Alarm Clock

Jonathan Hyams
Pascal Schmalz

6. Juni 2017

Inhaltsverzeichnis

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().isEnableDarkMode()) {
           scene.getStylesheets().add("dark.css");
       } else {
           scene.getStylesheets().add("styles.css");
5      }
```

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

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.

```
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

6.1 Interface: Notification

Das NotificationInterface gibt zwei Methoden vor:

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

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

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

send() wird vom Reminder aufgerufen, und zeigt dem User die Notification. Es gibt verschiene 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 implemenetieren Notification.

6.2 Class: ConsoleNotification

ConsoleNotification hat einen leeren Konstruktor aus mehreren Gründen. Wir haben ihn ähnlich wie den JavaFxNotification aufebaut, und JavaFX verlangt einen leeren Konstuktor, 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 eingentlich genau das gleiche. setReminders wird noch vom Notication 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.

```
  {
Stage stage = new Stage();
                    label = new Label("Hello: " + reminder.
                        toString());
                    Button okButton = new Button("Ok");
                    okButton.setOnAction(e -> {
                        stage.close();
                    });
                    VBox pane = new VBox(10, label, okButton);
                    pane.setAlignment(Pos.CENTER);
                    pane.setPadding(new Insets(10));
                    Scene scene = new Scene(pane);
                    stage.setTitle("Reminder");
                    stage.setScene(scene);
                    stage.setResizable(false);
                    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 zeigt alle schon vergangenen Reminder in einem einzigen Fenster an. Der Aufbau der Klasse ist genau gleich wie in JavaFxNotification und ConsoleNotification, nur dass anstatt einem Reminder hat er eine Liste von Reminder.

```
private Collection<Reminder> reminders;

String remindersText = "";
                    int i = 0;
                    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 habem, sonder nur die die bereits vergangen sind oder schon sehr bald erscheinen, werden diese im NotificationHandler noch gefiltert. Die Methoden dazu wären folgende:

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

6.5 Class: NotificationHandler

TODO

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.

```
#!/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

```
#!/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 6 Jun 15:02:54 CEST 2017
Git HEAD Version: 176