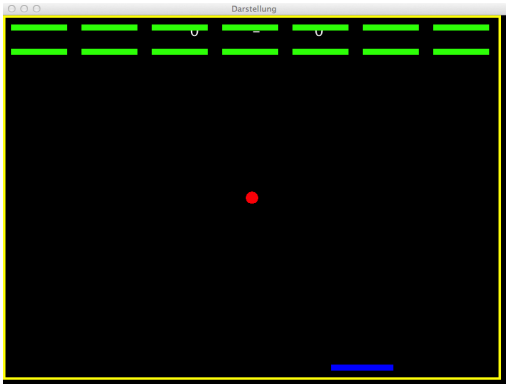
JAVA-Projekt : Das Spiel Breakout



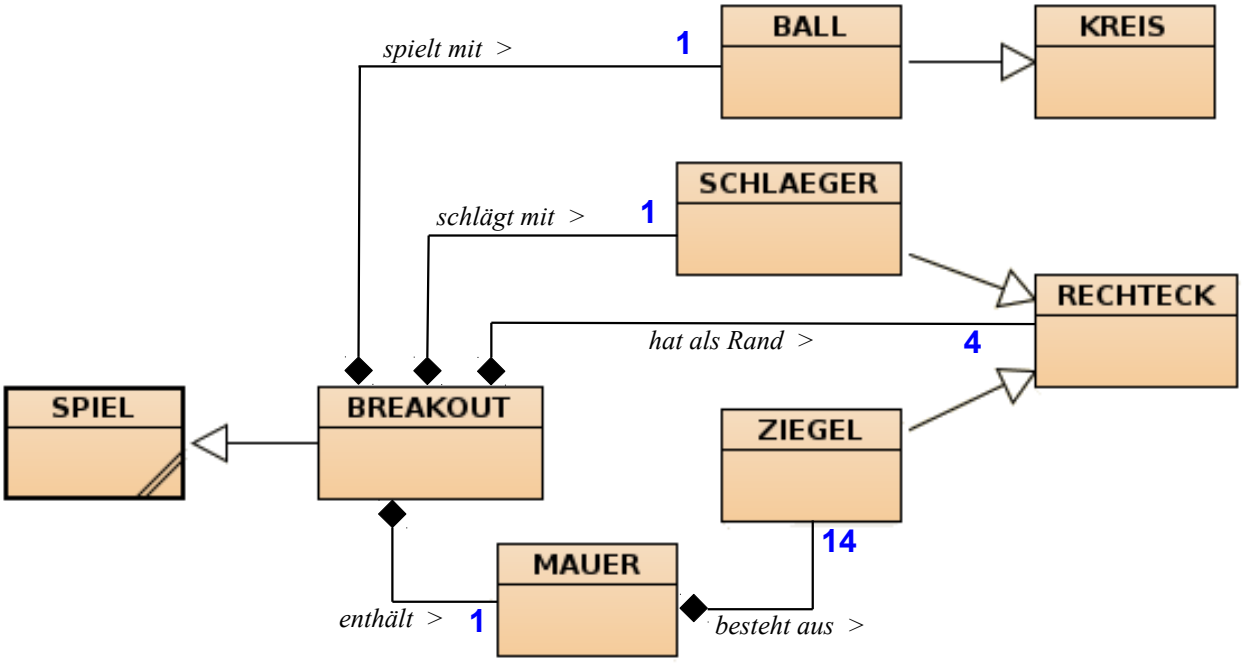
Eines der beliebtesten Spiele der späten 1970er Jahre war das Spiel **Breakout**, das von dem Apple-Mitbegründer Steve Wozniak entwickelt wurde. Das Spielprinzip ist schnell erklärt: Ein Rechteck dient als Schläger, der vom Spieler mit den Pfeiltasten nach rechts und links bewegt werden kann. Ein KREIS dient im Spiel als Ball. Weiterhin gibt es eine Mauer bestehend aus Ziegeln, welche durch Rechtecke dargestellt werden. Ziel des Spiels ist es, alle Ziegel der Mauer mit dem Ball zu treffen, bevor der Ball einmal den unteren Rand berührt. Der Ball wird vom linken, rechten und oberen Rand sowie von den Ziegeln reflektiert.

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, Rückgabe-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
- Felder
  Deklaration und Initialisierung, Zugriff

Klassen-Diagramm



Klassen-Karten

BALL
- deltaX : int - deltaY : int
+ BALL() + bewegen() : void + invertiereDeltaX() : void + invertiereDeltaY() : void + setzeNeueStartbedingen() : void

SCHLAEGER
- deltaX : int
+ SCHLAEGER() + bewegen() : void + erhoeheDeltaX() : void + setzeDeltaX(*deltaXneu : int*) : void + setzeNeueStartbedingen() : void + stopp(*wo : String*) : void + verringereDeltaX() : void

ZIEGEL
- getroffen : boolean
+ ZIEGEL(x : int, y : int) + nenneGetroffen() : boolean + setzeGetroffen(*getroffenNeu : boolean*) : void

MAUER
- anzahlZiegel : int - getroffeneZiegel : int - ziegel : ZIEGEL[ ]
+ MAUER() + testeTrefferMit(*b: BALL*) : void + alleZiegelGetroffen() : boolean + setzeNeueStartbedingen() : void

BREAKOUT
- ball : BALL - mauer : MAUER - randL : RECHTECK - randO : RECHTECK - randR : RECHTECK - randU : RECHTECK - schlaeger : SCHLAEGER
+ BREAKOUT() + tasteReagieren(*code : int*) : void + tick() : void

Schritt 1

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

- Lade dir das BlueJ-Projekt *BreakOut_Vorlage* herunter und öffne es in BlueJ.

- **Erstelle eine Klasse *BALL*, welche von *KREIS* erbt.**

- Welche Attribute und Methoden hast du dadurch geerbt? *(s. Klassenkarte von KREIS)*
  Deklariere nun alle Attribute, die **nicht** von *KREIS* geerbt wurden.
  Hierbei handelt es sich um die Schrittweiten in x- bzw. y-Richtung (`deltaX` und `detlaY`).

- Schreibe einen Konstruktor, welcher alle Attribute so initialisiert, wie es die Objekt-Karte darstellt.
  Arbeite bei Klassen-eigenen Attributen mit direkter Wert-Zuweisung und bei geerbten Attributen mit entsprechenden geerbten Methoden.

ball : BALL
deltaX = 1
deltaY = -2
farbe = rot
radius = 10
M_x = 400
M_y = 300

- Schreibe die Methode `bewegen()`, welche den Ball **um** (deltaX | deltaY) **verschiebt**.
  (s. geerbte Methoden)

- Damit der Ball später z.B. am oberen Rand reflektieren kann, braucht man die Methode `invertiereDeltaY()`, die das Vorzeichen von `deltaY` umkehrt.
  Analog dazu brauchst du eine Methode `invertiereDeltaX()`, damit der Ball am rechten bzw. linken Rand reflektieren kann. Diese Methode soll das Vorzeichen von `deltaX` umkehren.

- *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.

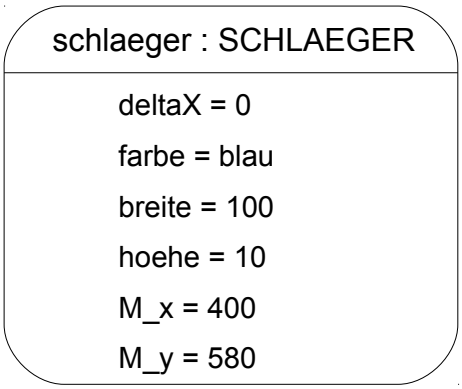
Schritt 2

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

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

- Welche Attribute und Methoden hast du dadurch geerbt? *(s. Klassenkarte von RECHTECK)*
  Deklariere alle Attribute, die **nicht** von *RECHTECK* geerbt werden.
  Hierbei handelt es sich um die Schrittweite in x-Richtung bei der Schläger-Bewegung.
  (`deltaX`)

- Schreibe einen Konstruktor, welcher alle Attribute so initialisiert, wie es die Objekt-Karte darstellt.
  Arbeite bei Klassen-eigenen Attributen mit direkter Wert-Zuweisung und bei geerbten Attributen mit entsprechenden geerbten Methoden.



- Schreibe die Methode `bewegen()`, welche den Schäger **um** (deltaX | 0) **verschiebt**.
  (s. geerbte Methoden)

- Damit der Schläger später über die Tastatur gesteuert werden kann, brauchst du folgende Methoden:
  - `verringereDeltaX()` setzt den Wert von `deltaX` um `1` herab
  - `erhoeheDeltaX()` setzt den Wert von `deltaX` um `1` hinauf
  - `setzeDeltaX(...)`, die einen ganzzahligen Übergabe-Parameter entgegen nimmt und den entsprechenden Attribut-Wert auf den Wert des Übergabe-Parameters setzt.

- Erzeuge ein Objekt der Klasse *SCHLAEGER* 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.

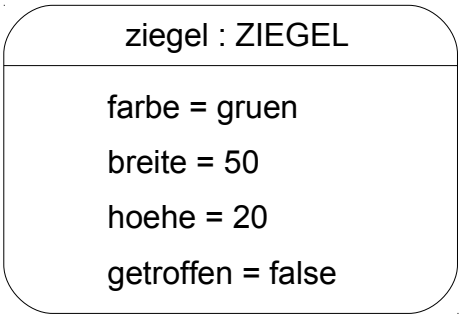
Schritt 3

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

- **Erstelle eine Klasse *ZIEGEL*, die von *RECHTECK* erbt.**

- Welche Attribute und Methoden hast du dadurch geerbt? *(s. Klassenkarte von RECHTECK)*
  Deklariere alle Attribute, die **nicht** von *RECHTECK* geerbt werden.
  Hierbei handelt es sich um ein Attribut, das angibt, ob der Ziegel bereits getroffen wurde oder nicht. Nenne es `getroffen`.

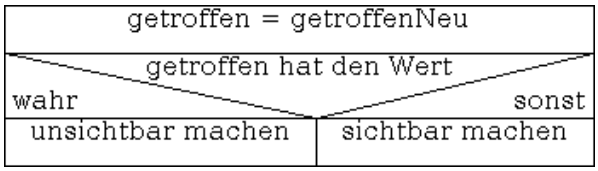
- Schreibe einen Konstruktor, welcher alle Attribute so initialisiert, wie es die Objekt-Karte darstellt.
  Arbeite bei Klassen-eigenen Attributen mit direkter Wert-Zuweisung und bei geerbten Attributen mit entsprechenden geerbten Methoden.



- Ergänze den Konstruktor um zwei ganzzahlige Übergabe-Parameter (x und y), mit deren Hilfe der Mittelpunkt eines Ziegels festgelegt wird.

- Schreibe die Methode `nenneGetroffen()`, die Auskunft darüber gibt, ob der Ziegel bereits getroffen wurde oder nicht.

- Schreibe eine Methode `setzeGetroffen(...)`.
  Die Methode soll zuerst den Wert des Attributs getroffen auf den Wert des Übergabe-Parameters setzen und danach den getroffenen Ziegel unsichtbar machen. Ein nicht getroffener Ziegel wird wieder sichtbar.



- Erzeuge ein Objekt der Klasse *ZIEGEL* und betrachte es im Objekt-Inspektor.
  Sind alle Attribute mit den richtigen Werten belegt?
  Teste nun alle Methoden und prüfe, 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 4

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

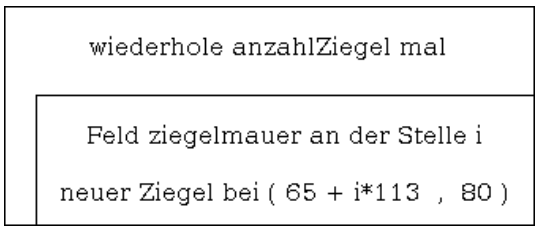
- **Erstelle eine Klasse *MAUER*.**

- Deklariere folgende Attribute:
  - ***anzahlZiegel*** : gibt an, aus wie vielen Ziegeln die Mauer insgesamt besteht.
  - ***getroffeneZiegel*** : Gibt an, wie viele der Ziegel schon getroffen wurden.
  - ***ziegelmauer*** : ein Feld (Array), welches Referenzen auf die einzelnen Ziegel-Objekte speichert.

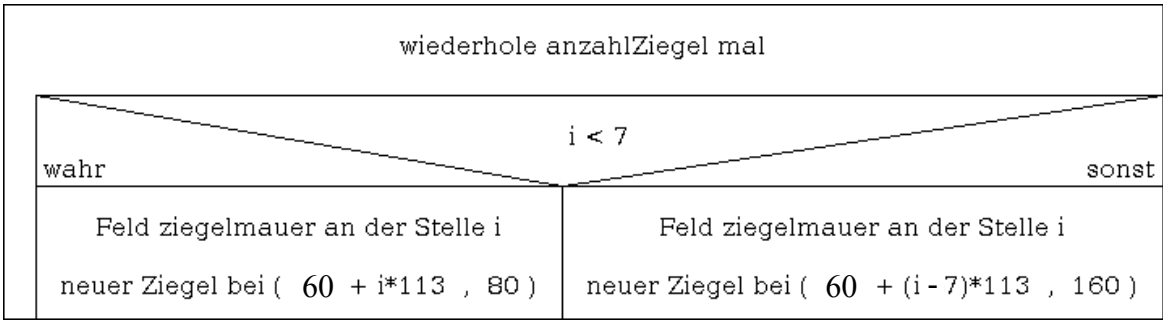
- Schreibe einen Konstruktor, der folgende Attribute initialisiert:
  - die Anzahl der Ziegel auf den Wert *7* (Profi: *14*) setzt.
  - Die Anzahl der getroffenen Ziegel auf *0* setzt.
  - das Feld *ziegelmauer* mit der Größe *anzahlZiegel* initialisiert.

- Ergänze im Konstruktor Anweisungen, welche das Feld *ziegelmauer* mit Ziegel-Objekten befüllen.

für 7 Ziegel



für 14 Ziegel



- Erzeuge ein Objekt der Klasse *MAUER* und prüfe, ob die Attribut-Werte richtig initialisiert wurden.

- 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 5

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

- **Erstelle eine Klasse *BREAKOUT*, die von *SPIEL* erbt.**

- Welche Attribute und Methoden hast du dadurch geerbt? *(s. Klassenkarte von RECHTECK)*
  Deklariere alle Attribute, die **nicht** von *RECHTECK* geerbt werden.
  Die Referenz-Attribute `ball` , `schlaeger` , `mauer` erklären sich von selbst.
  Bei `randL` , `randO` , `randR` , `randU` handelt es sich um die Spielfeldränder.

- Schreibe einen Konstruktor, der folgende Attribute initialisiert:
  - Den Ball
  - Den Schläger
  - Die Mauer
  - Die vier Ränder *(Breite 4 Pixel)*
    Jedes Objekt muss erst erzeugt und danach mit entsprechenden Methoden-Aufrufen eingerichtet werden, bis es die richtige Größe und Lage hat.
  - Setze das Ticker-Intervall auf 10 Millisekunden.

- Überschreibe die Methode `tick()`, die alle 10 ms automatisch aufgerufen wird.
  In ihr soll zunächst nur der Ball und der Schläger bewegt werden.

  (s. Sequenz-Diagramm auf der nächsten Seite)

- Überschreibe die Methode `tasteReagieren(int code)`, die bei jedem Tastatur-Ereignis automatisch aufgerufen wird.

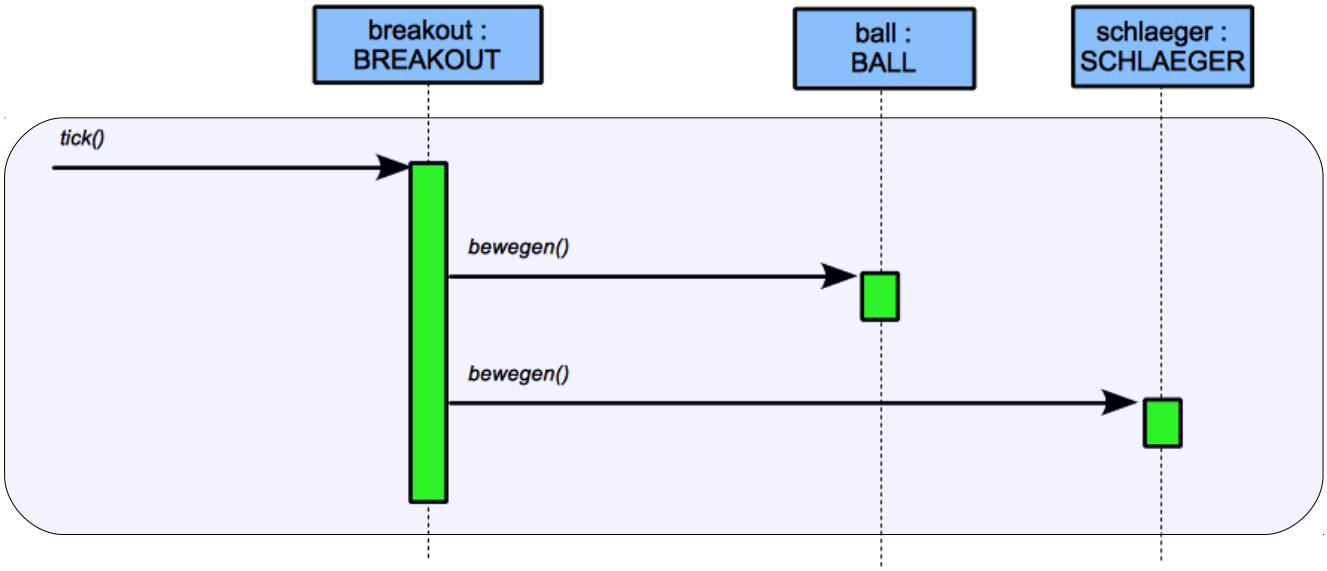
  (s. Sequenz-Diagramm auf der nächsten Seite)

  Der Schläger soll vom Benutzer mit den ***Pfeiltasten*** gesteuert werden.
  - ***Pfeil links*** verringert `deltaX`
  - ***Pfeil rechts*** erhöht `deltaX`.

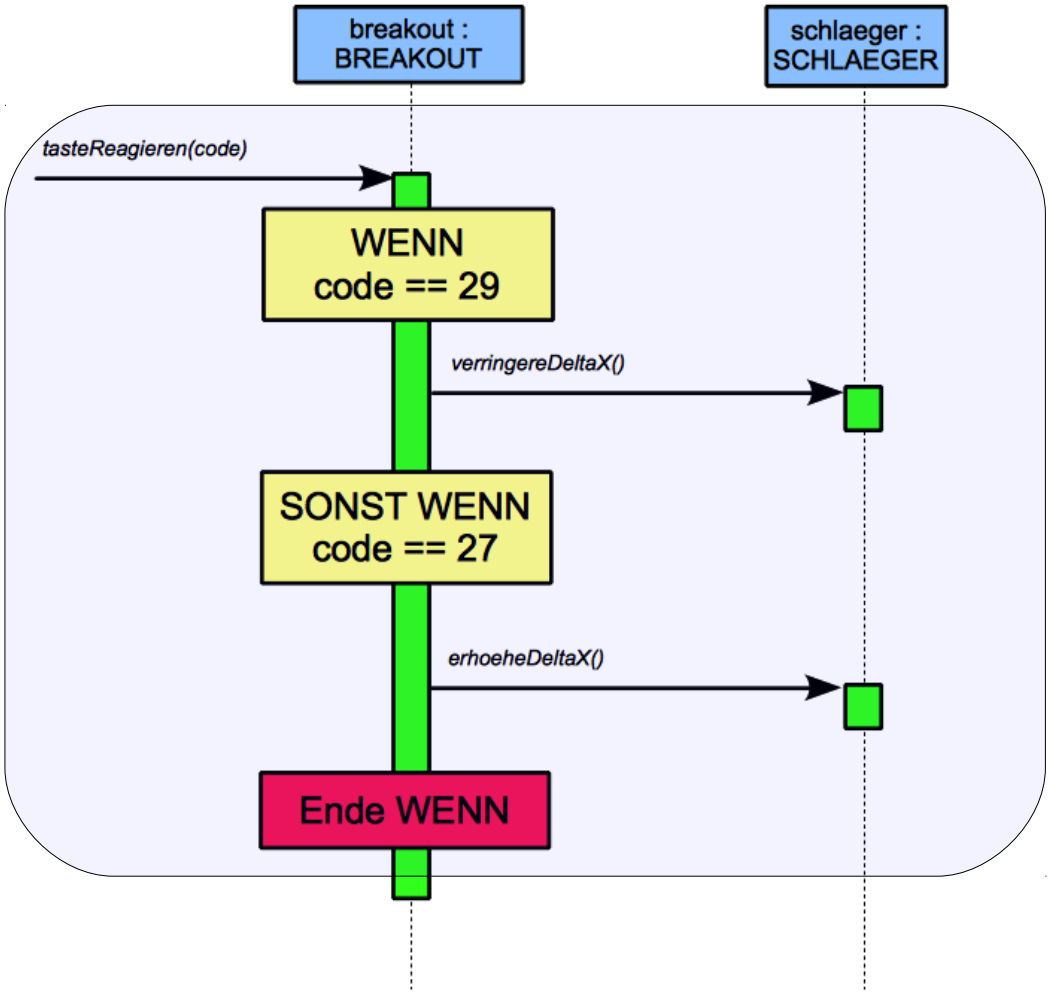
- Erzeuge ein Objekt der Klasse *BREAKOUT* und prüfe, ob sich Ball und Schläger bewegen. Reagiert der Schläger auf die Tasten?
  (Ball und Schläger werden noch das Spielfeld verlassen und nicht aufeinander reagieren.)

- 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()`



Sequenzdiagramm der Methode `tasteReagieren(int code)`



Schritt 6

Hake alle erledigten Arbeiten in den Klassenkarten ab !

Bringe die Schläger dazu, links und rechts 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 `"links"` und `"rechts"` sein.
  Im Rumpf der Methode schreibst du eine zweifache Fallunterscheidung:

Wert von 'wo' ist: links	rechts	Sonst
setze Mittelpunkt auf (55, derzeitiges M_y) setze deltaX auf 0	setzeMittelpunkt auf (745, derzeitiges M_y) setze deltaX auf 0	ø

***Experimentiere mit den Werten** 55 und 745 !*
*Der Schläger soll dabei den linken / rechten **Rand gerade nicht mehr berühren**.*
Im Falle einer Überschneidung wird deine Schläger-Steuerung NICHT funktionieren !!!

- Erzeuge ein Objekt der Klasse ***BREAKOUT*** und ein zusätzliches Objekt der Klasse ***SCHLAEGER*** und prüfe, ob die Methode `stopp(...)` des zweiten Schlägers auf die entsprechenden Übergabe-Werte richtig reagiert.
  Achte ganz genau auf einen kleinen Spalt (1 Pixel) zwischen Rand und Schläger!

- Deine Klassen *BALL* und *SCHLAEGER* haben die Methode `schneidet(...)` geerbt.

Sieh dir diese Methode in den Klassen-Karten und in der JAVA-Dokumentation genau an!
Die beim Übergabe-Parameter erwähnte Klasse 'Raum' ist eine (für uns unsichtbare) Superklasse aller grafischen Objekte innerhalb des Grafik-Fensters. Somit kann jedes sichtbare Objekt auf Überschneidung mit einem anderen sichtbaren Objekt geprüft werden.

Mit Hilfe der Methode `schneidet(...)` kannst du sehr leicht prüfen, ob ein *SCHLAEGER*-Objekt z.B. ein *RECHTECK*-Objekt des Spielfeldrands berührt (oder schneidet).

- Begib dich nun in die **Klasse *BREAKOUT*** und dort in die Methode `tick()`.
  Ergänze unmittelbar vor der Bewegung des Schlägers eine Fallunterscheidung:

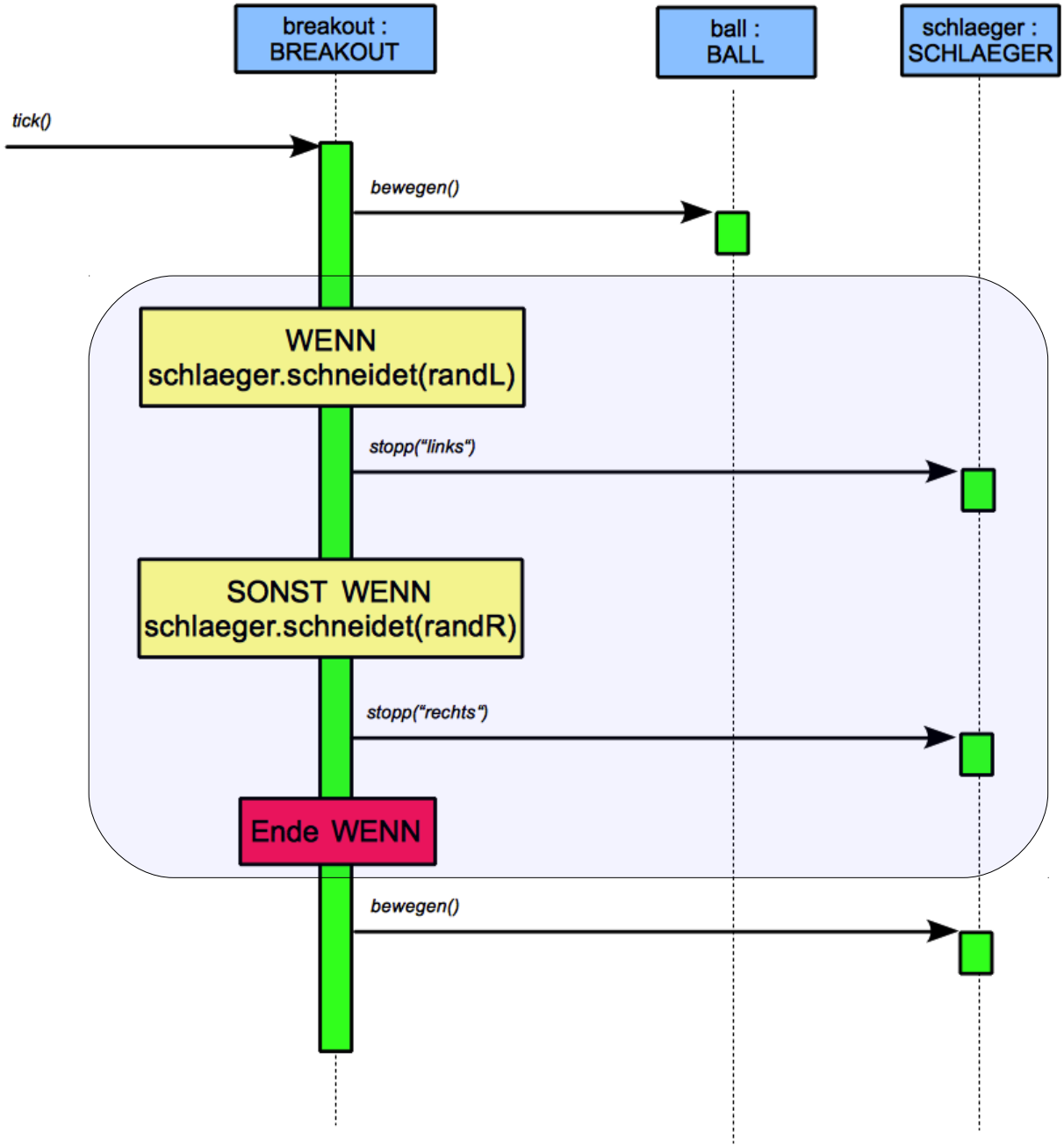
(s. Sequenz-Diagramm auf der nächsten Seite)

Schläger schneidet: linken Rand	rechten Rand
Schläger links stoppen	Schläger rechts stoppen

- Erzeuge nun ein Objekt der Klasse *BREAKOUT* und prüfe, ob der Schläger tatsächlich links und rechts stehen bleibt und ob er sich nach einem entsprechenden Tastendruck wieder bewegt.

- 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.

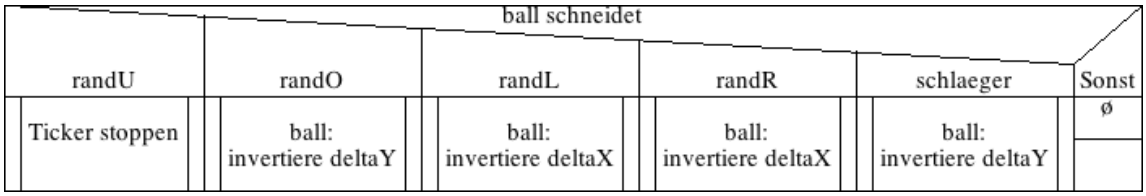
Neues Sequenz-Diagramm der Methode `tick()`



Schritt 7

Bringe den Ball dazu, an den Spielfeldrändern sowie am Schläger zu reflektieren.

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



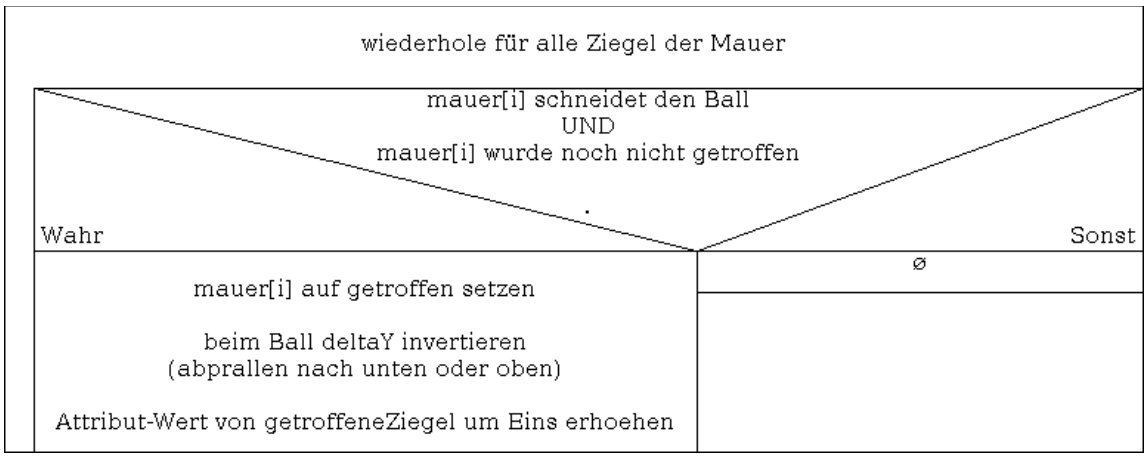
- Erzeuge nun ein Objekt der Klasse *BREAKOUT* und prüfe, ob der Ball am linken, rechten und obereren Rand sowie am Schläger abprallt.

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

  Du wirst noch so manches darin ergänzen … :-)

Bringe die Ziegel der Mauer dazu, auf den Ball zu reagieren.

- Begib dich nun in die **Klasse *MAUER***. Wir brauchen hier eine Methode `testeTrefferMit(Ball b)`, die jeden Ziegel der Mauer auf Kollision mit dem Ball überprüft. Wird ein Ziegel (der noch nicht getroffen wurde) nun getroffen, so soll das im Attribut `getroffen` dieses Ziegels gespeichert werden und der Ziegel soll ab nun unsichtbar sein und damit nicht auf weitere Treffer reagieren. Außerdem soll im Falle eines Treffers der Ball am Ziegel reflektiert werden.

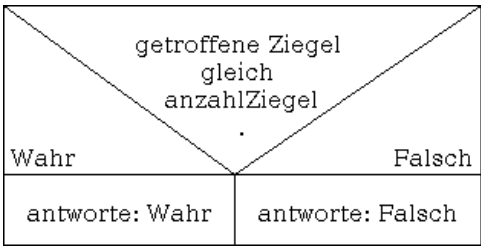


- Begib dich nun zurück in die **Klasse *BREAKOUT*** und dort in die Methode `tick()`. Rufe dort als letzten Befehl die neue Methode des Mauer-Objekts auf.

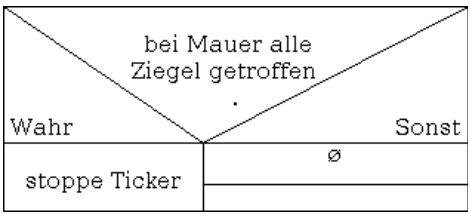
- Erzeuge wieder ein Objekt der Klasse *BREAKOUT* und prüfe, ob die Ziegel nun auf den Ball wie gewünscht reagieren.

Das Spiel beenden, wenn der Spieler alle Ziegel getroffen hat.

- Begib dich nochmal in die **Klasse *MAUER*** und schreibe dort eine Methode `alleZiegelGetroffen()` welche einen `boolean` als Antwort zurück liefert.
  Die Antwort soll `true` sein, wenn die Anzahl der getroffenen Ziegel gleich der Anzahl der vorhandenen Ziegel ist sonst lautet die Antwort `false`.



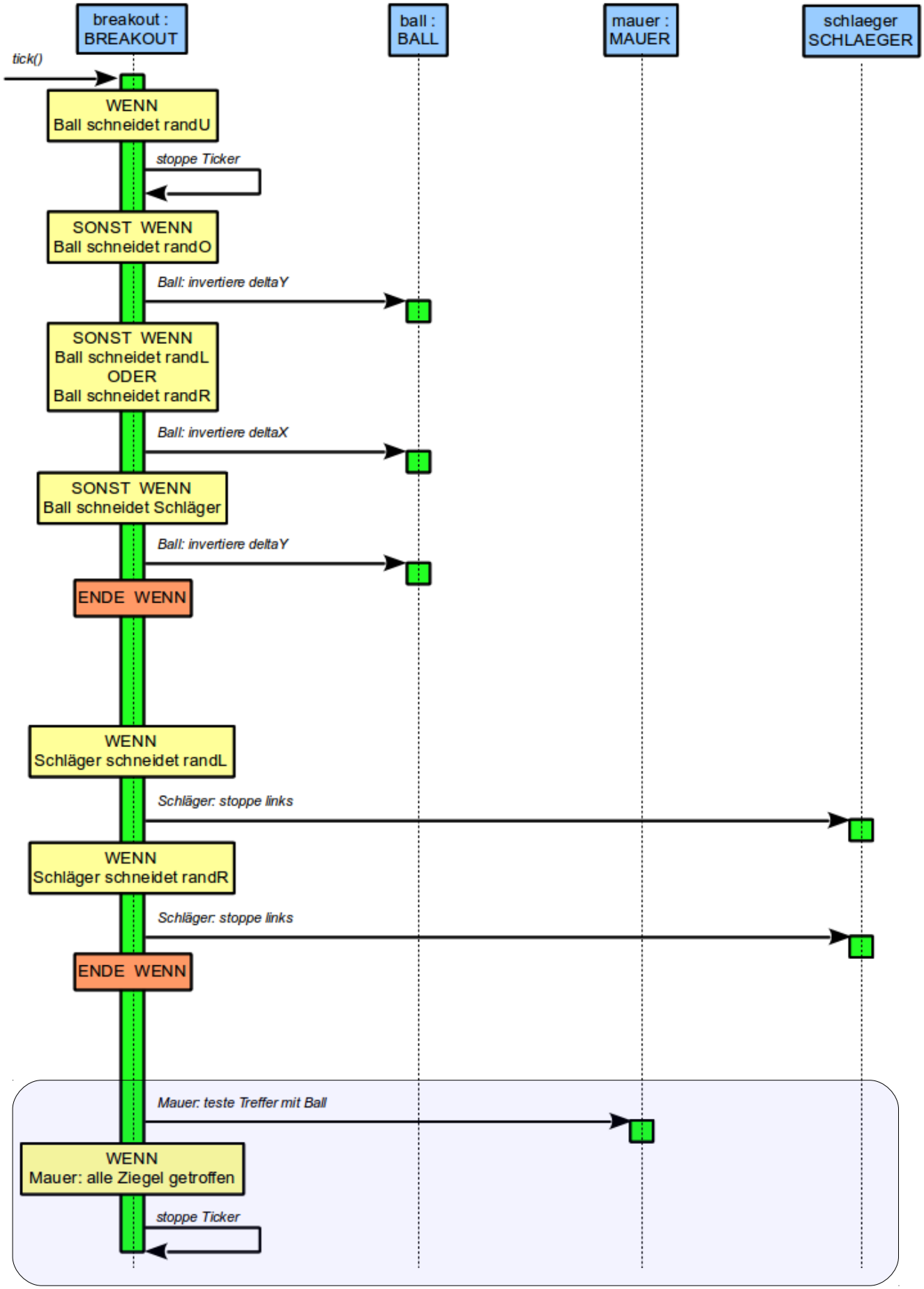
- Begib dich nun in die **Klasse *BREAKOUT*** und dort an das Ende der Methode `tick()` - direkt unterhalb des Treffer-Tests der Wand mit dem Ball.
  Füge hier eine bedingte Anweisung ein, die den (geerbten) Ticker stoppt, wenn alle Ziegel der Mauer getroffen sind.



Das gesamte Sequenz-Diagramm für die Methode `tick()` findest du auf der nachfolgenden Seite.

- Erzeuge wieder ein Objekt der Klasse *BREAKOUT* und prüfe, ob das Spiel nach dem Treffen des letzten Ziegels anhält.

Neues Sequenz-Diagramm der Methode tick()



Schritt 8

Mit Taste N ein neues Spiel beginnen

Zum Beginnen eines neuen Spiels müssen Ball und Schläger wieder in ihre Startpositionen gebracht und die Mauer neu aufgebaut werden. Weiterhin 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:

```
if (this.deltaX != 0) {
   this.deltaX = this.deltaX / Math.abs(this.deltaX);
}
else {
   this.deltaX = 1;
}
if (this.deltaY != 0) {
   this.deltaY = 2 * this.deltaY / Math.abs(this.deltaY);
}
else {
   this.deltaY = -2;
}
```

- 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.

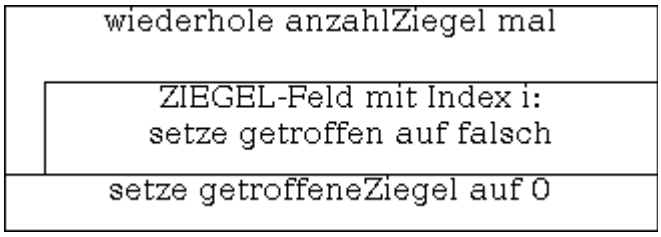
- Füge einen Java-Doc-Kommentar zu der neuen Methode hinzu und erzeuge die JAVA-Dokumentarion neu.

- Begib dich in die **Klasse *SCHLAEGER*** und schreibe dort eine neue Methode `setzeNeueStartbedingungen()`.
  - Setze den Mittelpunkt auf (400|580).
  - Setze das Attribut `deltaX` auf den Wert `0`.

- Erzeuge ein Objekt der Klasse *SCHLAEGER* und verschiebe es an eine beliebige Stelle. Teste nun die Methode `setzeNeueStartbedingungen()`.

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

- Begib dich in die **Klasse *MAUER*** und schreibe dort eine neue Methode `setzeNeueStartbedingungen()` unter Verwendung des nachfolgenden Struktogramms.

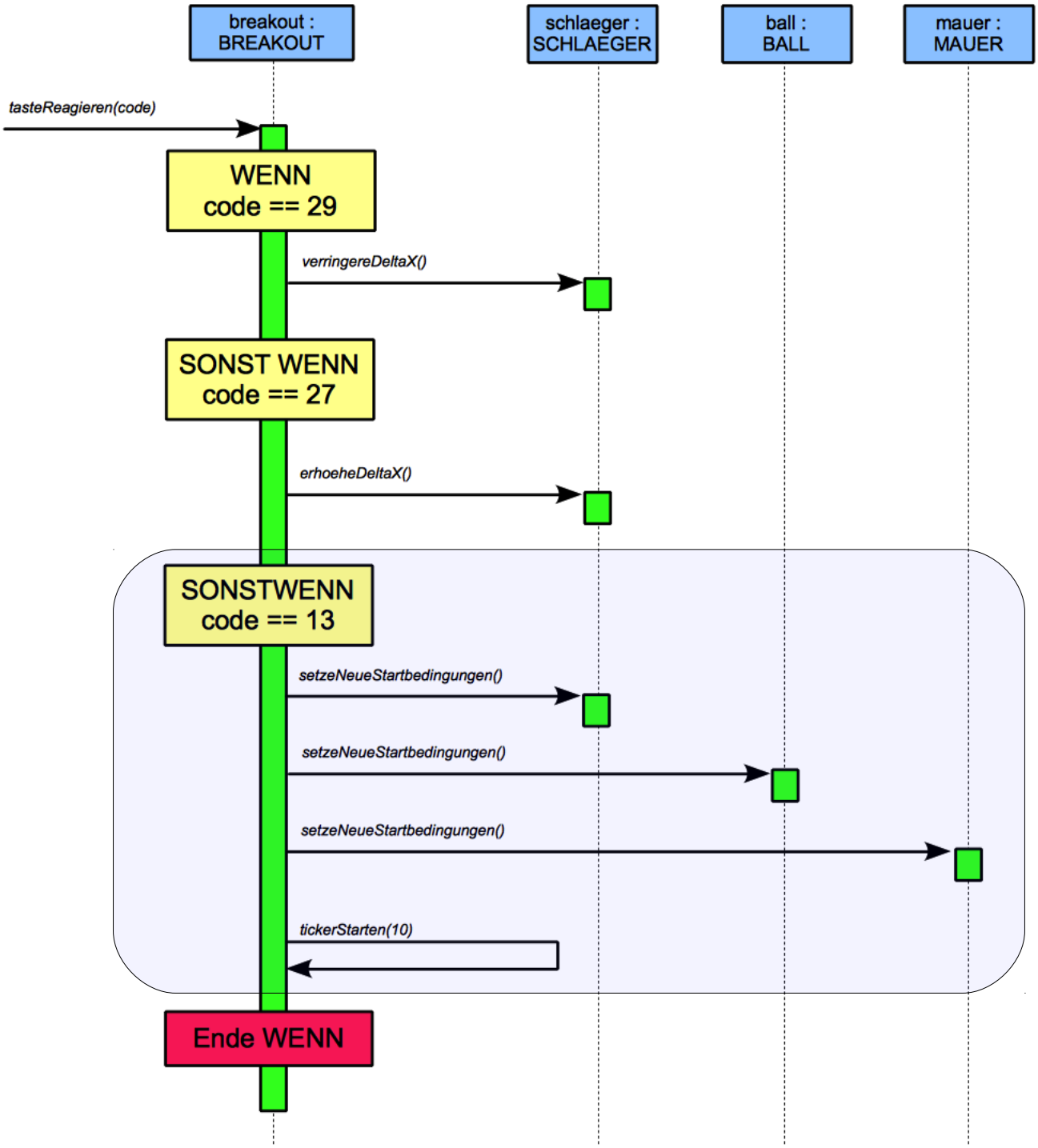


- Nun begibst du dich noch in die **Klasse *BREAKOUT*** und dort in die Methode `tasteReagieren(int code)`. Ergänze die Mehrfachfallunterscheidung um einen weiteren Fall:

Wert von code ist			
27	29	13	Sonst
...	...	schlaeger: setze neue Startbedingungen ball: setze neue Startbedingungen mauer: setze neue Startbedingungen breakout: Ticker mit 10ms neu starten	ø

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

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



Anregungen für weitere Spielvarianten

1. **Rahmen-Abpraller vom Schläger**

   Sowohl Schläger als auch Ball haben eine Methode `berechneAbstandX(...)` geerbt.
   Damit lässt sich sehr einfach der Abstand der Mittelpunkte in x-Richtung zweier Grafik-Objekte bestimmen.
   Wenn nun der Ball den Schläger sehr weit rechts außen trifft, so soll der Ball sein `deltaX` um Eins vergrößern. Wenn er den Schläger sehr weit links außen trifft, soll er sein `deltaX` um Eins verringern.
   Dazu braucht der Ball eine Methode `erhoheheDeltaX()` und `verringereDeltaX()`.

2. **Spielfeld vergrößern, mehr Ziegel, Punkteanzeige**

   Die Klasse SPIEL verfügt über einen zweiten Konstruktor, mit dem man die Spielfeld-Größe verändern und Punktestände einblenden kann. Sieh die diesen Konstruktor in der JAVA-Docu einmal genau an. (Den Parameter für die Maus belgst du einfach mit `false`)
   Vergrößere das Spielfeld, erzeuge mehr Ziegel – vielleicht auch mehr als zwei Reihen …
   Lasse im Linken Punktestand die getroffenen Ziegel und im rechten Punktestand die Schläge mitzählen.

3. **Interessantere Ziegel**

   Du kannst die Ziegel auch durch die Klasse BILD darstellen lassen anstatt durch schlichte Rechtecke.
   Sei kreativ, es müssen ja nicht unbedingt Ziegel sein. Wie wäre es mit Gesichtern, Tieren, Aliens …