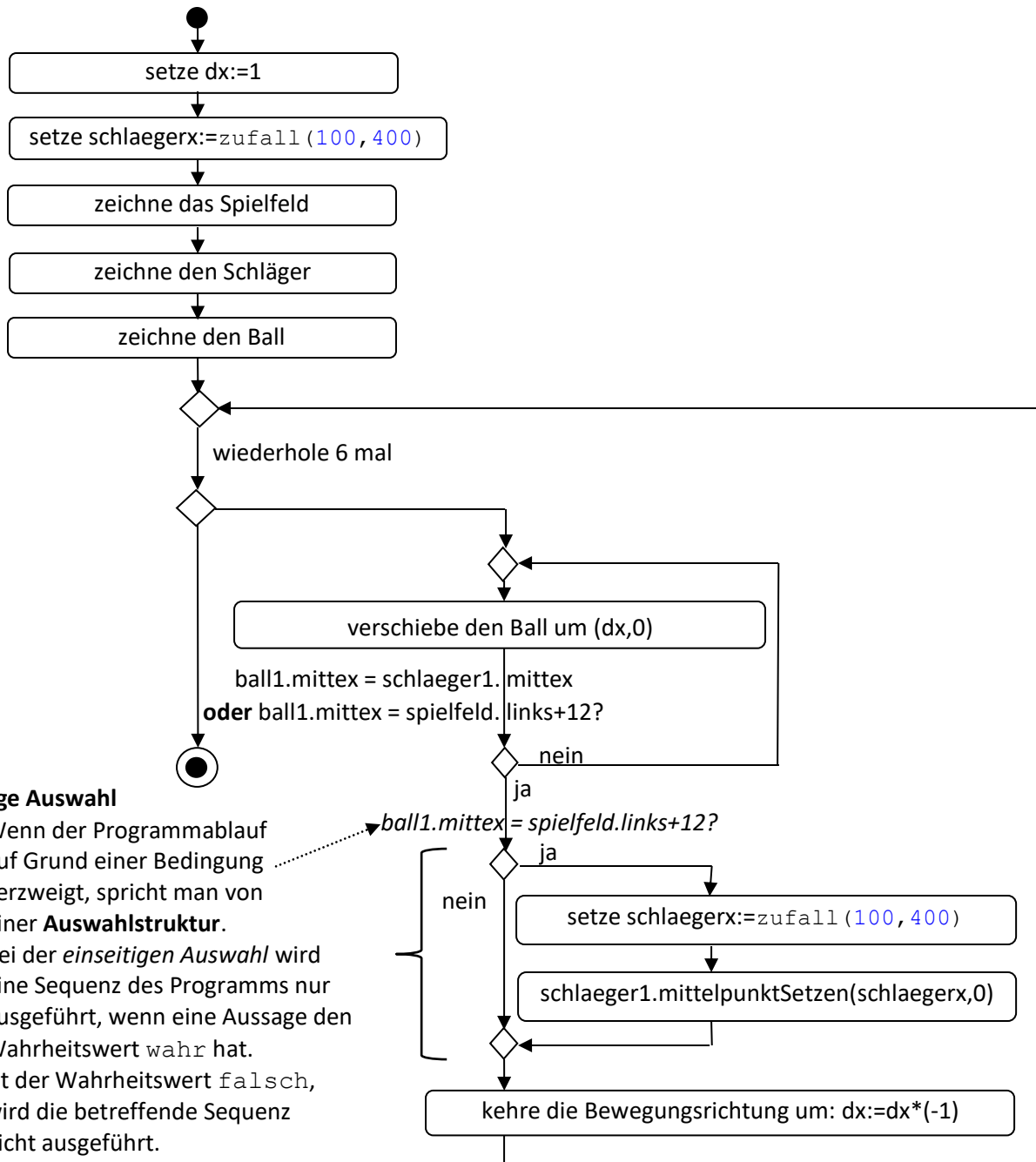




Die algorithmische Grundstruktur *Auswahl*

1. Im Programm *ballspiel3.eos* wird der Schläger nur einmal zu Beginn an eine zufällige Position gesetzt. Interessanter wäre es, wenn der Schläger immer wieder neu gesetzt würde, **wenn** der Ball links an der Bande abprallt.
 - Formuliere in dem Aktivitätsdiagramm unten die Bedingung dafür.



Einseitige Auswahl

- Wenn der Programmablauf auf Grund einer Bedingung verzweigt, spricht man von einer **Auswahlstruktur**. Bei der *einseitigen Auswahl* wird eine Sequenz des Programms nur ausgeführt, wenn eine Aussage den Wahrheitswert *wahr* hat. Ist der Wahrheitswert *falsch*, wird die betreffende Sequenz nicht ausgeführt.

Schreibweise in EOS:

```
wenn <Bedingung> dann
    <Anweisungen>
*wenn
```

- Optimierte die Programmversion *ballspiel3.eos*, indem der Schläger immer wieder neu gesetzt wird, wenn der Ball links an der Bande abprallt (*ballspiel4.eos*). (Vorlagedatei: *v06-ballspiel3.eos*)
vgl. `.\261-materialien\ballon\09-ballspiel4.eos`



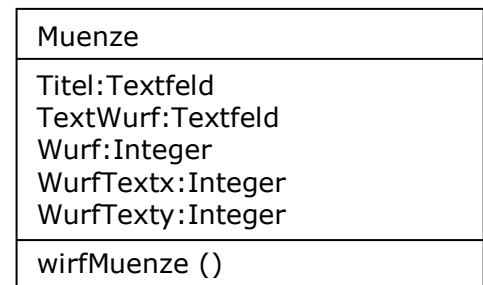
2.6.1 Modellieren und Codieren von Algorithmen

2. Ein weiteres Beispiel für einen Zufallsversuch ist das Werfen einer Münze. Hier können zwei Ereignisse eintreten: „Kopf“ oder „Zahl“.
Das Entwickeln eines Programms, mit dem das Werfen einer Münze simuliert und Ergebnisse als Text („Kopf“ oder „Zahl“) ausgegeben werden, ist in EOS mit Hilfe der Klasse *Textfeld* möglich.

Um das Werfen einer Münze zu simulieren, wird eine Zufallszahl von 1 bis 2 erzeugt und in der Variablen *Wurf* gespeichert. Für die Zahl 1 soll der Text 'Kopf' angezeigt werden, für die Zahl 2 der Text 'Zahl'.

In die Variablen *WurfTextx* und *WurfTexty* werden die x- bzw. y-Koordinaten des Textfelds eingetragen.

- Ergänze das Aktivitätsdiagramm.



Klassendiagramm

methode wirfMuenze

Wurf:=zufall(1,2)

wenn *Wurf*=1 **dann**

Text_Wurf.zeileHinzufügen('Kopf')

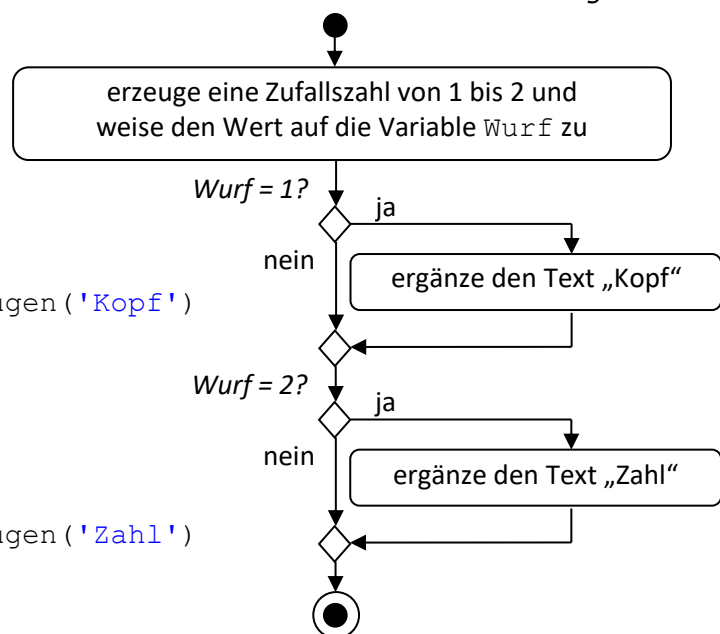
***wenn**

wenn *Wurf*=2 **dann**

Text_Wurf.zeileHinzufügen('Zahl')

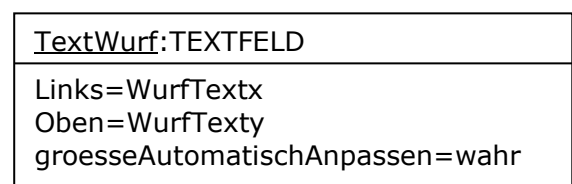
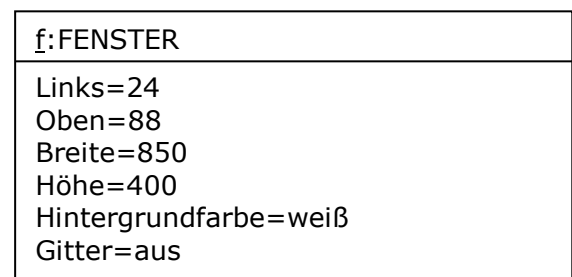
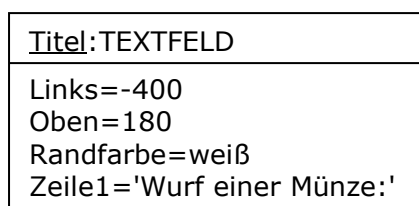
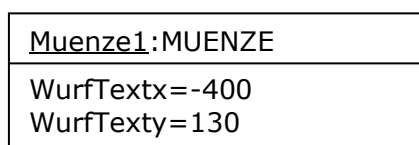
***wenn**

ende

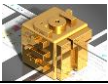


Aktivitätsdiagramm *wirfMuenze*

- Codiere die Aufgabenstellung in EOS.
Beachte die Objektdiagramme!
Speichere das Programm als *muenze1.eos*.
vgl. 261-materialien\muenze\01-muenze1.eos



- Teste das Programm, indem du es mehrmals ausführst und die Ergebnisse kontrollierst.
Dabei müssen die Ereignisse in einer zufälligen Abfolge eintreten.



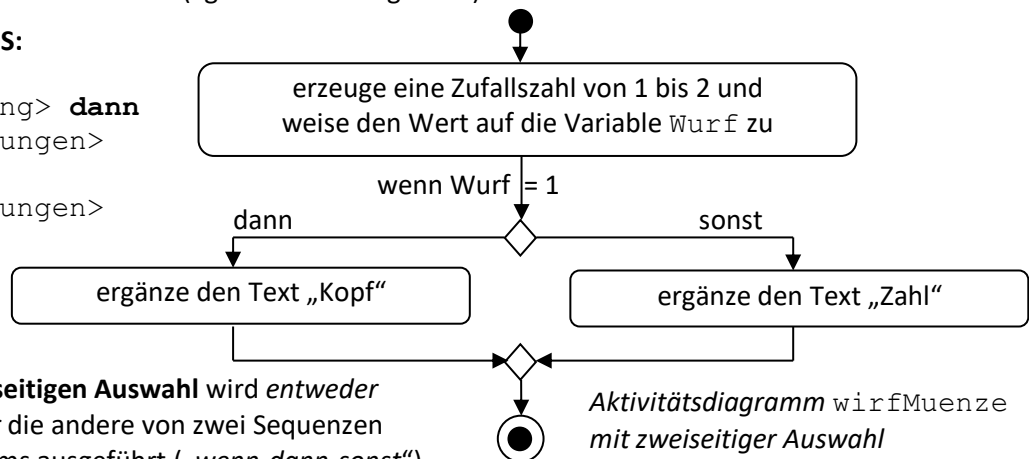
Zweiseitige Auswahl

Da hier genau zwei Möglichkeiten bestehen ('Kopf' oder 'Zahl'), kann die Auswahlstruktur auch vereinfacht werden: Wenn das Ergebnis nicht 'Kopf' ist, dann kann es ja nur 'Zahl' sein bzw.:

„Wenn 1 dann 'Kopf' sonst 'Zahl'.“ (vgl. Aktivitätsdiagramm)

Schreibweise in EOS:

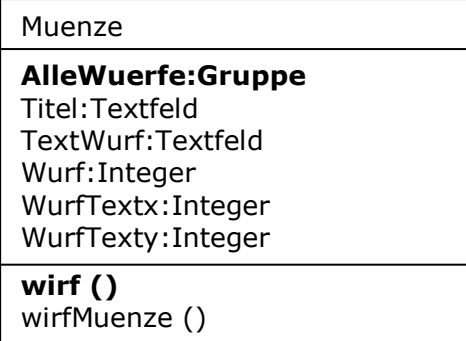
```
...  
wenn <Bedingung> dann  
    <Anweisungen>  
sonst  
    <Anweisungen>  
*wenn  
...
```



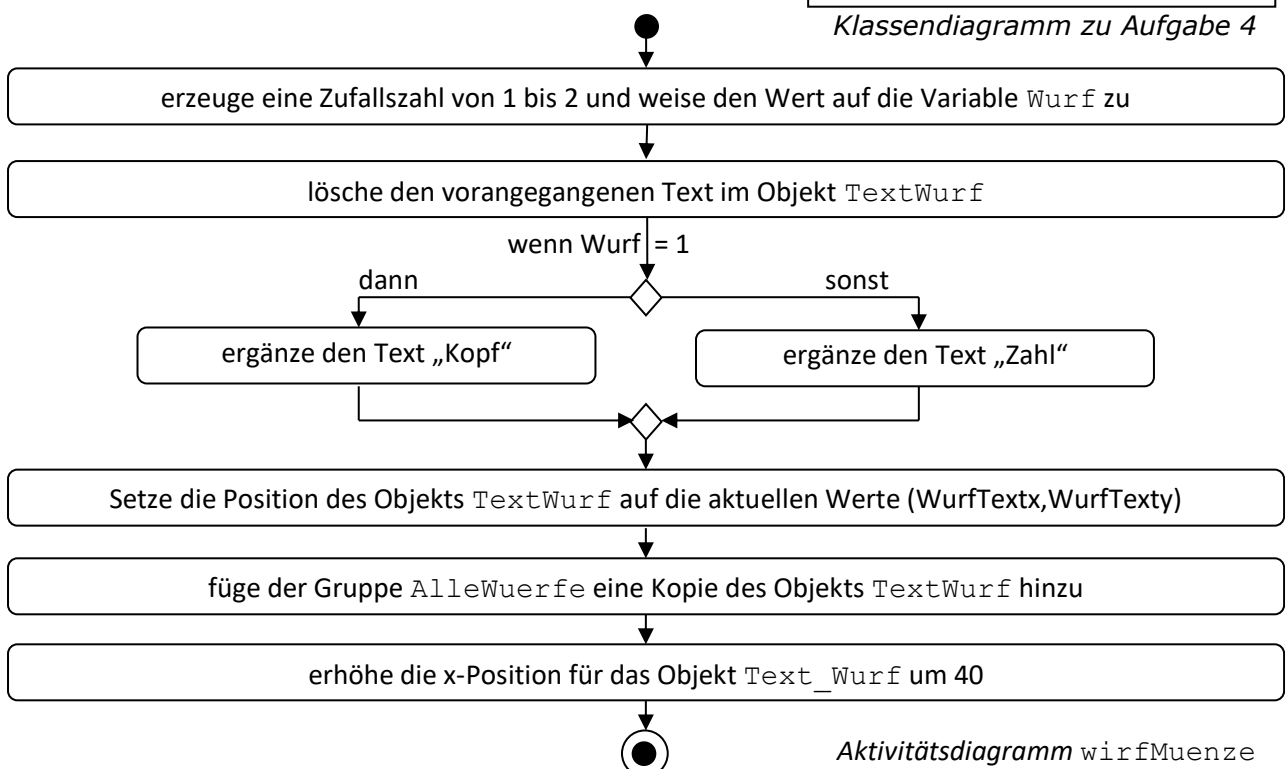
- Bei der **zweiseitigen Auswahl** wird *entweder* die eine *oder* die andere von zwei Sequenzen des Programms ausgeführt („wenn-dann-sonst“).

3. Ändere dein Programm *muenze1.eos* unter Verwendung der zweiseitigen Auswahl (*muenze2.eos*). (Vorlagedatei: *v07-muenze1.eos*); vgl. *261-materialien\muenze\02-muenze2.eos*

4. Ergänze dein EOS-Programm: Es sollen 20 Würfe ausgeführt werden, so dass eine Zeile des Fensters angefüllt wird. Erstelle dafür die Methode *wirf()* (*muenze3.eos*). vgl. *261-materialien\muenze\03-muenze3.eos*
Hinweis: Das Textfeld *TextWurf* kann wiederholt verwendet werden, wenn der jeweilige Wurf nicht direkt angezeigt wird, sondern in eine Gruppe, z. B. *AlleWuerfe*, kopiert wird:



Klassendiagramm zu Aufgabe 4





- Hinweis: Hier sind mehrere Zeilen erforderlich, um die Ergebnisse anzeigen zu können.
Das kann z. B. mit einer geschachtelten Wiederholung umgesetzt werden.

```
graph TD
    Start(( )) --> Merge1{ }
    Merge1 --> LoopHeader{ }
    LoopHeader -- "wiederhole 5 mal" --> Merge2{ }
    Merge2 --> LoopBody{ }
    LoopBody -- "wiederhole 20 mal" --> LoopBody
    LoopBody --> ThrowMuenze[wirfMuenze()]
    ThrowMuenze --> LoopBody
    LoopBody --> SetX[setze die x-Position für das Objekt  
Text_Wurf auf -400 (ganz links)]
    SetX --> DecY[verringere die y-Position für das  
Objekt TextWurf um 30 (nach unten)]
    DecY --> Merge2
    DecY --> End((( )))
```

The diagram illustrates the 'wirf' (throw) activity. It begins with a start node leading to a merge node. From there, it enters a loop labeled 'wiederhole 5 mal'. Inside this loop, there is another merge node leading to a loop body. The loop body contains a sub-loop labeled 'wiederhole 20 mal'. Within this sub-loop, the action 'wirfMuenze()' is performed. After the sub-loop, the action 'setze die x-Position für das Objekt Text_Wurf auf -400 (ganz links)' is executed, followed by 'verringere die y-Position für das Objekt TextWurf um 30 (nach unten)'. The loop body then merges back to the 'wiederhole 5 mal' loop. Finally, the activity ends at an end node.

-