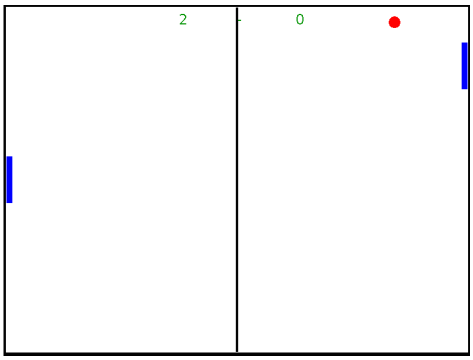
JAVA-Projekt : Das Spiel PingPong



Eines der ersten Computerspiele war „Ping-Pong“.

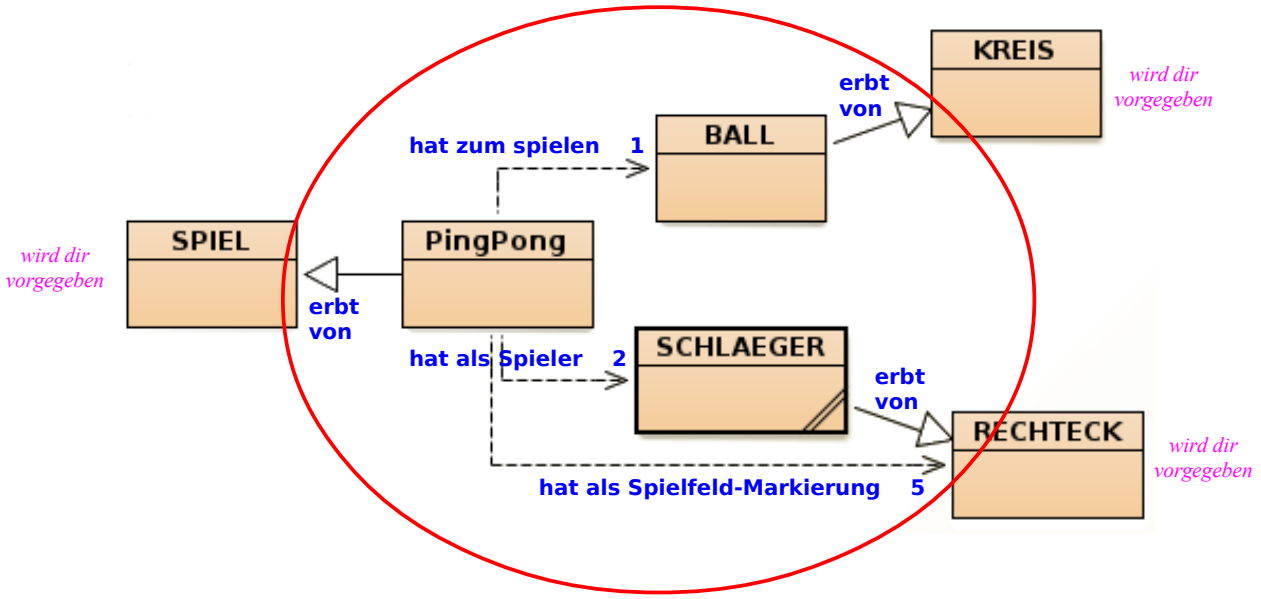
Zwei Rechtecke dienen als Schläger, ein Kreis dient als Ball. Der Ball reflektiert oben und unten am Spielfeldrand und auch an den Schlägern. Die Schläger können über die Tastatur nach oben und unten (vielleicht auch nach links und rechts) gelenkt werden. Es sind auch „Schmetterschläge“ und „Stops“ denkbar oder gar „Abpraller“ am Schlägerrand …

Wir wollen im Folgenden dieses uralte Spiel aus objektorientierter Sichtweise analysieren und es Stück für Stück programmieren.

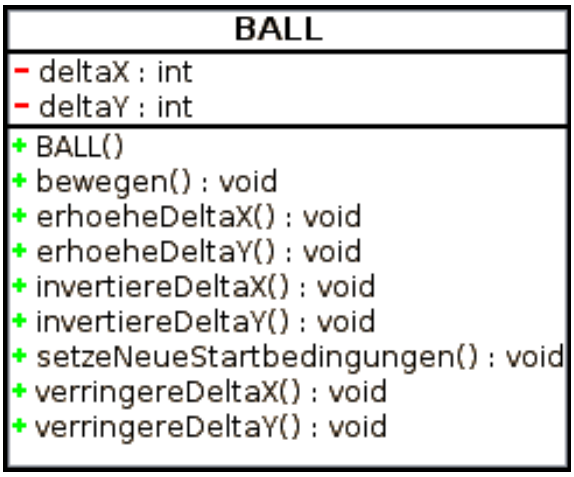
Voraussetzungen an die Programmierkenntnisse (JAVA):

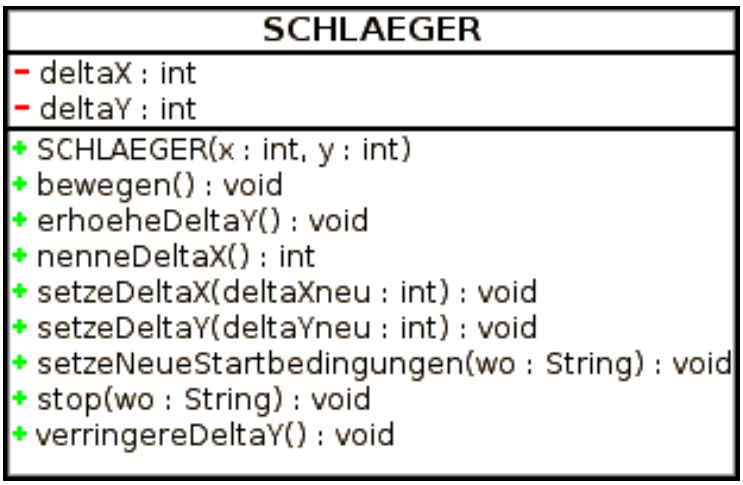
- elementarer Aufbau einer Klasse:
  Attribut, Daten-Typ, Modifikator, Wert-Zuweisung , Referenz, Methode, Übergabe-Parameter, Rückgebe-Wert, Konstruktor, Kapselungs-Prinzip
- grundlegendes Prinzip der Vererbung:
  übernehmen von Attributen und Methoden,
  ergänzen von Attributen und Methoden,
  überschreiben von Methoden
- einfache bis mehrfache Fallunterscheidungen:
  Vergleichs-Operatoren, logische Operatoren
- JAVA-Doc:
  Sinn und Aufbau von Dokumentations-Kommentaren,
  Erstellen einer HTML-Dokumentation
- Grund-Erfahrung mit der EDU-Variante der Engine-Alpha
  `http://engine-alpha.org`

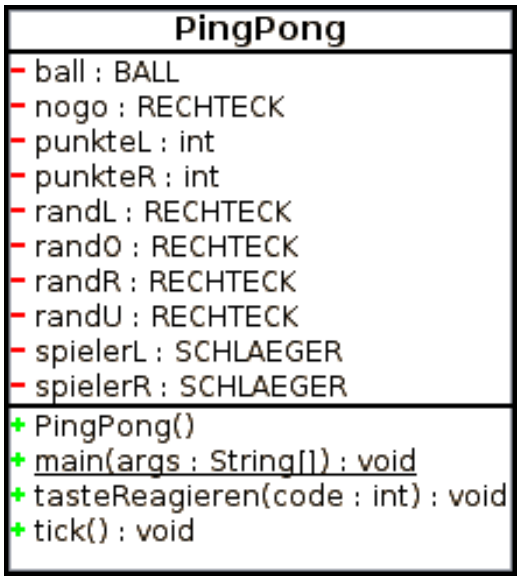
Klassen-Diagramm



Klassen-Karten







Du musst nur diese drei Klassen programmieren. Übernimm die Klassen SPIEL, RECHTECK, KREIS aus der Vorlage.

In der JAVA-Dokumentation des fertigen PingPong-Spiels findest du alle nötigen Detail-Informationen.

Die dort auftauchenden Klassen MANAGER und ZUFALL sowie die Interfaces TICKER und TASTENREAGIERBAR sind nicht wichtig für dein Verständnis. Sie gehören zur Klasse SPIEL.

Schritt 1

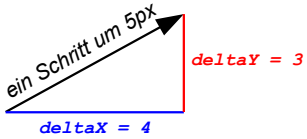
Hake alle erledigten Arbeiten im Klassendiagramm bzw. in der Klassenkarte *BALL* ab !

- Lade dir das BlueJ-Projekt *PingPong_Vorlage* herunter und öffne es in BlueJ. Speichere es sofort unter dem neuen Namen *PingPong_01*.

- Klicke mit Rechts auf die Klassen-Karte von KREIS und sieh dir genau an, welche Attribute und Methoden du gleich von dieser Klasse erben wirst. *Das ist WICHTIG!*

- **Erstelle nun selbst eine Klasse *BALL* welche von *KREIS* erbt.**

- Deklariere alle Attribute, welche NICHT schon von *KREIS* geerbt werden.
  Hierbei handelt es sich um die Schrittweiten (`deltaX` und `deltaY`) in x- bzw. y-Richtung. Sie geben an, wie weit der Ball sich (in Pixel) bei einem Schritt bewegen wird.



- Schreibe einen Konstruktor, welcher folgende Attribute initialisiert:
  Arbeite nur mit geerbten Methoden, wenn du geerbte Attribute verändern willst ! ! !
  Verändere geerbte Attriut-Werte NIE durch direkte Wert-Zuweisungen ! ! !
  - Die Farbe eines neuen Balls ist gelb.
  - Radius eines neuen Balls ist 10.
  - Mittelpunkt eines neuen Balls liegt bei (x|y) = (400|300)
  - Schrittweiten: `deltaX` ist 1 *(nach rechts)*,
    `deltaY` ist -1 *(nach oben)*

  Hier musst du natürlich mit Wert-Zuweisungen arbeiten, da es sich NICHT um geerbte Attribute handelt.

- Schreibe die Methode `bewegen()`, welche in ihrem Rumpf die geerbte Methode `verschiebenUm(... , …)` aufruft und übergib ihr die Schrittweiten `deltaX` und `deltaY` als Parameter.

- Damit der Ball später am Rand reflektieren kann, braucht er die Methode `invertiereDeltaY()`, welche das Vorzeichen von `deltaY` ändert.
  Analog dazu brauchst du eine Methode `invertiereDeltaX()`, damit der Ball an den Schlägern reflektieren kann.

- Erzeuge ein Objekt der Klasse *BALL* und betrachte es im Objekt-Inspektor.
  Sind alle Attribute mit den richtigen Werten belegt?
  Teste nun alle Methoden und prüfe (durch nachrechnen), ob die entsprechenden Attribut-Werte richtig verändert werden.

- Schreibe zu deiner Klasse und allen Methoden JAVA-Doc-Kommentare und erzeuge die JAVA-Dokumentation neu.
  Kontrolliere, ob all deine Programmierleistungen in der Dokumentation erscheinen und verständlich erklärt sind.

Schritt 2

Hake alle erledigten Arbeiten im Klassendiagramm bzw. in der Klassenkarte *SCHLAEGER* ab !

- **Erstelle eine Klasse *SCHLAEGER* welche von *RECHTECK* erbt.**

- Deklariere NUR Attribute, welche nicht von *RECHTECK* geerbt werden.
  Hierbei handelt es sich wieder um die Schrittweiten in x- bzw. y-Richtung.

- Schreibe einen Konstruktor mit zwei ganzzahligen Übergabe-Parametern, wodurch die Mittelpunkt-Koordinaten festgelegt werden können:
  Verwende zum Verändern und Abfragen von geerbten Attributen stets geerbte Methoden. Nur eigenen Attributen kann man mit dem Gleichheitszeichen einen neuen Wert zuweisen!
  - Die Farbe eines neuen Schlägers ist blau.
  - Ein neuer Schläger hat die Breite 10 und die Höhe 100.
  - Der Mittelpunkt eines neuen Schlägers wird auf die Werte der Übergabe-Parameter festgesetzt.
  - Ein neuer Schläger hat als Schrittweite: `deltaX` ist 0
    `deltaY` ist 1 *(nach unten)*

- Schreibe die Methode `bewegen()`, welche den Schläger um `deltaX` und `deltaY` verschiebt. *(s. Klassenkarte von RECHTECK)*

- Damit der Schläger später über die Tastatur gesteuert werden kann, brauchst du folgende Methoden:
  - `verringereDeltaY()` setzt den Wert von `deltaY` um `1` herab
  - `erhoeheDeltaY()` setzt den Wert von `deltaY` um `1` hinauf
  - Zusätzlich brauchst du noch `setzeDeltaX(...)` und `setzeDeltaY(...)`, welche jeweils einen ganzzahligen Übergabe-Parameter entgegennehmen und den entsprechenden Attribut-Wert auf den Wert des Übergabe-Parameters setzen.

- Erzeuge je ein Objekt der Klasse *SCHLAEGER* auf der linken und rechten Spielfeld-Seite und betrachte sie im Objekt-Inspektor.
  Sind alle Attribute mit den richtigen Werten belegt?
  Teste nun alle Methoden und prüfe (durch nachrechnen und Vergleich mit dem Objekt-Inspektor), ob die entsprechenden Attribut-Werte richtig verändert werden.

- Schreibe zu deiner Klasse und allen Methoden JAVA-Doc-Kommentare und erzeuge die JAVA-Dokumentation neu.
  Kontrolliere, ob all deine Programmierleistungen in der Dokumentation erscheinen und verständlich erklärt sind.

Schritt 3

Hake alle erledigten Arbeiten im Klassendiagramm bzw. in der Klassenkarte *PINGPONG* ab !

- **Erstelle eine Klasse *PINGPONG* welche von *SPIEL* erbt.**

- Deklariere alle Attribute, welche NICHT von *SPIEL* geerbt werden.
  Die Attribute `punkteL`, `punkteR` sowie die Referenz-Attribute `ball`, `spielerL`, `spielerR` erklären sich von selbst.
  Bei `randL`, `randO`, `randR`, `randU` handelt es sich um die Spielfeldränder.
  `netz` *(hieß früher nogo)* ist die Mittellinie.

- Schreibe einen parameterlosen Konstruktor, welcher folgende Attribute initialisiert:
  - Die beiden Punktestände haben den Startwert 0;
    Denke daran, die Punkte auch anzeigen zu lassen *(s. Klassenkarte von SPIEL)*
  - Der Ball muss erzeugt werden;
  - Die beiden Spieler müssen an unterschiedlichen Stellen erzeugt werden;
    Verwende im Konstruktor der Schläger geeignete Werte für M_x und M_y.
  - Die vier Ränder und das Netz müssen mit den richtigen Dimensionen *(Dicke 4 Pixel)* erzeugt und anschließend platziert werden*;*
  - Das Ticker-Intervall muss auf *10 (ms) eingestellt werden;* *(s. Klassenkarte von SPIEL)*

- Überschreibe die Methode `tick()`, welche alle 10ms automatisch aufgerufen wird.
  In ihr sollen der Reihe nach erst der Ball und danach der linke und der rechte Schläger bewegt werden. *(s. Sequenz-Diagramm auf der nächsten Seite)*

- Überschreibe die Methode `tasteReagieren(int code)`, welche bei jedem Tasten-Ereignis aufgerufen wird. *(s. Sequenz-Diagramm auf der nächsten Seite)*
  - Der linke Spieler soll mit den Tasten *W* und *S* gesteuert werden.
    W verringert `deltaY`, *S* erhöht `deltaY`.
  - Der rechte Spieler soll mit den *Pfeiltasten* gesteuert werden.
    Pfeil rauf verringert `deltaY`, *Pfeil runter* erhöht `deltaY`.

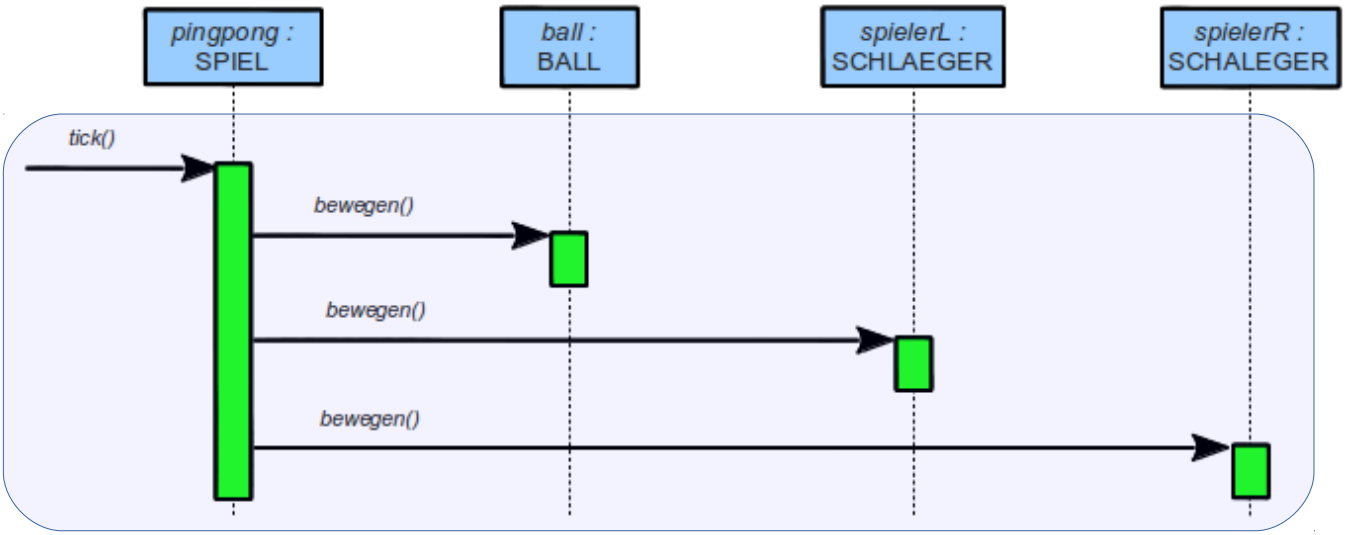
  Schreibe vor jeden Fall einen Kommentar um welche Taste es sich handelt damit du den Überblick nicht verlierst.

  Beispiel: `// Taste W`

- Erzeuge ein Objekt der Klasse *PINGPONG* und prüfe, ob sich Ball und Schläger bewegen. Reagieren die Schläger auf die Tasten?
  (Ball und Schläger werden noch das Spielfeld verlassen und nicht aufeinander reagieren.)

- Schreibe zu deiner Klasse und allen Methoden JAVA-Doc-Kommentare und erzeuge die JAVA-Dokumentation neu.
  Kontrolliere, ob all deine Programmierleistungen in der Dokumentation erscheinen und verständlich erklärt sind.

Sequenzdiagramm der Methode `tick()`

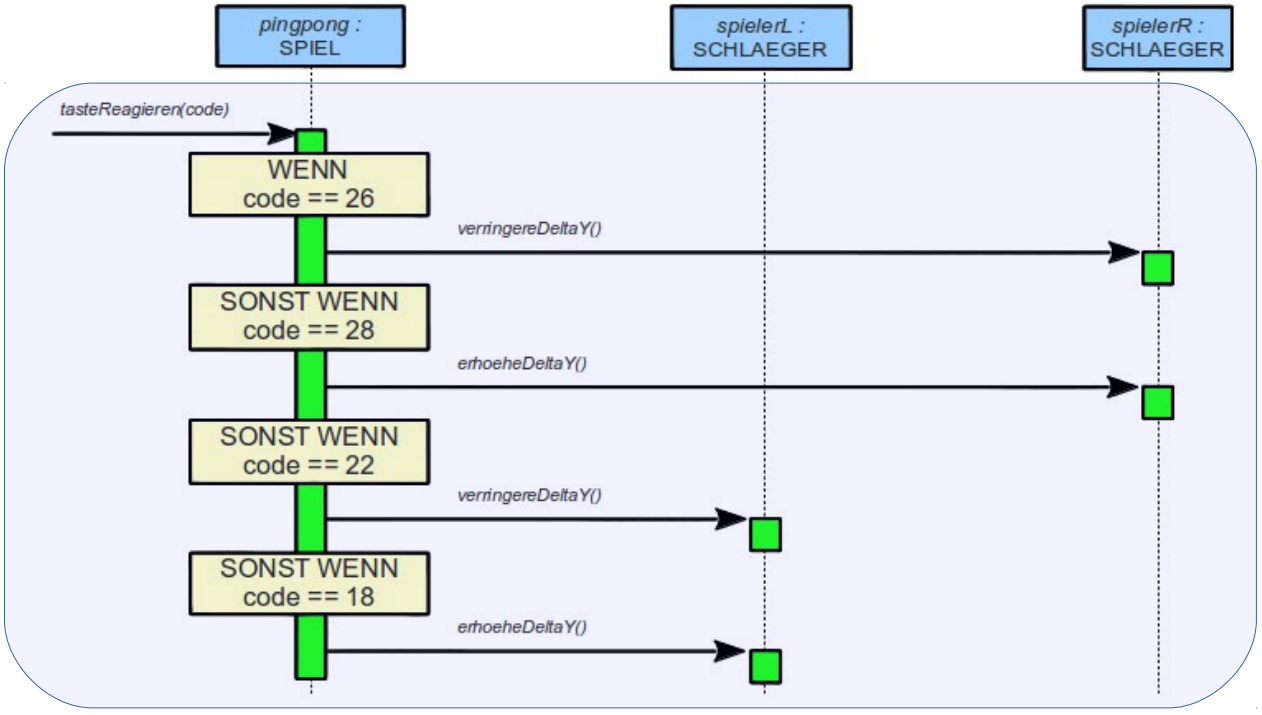


Lege in der Methode `tick()` unbedingt Kommentare an um den Überblick nicht zu verlieren. Es wir sehr viel Code in dieser Methode entstehen.

Beispiele:

```
// hier Behandlung des Balls
// hier Behandlung des linken Schlägers
// hier Behandlung des rechten Schlägers
```

Sequenzdiagramm der Methode `tasteReagieren(int code)`



Lege auch in der Methode `tasteReagieren(...)` unbedingt Kommentare an um den Überblick nicht zu verlieren. Die Nummern der Tasten sind doch recht verwirrend!

Beispiele:

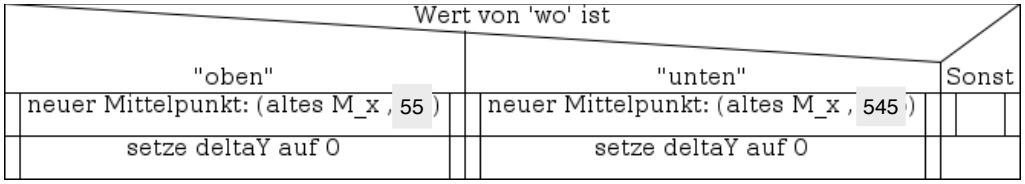
```
// Taste W
// Pfeil rauf
// ...
```

Schritt 4

Hake alle erledigten Arbeiten in den Klassenkarten ab !

Bringe die Schläger dazu, oben und unten am Spielfeldrand stehen zu bleiben.

- Dazu brauchst du in der **Klasse *SCHLAEGER*** eine Methode `stopp(String wo)` mit einem Übergabe-Parameter `'wo'` vom Typ `String`. Mögliche Werte für diesen Parameter sollen `"oben"` und `"unten"` sein.
  Im Rumpf der Methode schreibst du eine zweifache Fallunterscheidung:

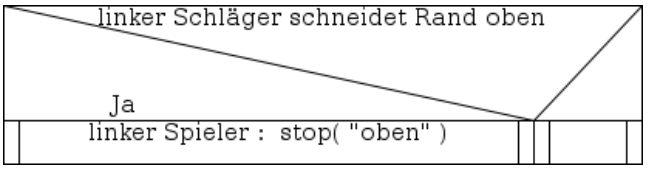


Damit stellst du sicher, dass der Schläger den oberen / unteren Rand um ein Pixel gerade nicht mehr berührt.
Das ist ganz wichtig! Sonst bleiben die Schläger später am Rand „kleben“!
Die y-Werte kommen folgendermaßen zustande: 55 = *halbe Schlägerhöhe + Randdicke + 1*
545 = *600 - halbe Schlägerhöhe - Randdicke – 1*

Du musst zwischen Schläger und Rand einen kleinen Spalt sehen können!

- Erzeuge ein Objekt der Klasse *SCHLAEGER* und prüfe, ob die Methode `stopp(...)` auf die entsprechenden Übergabe-Werte richtig reagiert.

- Deine Klassen *BALL* und *SCHLAEGER* haben die Methode `schneidet(...)` geerbt.
  Sieh dir diese Methode in den Klassenkarten und in der JAVA-Dokumentation genau an!
  Mit ihrer Hilfe kannst du sehr leicht prüfen, ob ein *SCHLAEGER*-Objekt ein *RECHTECK*-Objekt des Spielfeldrands berührt (oder schneidet).

- Begib dich nun in die **Klasse *PINGPONG*** und dort in die Methode `tick()`.
  Ergänze **unmittelbar vor** der Bewegung des linken Spielers eine bedingte Anweisung:
  (s. Sequenz-Diagramm auf der nächsten Seite)
  - 

  - Ergänze analog eine weitere bedingte Anweisung, die prüft, ob der untere Rand berührt (geschnitten) wird.

- Erzeuge nun ein Objekt der Klasse *PINGPONG* und prüfe, ob der linke Schläger tatsächlich oben und unten stehen bleibt und ob er sich nach einem entsprechenden Tastendruck wieder weiter bewegen lässt.
  Wenn nicht, so musst du die Werte 45 und 555 so verändern, dass der Schläger, wenn er stehen bleibt, den Rand NICHT berührt.

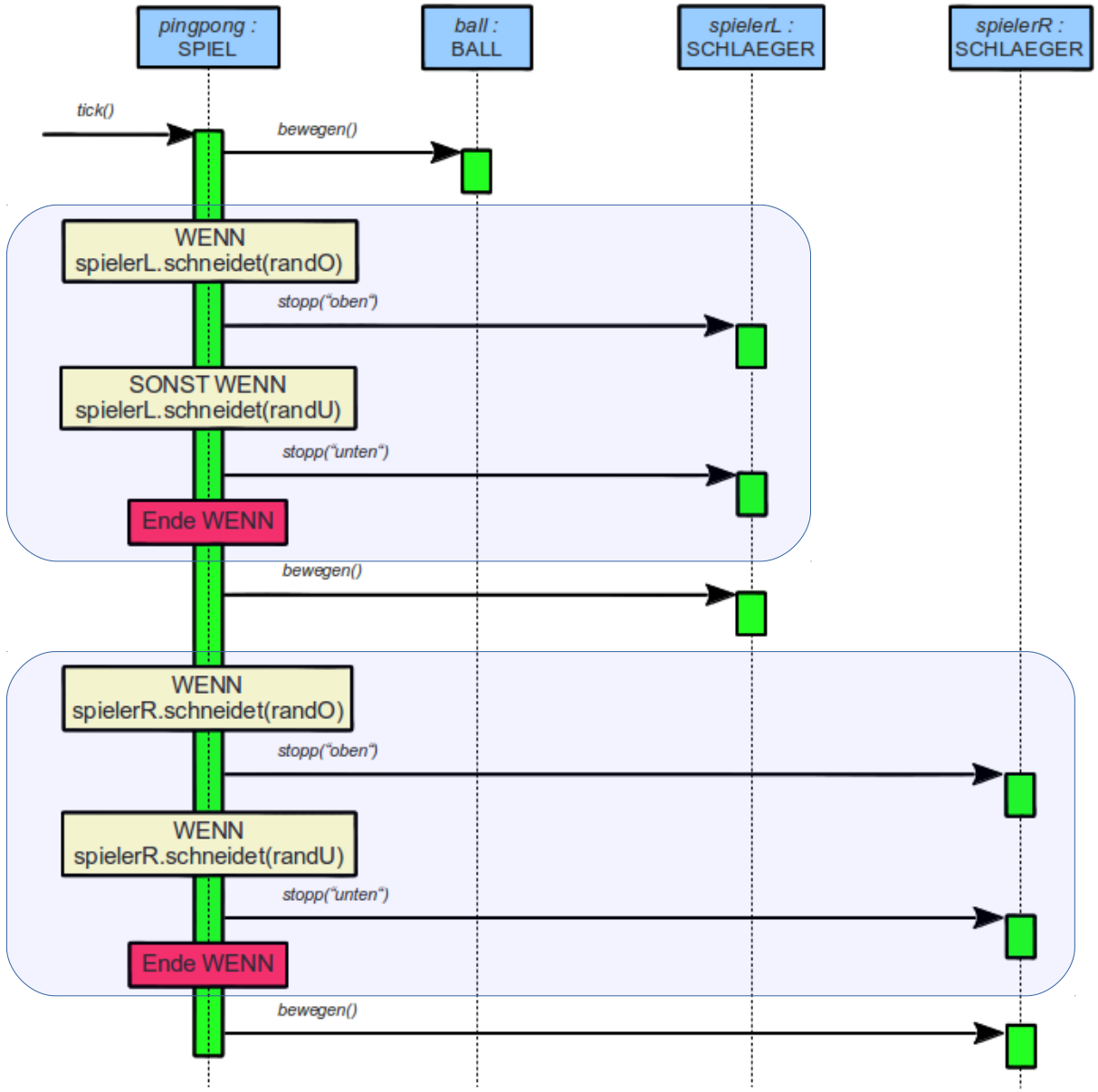
- Erledige nun dasselbe für den rechten Spieler.

- Erzeuge erneut ein Objekt der Klasse *PINGPONG* und prüfe auch den rechten Schläger.

- Schreibe zu allen neuen Methoden JAVA-Doc-Kommentare und erzeuge die JAVA-Dokumentation neu. Kontrolliere, ob all deine Programmierleistungen in der

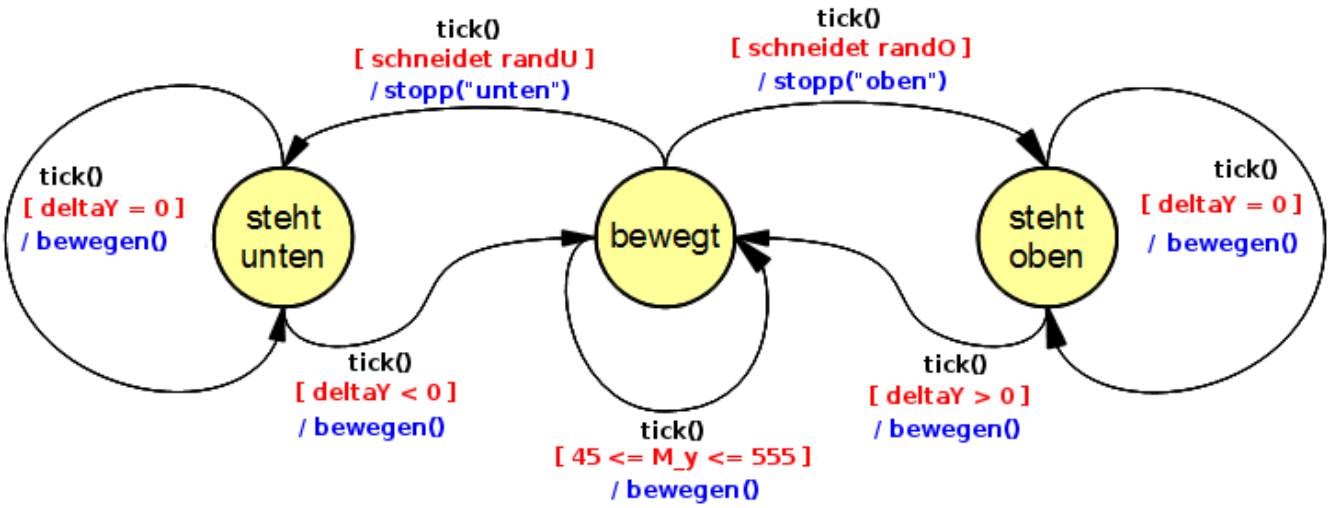
Dokumentation erscheinen und verständlich erklärt sind.

Neues Sequenz-Diagramm der Methode `tick()`



Ergänze deine Kommentare in der Methode `tick()`. *Es wird noch viel mehr Code kommen!*

Zustands-Übergangs-Diagramm des Schlägers



Schritt 5

Bringe den Ball dazu, am oberen und unteren Spielfeldrand sowie an den Schlägern zu reflektieren.

- Begib dich wieder in die **Klasse *PINGPONG*** und dort in die Methode `tick()`.
  Ergänze **unmittelbar vor** der Bewegung des Balls eine mehrfache Fallunterscheidung:

ball schneidet				
randO	randU	spielerL	spielerR	Sonst
ball: invertiere deltaY	ball: invertiere deltaY	ball: invertiere deltaX	ball: invertiere deltaX	

(s. Sequenz-Diagramm auf der nächsten Seite)

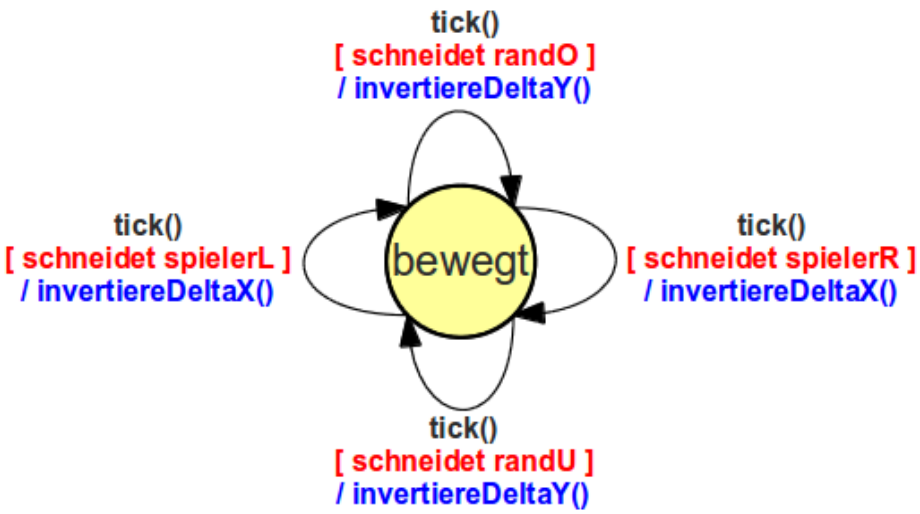
- Erzeuge nun ein Objekt der Klasse *PINGPONG* und prüfe, ob der Ball nun oben und unten sowie an den Schlägern abprallt.

- *Füge gewöhnliche Kommentare in die verschiedenen Fälle der Methode `tick()` ein, damit du dich später leicht wieder darin zurecht findest.*

Beispiele:

```
// Schnitt des Balls mit randO / randU
…
// Schnitt des Balls mit den Schlägern
…
```

Zustands-Übergangs-Diagramm des Balls

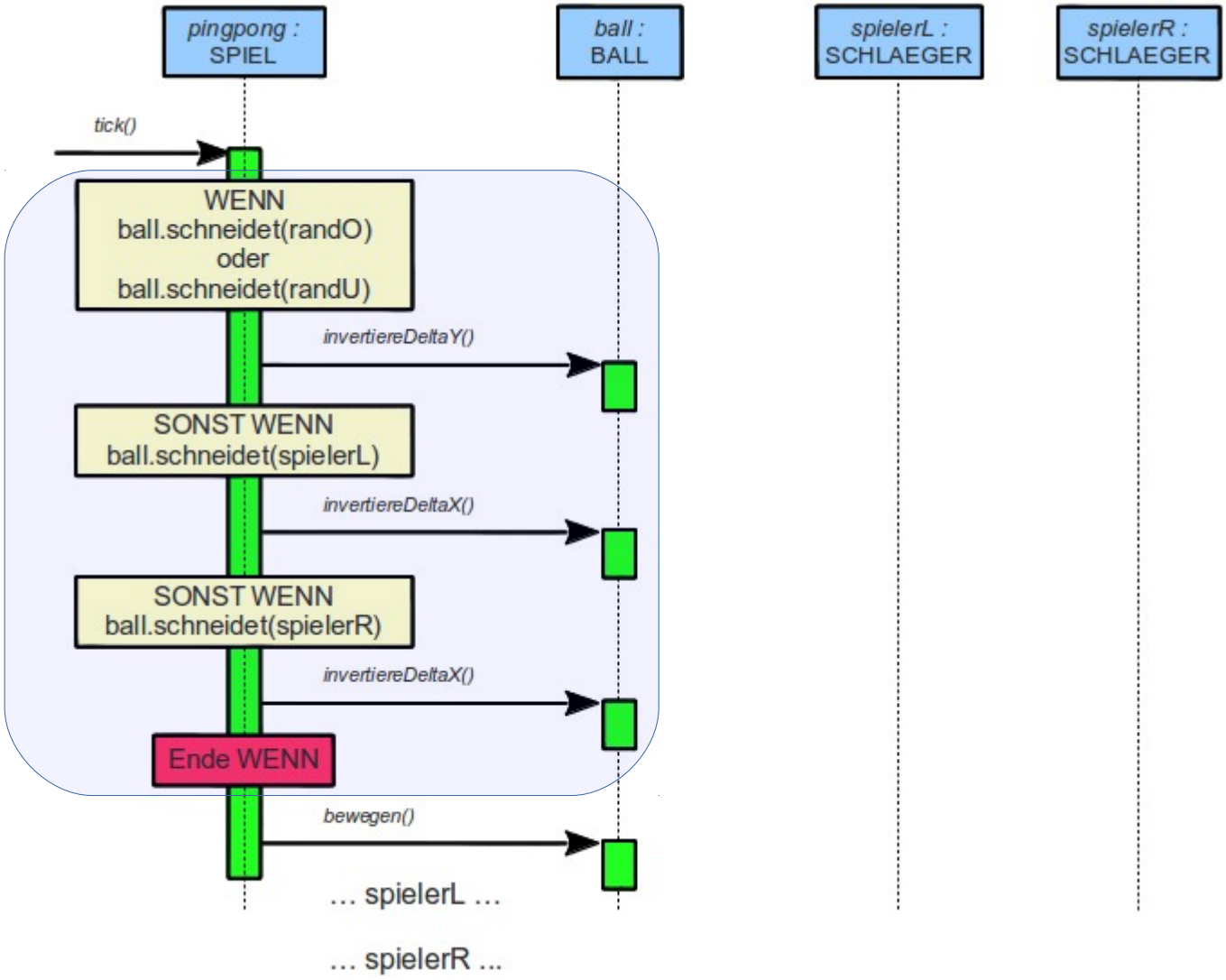


Genauere Betrachtung:

Die beiden Übergänge „schneidet randO“ und „schneidet randU“ haben dieselbe Wirkung. Deshalb kann man sie später zusammenfassen in „schneidet randU oder schneidet randO“. Ganz analog kann man auch „schneidet randL“ und „schneidet randR“ zusammenfassen. Dies erhöht auch die Übersicht!

Auf dieselbe Art und Weise kann später auch der JAVA-Code verkürzt und besser lesbar gemacht werden!

Neues Sequenz-Diagramm der Methode tick()



Ergänze deine Kommentare in der Methode `tick()`. *Es wird noch viel mehr Code kommen!*

Schritt 6

Verändere den Punktestand, wenn der Ball im Aus landet

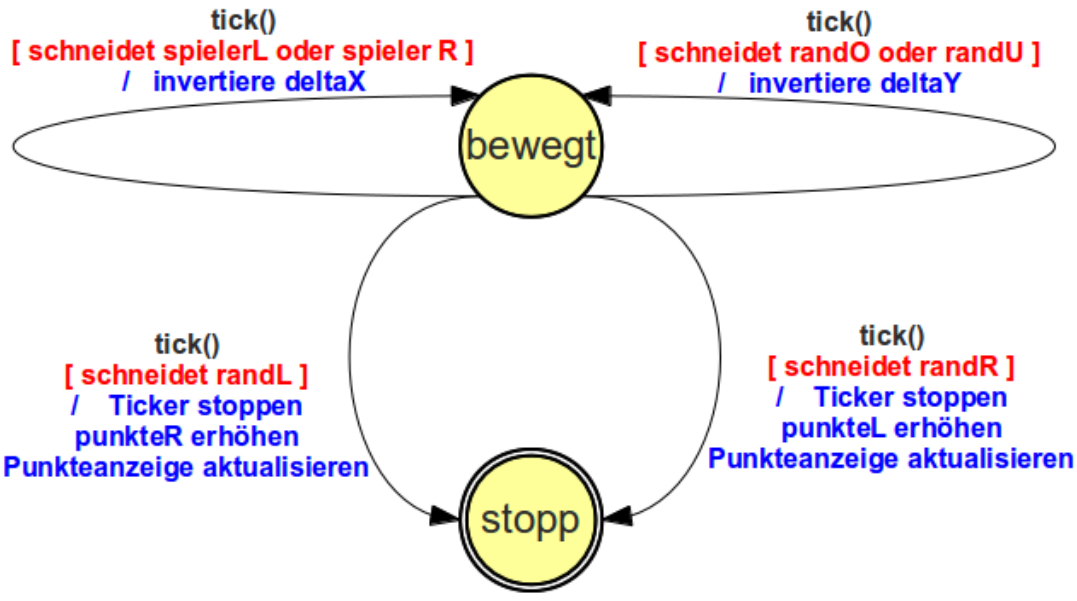
- Begib dich wieder in die **Klasse *PINGPONG*** und dort in die Methode `tick()`.
  Ergänze die mehrfache Fallunterscheidung aus Schritt 5 um zwei weitere Fälle:

Ball schneidet						
randO	randU	spielerL	spielerR	randL	randR	Sonst
...	...	...	...	Ticker stoppen punkteR erhöhen Punkteanzeige aktualisiern	Ticker stoppen punkteL erhöhen Punkteanzeige aktualisiern	

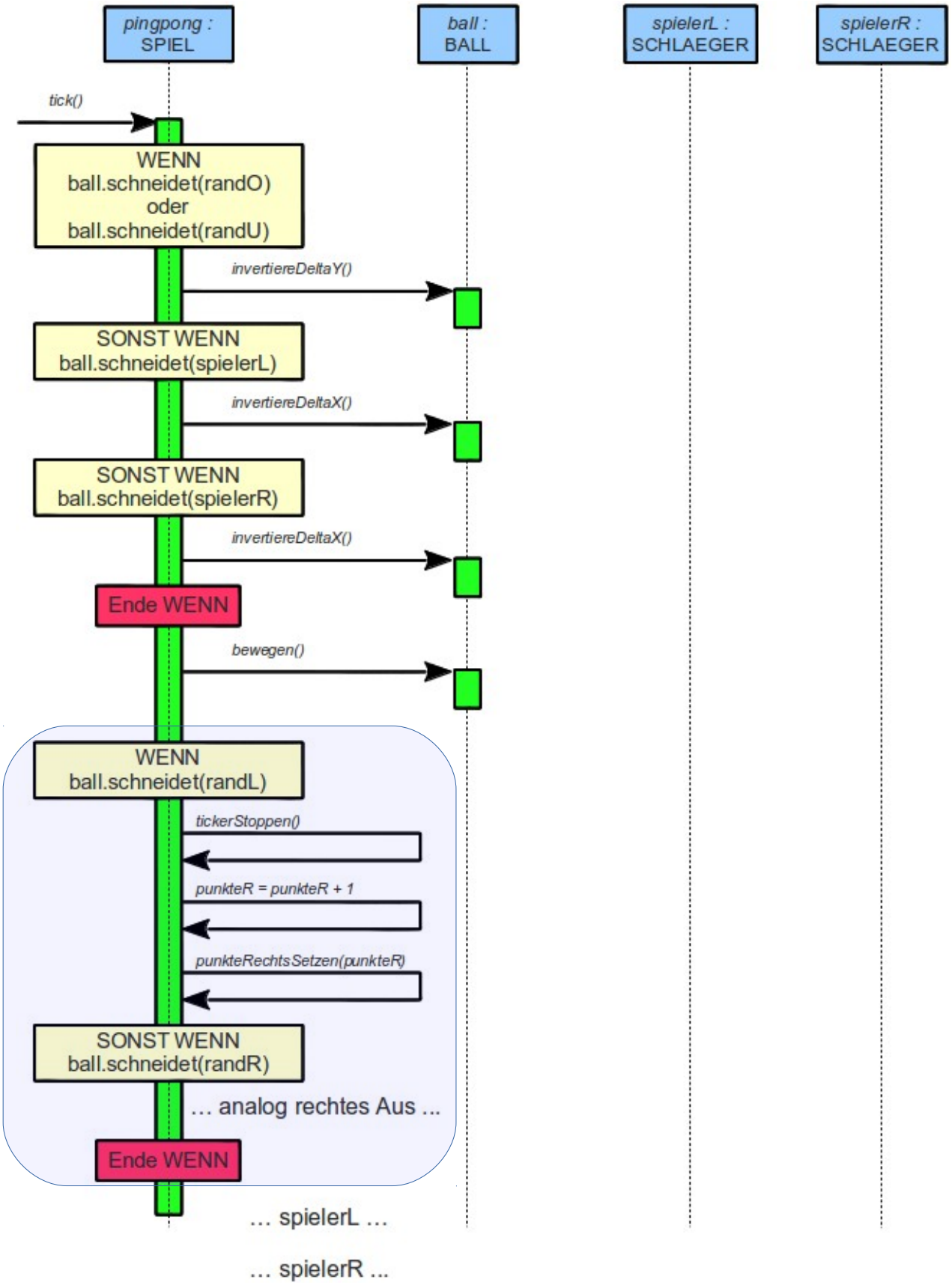
- Erzeuge nun ein Objekt der Klasse *PINGPONG* und prüfe, ob der Ball links bzw. rechts im Aus stehen bleibt und sich der entsprechende Punktestand erhöht.

- ***Füge wieder Kommentare für die neuen Fälle der Methode `tick()` ein,***
  damit du dich später leicht wieder darin zurecht findest.
  Achte darauf, dass dein immer länger werdender Code nicht chaotisch und unübersichtlich wird.
  → alles was dem Ball betrifft steht geschlossen im oberen Teil der Methode
  → im mittleren Teil deines Codes wird der linke Schläger behandelt
  → der untere Teil behandelt dann den rechten Schläger

Neues Zustands-Übergangs-Diagramm des Balls



Neues Sequenz-Diagramm der Methode tick()



Ergänze deine Kommentare in der Methode `tick()`. *Es wird noch viel mehr Code kommen!*

Schritt 7

Mit Taste N ein neues Spiel beginnen

Zum Beginnen eines neuen Spiels müssen Ball und Schläger wieder in ihre Startpositionen gebracht werden. Außerdem muss der Ticker neu gestartet werden.

- Begib dich in die **Klasse *BALL*** und schreibe dort eine neue Methode `setzeNeueStartbedingungen()`:
  - Setze den Mittelpunkt wieder in die Mitte des Spielfelds.
  - Setze die Schrittweite wieder auf den Betrag `1`, aber behalte die alte Richtung bei. Nutze dazu folgenden Code: *(der Wert wird durch seinen Absolut-Betrag dividiert)*

```
if (this.deltaX != 0) {
   this.deltaX = this.deltaX / Math.abs(this.deltaX);
}
else {
   this.deltaX = 1;
}
if (this.deltaY != 0) {
   this.deltaY = this.deltaY / Math.abs(this.deltaY);
}
else {
   this.deltaY = -1:
}
```

- Erzeuge ein Objekt der Klasse *BALL* und verschiebe es an eine beliebige Stelle. Teste nun die Methode `setzeNeueStartbedingungen()` und prüfe, ob der Ball wieder in der Spielfeldmitte steht.

- Füge einen Java-Doc-Kommentar zu der neuen Methode hinzu und erzeuge die JAVA-Dokumentarion neu.

- Begib dich in die **Klasse *SCHLAEGER*** und schreibe dort eine neue Methode `setzeNeueStartbedingungen(String wo)` mit einem Übergabe-Parameter `wo` vom Typ `String`.
  - Im Rumpf der Methode findet zuerst eine zwiefache Fallunterscheidung statt:

Wert von 'wo' ist		
"links"	"rechts"	Sonst
setze Mittelpunkt: (10,300)	setze Mittelpunkt: (790,300)	

  - Nach der Fallunterscheidung setzt du `deltaX` auf den Wert `0`, und anschließend setzt du `deltaY` auf den Betrag `1` und behältst die Richtung bei:

```
if (this.deltaY != 0) {
   this.deltaY = this.deltaY / Math.abs(this.deltaY);
}
else {
   this.deltaY = 1;
}
```

weiter auf der nächsten Seite ...

- Erzeuge ein Objekt der Klasse *SCHLAEGER* und verschiebe es an eine beliebige Stelle. Teste nun die Methode `setzeNeueStartbedingungen(String wo)` einmal mit Parameter `„links“` und noch einmal mit Parameter `„rechts“` und prüfe, ob der Schläger jeweils an der richtigen Stelle steht.

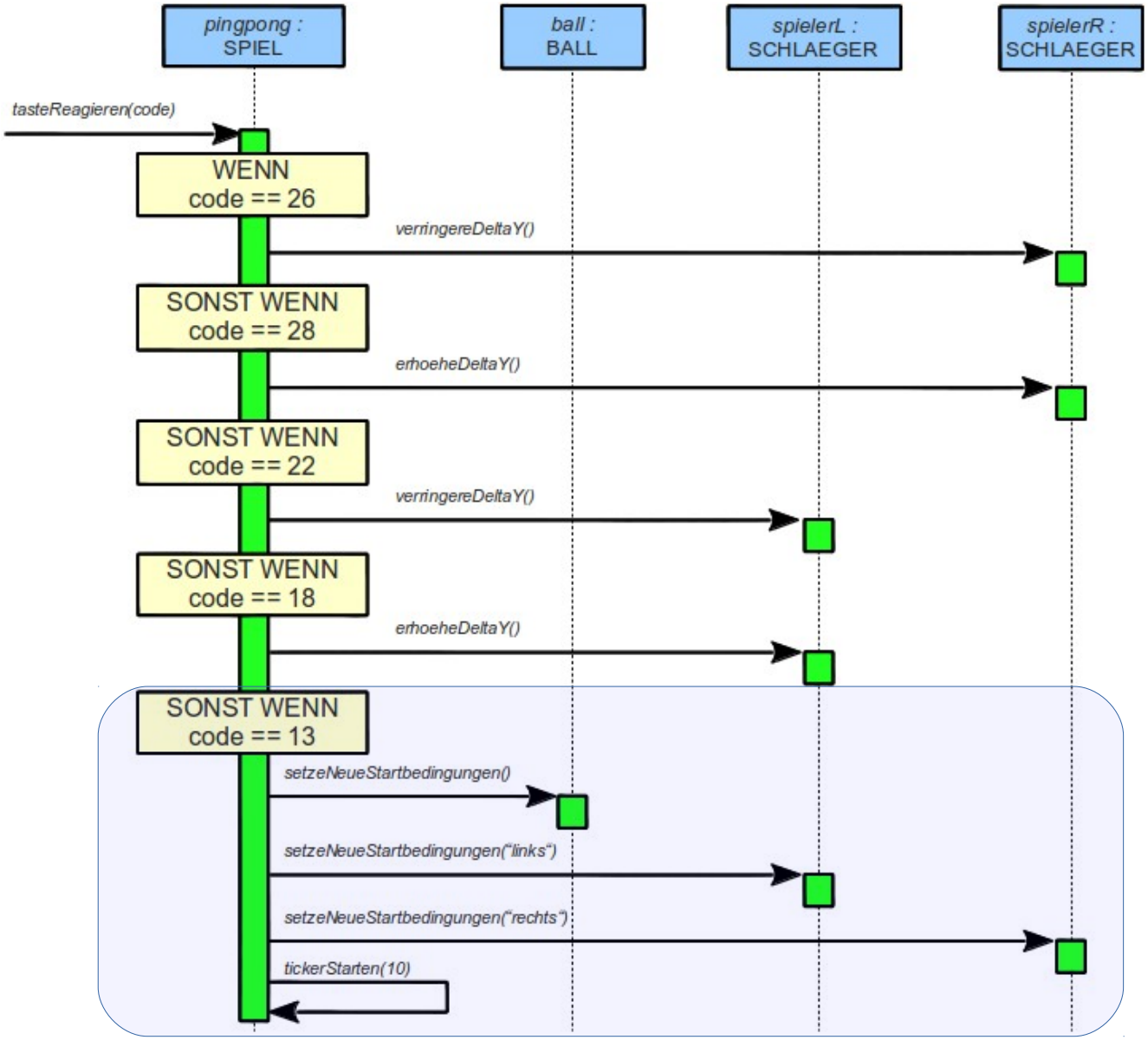
- Füge einen Java-Doc-Kommentar zu der neuen Methode hinzu und erzeuge die JAVA-Dokumentarion neu.

- Nun begibst du dich noch in die **Klasse *PINGPONG*** und dort in die Methode `tasteReagieren(int code)`. Ergänze die Mehrfachfallunterscheidung um einen weiteren Fall:

Wert von 'code' ist					
26	28	22	18	13	Sonst
...	...	...	...	ball: setze neue Startbedingungen spielerL: setze neue Startbedingungen: "links" spielerR: setze neue Startbedingungen: "rechts" pingpong: Ticker starten: 10ms	

- Erzeuge ein Objekt der Klasse *PINPONG* und drücke – nachdem der Ball im Aus ist – auf die Taste *N*. Beginnt ein neues Spiel?

Neues Sequenz-Diagramm der Methode `tasteReagieren(int code)`



Ergänze deine Kommentare in der Methode `tasteReagieren(...)`.

Schritt 8

Ermögliche „Rahmenabpraller“

Wenn der Ball einen Schläger ganz oben (unten) am Rand trifft, dann soll der Ball nach oben (unten) – also in y-Richtung – „vom Rahmen abprallen“. Für diese neue Verhaltensweise deines Balls brauchst du neue Methoden.

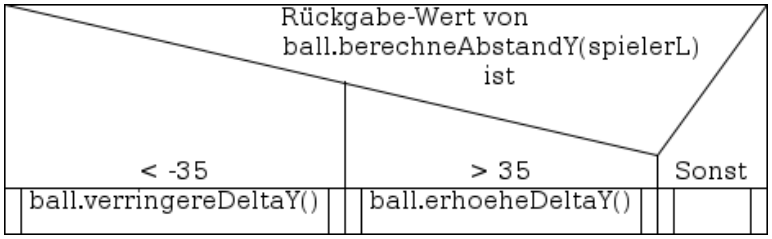
- Begib dich in die **Klasse *BALL***. Erstelle dort die beiden Methoden `erhoeheDeltaY()` und `verringereDeltaY()`:
  - `erhoeheDeltaY()` soll `1` zum Wert von `deltaY` dazu addieren.
  - `verringereDeltaY()` soll `1` vom Wert von `deltaY` subtrahieren.

- Erzeuge nun ein Objekt der Klasse *BALL* und betrachte es im Objekt-Inspektor. Rufe die Methoden `erhoeheDeltaY()` und `verringereDeltaY()` auf und beobachte den Attribut-Wert von `deltaY`. Rechne nach, ob er sich richtig verändert.

- Füge JAVA-Doc-Kommentare in die neuen Methoden ein und erzeuge die JAVA-Dokumentation neu.

Wenn der Ball nun einen Schläger trifft, dann prüfst du ob der Rahmen getroffen wurde oder nicht.
Hierfür eignet sich die Methode **`berechneAbstandY(...)`** *der Klassen BALL und SCHLAEGER. Sie vergleicht die Mittelpunkt-y-Koordinaten der beiden Objekte.*
Lies zum genaueren Verständnis die Dokumentation der Klasse RECHTECK oder KREIS durch.

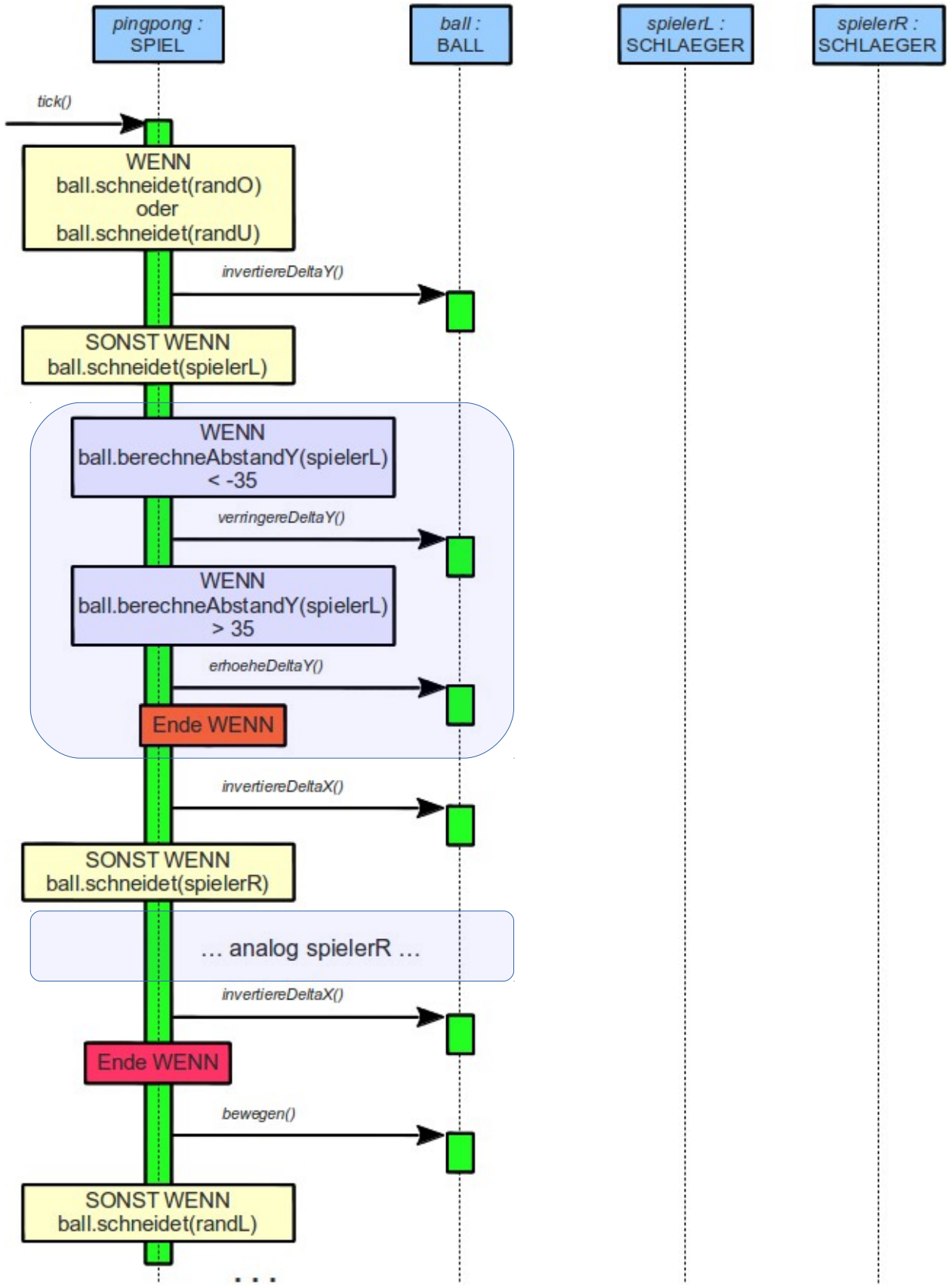
- Begib dich nun in der **Klasse *PINGPONG*** in die Methode `tick()`. Suche dort den Fall, wo der Ball den linken Schläger trifft (schneidet). Vor dem Invertieren des Wertes von `deltaX` fügst du eine doppelte Fallunterscheidung ein:



- Suche nun in der Methode `tick()` den Fall, wo der Ball den rechten Schläger trifft (schneidet) und verfahre analog.

- Erzeuge ein Objekt der Klasse *PINPONG* und versuche den Ball ganz am oberen (unteren) Rand des Schlägers zu treffen. Prallt er ab?

Neues Sequenz-Diagramm der Methode `tick()`



Ergänze deine Kommentare in der Methode `tick(...)`.

Schritt 9

Realisiere auch eine horizontale Schläger-Bewegung

- Begib dich wieder in die **Klasse *PINGPONG*** und dort in die Methode `tasteReagieren(int code)`.
  Ergänze die mehrfache Fallunterscheidung um zwei weitere Fälle für den rechten Schläger. Er soll auch auf `Pfeil rechts` und `Pfeil links` reagieren:

Wert von 'code' ist							
26	28	27	29	22	18	13	Sonst
...	...	spielerR: setze deltaX: 1	spielerR: setze deltaX: -1	...	...	...	

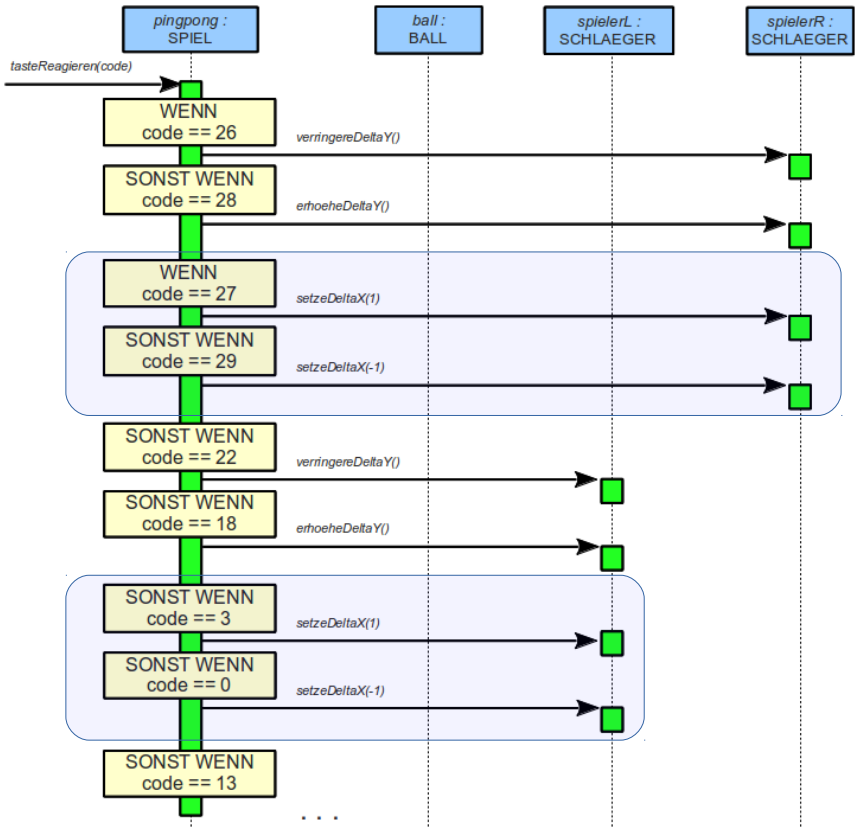
- Erzeuge nun ein Objekt der Klasse *PINGPONG* und prüfe, ob der rechte Schläger auf die neuen Tasten reagiert.
  Stört dich etwas an der neuen horizontalen Bewegung? *(Lösung in Schritt 10)*

- Ergänze nun analog zwei weitere Fälle für den linken Schläger.
  Er soll auch auf *Taste* `A` und *Taste* `D` reagieren:

Wert von 'code' ist								
26	28	27	29	22	18	3	0	13	Sonst
...	...	...	...	...	...	spielerL: setze deltaX: 1	spielerL: setze deltaX: -1	...	

- Erzeuge erneut ein Objekt der Klasse *PINGPONG* und prüfe, ob auch der linke Schläger auf die neuen Tasten reagiert.

Neues Sequenz-Diagramm der Methode `tasteReagieren(int code)`



Schritt 10

Schränke die Schläger-Bewegung auf eine Spielfeldhälfte ein

Ein Schläger-Objekt soll nun nicht nur oben und unten stehen bleiben können, sondern – je nachdem, welcher Spieler es ist – auch links, rechts, sowie am Netz.

- Begib dich zunächst in die **Klasse *SCHLAEGER*** und dort in die Methode `stopp(String wo)`.
  Ergänze die mehrfache Fallunterscheidung um vier weitere Fälle:

Wert von 'wo' ist						
"oben"	"unten"	"links"	"rechts"	"nogoL"	"nogoR"	Sonst
...	...	neuer Mittelpunkt: (10, altes M_y) setze deltaX: 0	neuer Mittelpunkt: (790, altes M_y) setze deltaX: 0	neuer Mittelpunkt: (392, altes M_y) setze deltaX: 0	neuer Mittelpunkt: (408, altes M_y) setze deltaX: 0	

- Erzeuge ein Objekt der Klasse SCHLAEGER an einem beliebigen Ort und rufe die Methode `stopp(String wo)` der Reihe nach mit den Übergabe-Werten `"links"`, `"rechts"`, `"nogoL"`, `"nogoR"` auf.
  Landet der Schläger auch immer an der richtigen Stelle?

- Liste alle zulässigen Werte für den Parameter `'wo'` im JAVA-Doc-Kommentar der Methode `stopp(String wo)` auf und erzeuge die JAVA-Dokumentation neu.

Wenn sich der Schläger auf den Ball zu bewegt, dann reagiert der Ball meist seltsam oder gar nicht auf den Schläger. Dies lässt sich ganz einfach beheben.

- Begib dich nun in die **Klasse *PINGPONG*** und dort in die Methode `tick()`.
  Suche den Fall *„ball schneidet spielerL“*. Ergänze am Ende (nach der Invertierung von `deltaX` des Balls) zwei Mal einen Methoden-Aufruf `bewegen()` an das Ball-Objekt.

- Verfahre ebenso beim Fall *„ball schneidet spielerR“*.

- Erzeuge ein Objekt der Klasse *PINGPONG* und teste, ob der Ball nun reflektiert, wenn ein Schläger sich auf ihn zu bewegt.
  Bist du mit der Reaktion des Balls auf den Schläger zufrieden? *(s. Schritt 11)*

- Suche nun die zweifache Fallunterscheidung *„spielerL schneidet randO / randU“*.
  Ergänze direkt darunter eine neue zweifache Fallunterscheidung:

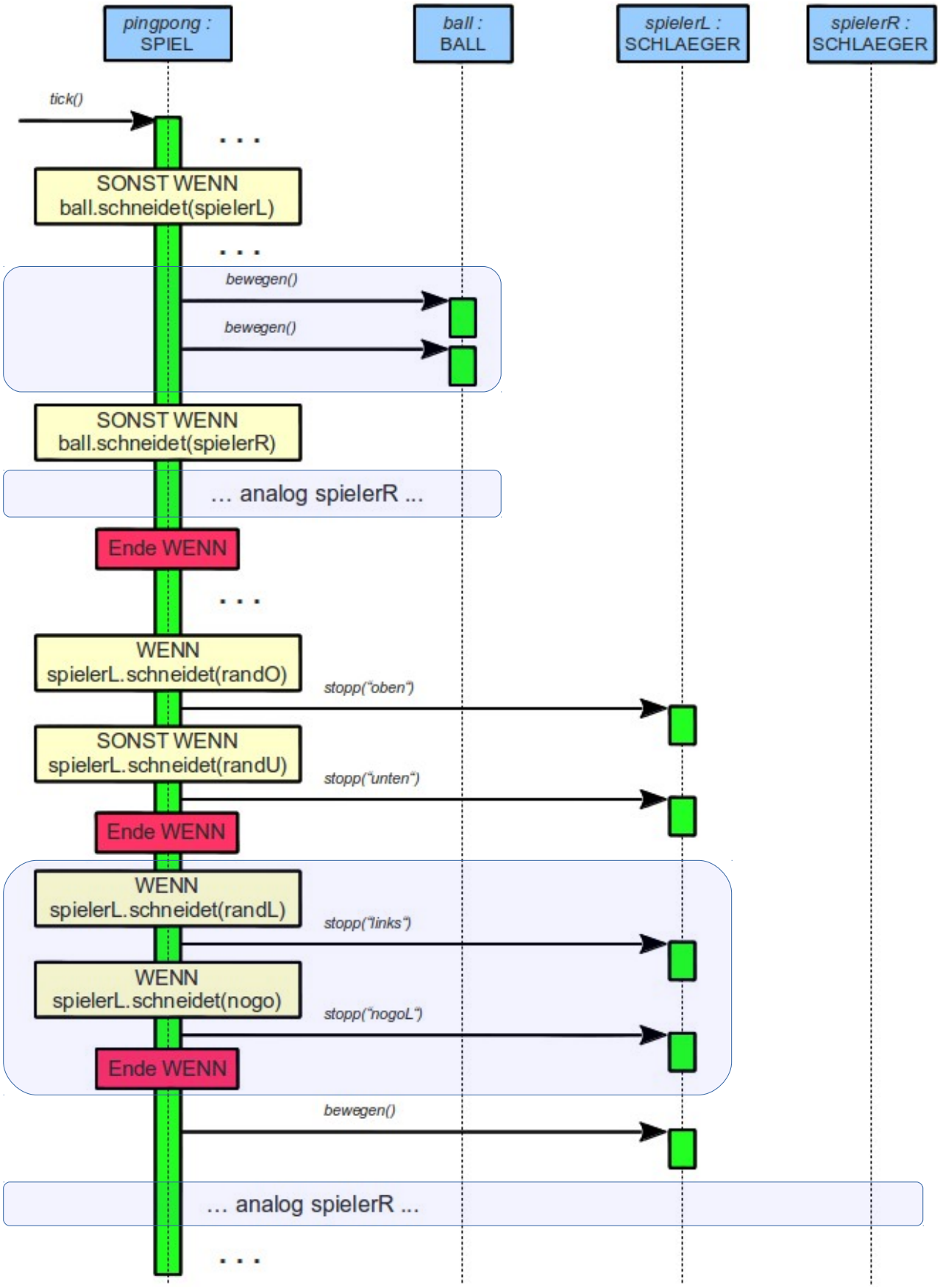
spielerL schneidet	
randL	nogo
spielerL: stopp: "links"	spielerL: stopp: "nogoL"

- Verfahre analog nach der zweifachen Fallunterscheidung *„spielerR schneidet randO / randU“*.

spielerR schneidet	
randR	nogo
spielerR: stopp: "rechts"	spielerR: stopp: "nogoR"

- Erzeuge ein Objekt der Klasse *PINGPONG* und teste, ob die Schläger nun innerhalb ihrer Spielfeldhälfte bleiben.

Neues Sequenz-Diagramm der Methode `tick()`



Ergänze deine Kommentare in der Methode `tick(...)`.

Schritt 11

Realisiere Schmetterschläge und Stoppbälle

- Begib dich in die **Klasse *SCHLAEGER*** und schreibe dort eine neue Methode `nenneDeltaX()`, welche den ganzzahligen Wert von `deltaX` zurück gibt.

- Füge der Methode einen JAVA-Doc-Kommentar hinzu und erzeuge die JAVA-Dokumentation neu.

- Begib dich in die **Klasse *BALL*** und schreibe dort eine neue Methode `erhoeheDeltaX()`, welche im Rumpf eine zweifache Fallunterscheidung hat:

Wert von deltaX ist		
> 0	< 0	Sonst
zähle 1 zu deltaX dazu	ziehe 1 von deltaX ab	

- Schreibe analog eine weitere Methode `verringereDeltaX()` in der Klasse BALL:

Wert von deltaX ist		
> 1	< -1	Sonst
ziehe 1 von deltaX ab	zähle 1 zu deltaX dazu	

- Erzeuge eine Objekt der Klasse *BALL* und prüfe, ob sich die beiden Methoden richtig verhalten. Kann `deltaX` den Wert `0` annehmen? Was würde das bedeuten?

- Füge den beiden neuen Methoden JAVA-Doc-Kommentare hinzu und erzeuge die JAVA-Dokumentation neu.

- Begib dich in die **Klasse *PINGPONG*** und suche in der Methode `tick()` den Fall *„ball schneidet spielerL“*. Füge vor der zweifachen Fallunterscheidung mit den Abstandsberechnungen eine weitere zweifache Fallunterscheidung ein:

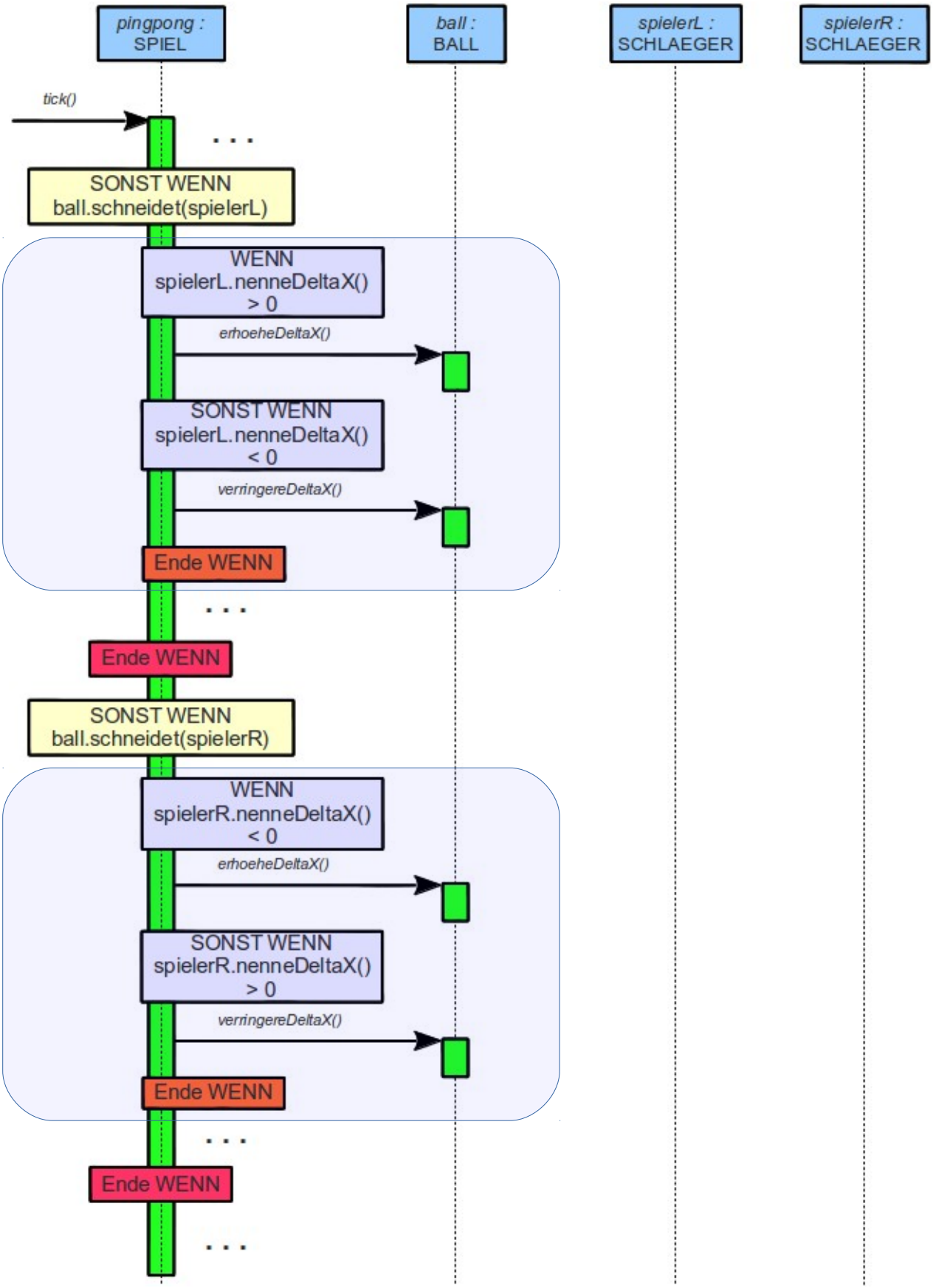
Rückgabe-Wert von spielerL.nenneDeltaX() ist		
> 0	< 0	Sonst
ball: erhoehe deltaX	ball: verringere deltaX	

- Verfahre analog für den Fall *„ball schneidet spielerR“:*

Rückgabe-Wert von spielerR.nenneDeltaX() ist		
< 0	> 0	Sonst
ball: erhoehe deltaX	ball: verringere deltaX	

- Erzeuge ein Objekt der Klasse *PINGPONG* und teste, ob der Ball schneller wird, wenn du ihm mit dem Schläger entgegen kommst. Wird er auch langsamer, wenn du ihm mit dem Schläger davon läufst?

Neues Sequenz-Diagramm der Methode `tick()`



Ergänze deine Kommentare in der Methode `tick(...)`.

Schritt 12

Eine für jeden Computer ohne BlueJ ausführbare Datei erstellen

- Begib dich ganz an das Ende der **Klasse *PINGPONG*** und schreibe dort die *"main"*-Methode, welche später das Spiel starten wird:

```
public static void main(String[] args)
{
    new PINGPONG();
}
```

- Klicke im Hauptmenü von BlueJ auf *Projekt / Als jar-Archiv speichern ...* .

  - Wähle die Klasse *PingPong* als Klasse mit *"main"*-Methode.
  - Speichere die *engine-alpha* als Benutzerbibliothek mit.
    Falls diese nicht angeboten wird, so ist sie im Projekt gespeichert und wird automatisch mitgenommen.
  - Klicke auf *Weiter*.
  - Wähle einen Speicherort und einen Namen.

- Am angegebenen Speicherort ist ein Ordner erstellt worden mit dem von dir angegebenen Namen. Dort findest du nun deine Datei mit der Endung *.jar* und daneben die *engine-alpha*.
  - Mache einen Rechtsklick auf deine jar-Datei, wähle Eigenschaften und mache die Datei ausführbar.
  - Doppelklicke auf die Datei um das Spiel zu starten.
    Sollte das nicht funktionieren, so mache einen Rechtsklick und wähle *öffnen mit JAVA Runtime*.

Anregung für fixe Köpfe:

Versuche doch, den einen Spieler automatisch spielen zu lassen und nicht von einem menschlichen Mitspieler.

Überlege dir Strategien für eine „künstliche Intellilgenz" deines Computer-Gegners ...