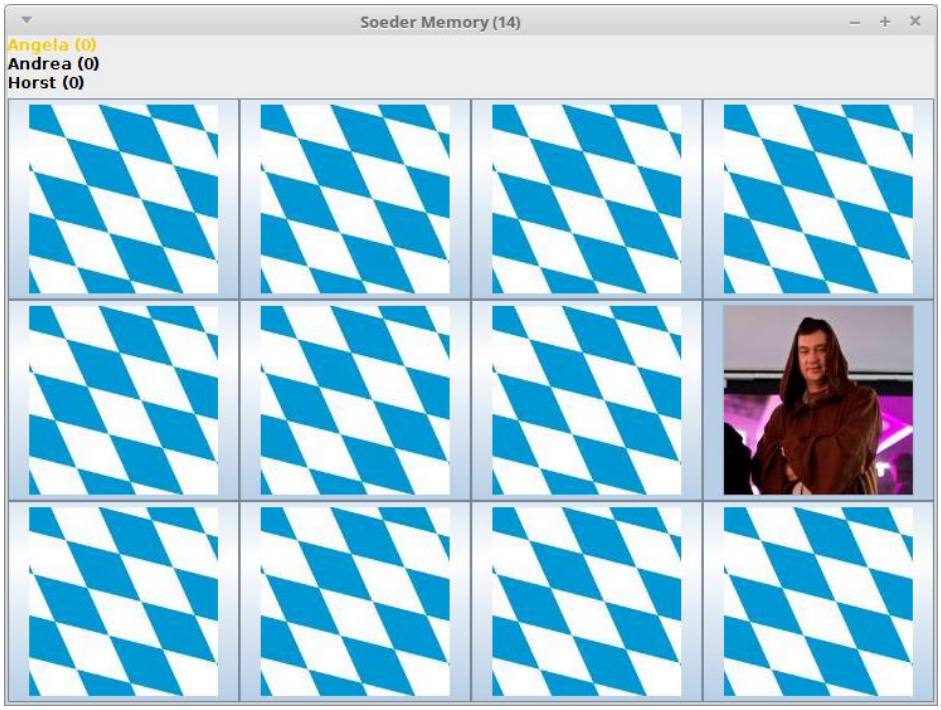
SOEDER MEMORY



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.

Anmerkung zu verwendeten Memory-Bildern:

- Bildnachweise siehe *JavaDoc* der bereitgestellten Klasse `MemoryImages`

DHBW Karlsruhe, Vorlesung Programmieren I+II, Probe-Programmentwurf

Juli 2019

Bearbeitungszeit: 120 Minuten

Aufgabe

Schreiben sie – passend zur bayerischen Landtagswahl am Wochenende – eine Java-Anwendung **`SoederMemory`**, welche ein Memory-Spiel mit exzentrischen Kostümierungen des CSU-Spitzenkandidaten Markus Söder realisiert.

Teilaufgabe a) **[10%]**

Schreiben Sie eine Klasse **`Player`** zur Repräsentation eines Spielers. Ein Spieler hat einen Namen (`name`) sowie seine aktuelle Punktzahl (`points`). Wählen Sie jeweils geeignete Datentypen. Darüber hinaus hat ein Spieler einen Zustand (`status`).

Dieser Zustand ist als *komplexer Aufzählungstyp* **`PlayerStatus`** zu definieren, welcher neben dem Zustand selbst auch dessen zugehörige Farbkodierung (`color`) festlegt:

Status	Farbe
`ACTIVE`	`Color.ORANGE`
`WAITING`	`Color.BLACK`
`FINISHED`	`Color.GRAY`

Definieren Sie für die Klasse `Player` auch einen Konstruktor, dessen Struktur durch die bereitgestellte Klasse `SoederMemory` (siehe USB-Stick) bereits vorgegeben ist. Die initiale Punktzahl beträgt 0 Punkte, der initiale Zustand ist `WAITING`.

Darüber hinaus stellt die Klasse `Player` eine Instanz-Methode `void addPoint()` bereit, welche die aktuelle Punktzahl um einen Punkt erhöht.

Teilaufgabe b) **[15%]**

Zur Verwaltung des Spiels definieren Sie nun eine Klasse **`MemoryGame`**. Diese enthält in geeigneter Form

- die Anzahl der Zeilen (`rows`),
- die Anzahl der Spalten (`cols`),
- die Liste der Spieler (`players`),
- die *für dieses Spiel ausgewählten* Bilder (`images`) (siehe auch mitgelieferte interne Klasse `MemoryImage`, die entsprechend importiert werden muss),
- einen Verweis auf den aktuell aktiven Spieler.

Definieren Sie einen Konstruktor – wie durch die bereitgestellte Klasse `SoederMemory` vorgegeben – welcher

- die Liste der Spieler,
- die Liste der *verfügbaren* Bilder für das Memory-Spiel,
- die Anzahl der Zeilen
- und die Anzahl der Spalten für Memory-Karten entgegennimmt.

Prüfen Sie, ob mindestens zwei Spieler übergeben wurden. Falls nicht, ist eine (selbst zu schreibende) **`MemoryException`** mit der Nachricht „`At least two players required`“ zu werfen. Ebenfalls ist zu prüfen, ob die Anzahl der übergebenen verfügbaren Memory-Bilder ausreicht, um ein Spiel mit den übergebenen Zeilen/Spalten durchzuführen. Dabei gilt:

- Jedes Memory-Bild wird natürlich später für zwei Memory-Karten verwendet.
- Sollte die Anzahl der benötigten Memory-Karten (`rows*cols`) ungerade sein, wird eine „Extra-Karte“ hinzugefügt (d.h. die „ungerade“ Extra-Karte ist für die Bestimmung, ob genügend Bilder vorhanden sind, irrelevant).

Sollten die Bilder nicht ausreichen, um alle benötigten Karten-Pärchen zu erzeugen, ist eine `MemoryException` mit der Nachricht „`Too few images available`“ zu werfen.

Sollten die Bilder ausreichen, so ist die benötigte Anzahl an Memory-Bildern *zufällig* aus dem übergebenen Bilderkatalog auszuwählen und in der o.g. Instanz-Variable zu speichern.

`MemoryGame` soll neben ggf. nötigen Getter-/Setter-Methoden noch mindestens folgende Methoden definieren:

- `public Player getCurrentPlayer()`: Liefert den aktuellen Spieler
- `public boolean isBlankRequired()`: Soll `true` liefern, wenn eine Extra-Karte benötigt wird (also die Kartenanzahl, die sich aus Zeilen/Spalten ergibt, ungerade ist), sonst `false`
- `public void nextPlayer()`: Setzt den Verweis für den aktuell aktiven Spieler auf den nächsten Spieler (am Ende der Spielerliste wird wieder von vorne begonnen).
  In den Spieler-Instanzen ist der Zustand entsprechend auf `PlayerStatus.ACTIVE` (Spieler der nun an der Reihe ist) bzw. `PlayerStatus.WAITING` (alle anderen Spieler) zu setzen.

Hinweis: Die eigentlichen Bild-Instanzen werden Ihnen als ImageIcon durch die bereitgestellte Klasse MemoryImages geliefert und sind dort base64-codiert hinterlegt, um Einlese-Problematiken zu vermeiden.

Teilaufgabe c) **[10%]**

Für die Spiel-Oberfläche ist nun die Klasse **`MemoryGameTerm`** zu realisieren. Ihr Konstruktor nimmt lediglich die `MemoryGame`-Instanz entgegen (vgl. Aufruf in `main`-Methode der bereitgestellten Klasse `SoederMemory`). Setzen Sie den Fenstertitel passend zum Spiel auf „`Soeder Memory`“ (vgl. Screenshot).

Im oberen Bereich ist eine Anzeige für die Liste aller Spieler inklusive ihrer aktuellen Punktzahl vorzusehen. Die Farbe der Labels ist entsprechend dem Zustand des jeweiligen Spielers zu setzen (vgl. Screenshot).

Screenshot des linken, oberen Bereichs des Spielfensters

Darunter ist basierend auf der in der `MemoryGame`-Instanz gespeicherten Zeilen- bzw. Spalten-Anzahl das Spielfeld mit den Memory-Karten zu realisieren (vgl. Screenshot/Deckblatt). Nutzen Sie dafür bspw. `JToggleButton`s. Folgende Kriterien sind dabei zu beachten:

- Jedes Memory-Bild wird für zwei Memory-Karten verwendet.
- Die Karten sind zufällig auf die Positionen zu verteilen.
- Eine nicht aufgedeckte Karte soll das von `MemoryImages.getBackside()` gelieferte Icon zeigen.
- Eine aufgedeckte Karte (bspw. selektierter `JToggleButton`) zeigt das entsprechende Memory-Bild an.
- Bei einer ungeraden Anzahl an Karten ist eine zusätzliche, einzelne Extra-Karte mit dem Memory-Bild `MemoryImages.getBlank()` einzufügen.

Hinweis: Hier muss noch kein echtes Spiel realisiert werden, in den Teilaufgaben d), e) und f) wird die Funktionalität von `MemoryGameTerm` noch erweitert.

Teilaufgabe d) **[15%]**

Ertüchtigen Sie Ihre `MemoryGameTerm`-Implementierung nun um die nötigen Features zur Durchführung eines echten Memory-Spiels, d.h.:

- Der erste Spieler beginnt das Spiel.
- Pro Spielrunde muss ein Spieler genau zwei Karten aufdecken.
- Wurden zwei Karten aufgedeckt, gibt es zwei Möglichkeiten:
  - Passen die aufgedeckten Karten zueinander soll
    - der aktuell aktive Spieler einen Punkt bekommen,
    - die beiden Karten deaktiviert werden und
    - der Spieler erneut an die Reihe kommen – sofern das Spiel nicht zu Ende ist (s. unten).
  - Passen die aufgedeckten Karten nicht zueinander soll
    - ein Dialog darüber informieren, dass die Karten nicht gepasst haben (s. Screenshot),
    - danach die beiden Karten wieder „zugedeckt“ werden und
    - der nächste Spieler an der Reihe sein.
- Falls das Spiel zu Ende ist, d.h. alle Kartenpaare erfolgreich aufgedeckt wurden, soll keine neue Runde starten. Stattdessen sollen sämtliche Spieler auf den Zustand `PlayerStatus.FINISHED` gesetzt werden.

Hinweis: Wir gehen von ehrlichen Spielern aus – sie müssen zur Vereinfachung nicht verhindern, dass nach dem Aufdecken der ersten Karte diese wieder zugedeckt werden kann.

In jedem Fall ist im Anschluss an eine Spielrunde die Spieler-Anzeige im oberen Bereich zu aktualisieren (Punkte und Farben passend zu den neuen Zuständen der Spieler-Instanzen).

Info-Dialog, falls die Karten nicht zueinander passen

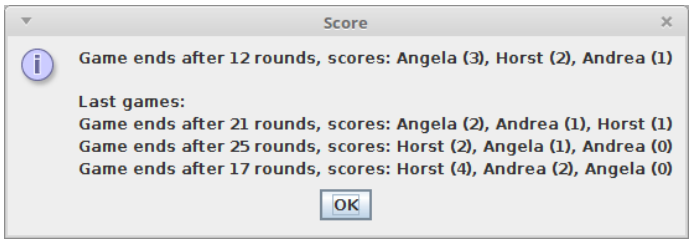
Teilaufgabe e) **[20%]**

Wenn das Spiel zu Ende ist (vgl. *Teilaufgabe d)*) soll zusätzlich ein Info-Dialog mit Informationen zum Ausgang des Spiels angezeigt werden.

Der Dialogtext soll hierfür folgende Informationen enthalten (Beispiel siehe Screenshot weiter unten):

- Wie viele Runden das Spiel gedauert hat (d.h. wie oft versucht wurde, ein korrektes Paar aufzudecken).
- Die Namen und Punktzahl aller Spieler. Dabei sollen die Spieler absteigend nach Punkten sortiert werden.

Falls eine Datei mit dem Namen `memory.txt` vorhanden ist (wird im weiteren Verlauf erstellt), ist deren Inhalt einzulesen und dieser mit einer Zwischenüberschrift „Last Games“ unterhalb des obigen Info-Texts für die aktuelle Spielrunde anzuzeigen:



Info-Dialog inkl. Historie

Schreiben Sie die Information zum Ausgang der aktuellen Spielrunde zusätzlich in die Datei `memory.txt`. Gibt es die Datei noch nicht, soll sie angelegt werden. Existiert sie bereits, ist eine neue Zeile an das Ende einzufügen.

Hinweis: Zeilenumbrüche in Dialogen sind bspw. mit „`\n`“ zu erreichen (plattformübergreifend).

Teilaufgabe f) **[5%]**

Zur Information wie lange das Spiel bereits andauert, soll der Fenstertitel (vgl. *Teilaufgabe c)*) zusätzlich einen Zähler bekommen, der sekündlich hochgezählt wird. Diese Aktion soll nebenläufig durchgeführt werden.

Fenster-Titel mit Sekundenzähler

Die Nebenläufigkeit endet, wenn das Spiel zu Ende ist (und zeigt somit nach Spielende die Anzahl der Sekunden an, die das Spiel insgesamt gedauert hat).

Allgemeine Hinweise

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

Schließen eines Fensters
Beim Schließen des 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.

Hinweis zu JToggleButtons
Unterschiedliche Icons für nicht gedrückte /gedrückte `JToggleButton`s können mit Hilfe der Methoden `setIcon` und `setSelectedIcon` gesetzt werden.