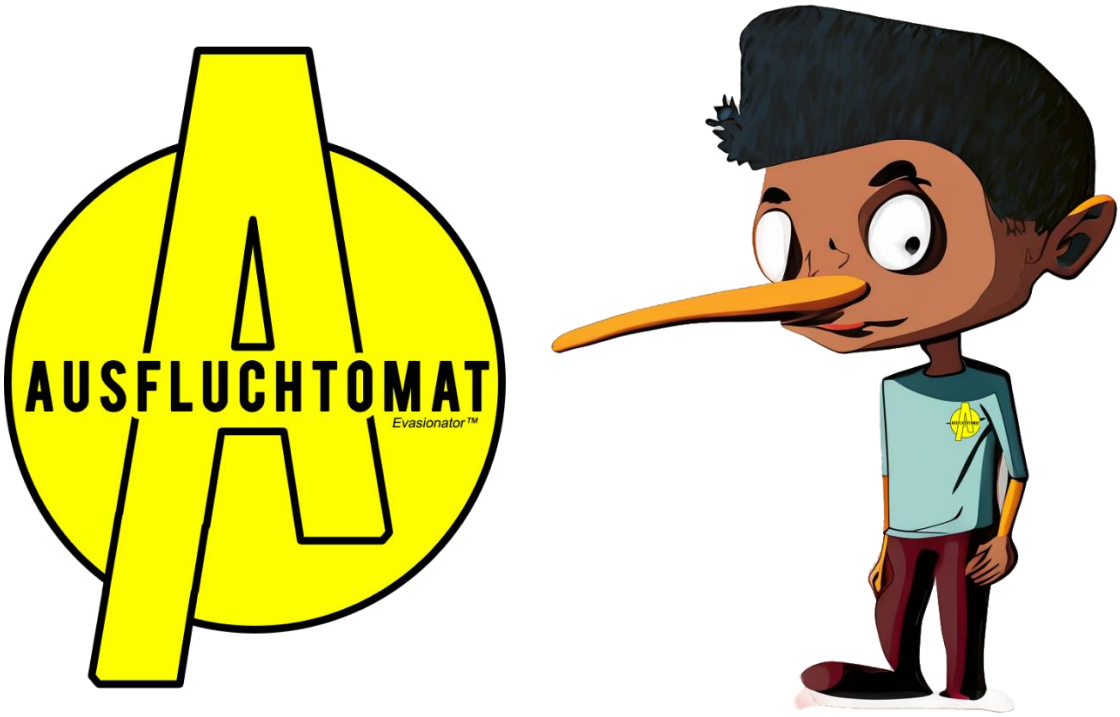
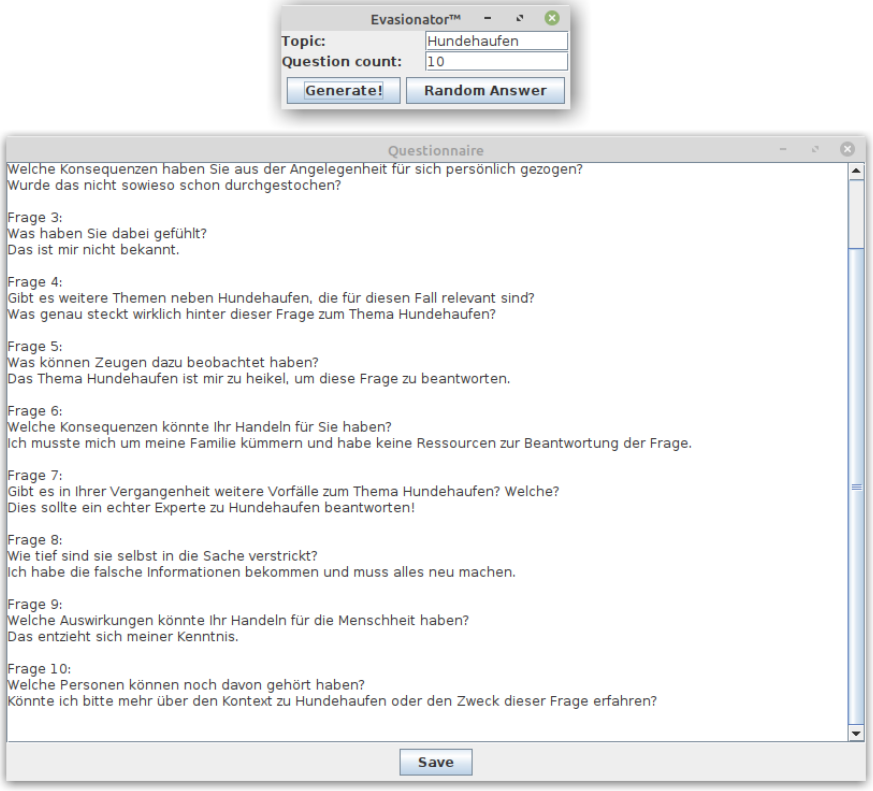




Hinweis zur Bewertung:

- 1/4 der Punkte (25%) wird nach Funktionstests Ihrer Lösung vergeben.
- 3/4 der Punkte (75%) werden entsprechend des in den Teilaufgaben angegebenen Schlüssels auf Basis des Quellcodes vergeben.

Logo-Männchen-Bild (ohne Badge) erzeugt mit Adobe Firefly (beta), Prompt: „Generate an image of a person with short legs and a very long, Pinocchio-like nose“

DHBW Karlsruhe, Vorlesung Programmieren I+II, Programmentwurf

Freitag, 13.10.2023

Bearbeitungszeit: 120 Minuten

Aufgabe

Heutzutage ist es gang und gäbe, dass man sich in verantwortungsreichen und für die Öffentlichkeit sichtbaren Positionen im Falle von eigenem Fehlverhalten eine geschickte Strategie zurechtlegen muss, wie man sich möglichst unbescholten aus der Affäre ziehen kann. Insbesondere unsere Politiker geben dabei momentan überwiegend kein gutes Bild ab. Daher wollen wir diese mit einer Software unterstützen, welche automatisiert in kürzester Zeit Ausflüchte (englisch: Evasion) für verschiedene Situationen generieren kann, die mit Sicherheit auch nicht schlechter sind als die, die man in den Medien liest oder hört. Schreiben zu diesem Zweck eine Java-Anwendung **`Evasionator`**, welche einen Fragenkatalog samt Antworten zu einem (fast) beliebigen Thema automatisch generieren kann.

Teilaufgabe a) **[16%]**

Schreiben Sie einen *komplexen Aufzählungstyp* **`QuestionType`**, mit welchem die unterschiedlichen Fragetypen abgebildet werden. Neben der eigentlichen Konstante besitzt der Fragentyp ein zusätzliches Label (`label`), welches den Fragetyp zusätzlich beschreibt.

Folgende Fragetypen sollen implementiert werden:

QuestionType	Label
`YES_NO`	`Yes/No question`
`WITNESS`	`Witness may be involved`
`GENERAL`	`General question`

Schreiben Sie außerdem eine Klasse **`Question`**, die eine Frage repräsentiert. Eine Frage besteht aus einem Fragetext (`text`), einem Attribut, welches anzeigt, ob es sich bei der Frage um eine Startfrage handelt oder nicht (`start`) und dem Typ der Frage (`type`). Wählen Sie sinnvolle Typen für diese Attribute!

Schreiben sie weiterhin eine Klasse **`Evasion`**, durch die eine Ausflucht dargestellt wird. Eine Ausflucht besteht aus einem Text (`text`), der die eigentliche Ausflucht wiedergibt, sowie aus einer Anzahl von Fragetypen (`types`), auf die sich die Ausflucht beziehen kann. Verwenden Sie auch hier sinnvolle Typen für die Attribute und ergänzen Sie außerdem eine Methode `public boolean isEvasionFor(Question q)`, welche für die übergebenen Frage zurückgibt, ob deren Typ in der Liste der möglichen Fragetypen dieser Ausflucht enthalten ist.

Teilaufgabe b) **[9%]**

Erstellen Sie eine Java-Schnittstelle **`AnswerGenerator`**, mit deren Hilfe Antwortgeneratoren umgesetzt werden können. Die Schnittstelle besitzt die folgenden beiden Methoden:

- `String generateAnswer(Question q, String topic)`: Wählt aus den vorhandenen Ausflüchten solange *zufällig* eine aus, bis deren Typ zur übergebenen Frage passt. Ist diese gefunden, wird daraus der fertige Antworttext *(siehe Hinweis unten!)* erstellt und zurückgeliefert.
  Sollte beim 10. Versuch noch keine zum Fragentyp passende Antwort gefunden worden sein, geben Sie stattdessen eine Standardfloskel zurück, z.B. „Keine Antwort ist auch eine Antwort!“.
- `Evasion getRandomEvasion()`: Gibt eine *zufällige* Ausflucht aus den zur Verfügung stehenden Ausflüchten zurück.

Schreiben Sie außerdem eine Klasse **`ListAnswerGenerator`**, welche die zuvor angefertigte Schnittstelle implementiert. Die Klasse besitzt als Attribut eine Liste von Ausflüchten (`evasions`), die der Generator zur Realisierung der in der Schnittstelle definierten Methoden verwenden soll. Implementieren Sie einen Konstruktor, welcher die Liste von Ausflüchten entgegennimmt und setzt (vgl. Verwendung in bereitgestellter Klasse `Evasionator`).

Hinweis: In den Antworttexten können ein oder mehrere Platzhalter für das Thema („{{TOPIC}}“) vorkommen. Um diese Platzhalter in den Ausflüchten mit dem übergebenen Thema zu ersetzen, nutzen Sie die statische Methode `Evasionator.replaceTopic( text, topic )`, welche Ihnen den „fertigen“ Ausflucht-Text liefert.

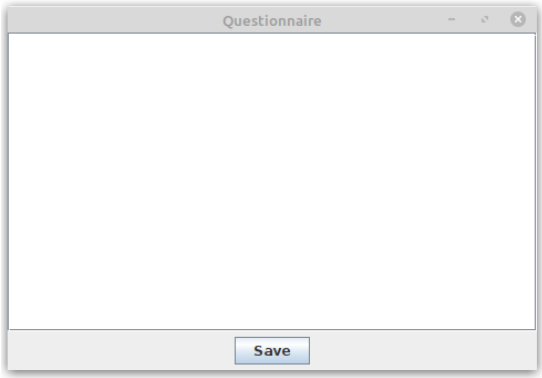
Teilaufgabe c) **[11%]**

Schreiben Sie eine Klasse **`QuestionCatalogue`**, welche eine Repräsentation des Fragenkatalogs ist. Der Fragenkatalog unterscheidet bei den Fragen jeweils allgemeine Fragen und Startfragen, welche speziell zu Beginn eines Fragebogens genutzt werden. Diese sollen in unterschiedlichen Datenstrukturen verwaltet werden. Die Klasse soll folgende Methoden besitzen:

- `public void addQuestion(Question q)`: Fügt dem Fragenkatalog eine Frage hinzu. Dabei soll geprüft werden, ob es sich bei der übergebenen Frage um eine Startfrage handelt. Je nachdem soll die Frage dann entweder der Datenstruktur für die Startfragen oder den allgemeinen Fragen hinzugefügt werden.
- `public List<Question> getStartQuestions()`: Gibt alle Startfragen zurück.
- `public List<Question> getQuestions()`: Gibt alle allgemeinen Fragen zurück.
- `public List<Question> getQuestions(int n)`: Gibt eine Liste mit *n* Fragen zurück. Dabei soll eine Startfrage am Anfang der Liste stehen, gefolgt von *n-1* allgemeinen Fragen. Der Katalog soll keine Duplikate beinhalten. Prüfen Sie außerdem zu Beginn, ob sich im Fragenkatalog genügend Fragen befinden. Sollte dies nicht der Fall sein, soll eine sprechende **`EvasionException`** geworfen werden, die ebenfalls von Ihnen implementiert werden soll.

Teilaufgabe d) **[10%]**

Schreiben Sie eine Klasse **`QuestionnaireTerm`**, in welcher die Fragen und Antworten zu einem bestimmten Thema in einer grafischen Oberfläche angezeigt werden (vgl. Screenshot rechts). Das Fenster soll den Titel „Questionnaire“ haben und als zentrales Anzeigeelement eine Textfläche (`JTextArea`) besitzen, deren Inhalt generiert und eingefügt wird. Daher soll die Textfläche vom Nutzer nicht veränderbar sein (siehe Hinweise). Außerdem soll die Textfläche einen Scrollbalken besitzen, mit dessen Hilfe die Fragen und Antworten angezeigt werden können, die nicht in das initiale Fenster passen (siehe Hinweise).



Implementieren Sie außerdem eine Methode `public void setQuestionnaire(String text)`, die die Textfläche auf den übergebenen Inhalt setzt!

Weiterhin soll das Fenster einen Button zum Speichern des Fragebogens besitzen. Dieser soll am unteren Rand des Fensters eingefügt werden.

Beim Klicken des Buttons soll zunächst überprüft werden, ob das Textfeld leer ist. In diesem Fall soll ein Fehlerdialog ausgegeben werden, der den Nutzer darauf hinweist, dass ein leerer Fragebogen nicht abgespeichert werden kann. Andernfalls soll der Inhalt des Fragebogens in eine Datei mit dem Namen „`questionnaires.txt`“ geschrieben und darin abgespeichert werden. Bei weiteren Fragebögen, die gespeichert werden sollen, sollen diese an den bestehenden Inhalt der Textdatei *angehängt* werden. Fügen Sie der Übersicht halber außerdem ein Trennzeichen ein, sodass man erkennen kann, wo der neue Fragebogen beginnt.

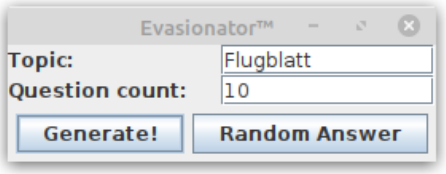
Hinweis: Nutzereingaben für eine `JTextArea` können mittels `textarea.setEditable(false)` unterbunden werden.

Hinweis: Damit bei Bedarf Scrollbalken eingeblendet werden fügen Sie statt direkt der `JTextArea` ein `JScrollPane` hinzu, welches die `JTextArea` beeinhaltet, bspw: `parent.add(new JScrollPane(textArea))`.

Hinweis: Die Funktionalität wird in Teilaufgabe g) noch erweitert!

Teilaufgabe e) **[16%]**

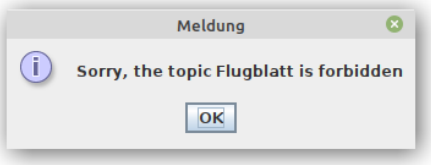
Schreiben Sie eine Klasse **`EvasionatorTerm`**, welche eine grafische Oberfläche für die Generierung eines Fragebogens bereitstellt. Der Konstruktor nimmt einen Fragenkatalog, einen Antwort-Generator sowie einen `QuestionnaireTerm` entgegen (vgl. Verwendung in bereitgestellter Klasse `Evasionator`).



Der Titel des Fensters soll auf „Evasionator ™“ gesetzt werden (™ optional, `\u2122`). Wie abgebildet (vgl. Screenshot), besteht das Fenster aus zwei Labels, welche das Thema sowie die Anzahl der Fragen anzeigen. Neben den Labels sind jeweils Textfelder vorzusehen, die dazu dienen, Werte für die beiden Eingaben entgegenzunehmen.

Darüber hinaus besitzt das Fenster einen Button *Generate*, der dazu dient, einen Fragebogen zu generieren und diesen an das `QuestionnaireTerm` weiterzuleiten. Beim Klicken des Buttons ist zunächst zu prüfen, ob es sich bei dem gewählten Thema um ein Tabu-Thema handelt, zu dem kein Fragebogen erstellt werden darf (siehe Hinweis).

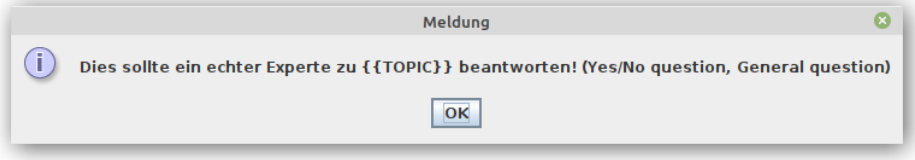
Ist es ein Tabu-Thema, so soll ein Dialogfenster den Nutzer darauf hinweisen, dass er ein unzulässiges Thema für seinen Fragebogen gewählt hat.



Andernfalls soll die eingegebene Anzahl an Fragen aus dem Fragenkatalog entnommen werden. Zu jeder Frage soll außerdem eine Antwort generiert werden. Dabei sind gleiche Antworten auf verschiedene Fragen ausdrücklich *erlaubt*, schließlich ist es nicht das Ziel des Ausfluchtomats, eine besonders differenzierte Abfolge von Fragen und Antworten zu liefern! Der gesamte, daraus entstehende Text soll anschließend dem `QuestionnaireTerm` zur Anzeige übergeben werden. Versehen Sie den Text mit etwas Formatierung (vgl. Screenshot Deckblatt, mindestens Nummer, Text und Antwort zur Frage, sowie eine Leerzeile zwischen den Fragen, siehe Hinweis).

Sollte beim Erzeugen des Fragebogens ein beliebiger Fehler (bspw. durch zu wenige Fragen im Katalog oder eine ungültige Eingabe im Textfeld für die Anzahl) auftreten, so ist dieser abzufangen und dessen Nachricht als Dialog anzuzeigen.

Implementieren Sie außerdem einen Button *Random Answer*, welcher eine zufällige Ausflucht vom Antwort-Generator holt und diese anschließend inkl. einer Information darüber, für welche Art von Fragen sich die Ausflucht eignet, mit Hilfe eines Dialogfensters dem Nutzer ausgibt.



Hinweis: Nutzen Sie für das Kontrollieren von Tabuthemen die vorgegebene statische Methode `Evasionator.containsUnreasonableKeyword`*!*

Hinweis: Leerzeilen in Zeichenketten können bspw. mit „ `\n` *“ eingefügt werden.*

Teilaufgabe f) **[8%]**

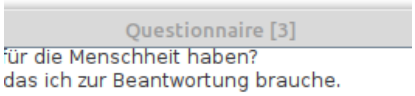
Ändern Sie die `loadEvasions`-Methode der Klasse `Evasionator` so ab, dass die Ausflüchte aus der bereitgestellten Textdatei `evasions.txt` zeilenweise eingelesen werden und *anstatt* der bereitgestellten Testdaten verwendet werden. Für das Parsen der Textzeilen in `Evasion`-Objekte nutzen Sie die zur Verfügung gestellte Methode `parseEvasion` aus der Klasse `Evasionator`!

Ändern Sie außerdem **zusätzlich** die `loadQuestions`-Methode so ab, dass analog wie oben die Fragen anstatt der Testdaten aus der bereitgestellten Textdatei `questions.txt` eingelesen werden. Auch hierfür stellt die Klasse eine Methode `parseQuestion` bereit, die Sie für das Umwandeln der Fragen verwenden können!

Teilaufgabe g) **[5%]**

Damit der für Ausflüchte ganz entscheidende Faktor „Bauchgefühl“ zum Tragen kommt, erweitern Sie die `setQuestionnaire`-Methode der Klasse `QuestionnaireTerm` nun so, dass beim Setzen eines Textes dieser nicht mehr beliebig lange angezeigt wird, sondern der Nutzer spontan entscheiden muss ob dieser gespeichert werden soll.

Zählen Sie hierfür im Titel des Fensters einen 10-Sekunden-Countdown herunter (vgl. Screenshot) und leeren Sie die Textfläche nach Ablauf des Countdowns.



Diese Aktion soll nebenläufig geschehen! Die Nebenläufigkeit endet entweder wenn der Countdown endet und die Textfläche geleert wurde oder wenn ein neuer Text vor Ablauf der 10 Sekunden gesetzt wird und ein neuer nebenläufiger Zähler startet.

Allgemeine Hinweise

Starten
Starten Sie die Anwendung mit der gegebenen Klasse `Evasionator` (siehe USB-Stick).

Schließen eines Fensters
Beim Schließen eines Fensters soll die komplette Anwendung beendet werden.

Sichtbarkeit von Instanz-Attributen
Sämtliche Instanz-Attribute sind als privat zu definieren und von außerhalb der Klasse ggf. mittels Getter- und/oder Setter-Methoden zu verwenden.