Bitte lösen Sie Aufgaben 17-19 bis zum 24.05.2020 und geben Sie Aufgabe 18 ab!

Aufgabe 17

javax.swing.Timer

In dieser Aufgabe lernen Sie, wie man eine Anwendung über einen **Timer** steuert.

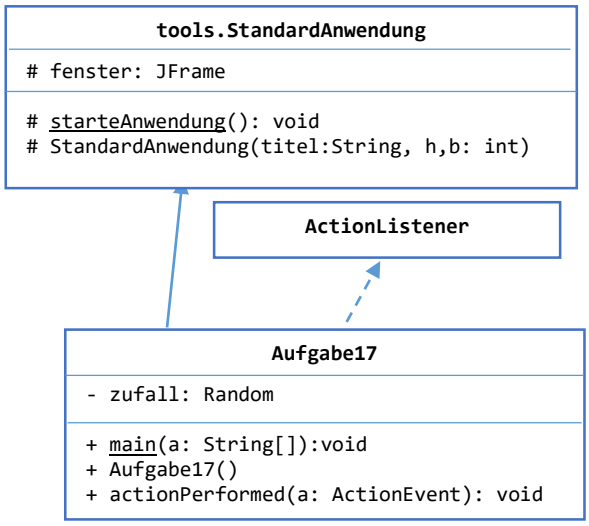
Wir schreiben dazu eine Java Klasse, welche die Hintergrundfarbe einmal in der Sekunde ändert.

Ein *javax.swing.Timer* Objekt ist eine Zeitschaltuhr, die in einem bestimmten Intervall einen oder mehrere *ActionListener* Objekte benachrichtigt, also deren *actionPerformed* Methode ruft. Der Konstruktor hat zwei Parameter:

```
Timer(ms: int, listener: ActionListener)
```

Wobei *ms* das Zeitintervall ist, nach dem jeweils die Methode *actionPerformed* von *listener* gerufen wird.

Der **Konstruktor** von *Aufgabe17* erzeugt den Timer mit einem Intervall von 1000 Millisekunden und *this* als *ActionListener* und speichert ihn in einer lokalen Variable. Nachdem er *zufall* mit dem Standardkonstruktor initialisiert hat, startet er den Timer durch Aufruf der Methode *start()*.

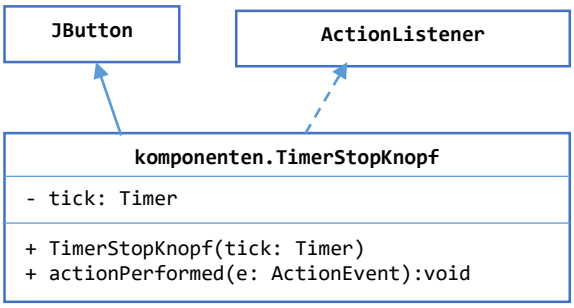


Die Methode *actionPerformed* setzt den Hintergrund auf eine zufällige Farbe.

Start und Stop des Timers

Um den Timer anhalten zu können, schreiben Sie im Paket *komponenten* die Klasse ***TimerStopKnopf*** nach dem nebenstehenden UML Diagramm. Die Aufschrift des Buttons soll „STOP“ sein, bei Klick hält er den Timer an.

Der **Konstruktor** setzt den Titel auf „STOP“, speichert den Parameter in der Instanzvariablen und registriert sich als sein eigener *ActionListener*.



In ***actionPerformed*** wird der Timer durch Aufruf dessen Methode *stop()* gestoppt.

Die weitere Klasse ***TimerStartKnopf*** entspricht weitgehend *TimerStopKnopf*. Objekte vom Typ *TimerStartKnopf* bekommen die Aufschrift „START“ und starten einen Timer.

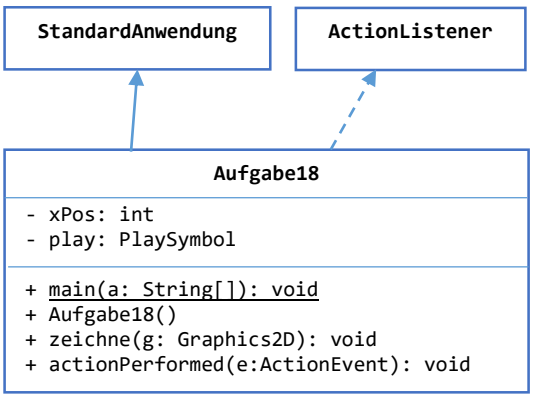
Der Konstruktor von *Aufgabe17* legt jeweils ein Objekt dieser Klassen auf seine Oberfläche.

Aufgabe 18

Eine Timer-gesteuerte Animation

In unserem nächsten Programm lassen wir eine kleine Animation laufen, indem wir einen Timer dazu verwenden, das Bild 25 Mal in der Sekunde neu zu berechnen.

Der **Konstruktor** erzeugt einen *javax.swing.Timer* mit einem Intervall von 40 Millisekunden, dessen *ActionListener* er selbst ist. Dann legt er jeweils einen *TimerStartKnopf* und einen *TimerStopKnopf* auf seine Oberfläche.



actionPerformed ändert xPos:

xPos = (xPos+1)% this.getWidth();

und ruft danach *this.repaint()* um den Display-Manager des Betriebssystems dazu zu veranlassen, das Panel neu darzustellen. Das löst kurz danach den Aufruf unserer Methode *zeichne* aus.

Die Methode ***zeichne*** übernehmen Sie aus der Klasse *Aufgabe 16*.

Achtung: Fügen Sie am Ende der Methode *PlaySymbol* noch die folgende Zeile ein:

```
g.translate(-x,-y);
```

sonst verschiebt es die Buttons mit!

Bei Klick auf *Start* sollte jetzt die Animation loslaufen.

Während wir in Aufgabe 16 die Animation durch die Mausbewegung ausgelöst haben, triggert der Timer jetzt unser *actionPerformed*.

Dort wird der nächste Spielzustand berechnet (neues *xPos*) und anschließend eine neue Darstellung des Spiels angefordert (*repaint*).

Wir haben jetzt das Gerüst einer Anwendung, die mit einer festen Bildwiederholrate von 40 fps (frames per second) ein Spiel steuert – fast schon eine primitive Game-Engine.

Aufgabe 19

Mittlerer Entscheidungsgehalt

Wieviel **Bit** braucht man jeweils **mindestens**, um folgende Information zu speichern:

- Die Ziffern {0,1,2,…,9},
- die Großbuchstaben,
- den Kilometerstand eines Autos, wenn der Tacho bis maximal 1.000.000km zählt,
- den IATA Code für einen Flughafen, der aus 3 Großbuchstaben (ohne Umlaute) besteht, z.B. STR, FRA, LAX, ATL, …
- den IATA Code für eine Flugnummer, die aus folgenden Teilen besteht:
  - Dem Code für die Fluggesellschaft, der aus zwei Zeichen besteht, die entweder eine Ziffer oder ein Großbuchstabe sein können
  - Der Nummer des Fluges, die aus maximal vier Ziffern besteht.
- Wie ändert sich die Mindestanzahl von Bits, wenn die beiden Teile (Fluggesellschaft, Nummer) getrennt gespeichert werden?

Das Wichtigste aus den ersten Wochen nochmal zusammengefasst

Sie kennen die folgenden Begriffe und Konzepte:

- Klasse, Objekt, Methode, Konstruktor, Instanzvariable, Klassenvariable, lokale Variable, Instanzmethode, Klassenmethode.
- Die **elementaren Datentype** *int*, *double* und *boolean* speichern ganze Zahlen, gebrochen Zahlen und logische Werte. Variablen dieser Datentypen speichern direkt einen Wert des jeweiligen Typs. Literale sind Zeichenketten für feste Werte, z.B. 1 1. 4 -3 true false 18.956.
- Klassen sind **Referenzdatentypen**. Variablen vom Typ einer Klasse speichern nicht das Objekt, sondern die **Referenz** auf ein Objekt. Sie werden automatisch mit ***null*** initialisiert, das bedeutet, dass sie zunächst auf kein Objekt zeigen.
- Die Klasse **String** ist in mehrfacher Hinsicht speziell:
  - Der Konstruktoraufruf new String("Hallo") kann durch "Hallo" ersetzt werden,
  - Zwei String-Objekte können mit + aneinandergehängt werden,
  - Beliebige Objekte oder elementare Daten können mit + an einen String gehängt werden.
- **Methoden**: Return-Typ, Sichtbarkeit (public, protected, private), Parameterliste, Signatur, **überladene** Methoden.
- Ein **Konstruktor** wird immer dann gerufen, wenn ein Objekt mit *new* erzeugt wird. Wichtigste Aufgabe des Konstruktors: Initialisierung aller Instanzvariablen. Ein **Standardkonstruktor** hat keine Parameter, ein **allgemeiner Konstruktor** hat Parameter. Falls in einer Klasse kein expliziter Konstruktor programmiert ist, bekommt sie automatsich einen **default Konstruktor**, das ist ein Standardkonstruktor der alle Instanzvariablen mit *0*, *0.0*, *false* bzw. *null* initialisiert.
- Das Schlüsselwort ***this***: ist eine Referenz auf das aktuelle Objekt. Nur in Instanzmethoden kann *this* verwendet werden, in Klassenmethoden gibt es kein *this*.
- Mit ***this(…)*** ruft ein Konstruktor einen überladenen Konstruktor.
- **Vererbung**: eine **Basisklasse** vererbt alle Eigenschaften und Methode auf eine **abgeleitete Klasse**. Jede Klasse ist zunächst von *java.lang.Object* abgeleitet, es sei denn, mit *extends* wird explizit ein andere Basisklasse angegeben.
- Das Schlüsselwort ***super*** ist eine Referenz auf das versteckte Objekt der Basisklasse, das in jedem Objekt enthalten ist. Mit ***super(…)*** kann ein Konstruktor einen Konstruktor der Basisklasse rufen, dies muss die erste Anweisung in einem Konstruktor sein.
- Eine **überschriebene Methode** hat die gleiche Signatur wie eine Methode der Basisklasse. Die Annotation *@Override* zeigt an, dass eine Methode überschrieben ist
- Wenn eine Klasse ein **Interface** implementiert, muss sie alle Methode des Interface enthalten.
- **UML** ist eine einfache Konvention, mit der die Vererbungshierarchie von Klassen dargestellt wird. Sie können ein UML Diagramm in eine Java Klasse umsetzen.
- Sie können Klassen wie *FarbKnopf* und *TimerStartKnopf* selbständig programmieren.
- Sie kennen die Begriffe Zeichenvorrat, Alphabet, Zeichenkette und Abstraktion.
- Der **mittlere Entscheidungsgehalt** ist die minimale Anzahl von Binärentscheidungen die notwendig ist, um eineen Zeichenvorrat zu codieren. Er wird in [bit] gemessen und berechnet sich als $H = log_2 |Z|$ wobei |Z| die Mächtigkeit des Zeichenvorrates ist.