OOP

Eine Klasse Basteln

Henrik Horstmann

27. November 2015

Inhaltsverzeichnis

1 Bedeutung der Symbole....1
2 Mehrere Würfel in einem Spiel....2
3 Die Klasse Würfel....3
3.1 Die Attribute....3
3.2 Der Konstruktor....3
3.3 Die Methoden....3
3.4 Das UML-Diagramm....4
4 Umsetzen in Programmcode....5
4.1 Anlegen einer neuen Klasse....5
4.2 Attribute deklarieren....6
4.3 Parameterliste des Konstruktors bearbeiten....7
4.4 Programmcode im Konstruktor....8
4.5 Die Methode wuerfeln....9
4.6 Die Methode letzteAugenzahl....13
5 Das "Rahmen"-Programm....14
6 Übungen....16
6.1 Ampel....16
6.2 Ampelschaltung....19
6.3 Fußgängerampel....20
6.4 Hampelmann....22
7 Lösungen zu den Aufgaben....26
7.1 Lösung zu Aufgabe 4....26
7.2 Lösung zu Aufgabe 5....27
7.3 Lösung zu Aufgabe 6....28
7.4 Lösung zu Aufgabe 8....29
7.5 Lösung zu Aufgabe 9....30
7.6 Lösung zu Aufgabe 11....31
7.7 Lösung zu Aufgabe 12.1....32
7.8 Lösung zu Aufgabe 12.2....33
7.9 Lösung zu Aufgabe 12.3....34

7.10 Lösung zu Aufgabe 12.4.....35
7.11 Lösung zu Aufgabe 12.5.....36
7.12 Lösung zu Aufgabe 12.6.....37

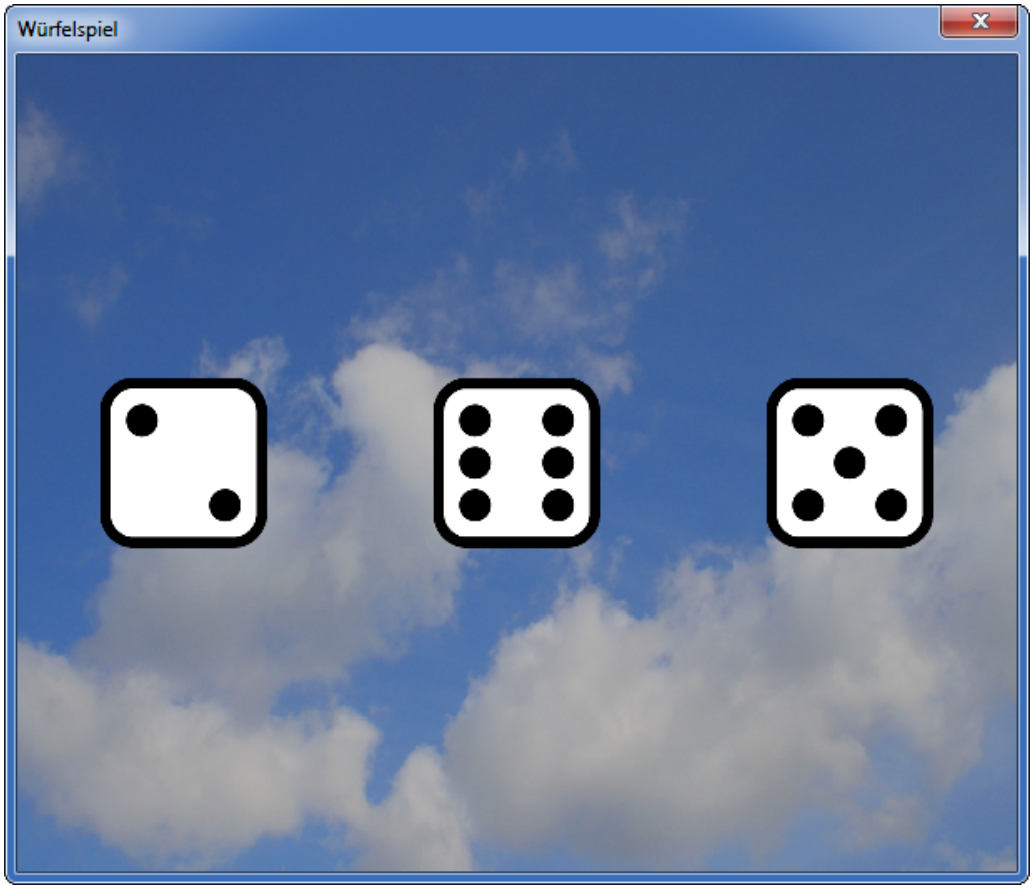
1 Bedeutung der Symbole

Symbol	Beschreibung
	Dieses Symbol kennzeichnet einen Tipp oder Hinweis.
	Aufgepasst, hier passieren leicht Fehler oder es ist mit Schwierigkeiten zu rechnen.
	Hier erfahren Sie, wie es gemacht wird.
	Es folgt eine Aufgabe die zu lösen ist.

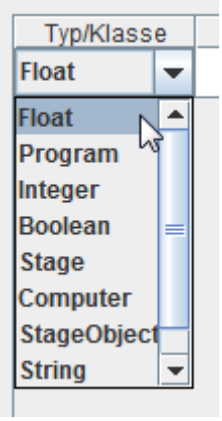
© 2015 Henrik Horstmann

2 Mehrere Würfel in einem Spiel

Stellen Sie sich folgendes Glücksspiel vor: Es wird nacheinander mit drei Würfeln gewürfelt. Ihr Punktestand ergibt sich aus der gewürfelten Augensumme.



Um dieses Spiel zu programmieren, wäre es schön eine Klasse zu haben, die einen solchen Würfel repräsentiert. Dann könnten drei Objekte dieser Klasse erzeugt werden und es müssten nur noch die Methoden zum würfeln und auslesen der Augenzahlen aufgerufen werden. Das Programm ließe sich schnell realisieren.



Ein Blick in die Liste der Standard mäßig zur Verfügung stehenden Klassen zeigt, dass es leider keine Klasse Würfel gibt.

Glücklicherweise gibt es die Möglichkeit selbst Klassen zu erstellen und das soll in diesem Tutorial gezeigt werden.

3 Die Klasse Würfel

Um das Spiel programmieren zu können werden Würfelobjekte benötigt, die Aktionen zum würfeln und Auslesen der Augenzahl haben. Diese Objekte sollen durch eine Klasse *Wuerfel* beschrieben werden.

3.1 Die Attribute

Der Würfel muss die Möglichkeit bekommen Zufallszahlen zwischen 1 und 6 zu erzeugen. Dazu benötigt er ein Objekt der Klasse *Computer*.

Zur Darstellung des Würfels auf der "Bühne" wird ein Objekt der Klasse *StageObject* benötigt.

Und schließlich muss der Würfel auch noch wissen, welche Augenzahl gerade gewürfelt wurde, dazu wird ein Objekt der Klasse *Integer* benötigt.

Es werden somit drei Attribute benötigt:

Computer *comp;*

StageObject *wuerfelanzeige;*

Integer *augenzahl;*

3.2 Der Konstruktor

Jede Klasse hat mindestens einen Konstruktor. Ein Konstruktor ist eine Methode, die genau einmal aufgerufen wird und zwar genau dann, wenn ein Objekt der Klasse erzeugt wird.

Der Konstruktor dient dazu ein neu geschaffenes Objekt mit Initialwerte (=Startwerte für die Attribute) zu versehen.

Der Konstruktor der Klasse *Wuerfle* soll dazu folgende Parameter erhalten:

StageObject *stage*	Die "Bühne", auf welcher der Würfel platziert werden soll.
Integer *xp*	x-Koordinate des Würfels auf der "Bühne".
Integer *yp*	y-Koordinate des Würfels auf der "Bühne".
Integer *scalefactor*	Skallierung (=Vergrößerung) des Würfels.

3.3 Die Methoden

Würfeln

Es wird eine Methode zum Würfeln benötigt. Sie macht nichts anderes, als den eine Zufallszahl zu erzeugen und die entsprechende Würfelanzeige auf der "Bühne" erscheinen zu lassen. Man könnte auch sagen, dass es sich um einen Befehl handelt.

© 2015 Henrik Horstmann

letzte Augenzahl

Bei der Methode "letzte Augenzahl" handelt es sich um eine Funktion. Nach dem Aufruf der Methode erwarte ich einen Rückgabewert.

3.4 Das UML-Diagramm

Im UML-Diagramm sieht die Klasse *Wuerfel* wie folgt aus:

Wuerfel
comp:Computer wuerfelanzeige:StageObject augenzahl:Integer
Wuerfel(stage:StageObject, xp:Integer, yp:Integer, scalefactor:Integer) wuerfeln() letzteAugenzahl():Integer

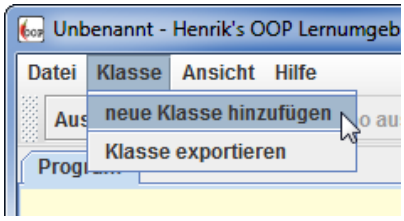
© 2015 Henrik Horstmann

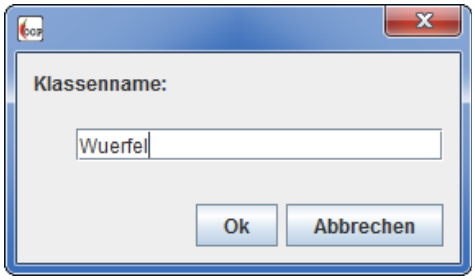
4 Umsetzen in Programmcode

4.1 Anlegen einer neuen Klasse

Zunächst muss eine neue Klasse angelegt werden. Dies geschieht über das Menü *Klasse→neue Klasse hinzufügen*.

Es erscheint ein Dialogfenster, in dem der Name der Klasse eingegeben werden muss.





Hinweis:
Klassenamen müssen mit einem Großbuchstaben beginnen und dürfen keine Umlaute, ß und Sonderzeichen enthalten.

Nach dem der Klassenname mit *Ok* bestätigt wurde, erscheint in der Entwicklungsumgebung ein neuer Reiter für die neue Klasse:

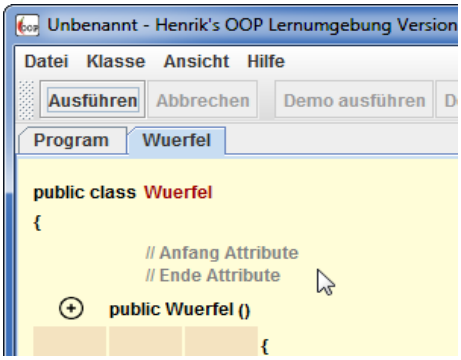
Aufgabe 1:
Führen Sie die oben beschriebenen Schritte in einem neuen Projekt aus und erstellen Sie eine Klasse *Wuerfel*. Speichern Sie das neue Projekt unter dem Namen *Wuerfelspiel.oop*.

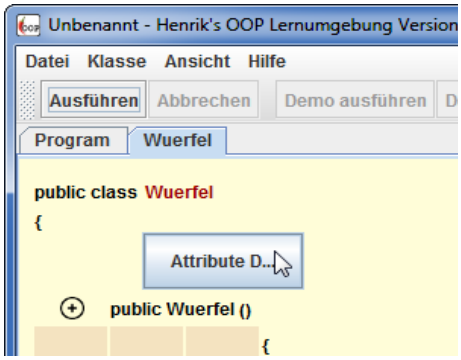
4.2 Attribute deklarieren

Um die drei Attribute zu deklarieren muss der Mauszeiger auf den Bereich mit den Kommentaren

```
// Anfang Attribute
// Ende Attribute
```

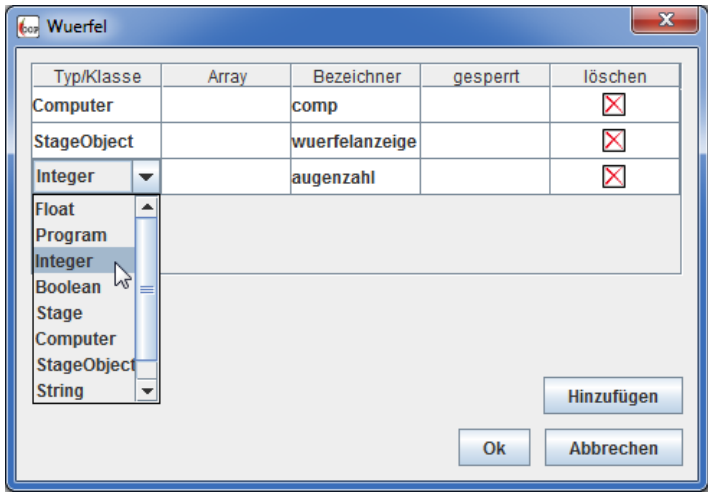
platziert werden. Es erscheint ein Button.





Ein Mausklick auf den Button öffnet ein Dialogfenster zum deklarieren der Attribute.

Sind alle Attribute deklariert, so wird die Eingabe mit einem Mausklick auf *Ok* beendet.



Hinweis:
Attributnamen müssen mit einem Kleinbuchstaben beginnen und dürfen eine Umlaute, ß und Sonderzeichen enthalten.

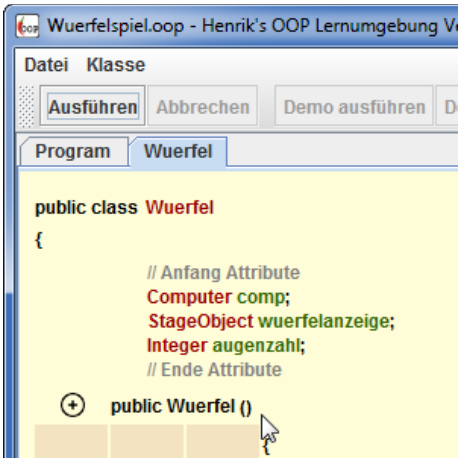
Aufgabe 2:
Führen Sie die oben beschriebenen Schritte im Projekt *Wuerfelspiel.oop* aus.

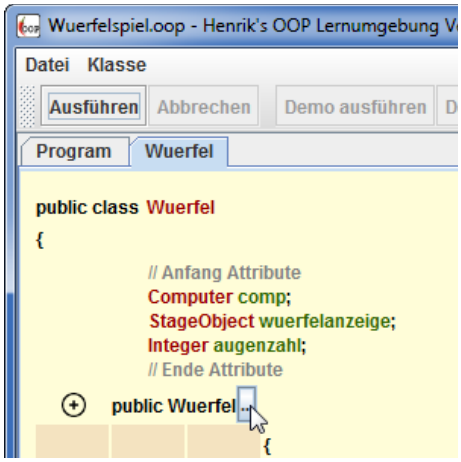
4.3 Parameterliste des Konstruktors bearbeiten

Da jede Klasse zwingend einen Konstruktor benötigt, wird beim Anlegen einer Klasse automatisch ein Konstruktor mit angelegt.

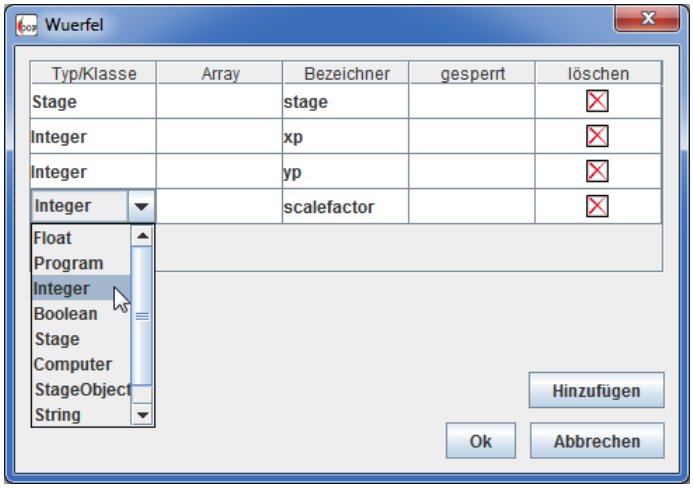
Dieser Konstruktor entspricht allerdings nicht den Anforderungen, die für die Klasse *Wuerfel* nötig sind. Zur Erinnerung: der Konstruktor der Klasse *Wuerfel* benötigt einige Parameter.

Um die Parameterliste zu bearbeiten, muss der Mauszeiger auf diese platziert werden. Es erscheint ein Button.





Mit einem Mausklick auf den Button öffnet sich ein Dialogfenster zum bearbeiten der Parameterliste.



Hinweis:
Parameternamen müssen mit einem Kleinbuchstaben beginnen und dürfen eine Umlaute, ß und Sonderzeichen enthalten.

Die Deklaration von Parametern geht auf die gleiche Weise, wie die Deklaration von Attribute. Alle Eingaben werden mit einem Mausklick auf den Button *Ok* bestätigt.

Die Parameterliste erscheint im Programmcode.

Aufgabe 3:
Führen Sie die oben beschriebenen Schritte im Projekt *Wuerfelspiel.oop* aus.

4.4 Programmcode im Konstruktor

Als nächstes muss Programmcode im Konstruktor eingefügt werden, der ausgeführt wird, wenn ein Objekt der Klasse erzeugt wird. Dabei sollten alle Attribute mit Startwerten versehen werden.

Aufgabe 4:
Fügen Sie in den Konstruktorrumpf eine Codezeile ein, die ein neues Objekt der Klasse *Computer* erzeugt und dieses Objekt dem Attribut *comp* zuweist.

Hier geht es zur Lösung auf Seite 26.

Aufgabe 5:
Erzeugen Sie ein neues Objekt der Klasse *StageObject* mit folgenden Parametern:

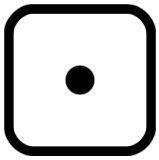
name	= "1"
x	= xp
y	= yp
resName	= "Wuerfel_1.png"
scaleFactor	= scalefactor

und weisen Sie es der Variable *wuerfelanzeige* zu.

Hier geht es zur Lösung auf Seite 27.

Hinweis:
```
new StageObject("1",xp,yp,"Wuerfel_1.png",scalefactor);
```
Da beim Erzeugen des *StageObject* für die Position die Parameter *xp* und *yp* angegeben werden, wird die Postion der Würfelanzeige erst beim erzeugen des *Wuerfel* Objekts festgelegt.
Gleiches gilt für den *scalefactor*.

Mit dem Objekt in *wuerfelanzeige* kann allerdings nur ein Würfel mit der Augenzahl 1 dargestellt werden. Um die anderen Seiten des Würfels darstellen zu können müsste für jede Seite ein weiteres *StageObject* erzeugt werden. Es geht aber auch einfacherer.



Wuerfel_1.png

Hinweis:
Einem Objekt der Klasse *StageObject* können mit der Methode
```
void addSprite(String name, String resName)
```
zusätzliche Grafiken zugefügt werden. Später kann dann eine Grafik zur Darstellung des *StageObject* ausgewählt werden.

Aufgabe 6:

1. Fügen Sie dem *StageObject* in *wuerfelanzeige* folgende Grafiken hinzu:

name	**resName**
"2"	Wuerfel_2.png
"3"	Wuerfel_3.png
"4"	Wuerfel_4.png
"5"	Wuerfel_5.png
"6"	Wuerfel_6.png

2. Machen Sie die *wuerfelanzeige* unsichtbar.

3. Platzieren Sie die *wuerfelanzeige* auf der *stage*.

4. Weisen Sie dem Attribut *augenzahl* eines Objekt der Klasse *Integer* mit dem Wert 0 zu.

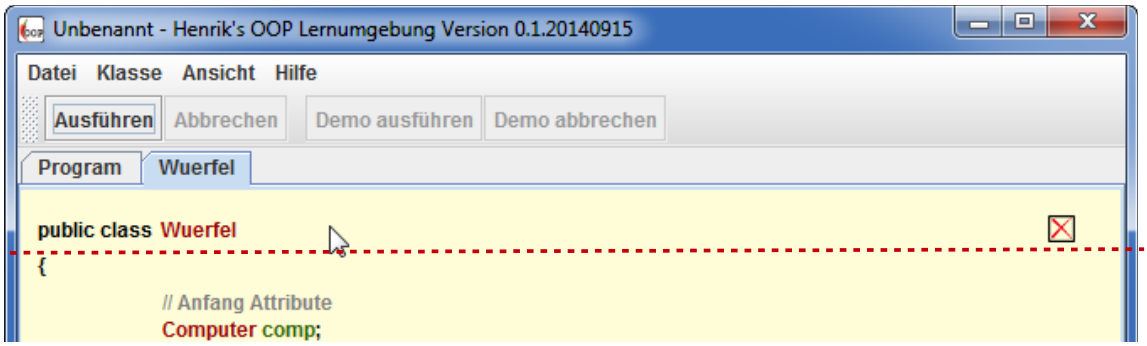
Hier geht es zur Lösung auf Seite 28.

4.5 Die Methode *wuerfeln*

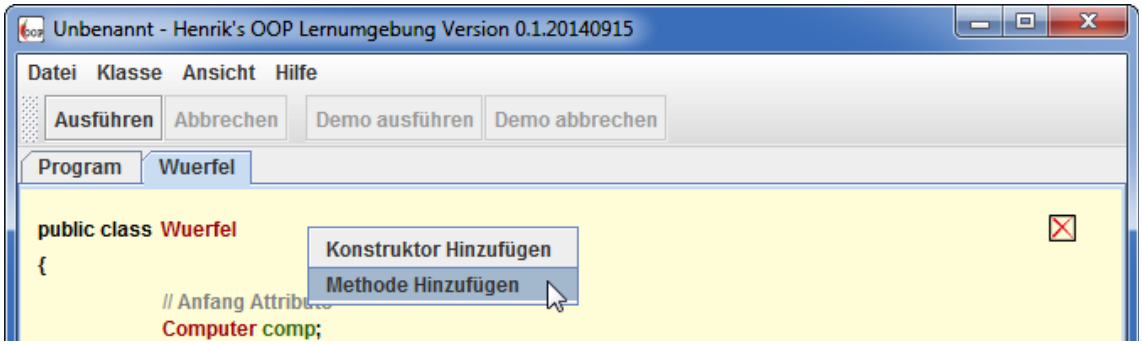
Als nächstes soll die Klasse eine Methode *wuerfeln* erhalten, mit dem der Wurf eines Würfels simuliert werden soll.

Einfügen einer neuen Methode:

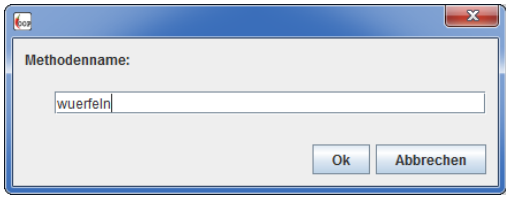
Den Mauszeiger in die erste Zeile der Klasse positionieren:



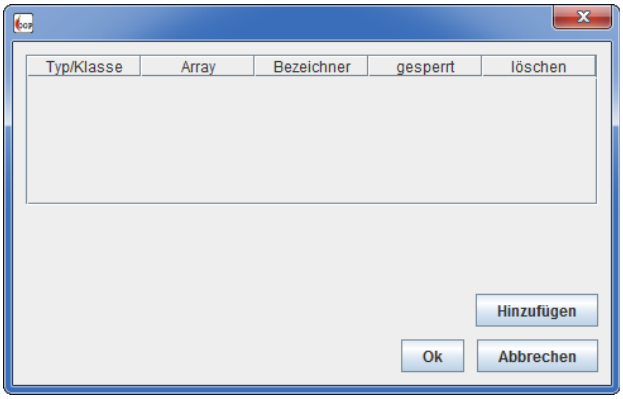
Durch drücken der rechten Mustaste öffnet sich ein Kontextmenü. Dort den Eintrag *Methode Hinzufügen* auswählen.



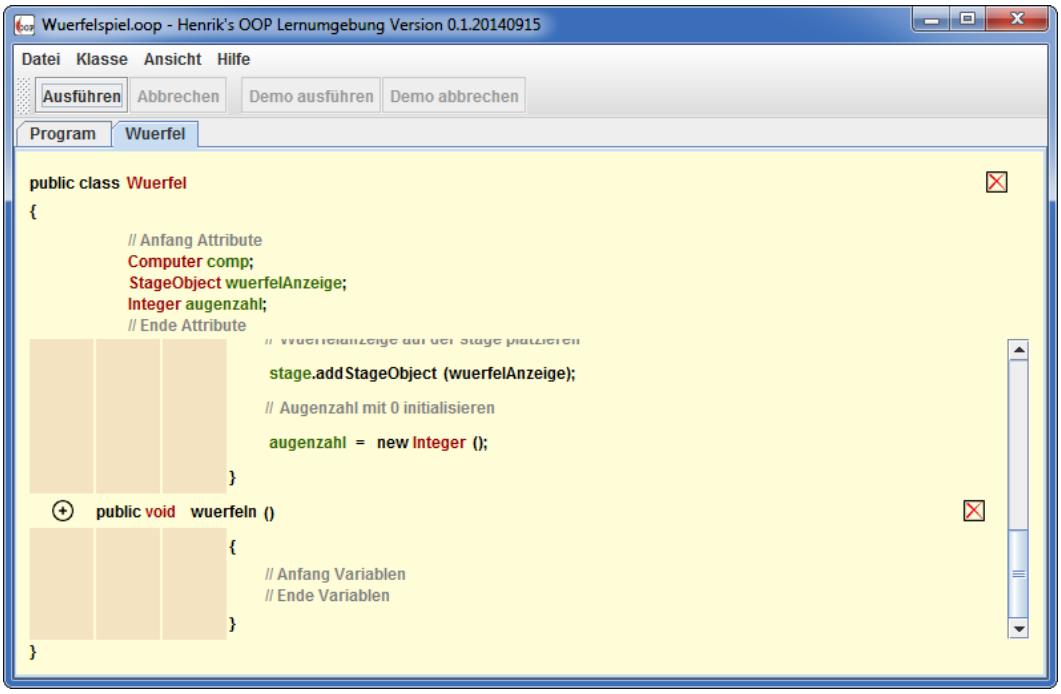
Ein Dialogfenster öffnet sich, in welchem der Name der Methode festgelegt wird:



Nachdem der Name mit *Ok* bestätigt wurde, öffnet sich ein weiteres Dialogfenster zum festlegen der Parameterliste:



Da die Methode *wuerfeln* keine Parmeter benötigt, kann der Dialog gleich mit einem Mausklick auf *Ok* geschlossen werden. In der Code-Ansicht ist die neue Methode zu sehen:



Aufgabe 7:
Führen Sie die oben beschriebenen Schritte im Projekt *Wuerfelspiel.oop* aus.

Die Methode *wuerfeln* mit "Leben" füllen:

Hinweis:
Um Zufallszahlen zu erzeugen wird die folgende Methode der Klasse *Computer* benötigt:

```
Integer rand(Integer min, Integer max)
```

rand liefert eine ganzzahlige Zufallszahl zwischen den Werten *min* und *max* zurück (*min* und *max* eingeschlossen).

Aufgabe 8:

Fügen Sie in den Methodenrumpf von *wuerfeln* eine Code-Zeilen zur Berechnung einer Zufallszahl zwischen 1 und 6. Die Zufallszahl soll dem Attribut *augenzahl* zugewiesen werden.

Hier geht es zur Lösung auf Seite 29.

Als nächstes soll die entsprechende Würfelanzeige auf der *Stage* angezeigt werden. Dazu muss die entsprechende Grafik aus der *wuerfelanzeige* ausgewählt werden.

Zur Erinnerung: der *wuerfelanzeige* wurden nacheinander folgende Grafiken hinzugefügt:

1. 2. 3. 4. 5. 6.

Um jetzt eine Grafik auszuwählen reicht es die Nummer der Grafik anzugeben.

Hinweis:
Allerdings beginnen die Informatiker immer mit 0 beim zählen und **nicht** mit 1.
Daraus ergibt sich wenn *augenzahl* = 1 ist, dann muss die Grafik mit der Nummer (Index) 0 ausgewählt werden. Ist *augenzahl* = 2 ist, dann muss die Grafik mit der Nummer (Index) 1 ausgewählt werden u.s.w.

Hinweis:
Eine Grafik eines *StagObject*s kann mit der Methode

```
void setSprite(Integer pos)
```

pos ist die Nummer der Grafik. Beachten Sie, dass die erste Grafik die Nummer 0 hat.

Aufgabe 9:

1. Fügen Sie in der Methode *wuerfeln* eine Variable *index* der Klasse *Integer* hinzu.
2. Berechnen Sie die Differenz *augenzahl*-1 und speichern Sie das Ergebnis in der neuen Variable *index*.
3. wählen Sie mit *setSprite* die Grafik mit der Nummer *index* aus.
4. Zeigen Sie die Würfelanzeige an.

Hier geht es zur Lösung auf Seite 30.

4.6 Die Methode *letzteAugenzahl*

Die Methode Augenzahl soll die zuletzt gewürfelte Augenzahl zurück geben.

Aufgabe 10:

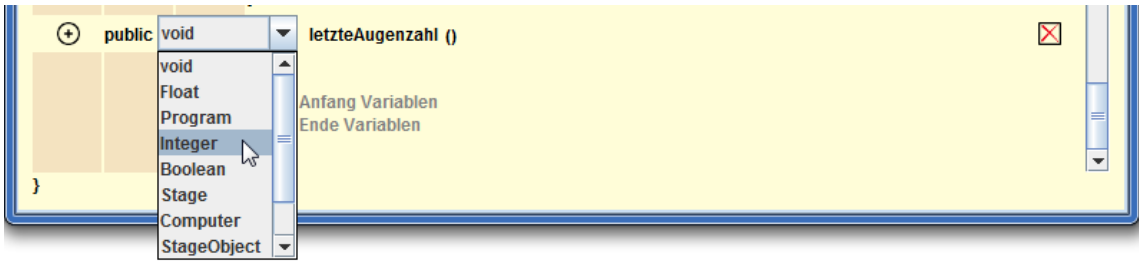
Fügen Sie in die Klasse *Wuerfel* eine neue Methode *letzteAugenzahl* ein. Die Parameterliste von *letzteAugenzahl* ist wiederum leer.

Im Gegensatz zur Methode *wuerfeln* liefert die Methode *letzteAugenzahl* einen Integer Wert zurück.

Um die Klasse für den Rückgabewert zu bestimmen, muss der Mauszeiger auf *void* positioniert werden

Es erscheint eine Dropdownliste, aus der die Klasse für den Rückgabewert ausgewählt wird.

In diesem Fall wird *Integer* ausgewählt, da die zuletzt gewürfelte Augenzahl zurück gegeben werden soll.



Nach dem die Klasse des Rückgabewerts bestimmt wurde, enthält der Methodenrumpf an letzter Stelle eine *return*-Anweisung.

© 2015 Henrik Horstmann

```
public Integer letzteAugenzahl ()
    {
        // Anfang Variablen
        // Ende Variablen
        return NULL;
    }
}
```

Hinweis:
Die *return*-Anweisung steht immer an letzter Stelle. Dem ***return*** folgt eine Variable, Parameter oder Attribut, dessen Objekt zurück gegeben werden soll:

return variable;

Aufgabe 11:

1. Ändern Sie die Klasse für den Rückgabewert der Methode *letzteAugenzahl* so, dass ein Integer Wert zurückgeben wird.
2. Ändern Sie die *return*-Anweisung in der Methode *letzteAugenzahl* so ab, dass der Wert des Attributs *augenzahl* zurückgeben wird.

Hier geht es zur Lösung auf Seite 31.

Die Klasse *Wuerfel* ist damit fertig. Es fehlt nur noch das "Rahmen"-Programm, dass die Würfe anstößt.

5 Das "Rahmen"-Programm

Das Rahmen-Programm wird in der *main*-Methode der Klasse *Programm* programmiert.

Aufgabe 12: Fügen Sie in die *main*-Methode der Klasse *Program* folgenden Code ein:

1. Erzeugen Sie ein *Stage* Objekt mit dem Titel ***Würfelspiel*** und der Größe von **600×480** Pixel. Als Hintergrundbild soll ***sky.jpg*** geladen werden.
   Hier geht es zur Lösung auf Seite 32.
2. Erzeugen Sie drei Objekte ***w1***, ***w2*** und ***w3*** der Klasse ***Wuerfel***. Die drei Würfel sollen um 50% herunterscalliert werden und an folgenden Positionen auf der *Stage* platziert werden:

Objekt	**x**	**y**
w1	100	240
w2	300	240
w3	500	240

Hier geht es zur Lösung auf Seite 33.

3. Machen Sie die *Stage* sichtbar.

   Hier geht es zur Lösung auf Seite 34.

4. Geben Sie auf der Konsole die Meldung "1. Würfel würfeln: Taste drücken" aus. Warten Sie auf einen Tastendruck und würfeln Sie den Würfel *w1*. Das gleiche wird mit den anderen beiden Würfeln gemacht.

   Hier geht es zur Lösung auf Seite 35.

5. Berechnen Sie die Augensumme der drei Würfel.

   Hier geht es zur Lösung auf Seite 36.

6. Geben Sie de Meldung "Es wurden *<augensumme> erreicht!" aus. <augensumme>* ist die in Schritt 5 berechnete Augensumme.

   Hier geht es zur Lösung auf Seite 37.

Testen Sie Ihr Programm.

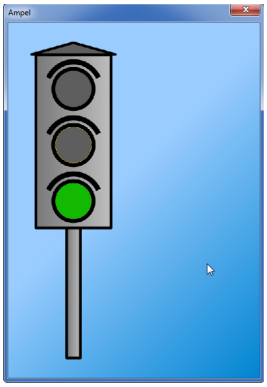
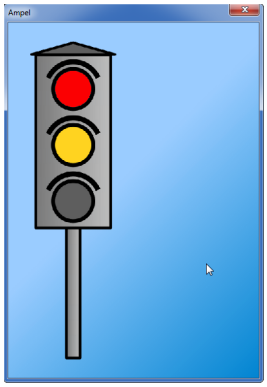
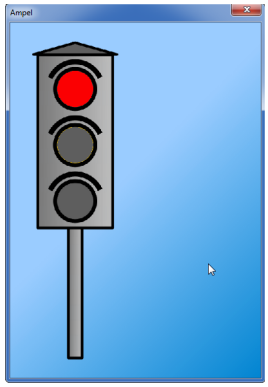
Herzlichen Glückwunsch, Sie haben soeben ein Programm mit mehreren Klassen geschrieben.

Vertiefen Sie Ihr neues wissen mit den Übungen auf den folgenden Seiten.

© 2015 Henrik Horstmann

6 Übungen

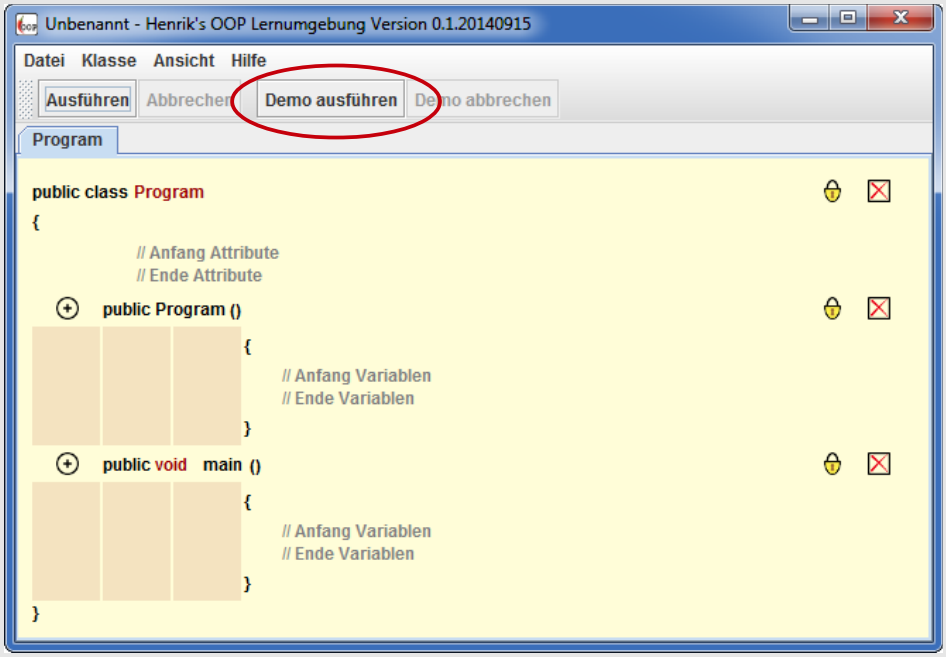
6.1 Ampel

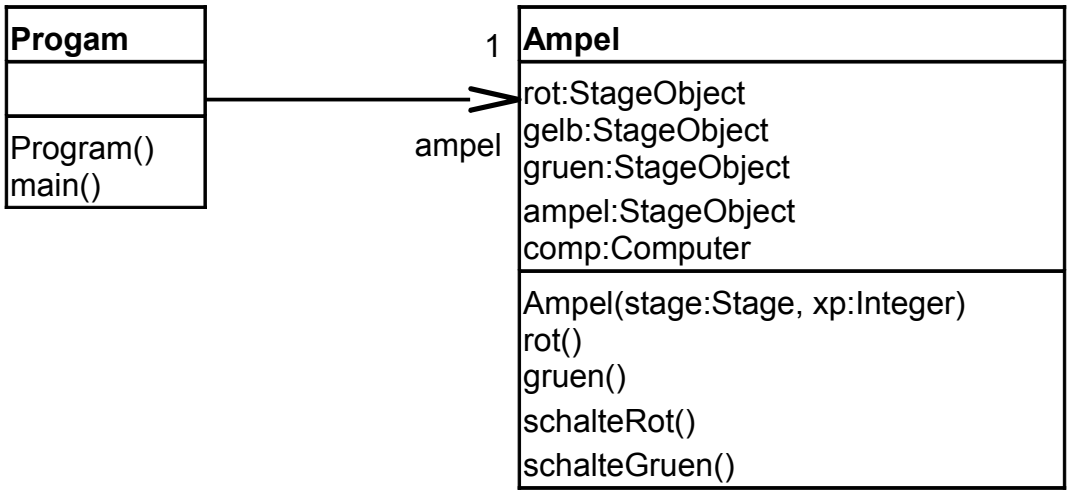
Das Programm *Ampel.oop* soll eine Verkehrsampel simulieren.

Die Funktionsweise des Programms wird mit dem Programm *Ampel.demo.oop* demonstriert.

Demo Programm:
Öffnen Sie das Programm *Ampel.demo.oop*. Zwar ist der Programmcode nicht sichtbar, trotzdem kann das Programm über den Button *Demo ausführen* gestartet werden.



UML-Diagramm des Programm Ampel.oop



Beschreibung der Klasse Ampel

Ampel(stage:Stage, xp:Integer)	
stage	Die "Bühne", auf der die Ampel platziert werden soll.
xp	Horizontale Position der Ampel
Es werden alle Attribute mit Startwerten versehen. Die Ampel wird auf der "Bühne" angezeigt und ist auf grün geschaltet.	

rot()
Nach dem Aufruf leuchtet das rote Licht. Alle anderen Lichter sind aus.

gruen()
Nach dem Aufruf leuchtet das grüne Licht. Alle anderen Lichter sind aus.

schalteRot()
Das rote und gelbe Licht werden ein- und das grüne Licht ausgeschaltet. Es wird 1 Sec. gewartet. Das gelbe Licht wird ausgeschaltet.

Hinweis:
Die Klasse *Computer* enthält eine Methode

```
void wait(Integer msec)
```

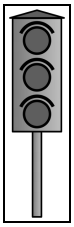
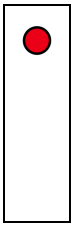
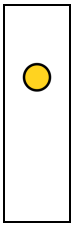
Ein Aufruf von *wait* verzögert die Programmausführung um msec Millisekunden.

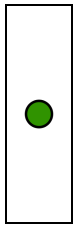
schalteGruen()
Das rote und grüne Licht werden aus- und das gelbe Licht eingeschaltet. Es wird 1 Sec. gewartet. Das gelbe Licht wird aus- und das grün Licht wird eingeschaltet.

Beschreibung der Klasse Program

main()
Es wird eine "Bühne" erzeugt und eine Ampel darauf platziert. Die Ampel ist auf grün geschaltet. Auf der Konsole wird die Meldung "Auf rot schalten: Taste drücken" ausgegeben. Nach dem drücken einer beliebigen Taste schaltet die Ampel auf rot. Auf der Konsole wird die Meldung "Auf grün schalten: Taste drücken" ausgegeben. Nach dem drücken einer beliebigen Taste schaltet die Ampel wieder auf grün.

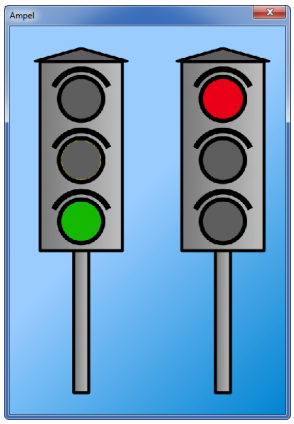
Liste der benötigten Grafiken

Dateiname	**Grafik**
bg_blue.png	
Ampel.png	
Ampel.rot.png	
Ampel.gelb.png	

Ampel.gruen.png	

6.2 Ampelschaltung

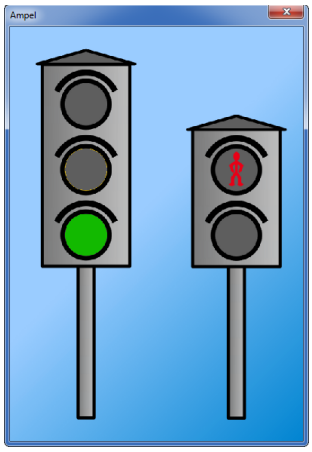
Erweitern Sie das Programm Ampel aus Übung 6.1 um eine weitere Ampel, die zu Programmstart auf rot geschaltet ist. Nach einem Tastendruck soll die erste Ampel auf rot und die zweite Ampel auf grün schalten. Nach dem zweiten Tastendruck soll die zweite Ampel wieder auf rot und die erste wieder auf grün schalten. Achten Sie darauf, dass zu keiner Zeit beiden Ampeln gleichzeitig auf grün geschaltet sind, denn das kann böse Folgen haben.



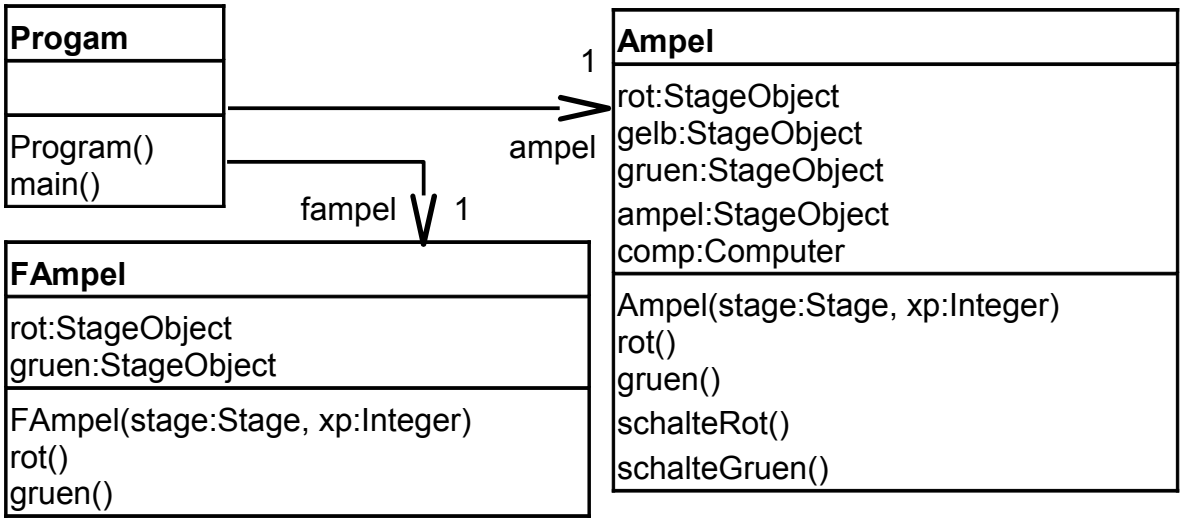
Demo Programm:
Um zu sehen, wie das Programm funktionieren soll, öffnen Sie das Programm *Ampelschaltung.demo.oop*. Zwar ist der Programmcode nicht sichtbar, trotzdem kann das Programm über den Button *Demo ausführen* gestartet werden.

6.3 Fußgängerampel

Das Programm *FussgaengerAmpel.oop* soll eine Fußgängerampel simulieren, die auf Anforderung das Fußgänger Signal auf grün schaltet ("Drückampel").



UML-Diagramm des Programm FussgaengerAmpel.oop



Beschreibung der Klasse FAmpel

FAmpel(stage:Stage, xp:Integer)	
stage	Die "Bühne", auf der die Ampel platziert werden soll.
xp	Horizontale Position der Ampel
Es werden alle Attribute mit Startwerten versehen. Die Ampel wird auf der "Bühne" angezeigt und ist auf rot geschaltet.	

rot()
Nach dem Aufruf leuchtet das rote Licht. Das grüne Licht ist aus.

gruen()
Nach dem Aufruf leuchtet das grüne Licht. Das rote Licht ist aus.

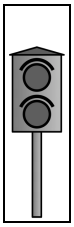
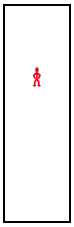
Beschreibung der Klasse Ampel

Siehe Beschreibung in Übung 6.1.

Beschreibung der Klasse Programm

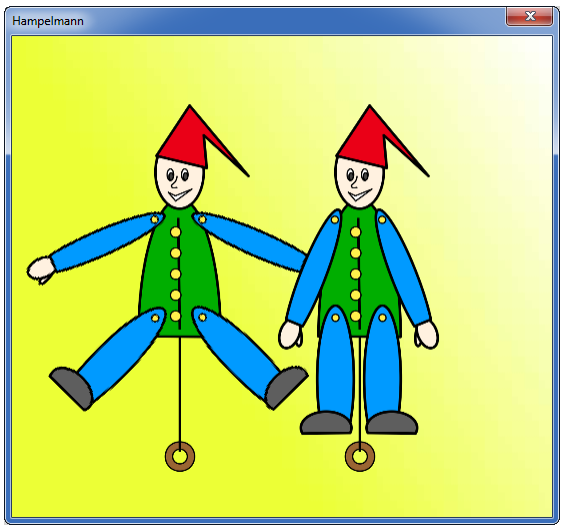
main()
Es wird eine "Bühne" erzeugt und eine Ampel, sowie eine Fußgängerampel darauf platziert. Die Ampel ist auf grün und die Fußgängerampel auf rot geschaltet. Auf der Konsole wird die Meldung "Auf grün schalten: Taste drücken" ausgegeben. Nach dem drücken einer beliebigen Taste schaltet die Ampel auf rot und die Fußgängerampel mit einer Verzögerung von 1 Sec. auf grün. Nach 2 Sec. schaltet die Fußgängerampel wieder auf rot und mit einer Verzögerung von 1 Sec. de Ampel wieder auf grün.

Liste der zusätzlich benötigten Grafiken

Dateiname	**Grafik**
FAmpel.png	
FAmpel.rot.png	
FAmpel.gruen.png	

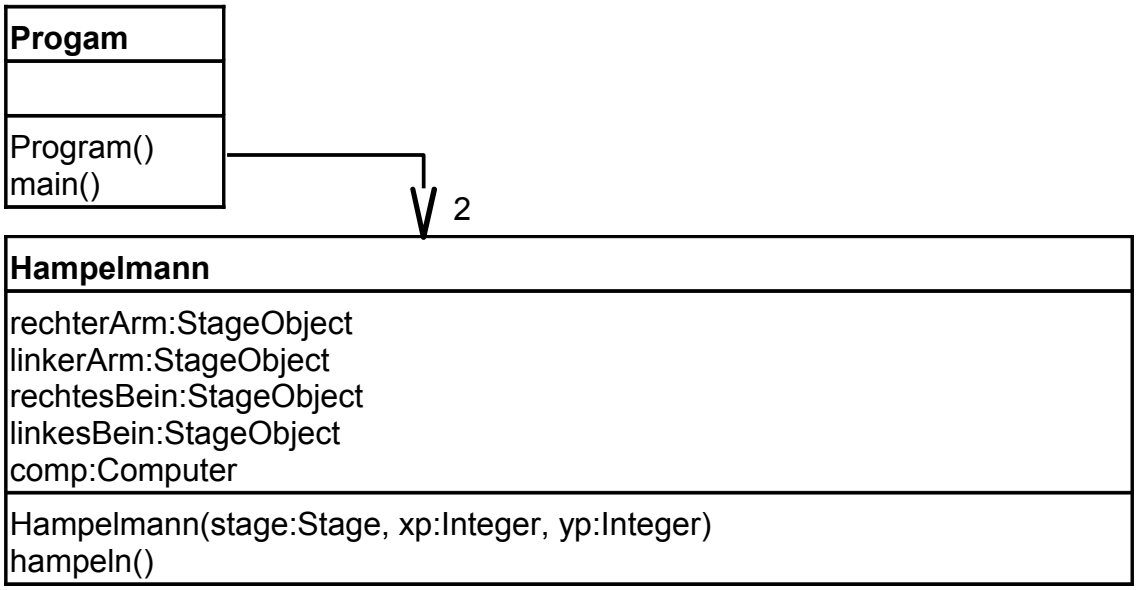
Demo Programm:
Um zu sehen, wie das Programm funktionieren soll, öffnen Sie das Programm *FussgaengerAmpel.demo.oop*. Zwar ist der Programmcode nicht sichtbar, trotzdem kann das Programm über den Button *Demo ausführen* gestartet werden.

6.4 Hampelmann



Auf Tastendruck sollen in dem Programm *Hampelmann.oop* nacheinander zwei Hampelmänner einen "Hampelmann" machen.

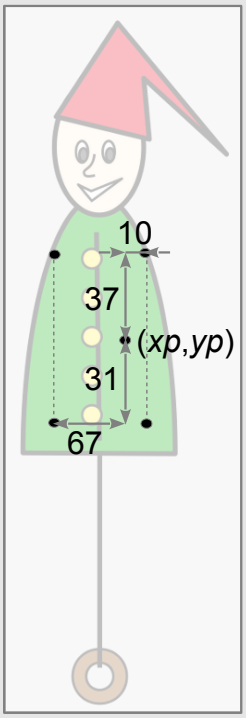
UML-Diagramm des Programm Hampelmann.oop



Beschreibung der Klasse Hampelmann

Hampelmann(stage:Stage, xp:Integer, yp:Integer)	
stage	Die "Bühne", auf der der Hampelmann platziert werden soll.
xp	Horizontale Position des Hampelmanns auf der "Bühne"
yp	Vertikale Position des Hampelmanns auf der "Bühne"
Es werden alle Attribute mit Startwerten versehen. Der Hampelmann wird an die Position (*xp*,*yp*) positioniert. Der Skalierungsfaktor beträgt 50%.	

Hinweis:
Die Abstände der Gelenke für Arme und Beine sind der nebenstehenden Grafik zu entnehmen. Die Angaben beziehen sich auf eine Skalierung von 50%.



hampeln()
Alle Gliedmaßen werden um 45° geneigt. Nach einer Verzögerung von 1 Sec. kehren die Gliedmaßen in ihre Ausgangsposition zurück.

StageObjects drehen:
Die Klasse *StageObject* besitzt eine Methode, mit der sie gedreht werden können:

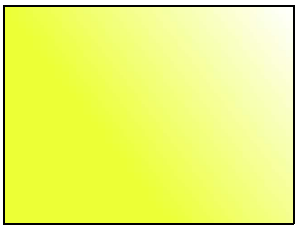
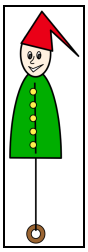
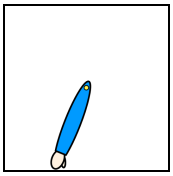
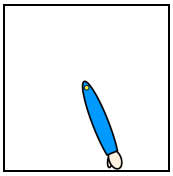
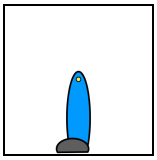
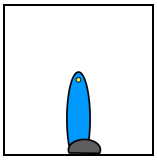
```
void StageObject.setRotation(Float angle)
```

Dabei ist *angle* der Winkel in Grad.

Beschreibung der Klasse Programm

main()
Es wird eine "Bühne" erzeugt und zwei Hampelmänner platziert. Auf der Konsole erscheint die Meldung "Um an der Schnur zu ziehen: beliebige Taste drücken." Nach dem drücke einer beliebigen Taste machen die beiden Hampelmänner nacheinander einen "Hampelmann".

Liste der benötigten Grafiken

Dateiname	**Grafik**
bg_yellow.jpg	
Hampelmann.png	
Hampelmann.Arm.links.png	
Hampelmann.Arm.rechts.png	
Hampelmann.Fuss.links.png	
Hampelmann.Fuss.rechts.png	

© 2015 Henrik Horstmann

Demo Programm:
Um zu sehen, wie das Programm funktionieren soll, öffnen Sie das Programm *Hampelmann.demo.oop*. Zwar ist der Programmcode nicht sichtbar, trotzdem kann das Programm über den Button *Demo ausführen* gestartet werden.

ENDE

7 Lösungen zu den Aufgaben

7.1 Lösung zu Aufgabe 4

```
public class Wuerfel
{
   // Anfang Attribute
   private Computer comp;
   private StageObject wuerfelanzeige;
   private Integer augenzahl;
   // Ende Attribute


   public Wuerfel (Stage stage,Integer xp,
                   Integer yp,Integer scalefactor)
   {
      // Anfang Variablen
      // Ende Variablen

      // Ein Objekt Computer erzeugen
      comp = new Computer();
   }
}
```

Zurück zur Aufgabe auf Seite 8.

7.2 Lösung zu Aufgabe 5

```
public class Wuerfel
{
   // Anfang Attribute
   private Computer comp;
   private StageObject wuerfelanzeige;
   private Integer augenzahl;
   // Ende Attribute


   public Wuerfel (Stage stage,Integer xp,
                   Integer yp,Integer scalefactor)
   {
      // Anfang Variablen
      // Ende Variablen

      // Ein Objekt der Klasse Computer erzeugen.
      comp = new Computer();
      // Wuerfelanzeige erzeugen.
      wuerfelanzeige = new StageObject("1",xp,yp,
                                       "Wuerfel_1.png",scalefactor);
   }
}
```

Zurück zur Aufgabe auf Seite 8.

7.3 Lösung zu Aufgabe 6

```
public class Wuerfel
{
   // Anfang Attribute
   private Computer comp;
   private StageObject wuerfelanzeige;
   private Integer augenzahl;
   // Ende Attribute


   public Wuerfel (Stage stage,Integer xp,
                   Integer yp,Integer scalefactor)
   {
      // Anfang Variablen
      // Ende Variablen

      // Ein Objekt der Klasse Computer erzeugen.
      comp = new Computer();
      // Wuerfelanzeige erzeugen.
      wuerfelanzeige = new StageObject("1",xp,yp,
                                       "Wuerfel_1.png",scalefactor);
      wuerfelanzeige.addSprite("2","Wuerfel_2.png");
      wuerfelanzeige.addSprite("3","Wuerfel_3.png");
      wuerfelanzeige.addSprite("4","Wuerfel_4.png");
      wuerfelanzeige.addSprite("5","Wuerfel_5.png");
      wuerfelanzeige.addSprite("6","Wuerfel_6.png");
      // Wuerfelanzeige verbergen
      wuerfelanzeige.hide();
      // Wuerfelanzeige auf der stage platzieren
      stage.addStageObject(wuerfelAnzeige);
      // Augenzahl mit 0 initialisieren
      augenzahl = new Integer();   }
}
```

Zurück zur Aufgabe auf Seite 9.

7.4 Lösung zu Aufgabe 8

```
...
   public void wuerfeln ()
   {
      // Anfang Variablen
      // Ende Variablen

      // Zufallszahl zwischen 1 und 6 erzeugen
      augenzahl = comp.rand(1,6);
   }
...
```

Zurück zur Aufgabe auf Seite 12.

7.5 Lösung zu Aufgabe 9

```
...

   public void wuerfeln ()
   {
      // Anfang Variablen
      Integer index;
      // Ende Variablen

      // Zufallszahl zwischen 1 und 6 erzeugen
      augenzahl = comp.rand(1,6);
      // Index fuer die Grafik zum Anzeigen des Wuerfels berechnen
      index = augenzahl.sub(1);
      // Grfik fuer die Wuerfelanzeige auswaehlen
      wuerfelAnzeige.setSprite(index);
      // Wuerfelanzeige anzeigen
      wuerfelAnzeige.show();
   }
...
```

Zurück zur Aufgabe auf Seite 13.

7.6 Lösung zu Aufgabe 11

```
public class Wuerfel
{
   // Anfang Attribute
   private Computer comp;
   private StageObject wuerfelAnzeige;
   private Integer augenzahl;
   // Ende Attribute

   public Wuerfel (Stage stage,Integer xp,
                   Integer yp,Integer scalefactor)
   {
      // Anfang Variablen
      // Ende Variablen
      // Ein Objekt der Klasse Computer erzeugen
      comp = new Computer();
      // Wuerfelanzeige erzeugen
      wuerfelAnzeige = new StageObject("1",xp,yp,
                                       "Wuerfel_1.png",scalefactor);
      wuerfelAnzeige.addSprite("2","Wuerfel_2.png");
      wuerfelAnzeige.addSprite("3","Wuerfel_3.png");
      wuerfelAnzeige.addSprite("4","Wuerfel_4.png");
      wuerfelAnzeige.addSprite("5","Wuerfel_5.png");
      wuerfelAnzeige.addSprite("6","Wuerfel_6.png");
      // Wuerfelanzeige verbergen
      wuerfelAnzeige.hide();
      // Wuerfelanzeige auf der stage platzieren
      stage.addStageObject(wuerfelAnzeige);
      // Augenzahl mit 0 initialisieren
      augenzahl = new Integer();
   }

   public void wuerfeln ()
   {
      // Anfang Variablen
      Integer index;
      // Ende Variablen
      // Zufallszahl zwischen 1 und 6 erzeugen
      augenzahl = comp.rand(1,6);
      // Index fuer die Grafik zum Anzeigen des Wuerfels berechnen
      index = augenzahl.sub(1);
      // Grfik fuer die Wuerfelanzeige auswaehlen
      wuerfelAnzeige.setSprite(index);
      // Wuerfelanzeige anzeigen
      wuerfelAnzeige.show();
   }

   public Integer letzteAugenzahl ()
   {
      // Anfang Variablen
      // Ende Variablen
      return augenzahl;
   }
}
```

Zurück zur Aufgabe auf Seite 14.

7.7 Lösung zu Aufgabe 12.1

```
public class Program
{
   // Anfang Attribute
   // Ende Attribute

   public Program ()
   {
      // Anfang Variablen
      // Ende Variablen
   }

   public void main ()
   {
      // Anfang Variablen
      Stage stage;
      // Ende Variablen

      // "Buehne" mit Himmelhintegrund ezeugen
      stage = new Stage("Würfelspiel",600,480);
      stage.loadBackground("sky.jpg");
   }
}
```

Zurück zur Aufgabe auf Seite 14.

7.8 Lösung zu Aufgabe 12.2

```
public class Program
{
   // Anfang Attribute
   // Ende Attribute

   public Program ()
   {
      // Anfang Variablen
      // Ende Variablen
   }

   public void main ()
   {
      // Anfang Variablen
      Wuerfel w2;
      Wuerfel w1;
      Wuerfel w3;
      Stage stage;
      // Ende Variablen

      // "Buehne" mit Himmelhintegrund ezeugen
      stage = new Stage("Würfelspiel",600,480);
      stage.loadBackground("sky.jpg");
      // 3 Wuerfelobjekte erzeugen und auf der Buehne platzieren
      w1 = new Wuerfel(stage,100,240,50);
      w2 = new Wuerfel(stage,300,240,50);
      w3 = new Wuerfel(stage,500,240,50);
   }
}
```

Zurück zur Aufgabe auf Seite 14.

7.9 Lösung zu Aufgabe 12.3

```
public class Program
{
   // Anfang Attribute
   // Ende Attribute

   public Program ()
   {
      // Anfang Variablen
      // Ende Variablen
   }

   public void main ()
   {
      // Anfang Variablen
      Wuerfel w2;
      Wuerfel w1;
      Wuerfel w3;
      Stage stage;
      // Ende Variablen

      // "Buehne" mit Himmelhintegrund ezeugen
      stage = new Stage("Würfelspiel",600,480);
      stage.loadBackground("sky.jpg");
      // 3 Wuerfelobjekte erzeugen und auf der Buehne platzieren
      w1 = new Wuerfel(stage,100,240,50);
      w2 = new Wuerfel(stage,300,240,50);
      w3 = new Wuerfel(stage,500,240,50);
      // "Buehne" anzeigen
      stage.start();
   }
}
```

Zurück zur Aufgabe auf Seite 14.

7.10 Lösung zu Aufgabe 12.4

```
public class Program
{
   // Anfang Attribute
   // Ende Attribute

   public Program ()
   {
      // Anfang Variablen
      // Ende Variablen
   }

   public void main ()
   {
      // Anfang Variablen
      Wuerfel w2;
      Wuerfel w1;
      Wuerfel w3;
      Stage stage;
      Computer comp;
      // Ende Variablen

      // "Buehne" mit Himmelhintegrund ezeugen
      stage = new Stage("Würfelspiel",600,480);
      stage.loadBackground("sky.jpg");
      // 3 Wuerfelobjekte erzeugen und auf der Buehne platzieren
      w1 = new Wuerfel(stage,100,240,50);
      w2 = new Wuerfel(stage,300,240,50);
      w3 = new Wuerfel(stage,500,240,50);
      // "Buehne" anzeigen
      stage.start();
      // Interaktion mit de Benutzer
      // 1. Aufforderung eine Tast zu druecken
      // 2. Nach Tastenduck Wurf durchfuehren
      comp = new Computer();
      comp.print("1. Würfel würfeln: Taste drücken.");
      stage.waitUntilKeyPressed();
      w1.wuerfeln();
      comp.println();
      comp.print("2. Würfel würfeln: Taste drücken.");
      stage.waitUntilKeyPressed();
      w2.wuerfeln();
      comp.println();
      comp.print("3. Würfel würfeln: Taste drücken.");
      stage.waitUntilKeyPressed();
      w3.wuerfeln();
   }
}
```

Zurück zur Aufgabe auf Seite 14.

7.11 Lösung zu Aufgabe 12.5

```
public class Program
{
   // Anfang Attribute
   // Ende Attribute

   public Program ()
   {
      // Anfang Variablen
      // Ende Variablen
   }

   public void main ()
   {
      // Anfang Variablen
      Wuerfel w2;
      Wuerfel w1;
      Wuerfel w3;
      Computer comp;
      Stage stage;
      Integer a;
      Integer summe;
      // Ende Variablen
      // "Buehne" mit Himmelhintegrund ezeugen
      stage = new Stage("Würfelspiel",600,480);
      stage.loadBackground("sky.jpg");
      // 3 Wuerfelobjekte erzeugen und auf der Buehne platzieren
      w1 = new Wuerfel(stage,100,240,50);
      w2 = new Wuerfel(stage,300,240,50);
      w3 = new Wuerfel(stage,500,240,50);
      // "Buehne" anzeigen
      stage.start();
      // Interaktion mit de Benutzer
      // 1. Aufforderung eine Tast zu druecken
      // 2. Nach Tastenduck Wurf durchfuehren
      comp = new Computer();
      comp.print("1. Würfel würfeln: Taste drücken.");
      stage.waitUntilKeyPressed();
      w1.wuerfeln();
      comp.println();
      comp.print("2. Würfel würfeln: Taste drücken.");
      stage.waitUntilKeyPressed();
      w2.wuerfeln();
      comp.println();
      comp.print("3. Würfel würfeln: Taste drücken.");
      stage.waitUntilKeyPressed();
      w3.wuerfeln();
      // Berechnung der Augensumme
      summe = w1.letzteAugenzahl();
      a = w2.letzteAugenzahl();
      summe = summe.add(a);
      a = w3.letzteAugenzahl();
      summe = summe.add(a);
   }
}
```

Zurück zur Aufgabe auf Seite 14.

7.12 Lösung zu Aufgabe 12.6

```
public class Program
{
   // Attribute

   public Program () { }

   public void main ()
   {
      // Anfang Variablen
      Wuerfel w1;
      Wuerfel w2;
      Wuerfel w3;
      Stage stage;
      Computer comp;
      Integer a;
      Integer summe;
      // Ende Variablen
      // "Buehne" mit Himmelhintegrund ezeugen
      stage = new Stage("Würfelspiel",600,480);
      stage.loadBackground("sky.jpg");
      // 3 Wuerfelobjekte erzeugen und auf der Buehne platzieren
      w1 = new Wuerfel(stage,100,240,50);
      w2 = new Wuerfel(stage,300,240,50);
      w3 = new Wuerfel(stage,500,240,50);
      // "Buehne" anzeigen
      stage.start();
      // Interaktion mit de Benutzer
      // 1. Aufforderung eine Tast zu druecken
      // 2. Nach Tastenduck Wurf durchfuehren
      comp = new Computer();
      comp.print("1. Würfel würfeln: Taste drücken.");
      stage.waitUntilKeyPressed();
      w1.wuerfeln();
      comp.println();
      comp.print("2. Würfel würfeln: Taste drücken.");
      stage.waitUntilKeyPressed();
      w2.wuerfeln();
      comp.println();
      comp.print("3. Würfel würfeln: Taste drücken.");
      stage.waitUntilKeyPressed();
      w3.wuerfeln();
      // Berechnung der Augensumme
      summe = w1.letzteAugenzahl();
      a = w2.letzteAugenzahl();
      summe = summe.add(a);
      a = w3.letzteAugenzahl();
      summe = summe.add(a);
      // Ausgabe des Ergebnis:
      comp.println();
      comp.print("Es wurden ");
      comp.print(summe);
      comp.print(" Punkte erreicht!");
   }
}
```

Zurück zur Aufgabe auf Seite 14.