

Programmierung II – Übung 8

In dieser Übung soll eine GUI für das Anlegen/Editieren eines Termins eingebaut werden. Außerdem soll es, auf der Basis der Serialisierung von Objekten, möglich sein die Inhalte des Terminplaners in einer Datei zu speichern und diese auch wieder in den Terminplaner einzuladen.

Arbeiten Sie die Änderungen in Ihr Terminplaner-Projekt aus der letzten Übung ein. Importieren Sie zunächst alle Java-Dateien aus dem Ordner *Sourcen* in das Paket *terminverwaltung*.

Speichern und Laden von Terminen in/aus einer Datei

Erweitern Sie die Klasse *Terminplaner* um zwei Methoden:

```
public void save (File file) throws IOException
```

Die Methode soll alle Termine des Planers in der übergebenen Datei in serialisierter Form abspeichern. Hierbei eventuell auftretende Exceptions sollen an den Aufrufer (das wird der *PlanerViewController* sein) weitergeleitet werden, um dort in einem Fehlerdialog angezeigt werden zu können.

```
public void load (File file) throws Exception
```

Diese Methode soll versuchen, aus der angegebenen Datei alle Termine einzulesen. Sollte dies erfolgreich gelaufen sein, so soll sie den Terminplaner zurücksetzen (Methode *initialisieren()*) und die eingelesenen Termine hinzufügen. Hierbei eventuell auftretende Exceptions sollen an den Aufrufer weitergeleitet werden.

Die beiden Methoden *save()* und *load()* sollen im *PlanerViewController* genutzt werden um die in den Menüpunkten *Laden* und *Speichern* hinterlegten Event Handler umzusetzen.

Schreiben Sie im *PlanerViewController* zunächst den Rumpf der Methode *saveTermine()*. Öffnen Sie hier einen *FileChooser*, und versuchen Sie in der ausgewählten Datei die Termine des Terminplaners zu speichern. Fangen Sie hier die eventuell auftretende *IOException* und zeigen Sie mit Hilfe der Klasse *ViewHelper* die Fehlermeldung „Datei konnte nicht gespeichert werden.“ an.

Probieren Sie zunächst aus, ob das Speichern von Terminen funktioniert, in dem Sie versuchen, die Termine in einer Datei *Termine.ser* zu speichern.

Schreiben Sie nun im *PlanerViewController* den Rumpf der Methode *loadTermine()*. Öffnen Sie auch hier einen *FileChooser*, und versuchen Sie aus der ausgewählten Datei die Termine des Terminplaners zu laden. Fangen Sie hier die eventuell auftretende *Exception* und zeigen Sie mit Hilfe der Klasse *ViewHelper* die Fehlermeldung „Datei konnte nicht geladen werden.“ an. Sorgen Sie dafür, dass die Termine des aktuell ausgewählten Datums angezeigt werden. Schauen Sie nach, ob es eine Methode im *PlanerViewController* gibt, die Sie für diese Aufgabe nutzen können.

Probieren Sie aus, ob Sie die gespeicherten Termine wieder einlesen können. Dazu sollten Sie im Konstruktor der Klasse *Terminplaner* den Aufruf der Methode *setTestData()* zunächst einmal auskommentieren, so dass initial kein Termin im Planer ist.

Anlegen/Bearbeiten von Terminen

Die View für das Anlegen und Bearbeiten von Terminen ist bereits angelegt, Sie finden Sie im Ordner *Sourcen* unter dem Namen *terminView.fxml*. Die Termin-View soll die Informationen über einen Termin anzeigen, wenn vorher in der Planer-View ein Termin ausgewählt wurde. Ansonsten sollen die Felder leer bleiben. Sehen Sie sich im *SceneBuilder* die View an und machen Sie sich mit den GUI-Komponenten vertraut. Sehen Sie sich besonders die Komponenten an, die mit einer *fx:id* versehen wurden (z.B. in der Auflistung unter Controller unten links).

Auch der zugehörige Controller ist in der Klasse *TerminViewController* schon als Fragment angelegt.

Beginnen Sie damit, in der Klasse *PlanerViewController* die nötigen Methoden für den Umgang mit der Bearbeitung bzw. dem Editieren von Terminen zu schreiben:

Schreiben Sie den Rumpf der Methode *addTermin()*. Diese wird aufgerufen, wenn der Add-Button in der Planer-View gedrückt wurde. Hier soll mit Hilfe der Klasse *ViewHelper* die Termin-View für das Anlegen eines neuen Termins geöffnet werden. Um dem *TerminViewController* mitzuteilen, dass es sich um das Anlegen eines neuen Termins handelt, müssen Sie ihm im Konstruktor den Wert *null* als Termin übergeben. Der *TerminViewController* bekommt außerdem das *PlanerViewController*-Objekt (*this*) übergeben, da diesem per Methodenaufruf später der neue Termin übergeben werden muss, damit er ihn im Terminplaner abspeichert.

Schreiben Sie nun den Rumpf der Methode *editTermin()*. Lesen Sie aus der *ListView* den selektierten Termin aus. Prüfen Sie, ob ein Update dieses Termins erlaubt ist (Methode *updateErlaubt()* von *Terminplaner*). Wenn dies so ist, so geben Sie beim Erzeugen des *TerminViewController*s im Konstruktor den Termin und das *PlanerViewController*-Objekt (*this*) an. Sollte das Editieren nicht erlaubt sein, so übergeben Sie dem *TerminViewController* anstelle des *PlanerViewController* Objekts den Wert *null*. Der Wert *null* soll dem Controller signalisieren, dass der Termin in der View nicht geändert werden darf.

Legen Sie im *PlanerViewController* eine öffentliche Methode

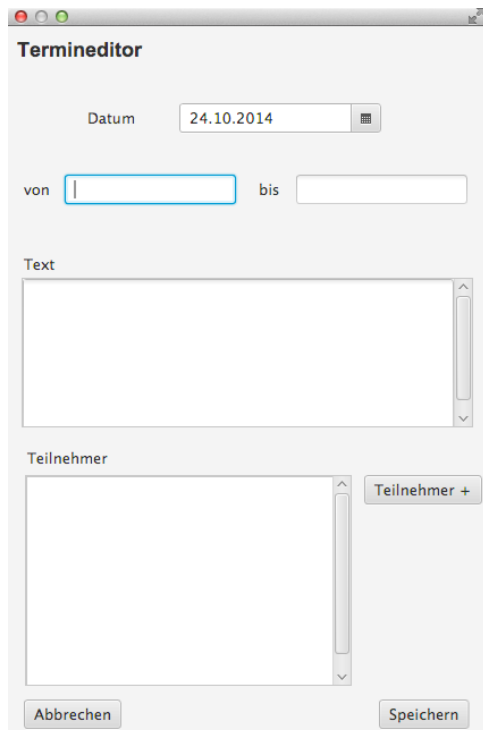
void processTermin(Termin t) throws TerminUeberschneidungException

an, der ein editiertes/neu angelegtes *Termin*-Objekt übergeben wird. Die Methode soll den Termin an den Terminplaner zum Aktualisieren übergeben (Methode *setTermin(..)*). Falls es dabei zu einer *TerminUeberschneidungException* kommen sollte, soll diese an den Aufrufer (das wird der *TerminViewController* sein) weitergeworfen werden. Sorgen

Sie dafür, dass nach dem Aktualisieren des Terminplaners die Termine neu in der Planer-View angezeigt werden.

Die Klasse *TerminViewController*

Der *TerminViewController* steuert die Anzeige eines Fensters für das Editieren eines bereits vorhandenen Termins, das Anzeigen eines Termins, der nicht editiert werden darf und für das Anlegen eines neuen Termins. Ergänzen Sie zunächst den Rumpf der Methode *inititalize()*. Je nach Situation muss das Fenster unterschiedlich eingestellt werden, nutzen Sie für jede Situation eine eigene Methode:



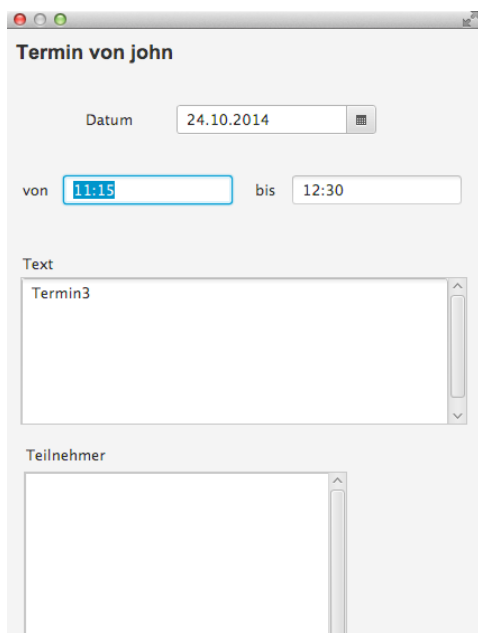
1. Editieren eines neuen Termins (Methode *initNewTermin()*)

Die View wird so eingestellt, wenn das Attribut *termin* den Wert *null* und *controller* einen Wert ungleich *null* hat. Hier muss folgendes in der Ausgangs-View verändert werden:

Titel: Termineditor

Datum: wird auf das aktuelle Datum gestellt

Save-Button: bekommt die Aufschrift *Speichern*



2. Editieren eines vorhandenen Termins des Planer-Besitzers (Methode *initUpdateTermin()*)

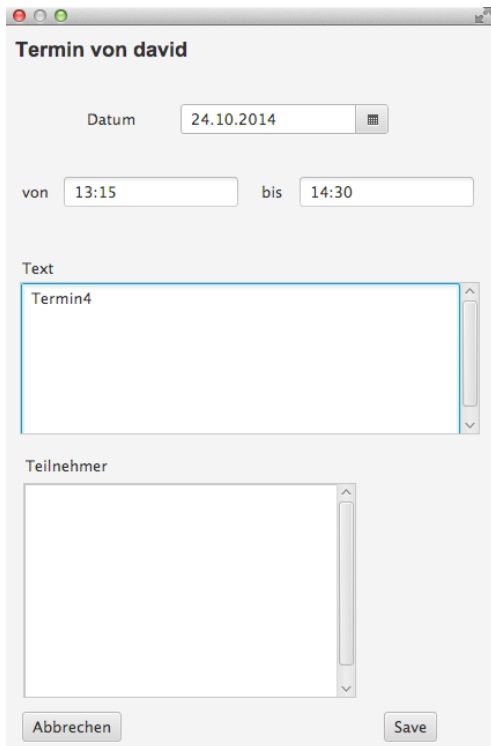
Die View wird so eingestellt, wenn die Attribute *termin* und *controller* einen Wert ungleich *null* haben. Hier muss folgendes in der Ausgangs-View verändert werden:

Titel: Dem vorhandenen Titel muss der Name des Termin-Besitzers angehängt werden

Datum: wird auf das Datum des Termins gestellt

Teilnehmer-Button: wird deaktiviert. Alle anderen Felder müssen mit den Inhalten des Termin-Objekts gefüllt werden.

3. Anzeigen eines fremden Termins (anderer Besitzer) (Methode *initShowTermin()*)



Wenn das Attribut *termin* einen Wert ungleich *null* und *controller* den Wert *null* hat, so bedeutet dies, dass der *TerminViewController* einen fremden Termin anzeigen soll. In diesem Fall sollen zwar die Informationen des Termins angezeigt werden wie in 2., aber alle Felder sollen auf nicht editierbar gesetzt werden. Vermeiden Sie Code-Duplikation!

Titel: Dem vorhandenen Titel muss der Name des Termin-Besitzers angehängt werden

Datum: wird auf das Datum des Termins gestellt

Teilnehmer-Button: wird deaktiviert

Save-Button: wird deaktiviert

Alle anderen Felder müssen mit den Inhalten des Termin-Objekts gefüllt werden.

Setzen Sie in *initialize()* nun noch den Event Handler des Buttons *addTeilnehmer* auf die Methode *showKontakte()*, den Event Handler des Buttons *save* auf die Methode *saveTermin()*. Dem Cancel-Button soll die Methode *close()* als Event Handler zugewiesen werden. Diese Methode schließt das Fenster mit der Termin-View.

Prüfen Sie, ob die Termin-View richtig initialisiert wird. Testen Sie den Aufruf für das Hinzufügen und das Editieren eines neuen Termins.

Implementieren Sie die Methode *saveTermin()*, die dafür zuständig ist, die Angaben aus den Eingabefeldern in den neuen bzw. zu aktualisierenden Termin einzutragen. Diese Methode soll folgendes tun:

- Wenn es sich um einen neuen Termin handelt (Attribut *termin* hat den Wert *null*) dann muss ein neues *Termin*-Objekt mit den Angaben aus dem Fenster erzeugt werden. Vergessen Sie nicht, die Teilnehmer aus der Teilnehmerliste zu lesen und dem Termin hinzuzufügen. Sie können die Methode *getTime(..)* nutzen, um die Zeitangaben in den Textfeldern in ein Objekt vom Typ *LocalTime* einzulesen.
- Wenn es sich um das Aktualisieren eines bereits bekannten Termins handelt (Attribut *termin* hat einen Wert ungleich *null*), so muss eine Kopie des Termins in *termin* erzeugt werden (hierfür können Sie die Methode *copyTermin()* der Klasse

Termin nutzen). In diese Kopie sollen dann die Infos aus den Eingabefeldern eingetragen werden.

Der daraus resultierende neue bzw. geänderte Termin soll dann an den *PlanerViewController* übergeben werden (Methode *processTermin(..)*). Beim Setzen der Zeitangaben oder beim Speichern des Termins (Terminüberschneidung) kann eine *UngueltigeTerminException* auftreten. Fangen Sie diese in der Methode *saveTermin()* und zeigen Sie die zugehörige Fehlermeldung in dem Label *error* an. Hat alles funktioniert, so schließen Sie das Termin-View-Fenster (Methode *close()*).

Die Methode *showKontakte()* realisieren Sie in der nächsten Übung.

Sehen Sie sich nun alles noch einmal genau an. Machen Sie sich klar, wie das Neuerstellen bzw. Aktualisieren eines Termins genau abläuft, vom *PlanerViewController* über den *TerminViewController* zurück zum *PlanerViewController* und dann zum *Terminplaner* bzw. zur *TerminVerwaltung*.

Probieren Sie gründlich aus, ob alles funktioniert:

Erstellen eines neuen Termins:

- a) Termin mit korrekten überschneidungsfreien Zeitangaben: Der Termin sollte danach korrekt in der Planer-View angezeigt werden.
- b) Termin mit falschen Zeitangaben *bis* vor *von* : Es sollte eine entsprechende Fehlermeldung unter den Zeitangaben erscheinen
- c) Termin mit Überschneidung am selben Tag: Es sollte eine entsprechende Fehlermeldung unter den Zeitangaben erscheinen

Editieren eines bestehenden Termins:

- a) Termin mit neuem Text: Der Termin sollte danach mit neuem Text in der Planer-View angezeigt werden.
- d) Termin mit neuen überschneidungsfreien Zeitangaben: Der Termin sollte danach mit den neuen Zeiten korrekt in der Planer-View angezeigt werden.
- e) Termin mit falschen Zeitangaben *bis* vor *von* : Es sollte eine entsprechende Fehlermeldung unter den Zeitangaben erscheinen
- f) Termin mit neuen Zeitangaben, die sich mit einem anderen Termin am selben Tag überschneiden: Es sollte eine entsprechende Fehlermeldung unter den Zeitangaben erscheinen