

Oefeningen

Test training



Oefeningen test training

Om de oefeningen te kunnen maken is het noodzakelijk om de bijbehorende Maven projecten op het halen.

Open een terminal en geef de volgende commando's.

```
# cd Documents
# hg clone https://bitbucket.org/infosupport/testtraining
Username: cursist
Password: p@ssw0rd
# cd testtraining
```

Oefening 1 – JUnit

Oefening voorbereiding

Importeer project 01-junit in Eclipse (File→ import → existing Maven project)

***Context:** Info Support wil haar cursisten beter van dienst zijn door cursisten de mogelijkheid te geven online een profiel aan te maken. Dit maakt het gemakkelijker om historie in te zien, inschrijven voor nieuwe cursussen en feedback te geven op cursussen. Ten behoeve daarvan wordt een cursist registratiesysteem gebouwd. Daarvan is een zeer klein deel reeds gemaakt. Aan u de taak om één van de meest essentiële zaken te bouwen: het inschrijfproces.*

De functionele eisen die aan het proces gesteld worden zijn de volgende:

- Wanneer een cursist zich registreert is het account standaard inactief;
- Na registratie ontvangt de cursist een activatiecode;
- Deze activatiecode gecombineerd met zijn gebruikersnaam zijn nodig om het account actief te maken;
- Er kunnen geen cursisten met dezelfde gebruikersnaam geregistreerd worden;
- Een cursist kan maar éénmaal geactiveerd worden.

Stap 0 – De basis

Schrijf unit testen voor de klasse CursistDao.

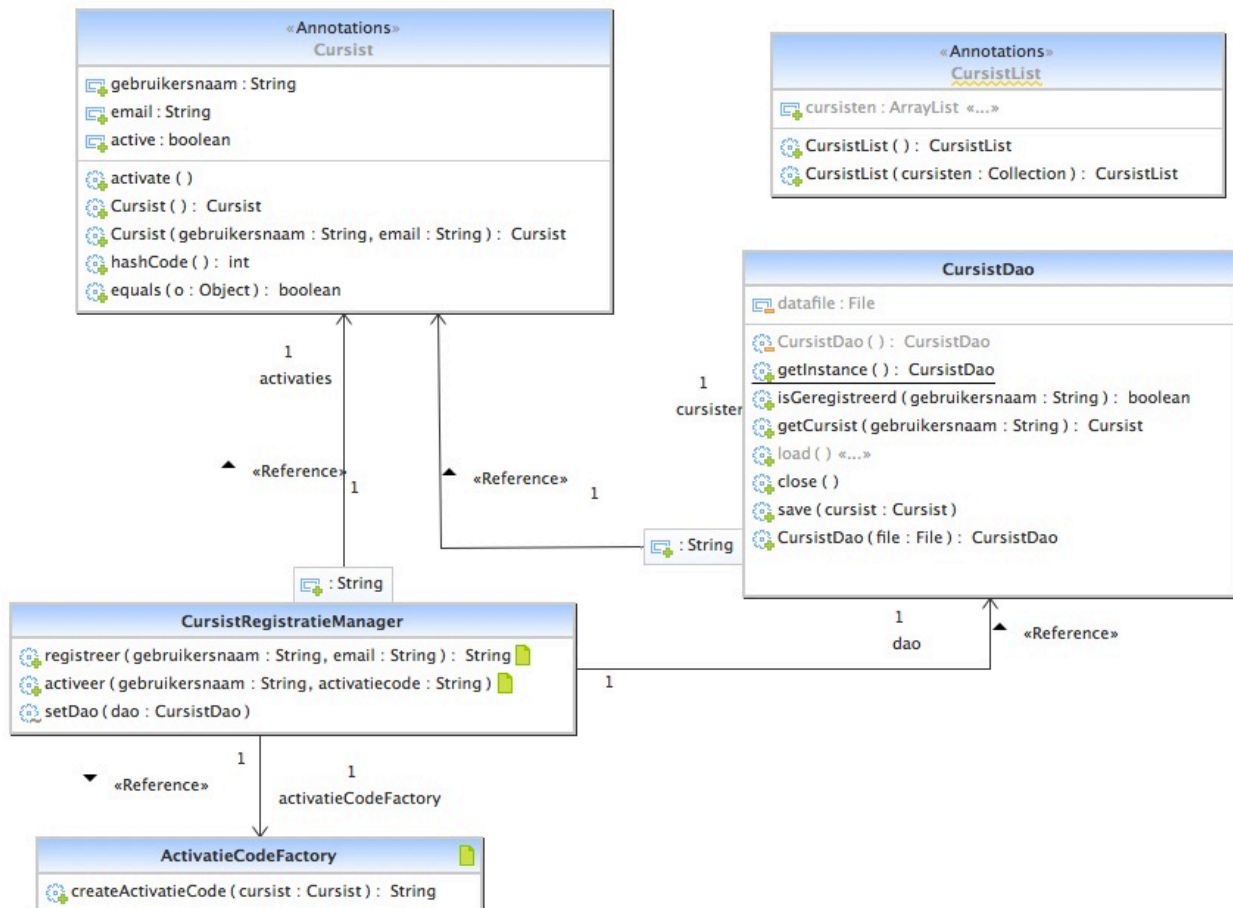
Stap 1 – Implementeer het registratie- en activatieproces

Open het project testtraining en bekijk de verschillende klassen. De klasse *CursistRegistratieManager* is de klasse die in deze opgave gebouwd gaat worden. Daar hoort de klasse *CursistRegistratieManagerTest* bij.

Bekijk de functionele eisen en bouw het registratie- en activatieproces. Dat wil zeggen: implementeer de *registreer(...)* en *activeer(...)* methoden volgens de genoemde eisen.

U heeft een skelet van de test klasse reeds gekregen.

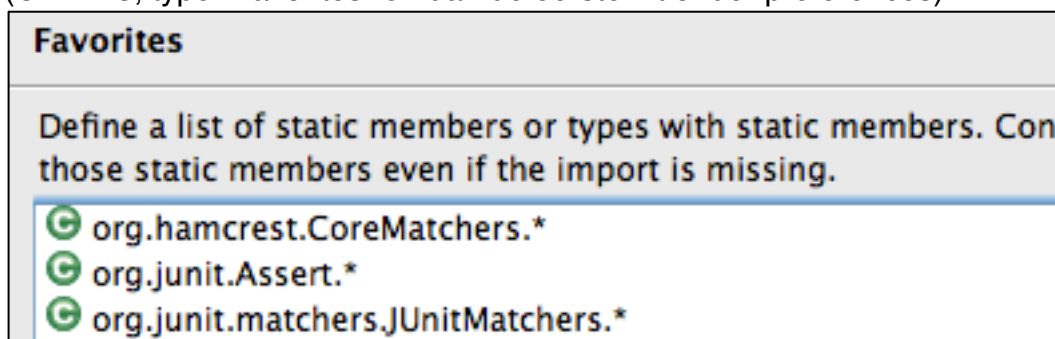
Een classdiagram is beschikbaar.



Een aantal tips voor het uitwerken:

- Werk de implementatie test driven uit;
- Voor het genereren van een activatiecode kan gebruik gemaakt worden van de activatiecodefactory (private field op de CursistRegistratieManager klasse);
- Om ervoor te kunnen zorgen dat CTRL+spatie gebruikt kan worden voor static imports, configureer eerst Eclipse door de volgende klassen als Favorites te registreren:

(CTRL+3, type "Favorites" en dan de eerste hit onder preferences)

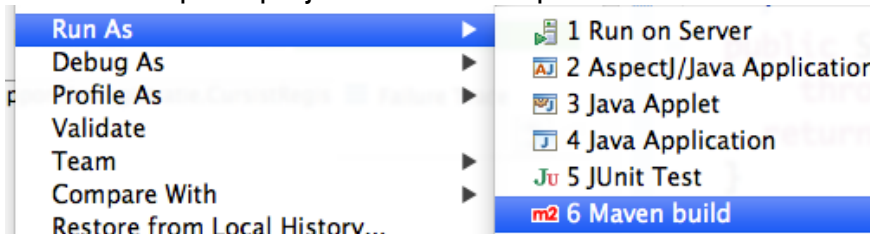


Stap 2 – Controleer de test coverage

Om er zeker van te zijn dat alle productiecode geraakt wordt door unit testen (wanneer test-driven gewerkt is, is dat het geval) is het goed om de test coverage te meten. Daarvoor kan gebruik gemaakt worden van een Eclipse gebaseerde plugin Jacoco of van een Maven gebaseerde plugin Cobertura.

In deze stap doen we het via Maven.

- Klik rechts op het project en kies de optie “Run As → Maven build”



- Geef de volgende goal op: `cobertura:cobertura` en klik op Run
- Ga nu naar de project map op schijf toe en open het bestand “`target/site/cobertura/index.html`” en bekijk de coverage score.

Stap 3 – Test suite

Maak een test suite aan voor de geïmplementeerde unit testen. Voeg in het vervolg elk te maken unit test klasse aan deze suite toe.

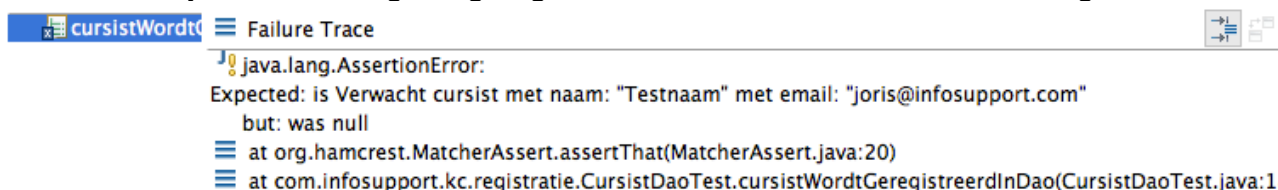
Oefening 2 – Matchers

Oefening voorbereiding

Importeer project 02-matchers in Eclipse (File→ import → existing Maven project)

Op verschillende plaatsen wordt gebruik gemaakt van de klasse `CursistDao`. Dit hart van het systeem is gebouwd zonder unit testen! U kunt zich voorstellen dat dit onacceptabel is. Vandaar dat het maken van unit testen hiervan nu de hoogste prioriteit heeft.

- Open de klasse: `CursistDaoTest`;
- De match logica staat nog in commentaar. Implementeer de nu nog lege klasse `CursistMatcher` zodat de unit test werkelijk iets test!
- Bij het niet overeenkomen van verwachte en werkelijke data, dient de matcher een duidelijke foutmelding terug te geven zoals in onderstaande afbeelding.



Oefening 3 – Mocking

Oefening voorbereiding

Importeer project 03-mocking in Eclipse (File→ import → existing Maven project)

Importeer de uitwerking van de vorige oefening.

Bekijk de uitwerking en vergelijk het met de eigen testuitwerkingen.

Open nu de klasse CursistDao en ga op zoek naar de save() methode. Zet de regel cursisten.put(...) in commentaar. En run nu opnieuw de unit testen van de CursistRegistratieManagerTest.

Merk op dat – hoewel er niets veranderd is aan de CursistRegistratieManager klasse – alle testen nu toch falen.


Hoe komt dit?


Om ervoor te zorgen dat we alleen het gedrag van de CursistRegistratieManager klasse aan het testen zijn, gaan we de CursistDao mocken.

Zorg ervoor dat de unit testen van de CursistRegistratieManagerTest weer werkend worden door het gedrag van de CursistDao met behulp van mocks te specificeren.

Tips:

- Voeg de klasse volgende klassen toe aan de Favorite preferences in Eclipse:

 org.mockito.Matchers.*

 org.mockito.Mockito.*

Oefening 4 – Test data constructie

Oefening voorbereiding

Importeer project 04-testdata in Eclipse (File→ import → existing Maven project)


Importeer de uitwerking van de vorige oefening.


Bekijk de CursistRegistratieManagerTest.


Het zal opvallen dat op verschillende plaatsen een Cursist object aangemaakt wordt. In alle plaatsen wordt naast de gebruikersnaam, ook het emailadres meegegeven. Uit de context van de unittest is niet direct af te leiden of dit emailadres werkelijk relevant is voor de test.


Om te voorkomen dat allerlei verplichte velden in unittests ingevuld moeten worden, dient de testdata constructie met object builders gemaakt te worden.

- Breid de lege klasse CursistBuilder uit de volgende methoden:

 metNaam(String) : CursistBuilder

 metEmail(String) : CursistBuilder

 build() : Cursist

 ^s eenCursist() : CursistBuilder

Tip: kijk ook op de slides!

- Laat de CursistRegistratieManagerTest klasse gebruik maken van de CursistBuilder.

Oefening 5 – Parameterized unit tests

Oefening voorbereiding

Importeer project 05-parameterized-tests in Eclipse (File→ import → existing Maven project)

Tot nu toe maakt het systeem gebruik van een klasse `ActivatieCodeFactory`. Hiervoor zijn wederom nog geen unittesten gemaakt. Hoog tijd om daar verandering in aan te brengen. Dit is typisch een klasse die niet één specifieke uitkomst wil testen, maar een lijst van input parameters door dezelfde test halen. Hiervoor zijn Parameterized testen uitermate schikt.

In onderstaande tabel is aangegeven wat de verwachte uitkomsten zijn behorend bij een cursist object.

Gebruikersnaam in cursistobject	Verwachte uitkomst als activatiecode
"Jan"	"74231"
"Piet"	"2487432"
"Klaas"	"72577842"

Maak een unittest in de klasse `ActivatieCodeFactoryTest` die met behulp van unittest parameters een test uitvoert waarbij de uitkomst gevalideerd wordt tegen de verwachte uitkomst uit bovenstaande tabel.

Oefening 6 – Rules en Categories

Oefening voorbereiding

Importeer project 06-rules-categories in Eclipse (File→ import → existing Maven project)

Stap 1

Open de klasse CursistDaoTest. In deze klasse bevindt zich een methode – die initieel bedoeld is om met rules te leren werken – waarmee gevalideerd wordt dat cursistobjecten naar een bestand geschreven en gelezen kunnen worden.

Deze unit test is nog niet volledig geïmplementeerd. Volg de aanwijzingen en maak de unit test af.

Stap 2

Tot nu toe zijn er alleen unit testen gemaakt. Daar zich nu een keten begint te vormen (Manager – Dao – File) is het goed om ook een integratietest te maken.

Open het bestand RegistratieManagerIntegrationTests en volg de aanwijzingen in de voor gedefinieerde test methode.

Stap 3

Maak een nieuwe test suite aan en zorg ervoor dat deze suite alleen integratietesten draait.

Tip: gebruik de slides – onderdeel categories!

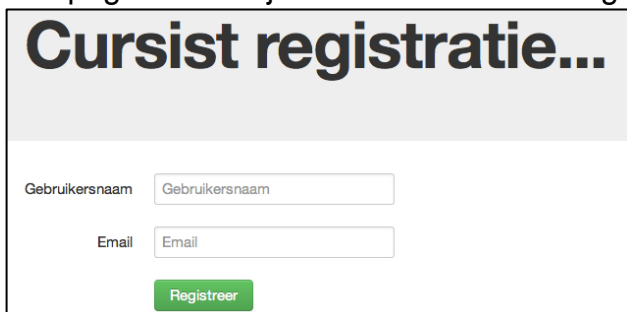
Oefening 7 – Testen van Web applicaties met Selenium

Oefening voorbereiding

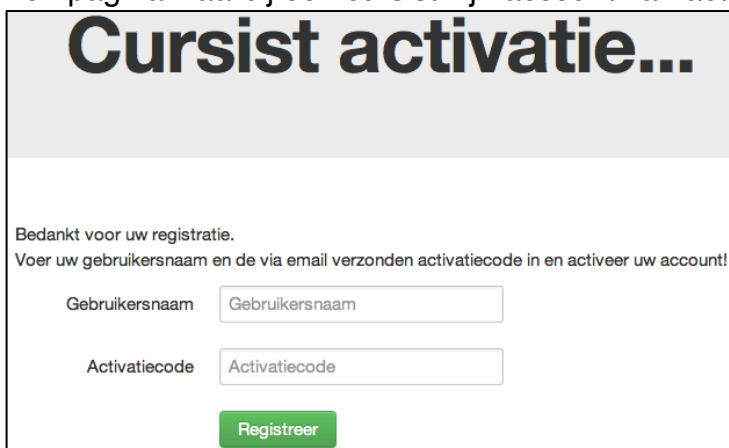
Importeer project 7-webtesten in Eclipse (File → import → existing Maven project)

In deze oefening maken we gebruik van dezelfde usecase alleen nu uitgewerkt in een web applicatie. De applicatie kent verschillende web pagina's:

1. Een pagina waarbij een cursist zich kan registreren en hij een activatiecode krijgt;



2. Een pagina waarbij een cursist zijn account kan activeren;



3. Een pagina waarbij hij de account details (nu alleen zijn naam) te zien krijgt.



Start de applicatie door rechts te klikken op het project en te kiezen voor run-as → Maven build...

Geef als Goals "jetty:run" en klik op Run.

Open daarna een browser en bekijk de applicatie via de url <http://localhost:8080>.

Stap 1 – Aanmaken RegistratiePage

In deze stap beginnen we met het aanmaken van de page die hoort bij de registratie pagina.

Maak een nieuwe klasse genaamd RegistratiePage in de package `com.infosupport.kc.registratie.web.pages`. Bekijk de registratie web pagina in de browser en merk op dat er een aantal web elementen van belang zijn: 1) het gebruikersnaam input veld 2) het emailadres inputveld en 3) de registreer knop.

Zorg ervoor dat deze velden als private attributen in de aangemaakte klasse beschikbaar zijn. Met een `@FindBy` annotatie kan aangegeven worden aan welke HTML elementen de attributen gekoppeld moeten worden. (Kijk in de slides!).

Voeg nu drie methoden toe:

1. `setGebruikersnaam(String naam)` waarmee de gegeven naam als waarde gezet wordt op het input veld;
2. `setEmail(String email)` waarmee het gegeven emailadres als waarde gezet wordt op het bijbehorende inputveld;
3. `registreer()` die een klik doet op het `registreerButton` veld en net zo lang wacht totdat de activeer-pagina getoond wordt.

Geef de page klasse een public constructor die een `WebDriver` object als argument binnen krijgt.

Om een instantie van de `RegistratiePage` aan te maken, maken we gebruik van een factory methode. Voeg een public static methode toe genaamd `open(WebDriver driver)` en maak met behulp van de `PageFactory` (zie slides) een instantie van de `RegistratiePage` aan.

Stap 2 – Testen van de Registratie page

Maak in de testfolder nu een nieuwe klasse aan genaamd: `RegistratieGuiTest` onder de package `com.infosupport.kc.registratie.web`.

Voeg daar een methode aan toe genaamd `before()` waarin een nieuwe `FirefoxDriver` instantie aangemaakt wordt. Zorg ervoor dat deze methode vóór elke unittest uitgevoerd wordt. Voeg tevens een methode toe die ná elke unittest uitgevoerd wordt waarin de driver gesloten wordt middels het aanroepen van de `close()` methode.

Maak nu een unit test genaamd:

`cursistKanZichRegistrerenMetCorrectIngevoerdeGebruikersnaamEnEmail()` en zorg ervoor dat er een instantie van een `RegistratiePage` object verkregen wordt. Set de gebruikersnaam en email en roep de `registreer()` methode aan.

Verifieer dat nu de pagina getoond wordt.

Run de test!

Stap 3 – Testen van de Activatie page

Maak nu een pageobject aan voor de Activatiepagina. De activatiecode – die standaard eigenlijk via mail uitgelezen moet worden – is nu: “secret-“ + gebruikersnaam.

Dus als de gebruikersnaam “jan” is, is de activatiecode “secret-jan”.

Pas nu de `registreer()` methode aan op de `RegistratiePage` en zorg ervoor dat in plaats dat deze methode void is, dat deze een instantie terug geeft van de `ActivatiePage`. Want, na klikken op registreer, wordt er immers een activatiepagina geopend.

Pas daarna de test aan zodat op activatiepage object de bijbehorende gegevens ingevuld worden om uiteindelijk een accountpage te krijgen.