Alarm Clock

Jonathan Hyams
Pascal Schmalz

6. Juni 2017

Inhaltsverzeichnis

1 Zweck des Dokument

2 Architektur

3 Main

Dies ist die Klasse die die Main methode enthält. Sie erbt von der Klasse Application, da Sie das JavaFX-GUI launched.

```
1     Parent root = FXMLLoader.load(getClass().getResource(
          windowName));
```

windowName ist der Name der fxml Datei (mainWindow.fxml), in der das Aussehen und der Controller definiert wird. "mainWindow.fxml"befindet sich im package "resources".

4 Controller

5 Filtering

5.1 Architektur

Wir wollten nach dm Objektorientierten Paradigma den einzelnen Klassen möglichst wenig wissen über die Internas von anderen Klassen zumuten. Somit sollte der Reminder selber prüfen obe er eine bestimmte Bedinung erfüllt und eine Notification absenden soll, anstatt seine Innereine dem NotificationHandler zu offenbaren.

Es drängte sich also auf dem Reminder eine Funtkiont zum testen zu übergeben. Ähnlich wie im Comand Pattern haben wir dies gelöst, indem wir Ein Objekt um die Funktion gewrappt haben. Anstatt execute() haben wir die funktion aber isTrue genannt, was ein gut lesbaren Quellcode mit den Tests erzeugt. Wie zum Beispiel der Criteriatester IsThisYear welcher einen Remeinder darauf testet, ob er in diesem Jahr ist und die Antwort als boolean zurück gibt.

```
1 boolean isThisYear = IsThisYear.isTrue(reminder);
```

5.2 Higher Order Functions

Die Marketingabteilung von Oracle behauptet gerne Java seie auch Funktional. Higer Order Functions werden aber nicht wirklich unterstützt. TODO wikipedia Higer Order function. Funktionen als Return Values Java unterstüzt leider keine Higher order functions. Man kann also keine Funktion als Inputparameter übergeben. Mittels Lambdas ist es lediglich möglich, die eine Funktion ausführen zu lassen und den Rückgabewert als Inputparameter weiter zu verwenden. Dies erlaubt eine kompaktere Notation. Dies reicht uns aber nicht, da wir den Remindern eine Funktion übergeben möchten, mit welcher jeder Reminder selber testet ob er eine Notification absenden soll.

5.3 echte Higher Order Functions in Java

Um dies zu erreichen haben wir eine Form von Higher Order Functions mit Objekten nachgebaut. Ein CriteraTester ist ein Objekt, welches als Wrapper für eine Funtion dient. Anstelle dieser Funktion übergibt man nun diesen FunktionsWrapper als Inputparameter. Somit konnten wir Funktionen als Inputparameter mittels Objektorientierten prinzipien nachbauen. Man muss nun für jede Funktion ein Objekt erstellen, welches das Interface CriteraTester implementiert.Und die Filterfunktion isTrue implementieren. Funktionen als Rückgabewert kann man so aber noch nicht wirklich nachbauen. Für uns war das aber nicht nötig.

Dank den oben erwähnten Lambdas kann man dies auch elegan on the Fly erledigen. Da es aber vorkommen kann dass man einen Filter mehrmals benutzt, habe wir uns entschieden die Filter jeweils als eigene Klassen zu implementieren.

5.4 Code Beispiel

```
Reminder r;
Collection<CriteriaTester> criteria = new ArrayList<>();
 criteria.add(new IsPassed());
  //     example how CriteraTester can be written on the
      fly
 //pus this to documentation
 criteria.add(
         r -> (!r.getTags().contains("hidden"))
 );
 //This lets the Reminder send a notification if the
    Reminder meets the criterias
 //The first criteia it must pass it th IsPassed()
 //the second criteria is defined on the fly on  line
    number TODO x. it tests if it contains the tag "
    hidden"


 r.notifyIf(criteria);
```

6 Notification

7 Resources

8 Tests

9 Tools

Um die Versionierung der Dokumentation automatisch generieren zu lassen, haben wir LaTeX so mit Scripts erweitert, so dass die git Head Versionsnummer direkt ins Dokument eingefügt wird. Somit bleibt diese Information auch auf einem Ausdruck akkurat. Gegenüber einer manuellen inkrementierung der Version, hat die automatisierung den Vorteil, dass sie auch im Stress nicht vergessen wird. Das Ergebniss sieht man am Ende des Dokuments.

LaTeXkann Code ausführen. wir haben den Code in Shellscripte ausgelagert, und lassen den compiler diese aufrufen. Damit dies funktioniert, muss dies aktiviert werden.

Im LaTeXfile steht nun folgender Code. Zuerst wird die Versionsinformation in eine Datei geschrieben, welche anschliessend in die Dokumentation eingebunden wird. Am schluss wird die Datei zurückgesezt, so dass sie eine Fehlermeldung im Dokument erzeugt, falls die Ausführung von Shellscripten im LaTeXcompiler ausgeschaltet ist. Die automatisch generierte Datei wird nach der Generierung in das Dokument eingebunden.

```
 \noindent
Automatische Versionierung:
\immediate\write18{../script/versionInfo.sh}
\input{../script/version}
\immediate\write18{../script/cleanup.sh}
```

Wir schreiben den output in eine version.tex Datei. Auf Zeile 9 lassen wir git HEAD nummer in die Datei speichern, welche beim pushen jeweils inkrementiert wird.

```
#!/bin/sh
OUTPUT="../script/version.tex"

echo "Last compiled: ">$OUTPUT
date >> $OUTPUT

echo "\n">>$OUTPUT

echo "Git HEAD Version: ">> $OUTPUT
git rev-list --count --first-parent HEAD >>$OUTPUT
```

Dann wird ein cleanup durchgeführt, dabei wird die Output Datei mit einer Fehlermeldung versehen, so dass der User bemerkt, falls die Automatische Versionierung fehlschlägt.

```
#!/bin/sh
OUTPUT="../script/version.tex"
echo "Fetching version information failed. Please enable shell-escape in your \LaTeX \~ compiler.">$OUTPUT
```

10 Versionskontrolle

Manuelle Version: 1.0.0

Automatische Versionierung: Last compiled: Tue Jun 6 15:02:11 CEST 2017
Git HEAD Version: 187