Bitte lösen Sie Aufgaben 22-23 bis zum 27.05.2020 **und geben Aufgabe 22 ab!**

Ziele des heutigen Übungstermins

- Sie schreiben eine weitere Basisklasse zur Vereinfachung der Entwicklung von Spielen.
- Sie zeichnen mit der Klasse *Stroke* in unterschiedlichen Strichstärken.
- Sie lernen die Methoden *darker* und *brighter* von *Color* kennen.
- Sie lernen wie man Sound abspielt.
- Sie verwenden die if-Anweisung.

Aufgabe 22

Basisklasse für Spielfiguren

Die Figuren in einem Spiel können vielfältig sein: Tiere, Autos, Raumschiffe …, aber auch Hindernisse sind Spielfiguren, die sich im Gegensatz zu den vorher genannten nicht bewegen.

Alle Spielfiguren haben Gemeinsamkeiten:

- Es gibt eine **Bounding-Box**, das ist ein Rechteck um die Spielfigur, innerhalb dessen ein Mausklicks die Figur trifft. Dazu verwenden wir die Klasse ***java.awt.Rectangle2D.Double***.
- Fast alle Figuren kommunizieren mit dem Spiel, z.B. registrieren sie sich als *MouseListener* oder *MouseMotionListener* des Spiels oder sie reagieren am Rand des Spielfeldes indem sie stoppen oder abprallen.
- Sie haben einen Bewegungsvektor (v_x, v_y) dessen Komponenten die Bewegung von einem Frame zum nächsten definiert.

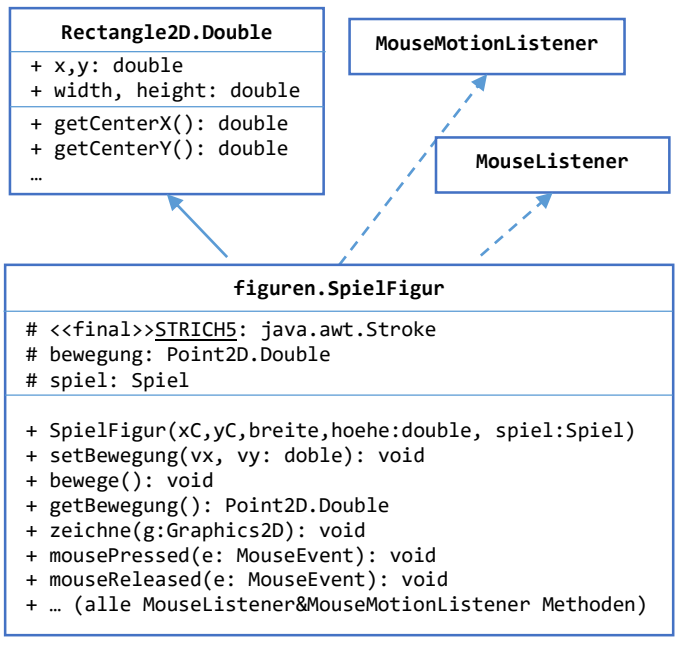
Um diese gemeinsamen Eigenschaften zu bündeln schreiben wir die Klasse ***SpielFigur*** mit folgenden gemeinsamen Eigenschaften von Spielfiguren:

- Das umschließende Rechteck, erben wir von der Basisklasse *Rectangle2D.Double*,
- die aktuelle Geschwindigkeit mit den Komponenten x und y,
- das Spiel, in dem sich die Figur bewegt.

Die Klassenvariable *STRICH5* wird als Strich der Stärke 5 Punkt initialisiert:

```
protected static final STRICH5 = new BasicStroke(5);
```

Die Parameter des **Konstruktors** sind die Mittelpunkts-Koordinaten der Spielfigur (*xC, yC*), ihre Breite und Höhe sowie das Spiel in dem die Figur agiert. Er ruft den Konstruktor der Basisklasse, welcher die Koordinaten der linke oberen Ecke sowie die Breite und Höhe des Rechtecks verlangt.
Da *xC* und *yC* die Koordinaten des Mittelpunktes der Figur sind, geben Sie als Koordinaten der linken obere Ecke `xC-breite/2` und `yC-hoehe/2` an den Basisklassenkonstruktor weiter.
Danach wird *spiel* in der Instanzvariable gespeichert und die Variable *bewegung* mit dem Standardkonstruktor initialisiert.
Abschließend registriert sich die Figur als *Mouse-* und *MouseMotionListener* von *spiel*.



Die Methode ***setBewegung*** speichert ihre Parameter in den Komponenten von *bewegung*.

In ***bewege*** addieren Sie die Komponenten von *bewegung* zu *this.x* und *this.y* hinzu.

Die Methode ***getBewegung*** gibt den Bewegungsvektor zurück.

Die Methode ***zeichne*** setzt die Strichstärke auf 5 Punkt (`setStroke(STRICH5)`) und stellt die Spielfigur als grünes Rechteck dar (`g.draw(this)`). Abgeleitete Klassen sollen diese Methode überschreiben, um individuelle Spielfiguren darzustellen.

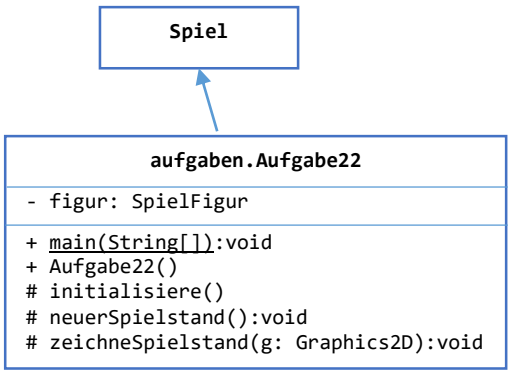
Alle Methoden von *MouseListener* und *MouseMotionListener* (***mousePressed, mouseReleased, mouseEnetered, mouseExited, mouseClicked, mouseMoved*** und ***mouseDragged***) bleiben **leer**. Abgeleitete Klassen müssen somit nur diejenigen Methoden überschreiben, die tatsächlich etwas tun.

Eine Spielfigur kann sich in beliebige Richtungen bewegen und wird als grünes Rechteck dargestellt.

Jetzt schreiben Sie die Klasse ***Aufgabe22*** und geben ihr eine Instanzvariable vom Typ *SpielFigur*.

In ***initialisiere (nicht im Konstruktor!)*** erzeugen Sie die Spielfigur mit der Größe (20,20) an der Position (10,10). Dann setzen Sie die Geschwindigkeit mit *setBewegung* auf (0.9,0.5).

In ***neuerSpielstand*** rufen Sie *figur.bewege*, in ***zeichne*** rufen Sie *figur.zeichne*.



Mit Hilfe der Klassen *Spiel* und *SpielFigur* haben wir ein einfaches aber sehr praktisches Framework für Spiele. Jetzt üben wir noch die Programmierung einiger typischer Verhaltensweisen von Spielfiguren:

MausOverFarbeFigur

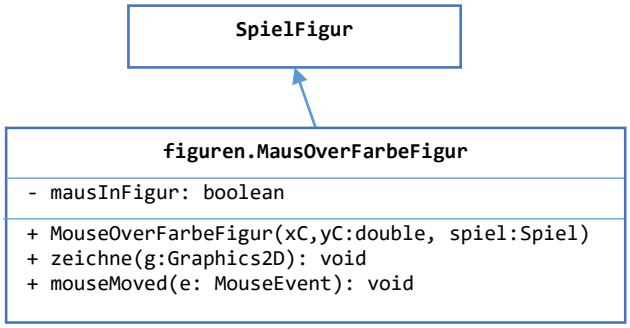
Diese Objekte erzeugen einen Mouse-Over Effekt, indem sie die Farbe ändern.

Der Konstruktor ruft den Basisklassenkonstruktor mit den angegebenen Koordinaten und einer Größe von (40,40) Pixeln. Dann setzt er die Bewegung auf (1,0).

Die Methode ***mouseMoved*** stellt fest, ob sich der Mauszeiger über der Spielfigir befindet und speichert das Resultat in der Instanzvariable:

```
mausInFigur = this.contains(e.getPoint());
```

Die Methode *contains* haben wir von *Rectangle2D.Double* geerbt, *getPoint* liefert die Mausposition.



In ***zeichne*** rufen Sie zunächst `g.setColor(Color.BLUE)`. Falls *mausInFigur* wahr ist, ändern Sie die Farbe auf Gelb. Danach füllen Sie die Figur in der aktuellen Farbe aus.

Fügen Sie zu ***Aufgabe22*** eine *MouseOverFarbeFigur* hinzu. Sie ändert ihre Farbe, wenn Sie den Mauszeiger darüber ziehen.

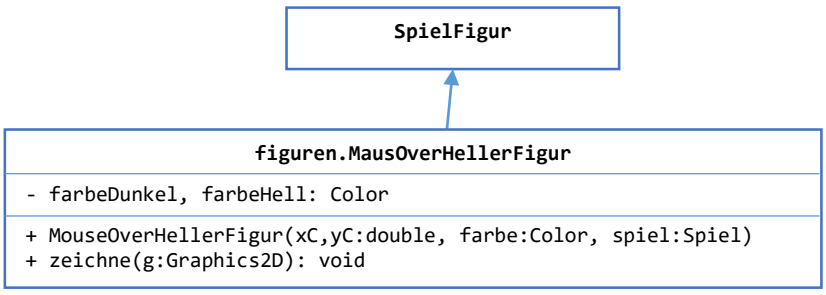
Allerdings hat sie einen Haken: wenn Sie den Mauszeiger vor die Figur setzen und warten, bis diese darunter durchfährt, passiert nichts. Das liegt daran, dass *mouseMoved* nur gerufen wird, wenn der Mauszeiger bewegt wird.

MausOverHellerFigur

Wir programmieren eine andere Methode, welche die Farbe auch dann verändert, wenn der Mauszeiger nicht bewegt wird. Und wir erzeugen einen Mouse-Over Effekt, indem wir die Farbe aufhellen.

Der Konstruktor ruft den Basisklassenkonstruktor mit den angegebenen Koordinaten und einer Größe von (40,40) Pixeln. Dann setzt er die Bewegung auf (1,0). In *farbeDunkel* und *farbeHell* speichert er unterschiedliche Intensitäten der gegebenen Farbe ab:

```
farbeDunkel = farbe.darker();
farbeHell = farbe.brighter().brighter();
```



In ***zeichne*** setzten wir die Farbe von *g* auf *farbeDunkel*. Dann fragen wir die aktuelle Mausposition innerhalb des Spiels ab:

```
Point mausPos =
spiel.getMousePosition();
```

Falls die Maus sich gerade **nicht** über dem Spiel befindet, ist *mausPos null*, deshalb programmieren wir eine kombinierte if-Abfrage:

```
if(mausPos!=null && this.contains(mausPos)){
```

innerhalb dieser Abfrage setzen wir die Farbe, mit der gezeichnet werde soll, auf *farbeHell*. Dann füllen wir die Figur aus.

Drag and Drop

Unsere nächste Figur soll sich mit der Maus bewegen lassen, d.h. bei gedrückter Maustaste soll sie dem Mauszeiger folgen.

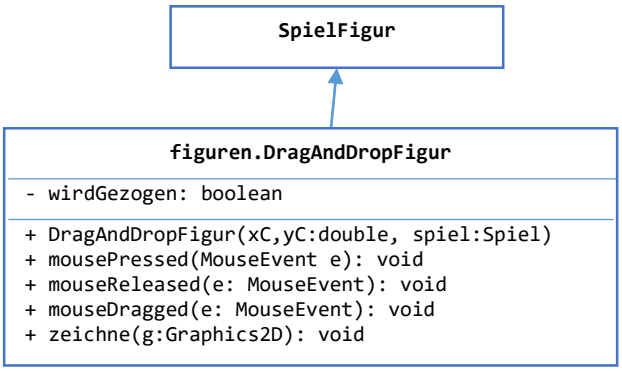
Der **Konstruktor** von ***DragAndDropFigur*** ruft den Basisklassenkonstruktor.

Die Methode ***mousePressed*** prüft, ob der Mauszeiger innerhalb der Figur liegt. Wenn ja, setzten wir *wirdGezogen* auf *true*.

mouseReleased setzt *wirdGezogen* auf *false*.

In ***mouseDragged*** wird geprüft, ob *wirdGezogen true* ist. Wenn ja, wird *this.x* und *this.y* auf die Mausposition gesetzt (*this.x = e.getPoint().x*)

In ***zeichne*** wird eine andere Farbe gewählt, wenn *wirdGezogen true* ist.



Aufgabe 23

Sound abspielen

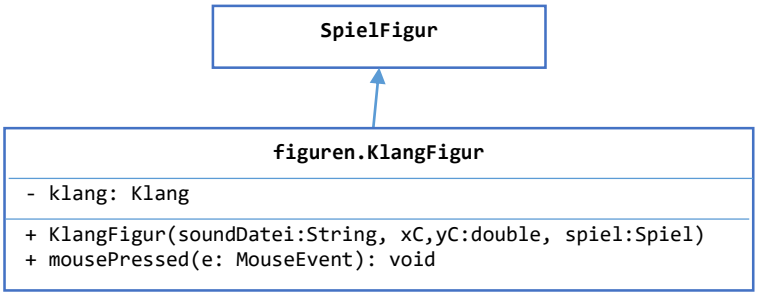
Laden Sie zunächst die Klasse tools.*Klang* aus den Online-Kurs in das Paket *tools*. Am einfachsten ist es, wenn Sie den Text der Klasse markieren (strg-a/strg-c) und dann in Eclipse in das Paket *tools* kopieren (strg-v). Eclipse erzeugt selbständig die Klasse.

Falls Sie den Fehler
„Access restriction: The type 'JFXPanel' is not API (restriction on required library … jfxrt.jar') “
angezeigt bekommen, gehen Sie wie folgt vor:

- Öffnen Sie den Dialog
  „Window→Preferences ->Java ->Compiler->Errors/Warnings ->Deprecated and Restricted API“.
- Wählen Sie bei „Forbidden reference (access rules)“ „Ignore“ und klicken Sie auf „Apply“

Jetzt entwickeln wir eine Spielfigur, die bei Klick einen Klang abspielt. Schreiben Sie dazu die Klasse ***KlangFigur*** anhand des UML Diagramms.

Der **Konstruktor** ruft den Basisklassenkonstruktor und legt seine Größe auf 30*60 Pixel fest. Dann erzeugt er ein *Klang* Objekt aus *soundDatei* und speichert es in *klang*.



Die Methode ***mousePressed*** spielt den Klang durch Aufruf der Methode *play* ab, falls der Klick innerhalb der Figur erfolgt ist (*if (this.contains(e.getPoint()){ …*).

Erzeugen Sie im Ordner ***src*** Ihres Eclipse-Projekts das Unterverzeichnis ***sound*** für die Sound-Dateien. Dort speichern Sie die Dateien *ton1.mp3* .. *ton8.mp3* aus dem Online-Kurs, dabei gehen Sie wie folgt vor:

- Laden Sie die Dateien zunächst wie gewohnt in Ihrem Download-Ordner herunter.
- In Eclipse gehen Sie mit Rechtsklick auf das Verzeichnis *sound* und klicken auf *„Import → File System“*
- Klicken Sie bei *„From-Directory“* auf *„Browse“* und navigieren Sie zu Ihrem Download-Ordner.
- Markieren Sie die 8 *.mp3*-Dateien und klicken auf *„Finish“*.

Verwenden Sie immer diese Methode um Dateien zu importieren, damit sie von Eclipse richtig verknüpft werden.

Bevor wir die Klasse *Aufgabe23* schreiben, **korrigieren wir die Klasse Spiel**. Da das Laden der Sound-Dateien lange dauern kann, brauchen wir eine Abfrage in der Methode *zeichne*, um eine *NullPointerException* bei der ersten Darstellung des Programms zu verhindern. Die rot markierte *if*-Anweisung fehlte in der ersten Version des Übungsblattes mit Aufgabe 21:

```
@Override
public final void zeichne(Graphics2D g) {
   if(timer.isRunning()) zeichneSpielstand(g);
}
```

Wichtig: jetzt darf im **Konstruktor** von *Spiel* **nicht** das rot markierte Wort stehen:

~~Timer~~ timer = new Time(40,this);

sonst ist die Instanzvariable *timer null*!

Klasse Aufgabe23

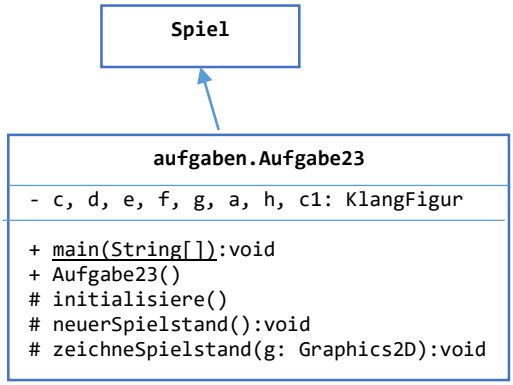
Die Methode ***initialisiere*** erzeugt ein neues *KlangFigur* Objekt:

```
this.c = new KlangFigur("/sound/ton1.mp3",100,100,this);
```

Die Methode ***zeichne*** stellt *ton1* dar.

Die Methode ***neuerSpielstand*** bleibt leer, da das Objekt sich nicht bewegen soll.

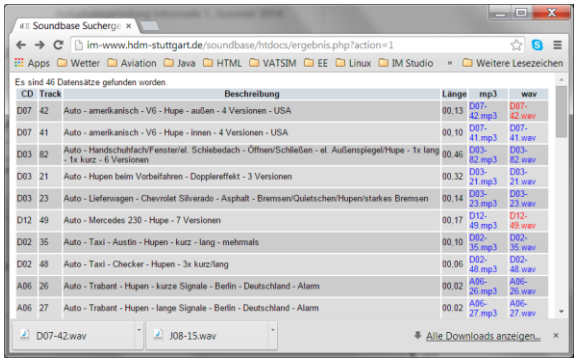
Mit den Dateien *ton2.mp3 .. ton8.mp3* können Sie weitere *KlangFigur* Objekte d, e, f, g, h und c1 erzeugen, die Sie nebeneinanderlegen, so haben Sie ein kleines Musikinstrument programmiert.



Mit diesem Tool kann man eigene Sounds erzeugen:
http://www.bfxr.net/

Allerdings kann es nur WAV-Dateien erzeugen, diese müssen Sie durch ein anderes Program (z.B. *VLC* oder *Audacity*) in MP3-Dateien umwandeln.

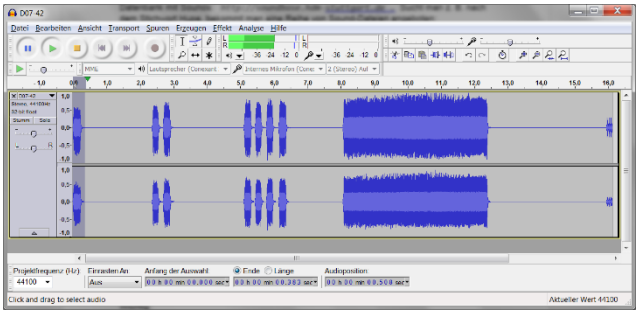
Wichtig: Sounds für Effekte bei Ereignissen sollten kurz sein, 0,2-1,5 Sekunden reichen meist.
Fertige Sound-Dateien bekommen Sie an vielen Stellen im Internet, die Hochschule hat auch eine Datenbank mit Sounds:
https://soundbase.hdm-stuttgart.de.



Sucht man z. B. nach dem Stichwort *Hupe*, bekommt man eine Reihe von Sound-Dateien angeboten:

Um einen Hupton zu erzeugen, können Sie sich entweder die Datei *hupe.mp3* aus dem online Kurs herunterladen, oder Sie gehen wir folgt vor:

Laden sie sich z.B. die Datei D07-42.wav herunter und öffnen Sie in einem Sound-Bearbeitungsprogramm, z. B. Audacity:
http://audacity.sourceforge.net/ .



Schneiden Sie einen Hupton von etwa einer halben Sekunde Länge aus der Datei aus und speichern ihn als *hupe.mp3* im ***sound*** Verzeichnis Ihres Eclipse-Projekts ab.

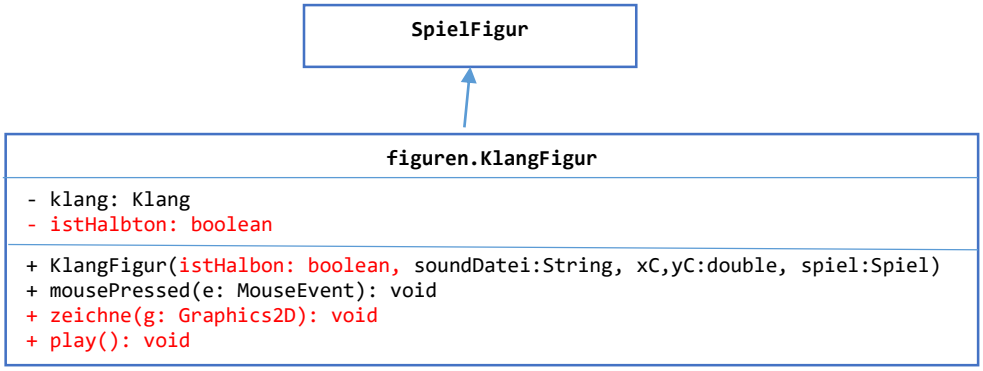
Um die Datei im MP3 Format zu speichern brauchen Sie das Audacity das Plug-In LAME (https://lame.buanzo.org).

Sie können jetzt verschiedene Sounds unterschiedlichen Ereignissen zuordnen. Oder Sie erstellen einen Sound, der als Loop im Hintergrund läuft – diesen starten Sie durch Aufruf der Methode *loop()*.

Im Verzeichnis *sound* im online Kurs finden Sie weitere vorgefertigte Sounds.

Eine Klaviatur

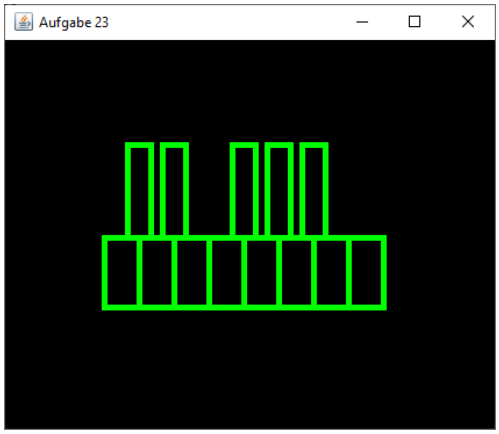
Wir können uns jetzt eine Klaviatur bauen, die auch Halbtöne enthält.
Zunächst erweitern wir die Klasse ***KlangFigur*** wie im UML Diagramm gezeigt:



Der **Konstruktor** speichert den neuen Parameter in der neuen Instanzvariable Dann fragt er ab, ob *istHalbton true* ist und setzt in diesem Fall die Abspielrate des Tons auf $\frac{16}{15}$, dem Verhältnis zwischen Ganz- und Halbton:

```
if(istHalbton) {
        klang.setRate(16.0/15.0);  // ohne die Dezimalpunkte würde 1 berechnet
        this.width = 20;           // die Halbton-Tasten sind schmaler
        this.height = 80;          // aber höher
}
```

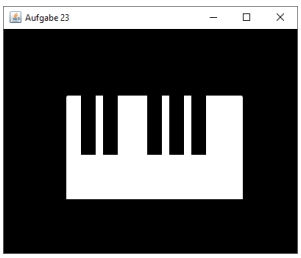
In ***zeichne*** können wir zunächst `super.zeichne(g)` rufen, dann wird die Figur noch als grünes Rechteck dargestellt.

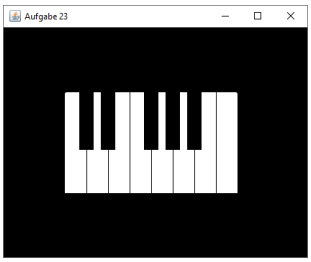


Die Methode ***play*** ruft klang.play()

Jetzt können Sie in **Aufgabe23** die weiteren *KlangFigur* Objekte *cis*, *dis*, *fis*, *gis* und *ais* erzeugen und z.B. so wie gezeigt anordnen.
Wenn Sie dann noch die in *Klangfigur.zeichne* die Tasten weiß bzw. schwarz einfärben und in ein weißes Rechteck dahinter malen, entsteht das unten links gezeigte Bild.

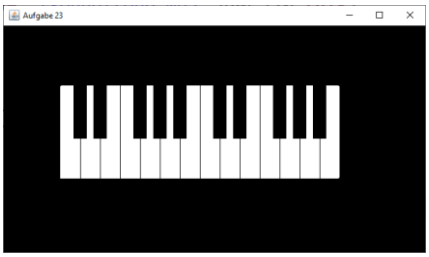
Eine Reihe senkrechter schwarzer Linien im Abstand von 30 Pixeln, am Ende von *Aufgabe23.zeichne* mit *g.drawLine(…)* dargestellt, vervollständigt die Illusion.





Und wer jetzt noch nicht genug hat, kann zwei Oktaven realisieren (**gehört nicht zur Abgabe**):

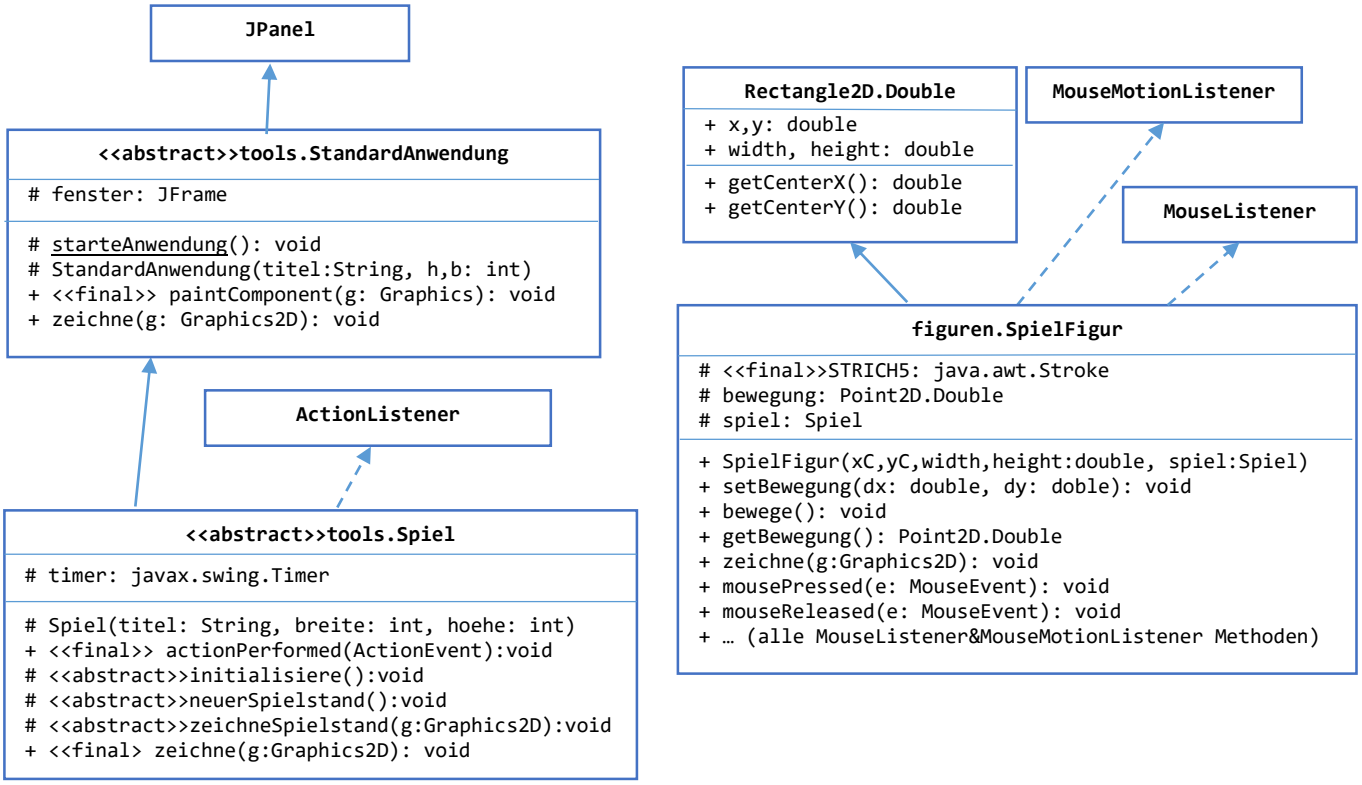
- Fügen Sie dem Konstruktor von *KlangFigur* den weiteren Parameter ***f*** vom Typ *double* hinzu,
- Im Konstruktor rufen Sie *klang.setRate(f)* bzw. beim Halbton auf *klang.setRate(f*16.0/15.0)*.
- Setzen Sie den Parameter f beim Erzeugen der bisherigen Tasten auf 1.
- Erzeugen Sie eine weitere Reihe von Tasten, mit *f=2* – der Ton ist dann eine Oktave höher. Einen Ton – den obersten oder untersten- können Sie weglassen, denn *c1* lag ja schon eine Oktave über *c*.



Und zum Schluss spielen Sie noch ein musikalisches Intro beim Spielstart:

```
private int time=0;
@Override
protected void neuerSpielstand() {
        time++;
        if(time==20) c.play();
        else if(time==40) e.play();
        else if(time==60) g.play();
}
```

Das Spiele-Framework für die Informatik-Übungen




```
//
//  Ein einfaches Spiel
//
public class Beispiel extends Spiel{
  public static void main(String[] a) {
    starteAnwendung();
  }

  private SpielFigur figur;

  public Beispiel() {
    super("Beispiel", 600, 500);
  }
  @Override
  public void initialisiere() {
    figur = new SpielFigur(100, 200, 20, 30, this);
    figur.setBewegung(0.5, 0.2);
  }
  @Override
  public void neuerSpielstand() {
    figur.bewege();
  }
  @Override
  public void zeichneSpielstand(Graphics2D g) {
    figur.zeichne(g);
  }
}
```

```
//
//  Ein einfache Spielfigur
//
public class MausOverFarbeFigur extends SpielFigur {
  private boolean mausInFigur;

  public MausOverFarbeFigur (double xC, double yC, Spiel s) {
        super(xC, yC, 40, 40, s);
        this.setBewegung(1, 0);
  }

  @Override
  public void zeichne(Graphics2D g) {
        g.setColor(Color.BLUE);
        if(mausInFigur) g.setColor(Color.YELLOW);
        g.fill(this);
  }

  @Override
 public void mouseMoved(MouseEvent e) {
        mausInFigur = this.contains(e.getPoint());
  }
}
```