Assignment 5

Abgabe bis zum 20. Mai 2025 (Dienstag) um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 20. Mai 2025 (Dienstag) 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 Sebastian Preuss. 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: Schatzsuche

Innerhalb dieser Aufgabe werden Sie einen endlichen deterministischen Automaten implementieren. Endliche Automaten bestehen aus einer endlichen Menge an Zuständen (States). Abhängig vom aktuellen Zustand geht man bei einer Eingabe (später Aktion) entweder in einen anderen Zustand über oder bleibt im selben Zustand. Deterministisch bedeutet, dass für jede Eingabe in dem aktuellen Zustand ein einzigartiger Folgezustand erreicht wird. Endlichen Automaten haben einen Startzustand und ggf. sogar mehrere Zielzustände. Ziel ist es den in Figure 1 gezeigten Automaten zu realisieren. States werden durch Inseln repräsentiert. Pfeile stellen Transitionen dar. Es sind in jedem State die Eingaben A und B möglich. Nicht eingezeichnete Pfeile sind Transitionen, die auf den selben Zustand verweisen.

Legen Sie selbständig eine geeignete Packagestruktur an. Schreiben Sie für alle Klassen, Methoden und Member-Variablen, die Sie erstellen oder ändern, JavaDoc Kommentare. Halten Sie sich an die bisherige JavaDoc Konvention.

a) Implementieren Sie den Enum-Typen *Action*, welche die Werte ”A” und ”B” annehmen kann. Erstellen Sie ebenfalls die Methode `public char str()`, welche eine eindeutige Repräsentation des Wertes zurück gibt.

b) Erstellen Sie das Interface *State*, welches die Methoden `public State transition(Action action)`, `public String str()` und `public String info()` darstellt. Die Methode *transition*, soll die Transition abhängig vom aktuellen Zustand und der übergebenen Aktion realisieren. Die Methode gibt den Folgezustand zurück. *Info()* gibt die Bezeichnung des aktuellen Zustands sowie die Übergangsmöglichkeiten (bspw. Aktion a -> Shipwreck Bay; Aktion b -> Musket Hill) als String zurück. *Str()* returned lediglich die Bezeichnung des jeweiligen Zustands.

c) Erstellen Sie für jeden in Figure 1 gezeigten Zustand eine Klasse mit dem jeweiligen Zustandsnamen (Inselnamen), welches das Interface aus b) implementiert.

d) Implementieren Sie die Klasse *StateMachine*, welche den kompletten Automaten darstellt. StateMachine speichert die aktuellen Zustand als Attribut und ob der Zielzustand erreicht wurde. Der Startzustand lautet *Pirates' Island*, der Zielzustand *Treasure Island*. Ergänzen Sie eine main Methode innerhalb derselben Klasse

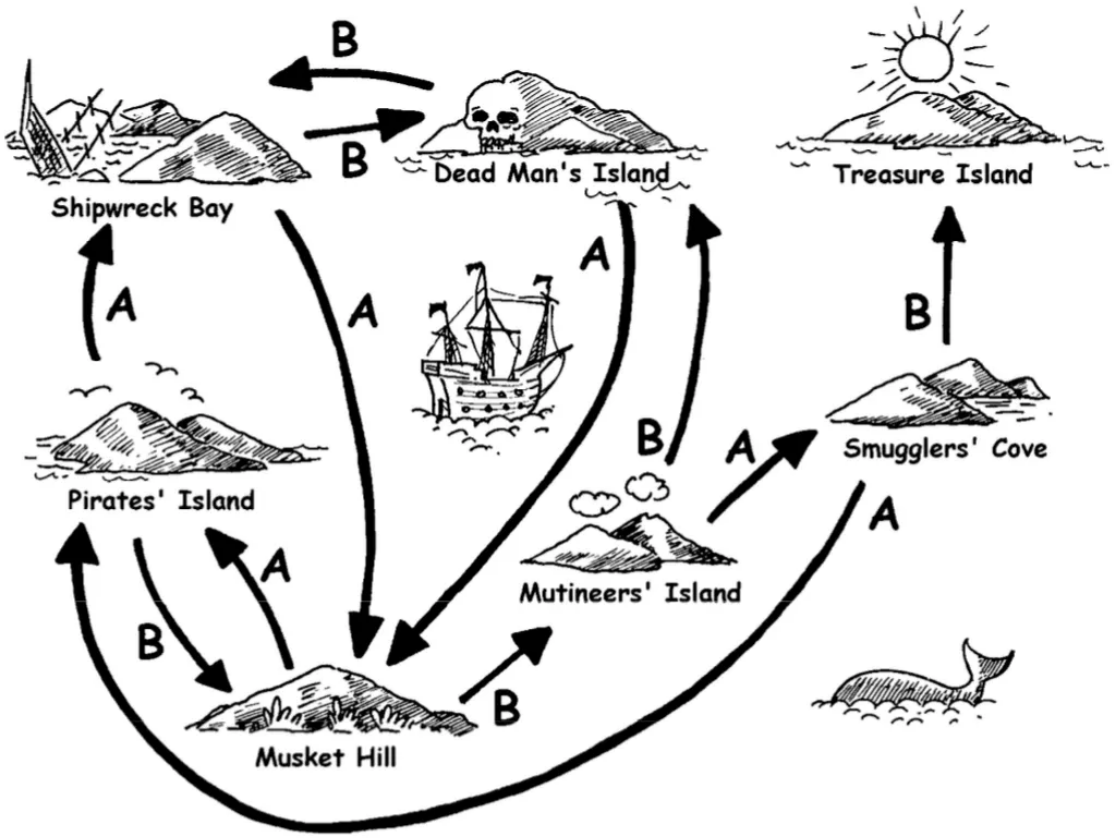

Figure 1: Endlicher Automat: States werden durch Insel repräsentiert. Pfeile stellen Transitionen dar. Es sind in jedem State die Eingabe A und B möglich. Nicht eingezeichnete Pfeile sind Transitionen, die auf den selben Zustand verweisen. Quelle

um ein Kommandozeileninterface, welches den Nutzer solange nach der Wahl zwischen den Aktionen "A" und "B" fragt, bis der Zielzustand erreicht wurde. Nach jedem Zustandsübergang soll die Info des aktuellen Zustands ausgegeben werden. Wurde der Zielzustand erreicht soll der komplette bewältigte Weg von dem Startzustand bis zum Endzustand ausgegeben werden.

Aufgabe 2: Marketplace - Online

Nutzen Sie für die Bearbeitung dieser Aufgabe das Template 'task2.zip', die im Stud.IP Dateibereich der Übung hinterlegt ist. Bearbeiten Sie diese Aufgabe in der vorgegebenen Reihenfolge (a), b),...h))

Diese Woche soll das Projekt angepasst werden, sodass es online funktioniert. Hierfür wurde die Musterlösung um die Klasse 'Communicator' im Package 'communication' erweitert. Diese Klasse realisiert die Kommunikation mit einem bereits existenten Server. Zunächst sind die Funktionen auskommentiert, da sonst das Projekt nicht kompiliert. Schrittweise sollen die Funktionen in den Teilaufgaben einkommentiert werden.

Während der Bearbeitung werden manche Methoden, Attribute, und sogar Klassen nicht mehr von Nöten sein. Diese sollen erst am Ende der Aufgabe entfernt werden.

Das Template kann wie folgt kompiliert/ausgeführt werden:

```
// Linux / MacOS
javac -cp .:task2/json.jar task2/Marketplace.java
java -cp .:task2/json.jar task2/Marketplace

// Windows
javac -cp .;task2/json.jar task2/Marketplace.java
java -cp .;task2/json.jar task2/Marketplace
```

a) Die Funktion `public User login()` durchsucht derzeitig die User, die in der Marketplace Klasse hinterlegt sind. Ändern Sie die Methode derartig, sodass stattdessen der Server gefragt wird, ob ein User mit demjenigen Benutzername und Passwort registriert ist. Ist der User vorhanden, soll der Benuzername unter dem Attribut *username* in der Marketplace Klasse gespeichert werden. Die Login-Funktion soll ab jetzt nichts mehr zurückgeben. Nutzen Sie die Methode `public static boolean login(String username, String password)` aus der 'Communicator' Klasse. Kommentieren Sie diese zunächst wieder ein.

Testen Sie, ob der Login für den Nutzer "test" mit dem Password "test" funktioniert.

b) Modifizieren Sie die gleiche Funktion, sodass der Benutzer zunächst gefragt wird, ob er sich einloggen oder sich registrieren möchte. Möchte sich der Nutzer registrieren, registrieren Sie ihn. Ist die Registrierung mit dem eingegebenen Benutzernamen nicht möglich, soll ein anderer Benutzername gewählt werden. Nach der Registrierung soll sich das Programm beenden. Nutzen Sie die Methode `public static boolean register(String username, String password)` aus der 'Communicator' Klasse. Kommentieren Sie zunächst diese wieder ein.

Registrieren Sie einen eigenen User, den Sie für die folgenden Teilaufgaben nutzen sollen. Benutzen Sie keine echten Daten!

c) Aus Kompatibiliätsgrunden muss die Item Klasse angepasst werden. Statt `User owner` speichern wir nur den Benutzernamen `String owner` (analog zu a)). Passen Sie alle damit verbundenen Funktionen bzw. Funktionsaufrufe an.

Items werden von dem Server durch eine Identifkationsnummer (int) identifiziert. Fügen Sie deshalb das Attribut `int itemId` der Klasse Item hinzu. Erweitern Sie den bereits vorhandenen Konstruktor, sodass itemId als -1 initialisiert wird. Erstellen Sie einen weitern Konstruktor in der Form `public Item(int itemId, String name, float price, String owner, String description, Category category)`. Ergänzen Sie die Getter-Methode `public int getItemId()`.

d) Die Funktion `public void addItemMenu()` fügt derzeitig Items hinzu, indem dem eingeloggten User (this.user) ein Item hinzugefügt wird. Stattdessen müssen wir dem Server mitteilen, welchem User welches Item zugefügt wird. Nutzen Sie die Methode `public static boolean addItem(String username, Item item)` aus der 'Communicator' Klasse. Kommentieren Sie diese zunächst wieder ein.

e) Die Funktion `public void searchMarketplace()` nutzt aktuell die Funktion `public void filterMarket(Category category)`. Stattdessen soll der Server nach dem Items mit derjenigen Kategorie gefragt und dise anschließend ausgegeben werden. Nutzen Sie die Methode `public static Item[] getItems(Category category)` aus der 'Communicator' Klasse. Kommentieren Sie zunächst die Funktionen `public static Item[] getItems(Category category)` und `public static Item[] getItemsWith(Category category, String username)` wieder ein.

Der Server gibt maximal 20 Items mit einer Anfrage zurück, damit ihre Kommandozeile nicht gesprengt wird.

f) Die Funktion `public void removeItemMenu()` iteriert derzeitig über die Items einer Users-Objekts und entfernt das vom Benuzer ausgewählte Item aus dem Array. Nun soll anstelle davon die Items des Users vom Server abgefragt werden. Das vom Benutzer ausgewählte Item wird dem Server zum löschen mitgeteilt. Nutzen Sie die Methode `public static Item[] getUserItems(String username)` und `public static boolean removeItem(Item item)` aus der 'Communicator' Klasse. Kommentieren Sie zunächst diese Funktionen wieder ein.

g) Setzen Sie auch das Ändern von Items auf dem Server um. Dies funktioniert ganz analog zur vorherigen Teilaufgabe. Statt `public static boolean removeItem(Item item)` wird jedoch `public static boolean updateItem(Item item)` genutzt. Diese Funktion ist ebenfalls einzukommentieren.

h) Entfernen Sie alle Klasse, Attribute und Methoden, die nicht zur Umsetzung des Kommandozeileninterfaces benötigt werden.

Aufgabe 3: Debugging

In diesen Aufgaben werden Sie einen fehlerhaften Java Codeabschnitt bekommen. Diese Fehler können **syntaktisch** oder **semantisch** sein. Es kann sich dabei um Compiler- oder Laufzeitfehler handeln. Sie dürfen den Code kompilieren und ausführen um die Fehler zu finden. Arbeiten Sie in der Template-Datei `Debug5.zip`. Diese finden Sie im Stud.IP.

a) Der Codeabschnitt enthält 5 Fehler darunter auch fehlende Methoden. Kompilieren und führen Sie den Code aus. Suchen Sie anhand der Fehlermeldungen des Compilers oder der Laufzeitfehlermeldungen Fehler im Code und korrigieren Sie diese. Kommentieren Sie am Ende der jeweiligen Zeile, was Sie korrigiert haben.

Erstellen Sie während der Fehlerkorrektur einen Blockkommentar unter dem Code, in dem Sie die Fehler dokumentieren. 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:

```
public class Debug { //K: class falsch geschrieben (ckass)

...

/*
...
Zeile 1: class Keyword falsch geschrieben (ckass):
Fehlermeldung:
**************
Debug.java:1: error: class, interface, or enum expected
public ckass Debug {
       ^
**************
Der Compiler erwartet eines der drei oben angegebenen Keywords, hat aber nur das falsch
    ↪ geschriebene ckass bekommen.
...
*/
```

Im Code vorgegebene Kommentare beschreiben immer das **korrekte** Verhalten des Programms.