Übungsblatt 02

3 Aufgaben, 20 Punkte

Objektorientierte Modellierung und Programmierung (inf031) Sommersemester 2020
Carl von Ossietzky Universität Oldenburg, Fakultät II, Department für Informatik

Dr. C. Schönberg

Ausgabe: 2020-04-28 14:00 **Abgabe:** 2020-05-05 12:00

Aufgabe 1: *Stern-Datenbank* **(3 + 1 + 1 + 1 Punkte)**

Der Astronom Stefan Sterngucker besitzt ein Java-Programm zur Verwaltung seiner astronomischen Beobachtungen. Es besteht aus einer Klasse `Star` zur Repräsentation von Sternen, einer Klasse `StarsDatabase` zur Verwaltung der eigentlichen Datenbank, und einer Testklasse `StarsTest` zum Testen der Implementierung:

```java
public class Star {
  private StarsDatabase database;
  private String name;
  private String id;
  private double distance;
  private double apparentMagnitude;
  private String type;
  public Star() { }
  public Star(String name, String id) {
    this.name = name;
    this.id = id;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  // other getters and setters ...
}
```

```java
/**
 * Represents a collection of Stars.
 */
public class StarsDatabase {

  private Star[] stars = new Star[0];

  /**
   * Add a new star to the database.
   * @param star The star to be added
   */
  public void add(Star star) {
    star.setDatabase(this);
```

```
14      Star[] tmp = new Star[stars.length + 1];
15      for (int i = 0; i < stars.length; i++) {
16        tmp[i] = stars[i];
17      }
18      tmp[stars.length] = star;
19      stars = tmp;
20    }
21
22    /**
23     * Remove an existing star from the database.
24     * Fails if the index is out of range.
25     * @param index The index of the star to be removed
26     */
27    public void remove(int index) {
28      Star[] tmp = new Star[stars.length - 1];
29      int j = 0;
30      for (int i = 0; i < stars.length; i++) {
31        if (i != index) {
32          tmp[j] = stars[i];
33          j++;
34        }
35      }
36      stars = tmp;
37    }
38
39    /**
40     * Retrieves a star from the database. Fails if the index is out of range.
41     * @param index The index of the star to be retrieved
42     * @return
43     */
44    public Star get(int index) {
45      return stars[index];
46    }
47
48    /**
49     * Returns the size of the database.
50     * @return The number of entries in the database
51     */
52    public int size() {
53      return stars.length;
54    }
55
56    /**
57     * Returns the star with the given id if it is in the database,
58     * null otherwise.
59     * @param id The id of the star to be retrieved
60     * @return The star with the given id, or null if no such star is in the
         database
61     */
62    public Star find(String id) {
63      for (Star star : stars) {
64        if (star.getId().equals(id)) {
65          return star;
66        }
67      }
68      return null;
69    }
70  }
```

```
1   public class StarsTest {
2
```

```
3   @Test
4   void testAddGetSize() {
5     StarsDatabase db = new StarsDatabase();
6     assertEquals(0, db.size());
7     Star star = new Star(”Sirius”, ”TYC 5949-2777-1”);
8     db.add(star);
9     assertEquals(1, db.size());
10    star = db.get(0);
11    assertNotNull(star);
12    assertEquals(”TYC 5949-2777-1”, star.getId());
13  }
14
15  @Test
16  void testAddRemoveSize() {
17    StarsDatabase db = new StarsDatabase();
18    Star sirius = new Star(”Sirius”, ”TYC 5949-2777-1”);
19    db.add(sirius);
20    db.add(new Star(”Alpha Centauri”, ”TYC 9007-5849-1”));
21    assertEquals(2, db.size());
22    db.remove(0);
23    assertEquals(1, db.size());
24    db.add(sirius);
25    assertEquals(”TYC 9007-5849-1”, db.get(0).getId());
26  }
27
28  @Test
29  void testFind() {
30    StarsDatabase db = new StarsDatabase();
31    db.add(new Star(”Sirius”, ”TYC 5949-2777-1”));
32    db.add(new Star(”Alpha Centauri”, ”TYC 9007-5849-1”));
33    Star star = db.find(”TYC 9007-5849-1”);
34    assertEquals(”TYC 9007-5849-1”, star.getId());
35    star = db.find(”TYC 5331-1752-1”);
36    assertNull(star);
37  }
38
39 }
```

Aufgabe: Zur Dokumentation der Implementierung und für mögliche Weiterentwicklungen möchte Stefan Sterngucker das Programm als Modell repräsentieren. Helfen Sie ihm dabei, indem Sie die Klassen `Star` und `StarsDatabase` als UML Klassendiagramm modellieren, und die drei Programmzustände nach Ausführung der drei Testmethoden aus `StarsTest` als jeweils ein UML Objektdiagramm modellieren. Insgesamt sollen Sie also vier Modelle erstellen.

Die getter- und setter-Methoden der Klasse `Star` dürfen Sie in Ihrem Modell weglassen.

Aufgabe 2: *Modellierung und Implementierung von Tänzen* **(6 + 2 + 2 Punkte)**

Der Tanzlehrer Quirin Quickfoot will für seine Schüler eine Wissensdatenbank für Tänze implementieren, in der sie strukturiert Informationen zu verschiedenen Tänzen und einzelnen Figuren nachschlagen können. Dazu will er zunächst folgende Daten repräsentieren:

- Das allgemeine zu beschreibende Konzept ist der TANZ.
- Die Stile STANDARD-TÄNZE und LATEIN-TÄNZE sind jeweils spezielle Arten von Tänzen.
- Jeder Tanz hat einen NAMEN und einen TAKT.
- Jeder Tanz besteht aus mehreren FIGUREN.
- Jede Figur hat einen NAMEN.
- Jede Figur ist entweder eine textuelle BESCHREIBUNG oder eine SEQUENZ von anderen (kleineren) Figuren.
- Der WALZER, der TANGO und der QUICKSTEP sind jeweils konkrete Instanzen von Standard-Tänzen.
- Der CHACHACHA, die RUMBA und der JIVE sind jeweils konkrete Instanzen von Latein-Tänzen.
- Der Walzer wird im "3/4"-Takt getanzt, alle anderen Tänze im "4/4"-Takt.
- Es gibt die Figuren GRUNDSCHRITT, RECHTSDREHUNG, KREISELDREHUNG, PROMENADE, CHASSÉ, FAN und WISCHER.
- Alle Figuren bis auf den Wischer bestehen nur aus einer Beschreibung.
- Der Wischer besteht aus einer Sequenz von Promenade und Chassé.
- Zum Walzer gehören die Rechtsdrehung, die Kreiseldrehung und der Wischer.
- Zum Tango gehört der Grundschritt und die Promenade.
- Zum Quickstep gehört der Grundschritt und die Kreiseldrehung. Unterschiedliche Ausprägungen der Figuren für unterschiedliche Tänze (z.B. der Grundschritt im Tango und im Quickstep) werden in der Modellierung nicht betrachtet und müssen in der Beschreibung behandelt werden (aber nicht von Ihnen!).
- Zum ChaChaCha und zur Rumba gehören der Grundschritt und der Fan.
- Zum Jive gehört der Grundschritt.

a) *Modellieren* Sie diese Struktur zunächst als UML Klassendiagramm und als UML Objektdiagramm (getrennt oder zusammen). Getter- und setter-Methoden können Sie weglassen.

b) *Implementieren* Sie Ihr Modell anschließend als Java-Klassen. Getter- und setter-Methoden können Sie sich von Ihrer IDE generieren lassen (z.B. Eclipse).
Erstellen Sie eine Klasse `DanceDatabase` mit einer `main`-Methode, in der Sie die entsprechenden Instanzen anlegen.

c) Eine Figur, die aus einer Sequenz von anderen Figuren besteht, darf sich nicht selbst direkt oder indirekt enthalten. Z.B. darf der Wischer keinen Wischer enthalten. Er darf aber auch keine andere Sequenz enthalten, die wiederum einen Wischer enthält. Fügen Sie Ihrer *Implementierung* der Sequenz eine Methode `add(figure: Figure): boolean` hinzu, welche diese Bedingung prüft. Sie soll **`true`** zurückgeben, wenn das Einfügen erfolgreich war, ansonsten **`false`**.
Müssen Sie noch weitere Änderungen vornehmen, um diese Bedingung sicherzustellen?

Hinweis: Um zu überprüfen, ob ein Objekt eine Instanz einer bestimmten Klasse ist, können Sie den **`instanceof`**-Befehl verwenden:

```
Number x = new Integer(5);
if (x instanceof Integer) {
  Integer y = (Integer) x;
  // ...
}
```

Falls Sie für Ihre Modellierung englische Begriffe verwenden wollen, können Sie diese folgender Tabelle entnehmen:

deutsch	**englisch**
Tanz	Dance
Standard-Tanz	Standard Dance
Latein-Tanz	Latin Dance
Walzer	Waltz
Tango	Tango
Quickstep	Quickstep
ChaChaCha	ChaChaCha
Rumba	Rumba
Jive	Jive
Name	Name
Takt	Beat
Figur	Figure
Beschreibung	Description
Sequenz	Sequence/Amalgamation
Grundschritt	Basic Movement
Rechtsdrehung	Natural Turn
Kreiseldrehung	Spin Turn
Promenade	Promenade
Chassé	Chassé
Fan	Fan
Wischer	Whisk

Aufgabe 3: *Überladen von Methoden* **(2 + 2 Punkte)**

Die Verwendung der Java-Methode `System.out.println()` ist Ihnen auf Dauer zu aufwändig. Sie wollen sich daher eine Klasse `Out` mit einer statischen Methode `out` definieren, die Sie zum Ausgeben von Werten verwenden können.

a) Verwenden Sie das Konzept des Überladens von Methoden, um `out`-Methoden für **`boolean`**, **`int`**, **`double`**, **`char`**, `String` und `Object` Werte zu definieren. Innerhalb dieser Methoden dürfen Sie die `System.out.println()`-Methode zur Ausgabe verwenden.

b) Sie stellen fest, dass einer Ihrer Kommilitonen eine ähnliche Funktionalität erzielt hat, indem er ausschließlich eine Methode für `Object`-Parameter definiert hat. Warum funktioniert das in diesem Fall, ist aber im Allgemeinen nicht unbedingt sinnvoll?

Eine mögliche Verwendung für Ihre Klasse sieht folgendermaßen aus:

```
Out.out(true);
Out.out(5);
Out.out(5.3);
Out.out('a');
Out.out("OUT");
Out.out(new Object());
```