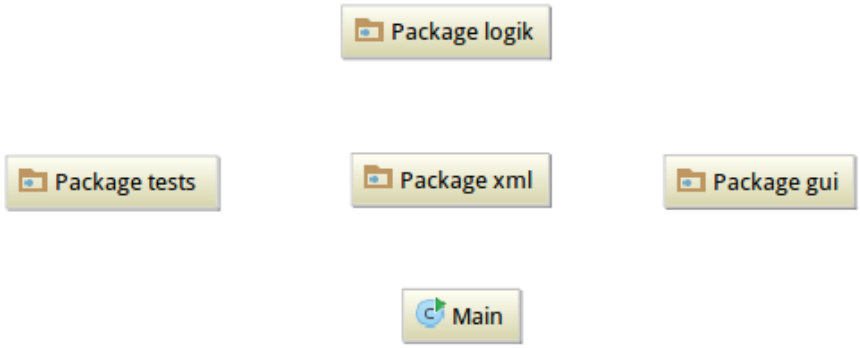
Systembeschreibung („Test Driven Development Trainer“ von „Amigos“)

In der vorliegenden Systembeschreibung soll das System zum einen als Ganzes betrachtet werden, um im Nachfolgenden die wichtigsten Komponenten noch einmal im Einzelnen zu betrachten.

Zur Betrachtung der Arbeitsweise des Teams betrachten Sie bitte die Protokolle für die einzelnen Wochen und das Abschlussprotokoll für eine zusätzliche Bewertung der gesamten Zeit.



Generelle Arbeitsweise

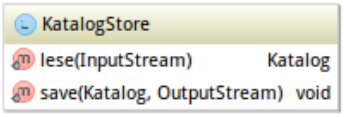
Nach Nutzerinteraktion wird ein Katalog aus einer XML-Datei geladen. Aus der XML-Datei wird also ein Katalog-Objekt erzeugt, das wiederum einzelne Aufgaben bzw. Exercises enthält. In den Aufgaben sind verschiedene Eigenschaften spezifiziert. Lesen Sie dazu den genauen Aufbau einer XML-Katalog-Datei in unserem Handbuch nach.

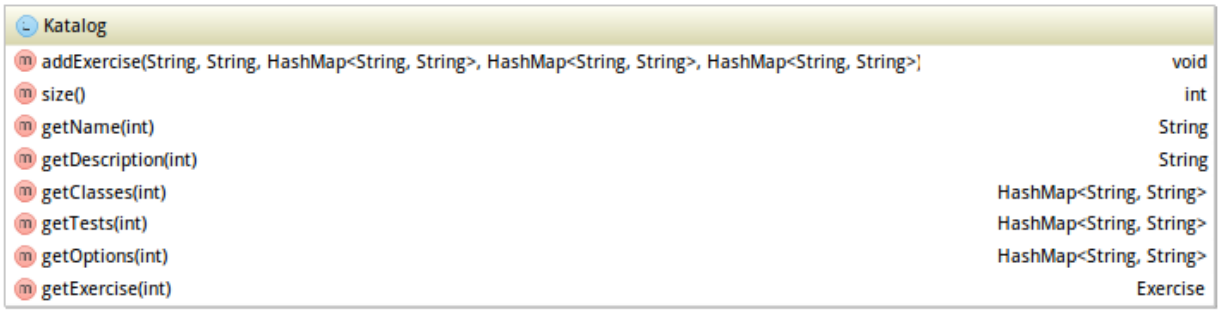
Die GUI, die den Katalog innehält, gibt eine Aufgabe, die der Benutzer ebenfalls dort auswählt, an den Logik-Handler, indem ein Objekt der Klasse *„LogikHandler“* mit der *Exercise* als Übergabeparameter an den Konstruktor instanziiert wird. Dieser *LogikHandler* übernimmt alle Aufgaben der Verwaltung des testgetriebenen Entwickelns. Der GUI werden so wenig Aufgaben wie möglich überlassen, sodass eine gute Testbarkeit gewahrt wird.

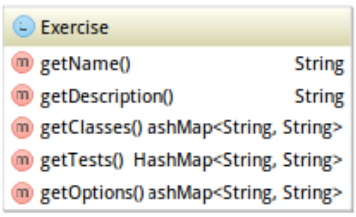
Diese Arbeitsweise ist auch noch einfach dadurch zu erweitern, dass man gegebenenfalls den Katalog auch wieder abspeichern kann. Da das Programm erst einmal lediglich die Arbeitsweise des testgetriebenen Entwickeln vermitteln soll, haben wir diese Funktion ausgelassen, aber seitens der XML-Verarbeitung ist das Feature vorhanden. Dazu weiter unten mehr.

Arbeitsweise im Detail

XML-Handling







Die operierende Klasse für das XML-Handling ist die „*KatalogStore*“-Klasse. Sie beinhaltet zum einen das Lesen einer Datei bzw. eines Input-Streams („*InputStream*“) und zum anderen das Speichern eines gegebenen Katalogs in einen *Output-Stream*. Die Entscheidung zur Abstraktion auf Streams in beiden Fällen hing mit den Faktoren zusammen, dass die Testbarkeit wieder gewahrt werden kann und andere Eingangsquellen wie zum Beispiel Internetadressen eingebunden werden können.

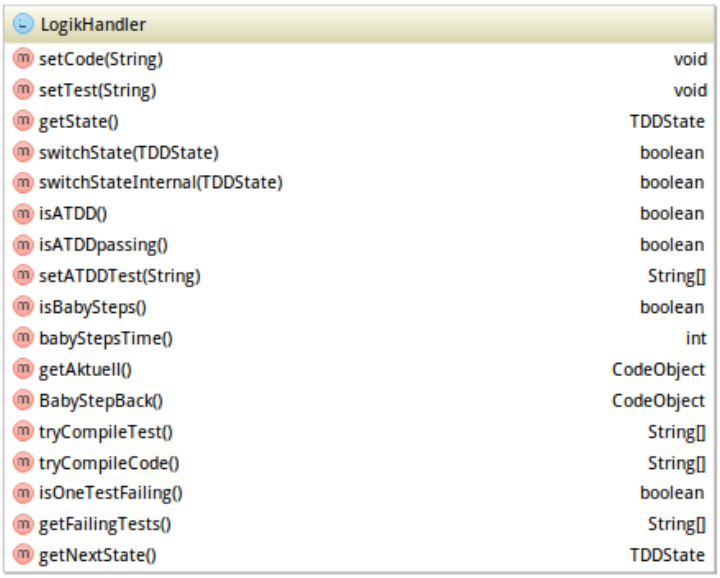
Die XML-Files werden innerhalb des Projektes als Katalog-Objekte gehandhabt. Diese enthalten neben einer Array-List von Exercises auch eine Reihe von Funktionen, die die Kommunikation der GUI mit der Gesamtheit des Katalogs vereinfachen soll. So kann man die Anzahl an Exercises erfragen, damit es nicht nötig ist, die ArrayList von Aufgaben als Ganzes auszugeben.

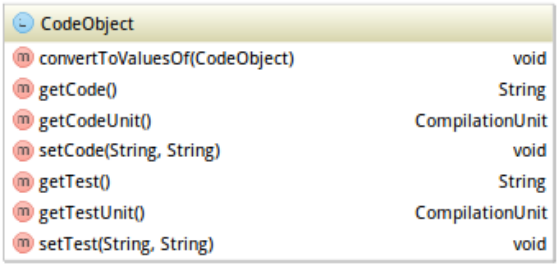
Die *Exercise*s spiegeln auch die Angaben aus dem XML-File wider. Es sind somit Name, Beschreibung, Klassen, Tests und Optionen enthalten. Es handelt sich dabei übrigens um eine variable Anzahl Klassen und Tests, obwohl bisher die Nutzung der ersten angegebenen Klasse sowie des ersten Tests implementiert ist. Die Möglichkeit zum Einlesen mehrerer Klassen und Tests soll die Erweiterbarkeit wahren, jedoch wurden die „Informatik I“-Aufgaben damals auch immer nur mit einem Test und einer Klasse ausgegeben, sodass wir uns für diese Implementierung, die man später auch ausbauen kann, entschieden.

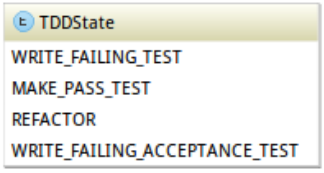
Liest man nun eine XML-Datei ein, so wird erst einmal der Datei-Inhalt in einem Dokument-Objekt gespeichert. Eine *NodeList* mit allen *Exercise*s wird aus dem Dokument erzeugt, die dann durchlaufen wird. Dem auszugebenden Katalog werden dann alle Daten angefügt.

Möchte man einen Katalog speichern, so wird aus dem Katalog ein neues Dokument erzeugt. Dieses wird mit den Daten aus dem Katalog nach demselben Schema wie bei dem Input-XML-File aufgebaut, sodass dieses Dokument am Ende in ein *DOMSource*-Objekt eingespeist wird und dann kann durch ein *Transformer*-Objekt das Dokument an seinem vorgegebenen Ziel gespeichert werden.

Logik-Handling







Für das Logik-Handling ist ein Objekt der Klasse „*LogikHandler*“ verantwortlich. Damit übernimmt ein solcher Handler eine *Exercise*. Diese wird bei der Instanziierung übergeben, sodass die nötigen Daten ausgelesen werden können. Diese sind für das Bearbeiten der Aufgabe zunächst einmal die beiden von uns implementierten Optionen aTDD und babySteps. Dafür anfangs erheblich sind nur die Informationen, ob jeweils die Option aktiviert ist. Dabei prüft man, ob in der Optionen-Liste die Option eingetragen ist, da nach unserer Konvention das XML-Dokument lediglich die Optionen beinhaltet, die bei der Bearbeitung der Aufgabe aktiv sein sollen.

Der Status im Entwicklungszyklus wird auch bei Initiierung festgelegt: Ist man in der ATDD-Entwicklung, so startet man in „Write Failing Acceptance Test“, ansonsten in „Write Failing Test“. Weiterhin liegen *CodeObject*s (beinhalten Klasse, Test und deren CompilationUnits) für aktuellen Code sowie letzten bestandenen Code vor. Für den Acceptance Test liegt auch ein String sowie eine *CompilationUnit* vor.

Mit *setCode* und *setTest* kann man die Klasse bzw. den Test dem Logik-Handler zur Verfügung

stellen, der im CodeObject des aktuellen Codes abspeichert. Die internen Methoden des CodeObjects erstellen indes übrigens eine *CompilationUnit*, die der *LogikHandler* auslesen kann.

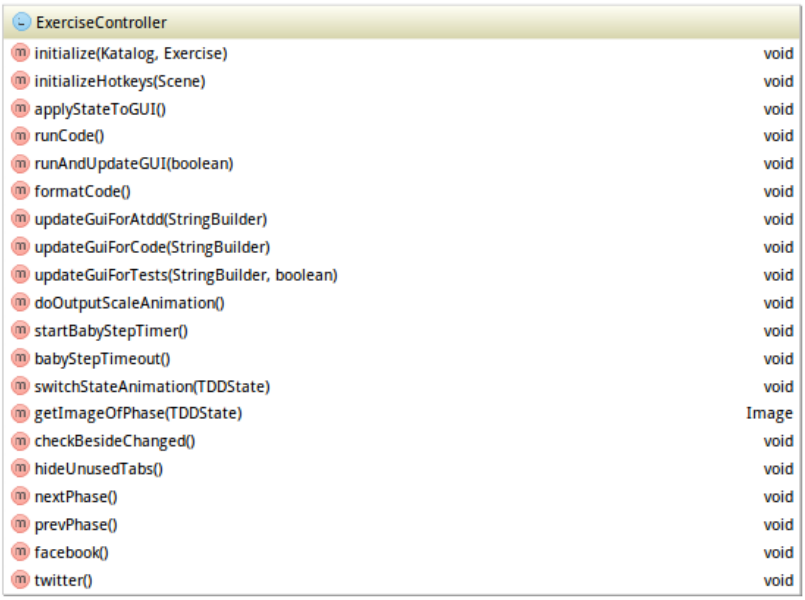
Die Methode *getState* gibt den aktuellen Status zurück, der als „*TDDState*“ gespeichert wird. Darin sind alle vier möglichen Stufen der Entwicklung abgespeichert, nämlich „Write Failing Acceptance Test“, „Write Failing Test“, „Make Test Pass“ und „Refactoring“.

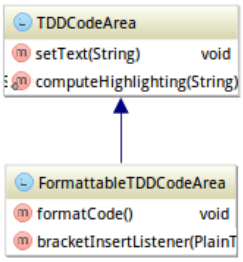
Die Methode *switchState* übergibt erst einmal zur Statusänderung den gewünschten Status an *switchStateInternal*. Diese Methode prüft, ob der Wechsel in Rücksicht auf Optionen sowie Kompilat möglich ist. Die letztendliche Entscheidung, ob die Möglichkeit bestand und auch sofort umgesetzt wurde, wird als ein Wahrheitswert zurückgegeben. Dann werden, falls *babySteps* aktiv sind und ein neuer akzeptiertes *CodeObject* entstanden ist, die Einträge des aktuellen *CodeObjects* in das *lastPassed* übernommen.

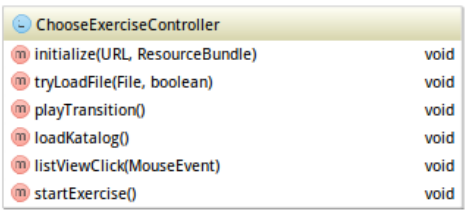
Danach folgen die Methode, die den ATDD-Test setzt (*setATDDTest*), wobei man einen String-Array mit *CompilationError*s übergeben bekommt, sollte etwas schief gegangen sein. Ist kein Fehler aufgetreten, so bekommt man einen null-Pointer.

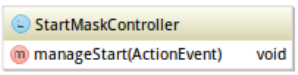
Nachfolgende wichtige Methoden sind noch *BabyStepBack()*, welche den letzten akzeptierten Code als aktuellen wieder einsetzt, wenn man die Zeit überschritten hat, und tryCompileTest() sowie *tryCompileCode()*, welche etwaige *CompileError*s zurückgeben, wobei keine *Error*s wieder als null ausgegeben werden. Die Methoden *isOneTestFailing()* (→ Gibt es überhaupt irgendeinen nicht akzeptierenden Test?) sowie *getFailingTests()* (→ Methodennamen nicht funktionierender Methoden in der Testklasse) geben Zustände über die Fehler zurück. In *getNextState()* bekommt man den nächsten Status im Entwicklungszyklus der testgetriebenen Entwicklung (je nach Optionen) ausgegeben.

GUI









Die GUI besteht aus 3 Scenes, die mithilfe von .fxml-Dateien definiert sind, und den jeweiligen Controllern: Die *StartScene* besteht lediglich aus einem StartButton (siehe Handbuch), welcher bei einem Klick die zweite Scene lädt, welche für die Auswahl der Übung da ist. Standardmäßig wird dabei versucht beim Start über das Interface Initializeable die Datei catalog.xml oder die Datei, die als 1. Aufrufparameter angegeben zu wurde, als Katalog zu laden. Dazu wird KatalogStore.lese(file) benutzt und die Namen der gelesenen Übungen in das ListView geschrieben. Im Falle eines Doppelklicks auf eine Übung oder beim Klick auf „Start Exercise“ wird das Editorfenster geladen. Dabei wird der geladene Katalog mit der jeweiligen Exercise vom ChooseExerciseController über die Methode Initialize(Katalog, Exercise) auf den ExerciseController weitergegeben, welcher anschließend ein logikHandler Objekt erstellt und als Instanzvariable gespeichert. Dieses LogikHandler Objekt nimmt wie bereits beschrieben der GUI (fast) die komplette Logik ab. Deswegen läuft die GUI im Groben immer wieder nach den selben Prinzip ab: Auf ein Event wird reagiert, indem die Daten durch Aufruf von Methoden an den LogikHandler weitergegeben wird und anschließend der Zustand im Logik-Objekt auf die einzelnen GUI-Elemente angewandt wird. Dabei haben wir versucht, die einzelnen Vorgänge in mehrere Methoden zu gliedern, um das ganze übersichtlich zu halten. So gibt runCode() z. B. den Code LogikHandler via setCode() setTest() etc. weiter und ruft anschließend die Methoden updateGuiForCode(StringBuilder), updateGuiForTests(StringBuilder), updateGuiForATDD(StringBuilder), welche jeweils die Inhalte und das Aussehen der Labels setzen und evtl. auftretende Fehler in den StringBuilder schreiben, dessen Text in die OutputTextbox geschrieben wird. Beim Schließen des Editorfensters erscheint dabei wieder die Auswahl der Übungen.
Der Texteditor wurde mithilfe von richtextfx umgesetzt, mit dessen Hilfe Codehighlighting realisiert wurde. Für die automatische Formatierung/Einrückung wurde Googles java-format Library verwendet.

Social Media-Integration

Während bei Twitter einfach der Standardbrowser aufgerufen werden konnte (in den man wahrscheinlich bereits eingeloggt ist) und man anschließend einen bereits vorgegebenen Text veröffentlichen konnte, so ist bei Facebook eine entsprechende Funktion leider nicht mehr verfügbar, da diese zum Spam missbraucht worden ist. Bei Facebook wäre es rein theoretisch möglich, über eine Facebook-App Posts zu erstellen. Allerdings müsste die App für die Berechtigung zum erstellen von Posts ein manuelles Review von Facebook durchlaufen, weswegen beim Klick auf den Facebook Button der Post in die Zwischenablage kopiert wird und der Nutzer diesen selber posten soll.