**Protokoll Übung 1**

Gruppe: **Schwabakla**

– Robert Dziuba, Tobias Klatt & Inga Schwarze

1. **Properties implementieren**

Aufgabenteil A:

Wir schreiben eine Klasse Appointment, die folgende Attribute als StringProperties erhält: datum, titel, startzeit, endzeit, kategorie, notiz. Die Klasse Zeitrechner ist unsere Kontrollklasse, die die Zeiten überprüft. Neue Objekte von StringProperties werden mit SimpleStringProperty initialisiert.

Properties haben den Vorteil, dass wir im späteren Verlauf in unserem Programm mit diesen arbeiten können und das ermöglicht uns, diese Attribute zu „beaobachten“ (> Observable) und miteinander zu „verbinden“ (> Binding). Dadurch kann der Wert des Attributes an eine andere Property gebunden werden und wird somit direkt und ständig synchronisiert (bei jeder Änderung).

Wir haben drei Konstruktoren: einen Standartkonstruktor (leer), einen Kopierkonstruktor und einen, der ein Appointment initialisiert und alle Attribute setzt.

Wir haben Getter und Setter, die wir zwingend für diesen ADT brauchen. Wir setzen also unsere Attribute damit fest.

Getter: Setzen

Setter: Holen

Mit diesen können wir die Werte verändern. > Inhalt der Properties verändern.

Property: Hier werden die Properties direkt übergeben. Da wir Sachen herausgeben, nennen wir sie Getter – StringProperty. > Gesamte Klasse Property mit Inhalt wird zurückgegeben. Das ist später nützlich, da diese Klasse z.B. das Binding enthält.

To String Methode: Um den ADT auszugeben. In der Konsole erscheint dann eine Übersicht unseres Termineintrags.

Unsere Setter sorgen dafür, dass die eingegebenen Daten überprüft werden. Wir haben dazu Methoden geschrieben. Datum: testeDatumString() überprüft, ob ein Datum vorliegt und ob dieses im richtigen Format (testeDatumFormat()) ist. Außerdem überprüfen wir ob unser Datum im Wertebereich liegt (testeDatumWertebereich()). Analog dazu testen wir auch die Zeit: testeZeitString(), testeZeitFormat(), testeZeitWertebereich().

Aufgabenteil B:

Bei den Settern kontrollieren wir zusätzlich, ob die Endzeit tatsächlich hinter der Startzeit liegt und andersherum (Aufgabenteil B). Die Methoden heißen testeZeitFenster().

Es wird direkt eine Exception geworfen, wenn die Startzeit nicht vor der Endzeit liegt.

1. **Editierbare ListView**

Aufgabenteil A:

Wir schreiben eine neue Klasse ObservableContactDetails, die von unserer Klasse ContactDetails erbt. Wir haben fünf Attribute vom Typ StringProperty: vorname, nachname, adresse, telefonnummer, mail. Die Klasse hat dieselben Eigenschaften wie unsere ContactDetails, bis auf die Tatsache, dass es sich nun um SimpleStringProperties handelt, damit wir später mit Binding arbeiten können.

Drei Konstruktoren: Standartkonstruktor, der ruft unseren normalen Konstruktor auf. Der normale K. ruft zusätzlich den K. von ContactDetails auf. Der Kopierk. Ruft den normalen K. auf. Die Getter geben die Werte für unsere Attribute, die Setter setzen die Attribute für die Property. Auch hier haben wir wieder die Möglichkeit, die Klasse Property/ das Objekt zurückzugeben. > StringProperty vornameProperty > return vorname;

Außerdem wie gewöhnlich eine ToString-Methode und eine Main-Methode zum Ausführen, um einen Überblick über die Funktion der Klasse zu bekommen.

Aufgabenteil B:

Aufgabenteil C:

Aufgabenteil D:

1. **Editierbare Tabelle**

Aufgabenteil A:

Idee: Wir trennen unsere Klassen nach dem MVC Prinzip. Das bedeutet, wir haben folgende Klassen:

View: Für die grafische Oberfläche, ruft Control auf

Control: Für die Logik des Programms, füllt die View

Die View-Klasse erzeugt eine BorderPane und ein Control-Objekt. Control ruft dann die Methode fuelleTabelle() auf, die in die Control-Klasse ausgelagert wird (> Logik!). Wir erzeugen außerdem eine Scene, füllen sie mit der BorderPane und legen eine feste Größe fest. Der Rest der Klasse erschließt sich von selbst (Main-Methode mit launch-Methode etc.).

Die Control-Klasse: Da wir hier auch mit dem AddressBook arbeiten und wir bisher nur die ContactDetails „observable“ gemacht haben, müssen wir nun auch unsere Klassen AddressBook und das dazugehörige Interface AddressbookInterface anpassen. Wir wandeln in beiden Klassen alle ContactDetails in ObservableContactDetails um. Unsere Attribute in der Control-Klasse sind nun:

TableView <ObservableContactDetails> tabelleFürKontakte = **new** TableView();

ObservableList <ObservableContactDetails> kontakte = FXCollections.*observableArrayList*();

BorderPane pane;

AddressBook book = **new** AddressBook();

Im Konstruktor übergeben wir die zuvor initialisierte BorderPane und rufen die Methode fuelleAdressBook() auf, damit dies stets zu Programmstart ausgeführt wird. Sie füllt unser Adressbuch mit Dummy-Werten.

In der Methode fuelleTabelle() erzeugen wir nun unsere Tabelle und erkennen den Vorteil von Properties.

Doch zunächst setzen wir ein Array von ObservableContactDetails auf null. Wir versuchen dann, alle eingetragenen Kontakte darin zu speichern und dann zu ObservableContactDetails hinzuzufügen. Wichtig: die setEditable-Methode, um die Einträge dann auch wirklich editieren zu können. Mit der setItems-Methode packen wir die Kontakte in die TableView.

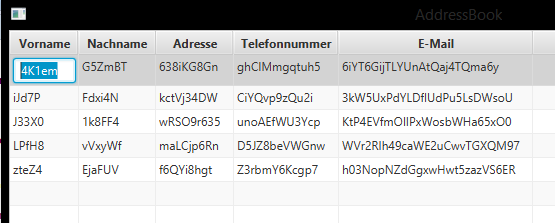
Wir müssen nun noch die Spalten der Tabelle erzeugen. Dies geschieht analog für jedes Attribut nach folgendem Schema:

TableColumn<ObservableContactDetails, String> vornameSpalte = **new** TableColumn<>("Vorname");

vornameSpalte.setCellValueFactory(**new** PropertyValueFactory<ObservableContactDetails, String>("vorname"));

vornameSpalte.setCellFactory(TextFieldTableCell.*forTableColumn*());

Die Spalten übergeben wir letztlich ganz einfach mit der addAll()-Methode. Wir können unsere View-Klasse nun ausführen und die Einträge erfolgreich ändern/umspeichern.



Aufgabenteil B:

Wir erstellen einen Knopf „Neu“, der mit einem einfachen Klick einen neuen Eintrag zum Editieren erzeugt. Im Konstruktor erzeugen wir dazu zunächst ein neues Objekt vom Typ Button und eine HBox, die später, zusammen mit dem „Drucken“-Knopf, beide Knöpfe anzeigen soll. Funktion fehlt noch.. Wie macht man diesen Knopf?

Aufgabenteil C:

Wir erstellen nun noch einen Knopf „Drucken“, der den Inhalt der Adressliste in der Konsole ausgibt. Wir erzeugen also wieder ein neues Objekt vom Typ Button und packen diesen in unsere bereits erzeugte HBox. Mithilfe der setOnMouseClicked-Methode schreiben wir einen Lambda-Ausdruck, in dem wir mit Syso den Inhalt unseres Adressbuchs ausgeben. Wichtig: Wir haben in unserer Klasse AddressBook eine ToString-Methode geschrieben, die diesen Inhalt übersichtlich und korrekt darstellt. Wir nutzen dafür einen StringBuilder.

Aufgabenteil E:

Frage muss noch beantwortet werden.. Vielleicht machen wir das zusammen?

Was mir noch aufgefallen ist: In Unserer Control ist nun eigentlich auch etwas Grafik enthalten. Die Buttons.. sollten wir die vielleicht besser in die View packen oder geht das nicht? Und noch etwas. Wir hatten anfangs definiert dass wir auch eine TableCell Klasse brauchen. Diese haben wir aber nun gar nicht geschrieben. Richtig so?