Arbeitsblatt 07 Unit-Testing mit JUnit 5

1 Einleitung

Als Softwareentwickler liegt es in Ihrer Verantwortung qualitativ hochwertigen Sourcecode für Ihre Projekte zu schreiben. Doch was ist hochwertiger Code und wie können Sie solchen schreiben? In der Softwareentwicklung gibt es verschiedene Paradigmen, Methodiken und Verfahren die, wenn Sie korrekt angewendet werden zu hochwertigen, das heisst, leicht erweiterbaren, wartbaren, verständlichen, übersichtlichen, testbaren und wiederverwendbarer Code führen.

Eine passende Architektur mit intelligentem Design legt den Grundstein, die korrekte Auswahl und Anwendung von Tools (hier Programmiersprachen) zusammen mit dem entsprechenden Programmierparadigma unter Anwendung von bestimmten Grundprinzipien und Methodiken ist ein wichtiger Teil der Arbeit als professionelle:r Softwareentwickler:in. In der Praxis werden diese Aufgaben meist auf mehrere Softwareentwickler und Softwareengineers aufgeteilt, da die Erfüllung der aufgeführten Tätigkeiten grosses Wissen und grosse Erfahrung benötigt.

Im Modul 404 haben Sie bereits einige wichtige Bausteine für die Produktion von hochwertigen Sourcecode kennengelernt. Sie haben angefangen Objektorientiert zu Programmieren und wenden die die Prinzipien der Kapselung und der Abstraktion an. In diesem Arbeitsblatt lernen Sie nun, wie Sie die Qualität Ihres Codes weiter steigern können, indem Sie ihn testbar machen.

1.1 Warum testen wir:

Wenn etwas produziert wurde, testet man es, um zu sehen, ob es entsprechend funktioniert. Software wird oft weiterentwickelt und wächst mit dem Lebenszyklus. Dabei wird einiges erweitert, abgeändert und gelöscht, doch meist bleiben grundlegende Funktionen erhalten und diese müssen immer wieder getestet werden. Aus diesem Grund ist es sinnvoll Zeit und Energie in sogenannte Unit-Tests zu investieren. Diese ermöglichen es Ihnen Code bzw. Funktionalitäten wiederholbar und effizient zu testen.

1.2 Wie testen wir:

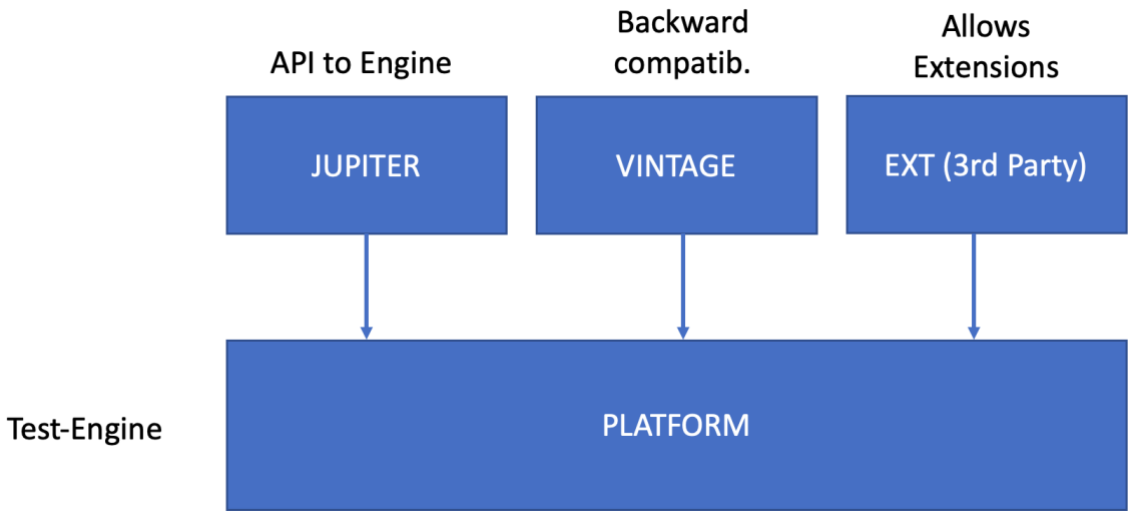
Für Java wird hauptsächlich das Testing Framework JUnit verwendet. Die neuste Version ist JUnit 5 und diese verwend wir in diesem Arbeitsblatt. Mithilfe des Frameworks sehen Sie wie einfach es ist Java Code testbar zu machen.

1.3 Was ist JUnit 5:

JUnit 5 ist das meistgenutzte Test-Framework für Java. Weil das Testen von Sourcecode für die Entwicklung so wichtig ist und JUnit 5 so populär ist, haben eigentlich alle grossen IDEs für Java wie IntelliJ oder Eclips dieses sehr gut integriert, was uns den Umgang damit erleichtert.

JUnit 5 besteht ausfolgenden 4 Komponenten:

1. JUnit Test-Engine oder Platform
2. Die Jupiter APIs (Werden von Entwicklern verwendet, um Tests zu schreiben)
3. Vintage (Erlaubt «Rückwärtskompatibilität zu JUnit 4»)
4. Extension erlaubt die Erweiterung von JUnit 5, bspw. durch 3rd Party Bibliotheken



Link zu JUnit 5 Hompage finden Sie hier.

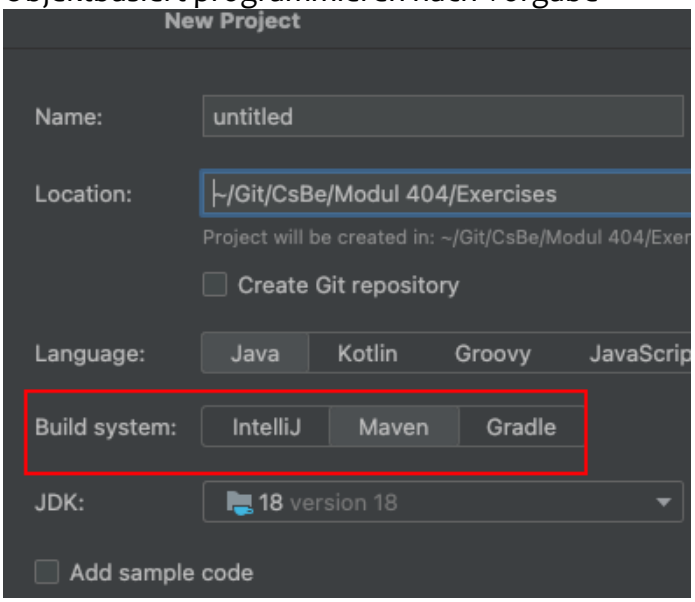
2 Wie wird JUnit 5 verwendet

JUnit 5 gehört nicht zu den Standardbibliotheken von Java und ist somit eine Dependency/Abhängigkeit. Sie kennen bereits Dependencies wie beispielsweise JavaFX welches Ihnen erlaubt grafische Benutzeroberflächen für Java Programme zu erstellen. Bei JavaFX wie auch bei JUnit verwenden wir deshalb ein Tool mit dem Namen Maven.

Maven ist ein Projektmanager welcher es uns unter anderem erlaubt Abhängigkeiten in einem Projekt einfach zu verwalten. Ausserdem können sich wiederholende Aufgaben wie bswp. Deployments oder Testing usw. über Maven durchgeführt werden. Maven bietet auch diverse Plug-Ins an, mit denen die Funktionalität noch weiter erweitert werden kann. Neben Maven gibt es noch andere populäre Tools wie bspw. Gradle.

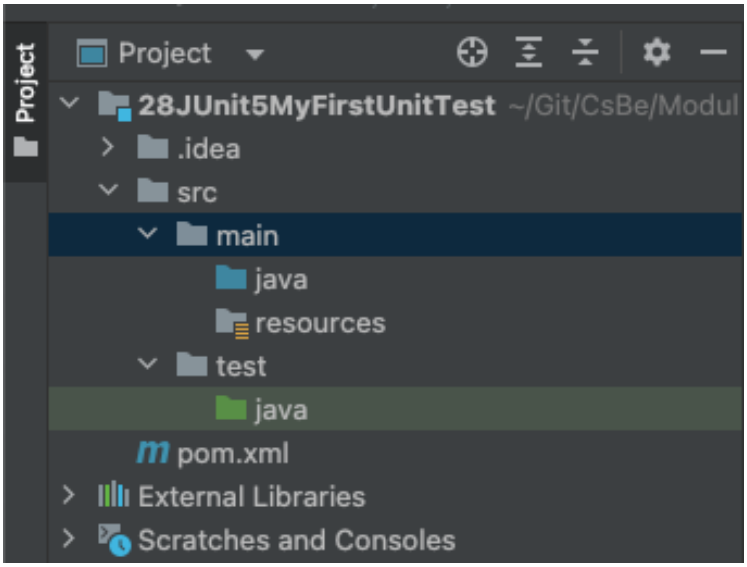
Den Link zu Maven finden Sie hier.

Wenn Sie also beabsichtigen Unit-Tests in ihrem Java-Projekt zu schreiben, stellen Sie sicher, dass Sie im IntelliJ entsprechend Maven als Build-Tool auswählen. Geben Sie dem Projekt einen Namen z.B. «28GeometryUnitTesting».



Wenn Sie das Projekt erstellt haben, sollte sich bereits die Maven Verwaltungsdatei pom (Project Object Model) geöffnet haben. Klappen Sie nun im Projektexplorer Ihr Projekt den Ordner src sowie die beiden Unterordnet main und test auf.

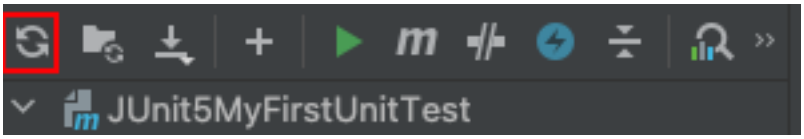
Sie erkennen, dass sich die Projektstruktur etwas verändert, hat zu einem gewöhnlichen IntelliJ Java Programm. Sie haben bereits den Ordner test, wo Sie später ihre Unit-Tests schreiben werden sowie die Projekt Object Model (pom.xml) Datei, mit der Sie Ihr Projekt verwalten können.



Fügen Sie die neuste Version von JUnit 5 als Dependency zu Ihrer pom.xml Datei hinzu. Öffnen Sie dazu diese und schreiben Sie folgenden Code unterhalb des Nodes <properties></properties>

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.8.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.8.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Öffnen Sie nun das Maven-Fenster mit einem Klick auf den Menüpunkt «View» → «Tool Windows» → «Maven» und klicken Sie auf «Reload all Maven Projects».

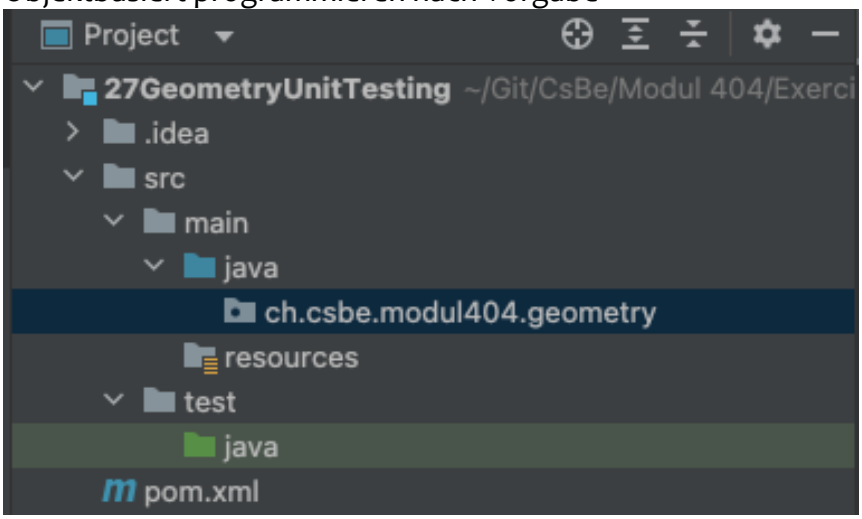


Nun sollten Sie im Projektexplorer unter «External Libraries» die nötigen Dependencies aufgeführt haben.

Jetzt haben Sie Ihr Projekt so weit aufgesetzt, damit Sie mit dem Schreiben des ersten Unit-Tests beginnen können.

3 Addition als erste Unit-Test

Erstellen Sie ein neues Paket unter src/main/java mit dem Namen **«ch.csbe.modul404.geometry»**.



Erstellen Sie nun die Klassen: Main mit der main()-Methode, die Klasse Rectangle, die Klasse Square, die Klasse RectangularTriangle und die Klasse Circle.

Öffnen Sie die Klasse Triangle und schreiben Sie folgendes Methodengerüst:

```
package ch.csbe.modul404.geometry;

public class Rectangle {

    public double calculateExtent(double length, double width) {
        return 0;
    }

    public double calculateArea(double length, double width) {
        return 0;
    }
}
```

Im Moment interessiert Sie nur welche Methoden Sie später implementieren wollen. Als erstes werden Sie nämlich die dazugehörigen Tests schreiben und erst nachdem die Tests geschrieben wurden den Code implementieren. Dies ist eine bekannte Methodik in der Softwareentwicklung und nennt sich Test Driven Development TDD.

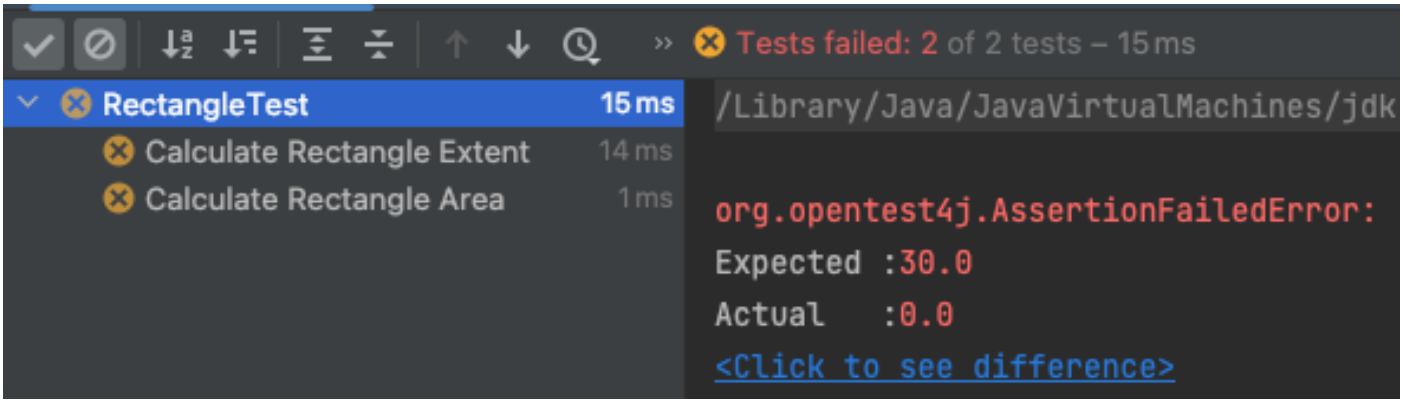
Navigieren Sie nun in das Verzeichnis src/test/java und erstellen Sie die Klassen: RectangleTest, SquareTest, RectangularTriangleTest und CircleTest. Navigieren Sie in die RectangleTest-Klasse und überlegen Sie sich kurz wie Sie Umfang und Fläche eines Rechtecks berechnen.

Machen Sie sich nun an die Implementierung der Testfälle. JUnit arbeitet sehr viel mit @Annotations, diese kennen Sie bereits aus JavaFX wo Sie die Annotation: **@FXML** verwendet haben. Hier verwenden Sie nun unter anderem die folgenden Annotations:

@Test	Zeichnet die nachfolgende Methode als Unit-Test aus.
@BeforeEach	JUnit LifeCycle Hook der vor der Ausführung jeder Test-Methode ausgeführt wird.
@DisplayName	Erlaubt die Angabe eines Benutzerdefinierten Namens für den Unit-Test.

```java
import ch.csbe.modul404.geometry.Rectangle;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class RectangleTest {

    private Rectangle rectangle;

    @BeforeEach()
    void init() {
        rectangle = new Rectangle();
    }

    @DisplayName("Calculate Rectangle Extent")
    @Test
    public void testCalculateExtent() {
         assertEquals(30, rectangle.calculateExtent(10, 5));
    }

    @DisplayName("Calculate Rectangle Area")
    @Test
    public void testCalculateArea() {
        assertEquals(50, rectangle.calculateArea(10, 5));
    }
}
```

Wenn Sie die Rectangle-Klasse nun testen wollen, klicken Sie entweder mit der rechten Maustaste im Projekt-Explorer auf die RectangleTest-Klasse und drücken auf «Run 'RectangleTest'» oder Sie führen den Test via Terminal aus indem Sie ***mvn -Dtest=RectangleTest test*** eingeben.

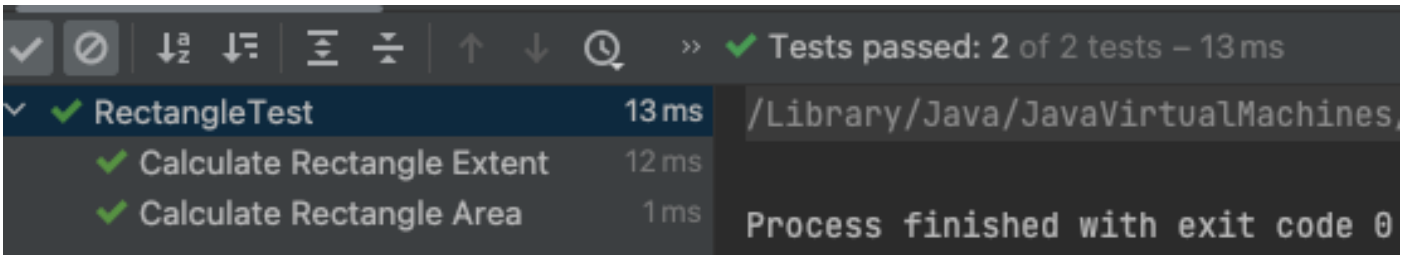


Wie erwartet sind die beiden geschriebenen Tests fehlgeschlagen, da die beiden Methoden ja noch nicht implementiert wurden und nur 0 zurückgeben. Nun geht es darum, ganz nach dem Test Driven Development, die Implementierung vorzunehmen, so dass die beiden Tests erfolgreich durchgeführt werden können.

Wechseln Sie also zurück in die Rectangle-Klasse und implementieren Sie die beiden Methoden wie folgt:

```java
package ch.csbe.modul404.geometry;

public class Rectangle {

    public double calculateExtent(double length, double width) {
        return (2 * length + 2 * width);
    }

    public double calculateArea(double length, double width) {
        return (length * width);
    }
}
```

Starten Sie nun die Test für RectangleTest erneut, nun sollten beide Unit-Test erfolgreich durchlaufen.



4 Auftrage

Schreiben Sie nun nach der Test Driven Development (TDD) Methodik die Testfälle und Implementierungen der Klassen Square, Triangle und Circle.

Die Klasse Square soll folgende Methoden beinhalten:

- calculateExtent(double site):double
- calculateArea(double site):double
- calculateSiteLength(double area):double
- calculateSiteLength(double extent):double

Die Klasse Triangle soll folgende Methoden beinhalten:

- calculateExtent(double a, double b, double c):double
- calculateArea(double g, double height):double
- calculateGround(double area, double height):double
- calculateHeight(double area, double ground):double
- calculateA(double b, double c):double
- calculateB(double a, double c):double
- calculateC(double a, double b):double

Die Klasse Circle soll folgende Methoden beinhalten:

- calculateExtent(double r):double
- calculateArea(double r):double
- calculateRadius(double area):double
- calculateRadius(double extent):double

Melden Sie sich bei der Lehrperson sollten Sie Fragen haben oder mit der Aufgabe fertig sein.