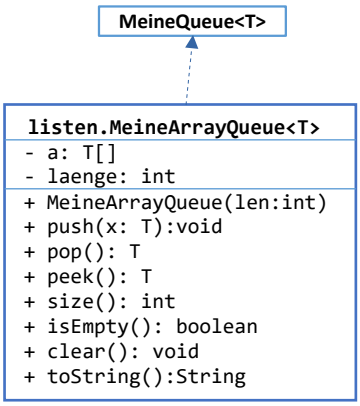
Bitte geben Sie ***MeineListQueue**.java*, ***MeineArrayQueuejava*** und ***SpielKarte.java*** bis zum 29.11.2020 ab!

# Aufgabe 16 Queue

Schreiben Sie im Paket *listen* das Interface ***MeineQueue<T>*** mit den folgenden Methoden:

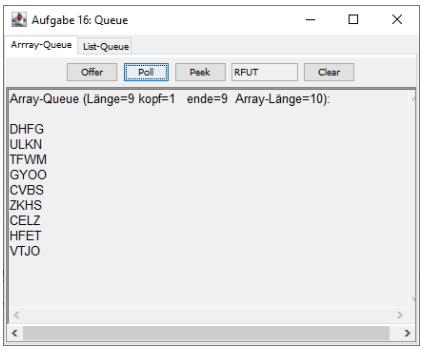
| | |
|---|---|
| `public void offer(T x)` | fügt ein neues Element in die Queue |
| `public T poll()` | holt das erste Element aus der Queue |
| `public T peek()` | liefert das erste Element ohne es aus der Queue zu entfernen |
| `public void clear()` | leert die Queue |
| `public int size()` | liefert die Länge der Queue |
| `public String toString()` | liefert die Queue als Zeichenkette |




Dann schreiben Sie die davon abgeleitete Klasse ***MeineArrayQueue.*** Der **Konstruktor** erzeugt das Array in der angegebenen Länge.

Als zweite Implementierung schreiben Sie die Klasse ***MeineListQueue***, als Queue mit Hilfe einer zirkulär verketteten Liste.

## Anwendung zum Testen der Queue




Testen Sie die Klasse mit der Anwendung ***Aufgabe16***, die genau so aufgebaut ist wie *Aufgabe14*.

Schreiben Sie dazu die Klasse ***QueueTestPanel*** die genau der Klasse *StackTestPanel* entspricht.

Die Anwendung soll wie im Bild links gezeigt beide Queue-Implementierungen testen,

# Aufgabe 17 Spielkarte, Affine Transformation

Als ersten Schritt für ein Kartenspiel schreiben wir die von *Spielfigur* abgeleitete Klasse ***SpielKarte***. Erzeugen Sie dazu das neue Paket **aufgabe17.**

Beim Darstellen der Spielkarte verwenden wir wieder folgenden Trick, der uns das Zeichnen von komplexen Spielfiguren erleichtert:

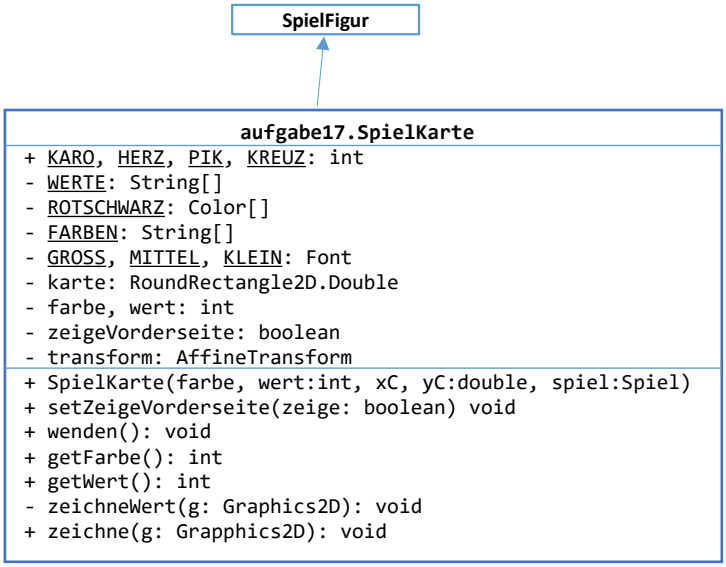
- Vor dem Zeichnen der Figur verschieben wir das Koordinatensystem der *Graphics2D* Zeichenfläche mit einer affinen Transformation in den Mittelpunkt der Spielfigur.
- Dann zeichnen wir die Spielfigur so, als läge sie im Ursprung des Koordinatensystems.
- Am Ende stellen wir das ursprüngliche Koordinatensystem wieder her, damit nachfolgende Zeichenoperationen nicht ebenfalls transformiert werden.

Auf den ersten Blick sieht die im UML Diagramm auf der nächsten Seite gezeigte Klasse kompliziert aus, es handelt sich aber überwiegend um feste Werte, die wie folgt definiert werden:

- ***KARO***, ***HERZ***, ***PIK*** und ***KREUZ*** bekommen die Werte 0,1,2,3.
- ***WERTE*** wird zu {"A", "2", "3", … "10", "B", "D", "K"} gesetzt.
- ***ROTSCHWARZ*** wird mit {Color.RED, Color.RED, Color.BLACK, Color.BLACK} initialisiert.
- ***FARBEN*** wird mit {"\u2666", "\u2665", "\u2660", "\u2663"} initialisiert. Das sind die Unicode Zeichen mit den Symbolen für die Spielkartenfarben. Verwendet man den Font „Osaka“, so sehen die Symbole recht gut aus (siehe z.B. http://www.alanwood.net/unicode/miscellaneous_symbols.html).
- Die Fonts ***GROSS*** und ***MITTEL*** werden mit "Osaka", *Font.PLAIN* und einer Größe von 60 bzw. 20 Punkt definiert.
- Der Font ***KLEIN*** wird mit "Serif", Font.BOLD und einer Größe von 14 Punkt definiert.
- ***karte*** und ***transform*** werden mit dem Standardkonstruktor initialisiert.
- ***zeigeVorderseite*** wird auf *true* gesetzt.

Jetzt kann man KARO, HERZ, PIK und KREUZ als Index in die Arrays *FARBEN* sowie *ROTSCHWARZ* verwenden, um für jede Spielkarte das passende Symbol und die passende Farbe zu bekommen.

Der in der Instanzvariable *wert* gespeicherte Wert einer Spielkarte soll zwischen 0 für die Karte „As“ und 12 für den König liegen und kann als Index in den Array *WERTE* genommen werden.




Im **Konstruktor** übergeben Sie dem Basisklassenkonstruktor die Koordinaten des Mittelpunktes und legen die Größe mit 110*160 Pixeln fest.Die Instanzvariablen ***wert*** und ***farbe*** werden mit den gleichnamigen Parametern initialisiert.

Dann definieren Sie die Geometrie von ***karte*** mit *setFrame*, die linke obere Ecke bekommt die Koordinaten (-55,-80), die Größe ist ebenfalls 110*160. Die Eigenschaften *arcwidth* und *archeight* von *karte* setzen Sie beide auf 15 Pixel, das ist der Radius der abgerundeten Ecken.

Die Methode ***setZeigeVorderseite*** setzt die gleichnamige Instanzvariable.

In ***wenden*** kehren Sie den Wert von *zeigeVorderseite* um.

Die Methoden ***getfarbe*** und ***getWert*** geben die entsprechenden Instanzvariablen zurück.

Die Methode ***zeichneWert*** bleibt zunächst leer.

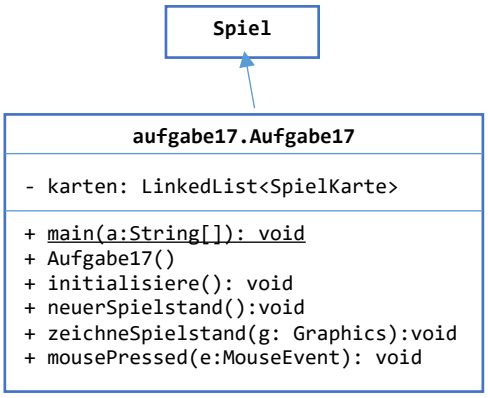
Die überschriebene Methode ***zeichne*** merkt sich die aktuelle Transformation (g.*getTransform*) in einer lokalen Variable, setzt *transform* auf den Kartenmittelpunkt (*transform.setToTranslation(getCenterX(), getCenterY())*) und ruft `g.setTransform(transform)`. Jede Zeichenoperation wird jetzt in die Position der Spielfigur transformiert, deshalb können wir in den folgenden Schritten mit festen Koordinate arbeiten.

Dann fragen Sie ab, ob *zeigeVorderseite true* ist. Wenn ja, führen Sie folgende Schritte aus:

- füllen Sie *karte* in der Farbe Weiß, und zeichnen Sie sie in Schwarz, um den Rand darzustellen,
- setzen Sie die Farbe (setColor) auf `ROTSCHWARZ[farbe]`und den Font (setFont) auf GROSS,
- schreiben Sie den String *`FARBEN[farbe]`* an die Position (-22, 19).
- rufen Sie die methode *zeichneWert* (die im Moment noch nichts macht). Später wird diese Methode das Zeichen für den Wert der Karte über dem Symbol der Farbe in die linke und rechte Ecke übernehmen
- adddieren Sie zu *transform* eine Drehung um180° (*rotate(Math.PI)*), rufen *`g.setTransform(transform)`* und dann nochmals *`zeichneWert(g)`*, damit der Wert der Karte auch am unteren Rand dargestellt wird.

Falls *zeigeVorderseite false* ist, also die **Rückseite** der Karte gezeigt werden soll, färben Sie *karte* in Grau und zeichnen im Font *GROSS* und in Blau den String "\u5289" an die Position(-30,20), das ist ein japanisches Schriftzeichen, das für diesen Zweck ganz gut aussieht. Später können wir hier z.B ein Bild von Ihnen und Ihrer Freundin / Ihrem Freund oder der Familie zeichnen...

WICHTIG: am Ende der Methode *zeichne* setzen Sie wieder die alte Transformation in Kraft (g.setTransform).




Dann schreiben Sie die Klasse ***Aufgabe17***

Der **Konstruktor** färbt den Hintergrund auf `Color.GREEN.darker()`.

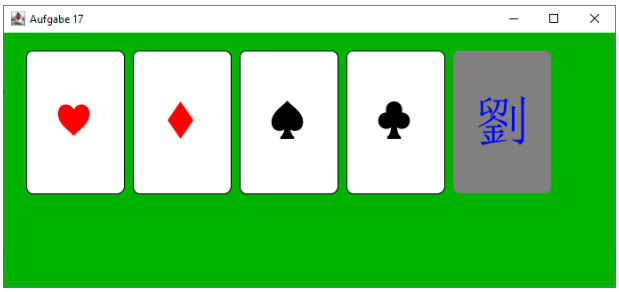
In ***initialisiere*** erzeugen Sie eine neue *LinkedList* und speichern dann darin fünf Spielkarten, die sie nebeneinander anordnen.

Die überschriebene Methode ***neuerSpielstand*** macht NICHTS.

Die überschriebene Methode ***zeichneSpielstand*** ruft für jede Karte *zeichne* (Schleife über die *LinkedList*).

Die überschriebene Methode ***mousePressed*** prüft für jede Karte, ob der Mauspunkt in der jeweiligen Spielkarte liegt (Methode *contains*). Wenn ja, ruft sie für die Spielkarte *wenden*.

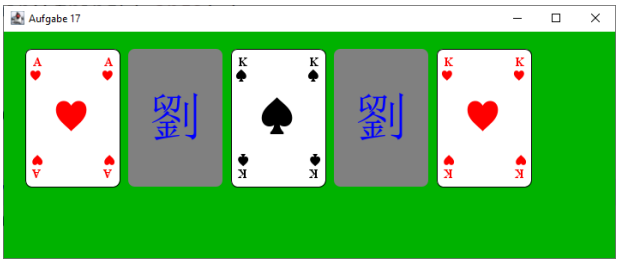
So bekommen Sie die nebenstehenden Spielkarten – noch wird der Wert der Karte nicht angezeigt.




Schließlich verbessern Sie das Aussehen der Vorderseite, indem Sie in die beiden oberen Ecken die Farbe und den Wert der Karte schreiben.

Dazu setzen Sie in **`zeichneWert`** die Farbe auf *ROTSCHWARZ[farbe]* ,den Font auf *KLEIN* und schreiben den String *WERTE[wert]* in an die Positionen (-46,-60) und (37,-60).
Dann wird mit dem Font *MITTEL* der String *FARBEN[farbe]* an die Positionen (-50,-42) und (34,-42) gezeichnet.

Jetzt sehen die Spielkarten so aus wie unten rechts gezeigt.




Sie können die Spielkarten personalisieren, indem Sie auf die Rückseite Ihren Namen schreiben…

Bis Weihnachten schreiben wir uns ein Solitär-Spiel, auf dem Sie z.B. Bilder von Freunden oder Familie platzieren können.

# Vorbereitung zur mündlichen Prüfung

- Sie kennen das Konzept der Rekursion und können kurze rekursive Methoden programmieren.
- Sie kennen den Begriff der Komplexität und die Regeln dazu.
- Sie kenne die Implementierungen von Bubble-Sort und Selection-Sort und können einzelne Methode daraus programmieren.
- Sie verstehen die Algorithmen von Insertion-, Shell und Quicksort sowie Merge-Sort und kennen die Komplexität der verschiedenen Sortieralgorithmen.
- Sie verstehen beide Implementierungen von Stack und Queue (Array und Liste) und sind in der Lage, einzelne Methoden dieser Klasse zu programmieren.
- Sie können Methoden programmieren, die Arrays verarbeiten und/oder zurückgeben.
- Sie kennen die Datenstrukturen doppelt verkettete Liste und Hash-Map sowie deren wichtigsten Eigenschaften und Konzepte.
- Sie kennen die Grundlagen der Objektorientierten Programmierung und die Java Konstrukte Interface, innere Klasse, sowie das Interface Comparable sowie generische Typen.
- Sie kennen das Konzept der ereignisgesteuerten Programmierung mit Listener Klassen (ActionListener, Mouse-Listener, MouseMotionListener) und können es in der Praxis anwenden.