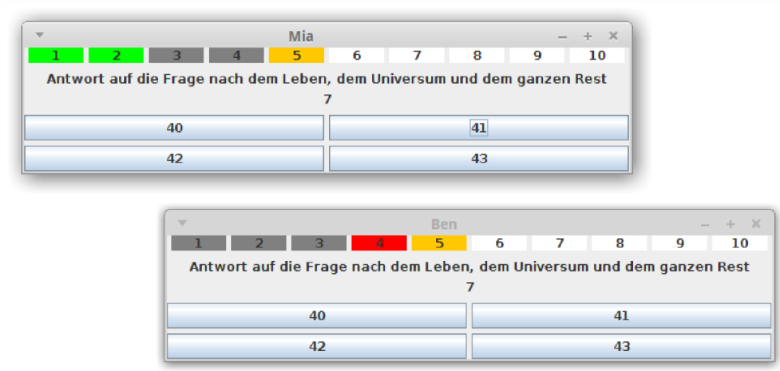
Bereich: 1. + 2. Semester

SpeedyQuiz

Package: `de.dhbwka.java.exercise.speedy`



Aufgabenstellung:

Schreiben Sie eine Java-Anwendung „*SpeedyQuiz*“, die es einem Spieler ermöglicht allein zu „quizzen“ oder gegen andere anzutreten!

Für jede Frage steht dabei nur ein kurzer Antwortzeitraum zur Verfügung. Ist die Zeit abgelaufen oder gibt ein Spieler eine Antwort, ist die aktuelle Fragerunde für alle Spieler beendet und die nächste Frage erscheint direkt im Anschluss.

Teilaufgabe a)

Schreiben Sie eine Klasse `Question` zur Repräsentation einer Frage! Eine Frage besteht aus dem Fragentext (`questionText`), den vier möglichen Antworten (`answers`) sowie dem Index der korrekten Antwort (`correctIndex`, 0-basierte Zählweise). Wählen Sie jeweils geeignete Datentypen!

Definieren Sie auch einen Konstruktor, dessen Struktur durch die bereitgestellte Klasse `SpeedyQuiz` (siehe Anhänge) bereits vorgegeben ist!

Teilaufgabe b)

Um im Falle eines großen Erfolgs des Spiels für Erweiterungen (die dann natürlich kostenpflichtig angeboten werden) gerüstet zu sein, definieren Sie zunächst die allgemeine Schnittstelle, die jeder Client für einen Spieler bedienen muss. Implementieren Sie hierfür ein Java-*Interface* `GameClient`, welches folgende Methoden definiert:

- `public String getPlayerName()`: liefert den Namen des Spielers
- `public int getPoints()`: liefert die Anzahl der Punkte (jede richtige Antwort zählt 1 Punkt, für jede falsche Antwort wird ein Punkt abgezogen. Wäre die Gesamtpunktzahl negativ ist 0 zurückzugeben)
- `public void setQuestion(int questionIndex, Question q)`: übergibt dem Client den 0-basierten Index der Frage (in der Liste der für das Spiel ausgewählten Fragen), sowie die Fragen-Instanz. Beim Aufruf soll jeder Client implizit für die Frage in den Zustand `ACTIVE` (siehe Tabelle unten) übergehen.

- `public void setRemainingSeconds(int seconds)`: teilt dem Client die Anzahl der verbleibenden Sekunden für die aktuelle Frage mit.
- `public void gameIsOver()`: teilt dem Client mit, dass die letzte Frage gespielt und das Spiel zu Ende ist
- `public void setAnswerState(int questionIndex, Status status)`: teilt dem Client mit, welchen Zustand er für die Frage mit dem gegeben, 0-basierten Index annehmen soll.

Der Zustand soll in Form eines *komplexen Aufzählungstyps* (`Status`) realisiert werden und neben dem Zustand auch die Farbcodierung sowie die zu verrechnenden Punkte für diesen Zustand festlegen:

Zustand	Farbe	Punkte	*Bedeutung*
ACTIVE	`Color.ORANGE`	0	*aktuell gestellte Frage*
CORRECT	`Color.GREEN`	+1	*richtig beantwortete Frage*
WRONG	`Color.RED`	−1	*falsch beantwortete Frage*
PENDING	`Color.WHITE`	0	*noch nicht gespielte/ausstehende Frage*
NO_ANSWER	`Color.GRAY`	0	*Frage auf die ein anderer Spieler geantwortet hat*

Sollte eine Implementierung des *Interface* `GameClient` eine Benutzeroberfläche bereitstellen (bspw. `GameTerm`, vgl. *Teilaufgabe d)*), ist diese selbstverständlich bei *jeder* der o.g. Methoden-Implementierungen entsprechend zu aktualisieren (bspw. auf der UI sichtbarer Fragentext, Antworten, etc. bei `setQuestion`).

Teilaufgabe c)

Schreiben Sie eine Klasse `Game`, welche als Verwaltungsinstanz für ein Spiel dienen soll! Sie verwaltet sowohl den Fragenkatalog, als auch sämtliche angemeldeten Clients und nimmt die Auswahl der Antworten im laufenden Spiel entgegen.

Zur Initialisierung soll ein Konstuktor, welcher die Liste möglicher Fragen sowie die Anzahl der Fragen für dieses Spiel entgegen nimmt (vgl. Verwendung in bereitgestellter Klasse `SpeedyQuiz`, siehe Anhänge).

Der Konstruktor soll aus dem bereitgestellten Fragenpool *zufällig* die übergebene Anzahl an Fragen auswählen und für das Spiel speichern. Duplikate sind zu vermeiden.

Sollte die Menge der Fragen nicht ausreichen für die übergebene Anzahl, soll eine sprechende `GameException` geworfen werden, der die Nachricht „`Too few questions available`“ übergeben wird.

Für den weiteren Verlauf des Spiels sollen zum Konstruktor noch (mind.) folgende Methoden implementiert werden:

- `public void registerClient(GameClient client)`: Anmelden eines Clients für das Spiel. Der Client soll nur aufgenommen werden wenn das Spiel noch nicht gestartet wurde.
- `public int getQuestionsCount()`: Liefert Anzahl der Fragen dieses Spiels.
- `public void startGame()`: Startet das Spiel für alle angemeldeten Clients (wird aufgerufen innerhalb der bereitgestellten Klasse `SpeedyQuiz`, siehe Anhänge). Der Aufruf soll allen Clients die erste Frage übermitteln.

- `public void answerSelected(GameClient client, int index)`: Teilt der Spielinstanz mit, dass in einem Client die Antwort mit dem gegebenen Index (0-basierte Zählweise) ausgewählt wurde. Beim Auswählen einer Antwort soll allen Clients ein Zustandsupdate für die aktuelle Frage mitgeteilt werden, d.h.:
  - Der Client, welcher die Antwort ausgewählt hat bekommt bei richtiger Antwort den Zustand `Status.CORRECT`, andernfalls `Status.WRONG`.
  - Alle anderen Clients bekommen für diese Frage den Zustand `Status.NO_ANSWER`.

Im Anschluss an das Zustandsupdate soll direkt die nächste Frage gestellt (d.h. an alle Clients verteilt) werden.

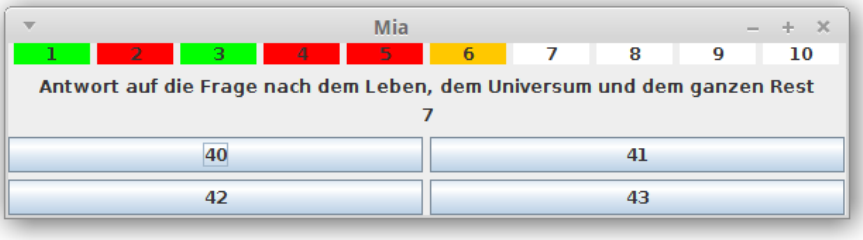
Ist keine weitere Frage mehr vorhanden, ist sämtlichen Clients das Ende des Spiels mitzuteilen.

Hinweis: diese Funktionalität wird in den Teilaufgaben f) und g) erweitert

Teilaufgabe d)

Entwickeln Sie eine Klasse `GameTerm`, welche eine einfache Nutzeroberfläche für ein `SpeedyQuiz`-Spiel erzeugt! Die Nutzeroberfläche ist selbstverständlich ein Client und implementiert daher das Interface `GameClient` und realisiert die Funktionalität der zu implementierenden Methoden gemäß der Beschreibung in *Teilaufgabe b)*.

Dem Konstruktor von `GameTerm` sollen als Parameter der Name des Spielers sowie eine `Game`-Instanz übergeben werden (vgl. gegebene Klasse `SpeedyQuiz`, siehe Anhänge).



`GameTerm` soll ein neues Fenster mit sinnvoll gewählter Größe öffnen. Der Fenstertitel ist auf den übergebenen Namen des Spielers zu setzen.

Im oberen Teil soll für jede Frage des übergebenen Spiels ein Label die Anzeige des Fragenzustands (vgl. *Teilaufgabe a)*) für den aktuellen Spieler eingefügt werden.

Schreiben Sie hierfür eine Klasse `QuestionNumberLabel`, welche von `JLabel` erbt und ein weiteres Attribut für den Zustand der Frage enthält! Wird dieser Zustand aktualisiert soll sich die Hintergrundfarbe entsprechend auf die Farbe des `Status`-Objekts anpassen. Initial ist jedes `QuestionNumberLabel` im Zustand `Status.PENDING` und sein Hintergrund somit weiß.

Hinweis: Damit der Hintergrund des `QuestionNumberLabels` angezeigt wird, rufen Sie in dessen Kontruktor `this.setOpaque(true)` auf. Die zentrierte Darstellung innerhalb eines `JLabel` erreicht man durch Aufruf von `this.setHorizontalAlignment(JLabel.CENTER)`

Unterhalb der Labels für die Zustände der Fragen soll der eigentliche Fragentext angezeigt werden. Direkt darunter dann die verbleibende Zeit in Sekunden (setzten Sie diese zunächst auf 10). Am unteren Rand sind die Buttons für die vier Antwortmöglichkeiten als 2x2-Matrix hinzuzufügen.

Erweitern Sie die Buttons für die Antworten von `GameTerm` nun so, dass bei Auswahl eines Antwort-Buttons der zugehörige Antwort-Index an die Game-Instanz (mit dem aktuellen `GameTerm` als übermittelnden `GameClient`) gesendet wird!

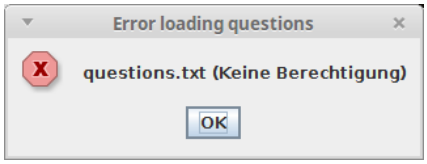
Ist das Spiel zu Ende (Aufruf von `gameIsOver()`) sind alle Buttons zu deaktivieren.

Teilaufgabe e)

Ändern Sie die Implementierung der Methode `loadQuestions` der gegebenen Klasse `SpeedyQuiz` (siehe Anhänge) so, dass die ebenfalls bereitgestellte Textdatei „`questions.txt`“ zeilenweise eingelesen wird! Dabei soll jede Zeile mit Hilfe der gegebenen Methode `parseQuestion` in ein `Question`-Objekt umgewandelt werden.

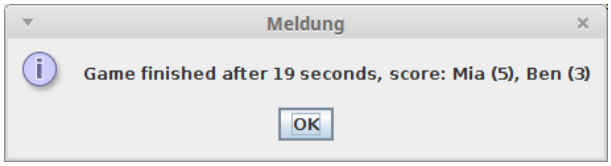
Die so eingelesenen Fragen aus der Datei sollen dann anstatt der bisherigen generierten Fragen zurückgeliefert werden.

Zeigen Sie im Falle eines Fehlers die *Message der Exception* als Nachricht in einer Error-Dialogbox (achten Sie auf das Icon!) mit dem Titel „Error loading questions“ an (im Beispiel fehlen die Lese-Rechte für die Datei):



Teilaufgabe f)

Erweitern Sie die Klasse `Game` dahingehend, dass nach dem Ende des Spiels eine Auswertung stattfindet! Zeigen Sie einen Nachrichten-Dialog, welcher die *Dauer des Spiels* seit dem Aufruf von `startGame` in Sekunden, sowie die *Namen und zugehörigen Punkte der Spieler* anzeigt, bspw:



Die Nachricht des Dialogs soll auch in eine Textdatei mit dem Namen „`highscore.txt`“ geschrieben werden. Sofern die Datei bereits existiert soll die Nachricht in einer neuen Zeile angehängt werden.

Teilaufgabe g)

Erweitern Sie die Klasse `Game` um eine weitere Funktionalität! Diese soll beim Start einer Frage einen nebenläufigen Countdown starten, der von bei 10 beginnend sekundenweise herunterzählt. Jede Sekunde ist allen registrierten `GameClient`-Instanzen die verbleibende Zeit mit Hilfe der `setRemainingSeconds`-Methode mitzuteilen.

Die Nebenläufigkeit endet,

- wenn der Countdown bei `0` angekommen ist (dann ist sämtlichen Clients für die aktuelle Frage der Zustand `Status.NO_ANSWER` zu geben und die nächste Frage zu starten)
- *oder* einer der Spieler eine Antwort gibt

Hinweis: Zur Vereinfachung ist keine Synchronisierung zur Vermeidung von Kollisionen bei quasi-zeitgleichem Countdown-Ablauf und Antwort-Abgabe vorzunehmen

Allgemeine Hinweise

Starten
Starten Sie die Anwendung mit der gegebenen Klasse `SpeedyQuiz` (siehe Anhänge).

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

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

Farben bei MacOS
Um Probleme mit (Hintergrund-)Farben beim Java Standard Look-And-Feel unter MacOS zu vermeiden erzwingt die `main`-Methode der o.g. Klasse `SpeedyQuiz` die Verwendung des *Cross-Platform Look-And-Feel*s. Auf Windows- und Linux-Systemen wäre dies nicht notwendig (ist jedoch auch kein Problem).

Anhänge:

- SpeedyQuiz.java
- questions.txt