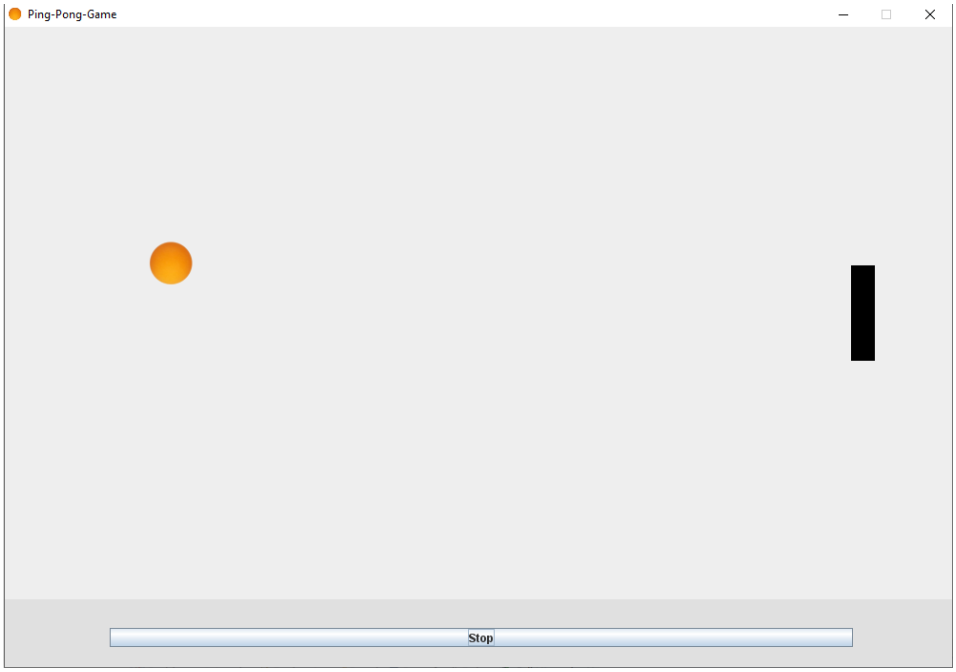
Übung : Ping-Pong Applikation mit Vererbung

In dieser Übung soll eine ***Applikation*** erstellt werden, mit welcher Ping-Pong gespielt werden kann.



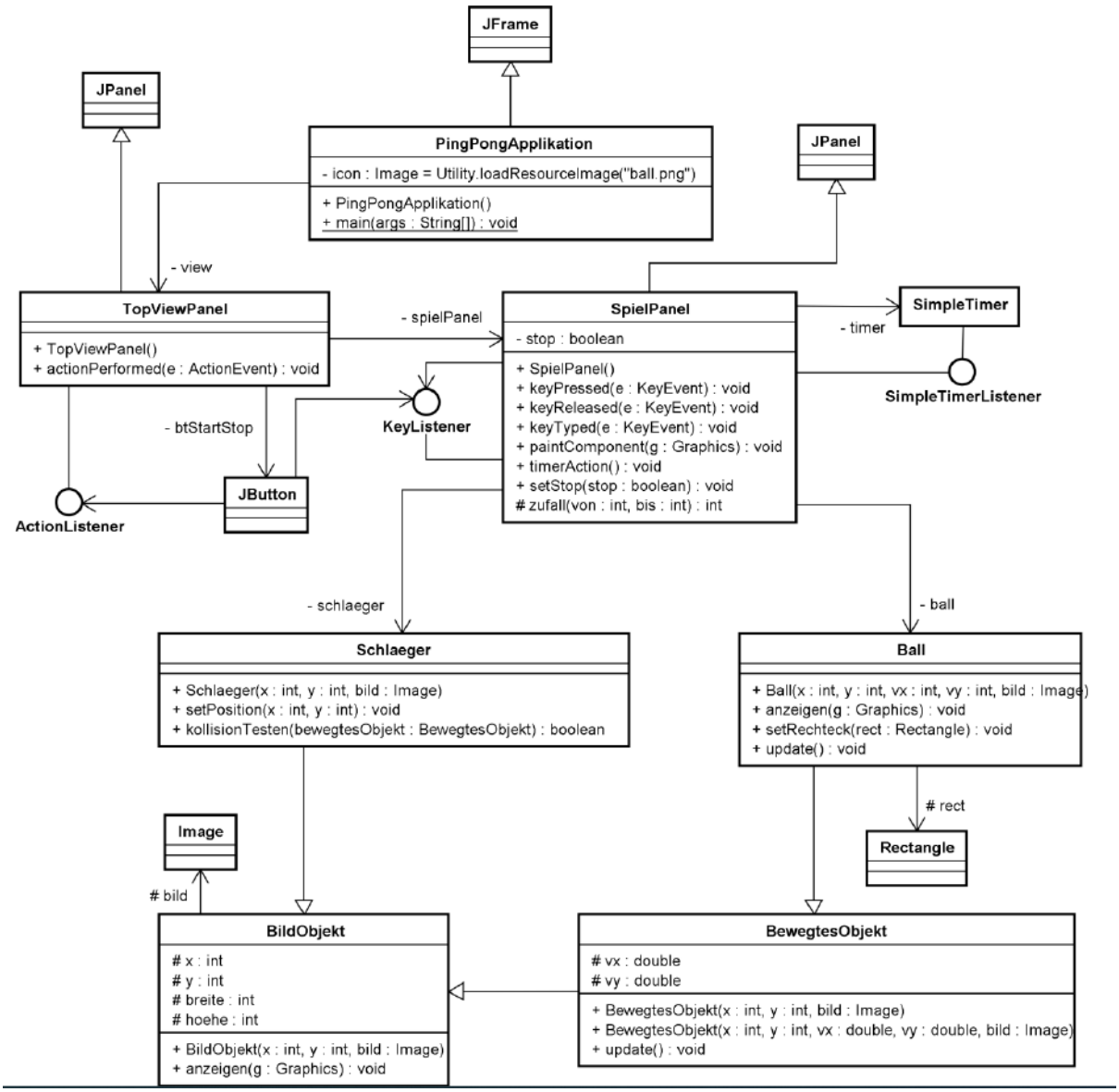
Anforderungen an das Programm:

- Beim Start fliegt ein Ball mit zufälligen **vx** und **vy** Geschwindigkeiten ins Spielfeld und muss vom Schlaeger abgefangen werden.
- Der Schlaeger lässt sich mit den Pfeiltasten (UP/DOWN) mit Hilfe eines **KeyListeners** steuern
- Der Ball reflektiert an allen Seitenlinien des Spielfelds ausser der rechten Seite hinter dem Schläger. Zur Überwachung dient ein Rechteck der Klasse **Rectangle**, welches als Attribut der Klasse **Ball** enthalten ist.
- Wird der Ball nicht abgefangen, so fliegt er aus dem Feld und es wird ein neuer Ball mit zufälliger Geschwindigkeit ins Feld gebracht
- Mit Hilfe eines Start/Stop **JButtons** kann das Spiel pausiert werden.
- Das Grund Design des Spiels entnehmen Sie dem Klassendiagramm. Das eigentliche Spiel ist im **SpielPanel** eingebettet, wo der Ball und der Schlaeger instanziert werden. Sowohl **Ball** als auch **Schlaeger** erweitern die Klasse **BildObjekt**. Die Klasse **Ball** ist eine Unterklasse der Klasse **BewegtesObjekt**.
  Gesteuert wird das Spiel durch den **SimpleTimer**, der für die Dynamik des Balls sorgt

Im Rahmen dieser Übung werden wir nun schrittweise das oben beschriebene Verhalten implementieren.

1) Grundgerüst:

Analysieren Sie das Klassendiagramm (Grundgerüst) und überlegen Sie sich die Aufgaben der einzelnen Methoden und Attribute.



2)Vorlage mit implementiertem Grundgerüst

In der Vorlage finden Sie bereits das implementierte Grundgerüst. Die Methoden sind mit PseudoCode beschrieben. Trotzdem ist es empfehlenswert, das Spiel schrittweise wie beschrieben aufzubauen. Dadurch bekommen Sie einen guten Einblick, wie die einzelnen Objekte hierarchisch miteinander verbunden sind.

3) Ball erzeugen und zunächst statisch auf dem SpielPanel zur Anzeige bringen

Wir kümmern uns zunächst um die statische Anzeige eines Balles. Die Bewegung folgt dann später. Um den Ball anzuzeigen, müssen einige Methoden der Klasse Ball und der Super-Klassen implementiert sein. Implementieren Sie zunächst die Klasse **BildObjekt**. Die Attribute ***breite*** und ***hoehe*** sollen direkt aus dem Image mittels

```
breite = bild.getWidth(null);
hoehe = bild.getHeight(null);
```

ermittelt werden.
Als nächstes benötigen wir die Konstruktoren der Klasse **BewegtesObjekt.** Damit der Ball angezeigt werden kann, werden auch Teile der Klasse Ball benötigt. Implementieren Sie alle Methoden der Klasse Ball, ausser ***update().***
Zur Darstellung erzeugen und starten wir im Konstruktor von **SpielPanel** einen **SimpleTimer** und rufen in der ***timerAction()*** die ***repaint()*** Methode auf. In der entsprechenden ***paintComponent()*** methode soll der Ball durch Aufruf der Klassenmethode dann angezeigt werden. Mit einem erzeugten "Testball" verifizieren wir die bisherige Implementierung

```
ball = new Ball(150, 300, -2, -2, Utility.loadResourceImage("ball.png", 50, 50));
```

Damit der Ball sichtbar wird, muss das **SpielPanel** natürlich noch auf dem **TopViewPanel** platziert werden `add(spielPanel).setBounds(0, 0, 1000, 600);`.

4) Zufallsgenerator für Ballgeschwindigkeit

Um die Geschwindigkeiten **vx** und **vy** zufällig zu bestimmen wird die SpielPanel-Methode ***zufall()*** verwendet, die eine Zufallszahl zwischen ***von*** und ***bis*** zurückgibt.

5) Dynamik erzeugen

Als nächstes soll der Ball sich bewegen und entsprechend der Vorgabe an den Wänden reflektiert werden. Dies geschieht dadurch, dass bei Erreichen der Spielfeldgrenzen (vom Design durch ball.rect vorgegeben) die Richtung der entsprechenden Geschwindigkeitskomponenten gedreht wird. Das Update der aktuellen Position erfolgt in der entsprechenden methode der super-Klasse **BewegtesObjekt.** Die Begrenzung auf das Spielfeld erfolgt in der Methode ***update()*** der Klasse **Ball**.

Damit die Dynamik entsteht, wird der update-Prozess im Rahmen der ***timerAction()*** durchgeführt. Wenn Ball die rechte Spielbegrenzung verlässt, soll ein neuer Ball erzeugt werden.

```
ball = new Ball(150, 350, zufall(-10, -5), zufall(-5, -1),
Utility.loadResourceImage("ball.png", 50, 50));
```

6) Spielunterbrechung

Mit einem **JButton** soll das Spiel pausiert werden könne. Hierzu erzeugen wir gemäss Klassendiagramm im **TopPanelView** einen **btStartStop** und implementieren die ***actionPerformed()*** und ***setStop()*** Methode entsprechend. In der ***timerAction()*** Methode wird dann anhand von stop geprüft, ob update durchgeführt wird oder nicht.

7) Schlaeger implementieren

Die Erzeugung und Darstellung des Schlaegers erfolgen wie beim Ball im SpielPanel.

```
schlaeger = new Schlaeger(900, 300, Utility.loadResourceImage("schlaeger.png", 25,
100));
```

Implementieren Sie die Klasse Schlaeger ausser der Methode ***kollisionTesten()*** und bringen Sie diesen in ***paintComponent()*** zur Anzeige.

Die Schlaegerbewegung erfolgt im Unterschied zum Ball nicht per Timer sondern mittels Keyboard UP/DOWN Tasten und dem **KeyListener.** Hierzu wird die ***keyPressed()*** Methode überschrieben.

```
public void keyPressed(KeyEvent e) {
            if (e.getKeyCode() == KeyEvent.VK_UP) { …
```

Bevor die Methode verlassen wird, soll mit ***repaint()*** neu gezeichnet werden.

Nun sollte der Schläger mit den Pfeiltasten nach oben und unten verschiebbar sein. Sorgen Sie dafür, dass der Schläger nicht über das Spielfeld hinausläuft.

8) Kollisionscheck

Ganz zum Schluss erfolgt nun noch die Prüfung, ob der Schlaeger den Ball getroffen hat. Dies passiert in der Schlaeger Methode ***kollisionTesten()***. Das soll natürlich auch in jedem Timer-Schritt erfolgen. Im Fall der Kollision soll der Ball wieder ins Spielfeld zurückfliegen.

Zur Abfrage auf Kollision ist eventuell die untere Skizze hilfreich. Aus dem Abstand der Referenzpunkte der Objekte und den Höhen und Breiten der Bildobjekte lässt sich die Logik für Kollision ableiten.