Testen

Unit testen

Maak een nieuw Java project.

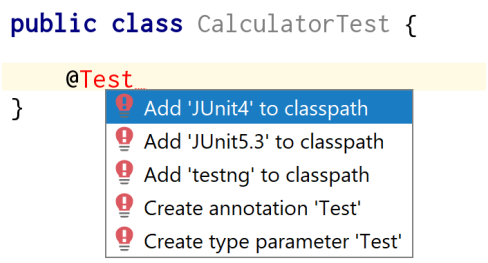
Maak een klasse Calculator met de functies:

```
public class Calculator {
   int sum(int a, int b) { return a + b;  }
   int mul(int a, int b) { return a * b;  }
   int div(int a, int b) { return a / b;  }
}
```

Maak een klasse CalculatorTest met de functie testSum, testMul en testDiv

Test ook op negatieve en nul waarden!

Bij voorkeur gebruik de @Before



Spring Test

Maak een nieuw Spring boot project met de Web library

Maak een CalculatorController met de functies sum, mul en div

```
@RestController
public class CalculatorController {
    @GetMapping("/sum")
    int sum(@RequestParam("a") int a, @RequestParam("b") int b) {
        return a + b;
    }
    // mul ... {}
    // div ... {}
}
```

```
//mvn spring-boot:run
```

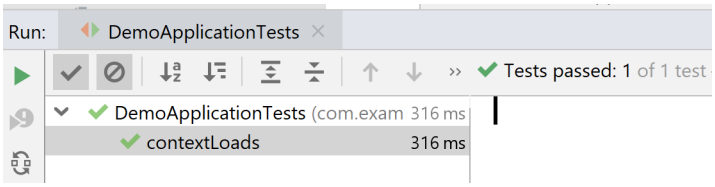
Je kunt deze testen in een browser.

http://localhost:8080/sum?a=3&b=4

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class DemoApplicationTests {

	@Autowired
	CalculatorController calculatorController;

	@Test
	public void contextLoads() {
		Assert.assertEquals(5, calculatorController.sum(2,3));
	}
}
```



AssertJ matchers

https://www.vogella.com/tutorials/AssertJ/article.html

```java
Assertions.assertThat(calculatorController.sum(2,3)).isEqualTo(5);
```

Hamcrest matchers

https://www.vogella.com/tutorials/Hamcrest/article.html

```java
MatcherAssert.assertThat(calculatorController.sum(2,3),
                         Matchers.equalTo(5));
```

Opdracht

Schrijf een Test met AssertJ matchers
Schrijf een Test met Hamcrest matchers

Mockito

Mockito wordt gebruik om een klasse te isoleren van andere klassen. In dit voorbeeld maken een CalculatorService, die een (flauwe) deelberekening uitvoert.

```java
@Service
public class CalculatorService {
    int add100(int a) { return a + 100;  }
}
@Autowired
CalculatorService calculatorService;

@GetMapping("/sum")
int sum(@RequestParam("a") int a, @RequestParam("b") int b) {
    return a + calculatorService.add100(b);
}}
```

Je kunt deze testen in een browser.

In de test mocken we de calculatorService.

```
@MockBean
CalculatorService calculatorService;

@Test
public void testMock() {
    Mockito.when(calculatorService.add100(4)).thenReturn(100);
    Assert.assertEquals(100, calculatorController.sum(0, 4));
}
```

Cucumber

http://javawerkplaats.nl/cucumber-java-skeleton.zip

Laad het project in IDEA en run de tests.

Maak een nieuw feature voor optellen.

Maak een nieuwe feature voor vermenigvuldigen.