OOP

Wenn es wahr ist, dann ...

Henrik Horstmann

23. September 2014

Inhaltsverzeichnis

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.

© 2014 Henrik Horstmann

2 Und nichts als die Wahrheit

2.1 Wahrheitswerte

Nehmen wir an es wird eine Münze geworfen, so kann das Experiment zwei mögliche Ergebnisse haben: Kopf oder Zahl. Die Frage, ob das Ergebnis des Experiments mit Kopf endet kann nur mit ja oder nein beantwortet werden. Ja und Nein sind sogenannte Wahrheitswerte.

2.2 Die Klasse *Boolean*

Um im Programm mit Wahrheitswerten zu arbeiten gibt es eine eigene Klasse dazu.

Ein Objekt der Klasse *Boolean* kann folgende zwei Werte repräsentieren:

Wahrheitswert	**Entsprechungen**
TRUE	ja, wahr, richtig
FALSE	nein, unwahr, falsch

Beispiel:

```
Boolean b;
b = new Boolean(TRUE);
```

In der 1. Zeile wird eine Variable *b* der Klasse *Boolean* deklariert. In der 2. Zeile wird ein Objekt der Klasse *Boolean* mit dem Wert *TRUE* (=wahr, richtig, ja, ...) erzeugt und der Variable *b* zugewiesen.

2.3 Bedingte Anweisungen

Die beiden Geschwister Tim und Tom streiten sich, wer in der Küche helfen muss. Um eine gerechte Entscheidung zu treffen werfen Sie eine Münze.

Kopf ≙ Tim
Zahl ≙ Tom

WENN beim Münzwurf der Kopf fällt

DANN muss Tim in der Küche helfen

SONST muss Tom in de Küche helfen

2.3.1 Wenn dann sonst

Solche "Wenn dann sonst" Formulierungen werden in der Programmierung Bedingte-Anweisung genannt.

© 2014 Henrik Horstmann

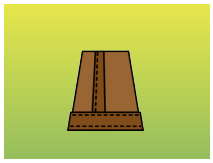
Bedingte Anweisung:

Bedingung

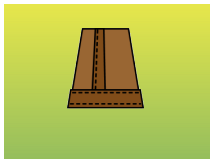
```
if ( muenze.istKopf() )          // WENN
{                                Rumpf der bedingten Anweisung
   // Tim hilft in der Küche.    // DANN
}
else                             // SONST
{
   // Tom hilft in der Küche.
}
```

3 Das Münzbecher Experiment

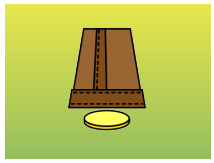
Vor einem Spieler steht ein Becher mit der Öffnung nach unten auf dem Tisch. Der Spieler muss nun raten, ob sich unter dem Becher eine Münze befindet. Danach wird der Becher angehoben. Hat der Spieler richtig geraten, so hat er gewonnen.



geschlossen



offen ohne Münze



offen mit Münze

3.1 Der gezinkte Münzbecher

Zunächst soll eine Klasse erstellt werden, die den Münzbecher repräsentiert.

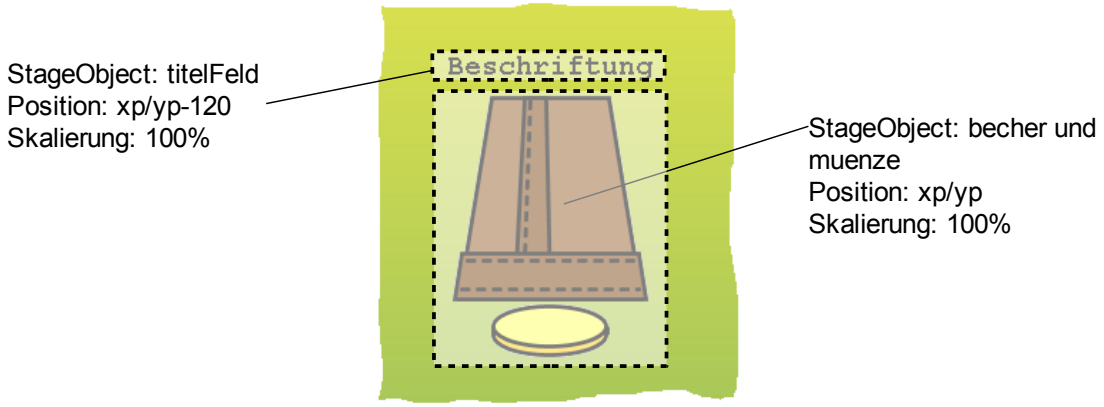
Muenzbecher
titelFeld:StageObject becher:StageObject muenze:StageObject mitMuenze:Boolean
Muenzbecher(stage:Stage,xp:Integer,yp:Integer,titel:String) oeffnen():Boolean

© 2014 Henrik Horstmann

Beschreibung der Klasse Muenzbecher

Muenzbecher(stage:Stage,xp:Integer,yp:Integer,titel:String)	
stage	Bühne auf der, der Münzbecher dargestellt werden soll.
xp	Horizontale Postion des Münzbechers auf der Bühne.
yp	Vertikale Postion des Münzbechers auf der Bühne.
titel	Beschriftung des Münzbechers

Auf der Bühne sollen drei StageObjekte platziert werden:



Das StageObject *titelFeld* bekommt keine Grafik, dafür wird der String aus dem Parameter *title* als Text gesetzt. Schriftgröße 20.

Das StageObject *muenze* bekommt die Grafik *Muenze.png* (Skalierung 100%).

Das StageObject *becher* bekommt zunächst keine Grafiken. Folgende Grafiken sollen in gleicher Reihenfolge als Sprites hinzugefügt werden: *Becher.geschlossen.png* und Becher.offen.png (Skalierung 100%).

Das Attribut *mitMuenze* soll mit dem Wert *FALSE* initialisiert werden.

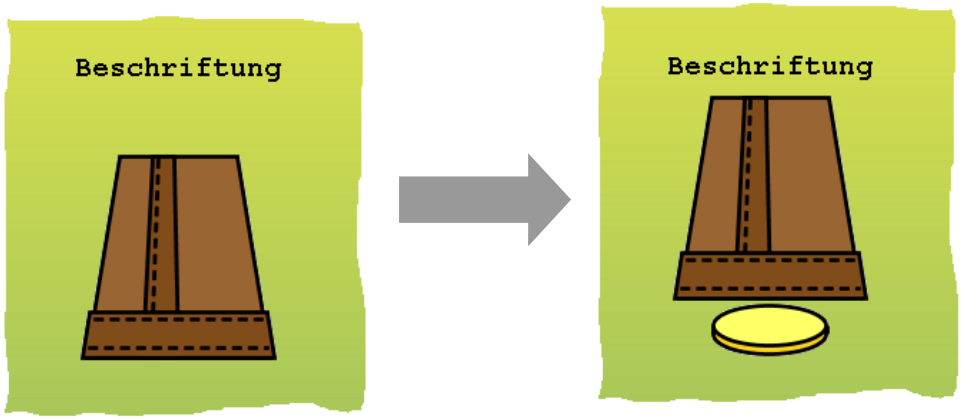
Aufgabe 1:

Erstellen Sie in dem Projekt *Muenzbecher.01.oop*.

Erzeugen Sie eine neue Klasse *Muenzbecher* und implementieren Sie den Konstruktor entsprechend der Beschreibung.

Hier geht es zur Lösung auf Seite 19.

© 2014 Henrik Horstmann

oeffnen():Boolean
Der Becher soll angehoben (geöffnet) werden:    Die Methode liefert den Wert des Attributs *mitMuenze* zurück.

Aufgabe 2:

Ergänzen Sie die Klasse *Muenzbecher* um die oben beschrieben Methode *oeffnen*.

Ein Tipp zum öffnen des Bechers ist auf Seite 20.

Hier geht es zur Lösung auf Seite 21.

3.2 Das Experiment

In der Klasse *Program* in der Methode *main* wird das Münzbecher Experiment implementiert.

Aufgabe 3:

Wechseln Sie in die Klasse *Program* und bearbeiten Sie die Methode *main*.

Fügen Sie der Methode *main* die Variablen *stage* der Klasse *Stage*, *mb1* der Klasse *Muenzbecher* und *comp* der Klasse *Computer* hinzu.

Erzeugen Sie ein neues Objekt der Klasse Stage und weisen Sie es der Variablen *stage* zu. Die Stage soll eine Größe von 640×480 Pixel haben und den Titel "Münzbecher Experiment" tragen.

BG_green.png soll das Hintergrundbild der Stage sein.

Erzeugen Sie ein neues Objekt der Klasse Muenzbecher. Der Münzbecher soll mittig auf der Stage in *stage* platziert werden. Der Münzbecher erhält keine Beschriftung. Das erzeugte Objekt soll der Variablen *mb1* zugewiesen werden.

© 2014 Henrik Horstmann

Die Stage mit dem Münzbecher soll angezeigt werden.

In der Konsole wird der Text "Becher öffnen: beliebige Taste drücken" angezeigt und das Programm unterbrochen, bis ein Tastendruck erfolgt.

Zuletzt wird der Becher geöffnet.

Testen Sie Ihr Programm.

Hier geht es zur Lösung auf Seite 22.

3.3 Ein echter Münzbecher

Der bis jetzt programmierte Münzbecher enthält immer eine Münze. Dies soll nun geändert werden. Nach dem Zufallsprinzip soll der Becher eine Münze enthalten.

Hinweis:

Erzeugen Sie mit

```
Computer.rand(0,1)
```

zufällig ene der Zahlen 0 oder 1. 0 bedeutet keine Münze unterm Becher, 1 bedeutet Münze ist unter dm Becher.

Aufgabe 4:

Erweitern Sie den Konstruktor der Klasse *Muenzbecher*.

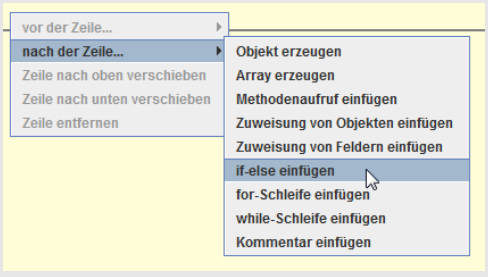
Erzeugen Sie eine Zufallszahl im Bereich von 0 bis 1 und weisen Sie das Ergebnis einer variablen *m* zu.

Hier geht es zur Lösung auf Seite 23.

Im Attribut *mitMuenze* soll gespeichert werden, ob unter dem Becher eine Münze liegt. Bis jetzt ist *mitMuenze* mit *FALSE* initialisiert, was soviel bedeutet, dass keine Münze unter dem Becher liegt. *mitMuenze* muss jedoch auf *TRUE* gesetzt werden, wenn die Variable *m* den Wert 1 (≙ Münze liegt unter dem Becher). Dazu wird eine bedingte Anweisung benötigt.

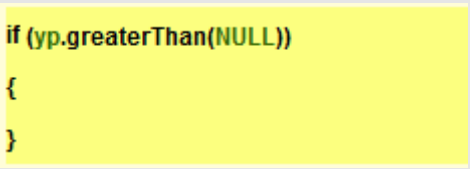
Bedingte Anweisungen einfügen:

Zunächst wird genauso vorgegangen, wie beim Einfügen einer neuen Code-Zeile:

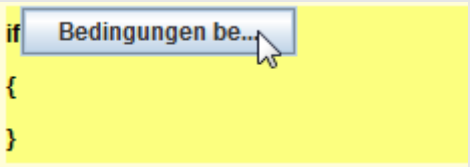


© 2014 Henrik Horstmann

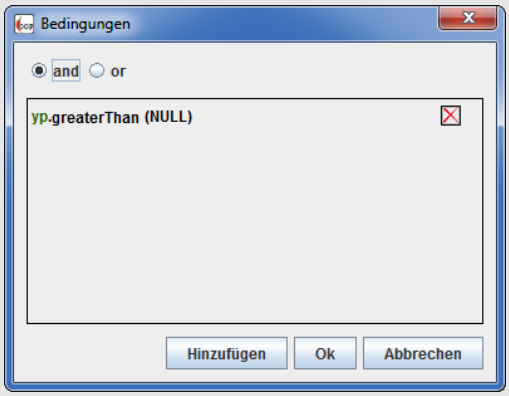
Den Eintrag *if-else einfügen* anklicken und eine bedingte Anweisung ist im Code eingefügt.



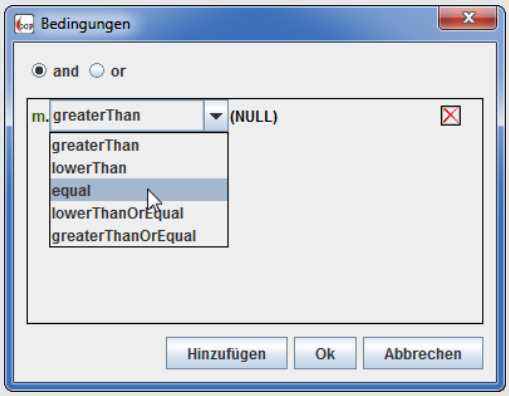
Editieren der Bedingung: Den Mauscursor auf die Bedingung positionieren:



Ein Button erscheint. Mit einem Mausklick auf den Button öffnet sich ein Dialogfenster:



Genau wie eine Codezeile lässt sich eine Bedingung bearbeiten. Den Mauszeiger auf die entsprechenden Elemente positionieren und es erscheinen Drop-Down-Listen oder Buttons:



Mit *Ok* werden die Eingaben bestätigt.

© 2014 Henrik Horstmann

Methoden zum Vergleichen:

Die Klassen *Integer* und *Float* besitzen Methoden mit denen Zahlen verglichen werden können. Die Methoden liefern alle Werte der Klasse *Boolean* zurück und können somit als Bedingungen in bedingten Anweisungen eingesetzt werden.

Hier ist eine Liste der Methoden und ihren Bedeutungen

Methode	**Beshreibung**
equal	Vergleicht zwei Zahlen. Das Ergebnis ist *TRUE* wenn die beiden Zahlen gleich sind.
greaterThan	Vergleicht zwei Zahlen. Das Ergebnis ist *TRUE* wenn die aufrufende Zahl größer als die übergebene Zahl ist.
greaterThanOrEqual	Vergleicht zwei Zahlen. Das Ergebnis ist *TRUE* wenn die aufrufende Zahl größer als die oder gleich der übergebene Zahl ist.
lowerThan	Vergleicht zwei Zahlen. Das Ergebnis ist *TRUE* wenn die aufrufende Zahl kleiner als die übergebene Zahl ist.
lowerThanOrEqual	Vergleicht zwei Zahlen. Das Ergebnis ist *TRUE* wenn die aufrufende Zahl kleiner als die oder gleich der übergebene Zahl ist.

Beispiel:

```
i = new Integer(2);
j = new Integer(5);
if (i.lowerThanOrEqual(j))
{
   // Wird ausgeführt, da die Bedingung
   // wahr ist.
}
```

Aufgabe 5:

Erweitern Sie den Konstruktor der Klasse *Muenzbecher*.

Fügen Sie am Ende des Konstruktors eine bedingte Anweisung ein. Sie soll ausgeführt werden wenn die Variable *m* den Wert 1 enthält.

© 2014 Henrik Horstmann

Im Anweisungsrumpf der bedingten Anweisung wird der Wert des Attributs *mitMuenze* auf *TRUE* gesetzt.

Hier geht es zur Lösung auf Seite 24.

So wie der Münzbecher jetzt programmiert ist, zeigt er beim Anheben des Bechers immer eine Münze an. Dies muss verhindert werden, wenn das Attribut *mitMuenze* den Wert *FALSE* enthält.

Mit einer weiteren bedingten Anweisung muss die Münze mit der Methode *hide* versteckt werden, wenn *mitMuenze* den Wert *FALSE* enthält.

Bedinge Anweisungen werden aber nur ausgeführt, wenn der Wert *TRUE* ist.

Wahrheitswerte negieren:

Die Bedingte Anweisung muss ausgeführt werden, wenn *mitMuenze* **nicht** den Wert *TRUE* enthält.

Für dieses **nicht** (negieren) besitzt die Klasse *Boolean* eine Methode:

```
Boolean b;
b = new Boolean (FALSE);
b = b.not();
```

Nach dem Methodenaufruf von *not* enthält die Variable *b* den Wert *TRUE*.

Aufgabe 6:

Erweitern Sie den Konstruktor der Klasse *Muenzbecher*.

Fügen Sie am Ende des Konstruktors eine bedingte Anweisung ein. Sie soll ausgeführt werden wenn die Variable *mitMuenze* den Wert *FALSE* enthält. Im Anweisungsrumpf der bedingten Anweisung wird der das StageObject *muenze* mit *hide* versteckt.

Testen Sie Ihr Programm.

Hier geht es zur Lösung auf Seite 25.

3.4 Der Spieler hat seinen Auftritt

Der Münzbecher funktioniert nun wie erwartet. Es kann ein Spieler kommen um sein Glück zu versuchen.

Dazu muss das Hauptprogramm in der Methode *main* der Klasse *Program* erweitert werden.

Bevor der Becher angehoben wird muss der Spieler seine Vermutung kund tun.

© 2014 Henrik Horstmann

Aufgabe 7:

Erweitern Sie die Methode *main* der Klasse *Program*.

Statt auf einen beliebigen Tastendruck des Benutzers zu warten, soll eine Abfrage erfolgen. Der Benutzer soll "j" eingeben, wenn er vermutet, dass sich eine Münze unter dem Becher befindet. Andernfalls soll der Benutzer "n" eingeben. Das Ergebnis der Abfrage soll in einer Variablen *v* gespeichert werden.

(Tipp: Eine Benutzereingabe wird mit der Methode

```
String Computer.read(String label)
```

durchgeführt)

Hier geht es zur Lösung auf Seite 26.

Zuletzt muss der Becher angehoben und überprüft werden, ob die Vermutung des Spielers richtig ist. Die Überprüfung lässt sich wie folgt formulieren:

WENN (v='j' **UND** Münze ist unter dem Becher)
ODER (v='n' **UND** Münze ist nicht unter dem Becher)
DANN hat der Spiele gewonnen
SONST hat der Spieler verloren

Um die Bedingung in HOOPLU zu realisieren müssen zunächst die beide Klammerausdrücke

(v='j' **UND** Münze ist unter dem Becher)

(v='n' **UND** Münze ist nicht unter dem Becher)

berechnet werden und die Ergebnisse in Variablen zwischen gespeichert werden.

Logische Operationen:

Die Klasse *Boolean* besitzt Methoden mit denen sich Wahrheitswerte verknüpfen lassen:

```
Boolean b1;
Boolean b2;
Boolean b3;

b3 = b1.and(b2);
// b1=FALSE, b2=FALSE => b3=FALSE
// b1=TRUE , b2=FALSE => b3=FALSE
// b1=FALSE, b2=TRUE  => b3=FALSE
// b1=TRUE , b2=TRUE  => b3=TRUE
```

© 2014 Henrik Horstmann

```
b3 = b1.or(b2);
// b1=FALSE, b2=FALSE => b3=FALSE
// b1=TRUE , b2=FALSE => b3=TRUE
// b1=FALSE, b2=TRUE  => b3=TRUE
// b1=TRUE , b2=TRUE  => b3=TRUE

b2 = b1.not();
// b1=FALSE => b2=TRUE
// b1=TRUE  => b2=FALSE
```

Strings vergleichen:

Zum Vergleichen, ob zwei Strings gleich sind, besitzt die Klasse *String* die Methode *equal*.

Beispiel:

```
String s;
Boolean b;


s = new String ("Hallo");
b = s.equal("Hallo"); // b=TRUE
b = s.equal("HALLO"); // b=FALSE
```

Feststellen, ob sich die Münze unter dem Becher befindet:

Die Methode *oeffnen* der Klasse *Muenzbecher* liefert als Ergebnis zurück, ob sih die Münze unter dem Becher befindet (=TRUE) oder nicht (=FALSE).

Aufgabe 8:

Erweitern Sie die Methode *main* der Klasse *Program*.

Speichern Sie das Ergebnis des Aufrufs

```
mb1.oeffnen();
```

in einer Variablen *m*.

Berechnen Sie den Ausdruck

(v='j' **UND** m)

und speichern das Ergebnis in einer Variablen *b1*.

Hier geht es zur Lösung auf Seite 27.

© 2014 Henrik Horstmann

Aufgabe 9:

Erweitern Sie die Methode *main* der Klasse *Program*.

Berechnen Sie den Ausdruck

(v='n' **UND NICHT** m)

und speichern das Ergebnis in einer Variablen *b1*.

NICHT m bedeutet, dass die Münze nicht unter dem Becher ist.

Hier geht es zur Lösung auf Seite 28.

Zuletzt muss die bedingte Anweisung eingefügt werden, die nun wie folgt aussieht:

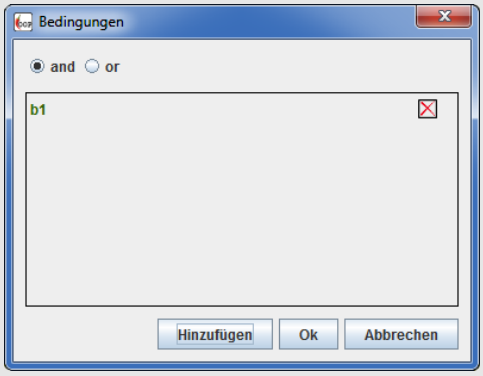
WENN b1 **ODER** b2

DANN hat der Spiele gewonnen

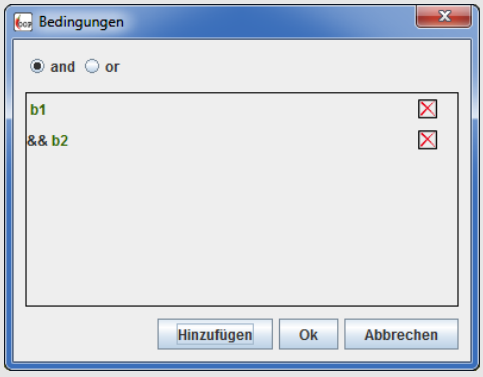
SONST hat der Spieler verloren

Mehre Bedingungen prüfen:

Ist eine bedingte Anweisung in den Code eingefügt worden, so muss das Dialogfenster zum Bearbeiten der Bedingungen geöffnet werden:



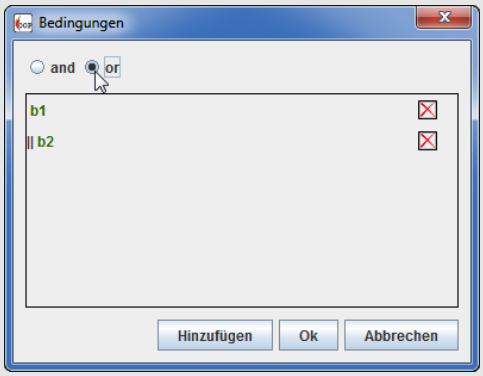
Mit einem Mausklick auf den Button Hinzufügen können weitere Bedingungen eingefügt werden:



Das **&&** bedeutet das beide Bedingungen mit **UND** verknüpft werden.

© 2014 Henrik Horstmann

In unserem Fall sollen die beiden Ausdrücke mit **ODER** verknüpft werden:



Mit der Maus den Eintrag *or* auswählen die beiden Bedingungen werden mit **||** verknüpft. **||** bedeutet **ODER**.

Aufgabe 10:

Erweitern Sie die Methode *main* der Klasse *Program*.

Fügen Sie am Ende der Methode eine bedingte Anweisung ein, die ausgeführt wird, wenn

b1 || b2 = TRUE

ist. In diesem Fall soll auf der Konsole die Meldung "Sie haben gewonnen!" ausgegeben werden.

Testen Sie Ihr Programm.

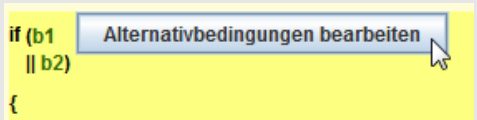
Hier geht es zur Lösung auf Seite 29.

Fehlt nur noch, dass wenn der Spieler verloren hat auch eine entsprechende Meldung ausgegeben wird.

Der SONST-Block:

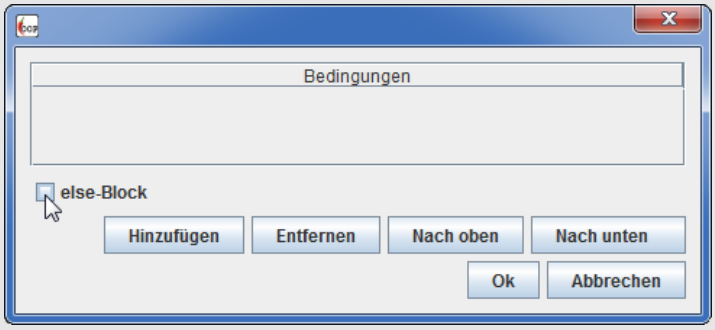
Einer bedingten Anweisung kann ein **else**-Block (else engl. für sonst) angehängt werden:

Den Mauszeiger neben die Bedingungen der bedingten Anweisung positionieren:

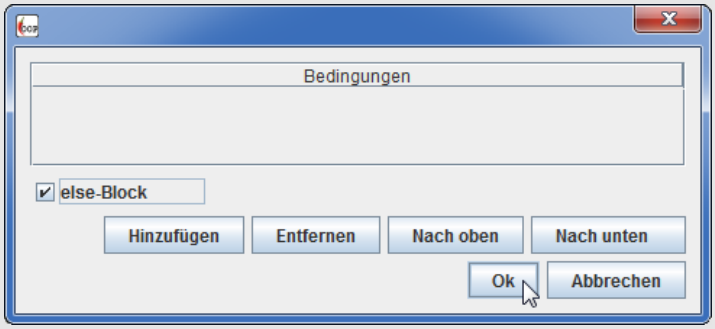


Es erscheint ein Button. Ein Mausklick auf diesen Button öffnet ein Dialogfenster:

© 2014 Henrik Horstmann



Ein Häkchen im Kontrollkästchen für else-Block setzen und mit *OK* bestätigen.



Anschließend besitzt de bedingte Anweisung einen else-Block:

```
if (b1
   || b2)
{
}
else
{

}
```

Aufgabe 11:

Erweitern Sie die Methode *main* der Klasse *Program*.

Fügen Sie in der bedingten Anweisung einen else-Block ein. Im else-Block soll die Meldung "Sie haben leider verloren!" ausgegeben werden.

Testen Sie Ihr Programm.

Hier geht es zur Lösung auf Seite 30.

Endlich ist das Spiel fertig.

© 2014 Henrik Horstmann

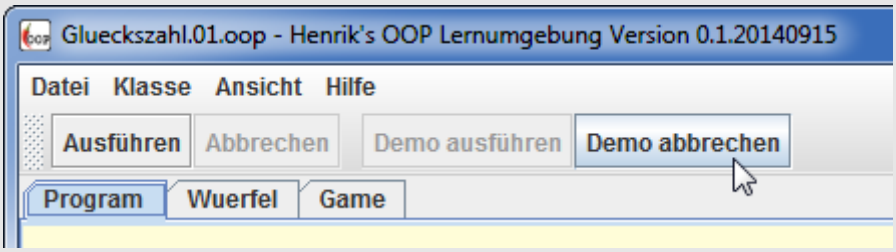
4 Übungen

4.1 Glückszahl

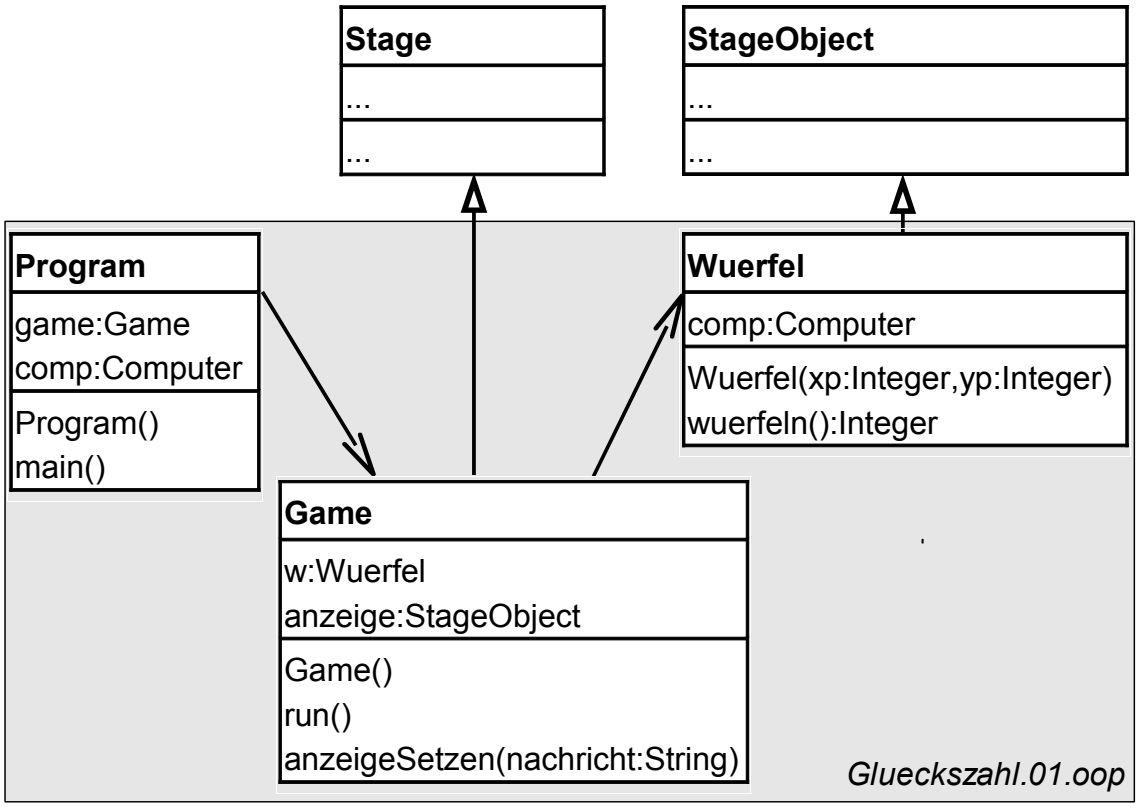
In dem folgenden Spiel bekommt der Spieler einen Würfel, mit dem er einen Wurf nach dem anderen macht. Jedes mal, wenn der Spieler eine 6 würfelt, dann hat er gewonnen.

Das Spiel können Sie mit *Gluekszahl.01.demo.oop* testen.

Das Programm *Gluekszahl.01.demo.oop* wird in einer Endlosschleife ausgeführt. Um es zu beenden muss in der Entwicklungsumgebung der Button *Demo Abbrechen* angeklickt werden.



Erstellen Sie ein Programm, dass das oben beschriebene Spiel realisiert. Öffnen Sie dazu das Projekt *Glueckszahl.01.oop.* Es enthält schon einige Klassen, die für das Programm benötigt werden:



Tipps und Hinweise zu der Übung:

Hinweis 1:

Ihren Programmcode müssen Sie in der Methode ***run()*** der Klasse ***Game*** einfügen.

Hinweis 2:

Die Klasse ***Wuerfel*** besitzt eine Methode ***wuerfeln*** mit der Sie eine Zufallszahl im Bereich 1 bis 6 erzeugen. Außerdem wird der Würfel auf der Stage angezeigt.

Hinweis 3:

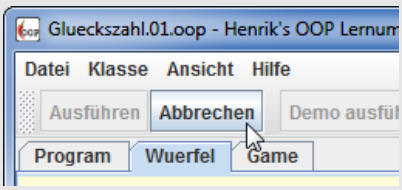
Die Klasse *Game* besitzt eine Methode

```
void Game.anzeigeSetzen(String nachricht)
```

mit der eine Nachricht im Programmfenster ausgegeben werden kann.

Hinweis 4:

Das Programm *Gluekszahl.01.oop* wird in einer Endlosschleife ausgeführt. Um es zu beenden muss in der Entwicklungsumgebung der Button *Abbrechen* angeklickt werden.



4.2 Glückszahl mit Zähler

Öffnen Sie dazu das Projekt *Glueckszahl.02.oop.* Es enthält schon einige Klassen, die für das Programm benötigt werden. Erweitern Sie das Programm aus Abschnitt 4.1 um einen Zähler, der jedes Mal um eins erhöht wird, wenn eine 6 gewürfelt wurde. Der Zähler soll im Programmfenster angezeigt werden.

Das Spiel können Sie mit *Gluekszahl.02.demo.oop* testen.

Tipps und Hinweise zu der Übung:

Hinweis 1:

Die Klasse *Game* besitzt eine Methode

```
void Game.punkteAnzeigen(Integer punkte)
```

die den Punktestand im Programmfenster ausgibt.

Tipp 1:

Der Zähler muss als Attribut angelegt werden, da er das ganze Spiel erhalten bleiben muss.

© 2014 Henrik Horstmann

4.3 Glückszahlen

Kopieren Sie die Projektdatei aus Abschnitt 4.2 und benennen Sie sie *Gluekszahlen.01.oop*. Erweitern Sie das Programm so, dass der Zähler jedes Mal erhöht wird, wenn die gewürfelte Zahl im Bereich von 3 bis 5 liegt.

Das Spiel können Sie mit *Gluekszahlen.01.demo.oop* testen.

4.4 Keine krummen Glückszahlen

Kopieren Sie die Projektdatei aus Abschnitt 4.3 benennen Sie sie *Gluekszahlen.02.oop*. Erweitern Sie das Programm so, dass der Zähler jedes Mal erhöht wird, wenn die gewürfelte Zahl gerade ist.

Das Spiel können Sie mit *Gluekszahlen.02.demo.oop* testen.

4.5 Pasch

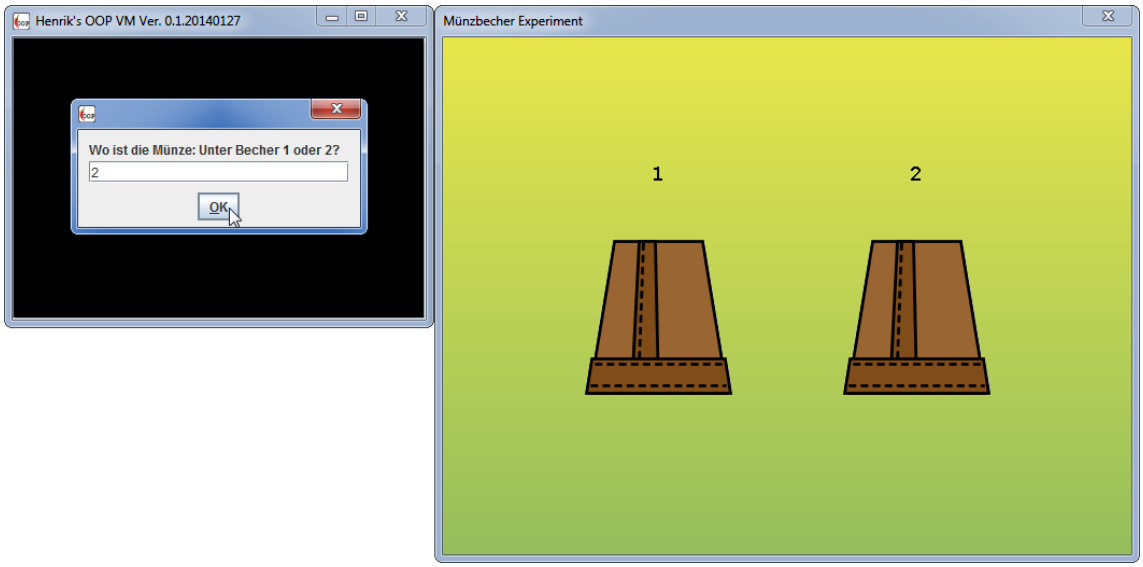
Kopieren Sie die Projektdatei aus Abschnitt 4.2 und benennen Sie sie *Pasch.oop*. Erweitern Sie das Programm so, dass es zwei Würfel hat. Der Zähler soll immer dann erhöht werden, wenn ein Pasch (beide Würfel haben die gleiche Augenzahl) gewürfelt wurde.

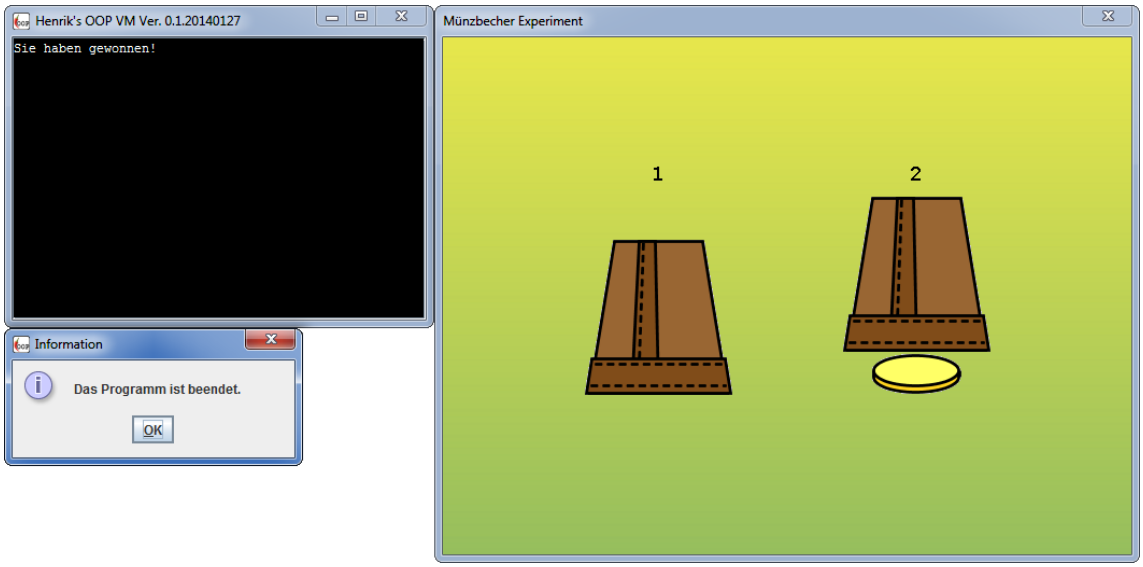
Das Spiel können Sie mit *Pasch.demo.oop* testen.

4.6 Unter welchem Becher ist die Münze?

Im folgenden sollen Sie eine Variante des Münzbecherspiels programmieren.

Es gibt zwei Münzbecher. Unter einem Becher befindet sich die Münze (garantiert). Der Spieler soll raten unter welchem der beiden Becher sich die Münze befindet. Dieser Becher wird dann angehoben und nach geschaut. Befindet sich die Münze unter dem Becher, so hat der Spieler gewonnen.





Das fertige Programm können Sie mit *Muenzbecher.02.demo.oop* testen.

Tipps zu der Übung:

Tipp 1:

Ändern Sie den Konstruktor der *Muenzbecher* Klasse so ab, dass beim erzeugen eines Objektes dem Münzbecher mitgeteilt wird, ob sich die Münze unter dem Becher befindet. Der Münzbecher berechnet somit nicht mehr selbst mit einer Zufallszahl, ob er eine Münze enthält.

Tipp 2:

Erzeugen Sie in der *main* Methode der Klasse *Program* eine Zufallszahl im Bereich 1 und 2. 1 bedeutet unter dem linken Becher befindet sich die Münze und 2 bedeutet die Münze befindet sich unter dem rechten Becher.

Tipp 3:

Sie müssen bedingte Anweisungen in einander verschachteln.

ENDE

© 2014 Henrik Horstmann

5 Lösungen zu den Aufgaben

5.1 Lösung zu Aufgabe 1

```
public class Muenzbecher
{
   // Anfang Attribute
   private StageObject titelFeld;
   private StageObject muenze;
   private StageObject becher;
   private Boolean mitMuenze;
   // Ende Attribute


   public Muenzbecher (Stage stage,Integer xp,Integer yp,String titel)
   {
      // Anfang Variablen
      Integer cy;
      // Ende Variablen

      cy = yp.sub(120);
      titelFeld = new StageObject(xp,cy,100);
      titelFeld.setText(titel,20);
      stage.addStageObject(titelFeld);
      muenze = new StageObject("Münze",xp,yp,"Muenze.png",100);
      stage.addStageObject(muenze);
      becher = new StageObject(xp,yp,100);
      becher.addSprite("geschlossen","Becher.geschlossen.png");
      becher.addSprite("offen","Becher.offen.png");
      stage.addStageObject(becher);
      mitMuenze = new Boolean(FALSE);
   }
}
```

Zurück zur Aufgabe auf Seite 4.

© 2014 Henrik Horstmann

5.2 Tipp zu Aufgabe 2

Tipp zum öffnen des Bechers:

Das StageObject *becher* enthält zwei Grafiken (Sprites). Eine (Nr. 0) für den Becher im geschlossenen und eine zweite (Nr. 1) für den Becher im geöffneten Zustand.

Um den Becher anzuheben muss lediglich die Grafik für den Becher im geöffneten Zustand ausgewählt werden. Dazu hat StageObject eine Methode *setSprite*:

```
becher.setSprite(1);
```

Zurück zur Aufgabe auf Seite 5.

© 2014 Henrik Horstmann

5.3 Lösung zu Aufgabe 2

```
public class Muenzbecher
{
   // Anfang Attribute
   private StageObject titelFeld;
   private StageObject muenze;
   private StageObject becher;
   private Boolean mitMuenze;
   // Ende Attribute


   public Muenzbecher (Stage stage,Integer xp,Integer yp,String titel)
   {
      // Anfang Variablen
      Integer cy;
      // Ende Variablen

      cy = yp.sub(120);
      titelFeld = new StageObject(xp,cy,100);
      titelFeld.setText(titel,20);
      stage.addStageObject(titelFeld);
      muenze = new StageObject("Münze",xp,yp,"Muenze.png",100);
      stage.addStageObject(muenze);
      becher = new StageObject(xp,yp,100);
      becher.addSprite("geschlossen","Becher.geschlossen.png");
      becher.addSprite("offen","Becher.offen.png");
      stage.addStageObject(becher);
      mitMuenze = new Boolean(FALSE);
   }

   public Boolean oeffnen ()
   {
      // Anfang Variablen
      // Ende Variablen

      becher.setSprite(1);
      return mitMuenze;
   }
}
```

Zurück zur Aufgabe auf Seite 5.

© 2014 Henrik Horstmann

5.4 Lösung zu Aufgabe 3

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Computer comp;
      Stage stage;
      Muenzbecher mb1;
      // Ende Variablen

      stage = new Stage("Münzbecher Experiment",640,480);
      stage.loadBackground("BG_green.png");
      mb1 = new Muenzbecher(stage,320,240,"");
      stage.start();
      comp = new Computer();
      comp.print("Becher öffnen: beliebige Taste drücken.");
      stage.waitUntilKeyPressed();
      mb1.oeffnen();
   }
}
```

Zurück zur Aufgabe auf Seite 5.

© 2014 Henrik Horstmann

5.5 Lösung zu Aufgabe 4

```
public class Muenzbecher
{
   // Anfang Attribute
   private StageObject becher;
   private StageObject muenze;
   private StageObject titelFeld;
   private Boolean mitMuenze;
   // Ende Attribute


   public Muenzbecher (Stage stage,Integer xp,Integer yp,String titel)
   {
      // Anfang Variablen
      Integer m;
      Computer comp;
      Integer cy;
      // Ende Variablen

      cy = yp.sub(120);
      titelFeld = new StageObject(xp,cy,100);
      titelFeld.setText(titel,20);
      stage.addStageObject(titelFeld);
      muenze = new StageObject("Münze",xp,yp,"Muenze.png",100);
      stage.addStageObject(muenze);
      becher = new StageObject(xp,yp,100);
      becher.addSprite("geschlossen","Becher.geschlossen.png");
      becher.addSprite("offen","Becher.offen.png");
      stage.addStageObject(becher);
      mitMuenze = new Boolean(FALSE);
      comp = new Computer();
      m = comp.rand(0,1);
   }

   public Boolean oeffnen ()
   {
      // Anfang Variablen
      // Ende Variablen

      becher.setSprite(1);
      return mitMuenze;
   }
}
```

Zurück zur Aufgabe auf Seite 6.

© 2014 Henrik Horstmann

5.6 Lösung zu Aufgabe 5

```
public class Muenzbecher
{
   // Anfang Attribute
   private StageObject titelFeld;
   private StageObject muenze;
   private StageObject becher;
   private Boolean mitMuenze;
   // Ende Attribute


   public Muenzbecher (Stage stage,Integer xp,Integer yp,String titel)
   {
      // Anfang Variablen
      Integer m;
      Computer comp;
      Integer cy;
      // Ende Variablen

      cy = yp.sub(120);
      titelFeld = new StageObject(xp,cy,100);
      titelFeld.setText(titel,20);
      stage.addStageObject(titelFeld);
      muenze = new StageObject("Münze",xp,yp,"Muenze.png",100);
      stage.addStageObject(muenze);
      becher = new StageObject(xp,yp,100);
      becher.addSprite("geschlossen","Becher.geschlossen.png");
      becher.addSprite("offen","Becher.offen.png");
      stage.addStageObject(becher);
      mitMuenze = new Boolean(FALSE);
      comp = new Computer();
      m = comp.rand(0,1);
      if (m.equal(1))
      {
         mitMuenze = new Boolean(TRUE);
      }
   }

   public Boolean oeffnen ()
   {
      // Anfang Variablen
      // Ende Variablen

      becher.setSprite(1);
      return mitMuenze;
   }
}
```

Zurück zur Aufgabe auf Seite 8.

© 2014 Henrik Horstmann

5.7 Lösung zu Aufgabe 6

```
public class Muenzbecher
{
   // Anfang Attribute
   private StageObject titelFeld;
   private StageObject muenze;
   private StageObject becher;
   private Boolean mitMuenze;
   // Ende Attribute


   public Muenzbecher (Stage stage,Integer xp,Integer yp,String titel)
   {
      // Anfang Variablen
      Integer m;
      Computer comp;
      Integer cy;
      // Ende Variablen

      cy = yp.sub(120);
      titelFeld = new StageObject(xp,cy,100);
      titelFeld.setText(titel,20);
      stage.addStageObject(titelFeld);
      muenze = new StageObject("Münze",xp,yp,"Muenze.png",100);
      stage.addStageObject(muenze);
      becher = new StageObject(xp,yp,100);
      becher.addSprite("geschlossen","Becher.geschlossen.png");
      becher.addSprite("offen","Becher.offen.png");
      stage.addStageObject(becher);
      mitMuenze = new Boolean(FALSE);
      comp = new Computer();
      m = comp.rand(0,1);
      if (m.equal(1))
      {
         mitMuenze = new Boolean(TRUE);
      }
      if (mitMuenze.not())
      {
         muenze.hide();
      }
   }

   public Boolean oeffnen ()
   {
      // Anfang Variablen
      // Ende Variablen

      becher.setSprite(1);
      return mitMuenze;
   }
}
```

Zurück zur Aufgabe auf Seite 9.

© 2014 Henrik Horstmann

5.8 Lösung zu Aufgabe 7

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      String v;
      Computer comp;
      Stage stage;
      Muenzbecher mb1;
      // Ende Variablen

      stage = new Stage("Münzbecher Experiment",640,480);
      stage.loadBackground("BG_green.png");
      mb1 = new Muenzbecher(stage,320,240,"");
      stage.start();
      comp = new Computer();
      v = comp.read("Münze unterm Becher: 'j' = ja, 'n'=nein");
      mb1.oeffnen();
   }
}
```

Zurück zur Aufgabe auf Seite 10.

© 2014 Henrik Horstmann

5.9 Lösung zu Aufgabe 8

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Boolean m;
      Boolean b1;
      Computer comp;
      String v;
      Stage stage;
      Muenzbecher mb1;
      // Ende Variablen

      stage = new Stage("Münzbecher Experiment",640,480);
      stage.loadBackground("BG_green.png");
      mb1 = new Muenzbecher(stage,320,240,"");
      stage.start();
      comp = new Computer();
      v = comp.read("Münze unterm Becher: 'j' = ja, 'n'=nein");
      m = mb1.oeffnen();
      b1 = v.equal("j");
      b1 = b1.and(m);
   }
}
```

Zurück zur Aufgabe auf Seite 11.

© 2014 Henrik Horstmann

5.10 Lösung zu Aufgabe 9

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Boolean m;
      Boolean b2;
      Computer comp;
      String v;
      Boolean b1;
      Stage stage;
      Muenzbecher mb1;
      // Ende Variablen

      stage = new Stage("Münzbecher Experiment",640,480);
      stage.loadBackground("BG_green.png");
      mb1 = new Muenzbecher(stage,320,240,"");
      stage.start();
      comp = new Computer();
      v = comp.read("Münze unterm Becher: 'j' = ja, 'n'=nein");
      m = mb1.oeffnen();
      b1 = v.equal("j");
      b1 = b1.and(m);
      m = m.not();
      b2 = v.equal("n");
      b2 = b2.and(m);
   }
}
```

Zurück zur Aufgabe auf Seite 12.

© 2014 Henrik Horstmann

5.11 Lösung zu Aufgabe 10

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Boolean m;
      Boolean b2;
      Computer comp;
      String v;
      Boolean b1;
      Stage stage;
      Muenzbecher mb1;
      // Ende Variablen

      stage = new Stage("Münzbecher Experiment",640,480);
      stage.loadBackground("BG_green.png");
      mb1 = new Muenzbecher(stage,320,240,"");
      stage.start();
      comp = new Computer();
      v = comp.read("Münze unterm Becher: 'j' = ja, 'n'=nein");
      m = mb1.oeffnen();
      b1 = v.equal("j");
      b1 = b1.and(m);
      m = m.not();
      b2 = v.equal("n");
      b2 = b2.and(m);
      if (b1 || b2)
      {
         comp.print("Sie haben gewonnen!");
      }
   }
}
```

Zurück zur Aufgabe auf Seite 13.

© 2014 Henrik Horstmann

5.12 Lösung zu Aufgabe 11

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Boolean m;
      Boolean b2;
      Computer comp;
      String v;
      Boolean b1;
      Stage stage;
      Muenzbecher mb1;
      // Ende Variablen

      stage = new Stage("Münzbecher Experiment",640,480);
      stage.loadBackground("BG_green.png");
      mb1 = new Muenzbecher(stage,320,240,"");
      stage.start();
      comp = new Computer();
      v = comp.read("Münze unterm Becher: 'j' = ja, 'n'=nein");
      m = mb1.oeffnen();
      b1 = v.equal("j");
      b1 = b1.and(m);
      m = m.not();
      b2 = v.equal("n");
      b2 = b2.and(m);
      if (b1 || b2)
      {
         comp.print("Sie haben gewonnen!");
      }
      else
      {
         comp.print("Sie haben leider verloren!");
      }
   }
}
```

Zurück zur Aufgabe auf Seite 14.

© 2014 Henrik Horstmann