Test Driven Development – Cheat Sheet

# JUnit

In de klasse:

* De te testen klasse als privaat attribuut
* Test Cases, aangeduid met @Test voor de methode.

In de test case:

* Arrange 🡪 combineren met @Before en constructor
* Act
* Assert 🡪 niet te veel asserts in één test case.

De @Test methodes moeten in willekeurige volgorde uitgeoefend kunnen worden.

Wat testen:

* Grenswaarden uit de realiteit
* Niet toegelaten waarden uit de realiteit
* Niet toegelaten waarden door techniek (vb null)
* Verzamelingen:
  + Normale verzameling
  + Verzameling met grenswaarden
  + Verzamelingen met 1 element
  + Lege verzamelingen
  + Null
* Strings:
  + Normale waarde
  + Lege String
  + Null
  + Te lange waarde/Te korte waarde
  + Strings met vreemde tekens
* Exceptions 🡪 @Test (expected = Exception.class)

Test suites runnen verschillende testklasses

* Voor testklasse:
  + @RunWith(Suite.class)   
    @Suite.SuiteClasses( {testen.Test1.class, testen.Test2.class })

Parameterized Test:

* Voor testklasse:
  + @RunWith(value=Parameterized.class)
* In de testklasse:
  + public static Collection<Object[]> getTestParameters() {  
     return Arrays.asList(new Object[][]{{"obj1", "obj2"}, {"obj1", "obj2"}});  
    }
  + public TestKlasse( Object ob1, Obj ob2){  
     // toewijzen van ob1 en ob2 aan private attributen  
    }

# Mock

1. De werkelijke implementatie, BImpl en de dummy, BDummy, implementeren eenzelfde interface I.
2. De dependeny wordt vastgelegd, in A, met een private variable van het I.
3. In A wordt:
   1. Een constructor met een parameter van het type I voorzien.
   2. Een set methode met een parameter van het type I voorzien.
4. Bij het uitvoeren wordt BImpl meegegeven.
5. Bij het testen wordt BDummy meegegeven.

In de dummy worden bij de methodes dan de gewenste antwoorden geretourneerd.

# Mockito

1. Maak alles klaar voor dependency injection. (Zie Mock stap 2-3)
2. Maak een private attribuut aan van het te mocken object en plaats er de annotatie @Mock voor.
3. Voeg in de @Before methode het volgende toe:  
    MockitoAnnotations.initMocks(this);  
    Injecteer de dummy
4. Train de dummy in de @Test methode  
    Mockito.when(dummymethode).thenReturn(gewensteResultaat);
5. Voer de test uit
6. Voeg na de asserts het volgende toe om te controlleren of de dummymethode werd opgeroepen:  
    Mockito.verify(gemocktAttribuut).gemockteMethode();