# Pong

## 1 Vektoren

### 1.1 Aufgabe

Erstelle eine Datenklasse `V2`. Die Objekte dieser Klasse sind Vektoren im zweidimensionalen Raum mit ganzzahligen Komponenten. Beide Komponenten sind unveränderlich.

```
1 V2(3, 2)
```

```
V2(x=3, y=2)
```

```
1 V2(1, -5)
```

```
V2(x=1, y=-5)
```

### 1.2 Aufgabe

Ergänze die Klasse `V2` mit einer Methode `plus`. Dieser wird ein Vektor übergeben. Es wird die Summe der beiden Vektoren zurückgegeben.

```
1 V2(3, 2).plus(V2(1, -5))
```

```
V2(x=4, y=-3)
```

```
1 V2(5, 4).plus(V2(-4, -5))
```

```
V2(x=1, y=-1)
```

**Hinweis:**

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \end{pmatrix}$$

### 1.3 Aufgabe

Ergänze die Klasse `V2` mit einer Methode `negateX`. Diese gibt den Vektor mit negierter $x$-Komponente zurück.

```
V2(3, 2).negateX()
```

```
V2(x=-3, y=2)
```

```
V2(3, 4).negateX()
```

```
V2(x=-3, y=4)
```

### 1.4 Aufgabe

Ergänze die Klasse `V2` mit einer Methode `negateY`. Diese gibt den Vektor mit negierter $y$-Komponente zurück.

```
V2(3, 2).negateY()
```

```
V2(x=3, y=-2)
```

```
V2(3, 4).negateY()
```

```
V2(x=3, y=-4)
```

## 2 Schläger

### 2.1 Aufgabe

Implementiere eine Datenklasse `Paddle`. Mit dieser werden die beiden Schläger dargestellt. Die Eigenschaften dieser Klasse sind

- `column`: Die Spalte in der sich der Schläger befindet
- `row`: Die Zeile in der die höchste Stelle des Schlägers ist
- `size`: Die Länge des Schlägers in $y$-Richtung

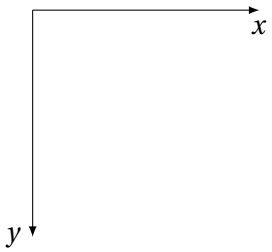
```
Paddle(3, 2, 7)
```

```
Paddle(column=3, row=2, size=7)
```

```
Paddle(1, 5, 3)
```

```
Paddle(column=1, row=5, size=3)
```

In den folgenden Aufgaben werden Terminal-Koordinaten verwendet. Der Unterschied zu dem Koordinatensystem aus dem Matheunterricht ist die Richtung der $y$-Achse.




## 2.2 Aufgabe

Ergänze die Klasse `Paddle` mit einer Methode `cells`. Diese gibt die Liste der Koordinaten der Terminal-Zellen zurück, die der Schläger belegt.

```
Paddle(3, 2, 2).cells()
```

```
[V2(x=3, y=2), V2(x=3, y=3)]
```

```
Paddle(1, 5, 3).cells()
```

```
[V2(x=1, y=5), V2(x=1, y=6), V2(x=1, y=7)]
```

## 2.3 Aufgabe

Ergänze die Klasse `Paddle` mit einer Methode `computeRowOnNextFrame`. Dieser werden die Anzahl der Zeilen des Spielfelds und ein eingegebener Buchstabe übergeben. Mit den Tasten `w` und `s` wird das Paddle nach oben bzw. nach unten bewegt. Die Methode gibt zurück, in welcher Zeile die höchste Stelle des Schlägers nach der Bewegung ist.

```
Paddle(3, 2, 7).computeRowOnNextFrame(15, 'w')
```

```
1
```

Die Zeilen des Spielfelds werden mit 0 beginnend durchnummeriert. Wenn das Paddle schon ganz oben ist, kann es nicht mehr weiter nach oben bewegt werden.

```
Paddle(3, 0, 7).computeRowOnNextFrame(15, 'w')
```

```
0
```

Beim Drücken von `s` wird das Paddle nach unten bewegt.

```
Paddle(1, 2, 4).computeRowOnNextFrame(7, 's')
```

```
3
```

Wenn sich der unterste Block des Schlägers bereits in der letzten Zeile befindet, wird der Schläger mit `s` nicht mehr bewegt.

```
Paddle(1, 2, 4).computeRowOnNextFrame(6, 's')
```

```
2
```

Wenn weder `s` noch `w` gedrückt wird, wird der Schläger nicht bewegt.

```
Paddle(1, 2, 4).computeRowOnNextFrame(10, 'i')
```

```
2
```

## 2.4 Aufgabe

Ergänze die Klasse `Paddle` mit einer Methode `move`. Die Argumente der Methode und die Logik sind wie in der letzten Aufgabe. Sie gibt aber den neuen Schläger zurück.

```
Paddle(3, 2, 7).move(15, 'w')
```

```
Paddle(column=3, row=1, size=7)
```

**Hinweis:** Nutze `computeRowOnNextFrame`!

## 2.5 Aufgabe

Ergänze die Klasse `Paddle` mit einer Methode `getMiddleRow`. Sie gibt zurück in welcher Zeile die Mitte des Schlägers ist.

```
Paddle(0, 2, 6).getMiddleRow()
```

```
5
```

Wenn es keine mittlere Zeile gibt, wird abgerundet.

```
Paddle(0, 2, 7).getMiddleRow()
```

```
5
```

```
Paddle(0, 2, 8).getMiddleRow()
```

```
6
```

# 3 Ball

## 3.1 Aufgabe

Implementiere eine Datenklasse `Ball`. Die Eigenschaften dieser Klasse sind

- `pos`: Die Koordinaten des Balls
- `vel`: Der Geschwindigkeitsvektor des Balls

```
Ball(V2(1, 2), V2(3, 4))
```

```
Ball(pos=V2(x=1, y=2), vel=V2(x=3, y=4))
```

```
Ball(V2(4, 1), V2(3, 2))
```

```
Ball(pos=V2(x=4, y=1), vel=V2(x=3, y=2))
```

Bei jedem Frame wird die Position durch Addition mit dem Geschwindigkeitsvektor aktualisiert.

## 3.2 Aufgabe

Ergänze die Klasse `Ball` mit einer Methode `hitsPaddleNextFrame`. Dieser wird ein `Paddle` übergeben. Sie bestimmt, ob der Ball nach einer Verschiebung mit dem Geschwindigkeitsvektor auf einem Block des `Paddles` ist.

```
Ball(V2(1, 2), V2(3, 4)).hitsPaddleNextFrame(Paddle(4, 2, 5))
```

```
true
```

```
Ball(V2(1, 2), V2(3, 4)).hitsPaddleNextFrame(Paddle(5, 2, 5))
```

```
false
```

**Hinweis:** Nutze `cells`, die Vektoraddition und Listenmethode `contains`!

## 3.3 Aufgabe

Ergänze die Klasse `Ball` mit einer Methode `computeNewVel`. Dieser werden die Zeilenanzahl des Spielfelds und die beiden `Paddle`-s übergeben. Sie gibt den neuen Geschwindigkeitsvektor zurück.

Falls der Ball in der obersten oder in der untersten Zeile des Spielfelds ist, wird die $y$-Koordinate des Geschwindigkeitsvektors negiert.

```
Ball(V2(3, 0), V2(3, -4)).computeNewVel(5, Paddle(0, 2, 5), Paddle(7, 2, 5))
```

```
V2(x=3, y=4)
```

```
Ball(V2(3, 4), V2(3, 4)).computeNewVel(5, Paddle(0, 2, 5), Paddle(7, 2, 5))
```

```
V2(x=3, y=-4)
```

**Hinweis:** Beachte, dass die Zeilen und Spalten mit 0 beginnend durchnummeriert werden.

Wenn der Ball im nächsten Frame einen Schläger berührt, wird die $x$-Koordinate des Geschwindigkeitsvektors negiert.

```
Ball(V2(3, 5), V2(1, 2)).computeNewVel(5, Paddle(0, 2, 5), Paddle(4, 3, 5))
```

```
V2(x=-1, y=2)
```

```
Ball(V2(3, 5), V2(-2, 3)).computeNewVel(5, Paddle(1, 2, 5), Paddle(4, 3, 5))
```

```
V2(x=-2, y=3)
```

Falls keine dieser Bedingungen erfüllt ist, wird der Geschwindigkeitsvektor unverändert zurückgegeben.

```
Ball(V2(3, 5), V2(1, 3)).computeNewVel(5, Paddle(1, 2, 5), Paddle(4, 3, 5))
```

```
V2(x=1, y=3)
```

**Hinweis:** Nutze `negateX`, `negateY` und `hitsPaddleNextFrame`!

## 3.4 Aufgabe

Ergänze die Klasse `Ball` mit einer Methode `update`. Dieser werden die Zeilenanzahl des Spielfelds und die beiden `Paddle`-s übergeben. Sie gibt Ball zurück, nachdem zuerst der Geschwindigkeitsvektor und anschließend die Position neu berechnet wurde.

```
Ball(V2(3, 0), V2(3, -4)).update(5, Paddle(0, 2, 5), Paddle(7, 2, 5))
```

```
Ball(pos=V2(x=6, y=4), vel=V2(x=3, y=4))
```

```
Ball(V2(3, 5), V2(1, 3)).update(5, Paddle(1, 2, 5), Paddle(4, 3, 5))
```

```
Ball(pos=V2(x=4, y=8), vel=V2(x=1, y=3))
```

**Hinweis:** Nutze `computeNewVel` und die Vektoraddition!

# 4 KI

## 4.1 Aufgabe

Ergänze die Klasse `Paddle` mit einer Methode `computeDirAI`. Mit dieser wird bestimmt, wie der Computer seinen Schläger bewegt. Der Methode wird der Ball übergeben. Sie gibt einen Buchstaben zurück, der angibt in welche Richtung der Schläger bewegt werden soll.

Falls die mittlere Postion des Schlägers unter der Position des Balls ist, soll der Schläger nach oben bewegt werden.

```
Paddle(3, 2, 6).computeDirAI(Ball(V2(3, 4), V2(3, -4)))
```

```
w
```

Falls die mittlere Zelle des Schlägers über dem Ball ist, soll der Schläger nach unten bewegt werden.

```
Paddle(3, 2, 6).computeDirAI(Ball(V2(3, 6), V2(3, -4)))
```

```
s
```

Falls die mittlere Postion des Schlägers genau auf der Höhe des Balls ist, soll der Schläger nicht bewegt werden.

```
Paddle(3, 2, 6).computeDirAI(Ball(V2(3, 5), V2(3, -4)))
```

In diesem Fall wird ein Leerzeichen zurückgegeben.

**Hinweis:** Nutze `getMiddlePos`!

## 4.2 Aufgabe

Ergänze die Klasse `Paddle` mit einer Methode `moveAI`. Der Methode wird die Anzahl der Zeilen auf dem Spielfeld und der Ball übergeben. Die Richtung, in die der Schläger bewegt wird, wir mit der letzten Methode bestimmt. Die Bewegung selbst erfolgt mit der Methode `move`.

```
Paddle(3, 2, 6).moveAI(5, Ball(V2(3, 6), V2(3, -4)))
```

```
Paddle(column=3, row=3, size=6)
```

**Hinweis:** Nutze `move` und `computeDirAI`!

# 5 GameState

## 5.1 Aufgabe

Implementiere eine Datenklasse `GameState`. Die Eigenschaften dieser Klasse sind

- `rows`: Die Anzahl der Zeilen auf dem Spielfeld
- `ball`: Der Pong-Ball
- `playerPaddle`: Der Schläger des Spielers
- `aiPaddle`: Der Schläger des Computers

```
GameState(7, Ball(V2(1, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2))
```

## 5.2 Aufgabe

Ergänze die Klasse `GameState` mit einer Methode `playerLost`. Diese gibt zurück, ob der Spieler verloren hat. Dies ist der Fall, wenn der Ball in der Spalte ganz links ist.

```
GameState(7, Ball(V2(0, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).playerLost()
```

```
true
```

```
GameState(7, Ball(V2(1, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).playerLost()
```

```
false
```

## 5.3 Aufgabe

Ergänze die Klasse `GameState` mit einer Methode `aiLost`. Diese gibt zurück, ob der Computer verloren hat. Dies ist der Fall, wenn der Ball und der Schläger des Computers in derselben Spalte sind.

```
GameState(7, Ball(V2(9, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).aiLost()
```

```
true
```

```
GameState(7, Ball(V2(8, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).aiLost()
```

```
false
```

### 5.4 Aufgabe

Ergänze die Klasse `GameState` mit einer Methode `gameOver`. Diese gibt zurück, ob das Spiel beendet wurde.

```
GameState(7, Ball(V2(0, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).gameOver()
```

```
true
```

```
GameState(7, Ball(V2(9, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).gameOver()
```

```
true
```

```
GameState(7, Ball(V2(8, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).gameOver()
```

```
false
```

**Hinweis:** Nutze `playerLost` und `aiLost`!

### 5.5 Aufgabe

Ergänze die Klasse `GameState` mit einer Methode `getWinner`. Diese wird erst aufgerufen, wenn das Spiel beendet wurde. Sie gibt einen String zurück, in dem steht, wer das Spiel gewonnen hat.

```
GameState(7, Ball(V2(0, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).getWinner()
```

```
Der Computer gewinnt!
```

```
GameState(7, Ball(V2(9, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).getWinner()
```

```
Du gewinnst!
```

### 5.6 Aufgabe

Ergänze die Klasse `GameState` mit einer Methode `getPixels`. Sie gibt die Koordinaten der Blöcke zurück, die vom Ball oder einem Schläger verdeckt werden.

```
GameState(7, Ball(V2(0, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).getPixels()
```

```
[V2(x=0, y=3), V2(x=0, y=4), V2(x=9, y=1), V2(x=9, y=2), V2(x=0, y=2)]
```

```
GameState(7, Ball(V2(9, 2), V2(3, 4)), Paddle(0, 3, 2), Paddle(9, 1, 2)).getPixels()
```

```
[V2(x=0, y=3), V2(x=0, y=4), V2(x=9, y=1), V2(x=9, y=2), V2(x=9, y=2)]
```

**Hinweis:** Nutze `cells`!

## 5.7 Aufgabe

Ergänze die Klasse `GameState` mit einer Methode `update`. Dieser wird die Eingabe des Spielers übergeben. Sie gibt einen neuen Spielzustand zurück. Dafür werden zunächst die beiden Schläger und anschließend der Ball aktualisiert.

```
GameState(
            7,
            Ball(V2(3, 2), V2(1, 1)),
            Paddle(0, 3, 3),
            Paddle(4, 2, 3)
        ).update('s') ==
                GameState(
                    7, Ball(V2(2, 3), V2(-1, 1)),
                    Paddle(0, 4, 3), Paddle(4, 1, 3)
                )
```

```
true
```

**Hinweis:** Nutze `move`, `update` und `moveAI`!

## 5.8 Aufgabe

Implementiere einen sekundären Konstruktor für die Klasse `GameState`. Diesem werden die gewünschte Spalten- und Zeilenanzahl übergeben. Der Koordinaten des Balls sind genau die Hälfte der Zeilen- bzw. Spalten. Die Anfangsgeschwindigkeit ist `V2(1, 1)`. Der Schläger des Spielers ist in der Spalte ganz links und der Schläger des Computers in der Spalte ganz rechts. Die Zeile, in der die beiden Schläger sind, ist genau die Hälfte der Zeilen des Spielfelds.

```
GameState(30, 20) ==
        GameState(
            20, Ball(V2(15, 10), V2(1, 1)),
            Paddle(0, 10, 5), Paddle(29, 10, 5)
        )
```

```
true
```

# 6 TUI

Implementiere die folgende Klasse in einer neuen Datei `TUI.kt`.

## 6.1 Aufgabe

Implementiere eine Datenklasse `TUI`. Die Eigenschaften dieser Klasse sind

- `columns`: Die Anzahl der Spalten des Spielfelds
- `rows`: Die Anzahl der Zeilen des Spielfelds

```
TUI(5, 10)
```

```
TUI(columns=5, rows=10)
```

## 6.2 Aufgabe

Ergänze in `build.gradle.kts` die folgende Zeile in den geschweiften Klammern hinter `dependencies`:

```
implementation("com.googlecode.lanterna:lanterna:3.1.3")
```

Kopiere die folgenden Zeilen in `TUI.kt` unter die Zeile, die mit `package` beginnt.

```
import com.googlecode.lanterna.TerminalSize
import com.googlecode.lanterna.input.KeyStroke
import com.googlecode.lanterna.terminal.DefaultTerminalFactory
import com.googlecode.lanterna.terminal.swing.SwingTerminalFontConfiguration
import java.lang.Thread.sleep
```

## 6.3 Aufgabe

Lege im Klassenkörper der Klasse `TUI` eine Eigenschaft `screen` an. Um ein Objekt von dieser Klasse zu erzeugen, musst du mit dem Konstruktor `DefaultTerminalFactory` eine `DefaultTerminalFactory` erzeugen. Dieser Konstruktor hat keine Parameter. Anschließend muss mit den folgenden Methoden der Klasse `DefaultTerminalFactory` definiert werden, welche Eigenschaften die erzeugten Terminals haben. Jede dieser Methoden gibt wieder eine `DefaultTerminalFactory` zurück.

- `setPreferTerminalEmulator`. Dieser Methode wird ein Boolean übergeben. Da wir einen `TerminalEmulator` verwenden wollen, müssen wir `true` übergeben.
- `setInitialTerminalSize`. Dieser Methode wird ein Objekt der Klasse `TerminalSize` übergeben. Dem Konstruktor der Klasse `TerminalSize` müssen wir die gewünschte Spalten und Zeilenanzahl übergeben. Verwende hier die Eigenschaften der Klasse `TUI`. Damit auch der Rand des Spielfelds angezeigt werden kann, benötigen wir 2 Zeilen bzw. Spalten mehr.

- `setTerminalEmulatorFontConfiguration`. Dieser Methode wird ein Objekt der Klasse `SwingTerminalFontConfiguration` übergeben. Dieses kannst du mit `SwingTerminalFontConfiguration.getDefaultOfSize(30)` erzeugen.

Wenn du alle diese Methoden aufgerufen hast, kannst du mit der Methode `createScreen` ein Objekt der Klasse `TerminalScreen` erzeugen.

## 6.4 Aufgabe

Lege einen `init`-Block an, in dem der `TerminalScreen` mit der Methode `startScreen` gestartet wird.

## 6.5 Aufgabe

Lege eine weitere Eigenschaft `textGraphics` vom Typ `TextGraphics` an. Erzeuge den Wert dieser Eigenschaft mit der Methode `newTextGraphics` der Klasse `TerminalScreen`.

## 6.6 Aufgabe

Implementiere eine Methode `close` an. Diese ruft die Methode `close` der Klasse `TerminalScreen` auf.

## 6.7 Aufgabe

Implementiere eine Methode `printString` an. Der Methode wird ein String übergeben. Sie

- ruft die Methode `clear` der Klasse `TerminalScreen` auf.
- gibt mit der Methode `putString` der Klasse `TextGraphics` den übergebenen String in der Mitte des Bildschirms aus. Der Methode `putString` werden eine $x-$ und $y-$Koordinate und der String, der an dieser Stelle ausgegeben werden soll, übergeben.
- ruft die Methode `refresh` der Klasse `TerminalScreen` auf
- ruft `sleep` mit dem Argument 10000 auf

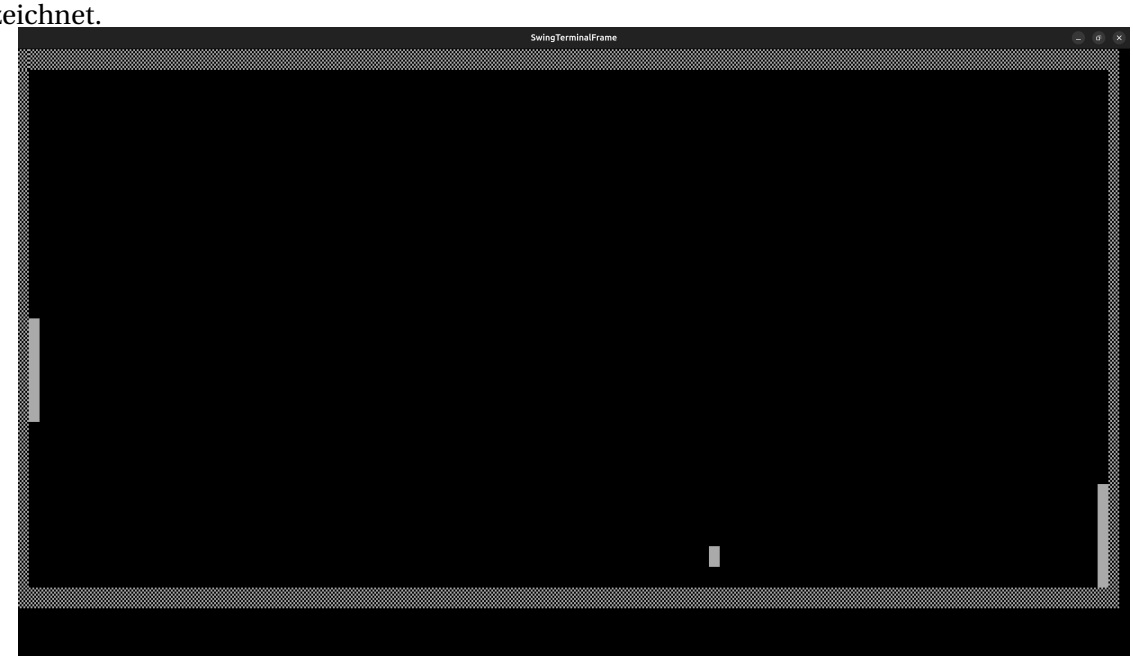
## 6.8 Aufgabe

Implementiere eine Methode `print` an. Dieser wird eine Liste an Vektoren übergeben. Wie die letzte Methode werden

- die Methode `clear` der Klasse `TerminalScreen`
- die Methode `refresh` der Klasse `TerminalScreen`
- die Funktion `sleep` mit dem Argument 10000

aufgerufen.

Nach dem Aufruf von `clear`, wird für jeden übergebenen Vektor ein ▉ mit `putString` gezeichnet. Die Lage des Blocks ist aber um jeweils eine Einheit in $x-$ und $y-$Richtung verschoben. Dies liegt daran, dass die Funktion auch an den Rändern des Spielfelds Blöcke zeichnet.




### 6.9 Aufgabe

Implementiere eine Methode `getPressedKey` an. Diese gibt je nach gedrückter Taste einen Character zurück. Wenn keine Taste gedrückt wurde, wird `' '` zurück gegeben. Nutze hierfür die Methode `pollInput` der Klasse `TerminalScreen`.

## 7 Zusammenfügen des Spiels

### 7.1 Aufgabe

Implementiere eine Funktion `playPong`. Der Funktion werden eine gewünschte Spalten- und Zeilenanzahl übergeben. Beim Aufruf wird eine Runde Pong auf einem Feld mit der entsprechenden Größe gespielt.

**Hinweis:** Nutze die Methoden `print`, `getPressedKey`, `printString` und `close` der Klasse `TUI` und die Methoden `update`, `getPixels`, `gameOver` und `getWinner` der Klasse `GameState`!

### 7.2 Aufgabe

Schütze möglichst viele Attribute und Methoden mit `private` durch Zugriff von außen.