Bitte lösen Sie Aufgaben 29-30 bis zum 05.07.2020 **und geben Aufgabe 30 ab!**

Aufgabe 29

Methoden, Rückgabewert, Rechnen

In dieser Übung programmieren Sie eine Klasse für einfache Vektorrechnungen.

Übernehmen Sie die folgende Klasse ***Vektor2D*** in Ihr Projekt in das Paket ***tools*** und lesen Sie sie aufmerksam durch – am besten tippen Sie sie ab!

Vektor2D Objekte sind zweidimensionale Vektoren, die wir brauchen werden, um die Billardkugeln aneinander abprallen zu lassen. Sie erben die Koordinaten *x* und *y* von *Point2D.Double*.

Die Methode *toString* erlaubt es, ein *Vektor2D* Objekt anstelle eines *Strings* zu verwenden, also z.B.:

```
Vektor2D v = new Vektor2D(2,3);
System.out.println(v);
```

Fügen Sie dieser Klasse eine *main* Methode hinzu, erzeugen Sie die beiden Vektoren *a=(2,3)* und *b=(4,7)*. Addieren Sie den Vektor *b* auf den Vektor *a* und überprüfen Sie das Ergebnis.

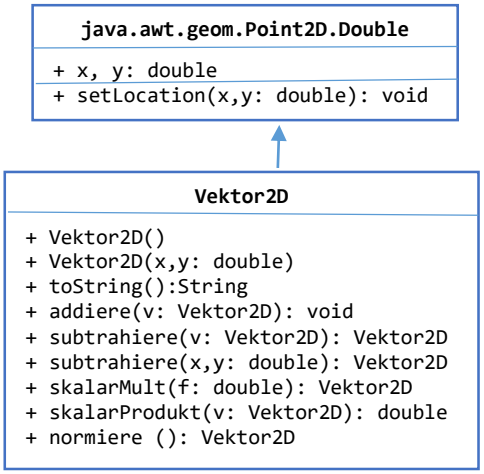
```
package tools;
/**
 * einfache 2D Vektorrechnung
 */
public class Vektor2D extends Point2D.Double{
  /**
   * Erzeugt den Vektor (x,y)
   */
   public Vektor2D(double x, double y){
       super(x,y);
   }

   public String toString(){
       return " ("+x+"/"+y+")";
   }

   /**
    * Addiert den Vektor v zu this hinzu
    * @param v zu addierender Vektor
    */
   public Vektor2D addiere(Vektor2D v){
       this.x = this.x + v.x;
       this.y = this.y + v.y;
       return this;
   }
}
```

Dann fügen Sie folgende Methoden hinzu und testen sie in *main*:

- Einen leeren **Standardkonstruktor**,
- die Methode ***subtrahiere(v:Vektor2D)***, welche den Vektor v vom aktuellen Vektor subtrahiert und *this* zurückgibt,
- die überladene Methode ***subtrahiere (x,y: double)***, welche den Vektor (x,y) vom aktuellen Vektor subtrahiert und *this* zurückgibt,
- die Methode ***skalarMult***, welche den aktuellen Vektor mit *f* multipliziert und *this* zurückgibt,
- die Methode ***skalarProdukt***, welche das Skalarprodukt von *v* und dem aktuellen Vektor berechnet und zurückgibt,
- die Methode ***normiere***, welche den Vektor normalisiert (so skaliert, dass seine Länge 1 ist) und *this* zurückgibt. Die Länge bekommen Sie durch Aufruf von *distance(0,0).* Falls der Vektor die Länge 0 hat, ändern Sie ihn nicht, um eine Division durch 0 zu vermeiden!



Sie können in main z.B. jede Methode prüfen, indem Sie auf bestimmte Werte testen, z.B.:

```
Vektor2D v1=new Vektor2D(46,70), v2=v1.addiere(new Vektor2D(10,20));
if(v1.distance(0,0)!=106) System.out.println("Fehler laenge: "+v1.distance(0,0));
if(v1.normiere().distance(0,0)!=1) System.out.println("Fehler normiere!");
```

An der letzten Zeile sehen Sie, welchen Vorteil es hat, wenn der Return-Typ der meisten Methoden *Vektor2D* ist: wir können die Operationen aneinanderhängen.

Aufgabe 30

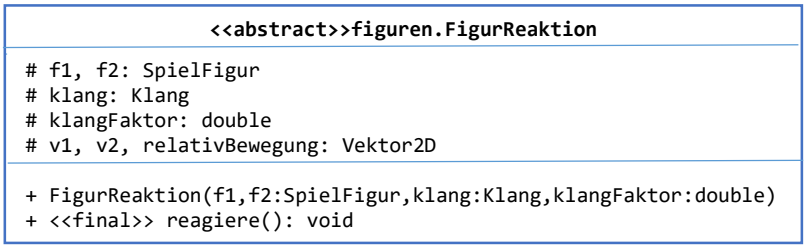
Billard Kugeln Elastischer Stoß

Jetzt erweitern wir das Billard-Spiel aus Aufgabe 27 so, dass die Kugeln sich gegenseitig abstoßen. Dazu benötigen wir die Klasse *Vektor2D* aus der vorherigen Aufgabe!

Erstellen Sie zunächst in Eclipse eine Kopie des Pakets *aufgabe27* und nennen es *aufgabe30*.

Wenn zwei Figuren in einem Computerspiel sich berühren, reagieren sie oft miteinander. Im Falle der Billardkugeln soll diese Reaktion ein elastischer Stoß sein, in anderen Spielen kann die eine Figur die andere zerstören, absorbieren, es kann zum Kampf kommen u.s.w.

Deshalb schreiben wir zunächst die **abstrake** Basislasse ***FigurReaktion***, welche zwei Figuren speichert, die potentiell miteinander reagieren können. Ein Objekt vom Typ *FigurReaktion* prüft in der Methode *reagiere*, ob die beiden Figuren sich berühren. Wenn ja, spielt es einen Sound ab und ruft die abstrakte Methode *reaktion*. Diese Methode wird von den abgeleiteten Klassen überschrieben und führt die eigentliche Reaktion durch, also z.B. den elastischen Stoß, den Kampf u.s.w.



Der **Konstruktor** von *FigurReaktion* speichert die Parameter in den entsprechenden Instanzvariablen und initialisiert ***v1, v2*** sowie ***relativBewegung*** mit dem Standardkonstruktor.

Die Methode ***reagiere*** wird *final* deklariert, damit sie von abgeleiteten Klassen nicht mehr überschrieben werden kann. Sie prüft, ob die beiden Spielfiguren *f1* und *f2* sich berühren oder durchdringen (Methode *intersects*). Wenn ja, speichert sie die Geschwindigkeiten der beiden Figuren in *v1* und *v2* ( )und die Relativbewegung *v1-v2* in der Instanzvariablen *relativBewegung*:

```
v1.setLocation(f1.getBewegung());
v2.setLocation(f2.getBewegung());
relativBewegung.setLocation(v1);
relativBewegung.subtrahiere(v2);
```

Die Variablen *relativBewegung, v1* und *v2* brauchen somit von abgeleiteten Klassen nicht mehr berechnet zu werden. Es ist sehr wahrscheinlich, dass sie diese für ihre Berechnungen benötigen. Wichtig ist, dass in *reagiere* keine neuen Objekte erzeugt werden, denn das würde viel Ressourcen kosten.

Falls *klang* nicht *null* ist, spielt *reagiere* den Klang ab und setzt dabei die Lautstärke auf

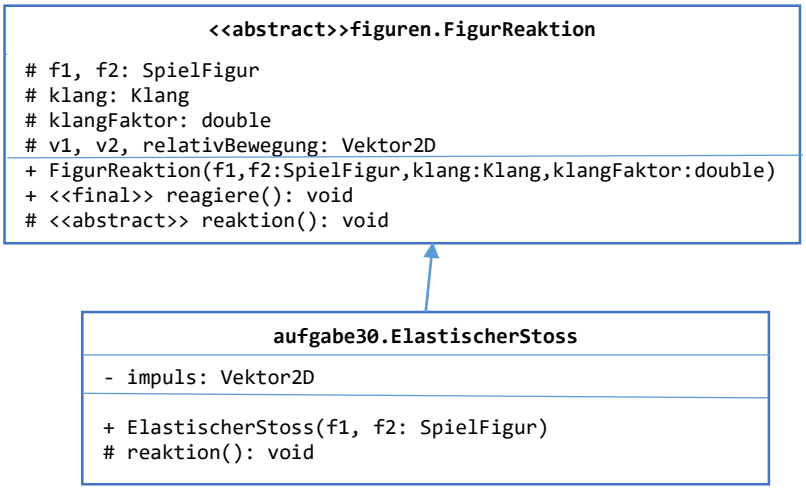
```
|relativBewegung|*klangFaktor
```

(für die Berechnung des Betrages verwenden wir die Methode *distance(0,0)*). Die Lautstärke des abgespielten Sounds richtet sich also danach, mit welcher Geschwindigkeit, die beiden Figuren aufeinanderprallen.

Am Ende von *reagiere* wird die abstrakte Methode *reaktion* gerufen, welche in den abgeleiteten Klassen die eigentliche Reaktion zwischen den beiden Figuren vornimmt.

Klasse *ElastischerStoss*

Für die Berechnung des elastischen Stoßes zwischen zwei Kugeln schreiben wir die von *FigurReaktion* abgeleitete Klasse ***ElastischerStoss***.



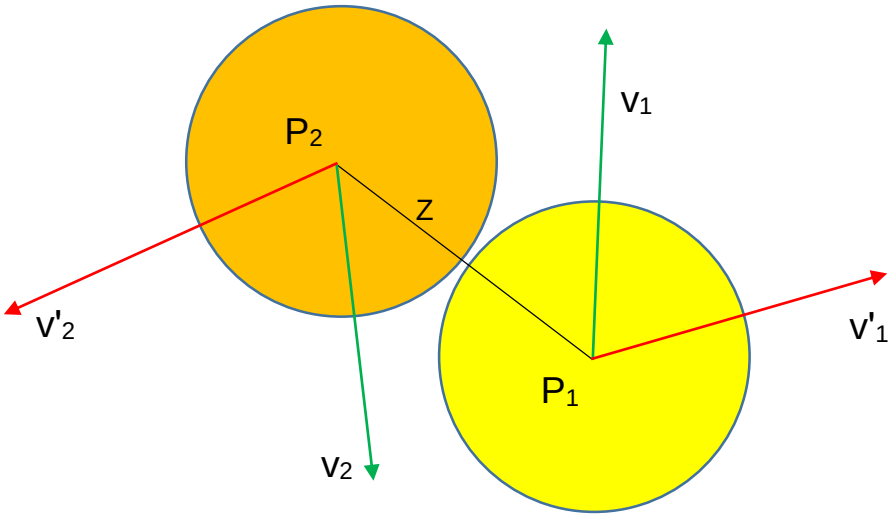
Die **Instanzvariable** *impuls* wird mit dem Standardkonstruktor initialisiert.

Der ***Konstruktor*** ruft lediglich den Basisklassenkonstruktor und übergibt ihm ein neues Objekt vom Typ *Klang* mit der Datei „*billard_kugel_kugel.mp3*“ sowie als Klangfaktor den Wert 0.01.

Bevor wir mit dem Programmieren der Methode ***reaktion*** beginnen, lesen Sie sich die kurze Abhandlung zum elastischen Stoß durch:

Kleiner Ausflug in die Physik: Elastischer Stoß

Im Folgenden sind nur die für uns relevanten Gleichungen zusammengefasst. Sie ergeben sich aus den Erhaltungssätzen von Energie und Impuls. Für eine genauere Herleitung des Elastischen Stoßes schauen Sie bitte in ein Physikbuch.



Zwei Kugeln mit den Massen m_1 und m_2 sowie den Geschwindigkeitsvektoren V_1 und V_2 stoßen zusammen. Gesucht sind die Geschwindigkeitsvektoren V'_1 und V'_2 nach dem Stoß.

Zunächst berechnen wir den normierten Zentralvektor *Z* aus den Ortsvektoren der Mittelpunkte P_1 und P_2 sowie den Stoßvektor *S* als Differenz der Geschwindigkeiten vor dem Aufprall.

Die Richtung des ausgetauschten Impulses ist die Projektion der Relativbewegung auf den Zentralvektor, seine Länge ergibt sich aus den Massen und den Geschwindigkeiten der beiden Kugeln.

Die neuen Geschwindigkeitsvektoren werden aus Summe bzw. Differenz der alten Geschwindigkeiten und dem Impulsvektor -skaliert mit der jeweiligen Masse- berechnet.

V_1, V_2 : Geschwindigkeitsvektoren vor dem Aufprall (wird in *FigurReaktion* berechnet)

V'_1, V'_2 : Geschwindigkeitsvektoren nach dem Aufprall

m_1, m_2 : Massen der Kugeln

Relativbewegung: `R = V1 - V2` (wird in *FigurReaktion* berechnet, Variable *relativBewegung*)

Zentralvektor zwischen den Mittelpunkten: `Z = P1 - P2`

Impulsfaktor $f = \sqrt{|\boldsymbol{R} \cdot \boldsymbol{Z} \cdot \boldsymbol{m_1} \cdot \boldsymbol{m_2}|}$

Impulsvektor: `P = f • Z`

$$V'_1 = V_1 + \frac{1}{m_1} P$$

$$V'_2 = V_2 - \frac{1}{m_2} P$$

Einbau in die Methode *reaktion*

Diese Gleichungen wollen wir jetzt in Java-Code übersetzen und in die Methode *reaktion* einsetzen. Zur Erinnerung: diese Methode wird in *FigurReaktion* nur gerufen, wenn die Kugeln sich berühren oder durchdringen.

Aus Gründen der Effizienz haben wir die Variable *impuls* als **Instanzvariable** definiert und bei der Definition initialisiert (s.o.), damit nicht bei jedem Stoß neue Objekte generiert werden!

Die Berechnung des elastischen Stoßes beginnt mit folgenden Anweisungen:

```
// impuls: normalisierten Vektor vom Mittelpunkt der ersten zur zweiten Kugel
impuls.setLocation(f1.getCenterX(), f1.getCenterY());
impuls.subtrahiere(f2.getCenterX(), f2.getCenterY());
if(impuls.distance(0,0)>42) return;  // berühren sich die Kugeln wirklich?
impuls.normiere();
```

Sie entsprechen der grünen Zeile in den Formeln, zusätzlich wird noch überprüft, ob die Kugeln sich wirklich berühren – wenn nicht, wird der elastische Stoß nicht ausgeführt. Die Variable *impuls* enthält jetzt also den Zentralvektor Z. Fügen Sie diesen Code ein und ergänzen Sie ihn um die folgenden Schritte:

- Berechnen Sie den Impulsfaktor ***double f*** als Quadratwurzel des Betrages des Skalarprodukts von Impuls und Relativbewegung multipliziert mit den Massen (rote Formel). Für den Betrag verwenden Sie die Methode *Math.abs*.
- Multiplizieren Sie den Impuls mit dem Faktor f (Methode *skalarMult*).
- Addieren Sie den Impuls geteilt durch m_1 zur Bewegung der Figur *f1*:
  `f1.setBewegung(v1.x + impuls.x/f1.getMasse(), v1.y + impuls.y/f1.getMasse());`
- Subtrahieren Sie den Impuls geteilt durch m_2 von der Bewegung der Figur *f2*:
  `f2.setBewegung(v2.x - impuls.x/f2.getMasse(), v2.y - impuls.y/f2.getMasse());`

Anwendung in der Klasse *aufgabe30.Billard*

Um auszuprobieren, ob das Ganze funktioniert, fügen Sie in Klasse ***aufgabe30.Billard*** die Instanzvariable ***reaktionen*** vom Typ *LinkedList< ElastischerStoss >* hinzu.

In **initialisiere** erzeugen Sie zunächst das *LinkedList* Objekt und dann fügen Sie so viele Objekte vom Typ *ElastischerStoss* hinzu, dass jede Kugel mit jeder anderen reagieren kann, dazu verwenden wir zwei *for*-Schleifen mit Zähler:

```
for(int i=0;i< kugeln.size()-1;i++){
	for(int j=i+1;j< kugeln.size();j++){
		reaktionen.add(new ElastischerStoss(kugeln.get(i), kugeln.get(j)));
	}
}
```

In ***neuerSpielstand*** rufen Sie nach der Bewegung der Kugeln für jeden elastischen Stoß die Methode *reagiere*.

Jetzt müssten die Kugeln aneinander abprallen, der Sound sollte von der Größe der Relativgeschwindigkeit abhängen.

Mehr Kugeln

Um statt der drei Kugeln eine Anfangsaufstellung von 15 Billardkugeln zu erzeugen, fügen Sie folgenden Code in die Methode **initalisiere von *aufgabe30.Billard*** ein:

```
double x=400, y, y0=400, distanz=41;
Color farbe = Color.WHITE;
for(int reihe=0; reihe<5; reihe++){
    if(reihe>3) farbe=Color.BLUE;
    else if(reihe>2) farbe=Color.MAGENTA;
    else if(reihe>1) farbe=Color.YELLOW;
    else if(reihe>0) farbe=Color.RED;
    y = y0-reihe*distanz/2.0;
    for(int j=0; j<=reihe; j++){
        kugeln.add(new BillardKugel(farbe, x, y, this));
        y = y+distanz;
    }
    x = x+distanz;
}
```

Flüssigere Animation

Die Animation und die Berechnungen gehen besser, wenn die Frame-Rate des Spiels auf 100fps hochgesetzt wird. Rufen Sie dazu in der Methode ***initialisiere*** `timer.setDelay(10)`. Allerdings kann es sein, dass Ihr Rechner jetzt überfordert ist – probieren Sie die maximal mögliche Bildwiederholrate aus.

Hintergrund-Sound

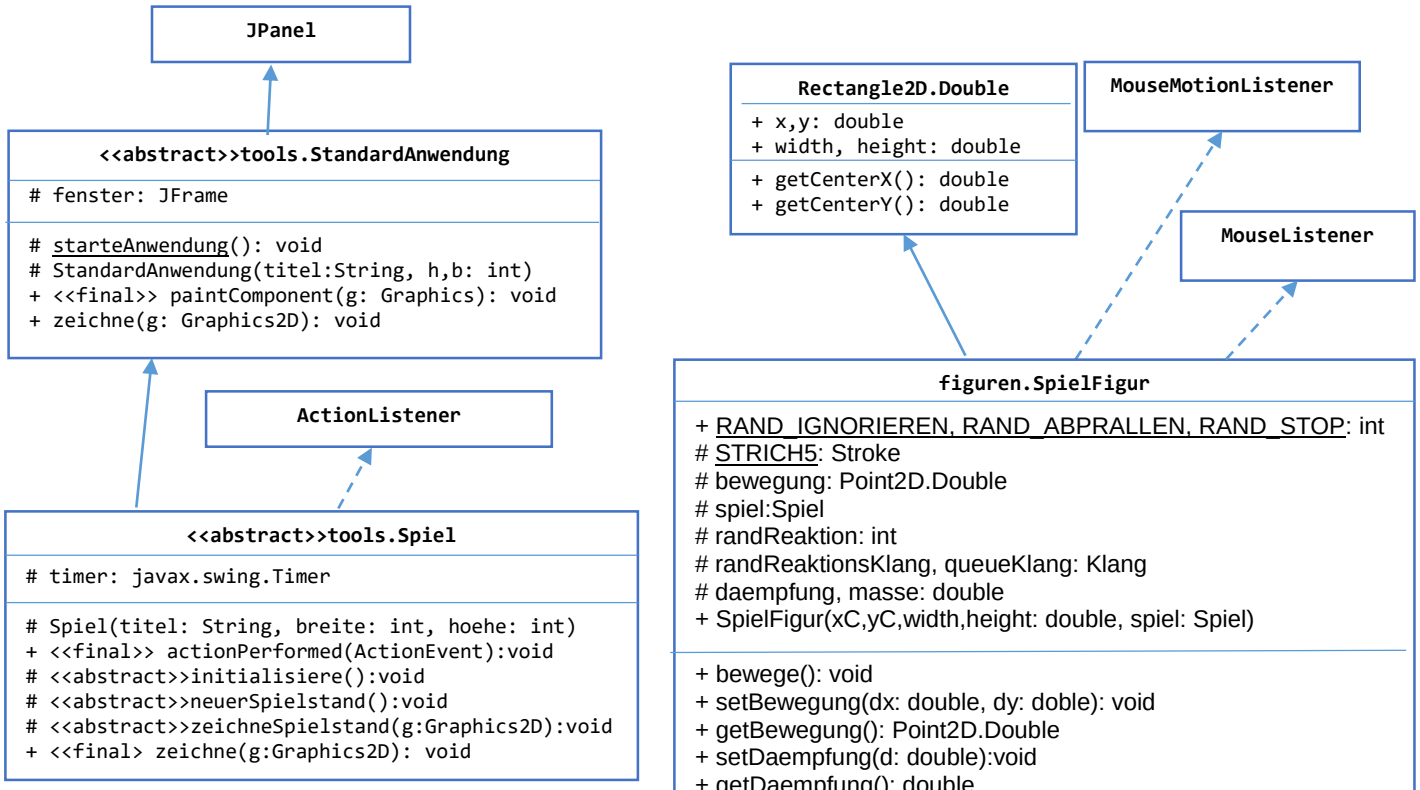
Als möglichen Hintergrund-Sound gibt es im online Kurs die Datei *atmo_bar.mp3*. Erzeugen Sie in der Methode ***initialisiere*** ein *Klang*-Objekt, passen den Basispegel an damit es nicht zu aufdringlich wirkt und rufen *loop*.

Erste Billard-Regel

Als Erweiterung können Sie z.B. in *mousePressed* dafür sorgen, dass nur die weiße Kugel angestoßen werden darf:

```
if(Color.WHITE.equals(farbe)){
    ...
}
```

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());
  }
}
```