11. Labwerk MockBean met Mockito

## Inleiding

Dit voorbeeld toont hoe Mockito kan worden gebruikt om een mock-object van MyClass te maken en het verwachte gedrag van de sum-methode te definiëren. Vervolgens wordt een instantie van de Calculator-klasse gemaakt die de mock-object gebruikt om de berekening uit te voeren. Ten slotte wordt gecontroleerd of de verwachte uitkomst wordt geretourneerd.

Kortom, JUnit wordt gebruikt om testcases uit te voeren en fouten te rapporteren, terwijl Mockito wordt gebruikt om mock-objecten te maken en het gedrag van die objecten te definiëren voor gebruik in tests.

## Maak een nieuw Maven-project

Maak een nieuw Maven-project in uw favoriete IDE en voeg de volgende afhankelijkheden toe aan het pom.xml-bestand:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <version>2.6.2</version>  
 <scope>test</scope>  
</dependency>  
  
<dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>3.13.0</version>  
 <scope>test</scope>  
</dependency>  
Copy

Dit zal de nodige afhankelijkheden toevoegen voor Spring Boot Starter Test en Mockito.

## Voeg een eenvoudige Service-klasse toe

Voeg een eenvoudige Service-klasse toe aan uw project. Dit zal de klasse zijn die we later zullen mocken. Het kan er als volgt uitzien:

public class GreetingService {  
 public String greet(String name) {  
 return "Hello, " + name + "!";  
 }  
}  
Copy

## Voeg een eenvoudige Controller-klasse toe

Voeg een eenvoudige Controller-klasse toe aan uw project. Dit zal de klasse zijn die we later zullen testen met behulp van de Mock-klasse. Het kan er als volgt uitzien:

@RestController  
public class GreetingController {  
   
 private final GreetingService greetingService;  
   
 public GreetingController(GreetingService greetingService) {  
 this.greetingService = greetingService;  
 }  
   
 @GetMapping("/greeting/{name}")  
 public String greeting(@PathVariable String name) {  
 return greetingService.greet(name);  
 }  
}  
Copy

## Voeg een Mock Bean-klasse toe

Voeg een Mock Bean-klasse toe aan uw project. Dit zal de klasse zijn die we later zullen gebruiken om de GreetingService-klasse te mocken. Het kan er als volgt uitzien:

@Configuration  
public class MockConfig {  
   
 @Bean  
 public GreetingService greetingService() {  
 return mock(GreetingService.class);  
 }  
}  
Copy

Hier hebben we een configuratieklasse gemaakt met een bean van het type GreetingService. We gebruiken de @Bean-annotatie om de bean te definiëren en mock(GreetingService.class) om een mock-object van de GreetingService-klasse te maken.

## Schrijf tests

Nu kunnen we tests schrijven om onze GreetingController-klasse te testen met behulp van de Mock-klasse die we zojuist hebben gemaakt. Het kan er als volgt uitzien:

@SpringBootTest  
@AutoConfigureMockMvc  
public class GreetingControllerTest {  
   
 @Autowired  
 private MockMvc mockMvc;  
   
 @Autowired  
 private GreetingService greetingService;  
   
 @Test  
 public void greetingShouldReturnMessageFromService() throws Exception {  
 String name = "John";  
 when(greetingService.greet(name)).thenReturn("Hello, " + name + "!");  
  
 mockMvc.perform(get("/greeting/{name}", name))  
 .andExpect(status().isOk())  
 .andExpect(content().string("Hello, John!"));  
   
 verify(greetingService, times(1)).greet(name);  
 }  
  
}  
Copy

## Observatie

Laten we deze test eens bekijken. We hebben de @SpringBootTest-annotatie gebruikt om een geïntegreerde test uit te voeren en de @AutoConfigureMockMvc-annotatie gebruikt om een MockMvc-instantie te configureren. We hebben ook de GreetingService-bean geïnjecteerd met behulp van @Autowired.

In de test hebben we de when()-methode van Mockito gebruikt om een mock van de greetingService.greet()-methode te maken en een resultaat terug te geven. We hebben vervolgens de mockMvc.perform()-methode gebruikt om een GET-verzoek naar de /greeting/{name}-endpoint te simuleren en de andExpect()-methode gebruikt om de verwachte resultaten te controleren.

Tot slot hebben we de verify()-methode gebruikt om te controleren of de greetingService.greet()-methode slechts één keer werd aangeroepen met het juiste argument.

## Conclusie

In deze tutorial hebben we geleerd hoe we een Mock Bean-klasse kunnen maken in een Spring Boot-applicatie met behulp van Maven en Spring Boot Starter Test. Dit kan helpen bij het begrijpen van de logica van een Mock-klasse en hoe deze kan worden gebruikt om tests uit te voeren. Door gebruik te maken van Mockito, een van de meest gebruikte Test-frameworks, kunnen we gemakkelijk Mock-objecten maken en testen schrijven die onze applicatie testen.