Aufgabe Milchstrasse

1 Unsere Sonnensystem: Sol

In dieser Aufgabe geht es um die Modellierung unseres Sonnensystems und seiner Planeten.
Es werden die Fähigkeiten zu den Themenkomplexen: Objektorientierung, Umgang mit Dateien und Oberflächen mit JavaFX beübt.
Es werden die folgenden Komponenten benötigt:

- Datenobjekte für Planeten, die Klasse `Planet`,
- eine Klasse für das Sonnensystem, `Sonnensystem`,
- eine Klasse `DateiVerbindung`, die Informationen aus Dateien liest,
- einem "DAO-Objekt" (=DAO für Data Access Object) für die Planeten und das Sonnensystem sowie
- den JavaFX-Container `PlanetView` und die JavaFX-Anwendung `SonnenSystemApp`.

1.1 Der Planet

Die Klasse `Planet` ist die Datenklasse in der die einzelnen Planeten verwaltet werden. Instanzen dieser Klasse kennen:

- den Namen des Planeten,
- seinen Durchmesser in Kilometern (Ganzzahl),
- seinen mittleren Abstand zur Sonne in Kilometern (große Ganzzahl),
- sein relatives Gewicht im Verhältnis zum Gewicht der Erde (Fließkommazahl),
- seine Umrundungszeit in Erdtagen (Fließkommazahl),
- seine Anzahl an Monden,
- und besondere Bemerkungen.

Alle Eigenschaften sind bereits mit Konstruktion bekannt und unveränderlich. Ein Zugriff darauf findet ausschließlich mit Hilfe von Gettern statt.

Zwei Planeten sind immer dann verschieden, wenn sie verschiedene Namen und unterschiedliche Abstände zur Sonne haben.

Für Planeten existiert eine natürliche Ordnung. Ein Planet ist dann "kleiner", wenn er näher an der Sonne ist.

1.2 Die Klasse `SonnenSystem`

Eine Instanz dieser Klasse repräsentiert ein Sonnensystem. Jedes Sonnensystem besitzt einen Namen, ein Feld für Bemerkungen und eine Liste von `Planet`en. Diese Daten werden später befüllt. Im Moment soll es einen Konstruktor geben, der einen Dateinamen übernimmt. In dieser Datei (die im weiteren Verlauf eingelesen wird, s. 1.3) stehen alle nötigen Informationen.

Die Planetenliste ist von außen nicht adressierbar. Stattdessen sind die Instanzen von `SonnenSystem` über ihre `Planet`en iterierbar.

Weiterhin gibt es zwei Methoden `naechster()` und `vorheriger()`, die jeweils den nächsten bzw. den vorherigen `Planet`en zurückgeben. Dazu ist es sinnvoll, den Listenplatz des aktuellen Planeten zu speichern. Zudem ist erforderlich, sich ein Verhalten für das "Anschlagen" an den Enden der Liste zu überlegen: Soll am Ende angehalten werden, oder vom Anfang begonnen werden? Wie auch immer diese Entscheidung aussieht, sie muss für beide Methoden konsistent implementiert werden.

1.3 Die Klasse `DateiVerbindung`

Ähnlich wie in der Termin-Aufgabe im Kapitel Datenbanken ist die Klasse `DateiVerbindung` für das Laden von Informationen aus Dateien verantwortlich.
Hierfür braucht es keine inneren Zustände oder verschiedenen Instanzen, die benötigten Methoden sind alle statisch. Es ist zudem sicherzustellen, dass `DateiVerbindung` nicht instanziiert werden kann.

In `DateiVerbindung` gibt es eine statische Methode `liesZeilenweise( String dateiName)`, die eine Datei zeilenweise liest und die Zeilen in einer Liste zurückgibt. Hier findet keinerlei Interpretation oder Filterung statt.

1.4 Die Klasse `PlanetDAO`

Ähnlich wie in der Termin-Aufgabe im Kapitel Datenbanken ist die Klasse `PlanetDAO` für die Bereitstellung fachlich interpretierter Daten zuständig.

Hierfür braucht es keine inneren Zustände oder verschiedenen Instanzen, die benötigten Methoden sind alle statisch. Es ist zudem sicherzustellen, dass `PlanetDAO` nicht instanziiert werden kann.

Eine statische Methode `ladePlaneten( String dateiName)` benutzt die Einlesemethode von `DateiVerbindung`, und kümmert sich um die Interpretation der Daten: Es werden Kopfzeilen übersprungen und Daten in die korrekten Typen umgewandelt. Sollten dabei Fehler auftreten, sind diese direkt hier zu behandeln.

Es ist die Struktur der beigefügten CSV-Datei zu beachten: in der ersten Zeile sind die Überschriften, die zweite Zeile enthält die Informationen über unsere Sonne. Erst ab der dritten Zeile stehen die Planeten.

Da für die CSV-Dateien nicht ausgeschlossen werden kann, dass Planeten doppelt vorkommen, ist mit einer geeigneten Datenstruktur sicherzustellen, dass ein Planet nicht zweimal hinzugefügt wird. Die Methode gibt diese geeignete Datenstruktur zurück.

1.5 Erweiterung der Klasse `SonnenSystem`

Jetzt kann die Arbeit am `SonnenSystem`-Konstruktor weiter gehen: In der Klasse `DateiVerbindung` werden aus der dem Konstruktor übergebenen CSV-Datei alle Informationen zu unserem Sonnensystem geladen werden können.

Die Informationen zum Sonnensystem selbst liegen im "nullten Planeten", der als Vehikel für den Transport dieser Daten missbraucht wird. Da es sich dabei um die Sonne des Sonnensystems handelt, wird sie natürlich nicht der Planetenliste hinzugefügt.

1.6 Testen und Sortieren

In dieser Teilaufgabe sollen die bis hierhin implementierten Anteile in einer Konsolenanwendung getestet werden.

Erweitern Sie hierfür die Klasse `Planet` um eine `toString`-Methode, die alle Informationen eines Planeten zurückgibt. Passen Sie die Gleitzahllattribute in `Planet` so an, dass eine vernünftige Anzahl von Nachkommastellen angezeigt wird.

Bislang werden in einer Instanz von `SonnenSystem` die Planeten in der Reihenfolge abgespeichert, in der sie auch aus der Datei gelesen und hinzugefügt wurden (ausser, Sie haben die "geeignete Datenstuktir" wirklich mit Bedacht gewählt). Die Instanzen von `Planet` sind jedoch schon vergleich- und sortierbar.
Ergänzen Sie `SonnenSystem` um eine `sort`-Methode, die die Planeten in der Liste entsprechend ihrer natürlichen Sortierung sortiert. Denken Sie dabei daran, den "aktuellen Planeten" auf einen sinnvollen Wert zurückzusetzen. Überladen Sie die Methode um eine weitere Variante, der ein `Comparator<T>`-Objekt mit übergeben wird. Legen Sie in der Klasse `Planet` mindestens vier verschiedene Komparatoren als öffentliche Klassenkonstanten an.

Testen Sie die Komparatoren.
Nutzen Sie bei Ihren Tests auch die Fähigkeit von `SonnenSystem` iterierbar zu sein.

1.7 Die JavaFX-Anwendung

Die Abbildung 1 zeigt den Aufbau der JavaFX-Anwendung.

Hauptbestandteil der JavaFX-Anwendung ist `PlanetView` der JavaFX-Container, der alle Informationen eines Planeten anzeigt. `PlanetView` ist von `javafx.scene.layout.BorderPane` abgeleitet.

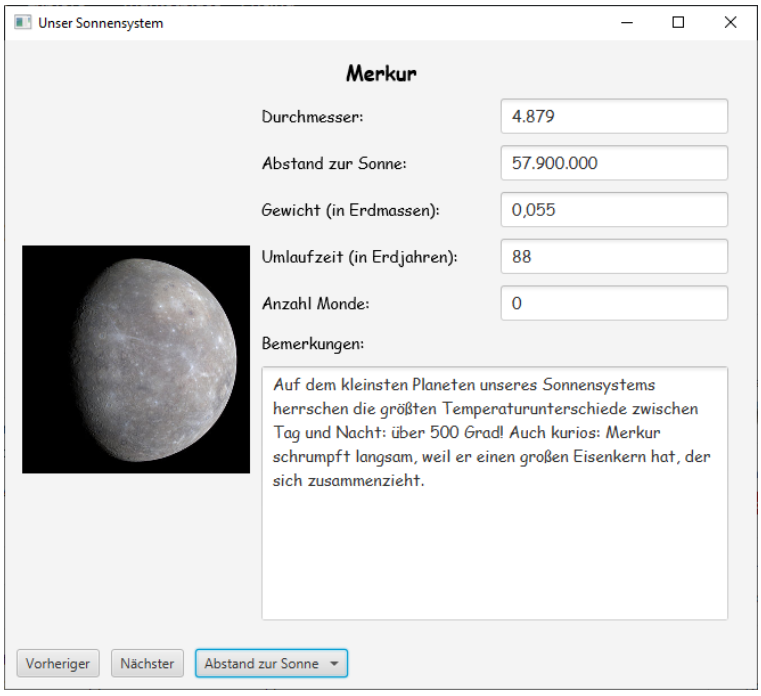

Abbildung 1: JavaFX-Anwendung

Im *Center*-Bereich werden die Detailinformationen zum Planeten entsprechend der Abbildung 1 dargestellt.

Im *Left*-Bereich soll ein Bild des Planeten dargestellt werden. Hierfür muss zunächst die Klasse `DateiVerbindung` um eine Methode `ladeBild` ergänzt werden.

Einer der Konstruktoren von `javafx.scene.image.Image` kann ein Bildobjekt aus einem Inputstream erzeugen, dieser kann hierfür verwendet werden.
Die Bilder der Planeten heißen alle so, wie der Planet heißt.

Objekte vom Typ `javafx.scene.image.Image` können in einem `javafx.scene.image.ImageView` angezeigt werden.

Ein Bild, das in einem `ImageView`-Objekt angezeigt wird, kann in seiner Größe beschränkt werden. Die Methoden `setFitWidth` und `setFitHeight` geben die Größe der “BoundingBox” vor, in die das Bild eingepasst wird.

`PlanetView` besitzt eine öffentliche Methode `setPlanet`, der ein `Planet`-Objekt übergeben wird. Die Methode befüllt alle Felder mit den richtigen Informationen und lädt das richtige Bild.

Die JavaFX-Anwendung `SonnenSystemApp` ist wie üblich von `javafx.application.Application` abgeleitet und ist der JavaFX-Programmeinstieg. Die Klasse sorgt dafür, dass eine Instanz von `PlanetView` angezeigt wird.
Weiterhin verfügt die Anwendung über zwei `Button`s, um zwischen den `Planet`en zu wechseln. Neben diesen `Button`s ist eine `javafx.scene.control.ChoiceBox<T>`, mit der die Sortierreihenfolge geändert werden kann.

Die einfachste Variante der Befüllung einer `ChoiceBox` ist:

```
new ChoiceBox<>(FXCollections.observableArrayList("Abstand zur Sonne", "Name",
    "Gewicht", "Anzahl Monde", "Umlaufzeit"));
```

Mit `getSelectionModel().selectedIndexProperty().addListener(...)` oder `getSelectionModel().selectedItemProperty().addListener(...)` kann auf eine geänderte Auswahl in der `ChoiceBox` reagiert werden.

Überlegen Sie sich eine sinnvolle Package-Struktur für die Anwendung!