**Een voorbeeld van het schrijven van een unittest**

**inleiding**

Het schrijven van een goede unittest is vaak niet eenvoudig en vereist soms enige creativiteit en vooral een strenge doch rechtvaardige blik. Hieronder wordt een aanpak getoond hoe je een unittest voor een klasse kunt schrijven. Uitgangspunt is uiteraard dat de betreffende klasse goed gespecificeerd is, dat wil zeggen *van elke publieke methode en constructor zijn eenduidig en volledig*

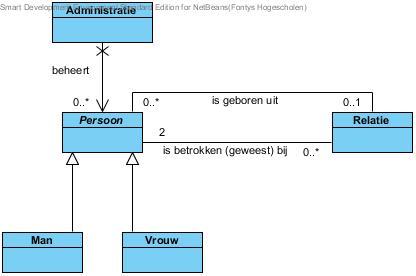
1. *het effect van de aanroep op de toestand van het object waarvoor de methode is aangeroepen,*
2. *zijn eventuele returnwaarde,*
3. *zijn eventuele precondities en*
4. *zijn aanleidingen tot eventuele excepties*

*gegeven.*

Het schrijven van een goede unittest begint dan met het kritisch lezen van de specificatie. Is hij wel duidelijk genoeg en is hij compleet? Zo neen, dan zullen, in overleg met de ontwerper van de klasse, eerst deze onnauwkeurigheden moeten worden verwijderd.

**stamboom**

In de cursus C2J31 hebben jullie kennisgemaakt met de stamboomapplicatie. Hieronder zie je het klassendiagram van een vorige versie (met overerving).



Voor deze applicatie was toendertijd geen unittest geschreven. We richten ons op de klasse Administratie (zie bijlage) en in het bijzonder op de methode addPersoon waarmee een persoon bij het Administratie-object wordt geregisteerd.

/\*\*

\* er wordt afhankelijk van het geslacht een man/vrouw gecreeerd met als

\* voornamen vnamen, achternaam anaam, tussenvoegsel tvoegsel, geboortedatum

\* gebdat en ouderlijke relatie ouders; van elke voornaam en achternaam zijn

\* zo nodig de eerste letter naar een hoofdletter en de andere letters naar

\* kleine letters getransformeerd; voor het tussenvoegsel zijn alle letters

\* zo nodig naar een kleine letter getransformeerd;

\* de persoon krijgt een uniek nummer toegewezen

\* deze persoon staat voortaan ook als kind bij ouders, mits bekend,

\* geregistreerd;

\* @param geslacht

\* @param vnamen alle voornamen zijn niet-lege strings en er is er ten

\* minste 1

\* @param anaam achternaam mag geen lege string zijn

\* @param tvoegsel mag een lege string zijn (dan is er geen tussenvoegsel)

\* @param gebdat

\* @param ouders mag null zijn indien ouders onbekend

\*/

public void addPersoon(Geslacht geslacht, String[] vnamen, String anaam,

String tvoegsel, GregorianCalendar gebdat, Relatie ouders) {

// TODO

}

In bovenstaande specificatie krijgen de meeste parameters een toelichting mee. De preconditie die er aan een parameter wordt opgelegd zul je normaliter nabij @param terugvinden. De parameters geslacht en gebdat hebben geen toelichting; hun preconditie wordt volledig afgedekt door het bijbehorende type, respectievelijk Geslacht en GregorianCalendar. We zeggen dat deze twee parameters geen (expliciete) preconditie hebben. Een preconditie is een eis. Soms bevat de parametertoelichting geen eisen maar een waarschuwing, hint of extra omschrijving.

Vraag: Voor welke parameters van addPersoon zijn er, strikt genomen, wel echte precondities geformuleerd?

**een werkwijze**

De werkwijze in deze beschouwing laat zich zoveel als mogelijk leiden door de specifieke documentatie van een methode/constructor, dus in dit geval door de documentatie van addPersoon.

Nadat we kritisch de hele specificatie van addPersoon hebben doorgenomen en we van mening zijn dat daar geen onduidelijkheden in staan, stappen we over op het aanpakken van de precondities. We stellen ons op het standpunt dat als een aanroeper van deze methode per ongeluk een preconditie overtreedt dat hij daar een melding van zou moeten krijgen. We spreken af dat de melding de vorm van een IllegalArgumentException krijgt. Daarna richten we ons op de primaire eigenschappen van de methode. Dit zijn dan de stappen die we zouden doorlopen:

1. Genereer binnen NetBeans een lege unit test (advies: maak gebruik van de optie New JUnit Test; de naam van de testklasse is gelijk aan de naam van de testen klasse gevolgd door Test; de naam van de package is stamboom)
2. Neem voor in principe elke publieke methode en constructor een testmethode op. Een test voor de eenvoudige get-methoden mag je achterwege laten omdat we aannemen dat deze methoden vanzelf bij de andere testmethoden aan bod komen.
3. Kopieer de documentatie van de methoden en constructoren naar de bodies van de gewenste testmethoden;
4. Neem voor het testen van elke exceptie (dus ook voor elke test van een preconditie) een try-catch of een aparte testmethode op;
5. Verplaats de precondities uit de documentatie naar de betreffende try catch of preconditietestmethode en de documentatie van de andere excepties naar de overeenkomstige try catch of exceptietestmethode;
6. Neem vervolgens de startpassage en de specificatie van de eventuele returnwaarde uit de documentatie (meteen achter het javadoc startsymbool /\*\*) op; probeer deze passage zoveel als mogelijk op te delen in losstaande uitspraken;
7. Bedenk welke objecten je nodig hebt om een uitspraak te testen; stel vast of het zinvol is deze initialisatie onder te brengen in de setUp-methode zodat andere testmethoden ook kunnen profiteren van deze initialisatiecode.
8. Verwijder de testmethoden waarvoor de testcode door andere testmethoden is afgedekt.

**aan de slag**

Na stap 2 hebben we met de volgende weergave te maken (tip: maak gebruik van de Netbeans-macro Toggle comment):

@Test

public void testAddPersoon() {

/\*\*

\* er wordt afhankelijk van het geslacht een man/vrouw gecreeerd met als

\* voornamen vnamen, achternaam anaam, tussenvoegsel tvoegsel, geboortedatum

\* gebdat en ouderlijke relatie ouders; van elke voornaam en achternaam zijn

\* zo nodig de eerste letter naar een hoofdletter en de andere letters naar

\* kleine letters getransformeerd; voor het tussenvoegsel zijn alle letters

\* zo nodig naar een kleine letter getransformeerd;

\* de persoon krijgt een uniek nummer toegewezen

\* deze persoon staat voortaan ook als kind bij ouders, mits bekend,

\* geregistreerd;

\* @param geslacht

\* @param vnamen alle voornamen zijn niet-lege strings en er is er ten

\* minste 1

\* @param anaam achternaam mag geen lege string zijn

\* @param tvoegsel mag een lege string zijn (dan is er geen tussenvoegsel)

\* @param gebdat

\* @param ouders mag null zijn indien ouders onbekend

\*/

}

Na de vierde stap hebben we dit commentaar enigszins gereorganiseerd. We hebben lege regels toegevoegd, ten teken van: hier moet testcode worden toegevoegd. Verder zijn er twee extra testmethoden als gevolg van twee precondities toegevoegd:

@Test

public void testAddPersoon() {

// /\*\*

// \* er wordt afhankelijk van het geslacht een man/vrouw gecreeerd met als

// \* voornamen vnamen, achternaam anaam, tussenvoegsel tvoegsel,

// \* geboortedatum gebdat en ouderlijke relatie ouders;

// \* van elke voornaam en achternaam zijn zo nodig de eerste letter

// \* naar een hoofdletter en de andere letters naar kleine letters

// \* getransformeerd; voor het tussenvoegsel zijn alle letters

// \* zo nodig naar een kleine letter getransformeerd;

// \* de persoon krijgt een uniek nummer toegewezen

// \* deze persoon staat voortaan ook als kind bij ouders, mits bekend,

// \* geregistreerd;

// \* @param tvoegsel mag een lege string zijn

// \* @param ouders mag null zijn indien ouders onbekend

}

@Test(expected = IllegalArgumentException.class)

public void testVnamen() {

// \* @param vnamen alle voornamen zijn niet-lege strings en er is er ten

// \* minste 1

}

@Test(expected = IllegalArgumentException.class)

public void testAnaam() {

// \* @param anaam mag geen lege string zijn

}

In verband met het registreren van een persoon met een gegeven ouderlijke relatie, is er de keuze gemaakt om in de setUp-methode een relatie-object te creëren die als ouder kan functioneren. Uiteraard heb je daar een administratie voor nodig. Merk op dat er binnen de setUp-methode geen assertions plaatsvinden.

private Administratie administratie;

private Relatie ouders;

private Man peter;

private Vrouw miep;

@Before

public void setUp() {

administratie = new Administratie();

administratie.addPersoon(Geslacht.MAN, new String[]{"Peter", "Jan"},

"Janssen","", new GregorianCalendar(1948, 1, 20), null);

peter = (Man)administratie.getPersoon(1);

administratie.addPersoon(Geslacht.VROUW, new String[]{"Miep"}, "Jeuken",

"van der", new GregorianCalendar(1950, 3, 22), null);

miep = (Vrouw)administratie.getPersoon(2);

administratie.addRelatie(miep,peter);

ouders = administratie.getRelatie(1);

}

Dit wordt dan de testcode van addPersoon. Merk op dat de eerste twee condities m.b.v. de aanroep van één aanroep van addPersoon worden getest.

@Test

public void testAddPersoon() {

// /\*\*

// \* er wordt afhankelijk van het geslacht een man/vrouw gecreeerd met als

// \* voornamen vnamen, achternaam anaam, tussenvoegsel tvoegsel, geboortedatum

// \* gebdat en ouderlijke relatie ouders;

// \* van elke voornaam en achternaam zijn

// \* zo nodig de eerste letter naar een hoofdletter en de andere letters naar

// \* kleine letters getransformeerd; voor het tussenvoegsel zijn alle letters

// \* zo nodig naar een kleine letter getransformeerd;

// nummering van maanden begint bij 0 binnen GregorianCalendar

administratie.addPersoon(Geslacht.MAN, new String[]{"wim", "FRANciscus",

"Petrus"}, "vELD", "OP HET", new GregorianCalendar(1978, 11, 22), ouders);

Persoon wim = administratie.getPersoon(3);

Assert.assertEquals("geslacht", wim.getGeslacht(), Geslacht.MAN);

Assert.assertEquals("voornamen", wim.getVoornamen(),

"Wim Franciscus Petrus");

Assert.assertEquals("achternaam", wim.getAchternaam(), "Veld");

Assert.assertEquals("tussenvoegsel", wim.getTussenvoegsel(), "op het");

Assert.assertEquals("geboortedatum", wim.getGebdat(), "22-12-1978");

Assert.assertEquals("ouders", wim.getOuders(), ouders);

// \* de persoon krijgt een uniek nummer toegewezen

System.out.println("uniciteit van persoonsnummer valt moeilijk t.o.v. "

+ "een black box te testen; advies: inspecteer programmacode");

// \* deze persoon staat voortaan ook als kind bij ouders, mits bekend,

// \* geregistreerd;

ArrayList<Persoon> kids = new ArrayList<Persoon>();

Iterator<Persoon> it = ouders.getKinderen();

while (it.hasNext()) { kids.add(it.next()); }

Assert.assertEquals("ouders hebben kind erbij", kids.contains(wim));

// \* @param tvoegsel mag een lege string zijn

administratie.addPersoon(Geslacht.MAN, new String[]{"wim", "FRANciscus",

"Petrus"}, "vELD", "", new GregorianCalendar(1978, 11, 22), ouders);

Persoon wim2 = administratie.getPersoon(4);

Assert.assertEquals("tussenvoegsel", wim2.getTussenvoegsel(), "");

// \* @param ouders mag null zijn indien ouders onbekend

administratie.addPersoon(Geslacht.VROUW, new String[]{"Maria"}, "Onbekend",

"", new GregorianCalendar(-23, 1, 2), null);

Assert.assertNotNull("ouders onbekend", administratie.getPersoon(5));

Assert.assertNull(("ouders null", administratie.getPersoon(5).getOuders()); }

Dan resteren nog de twee testmethoden i.v.m. de precondities. Na ampele overweging is besloten om het testen van de preconditie m.b.t. vnamen in twee aparte methoden te splitsen.

@Test(expected = IllegalArgumentException.class)

public void testGeenVoornaam1() {

// \* @param vnamen er is er ten minste 1

administratie.addPersoon(Geslacht.MAN, new String[]{}, "achternaam",

"", new GregorianCalendar(2000, 0, 1), ouders);

Assert.fail("ten minste 1 voornaam");

}

@Test(expected = IllegalArgumentException.class)

public void testGeenVoornaam2() {

// \* @param vnamen alle voornamen zijn niet-lege strings

administratie.addPersoon(Geslacht.MAN, new String[]{"Pieternel",""},

"achternaam","", new GregorianCalendar(2000, 0, 1), ouders);

Assert.fail("elke voornaam moet een niet-lege string zijn");

}

@Test(expected = IllegalArgumentException.class)

public void testAchternaam() {

// \* @param anaam mag geen lege string zijn

administratie.addPersoon(Geslacht.MAN, new String[]{"voornaam"}, "",

"", new GregorianCalendar(2000, 0, 1), ouders);

Assert.fail("achternaam moet een niet-lege string zijn");

}

bijlage

package stamboom;

import java.util.\*;

public class Administratie {

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*datavelden\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

private ArrayList<Persoon> personen;

private int nextRelNr;

private int nextPersNr;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*constructoren\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* er wordt een administratie gecreeerd met 0 personen en dus 0 relaties

\* de eerste persoon en eerste relatie die er in de toekomst zal

\* worden gecreeerd krijgt nummer 1

\*/

public Administratie() {

// TODO

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*methoden\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* er wordt afhankelijk van het geslacht een man/vrouw gecreeerd met als

\* voornamen vnamen, achternaam anaam, tussenvoegsel tvoegsel, geboortedatum

\* gebdat en ouderlijke relatie ouders; van elke voornaam en achternaam zijn

\* zo nodig de eerste letter naar een hoofdletter en de andere letters naar

\* kleine letters getransformeerd; voor het tussenvoegsel zijn alle letters

\* zo nodig naar een kleine letter getransformeerd; de persoon krijgt een uniek

\* nummer toegewezen; deze persoon staat voortaan ook als kind bij ouders,

\* mits bekend, geregistreerd;

\* @param geslacht

\* @param vnamen alle voornamen zijn niet-lege strings en er is er ten

\* minste 1

\* @param anaam achternaam mag geen lege string zijn

\* @param tvoegsel mag een lege string zijn (dan is er geen tussenvoegsel)

\* @param gebdat

\* @param ouders mag null zijn indien ouders onbekend

\*/

public void addPersoon(Geslacht geslacht, String[] vnamen, String anaam,

String tvoegsel, GregorianCalendar gebdat, Relatie ouders) {

// TODO

}

/\*\*

\* er wordt een (kinderloze) relatie tussen m1 en m2 gecreeerd; de start-

\* (en scheidings)datum is onbekend;

\* de relatie krijgt een uniek nummer toegewezen

\*/

public void addRelatie(Persoon m1, Persoon m2) {

// TODO

}

/\*\*

\*

\* @param nr

\* @return de persoon met nummer nr, als die niet bekend is wordt er null

\* geretourneerd

\*/

public Persoon getPersoon(int nr) {

// TODO

return null;

}

/\*\*

\* @param achternaam is geen lege string;

\* @return alle personen met een achternaam gelijk aan de meegegeven achternaam

\* (er is geen verschil tussen hoofd- en kleine letters)

\*/

public ArrayList<Persoon> getPersonen(String achternaam) {

// TODO

return null;

}

/\*\*

\*

\* @param relNr

\* @return de relatie met nummer nr, als die niet bekend is wordt er null

\* geretourneerd

\*/

public Relatie getRelatie(int relNr) {

// TODO

return null;

}

}