Test Driven Development Trainer

Benutzerhandbuch

Inhaltsverzeichnis

Diese Anwendung ist im Rahmen des Programmierpraktikums 2016 an der Heinrich-Heine-Universität Düsseldorf entstanden.

1 Einführung

Das sogenannte Test Driven Development (TDD) ist eine spezielle Form der Softwareentwicklung. Dabei werden noch vor dem eigentlichen Code Tests geschrieben, die es schrittweise durch Erweiterungen des Codes zu bestehen gilt. Es soll darauf Acht gegeben werden, dass das Programm gerade nur so viele Erweiterungen erfährt, dass der nächste anstehende Test gerade erfüllt wird.

Auch wenn es auf den ersten Blick aufwändiger erscheint, neben dem Code noch Tests zu schreiben, ist es vorteilhaft, testgetrieben zu programmieren. Zum einen wird nämlich durch die Spezifikation der Tests klarer, was genau man programmieren muss. Zum anderen wird sichergestellt, dass das programmiert wird, was auch gefordert ist, da sonst die Tests fehlschlagen würden. Weiterhin bewirkt der Einsatz dieser Technik, dass das Programm keine unnötigen Bestandteile enthält, weil es in jedem Schritt gerade nur den jeweiligen Test erfüllen muss. Insgesamt wird also auf diese Weise ein tiefergehendes Verständnis des Programms und der Anforderungen an das Programm sichergestellt.

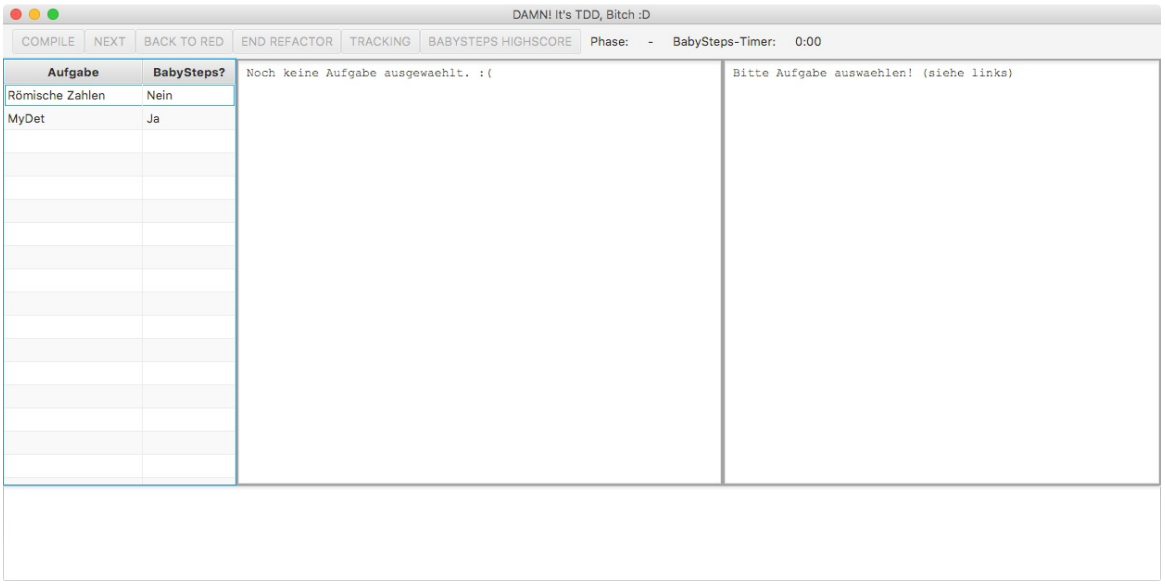
Über den TDD Trainer (TDDT) wird ein Werkzeug bereitgestellt, um die testgetriebene Entwicklung zu üben. Dabei besteht die Möglichkeit, aus einem Katalog an Aufgaben eine herauszuwählen und diese nach dem TDD-Prinzip zu programmieren. Das folgende Nutzerhandbuch soll in die Nutzung des TDDT einführen.

2 Die Anwendung

2.1 Der Grundaufbau

Nachdem Sie die Anwendung starten, öffnet sich das Hauptfenster in dem Sie sich fast ausschließlich während der Arbeit befinden. Bevor jedoch mit der Entwicklung gestartet werden kann, werden Sie dazu aufgefordert einen Aufgabenkatalog auszuwählen. Dieser muss vom XML-Dateiformat sein.

Die Auswahl eines Kataloges ist verpflichtend.



Links finden Sie den Katalog an Aufgaben, aus dem Sie per Linksklick wählen können. Das Erstellen eigener Aufgaben und Aufgabenkataloge ist Bestandteil des nächsten Abschnitts.

In der Mitte finden Sie zwei Texteingabe-Bereiche, in dem Sie Ihren Code (links), sowie Ihre Tests (rechts) schreiben können.

Diese Komponenten bilden zusammen den Arbeitsbereich.

Unterhalb des Arbeitsbereichs finden Sie ähnlich wie bei einer IDE einen Bereich für die Ausgabe von etwaigen Fehlermeldungen nach der Kompilierung oder eine Liste von Tests, die fehlgeschlagen sind.

Die Aufgabenstellung befindet sich als Blockkommentar oben im Editor für den Code. Sollte keine Aufgabenbeschreibung angegeben werden, so wird auch kein Blockkommentar erzeugt.

Ganz oben befindet sich eine Leiste mit sechs Buttons:

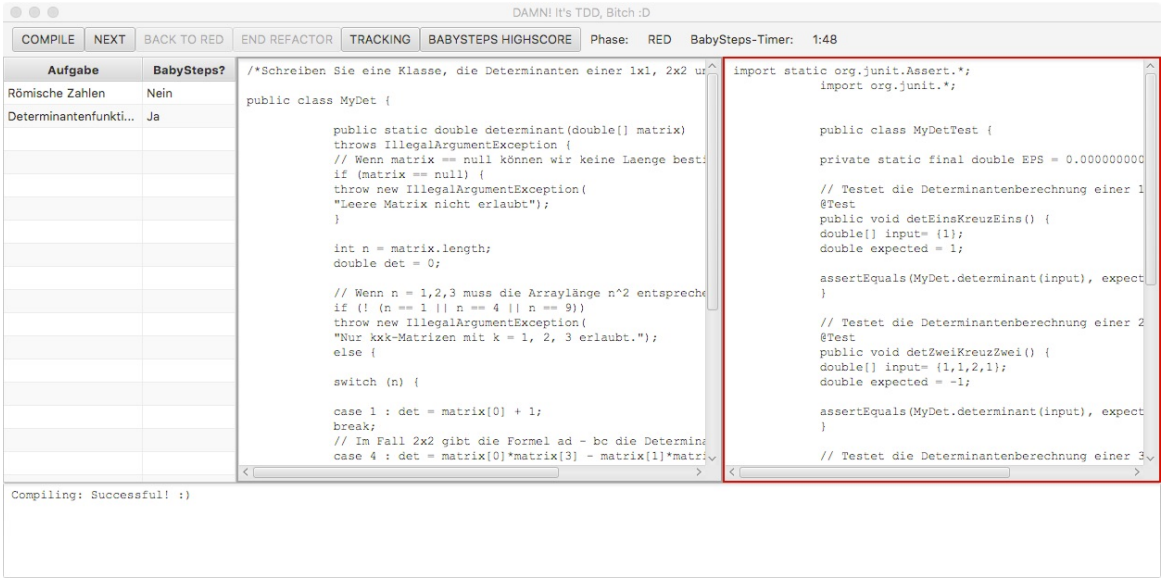
- Compile (Kompiliert den Code, führt aber keine Tests aus)
- Next (Kompiliert den Code und führt die Test aus)
- Back to Red (Wechselt zurück in die Phase *RED*, wenn in zur Zeit in *GREEN*)
- End Refactor (Beendet die *REFACTOR*-Phase)
- Tracking (Öffnet das Fenster für Tracking)
- Babysteps Highscorelist

2.2 Der Entwicklungslauf

Innerhalb der Anwendung gibt es drei Phasen: *RED, GREEN* und *REFACTOR*. In welcher aktuell gearbeitet wird lässt sich zum einen an der Phasenangabe direkt neben dem rechten Button `Babysteps-Highscores` und zum anderen an der Umrandung der Editoren. Ein roter Rand bedeutet *RED*, ein grüner *GREEN* und beide grau *REFACTOR*.

2.2.1 RED

Nachdem Sie eine Aufgabe aus dem Aufgabenkatalog gewählt haben, befinden Sie sich in der Phase *RED* und können mit der testgetriebenen Entwicklung sofort beginnen.



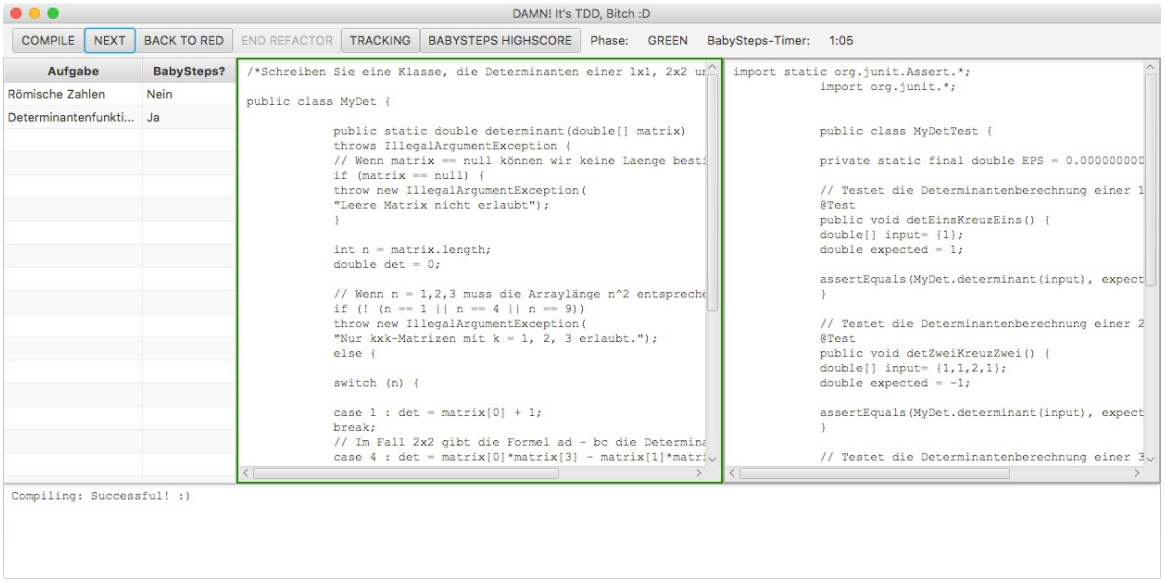
Innerhalb dieser Phase darf ausschließlich Code der Testklasse modifizieren. Es wird dabei verlangt, dass hier ein Test geschrieben wird, sodass mindestens ein Test fehlschlägt oder die Kompilierung nicht funktioniert.

Wenn Sie noch keinen Code geschrieben haben, weil Sie gerade erst begonnen haben, schlägt die Kompilierung fehl, sodass obige Bedingung auch dann erfüllt ist. Nachdem Sie einen Test geschrieben oder editiert haben, können Sie nun entweder durch einen Klick auf `Compile` kompilieren oder durch einen Klick auf `Next` in die Phase *GREEN* wechseln. Voraussetzung ist hierbei, dass genau ein Test fehlgeschlagen ist. (vgl. Abschnitt 1)

2.2.2 GREEN

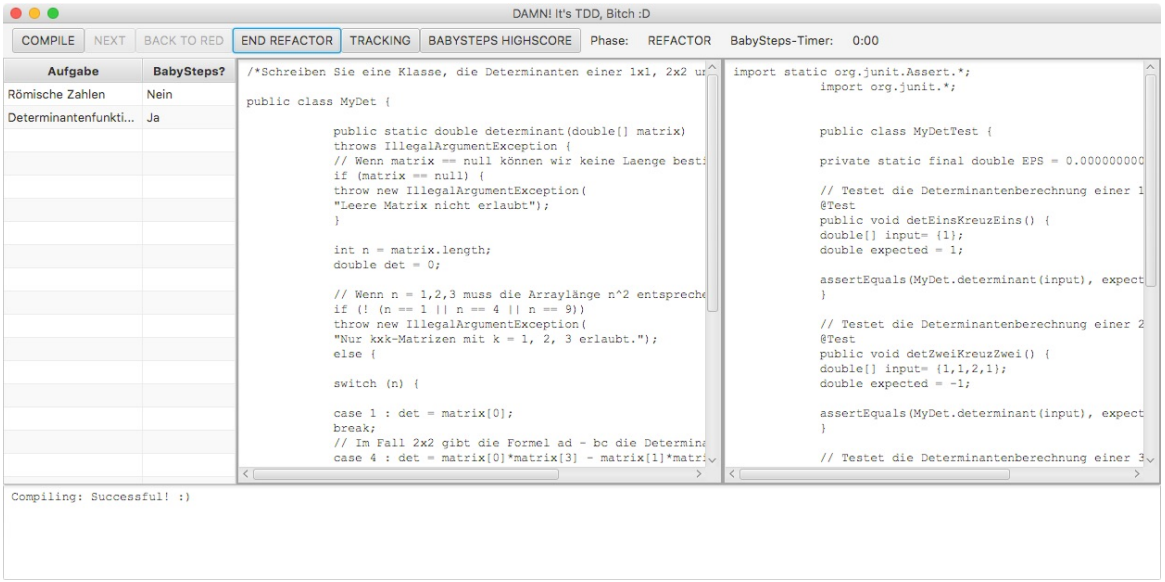
Befinden Sie sich in diesem Modus, so ist es Ihre Aufgabe den Code (linke Seite) so zu bearbeiten, dass alle Tests bestanden werden.

Der Wechsel des Modus wird erneut über einen Klick auf `Next` erfolgen. Hierbei können Sie sich entscheiden, ob sie in den Modus *REFACTOR* wechseln möchten oder in den Modus *RED*.



2.2.3 REFACTOR

In dieser Phase dürfen Code und Teste beliebig verändert werden. Diese Phase kann mit einem Klick auf `End Refactor` beendet werden. Voraussetzung dafür ist das erfolgreiche Kompilieren beider Klassen sowie das Bestehen aller Teste. Anschließend befinden Sie sich wieder im Modus *RED*.



3 Der Aufgabenkatalog

Der im vorherigen Abschnitt bereits angesprochene Aufgabenkatalog besteht aus einer XML-Datei. Der konkrete Aufgabenkatalog muss beim Starten der Anwendung ausgewählt werden.

3.1 Dateistruktur

Das Wurzelelement eines jeden Aufgabenkatalogs ist `<exercises>`. Ist dies nicht der Fall, so erhalten Sie eine entsprechende Fehlermeldung. Eine Aufgabe muss als Block `<exercise>` angegeben werden. Diese müssen *direkte* Söhne von `<exercises>` sein. Eine Aufgabe besteht dabei aus verschiedenen Eigenschaften, von denen manche verpflichtend, einige aber auch nicht angegeben werden müssen.

3.2 Beispielkonfiguration

Im folgenden sehen Sie einen Beispielaufgabenkatalog mit einer Aufgabe:

```
<?xml version="1.0" encoding="utf-8"?>
  <exercises>
    <exercise name="Polynomrechner">
      <description>
        Ein Polynomrechner in der Minimalversion
      </description>

    <classes>
      <class name="SimplePolynomialCalculator">
        public class SimplePolynomialCalculator {
          // Schreiben Sie Ihren Code hier
        }
      </class>
    </classes>

    <tests>
      <test name="SimplePolynomialCalculatorTest">
        import static org.junit.Assert.*;
        import org.junit.Test;
        public class SimplePolynomialCalculatorTest {
          @Test
          public void testSomething() {
          }
        }
    </test>
    </tests>

    <!-- Aktiviere oder deaktiviere Erweiterungen -->
    <config>
      <babysteps enable="true" time="120" />
    </config>
    </exercise>
```

```
</exercises>
```

Die Aufgabe enthält dabei die folgenden Daten:

- **Name:** Polynomrechner
- **Beschreibung:** Ein Polynomrechner in der Minimalversion
- Babysteps ist mit einem Timer von 120 Sekunden aktiviert.

4 Erweiterungen

4.1 BabySteps

Wenn Sie eine Aufgabe anklicken, für die die BabySteps-Option auf *Ja* gesetzt ist, so haben Sie nur eine begrenzte Bearbeitungszeit für die Phasen „Rot“ und „Grün“, d.h. für das Refactoring besteht auch dann keine zeitliche Begrenzung. Dabei können Sie die gewünschte Bearbeitungszeit in Sekunden in der Konfiguration der Aufgabe nach Belieben wählen, wobei zwei bis drei Minuten empfohlen werden, damit das Lernziel dieser Option nicht außer Acht gelassen wird. Wenn die Zeit abgelaufen ist, wird der jeweilige Test oder Code, der in dieser Phase zusätzlich geschrieben wurde, gelöscht und zur vorherigen Phase gewechselt. BabySteps soll dem Anwender das Programmieren in kleinen Schritten nahe legen und ihn im Hinblick darauf sensibilisieren, dass er sich nur auf das Wesentliche konzentriert.

Als Option für Aufgaben, für die die BabySteps-Funktion freigeschaltet ist, gibt es weiterhin die Highscore-Funktion. Die Zeit, die Sie zum dritten Erreichen der Phase „Grün“ gebraucht haben (ausgenommen die Refactor-Phasen) wird hierbei gemessen und den Zeiten anderer Nutzer, die gleiche Aufgabe betreffend, gegenübergestellt. Dazu klicken Sie den `BABYSTEPS HIGHSCORE`-Button in der oberen Leiste an, wenn Sie die Phase „Grün“ das dritte Mal erreicht haben, woraufhin Sie einen Namen eingeben müssen, unter dem Sie ggf. in der Highscoreliste aufgeführt werden. Im Anschluss werden Ihnen die 5 besten Zeiten angezeigt. Nachdem Sie diesen Button gewählt haben und das Highscore-Fenster geschlossen haben, können Sie nicht mehr mit der Bearbeitung der entsprechenden Aufgabe fortfahren. Sie müssten ggf. von neu anfangen, daher empfiehlt es sich, die Highscore-Funktion erst am Ende Ihrer Bearbeitung der Aufgabe anzuwählen, es wird dann nur die Zeit bis zur Vollendung der ersten zwei Zyklen gemessen.

4.2 Tracking

Das Tracking dient dazu eventuell aufgetretene Schwierigkeiten zu finden und zu Analysieren. Um dies zu ermöglichen, werden Ihnen drei Diagrammtypen zu Verfügung gestellt, zwischen denen oben links durch Klicken auf den entsprechenden Button gewechselt werden kann. Die Diagrammtypen sind:

- Kreisdiagramm
- Leiste
- Säulendiagramm

Neben jeder Phase steht die jeweilige Dauer in Sekunden. Klickt man auf eine Phase, so wird der dazugehörige Code und Test in den beiden Textfeldern nebenan angezeigt. Fährt man mit der Maus über eine Phase, so werden Code und Test nur vorläufig angezeigt. Verlässt man mit der Maus wieder die Phase, so wird der ursprüngliche Inhalt der Textfelder wieder angezeigt.