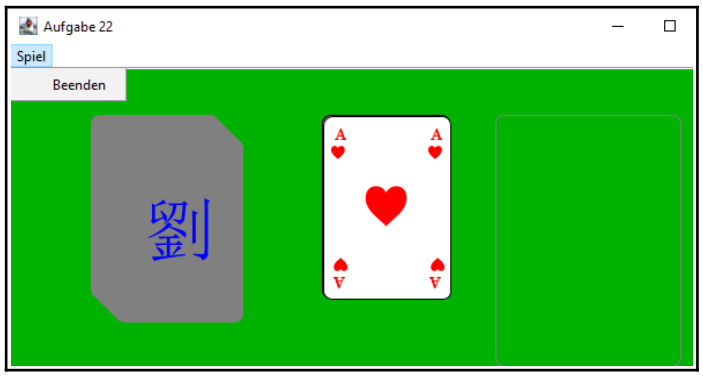
Bitte geben Sie ***NeuesSpielMenuItem.java, Aufgabe22.java*** sowie ***BaumLinearisierungPanel.java*** bis zum 20.12.2020 ab!

Aufgabe 22

Solitär Spiel weiterentwickeln

Ein Menüeintrag zum Starten eines neuen Spiels

Zunächst wollen wir eine Menü-Leiste zu unserem Spiel hinzufügen, dazu verwenden wir die folgenden drei Java Klassen:

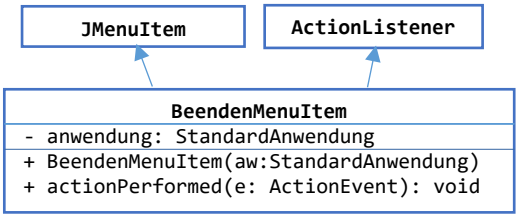


- **JMenuBar:** die Menü-Leiste am oberen Rand eines *JFrame*-Fensters, die ein oder mehrere Drop-Down Menüs enthält.
- **JMenu:** ein Drop-Down Menü mit mehreren Einträgen.
- **JMenuItem:** ein Eintrag in einem Menü.

Für die im Bild gezeigte Menüleiste muss man also drei Objekte erzeugen: ein *JMenuItem* mit der Aufschrift ‚Beenden', eine *JMenu* mit der Aufschrift ‚Spiel', welches das *JMenuItem* enthält sowie ein *JMenuBar* Objekt welche das *JMenu* enthält.

Wir beginnen mit dem Menüeintrag: erzeugen Sie im Paket *komponenten* die von *JMenuItem* abgeleitete öffentliche Klasse ***BeendenMenueItem***, die *ActionListener* implementiert.
Der **Konstruktor** ruft den Konstruktor der Basisklasse mit dem Text ‚Beenden', speichert den Parameter in der Instanzvariable und registriert sich als sein eigener *ActionListener.*

Die Methode ***actionPerformed*** fragt mit Hilfe von `JOptionPane.showConfirmDialog` nach, ob die Anwendung wirklich beendet werden soll (wenn der *return*-Wert gleich `JOptionPane.YES_OPTON` ist, rufen Sie `System.exit(0)`).
Geben Sie *anwendung* als ersten Parameter von *showConfirmDialog* an, damit der Dialog über dem Anwendungsfenster dargestellt wird.



Erstellen Sie eine Kopie des Pakets *aufgabe20* und nennen es ***aufgabe22***, darin nennen Sie die Klasse *Aufgabe20* in ***Aufgabe22*** um.

Dann schreiben Sie in ***Aufgabe22*** die Methode **`private void erzeugeMenuLeiste(){ …`**
In der Sie folgende Schritte programmieren:

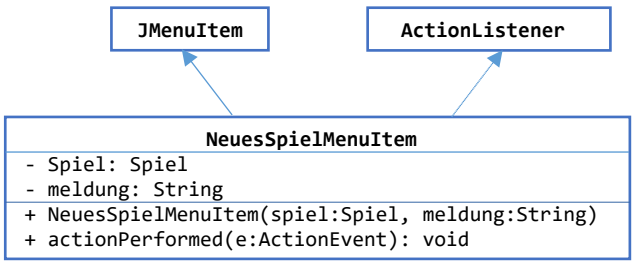
- Speichern Sie ein neues *JMenuBar* Obekt in der lokalen Variable *leiste*,
- speichern Sie ein neues *JMenuObjekt* mit dem Titel „Spiel" in der lokalen Variable *spielMenue*,
- fügen Sie zu *spielMenue* ein neues *BeendenMenuItem* Objekt hinzu (Methode *add*),
- legen Sie das Spiel-Menü auf die Menüleiste (Merthode *add*),
- fügen Sie die Menü-Leiste zur Anwendung hinzu: `this.fenster.setJMenuBar(leiste);`
  (Die Variable *fenster* erbt *Aufgabe22* von *StandardAnwendung*).

Lesen Sie sich die Beschreibungen von *JMenuItem*, *JMenu* und *JMenuBar* durch!

Jetzt schreiben Sie im Paket *komponenten* die nebenstehende Klasse ***NeuesSpielMenuItem***:

Der **Konstruktor** setzt den Titel auf „Neues Spiel", speichert die Parameter in den Instanzvariablen und registriert sich als sein eigener ActionListener

Die Methode ***actionPerformed*** ruft *spiel.gameOver(" "+meldung)*.



In ***erzeugeMenueLeiste*** fügen Sie einen Eintrag vom Typ *NeuesSpielMenuItem* mit dem Text „Spiel beendet" hinzu.

Jetzt sollten beim Auswählen des Menüpunktes „Neues Spiel“ die Meldung“Klick um weiter zu Spielen“ erscheinen und beim nachfolgenden Klick die Kartenstapel wieder in den Anfangszustand kommen.

Um die Meldung deutlicher zu zeigen, setzen Sie im Konstruktor der Klasse ***Spiel*** die Größe von ***meldungFont*** auf `40` und in der Methode ***zeichne*** vor der Darstellung der Game-Over Meldung die Farbe auf rot.

Erste Regeln für das Solitär Spiel

Die erste Regel programmieren wir in die Methode ***nimmFigur*** in der Klasse ***SpielKartenZielStapel.***

- Wenn der Stapel leer ist, wird jede Karte akzeptiert,
- Wenn der Stapel schon eine Karte enthält, werden nur Karten der gleichen Farbe akzeptiert. Dazu führen Sie folgenden Test durch:
  `if(stapel.peek().getFarbe()==ablage.peek().getFarbe()) …`

Jetzt kann man auf einen Stapel nur Bilder gleicher Farbe legen.

Spielkarten mit Bildern personalisieren

Wir personalisieren die Spielkarten, indem wir auf der Rückseite ein eigenes Bild platzieren, und bestimmte Karten (König, Dame, Bube) mit Bildern von Freunden oder Familienmitgliedern versehen.

Zunächst erzeigen Sie im ***src*-Ordner** Ihres Projekts das Verzeichnis ***bilder*** auf der gleichen Ebene wie unser *sound*-Verzeichnis. In dieses Verzeichnis legen Sie die Bilder, die Sie für Ihr Spielverwenden wollen. Die Spielkarte ist 110*160 Pixel groß, das Bild für die Rückseite sollte also etwas kleiner sein. Die Bilder für König, Bube und Dame sollten nicht größer als 65*75 Pixel sein.

SpielKarte SpielFigur
+ KARO, HERZ, PIK, KREUZ: int - WERTE: String[] - ROTSCHWARZ: Color[] - FARBEN: String[] - GROSS, MITTEL, KLEIN: Font - karte: RoundRectangle2D.Double - farbe, wert: int - zeigeVorderseite: boolean - transform: AffineTransform - bildHinten: ImageIcon - bildVorne: ImageIcon
+ SpielKarte(farbe, wert:int, xC, yC:double, spiel:Spiel) + setZeigeVorderseite(zeige: boolean) void + wenden(): void + getFarbe(): int + getWert(): int + zeichne(g: Grapphics2D): void - zeichneWert(g: Graphics2D): void

Die Klasse ***SpielKarte*** bekommt die beiden rot markierten neuen Variablen. Dabei wird ***bildHinten*** als Klassenvariable definiert, denn sie ist ja für alle Spielkarten gleich, während ***bildVorne*** sich für jede Karte unterscheidet und deshalb eine Instanzvariable ist.

Um ein Bild einzulesen definieren Sie im Konstruktor zunächst die lokale Variable ***url*** vom Typ *java.net.url* und definieren sie wie folgt:

```
URL url = SpielKarte.class.getResource("/bilder/hdm.jpg");
```

Der Umweg über die URL stellt sicher, dass das Bild auch gefunden wird, wenn die Anwendung in eine jar-Datei gepackt ist. Dann prüfen wir, ob das Bild gefunden wurde und lesen es ein:

```
if(url!=null) bildHinten = new ImageIcon(url);
```

Das Gleiche machen wir für *bildVorne*, z.B. für den König, der den Wert 11 hat:

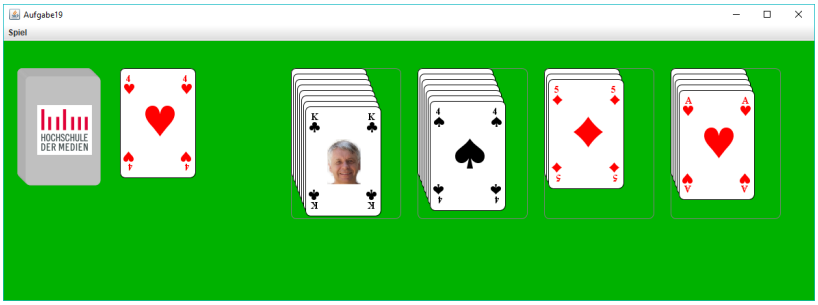
```
url=null;

if(wert==11) { // Koenig
    if (farbe==0) url = this.getClass().getResource("/bilder/schaugg.jpg");
    else if(farbe==1) url = this.getClass().getResource("/bilder/schulz.jpg");
    else if(…  // weitere Bilder für andere Karten
}
if(url!=null) bildVorne = new ImageIcon(url);
```

In der Methode ***zeichne*** gehen Sie beim Zeichnen der Vorderseite z.B. wie folgt vor:

```
  if(this.bildVorne!=null) {
        g.drawImage(bildVorne.getImage(), -30, -42, spiel);
  }else {
        g.setFont(GROSS);
        g.drawString(FARBEN[farbe], -22, 19);
}
```

Unser Spiel sieht jetzt aus wie rechts gezeigt. Es fehlen nur noch die sieben Ablagestapel und die Regeln für das Spiel, die wir in der nächsten Aufgabe implementieren.



Aufgabe 23

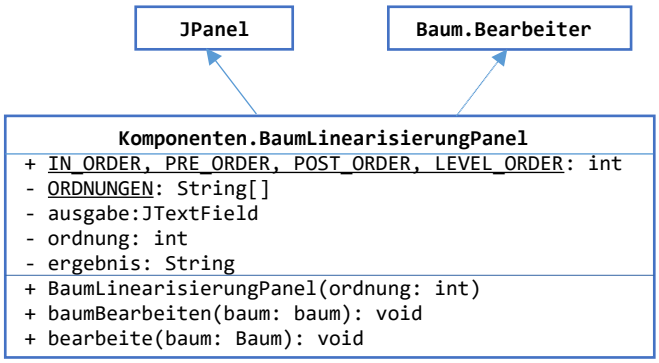
Baum-Linearisierungen

In dieser Aufgaben schreiben wir uns ein Panel zur Darstellung der verschiedenen Linearisierungen eines Binärbaums und wir implementieren den Level-Order Algorithmus.

Wir ergänzen die Klasse Baum durch die Methode **+levelOrder(b: Baum.Bearbeiter): void** , welche die folgenden Schritte durchführt:

- Eine Queue wird als temporäre Variable angelegt
- Die Wurzel des Baums (also *this*) wird in die Queue eingefügt,
- solange die queue nicht leer ist, werden:
  - das oberste Element aus der Queue geholt und bearbeitet,
  - das linke und rechte Kind in den Puffer eingefügt, falls sie nicht null sind.

Jetzt schreiben wir uns die von JPanel abgeleitete Klasse ***BaumlinearisierungPanel***, die eine Linearisierung eines Baumes als String darstellt.



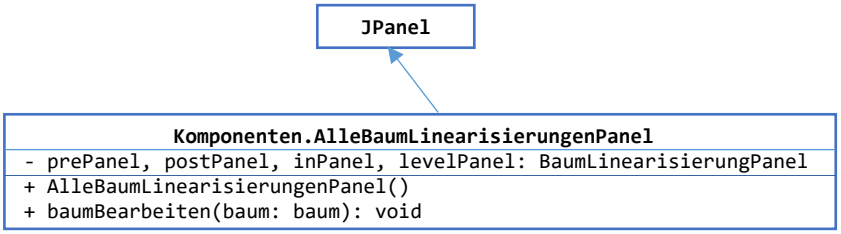
Die Klassenvariablen ***IN_ORDER, PRE_ORDER, POST_ORDER*** und ***LEVEL_ORDER*** werden mit 0, 1, 2 und 3 initialisiert, das Array ***ORDNUNGEN*** mit `{"In","Pre","Post","Level"}`. Das Text-Feld *ausgabe* bekommt die Länge 40.

Der **Konstruktor** wirft eine *IllegalArgumentException*, falls der Parameter nicht zwischen 0 und 3 liegt. Dann speichert er den Parameter in der Instanzvariable *ordnung*, gib sich eine *TitledBorder* mit dem Titel ORDNUNGEN[ordnung] `+"-Order",` legt das Text-Feld auf seine Oberfläche und macht es nicht-editierbar.

Die Methode ***bearbeite*** erweitert den String *ergebnis* um *baum.toString+" "*

`Die Methode` ***baumBearbeiten*** `setzt` *ergebnis* `zu ""`, prüft ob *baum* null ist und ruft dann *preOrder*, *postOrder*, *inOrder* oder *levelOrder* für den Baum, je nachdem welchen Wert die Variable *ordnung* hat.

Um alle Ordnungen auf einmal darstellen zu können, schreiben wir noch die Klasse ***AlleBaumLinearisierungenPanel***, welche alle vier Ordnungen für einen Baum darstellt.



Der **Konstruktor** erzeugt vier *BaumLinearisierungPanel* für die verschiedenen Linearisierungen und ordnet sie in einem Grid-Layout mit zwei Spalten und zwei Reihen an.

De Methode **baumBearbeiten** ruft für jedes der vier Panel die Methode **baumBearbeiten**.

Erzeugen Sie die Klasse ***Aufgabe23*** als Kopie von *Aufgabe21* und fügen eine Instanzvariable vom Typ *AlleBaumLinearisierungenPanel* hinzu, das Sie auf die Position *SOUTH* legen.

Am Ende der Methode ***actionPerformed*** rufen Sie *baumBearbeiten* für dieses Panel.

Jetzt können wir uns wie im Bild gezeigt die verschiedenen Linearisierungen für den Baum ausgeben lassen. Das ist eine gute Möglichkeit zum Überprüfen, ob sie die Linearisierungen auch auf einem Blatt Papier richtig erzeugen können.