Assignment 2

Abgabe bis zum 25. April 2023 um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 25. April 2023 um 23:59 Uhr auf folgender Webseite ab:
https://assignments.hci.uni-hannover.de
Ihre Abgabe muss aus einer einzelnen zip-Datei bestehen, die den Quellcode und ggf. ein PDF-Dokument bei Freitextaufgaben enthält. Beachten Sie, dass Dateien mit Umlauten im Dateinamen nicht hochgeladen werden können. Entfernen Sie die daher vor der Abgabe die Umlaute aus dem Dateinamen. Überprüfen Sie nach Ihrer Abgabe, ob der Upload erfolgreich war. Bei technischen Problemen, wenden Sie sich an Kevin Schumann oder Dennis Stanke. Es wird eine Plagiatsüberprüfung durchgeführt. Gefundene Plagiate führen zum Ausschluss von der Veranstaltung. In dieser Veranstaltung wird ausschließlich die Java/JavaFX Version 17 verwendet. Code und Kompilate anderer Versionen sind nicht zulässig. Ihre Abgaben sollen die vorgegebenen Dateinamen verwenden. Wiederholter Verstoß gegen diese Regel kann zu Punktabzügen führen.

Aufgabe 1: Binärer Suchbaum

Innerhalb dieser Aufgabe werden Sie eine Klasse, die einen binären Suchbaum repräsentiert, implementieren. Außerdem werden dafür typische Methoden realisiert. Ein binärer Suchbaum ist ein Binärbaum, bei dem der linke Teilbaum stets kleiner Werte beinhaltet. Größere Werte befindet sich im rechten Teilbaum. Des Weiteren sind beide Teilbäume ebenfalls binäre Suchbäume.

Jede Methode soll durch zwei Testfälle innerhalb der Main-Methode in der Main-Klasse getestet werden.

a) Erstellen Sie zunächst die Klasse Tree in dem unten gezeigten Package. Ein Tree besitzt einen Integer-Wert *value* und sowohl eine linken als auch rechten Teilbaum als Attribut. Implementieren Sie einen Konstruktor, der einen Integer-Wert annimmt und diesen unter *value* speichert. Setzen Sie weiter Getter/Setter-Methoden für alle Attribute um.

```
task1
├─ model
│  └─ Tree
└─ Main
```

b) Zur Ausgabe auf der Kommandozeile solle die Funktion `public String str()` realisiert werden. str() generiert den String *in-order*. Dementsprechend ist die Ordnung innerhalb des Strings: <linker Teilbaum> <Knoten> <rechter Teilbaum>.

Strings können wie folgt konkateniert werden:

```
String a = "Hello";
String b = "World";

// result entspricht "Hello World"
String result = a + " " + b;
```

Vorlesung: apl. Prof. Matthias Becker xmb@hci.uni-hannover.de
Übung: Kevin Schumann k.schumann@stud.uni-hannover.de
Organisation: M.Sc. Dennis Stanke dennis.stanke@hci.uni-hannover.de

c) Implementieren Sie die Methode `public void insertValue(int value)`. Diese Methode nimmt den Integer-Wert *value* entgegen und pflegt diesen in den binären Suchbaum ein, sodass die genannten Eigenschaften weiterhin erfüllt sind. Ist der Wert bereits vorhanden, soll kein neuer Knoten hinzugefügt werden.

d) Implementieren Sie die Methode `public boolean contains(int value)`. Falls der binäre Suchbaum den Wert *value* enthält, gibt die Methode *true* zurück anderenfalls *false*.

e) Um zu überprüfen, ob zwei binäre Suchbäume ausschließlich die gleichen Werte beinhalten, soll die Methode `public boolean hasSameValues(Tree other)` umgesetzt werden. Ist dies der Fall, gibt die Methode *true* zurück anderenfalls *false*.

Aufgabe 2: Marketplace

In dieser Aufgaben sollen Sie ein Marketplace (bspw. das schwarze Brett auf StudIP) erstellen. Dies geschieht inkrementell über mehrere Übungsblätter. Es werden dabei neue Vorlesungkonzepte eingearbeitet. Begründen Sie die Ihre Wahl innerhalb der Main-Methode.

a) Machen Sie sich mit der Namenskonvention für Java-Packages vertraut. Legen Sie anschließend die Packagestruktur für die URL `https://www.uni-hannover.de` und dem Packagenamen *task2* an. In den folgenden Teilaufgaben werden Sie darin weitere Subpackages anlegen.

b) Erstellen Sie die Klasse User im Subpackage *auth*. Die User-Klasse speichert:

- den Benutzernamen (String)
- das Passwort (String)
- ein Item-Array der Größe 10 (Item[])

Erstellen Sie alle sinnvollen Getter- und Setter-Methoden, sowie einen Konstruktor. Weiter sollen Nutzer in der Lage sein Items zu entfernen oder hinzuzufügen. Implementieren Sie hierfür die Methoden `public bool removeItem(Item item)` und `public bool addItem(Item item)`. Beide Methoden geben nur dann *true* zurück falls das Hinzufügen oder Entfernen erfolgreich war.

c) Nun implementieren Sie die Klasse `Item` im subpackage *offerings*. Die Klasse soll folgende Attribute haben:

- Name (String)
- Verkaufspreis (Float)
- Verkäufer (User)
- Beschreibung (String)

Implementieren Sie erneut sinnvolle Getter- und Setter-Methoden als auch einen Konstruktor. Machen Sie sich hierbei Gedanken darüber, wer ein Item überhaupt ändern darf und welche Werte der Verkaufspreis annehmen kann.

d) Erstellen Sie für die Klasse `Marketplace` und insbesondere einen Konstruktor im Package *task2*. Diese Klasse speichert ausschließlich bis zu 10 User. Durch die Methoden `public bool addUser(User user)` und `public bool removeUser(User user)` wird das Hinzufügen und Entfernen von Benutzern ermöglicht. Im Erfolgsfall geben beide Methoden *true* zurück.

Ergänzen Sie in jeder erstellten Klasse eine `public String str()` Methode, die das Objekt als String repräsentiert (Bspw: [Item: Tasche; Preis: 10.99; Verkäufer: Max Mustermann; Beschreibung: Eine Tasche]). Die Marketplace-Repräsentation soll alle Items kategorisiert nach den Usern beinhalten.

Erstellen Sie innerhalb der Main-Klasse einen Marketplace mit zwei Usern, die jeweils zwei Items anbieten. Geben Sie den Marketplace auf der Konsole aus, ändern Sie anschließend einen Preis, entfernen Sie einen User und und geben Sie den Marketplace erneut auf der Konsole aus.

Aufgabe 3: Debugging

In diesen Aufgaben werden Sie einen fehlerhaften Java Codeabschnitte bekommen. Diese Fehler können **syntaktisch** oder **semantisch** sein. Jedoch beschreiben Kommentare beschreiben immer das **korrekte** Verhalten des Programms. Sie dürfen den Code kompilieren und ausführen um Fehler zu finden. Arbeiten Sie in den Template-Dateien in der `Debug02.zip`.

a) Der Code enthält 6 Fehler. Kompilieren und führen Sie den Code aus. Suchen Sie anhand der Fehlermeldungen des Compilers oder der Laufzeitfehlermeldungen Fehler im Code und korrigieren Sie diese. Beschreiben Sie am Ende der jeweiligen Zeile, was Sie korrigiert haben.

Erstellen Sie während der Fehlerkorrektur einen Blockkommentar unter dem Code. Schreiben Sie die Zeile(n) des Fehlers auf und beschreiben Sie den Fehler. Kopieren sie die Fehlermeldung, sofern es eine gab, anhand welcher Sie den Fehler erkannt haben. Die Dokumentation soll wie in folgendem Beispiel aussehen:

```
1  public class Debug { //K: class falsch geschrieben (ckass)
2
3  ...
4
5  /*
6  ...
7  Zeile 1: class Keyword falsch geschrieben (ckass):
8  Fehlermeldung:
9  **************
10 Debug.java:1: error: class, interface, or enum expected
11 public ckass Debug {
12        ^
13 **************
14 Der Compiler erwartet eines der drei oben angegebenen Keywords, hat aber nur das falsch
       ↪ geschriebene ckass bekommen.
15 ...
16 */
```