

Alarm Clock

Jonathan Hyams
Pascal Schmalz

2. Juni 2017

Inhaltsverzeichnis

1	Zweck des Dokument	3
2	Architektur	3
3	Controller	3
4	Filtering	3
4.1	Architektur	3
4.2	Higher Order Functions	3
4.3	echte Higher Order Functions in Java	3
4.4	Code Beispiel	4
5	Notification	4
6	Resources	4
7	Tests	4
8	Tools	4
9	Main	4
10	Versionskontrolle	4

1 Zweck des Dokument

2 Architektur

3 Controller

4 Filtering

4.1 Architektur

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

Es drängte sich also auf dem Reminder eine Funktion zum Testen zu übergeben. Ähnlich wie im Command Pattern haben wir dies gelöst, indem wir ein Objekt um die Funktion gewrappert 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 Reminder darauf testet, ob er in diesem Jahr ist und die Antwort als boolean zurück gibt.

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

4.2 Higher Order Functions

Die Marketingabteilung von Oracle behauptet gerne Java sei auch Funktional. Higher Order Functions werden aber nicht wirklich unterstützt. [TODO wikipedia Higher Order function](#). Funktionen als Return Values Java unterstützt 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.

4.3 echte Higher Order Functions in Java

Um dies zu erreichen haben wir eine Form von Higher Order Functions mit Objekten nachgebaut. Ein `CriteriaTester` ist ein Objekt, welches als Wrapper für eine Funktion 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 `CriteriaTester` 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 elegant auf der Fly erledigen. Da es aber vorkommen kann dass man einen Filter mehrmals benutzt, haben wir uns entschieden die Filter jeweils als eigene Klassen zu implementieren.

4.4 Code Beispiel

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

5 Notification

6 Resources

7 Tests

8 Tools

Um die Versionierung der Dokumentation

9 Main

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

```
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".

10 Versionskontrolle

Manuelle Version: 1.0.0

Automatische Versionierung: Last compiled: Fri Jun 2 23:07:01 CEST 2017
Git HEAD Version: 174