

Alarm Clock

Jonathan Hyams
Pascal Schmalz

6. Juni 2017

Inhaltsverzeichnis

1	Zweck des Dokument	3
2	Architektur	3
3	Main	3
3.1	Class: Main	3
4	Controller	3
5	Filtering	3
5.1	Architektur	3
5.2	Higher Order Functions	4
5.3	echte Higher Order Functions in Java	4
5.4	Code Beispiel	4
5.5	Implementierungen von Filtern	5
6	Notification	5
6.1	Interface: Notification	5
6.2	Class: ConsoleNotification	5
6.3	Class: JavaFxNotification	5
6.4	Class: MultiReminderNotification	6
6.5	Class: NotificationHandler	7
6.5.1	Methode showPastEvents	8
7	Resources	9
8	Tests	9
9	Tools	9
10	Versionskontrolle	10

1 Zweck des Dokument

2 Architektur

3 Main

3.1 Class: Main

Dies ist die Klasse die die Main methode enthält. Sie erbt von der Klasse Application, da Sie das JavaFX-GUI launched. Application implementiert die start methode, die ein Stage als Parameter hat. Diese Stage ist das Hauptfenster mit der Tabelle von Reminders.

```
1 Platform.setImplicitExit(false);
```

Wenn Platform.setImplicitExit true ist, werden Befehle wie Platform.runlater() ignoriert, sobald alle JavaFX Fenster geschlossen sind. Das würde heissen, das wenn das Hauptfenster geschlossen wurde (und alle Popups), würden spätere Popups nicht mehr erscheinen. Dies wollen wir verhindern, also ist es auf false gesetzt.

```
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".

```
1     if (new ConfigReader().isEnabledDarkMode()) {
        scene.getStylesheets().add("dark.css");
    } else {
        scene.getStylesheets().add("styles.css");
5    }
```

Der ConfigReader kann so angepasst werden, das man hier das gewünschte CSS File bekommt.

4 Controller

5 Filtering

5.1 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 Innereine dem NotificationHandler zu offenbaren.

Es drängte sich also auf dem Reminder eine Funktione zum testen zu übergeben. Ähnlich wie im Command Pattern haben wir dies gelöst, indem wir ein Objekt um die Funktion gewrappet 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 sei auch Funktional. Higer Order Functions werden aber nicht wirklich unterstützt. TODO wikipedia Higer 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.

5.3 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 the 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.

5.4 Code Beispiel

```
1      Reminder r;  
      Collection<CriteriaTester> criteria = new ArrayList<>();  
      criteria.add(new IsPassed());  
      //      example how CriteriaTester can be written on the  
          fly  
5      //pus this to documentation  
      criteria.add(  
          r -> (!r.getTags().contains("hidden"))  
      );  
      //This lets the Reminder send a notification if the  
          Reminder meets the criterias  
10     //The first criteria 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);
```

5.5 Implementierungen von Filtern

Wir haben etliche Filter implementiert, der

6 Notification

6.1 Interface: Notification

Das NotificationInterface gibt zwei Methoden vor:

```
1 void setReminder(Reminder reminder);
```

setReminder übergibt den Reminder, für den man eine Notification erstellen will.

```
1 void send();
```

send() wird vom Reminder aufgerufen, und zeigt dem User die Notification. Es gibt verschiedene Arten von Notifications. JavaFxNotification ist das Popup das den Reminder aufzeigt, der gerade aktuell ist. MultireminderNotification zeigt alle schon vergangenen Reminders, und ConsoleReminder zeigt auf der Konsole (Shell) einen Reminder. All diese Notification-Klassen implementieren Notification.

6.2 Class: ConsoleNotification

ConsoleNotification hat einen leeren Konstruktor aus mehreren Gründen. Wir haben ihn ähnlich wie den JavaFxNotification aufgebaut, und JavaFX verlangt einen leeren Konstruktor, also haben wir ihn hier beibehalten. Auch haben wir einen Konstruktor mit einem Reminder im Parameter gemacht, also war der DefaultKonstruktor überschrieben. Der ConfigReader benutzt diese Klasse auch (inklusive dem leeren Konstruktor) und benötigt keinen direkten Reminder, also haben wir ihn so stehen lassen.

Der Konstruktor mit dem Reminder im Parameter und die setReminderMethode machen eigentlich genau das gleiche. setReminders wird noch vom Notification Interface verlangt. Die send() Methode wird vom Reminder aufgerufen wenn die Zeit soweit ist. Sie druckt einfach die toString() Methode des Reminders auf die Konsole.

6.3 Class: JavaFxNotification

Die JavaFxNotification Klasse ist identisch mit der ConsoleNotification, nur die send() Methode ist anders. Die send() Methode enthält die statische Methode:

```
1 Platform.runLater(() -> {...});
```

In den geschweiften Klammern wird ein Popup Fenster erstellt. Da wir mit Threads arbeiten, und diese Popups verspätet aufgerufen werden, müssen wir das GUI in das Platform.runlater einpacken. Die JavaDoc vom runlater() sagt 'Run the specified Runnable on the JavaFX Application Thread at some unspecified time in the future [...]' und das ist genau was wir brauchen. Lässt man es weg, bekommt man dutzende von Exceptions.

```

1  {
    Stage stage = new Stage();
        label = new Label("Hello: " + reminder.toString());
        Button okButton = new Button("Ok");
5   okButton.setOnAction(e -> {
            stage.close();
        });
    VBox pane = new VBox(10, label, okButton);
    pane.setAlignment(Pos.CENTER);
10  pane.setPadding(new Insets(10));
    Scene scene = new Scene(pane);
    stage.setTitle("Reminder");
    stage.setScene(scene);
    stage.setResizable(false);
15  stage.show();
    }

```

Der Code in den geschweiften Klammern macht ein simples JavaFX Fenster das den Reminder aufzeigt. Das Label wird mit der reminder.toString() Methode überschrieben. Dem okButton schliesst das Fenster wenn man ihn drückt. Die Komponenten Button und Label tun wir in ein VBox Behälter und machen ihn noch ein bisschen schöner mit setAlignment() und setPadding().

6.4 Class: MultiReminderNotification

MultiReminderNotification benutzen wir um alle schon vergangenen Reminder in einem einzigen Fenster darzustellen. Es ist aber auch möglich, eine andere aggregation von Remindern mit ihr darzustellen. Der Aufbau der Klasse ist genau gleich wie in JavaFxNotification und ConsoleNotification, nur dass anstatt einem Reminder hat er eine Liste von Reminder.

```

1  private Collection<Reminder> reminders;

    String remindersText = "";
        int i = 0;
5   for (Reminder r : reminders) {
        remindersText += "Passed Event No " +
            ++i + ":\n";
        remindersText += r.toString() + "\n";
    }

```

```

        System.out.print("added" + r.toString()
        );
    }

```

Hier iterieren wir durch alle Reminders in der Tabelle und fügen sie zum Label hinzu. Die Reminders sind in der reminders Liste. Damit wir nicht alle Reminders in diesem Popup haben, sondern nur die die bereits vergangen sind oder schon sehr bald erscheinen, werden diese im NotificationHandler noch gefiltert. Die Methoden dazu wären folgende:

```

1 criteria.add(new IsPassed());
  criteria.add(new IsThisYear());

```

6.5 Class: NotificationHandler

Der NotificationHandler handelt die Notifications. Dazu werden die einzelnen NotificationTypen mit den passenden CriteriaTesters konfiguriert. Im Konstruktor wird dem NotificationHandler eine ReminderList übergeben. Für diese Reminders werden beim aufruf der handel() methode die Notifications verwaltet.

in der handle() Methode wird über die ReminderList iteriert. für jeden NotificationTyp werden nun die passenden CriteriaTester angegeben. Wir betrachten nun lediglich den NotificationTyp, welcher sämtliche Reminders aufpoppen lässt, welche diesen Monat sind.

Listing 1: NotificationHandler.handle

```

1 Collection<CriteriaTester> importantStuffThisMonth = Arrays.
  asList(new IsThisMonth());
  if (!notifiedReminders.contains(reminder)) {
    /**
     * this passes the criteriaTesters to the Reminder itself, and
     * lets the Reminder send the notification if
5    * the criteria are met.
     */
    boolean success = reminder.notifyIf(importantStuffThisMonth
    );
    if (success) notifiedReminders.add(reminder);
  }

```

Dann wird für jeden Reminder, welcher noch keine Notification vom entsprechenden Typ abgesendet hat ein Reminder.notifyIf(CriteriaTester) aufgerufen. Der Reminder testet selbstständig, ob das Kriterium zutrifft, und er der Notification den Befehl gibt eine Meldung azusende. Falls dies geschieht, meldet der Reminder ein success zurück. Der Handler nimmt ihn dann in die Liste der Reminder auf, welche bereits eine Notification abgesendet haben.

Kurz vor dem Datum eines Reminders, wird nochmals auf diesen Reminder hingewiesen. Der Code funktioniert sehr ähnlich. Es wird aber anstatt ein CriteriaTester.IsThisMonth() ein CriteriaTester.IsNextSecond() übergeben.

6.5.1 Methode showPastEvents

Es gibt noch ein dritten Typ von Notifications diese zeigt eine Aggregation der Vergangenen Remindern. Sie werden in einer separaten Methode abgearbeitet. Der showPastEvents() Methode. Diese löst eine Notification aus, welche mehrere Reminders zusammen darstellt. Deswegen haben wir uns von dem Paradigma gelöst, dass nur der Reminder die Notification auslöst. Die Methode überprüft auf zwei Kriterien. Erstens ob der Reminder in der Vergangenheit angesiedelt ist und ob der Reminder in diesem Jahr war. Siehe Zeile 8 und 9. Dies geschieht indem man über die ReminderList iteriert, und die passenden Reminders in eine separate Liste namens passedReminders speichert. Diese wird dann der MultiReminderNotification übergeben, damit diese die aggregierte Notification absenden kann.

Listing 2: NotificationHandler.showPastEvents

```
1 public void showPastEvents() {
    ArrayList<Reminder> reminderList = reminders.
        getSerializable();
    ArrayList<Reminder> passedReminders = new ArrayList<>()
        ;
    Collection<CriteriaTester> criteria = new ArrayList<>()
        ;
5    /**
     * the both criteria filter the Reminders for Reminders
     * , which are dated in the past and dated this year.
     */
    criteria.add(new IsPassed());
    criteria.add(new IsThisYear());
10
    for (Reminder reminder : reminderList) {
        if (reminder.meetsCriteria(criteria))
            passedReminders.add(reminder);
    }
15    // gets sure that a Notification is only sent, if it is
    // not void.
    if (passedReminders.size() != 0) {
        new MultiReminderNotification(passedReminders).send
            ();
    }
}
```

Um sicherzustellen, dass die showPastEvents nur einmal dargestellt werden, muss der Poller sich merken, ob die Methode schon einmal aufgerufen wurde. Falls dies zutrifft, wird auf ein weiterer Aufruf verzichtet.

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.

L^AT_EX kann Code ausführen. Wir haben den Code in Shellscripte ausgelagert, und lassen den compiler diese aufrufen. Damit dies funktioniert, muss dies aktiviert werden.

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

```

1 \noindent
  Automatische Versionierung:
  \immediate\write18{../script/versionInfo.sh}
  \input{../script/version}
5 \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.

```

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

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

  echo "\n">>$OUTPUT

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

```

1 #!/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 16:20:57 CEST 2017
Git HEAD Version: 189