Aufgabe 1: Sammlungen in Sammlung

In der Klasse *Notenverwaltung* (siehe Blatt 1) werden die Noten aller Studierenden eines Studiengangs verwaltet. Die Klasse verfügt über Methoden, mit denen man Informationen zu den Studierenden (Klasse *StudentIn*, siehe Blatt2), deren Noten (Klasse *Note*, siehe Blatt2), die Module (Klasse *Modul*, siehe Blatt2) und die Bezeichner der Semester des Studiengangs abfragen kann.

Die Hashmap im Attribut `noten` enthält als Werte Objekte vom Typ *HashMap<String,Note>*, in denen unter dem Modulnamen die Note eines Studierenden in besagtem Modul gespeichert ist. Diese Notenmaps werden in der Hashmap `noten` unter der Matrikelnummer des Studierenden abgespeichert.

Schreiben Sie die Methode `getAlleNoten(String matrNr)`, die eine Matrikelnummer übergeben bekommt und eine Liste mit allen Noten zu der Nummer zurück liefert. Nutzen Sie die verfügbaren Methoden der Klasse *Notenverwaltung*. Es werden in der HashMap `noten` nur Einträge zu Studierenden verwaltet, die mindestens eine Modulnote besitzen. (10 Punkte)

Tipp: Eine kurze Zusammenfassung der wichtigsten Methoden für die Handhabung der Klasse *HashMap* finden Sie auf Blatt 5.

(0)

public List<Note> getAlleNoten(String matrNr){

Aufgabe 2: Exceptions

a) Schreiben Sie den Konstruktor `Note(Modul m, String su, String ue)`

der Klasse *Note* (siehe Blatt 2), der die Attribute des *Note*-Objekts mit den Werten aus den Methodenparametern belegt. Die Parameter su und ue müssen in Float-Werte gewandelt werden und sollen geprüft werden, ob sie gültige Notenwerte enthalten (der übliche Wertebereich für Noten). Ist einer der beiden Werte ungültig, so soll eine *UngueltigeNoteException* (siehe Blatt 2) geworfen werden. (11 Punkte)

11
(1-5 sie sagt)

b) Schreiben Sie für die Klasse *Notenverwaltung* (siehe Blatt 1) die Methode `addNote(..)`, die es erlaubt, eine neue Note für eine Matrikelnummer einzufügen. Hat der Studierende für das Modul bereits eine Note bekommen, so soll eine *NoteBereitsVorhandenException* geworfen werden. Denken Sie daran, dass es noch keinen Eintrag für den Studierenden gibt, falls dieser bisher noch kein Modul bestanden hat. (15 Punkte)

12

a) Hier void addNote (String matrNr, Note note) throws NoteBereits-

Aufgabe 3: Datei Ein-/Ausgabe

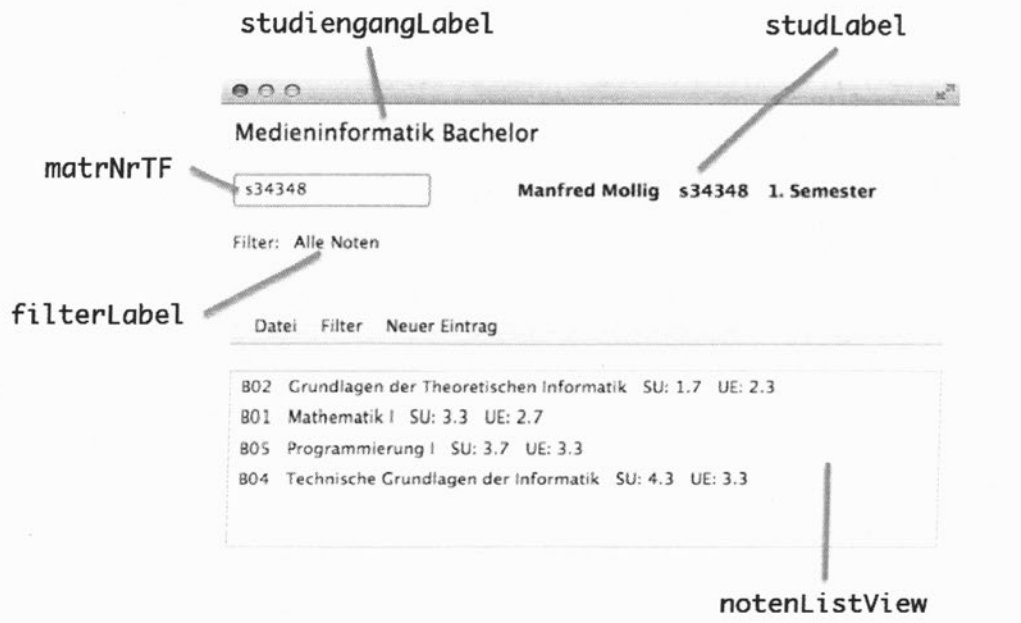
Schreiben Sie die Methode `ladeNotenStudi(String matrNr)` der Klasse *Notenverwaltung* (siehe Blatt 1). Falls Noten gespeichert wurden, befinden diese sich in einer Datei im Projektverzeichnis, die mit der Matrikelnummer des Studierenden benannt ist (z.B. *s34346.ser*). Die Noten sind in Form eines Objekts vom Typ *List<Note>* gespeichert. Die eingelesenen Noten sollen die vorhandenen Einträge zu der Matrikelnummer ersetzen. Die Methode gibt den Wert *true* zurück, wenn das Laden geklappt hat, sonst den Wert *false*. Sollten keine *Noten*-Objekte in der Datei gespeichert sein, so soll der Inhalt der Notenverwaltung nicht verändert werden. (20 Punkte)

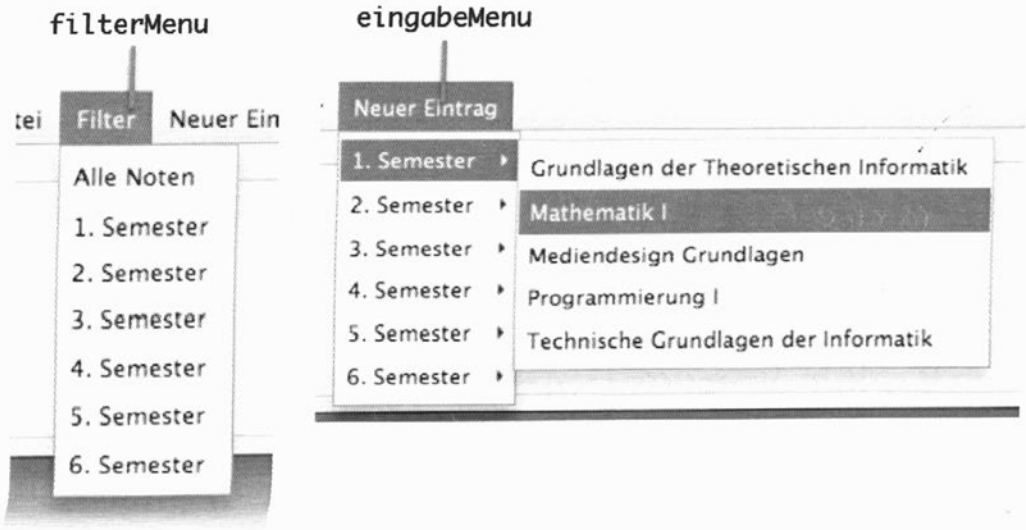
8

Aufgabe 4: GUI 1

Die Abbildungen unten zeigen die View und die Menüs, die in der Klasse *NotenViewController* (siehe Blatt 3) initialisiert werden. In der View werden die Noten zu einer Matrikelnummer angezeigt und neue Noten für die Matrikelnummer erfasst. Das Menü *Datei* erlaubt das Laden und Speichern der Notenliste zu der Matrikelnummer. Das Menü *Filtern* filtert die Anzeige der Noten in der `notenListView` nach den Semestern des Studiengangs. Das Menü *Eingabe* erlaubt es, eine neue Modulnote einzutragen.

Die Klasse *NotenViewController* enthält im Attribut `notenverwaltung` die Instanz der Klasse *Notenverwaltung*, mit der der Controller arbeiten soll.





a) Schreiben Sie die Methode `configureEingabeMenu()`, die das Menü im Attribut `eingabeMenu` aufbaut. Die Methode bezieht die benötigten Informationen von der Notenverwaltung. Bei allen MenuItems soll als Event Handler die Methode `openNoteneingabe(ActionEvent e)` angemeldet werden. (12 Punkte)

0

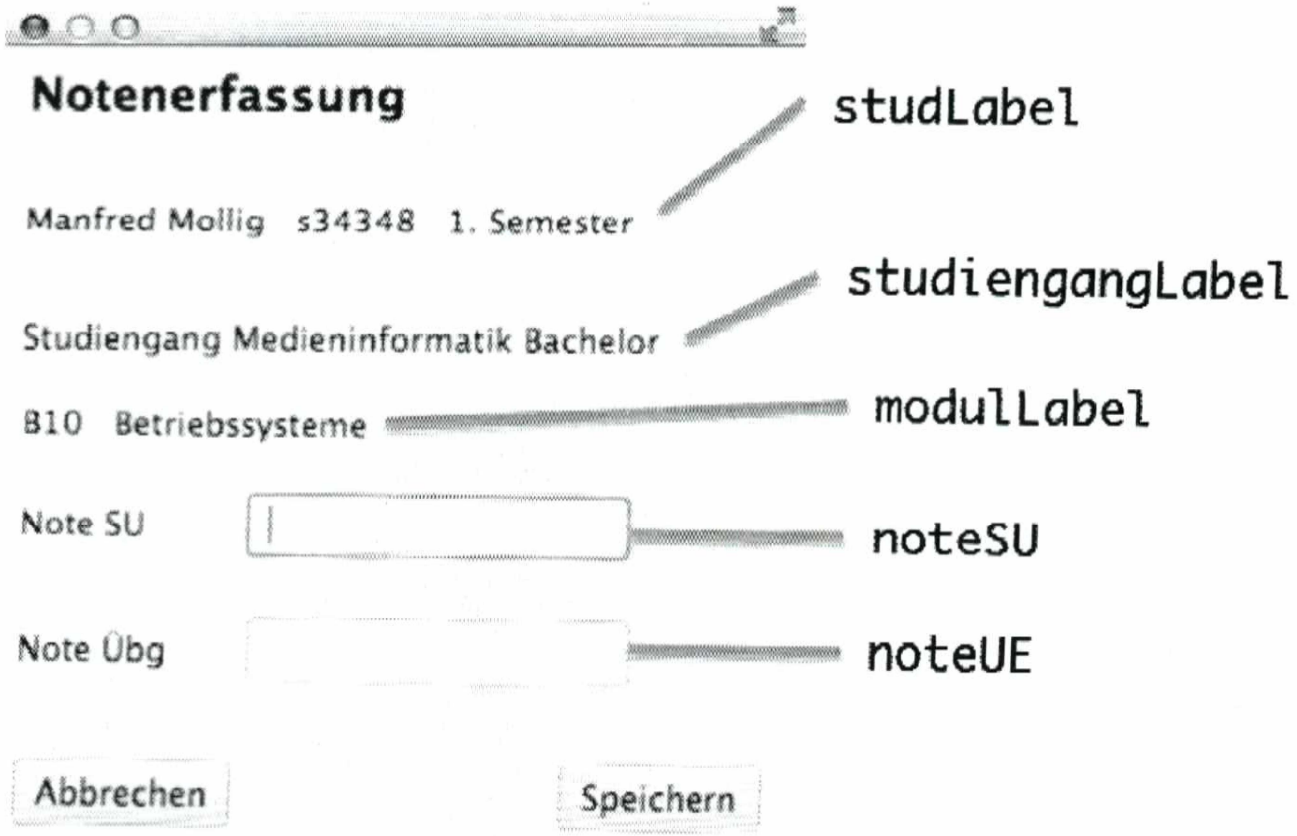
b) Ergänzen Sie die Klasse *NotenViewController* (siehe Blatt 3) um die Methode `searchStudentin()`, die aufgerufen wird, wenn die Eingabe im Textfeld `matrNrTF` bestätigt wurde. Sorgen sie dafür, dass in den Anzeigeelementen die zugehörigen Informationen angezeigt werden. Wurde kein Studierender zu der Eingabe gefunden, so soll die ListView geleert und die Labels mit dem leeren String versehen werden.

(14 Punkte)

1

Aufgabe 5: GUI 2

Die View für das Erfassen einer *Note* ist mit dem *NoteneingabeViewController* (siehe Blatt 4) gekoppelt. Der Konstruktor von *NoteneingabeViewController* bekommt das *Modul*-Objekt (siehe Blatt 2) einen String mit den Studierendeninfos, sowie den *NotenViewController* (siehe Blatt 3) übergeben.



a) Schreiben Sie die Methode `initialize(..)` der Klasse *NoteneingabeViewController*. Hier werden die *Label*-Objekte mit den anzuzeigenden Informationen belegt und die Buttons mit den entsprechenden Event Handlern versehen. (7 Punkte)

b) Schreiben Sie den Event Handler des Button `saveB`. Dieser soll aus den Angaben in den Textfeldern ein neues *Note*-Objekt erzeugen und dieses dem *NotenViewController* übergeben. War der Vorgang erfolgreich, so wird das Fenster geschlossen. Zeigen Sie eventuell auftretende Exceptions mit Hilfe der Klasse *ViewHelper* an. **(11 Punkte)** (7)

```
public class Notenverwaltung {

   private String studiengang;
   // key: Matrikelnummer
   // value: HashMap mit Abbildung Modulname (key) auf Note-Objekt
   // (value)
       private HashMap<String, HashMap<String, Note>> noten;
   ...

       public Notenverwaltung (String studiengang) {...}

   // Diese Methode schreiben Sie in Aufgabe 1:
   // public List<Note> getAlleNoten(String matrNr)

   // Diese Methode schreiben Sie in Aufgabe 2b:
   // public void addNote(String matrNr, Note note)

   // Fügt alle Noten der Liste list zu der matrNr hinzu
       private void addNoten(String matrNr, List<Note> list) {...}

   // Liefert alle Noten der matrNr im Semestern sem
       public List<Note> getNoten(String matrNr, String sem) {...}

   // Liefert alle Semesterbezeichner des Studiengangs
       public List<String> getSemesterBezeichner() {...}

   // Liefert alle Modulnamen im Semestern sem
       public List<String> getModulBezeichner(String sem) {...}

   // Liefert den Studierenden zu der matrNr
       public StudentIn getStudentIn(String matrNr) {...}

   // Diese Methode schreiben Sie in Aufgabe 3:
   // public boolean ladeNotenStudi(String matrNr)

   ...
}

class NoteBereitsVorhandenException extends Exception {
```

```
public class Note implements Serializable {
      private Modul modul;
      private float suNote;
      private float ueNote;

   // Den Konstruktor schreiben Sie in Aufgabe 2a:
   // Note(Modul m, String su, String ue)

      public String getModulname() {...}

    ...

      public String toString() {
        return modul.toString() + "   SU: " + suNote + "   UE: " + ueNote;
      }
}


public class Modul implements Serializable {

       private String kuerzel;
       private String name;
       private String studiengang;
       private String semester;

       public Modul(String kuerzel, String titel, String studiengang,
                                                        String semester) {...}
       public String getStudiengang() {...}

       public String toString() {
         return kuerzel + "   " + name;
       }
}


public class StudentIn implements Serializable {
   ...

      public String toString() {
        return name + "   " + matrikelNr + "   " + semester;
      }
}


public class UngueltigeNoteException extends Exception {

      public UngueltigeNoteException(Modul modul, String note) {
          super("Die Note " + note + " für das Modul " +
                              getModulname() + " ist nicht gueltig.");
      }
}
```

```
public class NotenViewController implements Initializable {
        @FXML
        private Label studiengangLabel;
        @FXML
        private TextField matrNrTF;
        @FXML
        private Label studLabel;
        @FXML
        private Label filterLabel;
        @FXML
        private ListView<Note> notenListView;
        @FXML
        private Menu dateiMenu;
        @FXML
        private Menu filterMenu;
        @FXML
        private Menu eingabeMenu;



        private ObservableList<Note> notenliste;
        private Notenverwaltung notenverwaltung;
        private StudentIn studierender;

        @Override
        public void initialize(URL url, ResourceBundle rb) {...}

        ...

    // Diese Methode schreiben Sie in Aufgabe 4a:
    // private void configureEingabeMenu() {...}

    // Diese Methode Schreiben Sie in Aufgabe 4b:
    // private void searchStudentIn()


    // Zeigt die Noten der Liste in der ListView an.
       private void showNoten(List<Note> liste) {...}

    // fügt der Notenverwaltung die Note note für die ausgewählte
    // Matrikelnummer hinzu
       public void neueNote(Note note) {...}

      ...

}
```

```
public class NoteneingabeViewController implements Initializable {
        @FXML
        private Label studLabel;
        @FXML
        private Label studiengangLabel;
        @FXML
        private Label modulLabel;
        @FXML
        private TextField noteSU;
        @FXML
        private TextField noteUE;
        @FXML
        private Button cancelB;
        @FXML
        private Button saveB;

        private Modul modul;
        private String studi;
        private NotenViewController notenController;


        public NoteneingabeViewController(Modul mod, String studInfo,
                                          NotenViewController contr) {...}

    // Diese Methode Schreiben Sie in Aufgabe 5a:
    // public void initialize(URL url, ResourceBundle rb)

        private void close() {...}

    // Diese Methode Schreiben Sie in Aufgabe 5b:
    // private void save()

}
```

Umgang mit *HashMap*

```
HashMap<String, Hund> hundePension = new HashMap<String, Hund>();
```

Objekt **hinzufügen** (Klasse Hund hat get-Methode für Namen):

```
Hund hund1 = new Hund("Bello"); //String wird im Attribut name abgelegt
hundePension.put(hund1.getName(), hund1);
```

Objekte **zugreifen** mit Hilfe des Schlüssels (Name des Hundes):

```
Hund hund2 = hundePension.get("Bello");
```

Ein Objekt aus der HashMap **löschen**: (liefert gelöschtes Objekt zurück)

```
Hund geloeschterHund = hundePension.remove("Bello");
```

Abfrage, ob unter einem bestimmten **Schlüssel** ein Objekt abgespeichert ist:

```
boolean belloExistiert = hundePension.containsKey("Bello");
```