Mockito Unit Test mit Eclipse Maven

Ziel

In komplexen verteilten Architekturen sind Java Unit Tests nur schwierig realisierbar, da solche auf anderen teilweise nicht verfügbaren Systemen basieren. In diesen Fällen dienen uns Mock Frameworks als ideale Lösung. Mit Mocks werden Systeme (Klassen, Interfaces, ...) gemockt und damit als Fake Instanzen gebaut und verwendet.Das Spring Boot Framework bietet hier selber einige Lösungen und damit Abhilfe. Dieser Blog zeigt den minimalen Setup für den Einsatz des Mockito Framework zwecks Mocking einer nicht verfügbaren Umgebung.

Voraussetzungen

Für diesen Blog verwenden wir Java Version 11+, die Eclipse IDE mit Maven Plugin sowie Maven 3.6+.

Eclipse Maven Projekt Setup

Wir starten die Eclipse IDE in einem eigenen Workspace (z.b. /home/user/blog/mockito):Wir erstellen ein Maven Projekt:Das Maven Projekt ist nun erstellt:Achten Sie darauf, dass Sie die korrekt Java Version im Projekt verwenden.

Random Anwendung

Als Anwendung wollen wir 10 Random zahlen im Bereich 1 - 1000 generieren und Minimum/Maximum sowie Mittelwert bestimmen. Wir implementieren das folgende UML Diagramm:Die Random Quelle kapseln wir über das Interface IRandom:package ch.std.blog.random;

import java.util.List;

public interface IRandom<T> {

 public abstract List<T> newRandom(int size, int min, int max);

}Das Interface wird von der Klasse RandomDoubleData verwendet als Quelle der Random Daten:package ch.std.blog.random;

import java.util.DoubleSummaryStatistics;
import java.util.List;

public class RandomDoubleData {

 private IRandom<Double> random;
 private List<Double> doubleList;
 private DoubleSummaryStatistics stats;

 public RandomDoubleData(IRandom<Double> random) {
  this.random = random;
  this.doubleList = this.random.newRandom(10, 1, 1000);
  this.stats = doubleList.stream().mapToDouble(d -> d).summaryStatistics();
 }

 public List<Double> getDoubleList() {
  return doubleList;
 }

 public Long getCount() {
  return this.stats.getCount();
 }

 public Double getMin() {
  return this.stats.getMin();
 }

 public Double getMax() {
  return this.stats.getMax();
 }

 public Double getAverage() {
  return this.stats.getAverage();
 }

}Wir haben 3 Implementationen, welche Random Daten auf verchiedene Arten generieren.Die erste Implementation ClassicRandomImpl arbeitet mit der Java Standard Klasse java.util.Random:package ch.std.blog.random.impl;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import ch.std.blog.random.IRandom;

public class ClassicRandomImpl implements IRandom<Double> {

 public List<Double> newRandom(int size, int min, int max) {
  List<Double> randomList = new ArrayList<>();
  Random r = new Random();
  for (int i = 0; i < size; i++) {
   double randomValue = min + (max - min) * r.nextDouble();
 randomList.add(randomValue);
  }
  return randomList;
 }

}Die Implementation LambdaRandomImpl arbeitet mit dem Java Lambda Ansatz:package ch.std.blog.random.impl;

import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

import ch.std.blog.random.IRandom;

public class LambdaRandomImpl implements IRandom<Double> {

 public List<Double> newRandom(int size, int min, int max) {
  return new Random().doubles(size, min, max).boxed().collect(Collectors.toList());
 }

}Die Klasse UrlFakeRandomImpl liest die Random Daten vom JSON Fake Service von der folgenden URL https://www.simtech-ag.ch/std-ajax/randomservice?min=0&max=1000package ch.std.blog.random.impl;

import java.io.InputStream;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;

import

org.json.JSONObject;
import org.json.JSONTokener;

import ch.std.blog.random.IRandom;

public class UrlFakeRandomImpl implements IRandom<Double> {

 private URL url;

 public UrlFakeRandomImpl(URL url) {
  this.url = url;
 }

 public List<Double> newRandom(int size, int min, int max) {
 List<Double> randomList = new ArrayList<>();
  for (int i = 0; i < size; i++) {
  JSONObject jsonObject = this.getJSONObject();
  Double value = jsonObject.getDouble("value");
  randomList.add(value);
  }
  return randomList;
 }

 private JSONObject getJSONObject() {
  try (InputStream is = this.url.openStream()) {
  JSONTokener tokener = new JSONTokener(is);
  JSONObject root = new JSONObject(tokener);
  return root;
  } catch (Exception e) {
  return null;
  }
 }

}

Unit Tests (Non Mock)

Random Zahlen sind nicht direkt testbar, weil das erwartete Ergebnis zufällig ist. Wir arbeiten ohne Mock mit einer Test Implementation, welche mit fixen Werten arbeitet:package ch.std.blog.random.test;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.Arrays;
import java.util.List;

import org.junit.Assert;
import org.junit.jupiter.api.Test;

import ch.std.blog.random.IRandom;
import ch.std.blog.random.RandomDoubleData;
import ch.std.blog.random.impl.UrlFakeRandomImpl;

class TestRandom implements IRandom<Double> {

 @Override
 public List<Double> newRandom(int size, int min, int max) {
  return Arrays.asList(1.0, 2.0, 3.0);
 }

}

public class RandomDoubleUnitTest {

 @Test
 public void testDoubleRandom() {
  RandomDoubleData rdd = new RandomDoubleData(new TestRandom());
  double expectedMin = 1.0;
  double expectedMax = 3.0;
  double expectedAverage = 2.0;

  Assert.assertEquals(expectedMin, rdd.getMin(), 0.0);
 Assert.assertEquals(expectedMax, rdd.getMax(), 0.0);
 Assert.assertEquals(expectedAverage, rdd.getAverage(), 0.0);

 }

 @Test
 public void testDoubleFakeRandom() throws MalformedURLException {
 RandomDoubleData rdd = new RandomDoubleData(new UrlFakeRandomImpl(new URL("https://www.simtech-ag.ch/std-ajax/randomservice?min=0&max=1000")));
  Long expectedCount = 10L;
  Assert.assertEquals(expectedCount, rdd.getCount());

 }

}

Mockito Unit Tests

Mit dem Mockito Framework können wir Interfaces als Mock implementieren. Das folgende Listing zeigt diesen Ansatz:package ch.std.blog.random.test;

import static org.mockito.ArgumentMatchers.anyInt;
import static org.mockito.Mockito.when;

import java.util.Arrays;

import org.junit.Assert;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import ch.std.blog.random.IRandom;
import ch.std.blog.random.RandomDoubleData;

@ExtendWith(MockitoExtension.class)
public class MockRandomDoubleUnitTest {

 @Mock
 private IRandom<Double> mockRandom;

 @Test
 public void testDoubleMockRandom() {

  when(mockRandom.newRandom(anyInt(),anyInt(), anyInt())).thenReturn(Arrays.asList(1.0, 2.0, 3.0));
  RandomDoubleData rdd = new RandomDoubleData(mockRandom);
  double expectedMin = 1.0;
  double expectedMax = 3.0;
  double expectedAverage = 2.0;

  Assert.assertEquals(expectedMin, rdd.getMin(), 0.0);
  Assert.assertEquals(expectedMax, rdd.getMax(), 0.0);
 Assert.assertEquals(expectedAverage, rdd.getAverage(), 0.0);

 }

}Damit dies kompilierte benötigen wir die Mockito Library inkl. JUnit 5 Jupiter Support:<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

<groupId>blog.mockito</groupId>

<artifactId>mockitoeclipsemaven</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>blog mockito eclipse maven</name>

<dependencies>
  <!-- https://mvnrepository.com/artifact/org.json/json -->
  <dependency>

<groupId>org.json</groupId>

<artifactId>json</artifactId>

<version>20210307</version>

</dependency>

  <!-- https://mvnrepository.com/artifact/org.mockito/mockito-core -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.9.0</version>

<scope>test</scope>

</dependency>

  <dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-junit-jupiter</artifactId>

<version>2.23.0</version>

<scope>test</scope>
  </dependency>

</dependencies>

</project>Mit der @Mock Annotation wird die betroffene Klasse via Proxy Pattern gespiegelt und als Fake Instanz implementiert. Mit der when Anweisung wird die Mock Instanz für den Test vorbereitet, so dass die gesuchten Daten geliefert werden.

Komplettes Eclipse Projekt

Das komplette Eclipse Projekt findet man unter dem Link mockitoeclipsemaven.zip.

Feedback

War dieser Blog für Sie wertvoll. Wir danken für jede Anregung und Feedback

Kontakt

Simtech AG
Finkenweg 23
3110 Münsingen
Schweiz

Impressum

Copyright

https://www.simtech-ag.ch/blog/java/mockitoeclipsemaven