Test Driven Development

# Voordelen TDD

* Testen schrijven voor coderen brengt meer inzicht in wat code moet doen
* Laat het design toe om te evolueren en aan te passen in het zicht op het probleem
* Korte feedback loop
* Creëert een gedetailleerde specificatie
* Verminderd werk in herschrijven
* Minder tijd in debugging en een probleem sneller onderscheppen

# TDD workflow

* Klassendiagram maken
* Testen schrijven
* Testen uitvoeren
* Schrijf code om testen te doen slagen, testen opnieuw uitvoeren en code aanpassen

Optional operation: bepaalde methode met een body die een error gooit, zodanig dat je de methode niet hoeft te implementeren.

! Arrays.asList : array wordt omringd door wrapper, maar de array blijft dus een array. Die heeft dan de optional methodes add, get, … omdat die niet werken op de lijst dat eigenlijk een array is.

# Test case

Een test case is een klasse die test bevat. Elke methode voorafgegaan met @Test wordt aanzien als test methode.

# Static methode in klasse assertions

* assertEquals(expected, actual): gelijk aan elkaar?
* assertEquals(expected, actual, delta): gelijk aan elkaar, maar met delta als maximaal verschil. Alle waarden uitgedrukt in double of float.
* assertTrue/false
* assertNull
* assertThrows: Assertions.assertThrows(IllegalArgumentException.class, () -> {...});
* assertDoesNotThrow

# Parameterized test

ValueSource: gewoon waardes doorgeven

CsvSource: set van data in string doorgeven, om meerdere sets door te geven.

NullAndEmptySource: null en leeg testen

MethodSource: opsomming hergebruiken voor meerdere parameterized tests. Kan een of meerder waarden bevatten. Kan objecten bevatten.  
Werkt met static methode: private static Stream<Arguments> naam(){ return Streams.off( Arguments.off(new int[]{5}, 5), …);}

Afbeelding met tekst

Automatisch gegenereerde beschrijving

# Dependencies

Als klasse A in een methode een methode oproept van klasse B, heeft klasse A een dependency op A

Aangeduid met stippellijn

Problemen

* als b nog niet bestaat of geïmplementeerd is, kan je niet testen.
* Als B user interface is, wacht je telkens op gebruikerinvoer, wat weer de test laat falen.

Oplossing: mockobject: object speciaal gemaakt om de eigenschappen en gedragingen te simuleren van 1 of meerdere objecten.

Mock object wordt door dependency injection in de code geplaatst.

Volgens strategy pattern

Implementatie kan met mockito, hoeft dan geen interface te gebruiken.

DataTransfer object: object dat naar UI gegeven wordt omdat het geen echt domein object is, Voorbeeld is een record