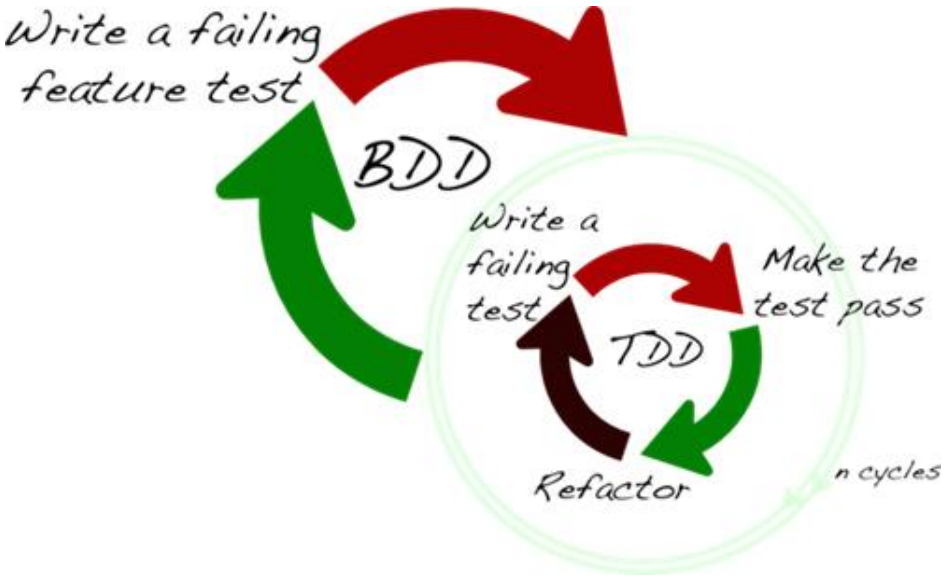
Introductie Cucumber

Iedere software professional is wel bekend met Test Driven Development (TDD). Dit is een binnen SCRUM vaak geziene werkwijze waarbij een team eerst testen schrijft voordat wordt begonnen met implementatie van functionaliteit. Aanvankelijk zullen alle testen falen omdat de functionaliteit nog niet gerealiseerd is. Gaandeweg de iteratie zullen er echter steeds meer testen slagen en aan het einde van de iteratie is het team klaar wanneer alle testen slagen. TDD testen worden vaak gerealiseerd op unit- of systeem-niveau.

Behaviour Driven Development (BDD) tilt dit process naar een hoger niveau. Ook hier worden op voorhand testen geschreven en is men klaar wanneer alle testen slagen. Echter, BDD beschouwt het gedrag van het hele systeem (eventueel in de keten) en niet het gedrag van een systeemcomponent, zoals het geval is bij TDD. Door het specificeren van systeemgedrag op het hoogste niveau zijn alle stakeholders (ontwikkelaars, testers, product owners, architecten, etc) in staat om mee te praten over de testgevallen. BDD ligt in het verlengde van Acceptance Test Driven Development (ATDD) en Specification by Example (SbE).



Dan North (vaak gezien als de grondlegger van BDD) beschrijft BDD middels de volgende principes:

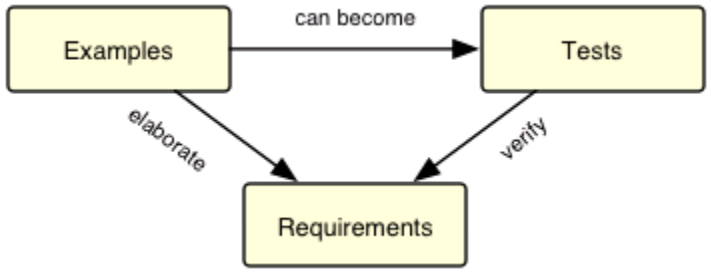
- Enough is enough: doe zoveel aan planning, analyse en ontwerp als noodzakelijk is, maar sla niet door. Ga dus niet nodeloos alles vooraf bedenken maar doe dit op het juiste moment.

- Deliver stakeholder value: alles wat je doet moet waarde toevoegen voor de stakeholders of het vermogen om dit te doen vergroten.
- It's a behavior: iedereen die betrokken is bij het project/systeem moet op dezelfde manier mee kunnen praten over wat het systeem doet. Testgevallen in begrijpbare taal vergroten het gedeelde begrip.

Cucumber is een doorontwikkeling van RSpec, een testframework om deploymentscripts mee te testen. Hierdoor is het out-of-the-box toepasbaar op Ruby software. Door de jaren heen is het framework echter aangepast om het op veel andere talen

Features en Scenarios

Een erg groot voordeel van Cucumber is dat de testgevallen worden uitgedrukt in natuurlijke taal. Om tot deze testgevallen te komen moedigt BDD aan om (functionele of technische-) requirements uit te werken tot concrete voorbeelden. Door dit met verschillende stakeholders te doen (bijvoorbeeld een product-owner, tester en ontwikkelaar) worden de requirements als het ware 'geconcretiseerd'. Bijkomend voordeel is dat randgevallen en uitzonderingssituaties expliciet worden. Door vastlegging in natuurlijke taal wordt voorkomen dat alleen technisch-georienteerde stakeholders zich in de discussie kunnen mengen. Ook zijn de overeengekomen voorbeelden direct bruikbaar als testgevallen. Onderstaande afbeelding geeft de afhankelijkheden weer:



Cucumber groepeert systeemfunctionaliteit in zgn. features. Een feature is niets meer dan een verzameling gerelateerd systeemgedrag. Een feature is op zijn beurt weer opgebouwd uit scenarios. Ieder scenario omschrijft een voorbeeld van systeemgedrag (eventueel geparameteriseerd) en is in wezen een testgeval. Features worden gespecificeerd in de taal Gherkin. Optioneel kunnen features ook worden voorzien van een uitgangssituatie (background/achtergrond-clausules) die de gewenste staat van het systeem uitdrukken.

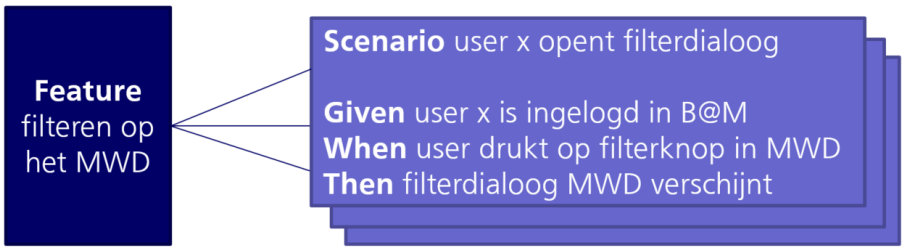
Een scenario is opgebouwd uit een aantal stappen:

- **Given**: Drukt een preconditie uit voor het scenario, bijvoorbeeld een gebruiker is ingelogd of een bepaalde entiteit is aanwezig in het systeem.
- **When**: Drukt een actie uit, bijvoorbeeld een gebruiker opent een dialoog of voert een bewerking uit.

- **Then**: Drukt een postconditie uit, bijvoorbeeld de gebruiker ziet een dialoog of output op het scherm of in een randsysteem verschijnt.

Een scenario is zeker niet beperkt door de drie bovengenoemde stappen. Om de leesbaarheid te bevorderen kunnen meerdere stappen worden opgegeven middels de keywords **And** en **But**. In het Nederlands worden de stappen respectievelijk uitgedrukt met **Gegeven, Als, Dan, En** en **Maar**.

Hieronder staat een voorbeeld van een feature met daarin een scenario waarin een gebruiker een filterdialoog opent.



Een uitgebreidere feature zou er als volgt uit kunnen zien:

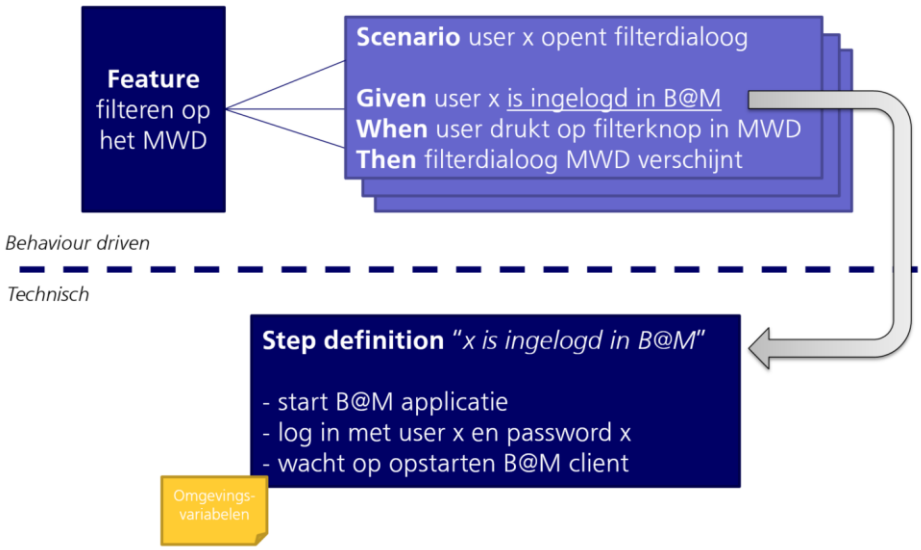
Voorbeeld Cucumber feature

```
#language: nl
@KT
Functionaliteit: KT3 - ProRail BAD

  Achtergrond:
    Gegeven de ESB verstuurt drglmutatieberichten naar BAD
    En de ESB verstuurt ua-berichten naar BAD

  Scenario: KT8 Trein met drgl inleggen
    Gegeven datum 08-06-2016 valt binnen de huidige werkperiode
    Als ProRail drglmutatiebericht KT8.xml verstuurt naar de ESB
    En 2 seconden wacht
    Dan ontvangt BAD een drglmutatiebericht met element
"_5:OperationalTrainNumber" en waarde "15"
    En bevat BAD een ingelegde trein 15 met datum 08-06-2016
    En bevat BAM een ingelegde trein 15 met datum 08-06-2016
```

Step definitions

Features en scenarios drukken het gedrag van het systeem uit in natuurlijke taal. De stappen die hierin worden gebruikt moeten nog worden vertaald naar concrete systeemacties. Dit gebeurt door step definitions in het Cucumber framework. Step definitions zijn stukjes glue-code die d.m.v. pattern-matching worden gekoppeld aan scenario-stappen. Het zijn dus stukje code die Cucumber aanroept bij het uitvoeren van een testgeval en maken de vertaling van behavior-driven gedrag naar technische acties. Hieronder een grafisch voorbeeld van een stap die wordt

vertaald naar enkele technische acties. Door omgevingsvariabelen te parameteriseren kan een testgeval door het framework worden uitgevoerd in verschillende omgevingen.



Een concreet voorbeeld behorende bij het feature hierboven zou er als volgt uit kunnen zien:

Voorbeeld step definition Java7

```
@Als("^ProRail drglmutatiebericht ([^\"]*) verstuurt naar de ESB$")
public void prorail_drglmutatiebericht_verstuurt_naar_de_ESB(String
bestandsnaam) throws Throwable {
  int responsecode = esbTSITAFDrglPublisher.publishFileMsg(bestandsnaam);
  assertThat(responsecode, is(200));
}
```

Dankzij de annotatie aan het begin kan Cucumber de functie koppelen aan een stap in een scenario. Daarbij herkent Cucumber automatisch een megegeven parameter als bestandsnaam en publiceert het framework een bericht. Door midddel van een assertie wordt gecontroleerd dat het bericht correct is verstuurd.

Hetzelfde voorbeeld kan vanaf Java 8 ook met behulp van een Lambda functie omschreven worden:

Voorbeeld step definition Java8

```
Als("^ProRail drglmutatiebericht '(.*)' verstuurt naar de ESB$", (String
bestandsnaam) -> {
  int responsecode = esbTSITAFDrglPublisher.publishFileMsg(bestandsnaam);
  assertThat(responsecode, is(200));
});
```

Feature layout

Cucumber testbestanden worden features genoemd, de extensie van deze bestanden is dan ook ".feature".

Tags

Tags geven de mogelijkheid om testen uit te voeren voor een specifieke tag. Tags kunnen worden toegekend aan de feature of aan specifieke scenario's. Iedere tag die op feature niveau is toegekend zal ook aan de onderliggende scenario's worden toegekend.

De volgende kunnen bijvoorbeeld worden gebruikt

Tag	**Doel**
@ignore	Schakel hiermee een scenario of feature uit
@runonly	Een tijdelijke tag voor het runnen van een kleinere selectie scenario's
@smoke	Onderdeel van de smoketest
@integratie	Integratie testen

Het is niet toegestaan om spaties in een tag te gebruiken, daarnaast zijn tags hoofdletter gevoelig.

Functionaliteit

De functionaliteit van het feature bestand bestaat uit 3 onderdelen: functionaliteit:-sleutelwoord, *naam* van feature (op dezelfde regel) en optioneel (maar zeer aan te raden) een *omschrijving* van de functionaliteit.

Cucumber doet in principe niets met de naam of beschrijving van de functionaliteit, het is echter dé plek om informatie voor de gebruiker om belangrijke informatie te definiëren.

Voorbeeld:

```
Functionaliteit: Retourneer item
  Een verkoper moet in staat zijn om een klant zijn geld terug te geven.
  Dit is een wettelijke verplichting en maakt de klant ook blij.

  Regels:
  - Klant moet bewijs van aankoop tonen
  - Aankoop moet minder dan 30 dagen geleden zijn
```

Achtergrond

In sommige gevallen kan het zijn dat dezelfde Stel stappen in ieder scenario herhaald worden. Omdat deze stappen niet essentieel zijn voor het beschrijven van het scenario, maar meer incidentele details, kunnen deze stappen naar de Achtergrond geplaatst worden.

```
Achtergrond:
  Stel een magnetron van €100 was verkocht op 03-11-2015
  En vandaag is het 18-11-2015
```

Scenario

Een scenario beschrijft een concreet voorbeeld dat een business regel illustreert. Het bestaat uit een lijst van stappen (zie 4.3.6). Deze lijst van stappen kan zo lang worden als nodig is, maar het advies is om deze klein te houden.

Scenario’s beschrijven telkens hetzelfde patroon:

- Optionele omschrijving van scenario of extra commentaar naast titel
- Omschrijf initiële context (Stel)
- Omschrijf een actie/event (Als)
- Omschrijf de verwachte uitkomst (Dan)

```
Scenario: het voeden van een kalf
  Stel een koe weegt 400 kg
  Als we de voedingseisen berekenen
  Dan moet de energie 26500 MJ bedragen
    En moet er 215 kg eiwit zijn
```

Scenario met extra omschrijving/commentaar:

```
Scenario: het voeden van een kalf
  Berekening van de hoeveelheid energie en eiwitten

  Stel een koe weegt 400 kg
  Als we de voedingseisen berekenen
  Dan moet de energie 26500 MJ bedragen
    En moet er 215 kg eiwit zijn
```

Abstract Scenario

Wanneer er complexe business regels van toepassing zijn, kan het voorkomen dat er meerdere scenario's geschreven worden waarbij enkel waardes verschillen. In dat geval is een abstract scenario een beter alternatief.

```
Abstract scenario: het voeden van een kalf
  Stel een koe weegt <gewicht> kg
  Als we de voedingseisen berekenen
  Dan moet de energie <energie> MJ bedragen
  En moet er <eiwit> kg eiwit zijn

  Voorbeelden:
    | gewicht | energie | eiwit |
    |    450  |  26500  |   215 |
    |    500  |  29500  |   245 |
```

Voorbeelden

Een abstract scenario wordt altijd gevolgd met één of meerdere voorbeelden secties. De tabel onder de voorbeelden moet als kolom titel de variabele in het abstracte scenario gebruiken.

Iedere regel in de tabel wordt uitgevoerd in het scenario met die waarden. In het voorbeeld hierboven zal het scenario dus twee maal worden uitgevoerd.

Stappen

Een stap begint meestal met stel, als of dan. Als er meerdere stel, als of dan stappen onder elkaar komen, dan kan er gebruikt worden gemaakt van en of maar. Cucumber maakt geen onderscheid tussen de zoekwoorden, maar het kiezen van de juiste is belangrijk voor de leesbaarheid van het scenario in zijn geheel.

Stel

Stel stappen worden gebruikt om de initiële context van het systeem beschrijven – de scene van het scenario. Het is typisch iets dat in het verleden gebeurde.

Wanneer Cucumber een stel stap uitvoert zal het systeem in een goed gedefinieerde toestand worden geconfigureerd, zoals het creëren en configureren van voorwerpen of het toevoegen van gegevens aan de testdatabase.

Het is ok om meerdere stel stappen te hebben (advies is om gebruik te maken van en of maar voor de tweede en later om het beter leesbaar te maken).

Als

Als stappen worden gebruikt om een gebeurtenis of een actie te beschrijven. Dit kan interactie van een persoon met het systeem zijn, of het kan een gebeurtenis vanuit een ander systeem zijn.

Het wordt sterk aanbevolen slechts één enkele Als stap per scenario te gebruiken. Indien er meerdere als stappen nodig zijn dan kan het een teken zijn dat het scenario gesplitst moet.

Dan

Dan stappen worden gebruikt om een verwachte uitkomst of resultaat te beschrijven.

De stap definitie van een Dan stap moet een bewering gebruiken om de werkelijke uitkomsten (wat het systeem eigenlijk doet) naar het verwachte resultaat (wat de stap zegt dat het systeem moet doen) te vergelijken.

En / Maar

Zoals hierboven als beschreven is kan een scenario meerdere malen een stap sleutelwoord gebruiken. Het is in dat geval aan te raden om gebruik te maken van en of maar voor de tweede en later om het beter leesbaar te maken. Hierbij kan de maar gebruikt worden als tegenstelling.

Stap argumenten

In sommige gevallen kan het nodig zijn om een groter stuk tekst of een tabel met gegevens door te geven aan een stap dat niet past op een enkele lijn.

Hiervoor zijn twee opties mogelijk: Doc Strings en gegevens tabellen (Data Tables).

Doc String

Doc Strings zijn handig voor het doorgeven van een groter stuk tekst naar een stap definitie. De syntax is geïnspireerd op Python's Docstring[1] syntax.

De tekst moet worden afgebakend met drie dubbele aanhalingstekens op een eigen regel:

```
Stel een blog bericht genaamd "Random" met opgemaakte tekst
  """
  Some Title, Eh?
  ===============
  Here is the first paragraph of my blog post. Lorem ipsum dolor sit amet,
  consectetur adipiscing elit.
  """
```

Het inspringen van de opening """ is onbelangrijk , hoewel het gebruikelijk is twee spaties van de omsluitende stap . Het inspringen binnen de drievoudige quotes is echter wel significant. Elke regel van het Doc String wordt teruggesprongen volgens de opening """. Inspringen voorbij de kolom van de opening """ wordt dus behouden.

Gegevens tabellen

Gegevens tabellen zijn handig voor het doorgeven van een lijst met waarden aan een stap definitie:

```
Stel de volgende gebruikers bestaan:
  | name   | email              | twitter         |
  | Aslak  | aslak@cucumber.io  | @aslak_hellesoy |
  | Julien | julien@cucumber.io | @jbpros         |
  | Matt   | matt@cucumber.io   | @mattwynne      |
```

Net als Doc Strings, zullen gegevens tabellen worden doorgegeven aan de stap definitie in Java als laatste argument.

Het type van dit argument zal DataTable zijn. Zie de API-documentatie voor meer informatie over hoe de rijen en cellen te benaderen zijn. Zie ook paragraaf 4.4 voor wat extra handigheidjes.

Commentaar

Gherkin biedt tal van plekken om functionaliteiten en scenario's te documenteren. De meest ideale plek is in de beschrijvingen.

Mocht het toch nodig zijn extra commentaar toe te voegen net voor een scenario, bijvoorbeeld met een spoorbezettingstekening, dan kan dit door gebruikt te maken van het commentaar symbool; #
Het # vertelt Cucumber dat de rest van de regel commentaar is en dat deze niet mag worden uitgevoerd.

```
# Onderstaande gebruikers zijn nodig als initiële vulling zodat we er acties
# op kunnen uitvoeren
Stel de volgende gebruikers bestaan:
  | name   | email              | twitter         |
```

Cucumber Anti-Patterns

Interessant en leerzaam artikel over de valkuilen bij het schrijven van feature files:
http://www.thinkcode.se/blog/2016/06/22/cucumber-antipatterns

---

Termen en vertalingen

Binnen TOPAAS wordt gebruik gemaakt van de Nederlandse document structuur van Cucumber. Echter om meer informatie op het internet te vinden is het handiger om de Engelse termen te gebruiken. Hieronder staan alle vertalingen van de gebruikte Cucumber termen.

Nederlands	Engels
Abstract scenario	Scenario outline
Achtergrond	Background
Als	When
Commentaar	Comment
Dan	Then
Doc String	Doc String
En	And
Functionaliteit	Feature
Gegeven	Given
Gegevens tabel	Data Table
Maar	But
Scenario	Scenario
Stel	Given
Voorbeelden	Examples

Parameters in stappen

In cucumber is het gebruik van parameters in je stappen een veel gebruikte feature. Het opstellen van de juiste reguliere expressie in de step is daarmee een must. Cucumber kan automatisch al een voorstel doen, maar soms wil je net iets meer kunnen of net iets slimmere afhandelingen gebruiken. Kijk overigens voor de technische afhandeling van de step annotatie ook eens naar Transformers in Steps.

Simpele voorbeelden

Hieronder staan een paar snelle voorbeelden, lees verder voor de theorie achter deze voorbeelden.

#	Omschrijving	Code	Voorbeeld
1	Een variabele stukje tekst (String) tussen aanhalingstekens. De variable mag daarbij alles zijn behalve een aanhalingsteken	`([^\"]*)`	`Als ik naar "([^\"]*)" loop`
2	Meer mogelijkheden binnen een zin (waarvan er wel één gekozen moet worden)	`(optie1|optie2|optie3|etc)`	`Ik ga naar (buiten|binnen|school|werk|etc)`
3	Getallen	`(\\d+)`	`Ik ga om (\\d+) uur naar huis`

Reguliere expressies

Een reguliere expressie (kort RegEx) omschrijft een verzameling tekenreeksen (strings) zonder ze allemaal op te noemen. De drie strings *Handel*, *Händel* en *Haendel* kunnen bijvoorbeeld beschreven worden met het patroon “H(a|ä|ae)ndel”.

Gewone letters en cijfers in de reguliere expressie herkennen hetzelfde teken in de te vinden tekenreeks. Een aantal tekens hebben speciale betekenis:

- Een punt (**.**) staat voor een willekeurig teken, behalve het teken voor een newline (**\n**).
- Vierkante haken geven een lijst van mogelijke tekens: **[**abc**]**.

    - Binnen vierkante haken staat een minteken voor een reeks: **[**a-zA-Z**]** is het patroon waarmee alle letters “gevangen” worden.
    - Een dakje als eerste teken binnen de vierkante haken verandert de tekenverzameling in het omgekeerde: **[**^0-9**]** herkent alles wat geen cijfer is.
- Een dakje ^ staat voor het begin van de regel.
- Een dollarteken **$** staat voor het eind van de regel.

Deze basiselementen kunnen worden gecombineerd met de volgende constructies:

Keuze

Een verticale balk scheidt de alternatieven, bijvoorbeeld “groen|rood” herkent “groen” of “rood”.

Kwantificatie

Een kwantor achter een teken geeft aan hoe vaak dat teken voor mag komen. De meest voorkomende kwantoren zijn +, ? en *:

Kwantor	**Uitleg**
+	Een plusteken geeft aan dat het voorafgaande teken ten minste één keer moet voorkomen, bijvoorbeeld “goo+gle” herkent google, gooogle, goooogle, enz.
?	Een vraagteken geeft aan dat het voorafgaande teken ten hoogste één keer mag voorkomen, bijvoorbeeld “De Bruij?n” herkent “De Bruin” en “De Bruijn”.
*	Een sterretje geeft aan dat het voorafgaande teken nul of meer keer mag voorkomen, bijvoorbeeld “0*42” herkent 42, 042, 0042, enzovoort. Het gebruik van de veelvoorkomende constructie .* (deze vind alle tekst) is een afrader in Cucumber

Groepering

Haakjes maken een eenheid van het patroon waar ze omheen staan, bijvoorbeeld “(va|moe)der” is hetzelfde als “vader|moeder” en “(groot)?vader” herkent zowel “vader” als “grootvader”.

Deze constructies kunnen worden gecombineerd in hetzelfde patroon, zodat “H(ae?|ä)ndel” hetzelfde is als “H(a|ae|ä)ndel”.

Capturing versus non-capturing group

Meestal zal een reguliere expressie in Cucumber gebruikt worden om een variabele in de step op te nemen voor verdere afhandeling (de capturing group). Echter in sommige gevallen kan de variabele enkel voor tekstuele doeleinden zijn (de non-capturing group).

Groepering wordt aangegeven met de haakjes, bijvoorbeeld (abc). Een non capturing group heeft daarbij nog een vraagteken en dubbele punt, bijvoorbeeld (?:abc).

```
@Stel("^Ik wil gebruik maken van \"([^\"]*)\"$");
public void stelTest1(String var1){
 /* implementatie met 1 variabele, maar accepteerd alleen:
    Ik wil gebruik maken van "test"
 */
}
@Stel("^(Ik|hij) wil gebruik maken van \"([^\"]*)\"$");
public void stelTest1(String ikHij, String var1){
 /* implementatie met 2 variabelen, maar accepteerd zowel:
    Ik wil gebruik maken van "test"
    Hij wil gebruik maken van "test"
 */
}

@Stel("^(?:Ik|Hij) wil gebruik maken van \"([^\"]*)\"$");
public void stelTest1(String var1){
 /* implementatie met 1 variabele, maar accepteerd zowel:
    Ik wil gebruik maken van "test"
    Hij wil gebruik maken van "test"
 */
}
```

Het escape karakter: Slash (\ )

In Java is de slash (\) een escape karakter, dus in voorbeeld 1 willen we zoeken op een quote in plaats van de string afsluiten. Dit kan door de quote voor te gaan met een slash. Onderstaand staan vier voorbeelden waar door de highlighting van Java het verschil zichtbaar is.

```
String test1 = "Als ik naar "ASD" reis"; /* syntax error */
String test2 = "Als ik naar \"ASD\" reis"; /* wordt weergegeven zonder
quotes: Als ik naar "ASD" reis */
```

```
String locatie = "ASD";
String test3 = "Als ik naar " + locatie + " reis"; /* wordt weergegeven
zonder quotes: Als ik naar ASD reis */
String test4 = "Als ik naar \"" + locatie + "\" reis"; /* wordt weergegeven
zonder quotes: Als ik naar "ASD" reis */
```

ASD is in het geval van test1 geen onderdeel van de String en zal zelfs een syntax foutmelding geven. Test2 omvat de gehele string zoals we in voorbeeld in wilde hebben. Test3 is een voorbeeld in het gebruik van een variabele waarde (in dit geval ook een String). Deze zin zal worden weergegeven zonder quotes rondom de waarde van de locatie variabele. Het plus karakter is in dit geval de lijm tussen de verschillende strings. Test4 geeft de string weer aan met quotes, let hierbij op dat de eerst quote wordt ge-escaped en de tweede quote niet.

ASD is in dit geval natuurlijk een vaste string, maar kan natuurlijk ook worden vervangen door een reguliere expressie, bijvoorbeeld [A-Z]+ (enkel hoofdletters en minstens 1 karakter lang).

RegEx valideren

Om reguliere expressies te testen zijn er verscheidene tools op internet beschikbaar. Een erg handige is http://regexr.com/ deze heeft namelijk ook cheatsheets, deze staat ook op Reguliere expressie Cheatsheet

Let op dat de extra escape karakters verwijderd zijn, dus \\ zou niet meer voor moeten komen (behalve als je zoekt op een slash).

Bronnen:

https://nl.wikipedia.org/wiki/Reguliere_expressie

http://regexr.com/

Reguliere expressie Cheatsheet

Als in onderstaande tabel een slash (\) wordt gebruikt dan moet deze in java voorgegaan worden door het escape karakter.

Character classes	
.	any character except newline
\w \d \s	word, digit, whitespace
\W \D \S	not word, digit, whitespace
[abc]	any of a, b, or c
[^abc]	not a, b, or c
[a-g]	character between a & g
Anchors	
^abc$	start / end of the string
\b \B	word, not-word boundary
Escaped characters	
\. * \\	escaped special characters
\t \n \r	tab, linefeed, carriage return
\u00A9	unicode escaped ©
Groups & Lookaround	
(abc)	capture group
\1	backreference to group #1
(?:abc)	non-capturing group
(?=abc)	positive lookahead
(?!abc)	negative lookahead
Quantifiers & Alternation	
a* a+ a?	0 or more, 1 or more, 0 or 1
a{5} a{2,}	exactly five, two or more
a{1,3}	between one & three
a+? a{2,}?	match as few as possible

ab|cd	match ab or cd