Beuth Hochschule für Technik Berlin
Prof. Dr. Simone Strippgen

Klausur
Programmierung II

Datum: .....

Name: ..... Vorname: .....

Matr.-Nr.: ..... letzter Versuch: ❑ Ja ❑ Nein

Bitte zuerst den Kopf dieses Blattes vollständig und leserlich ausfüllen. Lassen Sie bei der Bearbeitung der Aufgaben die Blätter zusammen geheftet. Bitte schreiben Sie die Lösungen unter den jeweiligen Aufgabentext. Benutzen Sie kein eigenes Papier (außer zum Vorschreiben).

Viel Erfolg!

Aufgabe	**Mögliche Punkte**	**Erstkorrektur**	**Zweitkorrektur**
Aufgabe 1 : Sammlungen in Sammlung	9		
Aufgabe 2: Exceptions	22		
Aufgabe 3: Datei Ein-/Ausgabe	17		
Aufgabe 4: GUI 1	29		
Aufgabe 5: GUI 2	23		
Punkte Gesamt	**100**		
Note			

Aufgabe 1: Sammlungen in Sammlung

In der Klasse *CDSammlung (siehe Blatt 1)* werden Sammlungen von CDs im Attribut sammlung vom Typ HashMap verwaltet. Hierin ist die Sammlung der CDs eines Interpreten unter dem Namen des Interpreten abgespeichert. Die Sammlung der CDs eines Interpreten wird ebenfalls als *HashMap* organisiert. Hierin wird der Name des Album als Schlüssel für den Zugriff auf die Instanzen der Klasse *CD* verwendet.

Schreiben Sie die Methode *loeschen(String interpret, String album)*, die den Interpretennamen und den Albumnamen übergeben bekommt und die zugehörige CD aus der HashMap *sammlung* löscht. Die gelöschte CD soll an den Aufrufer zurückgegeben werden. Es sollen in der Sammlung nur Interpreten verwaltet werden, von denen mindestens ein Album in der Sammlung ist.

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

(9 Punkte)

```
public CD loeschen(String interpret, String album)       {




}
```

Aufgabe 2: Exceptions

a) Schreiben Sie die Methode

```
public void setzeInfos(String album, String interpret, int tanzahl)
```

der Klasse CD (siehe Blatt 2), die Attribute des CD-Objekts mit dem Werten aus den Methodenparameter belegt. Die Parameter album und interpret sollen allerdings vor der Zuweisung daraufhin geprüft werden, ob sie leere Zeichenketten (kein Zeichen oder nur Leerzeichen) oder null Werte enthalten. Ist einer der beiden Werte ungültig, so soll eine UngueltigeCDException (siehe Blatt 1) geworfen werden. Achten Sie darauf, dass keine NullPointerException auftreten kann.

(9 Punkte)

b) Schreiben Sie für die Klasse *CDSammlung* (siehe Blatt 1) eine Methode *hinzufuegen*, die es erlaubt, ein neues CD-Objekt in die Sammlung einzufügen. Sollte sich die CD bereits in der Sammlung befinden, so soll eine *AlbumBereitsVorhandenException* geworfen werden. Denken Sie daran, dass Sie eine neue *HashMap* für die CDs des Interpreten erstellen und in die Sammlung einfügen müssen, falls noch gar keine CD des Interpreten in der Sammlung enthalten ist.

(13 Punkte)

Aufgabe 3: Datei Ein-/Ausgabe

Schreiben Sie die Methode `laden(File path)` der Klasse *CDSammlung* (siehe Blatt 1). Diese bekommt einen Pfad zu einer Datei in der serialisierte *CD*-Objekte zu finden sind und soll alle *CD*-Objekte aus der Datei einlesen und die Sammlung damit befüllen (alle alten Einträge sollen vorher verworfen werden). Gehen Sie davon aus, dass die Objekte dann komplett ausgelesen sind, wenn die entsprechende Methode zum Auslesen eines Objektes aus dem Strom den Wert *null* liefert.

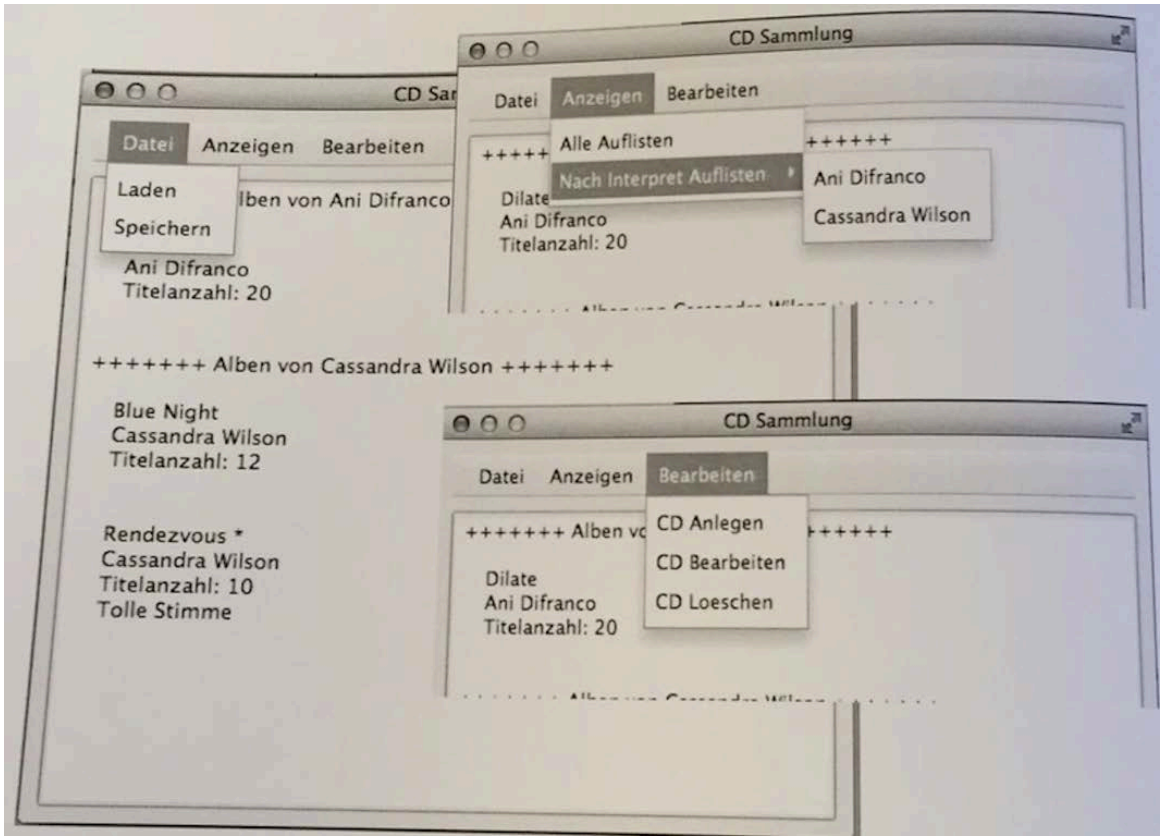
(17 Punkt)

```
public void laden(File path) {

}
```

Aufgabe 4: **GUI 1**

Die Abbildungen unten zeigen das Hauptfenster der Anwendung, das in der Klasse *SammlungViewController* (siehe Blatt 3) ergänzt wird. Das Menü *Datei* erlaubt es die CDs der Sammlung in einer Datei zu speichern und aus der Datei wieder einzulesen. Das Menü *Anzeigen* erlaubt es in der *Textarea* im Attribut *textArea* alle CDs oder die CDs eines ausgewählten Interpreten aufzulisten. Das Menü *Bearbeiten* erlaubt es eine ausgewählte CD zu bearbeiten oder zu löschen und der Sammlung eine neue CD hinzufügen.



a) Schreiben Sie die Methode *addMenuAnzeigen()* der Klasse *SammlungViewController*, die dem Menü *Anzeigen* im Attribut *anzeigenMenu* die benötigten Menüpunkte hinzugefügt. Lagern Sie das Hinzufügen der Menüpunkte zu dem Untermenü *Nach Interpreten Auflisten* in die Methode *addInterpretItems(Menu menu)* aus. Diese Methode bekommt das Menü übergeben, in das die Interpreten-Menüpunkte eingefügt werden sollen. Die Interpretennamen müssen über die CD-Sammlung bezogen werden. Melden Sie bei allen Menüpunkten die Methoden *handleMenu(ActionEvent e)* als Event Handler an.

(12 Punkte)

```
private void addMenuAnzeigen() {









}

private void addInterpretenItems(Menu menu) {









}
```

a) Schreiben Sie die Mthode *handleMenu(ActionEvent e)* der Klasse *SammlungViewController*, die auf alle Menü-Aktionen des Hauptfensters reagiert. Finden Sie heraus, welcher Menüpunkt ausgewählt wurde, aber behandeln Sie hier nur den Menüpunkt mit der Aufschrift *Laden*, die anderen Menüpunkte können weggelassen werden. Bei dieser Aktion soll ein FileChooser geöffnet werden (Pfad-Voreinstellung: */user/music/cds*) um die gewünschte Datei auszuwählen. Die Objekte der Datei sollen in die CD-Sammlung geladen (siehe Aufgabe 3) werden. Danach soll die Anzeige für die Auflistung des gesamten CD-Bestands aktualisiert und die Menüpunkte für das Untermenü *Nach Interpreten Auflisten* neu gesetzt werden.

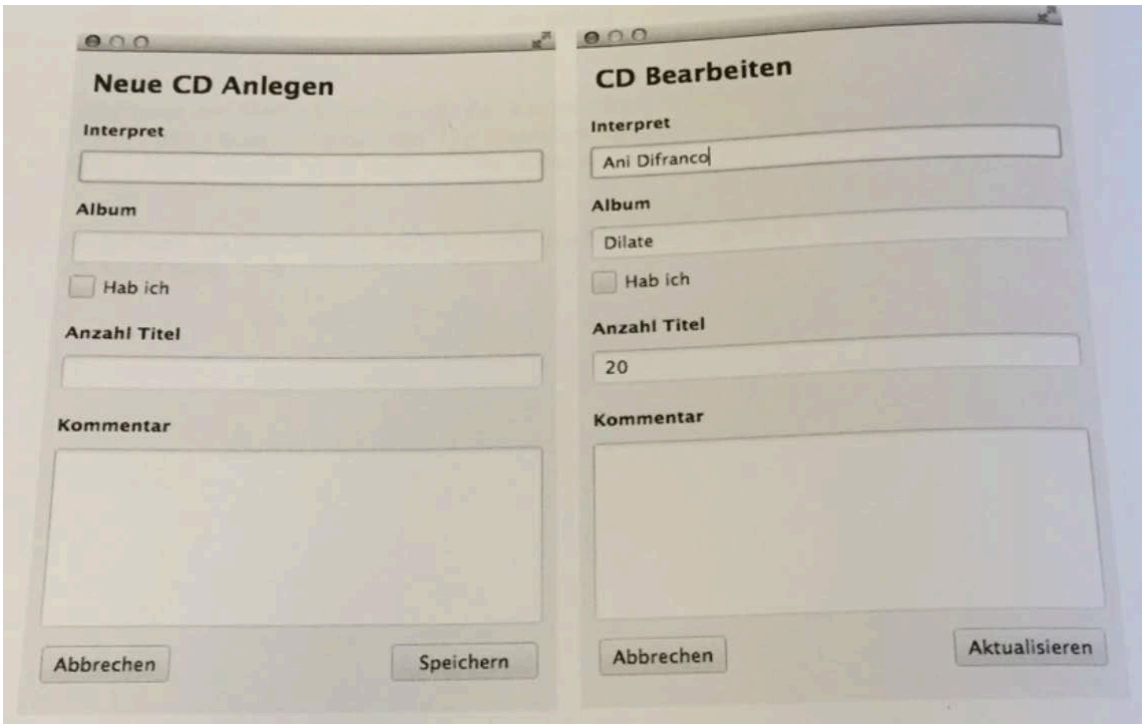
(17 Punkte)

Tipp: mit der Methode *getItems()* der Klasse Menu, bekommt man alle Menüpunkte in Form einer *ObservableList* vom Typ *MenuItem* zurück. Hier lässt sich mit der Methode *get(..)* unter Angabe eines Index auf die Items zugreifen.

Sie können die Abarbeitung auch auf mehrere Methoden verteilen. Mehr Platz ist auf der nächsten Seite

Aufgabe 5: **GUI 2**

Die GUI für das Erstellen bzw. Editieren einer CD ist mit dem *CDEditViewController* (siehe Blatt 4) gekoppelt. Der Konstruktor der Klasse *CDEditViewController* bekommt im Methodenparameter *cd* den Wert *null* übergeben, wenn eine neue CD angelegt und eine Instanz der Klasse *CD*, wenn eine CD bearbeitet werden soll. Inder Methode *initialize()* wird, je nachdem mit welchem Wert das Attribut *cd* initialisiert wurde, die GUI unterschiedlich konfiguriert.



Tipp: Bei einer Instanz von *CheckBox* lässt sich mit *boolean getSelected()* abfragen, ob die Checkbox selektiert ist, mit *setSelected(boolean s)* lässt sich der Wert setzen.

a) Schreiben Sie die Methoden *initNewTermin()* und *initUpdateTermin()* so, dass die GUIs wie oben abgebildet und unten beschrieben initialisiert werden.

initNewCD(): Titel wird gesetzt, rechter Button (*saveButton*) bekommt die Aufschrift *Speichern*, die Methode *speichern()* wird dort als Event Handler angemeldet.

initUpdateCD(): Titel wird gesetzt, die Eingabemaske wird mit den Informationen aus der CD in *cd* gefüllt und als Event Handler wird bei dem rechten Button die Methode *aktualisieren()* angemeldet.

(11 Punkte)

```
private void initNewCD() {









}

private void initUpdateCD() {









}
```

b) Schreiben Sie die Methode aktualisieren(). Diese wird aufgerufen wenn man auf den Button mit der Aufschrift Aktualisieren gedrückt hat. Die im Fenster eingetragenen Änderungen an der CD sollen in der CD-Sammlung im Attribut sammlung aktualisiert und danach das Fenster geschlossen werden. Fangen Sie die hierbei möglicherweise auftretenden Exceptions und zeigen Sie deren nachricht mit Hilfe der Klasse ViewHelper (so wie in der Übung verwendet) an.

(12 Punkte)

```
private void aktualisieren()  {
```

```
public class CDSammlung {

    private HashMap<String, HashMap<String, CD>> sammlung;

    public CDSammlung()  {
          sammlung = new HashMap<String, HashMap<String, CD>>();
    }


    //Die Methode hinzufuegen(CD cd) schreiben Sie in Aufgabe 2b


    public CD loeschen(String interpret, String album) {...}
          // schreiben Sie in Aufgabe 1a


    // Gibt die CD zu dem Interpret und Album zurück
    public void CD gibCD(String interpret, String album) {...}


    // Gibt die Namen aller Interpreten zurück
    public String[] gibAlleInterpreten() {...}


    // Gib alle CDs des genannten Interpreten
    public CD[] gibAlleCDsVon(String interpret) {...}


    // Entfernt die alte CD und ersetzt sie durch die neue
    public void aktualisierenCD(CD alteCD, CD neueCD)
                      throws AlbumBereitsVorhandenException {...}


    public void laden(File path) {...}
          // schreiben Sie in Aufgabe 3

    public void speichern(File path) {...}

}


public class AlbumBereitsVorhandenException extends Exception {

      public AlbumBereitsVorhandenException(String album, String interpret) {
            super("Album" + album + "von" + interpret + "ist bereits vorhanden.");
      }
}
```

```
public class CD implements Serializable {

    private String album;
    private String interpret;
    private int titelAnzahl;
    private boolean habIch;
    private String Kommentar;

    public CD() {
      album = " ";
      interpret = " ";
      kommentar = "<kein Kommentar>";
    }

    // Die Methode
    //     setzeInfos(String album, String interpret, int tAnzahl)
    // schreiben Sie in Aufgabe 2a


    public void setzeBesitzInfos(boolean habIch, String kommentar) {...}


    public String gibKommentar() {...}
    public String gibInterpret() {...}
    public String gibAlbum() {...}
    public int gibAnzahlTitel() {...}
    public boolean gibVorhanden() {...}
    public String toString() {...}
}




public class UngueltigeCDException extends Exception {

   public UngueltigeCDException(String interpret, String album) {
     super("ungueltige Eingabe!\n" +
           "Interpret: " + interpret +
           "\nAlbum: " + album);
   }
}
```

```
public class SammlungViewController implements Initializable {

  @FXML
  private MenuBar menuBar;
  @FXML
  private Menu dateiMenu;
  @FXML
  private Menu anzeigenMenu;
  @FXML
  private Menu bearbeitenMenu;
  @FXML
  private TextArea textArea;

  private CDSammlung sammlung;


  @Override
  public void initialize(URL url, ResourceBundle rb) {
        sammlung = new CDSammlung();
        textArea.setEditable(false);
        configureMenu();
  }

  private void configureMenu() {...}

  private void addMenuAnzeigen() {...}
        // schreiben Sie in Aufgabe 4a

  private void addInterpretenItems(Menu menu) {...}
        // schreiben Sie in Aufgabe 4a

  private void handleMenu(ActionEvent e) {...}
        // schreiben Sie in Aufgabe 4b


  // Fügt die Auflistung aller CDs des Interpreten am Ende der
  // TextArea an

  private void addInterpret(String interpret) {...}


  // Aktualisiert die Anzeige des gesamten CD-Bestands in der TextArea

  private void listAlle() {...}

}
```

```
private class CDEditViewController implements Initializable {

  @FXML
  private TextField interpretTF;
  @FXML
  private Label titel;
  @FXML
  private TextField albumTF;
  @FXML
  private TextField titelAnzahlTF;
  @FXML
  private TextArea kommentarTA;
  @FXML
  private Checkbox besitzCB;
  @FXML
  private Button cancelButton;
  @FXML
  private Button saveButton;

  private CDSammlung sammlung;

  private CD cd;

  CDEditViewController(CDSammlung bib, CD cd) {
      sammlung = bib;
      this.cd = cd;
  }

  @Override
  public void initialize(URL url, ResourceBundle rb) {
      if (cd == null) initNewCD();
      else initUpdateCD();
      ...
  }

  private void initUodateCD() {...}
          // schreiben Sie in Aufgabe 5a

  private void initNewCD() {...}
          // schreiben Sie in Aufgabe 5a

  private void aktualisieren() {...}
          // schreiben Sie in Aufgabe 5b

  private void speichern() {...}

  private void close() {...}
```

Umgang mit HashMap

```
HashMap<String, Hunde> hundPension = new HashMap<String, Hunde>();
```

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

```
hundePension.remove("Bello")
```

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

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