, zoals Subversion.
6. Je tikt bij Repository URL de URL van je GitHub repository, je verlaat het veld en wacht enkele seconden. Als de ingetikte URL verkeerd is, zie je een foutmelding
7. Je plaatst bij Build Triggers een vinkje bij Poll SCM.
Je tikt daarna H/2 * * * * bij Schedule
Je geeft daarmee aan hoe regelmatig Jenkins moet controleren of een nieuwe versie van de applicatie gepusht is naar de Git repository.
 - H/2 iedere 2 minuten
 - * van ieder uur
 - * van iedere dag
 - * van iedere maand
 - * van ieder jaar


8. Je kies Save

Hiermee heb je gedefinieerd wat de job moet doen.

4.2 De job uitvoeren

Als je twee minuten wacht, voert Jenkins de job uit, want je hebt in de job aangegeven dat Jenkins om de twee minuten de Git repository moet nazien op wijzigingen.

Je zal niet zolang wachten en de job manueel starten.

Je kiest daartoe links Build Now of het icoon 

Je zal dit icoon op veel pagina's zien. Je kan het altijd gebruiken om de job manueel te starten.

Je ziet na enkele seconden links een balk die de vooruitgang van de job aangeeft









Als deze balk verandert in een hyperlink, is de job uitgevoerd.

4.3 Het resultaat van de uitvoering van de job

Je kiest links boven Jenkins om de beginpagina te zien

Je ziet in het midden van de pagina een samenvatting van de uitgevoerde job

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Goede doel	4 min 32 sec - #1	N/A	14 sec

- De 1° kolom geeft informatie over de meest recente uitvoering van de job.
 -  uitgevoerd zonder fouten
 -  slecht uitgevoerd (slecht geconfigureerde job of compilerfouten)
 -  foutieve testen of (later in de cursus) code van mindere kwaliteit
- De 2° kolom geeft informatie over *alle* uitvoeringen van de job (niet enkel de meest recente uitvoering).
 - Naarmate een job meer met succes uitgevoerd werd, zie je een icoon dat mooier weer voorstelt, zoals 
 - Naarmate een job meer met fouten uitgevoerd werd, zie je een icoon dat slecht weer voorstelt, zoals  of 

Als je in de 3° kolom [Goede doel](#) kiest zie je een pagina met details over de uitvoeringen van de job. De pagina bevat een hyperlink [Latest Test Result](#).

Als je deze kiest, zie je een rapport met de resultaten van de JUnit tests van het project

Test Result

0 failures

				
			1 tests	
Module	Fail	(diff)	Total	(diff)
be.vdab.goededoel	0		1	+1

De tekst 0 failures met daaronder een blauwe balk geeft aan dat alle tests slaagden.

Als je links Console Output kiest, zie je de output van het uitvoeren van de job.

Dit is interessant als die uitvoering mislukt, omdat je dan foutmeldingen ziet in deze pagina.

5 EEN UNIT TEST DIE FAALT

Je zal aan het project een falende unit test toevoegen en leren hoe Jenkins hiermee omgaat.

5.1 De falende test toevoegen

Je voegt in Eclipse aan de class `GoedeDoelTest` een method toe die controleert of een nieuw doel nul € opgebracht heeft.

```
@Test
public void eenNieuwDoelHeeftNogNietsOpgebracht() {
    assertEquals(0, doel.getOpgebracht().compareTo(BigDecimal.ZERO));
}
```

Deze test zal falen met een `NullPointerException`, omdat in de class `GoedeDoel` de variabele `opgebracht` niet geïnitieerd wordt.

Je doet een Git commit van deze uitbreiding en je pusht het project naar GitHub.

5.2 De Jenkins job uitvoeren

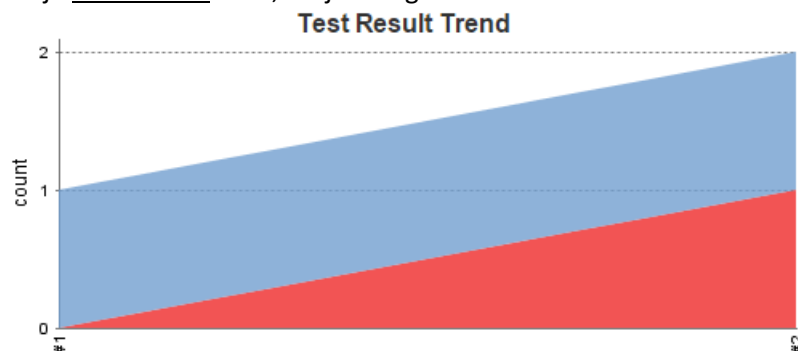
Je voert in Jenkins de job `Goede doel` opnieuw uit.

Je ziet daarna in de welkompagina volgend resultaat

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Goede doel	21 sec - #2	N/A	8,4 sec 

 wijst op één of meerdere falende tests.

Als je [Goede doel](#) kiest, zie je een grafiek met statistieken over de uitgevoerde JUnit tests



#1 onder de grafiek staat voor de 1° uitvoering van de job.

Hierbij werd één test uitgevoerd (count 1) en die had succes (blauwe kleur).

#2 onder de grafiek staat voor de 2° uitvoering van de job.

Hierbij werden twee tests uitgevoerd (count 2)

één had succes (blauwe kleur) en één mislukte.

Als je in deze pagina [Latest Test Results](#) kiest, zie je volgende pagina

Test Result

1 failures (+1)

2 tests (+1)			
Module	Fail	(diff)	Total (diff)
be.vdab.goededoel	1	+1	2 +1

Failed Tests

[be.vdab.goededoel](#)

Test Name	Duration	Age
be.vdab.entities.GoedeDoelTest.eenNieuwDoelHeeftNogNietsOpgebracht	0.003	1

Je kan met [+](#) meer detail zien over de mislukte test

Failed Tests

[be.vdab.goededoel](#)

Test Name
be.vdab.entities.GoedeDoelTest.eenNieuwDoelHeeftNogNietsOpgebracht
Stack Trace
java.lang.NullPointerException at be.vdab.entities.GoedeDoelTest.eenNieuwDoelHeeftNogNietsOpgebracht(GoedeDoelTest.java:24)

5.3 De correctie

Je corrigeert de fout in Eclipse met een aanpassing in de class GoedeDoe1

private BigDecimal **opgebracht** = **BigDecimal.ZERO**;

Je doet een Git commit van deze uitbreiding en je pusht het project naar GitHub.

5.4 De Jenkins job terug uitvoeren

Je voert in Jenkins de job opnieuw uit. Je ziet daarna in de welkompagina volgend resultaat

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Goede doel	16 sec - #3	N/A	12 sec 

Als je [Goede doel](#) kiest, zie je terug de grafiek met statistieken over de uitgevoerde JUnit tests. De derde uitvoering (#3) van de job bevat twee geslaagde tests.



6 CODE COVERAGE MET COBERTURA

Een code coverage tool controleert welke code (bvb. uit de class GoedeDoel) doorlopen wordt bij het uitvoeren van de bijbehorende test (bvb. GoedeDoelTest) en maakt hiervan een rapport.

Als je ziet dat bepaalde code helemaal niet, of slechts gedeeltelijk doorlopen wordt door de tests, is dit een indicator dat je voor die code extra tests moet schrijven.

Er bestaan meerdere Java code coverage tools. Cobertura is er één van.

6.1 Uitbreiding code

Je voegt aan de class GoedeDoel code toe, waarvoor je geen unit test schrijft:

```
@Override
public boolean equals(Object object) {
    if (!(object instanceof GoedeDoel)) {
        return false;
    }
    return ((GoedeDoel) object).naam.equalsIgnoreCase(this.naam);
}
```

6.2 pom.xml

Jenkins zal Cobertura uitvoeren als één van de stappen van het Maven build process van je project. Je voegt in Eclipse aan pom.xml de Maven plugin voor Cobertura toe, voor de regel </plugins>

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>cobertura-maven-plugin</artifactId>
  <version>2.7</version>
  <configuration>
    <formats>
      <format>xml</format>
    </formats>
  </configuration>
</plugin>
```

Je doet een Git commit en je pusht het project naar GitHub.

6.3 De Jenkins job


Je breidt in Jenkins de job Goede doel uit, zodat hij als een onderdeel van het Maven build process ook Cobertura uitvoert, via de plugin die je aan pom.xml toevoegde.


1. Je kiest in de beginpagina in het midden Goede doel
2. Je kiest links Configure
3. Je tikt cobertura:cobertura bij Goals and options
Het uitvoeren van Cobertura wordt zo een stap van het Maven build proces.
4. Je kiest Add post-build action bij Post-Build Actions
en je kiest Publish Cobertura Coverage Report
5. Je tikt **/coverage.xml bij Cobertura xml report pattern
6. Je kiest Save

6.4 De Jenkins job uitvoeren

Je voert de job opnieuw uit. Je ziet daarna in de welkompagina volgend resultaat

S	W	Name ↓	Last Success	Last Failure	Last Duration	
		Goede doel	51 sec - #9	7 min 43 sec - #6	11 sec	

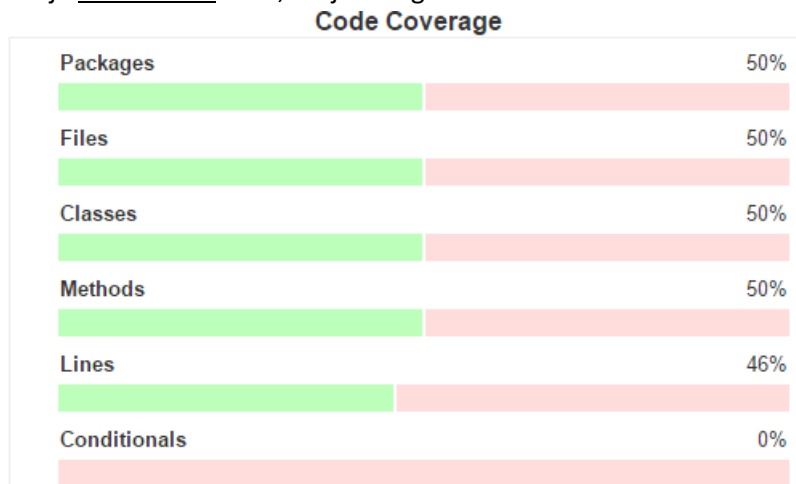
 betekent dat de laatste uitvoering van de job lukte.

 betekent dat over alle uitvoeringen van de job er kwaliteitsverlies is.

Dit is ontstaan door het gebruik van Cobertura, die merkt dat we er code is die niet getest wordt.

6.5 Het Cobertura rapport

Als je Goede doel kiest, zie je een grafiek met statistieken over het uitvoeren van Cobertura



- De 1° balk geeft aan dat de tests 50% van de packages uitvoeren
- De 2° balk geeft aan dat de tests 50% van de source bestanden uitvoeren
- De 3° balk geeft aan dat de tests 50% van de classes uitvoeren
- De 4° balk geeft aan dat de tests 50% van de methods uitvoeren
- De 5° balk geeft aan dat de tests 46% van de lijnen code uitvoeren
- De 6° balk geeft aan dat de tests 0% code binnen if of else uitvoeren



Het is niet de bedoeling dat je op al deze items 100% haalt.

Je source bevat eenvoudige code, zoals getters en setters aangemaakt door Eclipse, waarvoor je geen JUnit tests schrijft en die de percentages van Cobertura doen zakken.

Als je midden in de pagina Coverage Report kiest, zie je de details van het cobertura rapport

Je kiest daarin be.vdab.entities en daarin GoedeDoel.java

Je ziet roze blokken in de marge voor de code van de method equals.

Dit betekent dat deze code niet uitgevoerd werd tijdens de unit test

```
@Override
public boolean equals(Object object) {
    if (!(object instanceof GoedeDoel)) {
        return false;
    }
    return ((GoedeDoel) object).naam.equalsIgnoreCase(naam);
}
```

6.6 Extra tests

Je voegt in Eclipse volgende code toe aan de class GoedeDoelTest

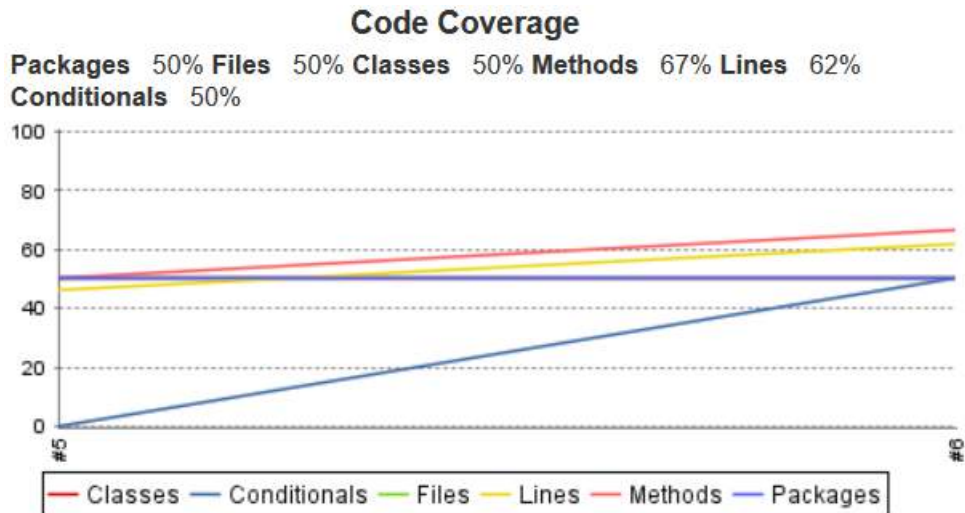
```
@Test
public void goedeDoelenMetDezelfdeNaamMoetenVolgensEqualsGelijkZijn() {
    assertEquals(new GoedeDoel("CLINICLOWNS"), doel);
}

@Test
public void goedeDoelenMetVerschillendeNaamMoetenVolgensEqualsVerschillendZijn()
{
    assertNotEquals(new GoedeDoel("Unicef"), doel);
    // extra import: import static org.junit.Assert.assertNotEquals;
}
```

Je doet Git een commit en je pusht het project naar GitHub.

Je voert in Jenkins de job terug uit.

De job bevat vanaf nu ook een Cobertura grafiek over het uitvoeren van de jobs heen



- Je hebt al meer methods getest (67% nu, 50% vroeger)
- Je hebt meer lijnen code getest (62% nu, 46% vroeger)
- Je hebt meer code binnen een if of else getest (62% nu, 0% vroeger)

Als je midden in de pagina Coverage Report kiest, zie je de details van het cobertura rapport

Je kiest daarin [be.vdab.entities](#) en daarin [GoedeDoel.java](#)

Je ziet via het roze blok dat er nog een deel van de method niet getest wordt.

```
@Override
public boolean equals(Object object) {
    if (!(object instanceof GoedeDoel)) {
        return false;
    }
    return ((GoedeDoel) object).naam.equalsIgnoreCase(naam);
}
```

Je voegt in Eclipse volgende code toe aan de class GoedeDoelTest om dit te verhelpen

```
@Test
public void goedDoelVerschiltVanEenObjectMetEenAnderType() {
    assertEquals(doe1, new Date());
}
```

Je doet een Git commit en je pusht het project naar GitHub.

Je voert in Jenkins de job terug uit.

Je hebt nu 100% op conditionals en de class GoedeDoel en de conditionals.

Je kan ook zien dat de tests alle code in de method equals uitvoeren

```
@Override
public boolean equals(Object object) {
    if (!(object instanceof GoedeDoel)) {
        return false;
    }
    return ((GoedeDoel) object).naam.equalsIgnoreCase(naam);
}
```

7 STATIC CODE ANALYSIS MET FINDBUGS

Een static code analysis tool controleert of je project geen fouten bevat die veel voorkomen:

- Strings vergelijken met `==` in plaats van met `equals`
- databaseconnecties vergeten te sluiten
- ...

Zo'n tool bevat een grote database met gekende fouten. De tools controleert je code ten opzichte van deze fouten. Fouten die jij ook maakte komen in een rapport terecht.

FindBugs is zo'n tool en is open source.

7.1 pom.xml

Jenkins zal FindBugs uitvoeren als één van de stappen van het Maven build process van je project. Je voegt in Eclipse aan `pom.xml` de Maven plugin voor FindBugs toe, voor de regel `</plugins>`

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>findbugs-maven-plugin</artifactId>
  <version>3.0.4</version>
  <configuration>
    <xmlOutput>true</xmlOutput>
    <effort>Max</effort>
  </configuration>
</plugin>
```

Je doet een Git commit en je pusht het project naar GitHub.

7.2 De Jenkins job

Je breidt in Jenkins de job Goede doel uit, zodat hij als een onderdeel van het Maven build process ook FindBugs uitvoert, via de plugin die je aan `pom.xml` toevoegde.

1. Je kiest in de beginpagina in het midden Goede doel
2. Je kiest links Configure
3. Je voegt een spatie en `findbugs:findbugs` toe bij Goals and options
Het uitvoeren van FindBugs wordt zo een stap van het Maven build proces.
4. Je plaatst een vinkje bij Publish FindBugs analysis results onder Build Settings
5. Je kiest Save

7.3 De Jenkins job uitvoeren

Je voert in Jenkins de job Goede doel opnieuw uit.

De pagina met de job bevat nu links een onderdeel FindBugs warnings. Je kiest deze.

Je ziet dat FindBugs een fout heeft gevonden in je code: je hebt de method `equals` overridden, maar niet de method `hashCode`. Je verbreekt zo de regel dat als twee objecten volgens `equals` gelijk zijn, zij dezelfde returnwaarde moeten hebben van de method `hashCode`

GoedeDoel.java:40, HE_EQUALS_USE_HASHCODE, Priority: High

be.vdab.entities.GoedeDoel defines equals and uses Object.hashCode()

This class overrides `equals(Object)`, but does not override `hashCode()`, and inherits the implementation of `hashCode()` from `java.lang.Object` (which returns the identity hash code, an arbitrary value assigned to the object by the VM). Therefore, the class is very likely to violate the invariant that equal objects must have equal hashcodes.

If you don't think instances of this class will ever be inserted into a `HashMap/HashTable`, the recommended `hashCode` implementation to use is:

```
public int hashCode() {
    assert false : "hashCode not designed";
    return 42; // any arbitrary constant will do
}
```


Je voegt in Eclipse volgende code toe aan de class GoedeDoe1 om dit te verhelpen

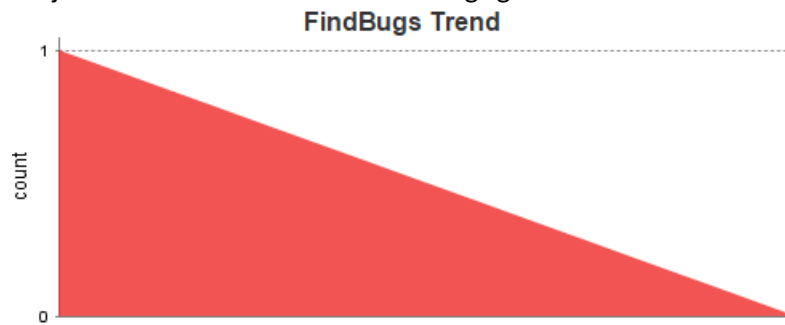
```
@Override  
public int hashCode() {  
    return this.naam.toUpperCase().hashCode();  
}
```

Je voegt aan GoedeDoe1Test een bijbehorende test toe

Je doet een Git commit en je pusht het project naar GitHub.

Je voert in Jenkins de job terug uit.

De job bevat vanaf nu ook een FindBugs grafiek over het uitvoeren van de jobs heen

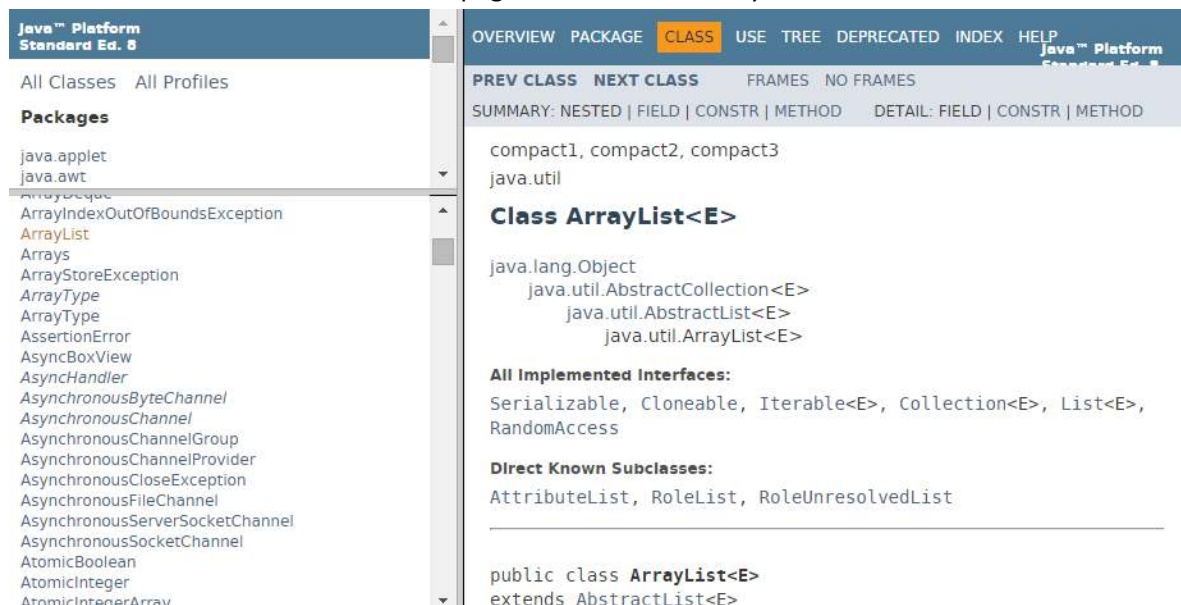


Gezien er geen FindBugs fouten meer zijn, is er links geen hyperlink [FindBugs warnings](#).

8 JAVADOC

Een onderdeel van een Jenkins job kan het genereren van project documentatie zijn.

Deze documentatie bestaat uit HTML pagina's met dezelfde lay-out als de Java API documentatie



Deze documentatie wordt gegenereerd op basis van commentaar die je aan de te documenteren classes toevoegt.

Je schrijft juist voor de class GoedeDoel zo'n commentaar

```
/**
 * Een <strong>goed doel</strong> waarvoor men geld inzamelt
 * @author Joe Dalton
 */
```

①
②
③
④

- (1) Deze commentaar begint met /**
- (2) Iedere regel begint met *
Je beschrijft de betekenis van de class.
Je kan in de tekst ook HTML tags, zoals gebruiken
- (3) Je schrijft na @author de naam van de auteur van de class.
- (4) Je eindigt de commentaar met */

Je kan met zo'n commentaar ook een constructor of method documenten.

Je schrijft juist voor de constructor

```
/**
 * Maakt een GoedeDoel object
 * @param naam De naam van het goede doel
 */
```

①

- (1) Je schrijft per method- of constructor parameter een regel die begint met @param
Je schrijft daarna de naam van de parameter
Je schrijft daarna de betekenis van de parameter

Je schrijft juist voor de method getNaam

```
/**
 * Geeft de naam terug
 * @return de naam
 */
```

①

- (1) Als de returnwaarde van de method verschilt van void, schrijft je een regel die begint met @return, gevolgd door de betekenis van de returnwaarde.

Je schrijft juist voor de method `getOpgebracht`

```
/**
 * Geeft terug hoeveel &euro; dit doel heeft opgebracht
 * @return het opgebracht bedrag
 */
```

Je kan ook documentatie schrijven over een package.

Je voegt daartoe een bestand met de naam `package-info.java` toe aan de package.

Je klikt de package `be.vdab.entities` aan met de rechtermuisknop

en je kiest New, Other, General, File

```
/**
 * De classes in deze package stellen dingen uit de werkelijkheid voor
 */
package be.vdab.entities;
```

8.1 pom.xml

Jenkins zal op basis van de commentaar de documentatie in HTML formaat maken als één van de stappen van het Maven build process van je project.

Je voegt aan `pom.xml` de Maven plugin voor javadoc toe, voor de regel `</project>`

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>2.10.4</version>
    </plugin>
  </plugins>
</reporting>
```

Je doet een Git commit (waarbij je ook `package-info.java` aanvinkt)

en je pusht het project naar GitHub.

8.2 De Jenkins job

Je breidt in Jenkins de job Goede doel uit, zodat hij als een onderdeel van het Maven build process ook de documentatie, via de plugin die je aan `pom.xml` toevoegde.

1. Je kiest in de beginpagina in het midden Goede doel
2. Je kiest links Configure
7. Je voegt een spatie en `javadoc:javadoc` toe bij Goals and options
Het aanmaken van documentatie wordt zo een stap van het Maven build proces.
3. Je kiest Save

8.3 De Jenkins job uitvoeren

Je voert in Jenkins de job Goede doel opnieuw uit.

De pagina met de job bevat nu een hyperlink Javadoc. Je kiest deze.

Je ziet documentatie bij de package `be.vdab.entities`

`be.vdab.entities` De classes in deze package stellen dingen uit de werkelijkheid voor

Je kiest deze package. Je ziet documentatie over de class `GoedeDoel`

`GoedeDoel` Een goed doel waarvoor men geld inzamelt

Je kiest deze class. Je ziet documentatie over de constructor en de methods

Constructor Summary

Constructors

Constructor and Description
<code>GoedeDoel(String naam)</code> Maakt een GoedeDoel object

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
boolean	<code>equals(Object object)</code>	
String	<code>getNaam()</code> Geeft de naam terug	
BigDecimal	<code>getOpgebracht()</code> Geeft terug hoeveel € dit doel heeft opgebracht	

Je kiest de constructor. Je ziet documentatie over de constructor parameter.

Je ziet bij de method `getNaam` ook documentatie over de returnwaarde

Constructor Detail

GoedeDoel

```
public GoedeDoel(String naam)
```

Maakt een GoedeDoel object

Parameters:

`naam` - De naam van het goede doel

Method Detail

getNaam

```
public String getNaam()
```

Geeft de naam terug

Returns:

de naam

9 E-MAIL NOTIFICATIE

Nadat Jenkins een job heeft uitgevoerd, kan Jenkins een e-mail sturen naar één of meerdere personen. Dit zullen normaal de personen zijn die meewerken aan de te ontwikkelen applicatie.

Jenkins stuurt een e-mail

- als de uitvoering van een job mislukt
- als de uitvoering van een job lukt nadat die bij de vorige uitvoering mislukte

9.1 De Jenkins job

Je breidt in Jenkins de job Goede doel uit, zodat hij een e-mail verstuurt

1. Je kiest in de beginpagina in het midden Goede doel
2. Je kiest links Configure
3. Je plaatst een vinkje bij E-mail notification
4. Je tikt één of meerdere e-mail adressen bij Recipients, gescheiden door een spatie
5. Je kiest Save

9.2 De Jenkins job uitvoeren

Je wijzigt een unit test zodat die mislukt.

Je voert in Jenkins de job Goede doel opnieuw uit.

Je ziet daarna in de in-box van één van de recipients een mail met het rapport van het uitvoeren van de job.

Je corrigeert de unit test en je voert de job opnieuw uit.



Opmerking: er bestaan Jenkins plugins waarmee je op andere manieren dan een e-mail personen kan waarschuwen dat een job is uitgevoerd. Er bestaat een plugin om een SMS te versturen, een plugin om een boodschap te tonen rechts in de taakbalk van de computer van de persoon, ...

10 CONTINUOUS DELIVERY

De stappen die je job tot nu toe uitvoert vallen onder de naam continuous integration, zoals beschreven in het begin van de cursus.

Continuous delivery gaat nog een stap verder na continuous integration:

als de job met succes is uitgevoerd, wordt het eindproduct geïnstalleerd

- op de test webserver waar testers de website vanuit hun browser testen
- of zelfs op de productie webserver
zodat de eindgebruikers de nieuwe versie van de applicatie gebruiken.

10.1 Het war bestand ter beschikking stellen

Als eerste stap van continuous delivery zal je het war bestand, dat het resultaat is van een geslaagde job uitvoering, ter beschikking stellen van de beheerders van de test servers of productieservers. Zij kunnen dit war bestand dan installeren op de test servers of productieservers. Dit is nog niet de *automatische* continuous delivery, want zij installeren het bestand *handmatig*.

Je breidt in Jenkins de job Goede doel uit, zodat hij het war bestand ter beschikking houdt

1. Je kiest in de beginpagina in het midden Goede doel
2. Je kiest links Configure
3. Je voegt een spatie en package toe bij Goals and options
4. Je kiest Add post-build action
5. Je kiest Archive the artifacts
6. Je tikt `**/*.war` bij Files to archive
Dit betekent het bestand met de extensie war (`*.war`) in om het even welke directory die zelf een subdirectory kan zijn (`**`)
7. Je kiest Save

Je voert in Jenkins de job Goede doel opnieuw uit.

De pagina met de job bevat nu een hyperlink [goededoel-0.0.1-SNAPSHOT.war](#)

De beheerders van de test servers of productieservers kunnen hiermee de gebouwde applicatie downloaden en installeren op hun servers.

10.2 Het war bestand deployen op een webserver

Je start Tomcat standalone (via startup.bat) (niet via Eclipse of Tomcat) op poort 8080

Je breidt in Jenkins de job Goede doel uit, zodat hij het war bestand installeert op die Tomcat

1. Je kiest in de beginpagina in het midden Goede doel
2. Je kiest links Configure
3. Je kiest Add post-build action
4. Je kiest Deploy war/ear to a container
5. Je tikt `**/*.war` bij WAR/EAR files
6. Je tikt goededoel bij Context path
Dit is het context path waarmee het war bestand op de Tomcat geïnstalleerd wordt.
7. Je kiest Add Container
8. Je kiest Tomcat 7.x (ook als je Tomcat 8 gebruikt)
9. Je tikt `cursist` bij Manager user name
Dit is een Tomcat gebruiker die de rol manager-script heeft (zie cursus JSP/Servlets)
10. Je tikt `cursist` bij Manager password
11. Je tikt `http://localhost:8080` bij Tomcat URL
12. Je kiest Save

Je voert in Jenkins de job Goede doel opnieuw uit.

Je kan daarna surfen naar <http://localhost:8080/goededoel> om de gedeployde applicatie op Tomcat in werking te zien.

11 COLOFON

Domeinexpertisemanager:	Jean Smits
Moduleverantwoordelijke:	Hans Desmet
Medewerkers:	Hans Desmet
Versie:	27/3/2017
Nummer dotatielijst:	