

## Objektorientierte Analyse – Teil IV

Viele Situationen, in denen Objekte miteinander kommunizieren müssen, sind Standard. So z.B. ein Mausklick auf ein Objekt, das Reagieren eines Objekts auf Tastatureingaben oder auch ein im Hintergrund unsichtbarer Taktgeber, der in regelmäßigen Abständen eine Methode eines Objekts aufruft.

Wird zum Beispiel auf ein Objekt (Button) geklickt, so ruft man dazu eine Methode `onClick()` des Objekts auf. Verfügt ein Objekt nicht über diese Methode, so kann man es eben nicht anklicken. Was bei einem Klick geschieht, entscheidest du als Programmierer: z.B. wird ein Datei-Auswahl-Dialog geöffnet, ein anderes Fenster angezeigt, eine Datei gespeichert, ...

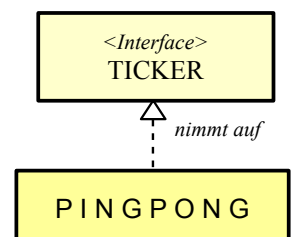
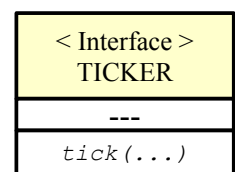
Den Mechanismus hinter z.B. einem Mausklick jedes Mal von Grund auf neu zu implementieren wäre müßig. Oder möchtest du dir bei jedem Mausklick Gedanken machen, woher dein Programm weiß, was eine Maus überhaupt ist und auf welches Objekt geklickt wurde? Diese Arten der Objekt-Kommunikation sind bereits vorgefertigt in Form eines sog. **Interface** (deutsch: Schnittstelle) und eventuell weiterer Klassen, welche dir diese Arbeit abnehmen. Du musst deiner Klasse (z.B. deinem Button) sagen, dass man auf ihre Objekte klicken kann und festlegen, was konkret bei z.B. deinem Mausklick geschehen soll.



- Ein **Interface** gibt eine besondere Fähigkeit an („Ich kann was!“), welche es einer Klasse ermöglicht, von anderen – nicht von dir gefertigten – Klassen angesprochen zu werden.
- Nimmt eine Klasse ein Interface auf, so muss sie die von diesem Interface vorgeschriebenen Methoden aufweisen können. Dazu müssen die vom Interface vorgegebenen Methodenköpfe übernommen und mit Inhalt gefüllt werden.
- Den Vorgang des Schreibens einer so vorgegebenen Methode nennt man **überschreiben**.

### Beispiel:

- Jede Klasse kann das Interface **TICKER** implementieren:  
Für eine Klasse bedeutet das so viel wie  
*„Ich kann auf die Signale eines Taktgebers reagieren!“*
- Wenn die Klasse PINGPONG behauptet:  
*„Meine Objekte können auf die Signale eines Tickers (Taktgeber) reagieren!“*,  
dann muss die Klasse PINGPONG sich dafür „verbürgen“.
- Hierzu muss die Klasse das Interface **TICKER** aufnehmen und über eine Methode `tick()` verfügen.  
Deshalb muss die Methode `tick()` des Interfaces **TICKER** in der Klasse **PINGPONG** **überschrieben** werden.
- Damit die Klasse PINGPONG nicht „lügt“, also behauptet,  
*„Meine Objekte können auf Ticker-Signale reagieren“*  
was dann gar nicht stimmt, „meckert“ solange der Compiler,  
bis es diese Methode auch wirklich gibt.



Damit diese Erklärung sich setzen und ihre Wirkung zeigen kann, muss ein Praxis-Beispiel folgen ...

## Die Programmier-Sprache JAVA – Interface (Teil I)

Die Klasse *PINGPONG* implementiert das Interface *TICKER* und braucht damit die Methode *tick()*. Die Methode *tick()* soll bei jedem Taktgeber-Signal beim BALL-Objekt und den beiden SCHLAEGER-Objekten jeweils die Methode *bewegen()* aufrufen.

 Ball – bewege dich ...



1. Öffne in BlueJ dein Projekt mit den Klassen *PINGPONG*, *BALL*, *SCHLAEGER*, *KREIS* und *RECHTECK*.
2. Importiere aus dem Projekt *Manager\_Funktionen* ([http://engine-alpha.org/html/unterricht/Manager\\_Funktionen.zip](http://engine-alpha.org/html/unterricht/Manager_Funktionen.zip)) die Klasse *MANAGER* sowie die beiden Interfaces *TICKER* und *TASTENREAGIERBAR*.  
Hauptmenü: Bearbeiten / Klasse aus Datei hinzufügen ...

3. Deklariere zunächst in deiner Klasse *PINGPONG* ein *MANAGER*-Objekt:

```
private MANAGER manager;
```

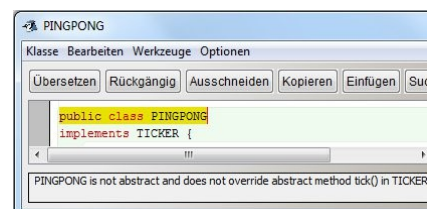
und initialisiere es im Konstruktor:

```
this.manager = new MANAGER();
```

Übersetze und erstelle ein *PINGPONG*-Objekt. Nun hast du eine Spielstands-Anzeige.

4. Implementiere in deiner Klasse *PINGPONG* das Interface *TICKER*:

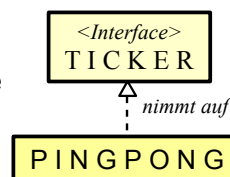
```
/**
 * Spielsteuernde Klasse PingPong
 */
public class PINGPONG
implements TICKER {
    ...
}
```



und versuche, deine Klasse zu übersetzen.

Deine Klasse muss – wie vorher gesagt – über die Methode *tick()* verfügen.

*Diese Methode tut bei jedem Signal immer wieder dasselbe. Was muss sie bei dir alles erledigen?*



„Ich kann nun auf  
Ticker-Signale  
reagieren!“

5. Schreibe nun am Ende deiner Klasse die Methode *tick()*:

```
@Override
/**
 * verpflichtende Methode des Interface TICKER
 */
public void tick() {
    this.links.bewegen();
    this.rechts.bewegen();
    this.ball.bewegen();
}
```

Übersetze nochmal ... der Compiler ist zufrieden ...

6. Das `@Override` kündigt an, dass hier eine Methode überschrieben wird. Das hilft einerseits beim Lesen des JAVA-Codes, andererseits hilft es aber auch, gefährliche Tippfehler zu erkennen und zu vermeiden:

Nenne die Methode nun fälschlicherweise `ticke()` und lösche das `@Override`.

Übersetze!

Die Fehlermeldung des Compilers ist sehr verwirrend, wenn du den Tippfehler nicht bemerkst!

Nimm nun das 'e' wieder weg, verzichte weiter auf das `@Override` aber entferne die Zeile `implements TICKER`.

Übersetze!

Es gibt keine Fehlermeldung obwohl das Vergessen des Interfaces zur Folge hätte, dass kein Ticker da ist, der ticken könnte obwohl du eine Methode `tick()` hast !!!

Schreibe nun `@Override` wieder über die Methode `tick()` (ohne `TICK` zu implementieren).

Übersetze!

Die Fehlermeldung zeigt dir nun, dass es nichts zu überschreiben gibt.  
(da kein Interface implementiert wird)

Füge nun die Zeile `implements TICKER` wieder hinzu.

Noch wird sich nichts bewegen, weil du noch kein Zeitintervall festgelegt und den Ticker auch noch nicht gestartet hast.

7. Melde nun im Konstruktor den Ticker am MANAGER-Objekt an. Teile ihm dabei durch den ersten Übergabe-Parameter mit, dass dein Spiel von jedem Taktsignal informiert werden möchte und durch den zweiten Übergabe-Parameter, dass der Ticker alle 10 Millisekunden ticken soll:

`this.manager.tickerAnmelden(this, 10);`

Referenz auf ein Objekt mit Methode `tick()`: dieses Spiel

ticke alle 10 Millisekunden

```
public PINGPONG() {
    this.ball = new BALL(400, 300, 1, -1, 10, "weiss");
    this.links = new SCHLAEGER(10, 300, 0, 1, 20, 100, "blau");
    this.rechts = new SCHLAEGER(790, 300, 0, -1, 20, 100, "blau");
    this.manager = new MANAGER();
    this.manager.tickerAnmelden(this, 10);
}
```

Übersetze deine Klasse PINGPONG und erzeuge ein Objekt.

Noch hast du keinen Einfluss auf die Schläger und der Ball ignoriert die Schläger und das Aus.

Aber der Ball und die Schläger bewegen sich, der Ball reflektiert oben, die Schläger bleiben oben bzw. unten stehen.

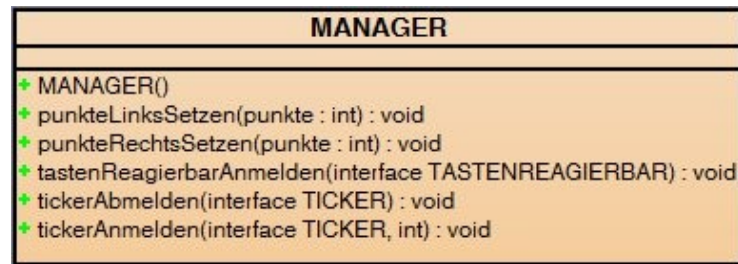
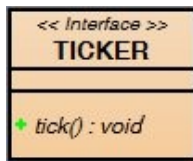
Du hast nun zum ersten Mal komplexe Bausteine der EDU-Variante der Engine Alpha eingesetzt.

Bevor du daran denkst, den letzten Rest des Spiels fertig zu stellen – richtig gelesen, du wirst das alleine machen – solltest du dir unbedingt im Klaren sein, was hier eigentlich im Detail geschehen ist.

Sehen wir uns also die EDU-Version der Engine Alpha einmal genauer an.

**Was ist da gerade alles beim Programmieren passiert?**

## Die EDU-Version der Engine Alpha – automatische Bewegung



### Die Aufgabe des Interface TICKER

Das Interface *TICKER* sagt: „Ich kann auf Ticker-Signale reagieren!“ und fordert dafür die Methode `tick()`.

Immer wenn der Ticker ein Signal gibt, wird diese Methode aufgerufen werden. Du musst beim Schreiben dieser Methode nur angeben, was bei jedem Ticker-Signal geschehen soll.

#### Beispiel:

```

@Override
/**
 * verpflichtende Methode des Interface TICKER
 */
public void tick() {
    this.links.bewegen();
    this.rechts.bewegen();
    this.ball.bewegen();
}
  
```



Wenn eine Klasse ein Interface implementiert, so ist jedes spätere Objekt einerseits vom Daten-Typ dieser Klasse, andererseits aber auch gleichzeitig vom Daten-Typ des Interfaces.

Unser Spiel ist also ein PINPONG-Objekt und gleichzeitig ein TICKER-Objekt.

### Die Aufgabe der Klasse MANAGER

Mit dem Aufruf der Methode `tickerAnmelden(...)` geschehen drei Dinge:

- Der erste Übergabe-Parameter ist ein Ticker-Objekt, dessen Methode `tick()` bei jedem Ticker-Signal aufgerufen werden soll.
- Der zweite Übergabe-Parameter ist die Zeit zwischen jeweils zwei aufeinander folgende Ticker-Signale (in Millisekunden).
- Der Methodenaufruf an sich startet den Ticker.

Möchte man den Ticker wieder anhalten, so braucht man nur die Methode `tickerAbmelden()` aufrufen. Ein erneuter Start des Tickers ist natürlich auch wieder möglich ...

**Beispiel:** `this.manager.tickerAnmelden(this, 10);`



Das Schlüsselwort `this` ist streng genommen eine Referenz auf das eigene Objekt

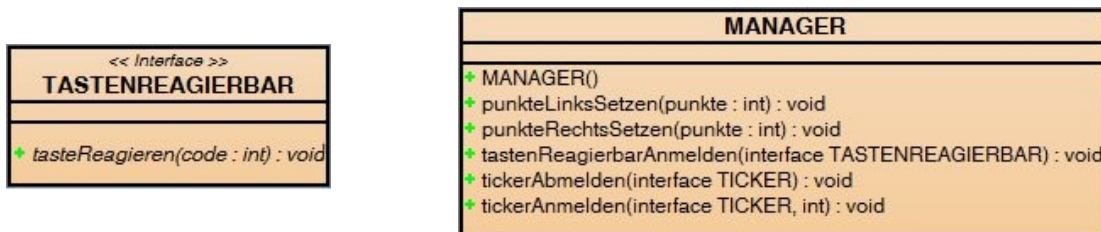
`this.manager.tickerAnmelden(...)` bedeutet quasi

„**Mein** Manager, melde mich (das PingPong-Objekt) bitte beim Manager an, damit er bei mir regelmäßig (alle 10 ms) die Methode `tick()` aufruft.“

Die Klasse `MANAGER` bringt ganz nebenbei auch eine **Spielstands-Anzeige** mit. Wird also irgendwo im Projekt ein `MANAGER`-Objekt erzeugt, so erscheint automatisch eine Spielstands-Anzeige im Fenster. Mit den Methoden `punkteLinksSetzen(...)` und `punkteRechtsSetzen(...)` kann man den Punktestand verändern.

Nun „drehen wir den Spieß um“ und sehen uns erst die Theorie weiter an und setzen sie danach in die Praxis um!

## Die EDU-Version der Engine Alpha – Tastatur-Ereignisse



„Ich kann nun auf  
Tastatur-Ereignisse  
reagieren!“

### Die Aufgabe des Interface TASTENREAGIERBAR

Das Interface `TASTENREAGIERBAR` sagt: „Ich kann auf Tastatur-Ereignisse reagieren!“ und fordert dafür die Methode `tasteReagieren(int code)`.

Der Übergabe-Parameter `code` ist für jede Taste der realen Tastatur ein anderer:

`A=0, B=1, ... Z=25, Pfeil oben=26, rechts=27, unten=28, links=29, ...`

Jede gedrückte Taste übergibt selbständig ihren `code` an die Methode `tasteReagieren(...)`.

Du musst beim Schreiben dieser Methode nur durch bedingte Anweisungen festlegen, was bei bestimmten Tasten geschehen soll.

**Beispiel:**

```

@Override
/**
 * verpflichtende Methode des Interface TASTENREAGIERBAR
 */
public void tasteReagieren(int code) {
    if (code == 0) { // Taste A
        // linker Schläger: vY verringern
    }
}
  
```

## Die Aufgabe der Klasse **MANAGER**

Die Klasse **MANAGER** verfügt über die Methode `tastenReagierbarAnmelden(...)`.

- Der einzige Übergabe-Parameter ist ein **TASTENREAGIERBAR**-Objekt, das über die Methode `tasteReagieren(int code)` verfügt.

**Beispiel:** `this.manager.tastenReagierbarAnmelden(this);`



### zur Sicherheit nochmal:

Wenn eine Klasse ein Interface implementiert, so ist jedes spätere Objekt einerseits vom Daten-Typ dieser Klasse, andererseits aber auch gleichzeitig vom Daten-Typ des Interfaces.

Unser Spiel ist also ein **PINPONG**-Objekt und gleichzeitig ein **TASTENREAGIERBAR**-Objekt.

- Bei jedem Tastendruck der realen Tastatur wird nun von diesem **TASTENREAGIERBAR**-Objekt die Methode `tasteReagieren(code)` aufgerufen. Dabei übergibt jede Taste ihren speziellen `code`.

## Die Programmier-Sprache JAVA – Interface (Teil II)

Das ist nun der versprochene Teil, den du selbst bewältigen sollst. Keine Angst! Du wirst Schritt für Schritt geführt, aber du sollst zeigen, dass du die letzte praktische Aufgabe wirklich begriffen hast. Es wird also etwas Transfer und Eigenleistung von dir verlangt.

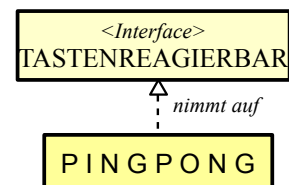
Durch Tastatur-Eingaben soll nun der Wert von `vY` der Schläger verändert werden, so dass du ihre Bewegung beeinflussen kannst



Alle Schläger hören auf mein Kommando ...

1. Schreibe zunächst in deiner Klasse `SCHLAEGER` eine Methode `erhoehevY()`.  
Diese Methode soll den Wert von `vY` um Eins erhöhen. *TIPP: komplexe Wert-Zuweisung*  
*Vergiss den Java-Doc-Kommentar nicht!*  
Teste deine Methode auf korrekte Funktion.
2. Schreibe analog dazu in deiner Klasse `SCHLAEGER` eine Methode `verringerevY()`.  
Diese Methode soll den Wert von `vY` um Eins verringern.  
*Vergiss den Java-Doc-Kommentar nicht!*  
Teste auch diese Methode auf korrekte Funktion.
3. Importiere in dein BlueJ-Projekt nun auch noch das Interface `TASTENREAGIERBAR`.
4. Implementiere in der Klasse `PINGPONG` zusätzlich zum Interface `TICKER` nun auch noch das Interface `TASTENREAGIERBAR`:

```
/**
 * Spielsteuernde Klasse PingPong
 */
public class PINGPONG
implements TICKER, TASTENREAGIERBAR {
    ...
}
```



5. Übersetze deine Klasse `PINGPONG`, beachte die Fehlermeldung des Compilers und implementiere die Methode `tasteReagieren(...)`.

Vergiss unmittelbar darüber nicht `@Override` zu schreiben.

Die Methode soll folgendes leisten:

- Bei Taste W soll `vY` vom linken Schläger um Eins verringert werden.
- Bei Taste S soll `vY` vom linken Schläger um Eins erhöht werden.
- Bei Pfeil rauf soll `vY` vom rechten Schläger um Eins verringert werden.
- Bei Pfeil runter soll `vY` vom rechten Schläger um Eins erhöht werden.

*Vergiss den Java-Doc-Kommentar nicht!*

6. Melde im Konstruktor der Klasse `PINGPONG` dein `TASTENREAGIERBAR`-Objekt am `MANAGER`-Objekt an.



Dein Spiel konnte nur deshalb am **MANAGER**-Objekt angemeldet werden, weil es auch vom Daten-Typ der Interfaces ist.

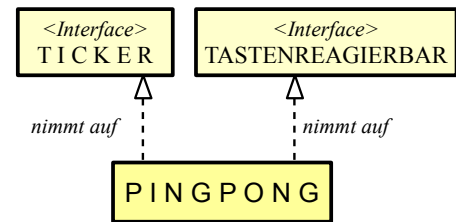
```
tickerAnmelden(TICKER t)
```

```
tastenReagierbarAnmelden(TASTENREAGIERBAR t)
```

Der Programmierer der Klasse **MANAGER** konnte nicht wissen, dass dein Spiel vom Daten-Typ **PINGPONG** sein wird.

Er wusste aber:

- Es wird vom Daten-Typ **TICKER** sein, wenn es auf Ticker-Signale reagieren will.
- Es wird vom Daten-Typ **TASTENREAGIERBAR** sein, wenn es auf Tastatur-Ereignisse reagieren will.



### praktische Zwischenübung erforderlich ... ?

Erinnerst du dich noch an das Projekt mit der Klasse **AMPEL**?

Die Ampel verfügt über die Methoden `einschalten()`, `ausschalten()`, `rotSchalten()`, `gruenSchalten()` und `weiterSchalten()`.

Implementiere nun die Interfaces **TICKER** und **TASTENREAGIERBAR** und erweitere deine Klasse **AMPEL** um folgende Funktionen:

- Die Ampel soll über Tastatur-Eingaben ein- und ausgeschaltet werden können.
- Die Ampel soll – wenn sie an ist – zeitgesteuert weiter schalten.

Vergiss das `@Override` und die Java-Doc-Kommentare nicht!



## Der Ball ist im Aus

Wenn der Ball links oder rechts am Spielfeldrand nicht auf einen Schläger trifft, so soll das Spiel kurz anhalten und der Punktestand aktualisiert werden.. Mit Hilfe eines Tastendrucks soll ein neuer Ball erscheinen und ein neues Spiel beginnen.

- ❓ Brauchst du in irgend einer Klasse neue Attribute?  
Von welchem Daten-Typ sind diese Attribute?  
Wo im JAVA-Code der Klasse musst du Veränderungen vornehmen?
- ❓ Wie erhöht man in der EDU-Version der Engine Alpha den Punktestand?
- ❓ Formuliere in Alltagssprache ein Kriterium, wann der Ball im Aus ist.
- ❓ Kannst du dein Kriterium mit Hilfe von Methoden der Klassen BALL, SCHLAEGER oder PINGPONG formulieren?  
Oder müssen in den Klassen dazu neue Methoden geschrieben werden?
- ❓ Was muss getan werden, damit z.B. beim Drücken der Taste N ein neuer Ball erscheint und damit ein neues Spiel beginnt.  
Schreibe ganz genau alle nötigen Einzelschritte auf!
- ❓ In welcher Klasse und an welcher Stelle dort musst du den JAVA-Code für den Fall „Der Ball ist im Aus“ schreiben.



### Spielregel: Ball im Aus

1. Deklare und initialisiere in der Klasse *PINGPONG* Attribute für die beiden Punktestände.
2. In der Methode Tick müssen nach den drei Bewegungs-Anweisungen folgende Dinge erledigt werden:
  - Der Fall „Ball rechts im Aus“  
**WENN** die x-Koordinate des Balls größer ist als die x-Koordinate des rechten Schlägers  
**DANN** erhöhe den Punktestand links um Eins,  
aktualisiere die Punkte-Anzeige entsprechend,  
bitte den Manager, den Ticker wieder abzumelden.
  - Der Fall „Ball links im Aus“  
analog ... aber bedenke, dass die x-Koordinate des Balls nun kleiner sein muss ...
3. Schreibe in der Klasse *PINGPONG* die neue Methode *neuerBall()*:
  - Setze den Ball zurück auf die Spielfeld-Mitte.
  - Bitte den Manager, den Ticker wieder mit 10 Millisekunden beim PingPong anzumelden.
4. Führe in der Methode *tasteReagieren(...)* der Klasse *PINGPONG* eine bedingte Anweisung für die Taste N ein und lasse in diesem Fall die Methode *neuerBall()* ausführen.

## Der Ball trifft den Schläger

Noch ignoriert der Ball die Schläger und läuft einfach durch sie hindurch. Das muss sich ändern, damit wir endlich ein funktionierendes Spiel haben.



Mache dir noch einmal ganz unmittelbar bewusst, welche Klasse über Referenz-Attribute vom Daten-Typ anderer Klassen verfügt. Beantworte nun folgende Fragen:



Kann aus der Klasse PINGPONG heraus ein Ball-Objekt angesprochen werden?

Kann aus der Klasse PINGPONG heraus ein SCHLAEGER-Objekt angesprochen werden?

Kann aus der Klasse BALL heraus ein SCHLAEGER-Objekt angesprochen werden?

Kann aus der Klasse SCHLAEGER heraus ein BALL-Objekt angesprochen werden?



In welcher Klasse muss nach diesen Erkenntnissen also der „Kollisionstest“ stattfinden?



Formuliere in Alltagssprache ein Kriterium, wann der Ball einen Schläger trifft.



Kannst du dein Kriterium mit Hilfe von Methoden der Klassen BALL, SCHLAEGER oder PINGPONG formulieren?



Was muss eigentlich „unter der Haube“ geschehen, wenn der Ball auf einen Schläger trifft?

Die Frage danach, ob der Ball einen Schläger trifft ist in ihrer Formulierung so komplex, dass es sich anbietet, hierfür eine eigene Methode zu schreiben. Da man auf eine Frage auch eine Antwort erwartet, braucht diese Methode eine Rückgabe.



In welcher Klasse muss diese Methode geschrieben werden?

Von welchem Daten-Typ ist die Rückgabe dieser Methode?

Braucht sie Übergabe-Parameter?




Von welchem Daten-Typ ist der Parameter.



Auf Grund der Komplexität dieses Problems wird dir diese Methode hier vorgegeben, falls du keine eigene Lösung gefunden haben solltest:

Schreibe in der Klasse PINGPONG folgende Methode:

```
/**
 * Methode zum Kollisions-Test von Ball und Schlaeger
 *
 * @param b    das BALL-Objekt
 * @param s    eines der beiden SCHLAEGER-Objekte
 */
private boolean trifft(BALL b, SCHLAEGER s) {
    if ( b.nenneX() > s.nenneX() - s.nenneBreite()/2 - b.nenneRadius() &&
        b.nenneX() < s.nenneX() + s.nenneBreite()/2 + b.nenneRadius() &&
        b.nenneY() > s.nenneY() - s.nenneHoehe()/2 - b.nenneRadius() &&
        b.nenneY() < s.nenneY() + s.nenneHoehe()/2 + b.nenneRadius() ) {
        return true;
    }
    else {
        return false;
    }
}
```

-  Versuche die Bedingungen in Alltagssprache zu formulieren. Mache dir dazu eine Skizze.
-  Wieso erhält diese Methode wohl den Modifikator *private* und nicht *public*?
-  Sieh dir deine Methode *tick()* in der Klasse *PINGPONG* noch einmal ganz genau an.  
Wo genau in der Methode *tick()* musst du den Kollisionstest unterbringen?



#### Spielregel: Ball trifft Schläger

1. Füge in der Methode *tick()* unmittelbar vor die Zeile *this.ball.bewegen()* eine bedingte Anweisung ein:

**WENN** der Ball den linken Schläger trifft **ODER** der Ball den rechten Schläger trifft,

**DANN** ändere das Vorzeichen von *vX* beim Ball.

(TIPP: Schreibe in der Klasse *Ball* eine Methode *vorzeichenVonVxAendern()*, welche den bisherigen Wert von *vX* mit *-1* multipliziert.)

Erst nach dieser bedingten Anweisung soll der Ball bewegt werden.

Gratuliere! Du hast dein erstes Spiel funktionsfähig – wenn auch noch nicht sehr aufregend – fertig programmiert.


Spiele ein paar Runden um zu sehen, ob auch wirklich alles so funktioniert, wie du es dir vorgestellt hast.

Nun ist es an der Zeit, alles neu Gelernte erst einmal zu wiederholen und zu festigen ...

## Selbstkontrolle: Hast du auch alles verstanden?


... dann kannst du bestimmt die folgenden Fragen ohne Probleme beantworten.


Wenn nicht, dann arbeite das entsprechende Teil-Kapitel noch einmal durch ...

 Nimm dir etwas Zeit und versuche alle neu gelernten Fachbegriffe in eigenen Worten zu erklären:


*Interface, Überschreiben einer Methode,*


Ergänze hierzu dein „Vokabelheft“ ... ergänze bei Bedarf eigene Beispiele ...

 Unterscheide die Begriffe *Methode, Methoden-Kopf, Methoden-Rumpf*.

 Wozu dient die Klasse *MANAGER*?  
Was hat sie mit den Interfaces *TICKER* und *TASTENREAGIERBAR* zu tun?

 Was ist alles zu tun, wenn du das Interface *TICKER* in eine eigene Klasse implementieren willst?

 Was ist alles zu tun, wenn du das Interface *TASTENREAGIERBAR* in eine eigene Klasse implementieren willst?

 Erkläre den Sinn des Schlüsselworts *@Override*.

 Was weißt du alles über die Verwendung des Schlüsselworts *this*?

 Fertige ein Klassen-Diagramm deines PINGPONG-Spiels nach dem bisherigen Stand.

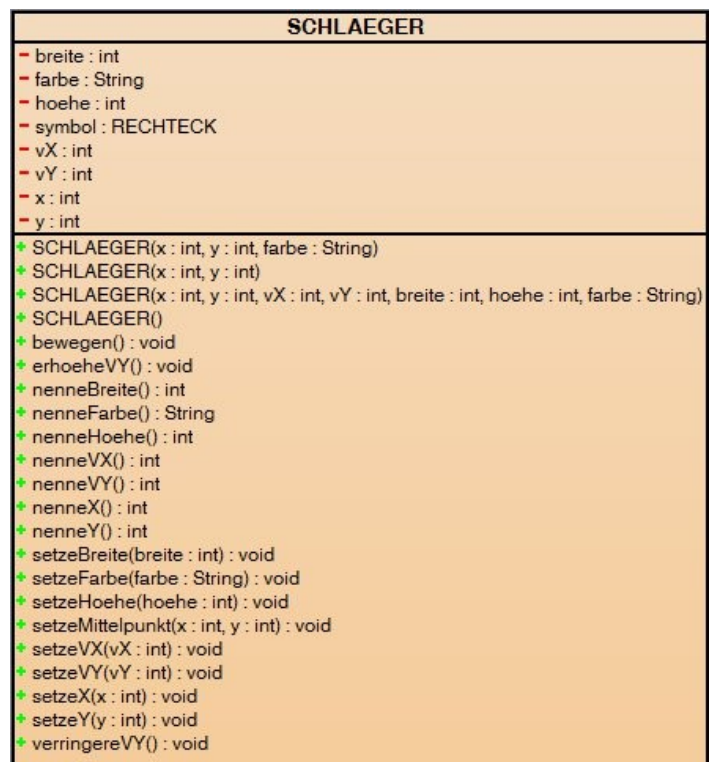
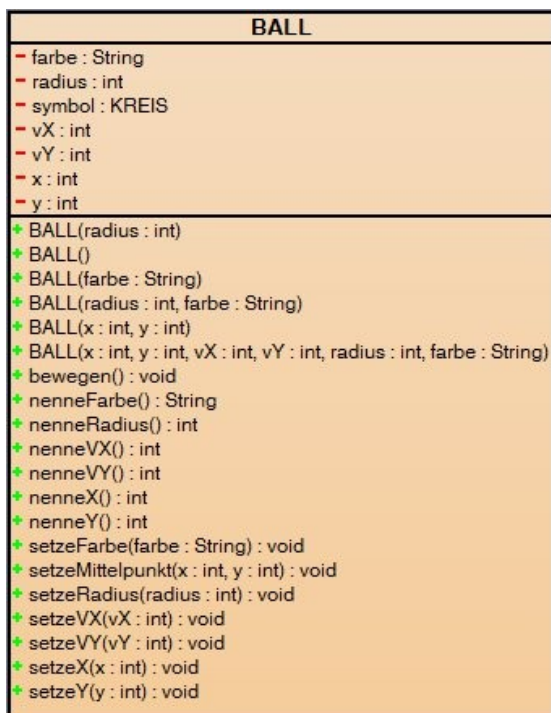
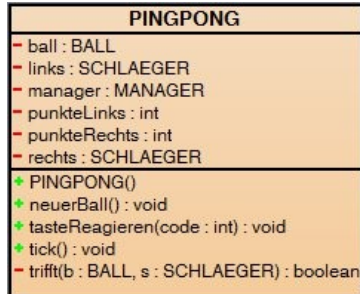
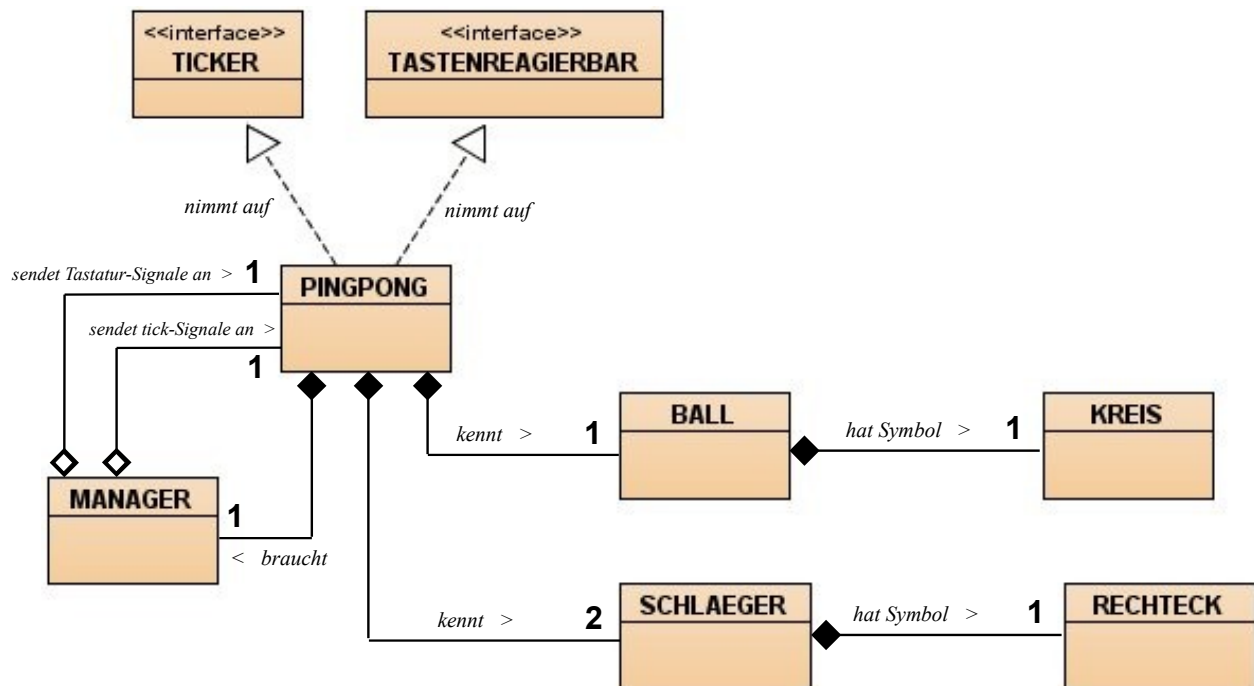
 Fertige eine ausführliche Klassenkarte deiner Klasse *PINGPONG*.

 Zeichne ein Sequenz-Diagramm der Methode *neuerBall()* in deiner Klasse *PINGPONG*.

 Zeichne ein Sequenz-Diagramm der Methode *tick()* in deiner Klasse *PINGPONG*.

## Zusammenfassung

Du hast nun ein funktionstüchtiges Spiel programmiert, dass sich folgendermaßen in seinem Logischen Aufbau darstellen lässt:



## Ausblick

Im nächsten Teil wirst du

- einige **weitere Spielregeln implementieren**, damit das Spiel etwas aufregender wird.
- **Gemeinsamkeiten** von BALL und SCHLAEGER **erkennen**.
- Mit dieser Erkenntnis **deinen Programmierstil verbessern**.
- **Ein ausführbares Programm** aus deinem Spiel machen, das jeder an jedem Rechner spielen kann.

