OOP

Wenn Computer Erben

Henrik Horstmann

2. Dezember 2015

Inhaltsverzeichnis

1 Bedeutung der Symbole....1
2 Die Erben....2
2.1 Das Beispiel....2
2.2 Gemeinsamkeiten....2
2.3 Unterschiede....3
3 Lösung durch Erbschaften....3
4 Vererbung programmieren....4
4.1 Die Klasse Mitarbeiter....4
4.2 Der Facharbeiter....7
4.3 Die Erbschaft antreten....7
4.4 Die Methoden des Facharbeiters....9
4.5 Alles super....10
4.6 Einen Facharbeiter einstellen....10
4.7 Die Firma wächst....12
4.8 Übung....16
5 Überschreiben von Methoden....17
5.1 Der Zug....18
5.2 Der Wagon....18
5.3 Der Personenwagon....20
5.4 Überschreiben von Methoden....20
5.5 Ohne Kenntnis einen Personenwagon verwenden....22
6 Übungen....23
6.1 Ein vollständiger Personenzug....23
6.2 Noch mehr Züge....23
7 Lösungen zu den Aufgaben....27
7.1 Lösung zu Aufgabe 1....27
7.2 Lösung zu Aufgabe 2....29
7.3 Lösung zu Aufgabe 3....30
7.4 Lösung zu Aufgabe 4....31
7.5 Lösung zu Aufgabe 5....32

7.6 Lösung zu Aufgabe 6.....33
7.7 Lösung zu Aufgabe 7.....34
7.8 Lösung zu Aufgabe 8.....35
7.9 Lösung zu Aufgabe 9.....36
7.10 Lösung zu Aufgabe 11.....37
7.11 Lösung zu Aufgabe 12.....38
7.12 Lösung zu Aufgabe 13.....39
7.13 Lösung zu Aufgabe 14.....40
7.14 Lösung zu Aufgabe 15.....41
7.15 Lösung zu Aufgabe 16.....42

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 Die Erben

2.1 Das Beispiel

Als Beispiel soll uns die Belegschaft einer Firma dienen. In der folgenden Tabelle wird die Belegschaft dargestellt:

Belegschaft		
Facharbeiter		
	Fritz Müller	Verdienst berechnet sich aus den Arbeitsstunden und einem Stundenlohn.
	Silke Schmitt	
	Eckart Beutenmüller	
Angestellte		
	Volker Blaufuss	Verdienst ist ein festes Gehalt von 4000 €.
	Simone Fischer	
Manager		
	Dr. Mattes Friedbacher	Verdienst setzt sich aus einem festen Grundgehalt zuzüglich einem prozentualen Anteil vom Gewinn zusammen.

2.2 Gemeinsamkeiten

Gemeinsame Eigenschaften aller Mitarbeiter sind:

- jeder Mitarbeiter hat einen Namen
- jeder Mitarbeiter bekommt am Monatsende seinen Verdienst ausgezahlt

Ein Mitarbeiter kann durch eine Klasse *Mitarbeiter* repräsentiert werden:

Mitarbeiter
name:String
Mitarbeiter() setzeName(name:String) auszahlen(verdienst:Float)

Der auszuzahlende Verdienst muss allerdings außerhalb der Klasse berechnet werden. Wünschenswert ist jedoch, wenn die Klasse das selbst erledigen könnte.

© 2015 Henrik Horstmann

2.3 Unterschiede

Allerdings gibt es bei den Mitarbeitern auch Unterschiede:

- der Verdienst berechnet sich bei Facharbeitern, Angestellten und Managern unterschiedlich.

Daraus ergeben sich drei verschiedene Klassen:

Facharbeiter
name:String
Facharbeiter() setzeName(name:String) auszahlen(stunden:Integer, stundenLohn:Float)

Angestellter
name:String
Angestellter() setzeName(name:String) auszahlen()

Manager
name:String
Manager() setzeName(name:String) auszahlen(prozent:Float, gewinn:Float)

Der Nachteil dieser Lösung ist, dass die Gemeinsamkeiten der drei Klassen (z.B. *setzeName(name:String)*) in jeder Klasse separat programmiert werden müssen, was zu redundantem Code führt.

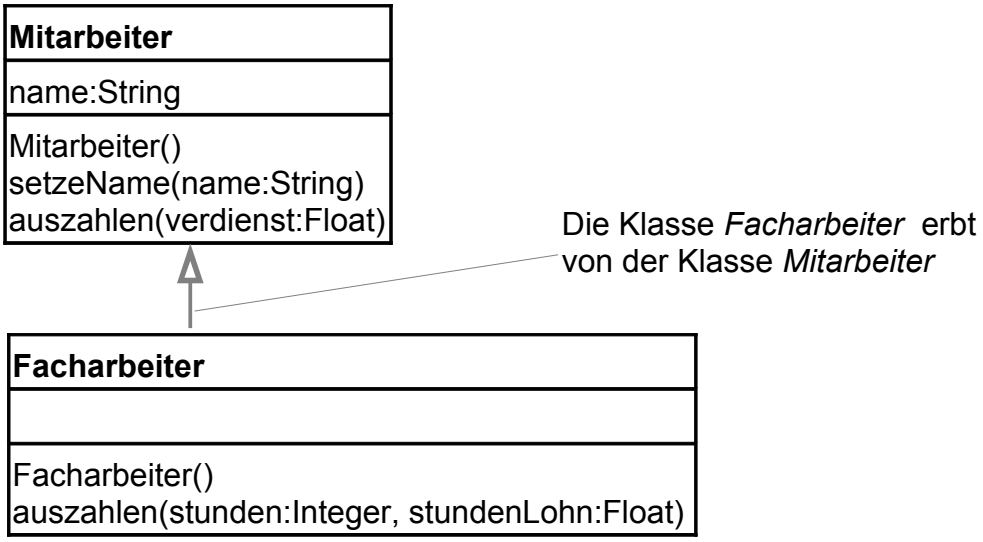
3 Lösung durch Erbschaften

Um das Problem, dass im vorhergehenden Abschnitt beschrieben ist zu lösen können Klassen voneinander Erben. In einer Basisklasse werden alle Gemeinsamkeiten implementiert:

Mitarbeiter
name:String
Mitarbeiter() setzeName(name:String) auszahlen(verdienst:Float)

© 2015 Henrik Horstmann

Für die einzelnen Gruppen von Mitarbeitern werden spezialisierte Klassen eingefügt, die alle Eigenschaften der Klasse *Mitarbeiter* "erben".



Die Erbschaft bedeutet, dass die Klasse *Facharbeiter* ebenfalls über Methoden

- *setzeName(name:String)* und
- *auszahlen(verdienst:Float)*

verfügt.

Die Klasse *Mitarbeiter* ist die **Superklasse** der Klasse *Facharbeiter.*

4 Vererbung programmieren

In einem neuen Projekt *company.oop* soll das obige Beispiel umgesetzt werden.

4.1 Die Klasse *Mitarbeiter*

Um später die Mitarbeiter grafisch darstellen zu können erhält die Klasse weitere Attribute und Methoden:

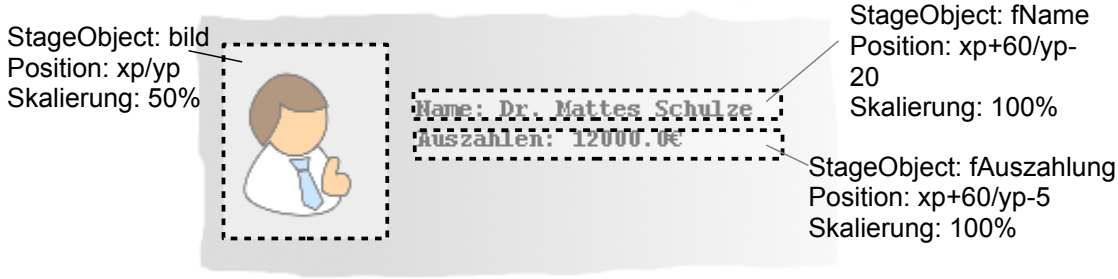
Mitarbeiter
bild:StageObject fName:StageObject fAuszahlen:StageObject
Mitarbeiter() zurStageHinzufuegen(stage:Stage, xp:Integer, yp:Integer) setzeBild(bildName:String) setzeName(name:String) auszahlen(verdienst:Float)

Beschreibung der Klasse Mitarbeiter

Mitarbeiter()
Der Konstruktor hat keine Aufgaben.

zurStageHinzufuegen(stage:Stage, xp:Integer, yp:Integer)	
stage	Bühne auf der, der Mitarbeiter dargestellt werden soll.
xp	Horizontale Postion des Mitarbeiters auf der Bühne.
yp	Vertikale Postion des Mitarbeiters auf der Bühne.

Auf der Bühne sollen drei StageObjekte platziert werden:



Der Text soll linksbündig sein (siehe Textbox „Text linksbündig ausrichten“)
Hinweis: Die StageObjekte enthalten noch keine Grafiken!

Text linksbündig ausrichten:

Um einen Text in einem StageObject linksbündig auszurichten steht folgende Methode zur Verfügung:

```
void setTextAlignmentLeft()
```

Entsprechend kann mit den Methoden

```
void setTextAlignmentCenter()
void setTextAlignmentRight()
```

ein Text zentriert oder rechtsbündig ausgerichtet werden.

setzeBild(bildName:String)	
bildName	Dateiname der Grafik die für den Mitarbeiter angezeigt werden soll.

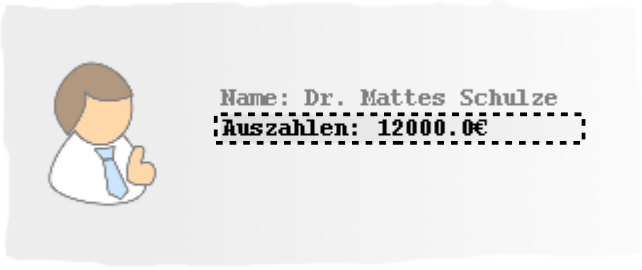
Die Grafik *bildName* wird in das StageObject für das Bild eingefügt.

setzeName(name:String)	
name	Name des Mitarbeiters.

Gibt dem Mitarbeiter einen Namen.
Dazu wird in dem Feld *fName* der Text „Name: <name>“ gesetzt.

auszahlen(verdienst:Float)	
verdienst	Verdienst des Mitarbeiters.

Bestimmt den Verdienst des Metarbeiters.
Dazu wird in dem Feld *fAuszahlen* der Text „Auszahlen: <verdienst>“ gesetzt.



© 2015 Henrik Horstmann

Aufgabe 1:

Erstellen Sie in dem Projekt *company.oop* die oben beschriebene Klasse Mitarbeiter.

Hier geht es zur Lösung auf Seite 27.

4.2 Der Facharbeiter

Der Facharbeiter ist ein spezialisierter Mitarbeiter des Unternehmens. Im Programm zeichnet er sich vor allem dadurch aus, dass sein Verdienst auf der Arbeitszeit und einem Stundenlohn basiert.

Um später den Facharbeiter grafisch darstellen zu können erhält die Klasse weitere Methoden:

Facharbeiter
Facharbeiter() setzeBild() auszahlen(stunden:Integer, stundenlohn:Float)

4.3 Die Erbschaft antreten

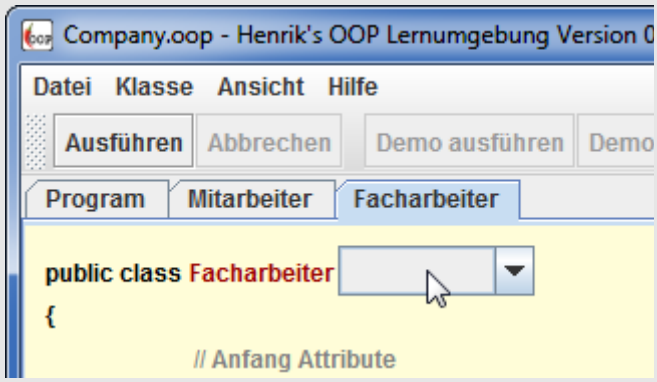
Damit die Klasse *Facharbeiter* alle Attribute und Methoden von der Klasse *Mitarbeiter* erbt, muss dies dem Programm mitgeteilt werden:

Wie Klassen erben:

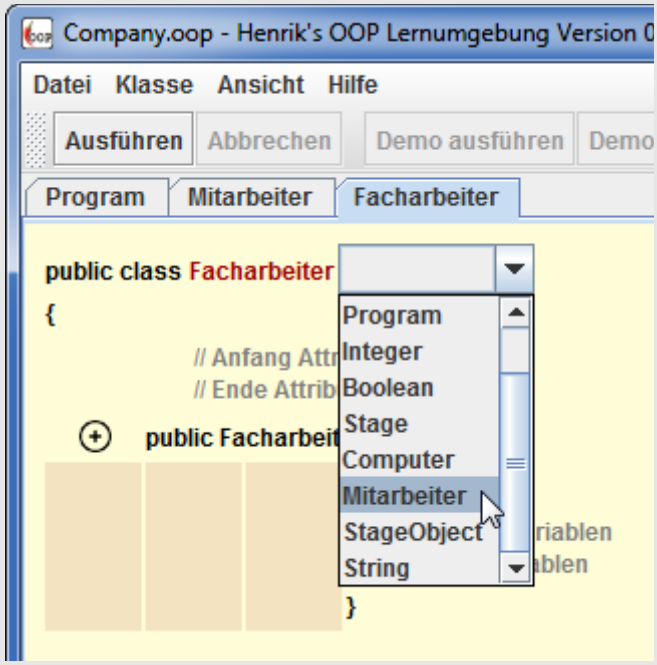
```
public class Facharbeiter extends Mitarbeiter
{
    ...
```

Extends (eng. e*rweitern*) bedeutet, dass die Klasse *Facharbeiter* alle Attribute und Methoden der Klasse *Mitarbeiter* erbt.

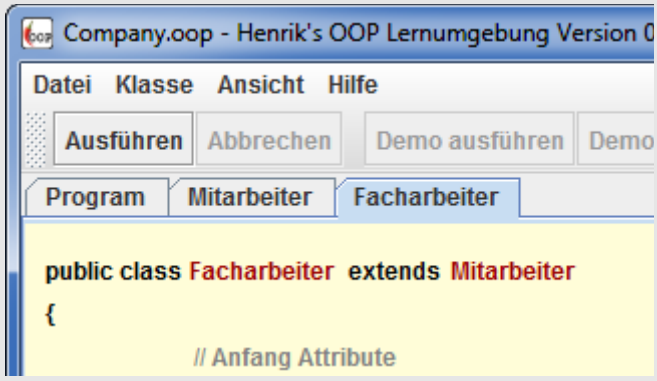
In *Hooplu* wird das Schlüsselwort ***extends*** eingefügt, in dem der Mauszeiger direkt nach dem Klassennamen positioniert wird:



Eine Dropdown-Liste erscheint. Aus dieser muss die Klasse ausgewählt werden, von der abgeleitet werden soll:



Im Anschluss sieht die erste Zeile in der Klassenbeschreibung wie folgt aus:



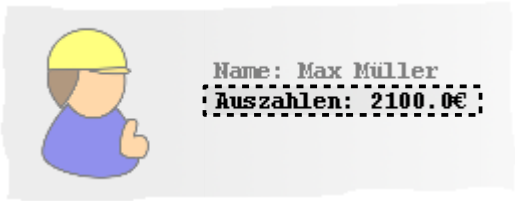
Nun erbt die Klasse *Facharbeiter* alle Attribute und Methoden der Klasse *Mitarbeiter*.

4.4 Die Methoden des Facharbeiters

Beschreibung der Klasse Mitarbeiter

Facharbeiter()
Der Konstruktor hat keine Aufgaben.

setzeBild()
Der Facharbeiter soll auf der Bühne mit folgendem Bild repräsentiert werden: *Facharbeiter.png*

auszahlen(stunden:Integer,stundenlohn:Float)	
stunden	Anzahl der Stunden, die der Facharbeiter gearbeitet hat.
stundenlohn	Lohn, der je gearbeitete Stunde an den Facharbeiter ausgezahlt wird.
Berechnet den Verdienst des Facharbeiters $(= stundenlohn \cdot stunden)$ und zeigt diesen in dem Feld *fAuszahlen* an. 	

4.5 Alles super

Um das Bild des Facharbeiters anzuzeigen muss die Methode der Klasse *Facharbeiter* die Methode *setzeBild(bild:String)* der **Superklasse** *Mitarbeiter* aufrufen.

Aufrufen von Methoden der Superklasse:

Die Methoden der Superklasse werden durch über das Schlüsselwort **super** aufgerufen:

```
super.setzeBild("Facharbeiter.jpg");
```

Aufgabe 2:

Erstellen Sie in der Klasse *Facharbeiter* eine Methode *setzeBild()*. Die Methode soll lediglich die Methode *setzeBild(bild:String)* der Superklasse aufrufen um das Bild für Facharbeiter anzuzeigen.

Hier geht es zur Lösung auf Seite 29.

Aufgabe 3:

Erstellen Sie in der Klasse Facharbeiter eine Methode

auszahlen(stunden:Integer,stundenlohn:Float).

Die Methode soll

1. den Verdienst des Facharbeiters $(=stundenlohn \cdot stunden)$ berechnen.
2. mit einem Aufruf der Methode *auszahlen(verdienst:Float)* der Superklasse *Mitarbeiter* den Verdienst im Feld *fAuszahlen* anzeigen.

Hier geht es zur Lösung auf Seite 30.

4.6 Einen Facharbeiter einstellen

In diesem Abschnitt soll ein Objekt der Klasse *Facharbeiter* erzeugt und angezeigt werden.

Aufgabe 4:

Alle folgenden Änderungen werden an der Klasse ***Program*** in der Methode ***main()*** vorgenommen.

Erzeugen Sie eine *Stage* mit dem Namen "Company" und der Größe von 640×480 Pixel.

Fügen Sie der *Stage* das Hintergrundbild *BG.GW.png* hinzu.

Erzeugen Sie ein neues Objekt der Klasse *Facharbeiter*. Mit der Methode *zurStageHinzufuegen(stage:Stage,xp:Integer,yp:Integer)* platzieren Sie den Facharbeiter auf der *Stage*. Die Position des Facharbeiters auf der *Stage* ist (50,50).

Rufen Sie die Methode *setzeBild()* und *setzeNamen(name:String)* mit "Max Müller" des Facharbeiters auf.

Rufen Sie die Methode *start()* der *Stage* auf und testen Sie Ihr Programm. Die *Stage* sollte wie folgt aussehen:

Hier geht es zur Lösung auf Seite 31.

Im nächsten Schritt soll die Auszahlung erfolgen.

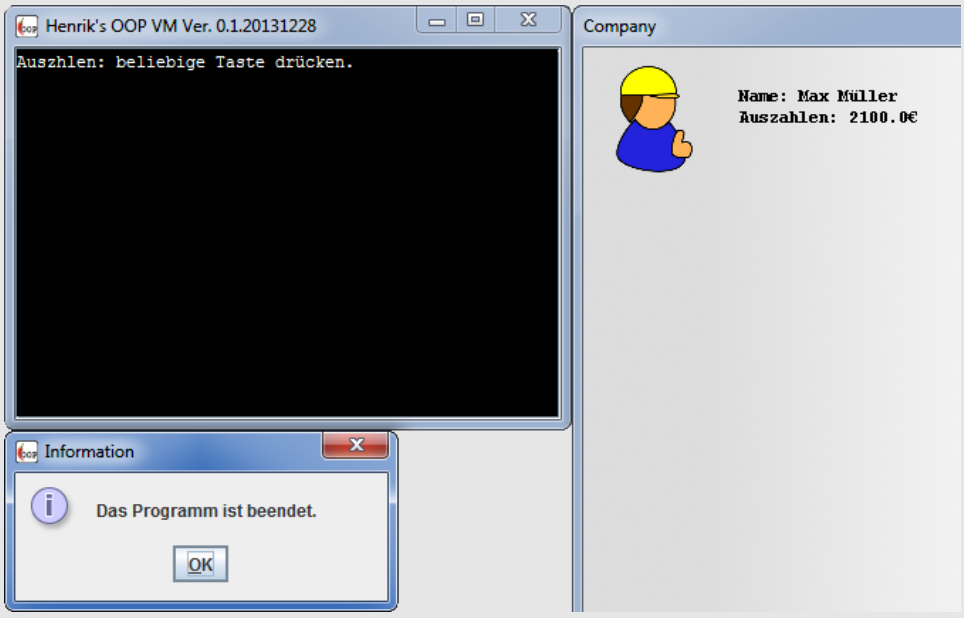
Aufgabe 5:

Das Programm soll auf der Konsole den Text "Auszahle: beliebige Taste drücken" ausgeben.

Das Programm wartet auf darauf, dass der Benutzer eine Taste betätigt.

Anschließend wird der Verdienst des Facharbeiters angezeigt. Der Facharbeiter soll einen Stundenlohn von 15,00 € erhalten. Es wird davon ausgegangen, dass der Facharbeiter 140 Stunden gearbeitet hat.

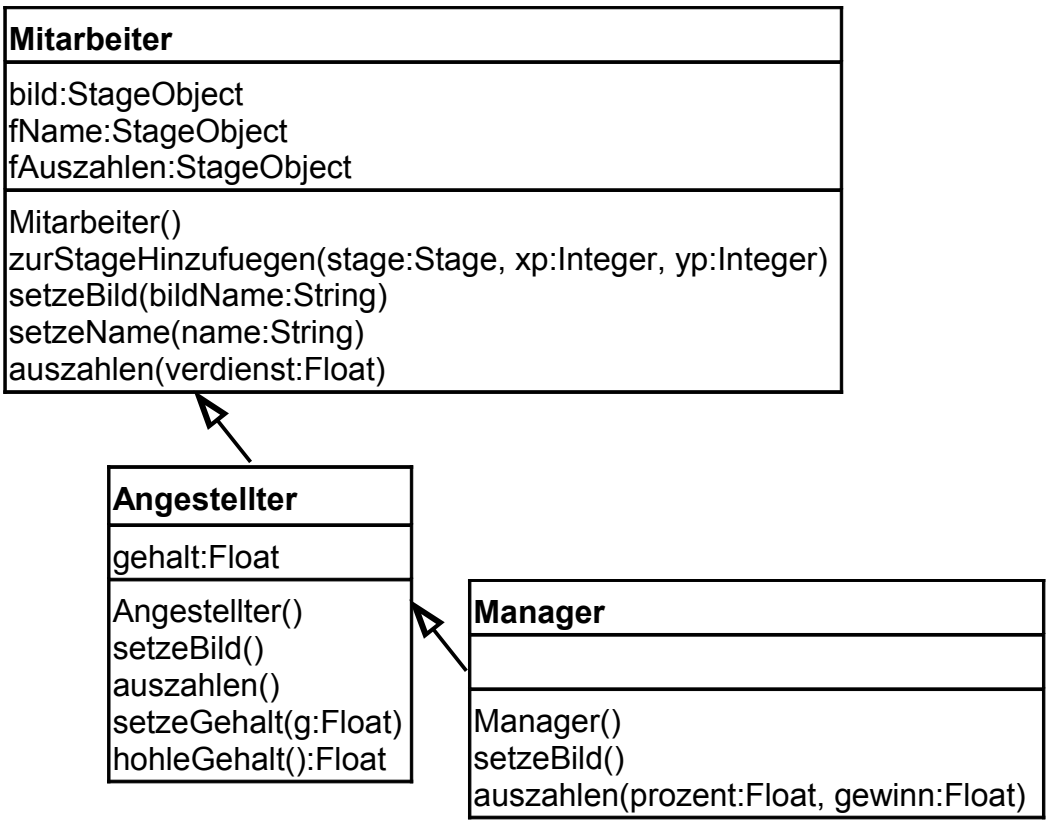
Das Ergebnis sollte wie folgt aussehen:



Hier geht es zur Lösung auf Seite 32.

4.7 Die Firma wächst

Als nächstes sollen entsprechende Klassen für Angestellte und Manager implementiert werden.



© 2015 Henrik Horstmann

Beschreibung der Klasse Angestellter

Angestellter()
Der Konstruktor hat keine Aufgaben.

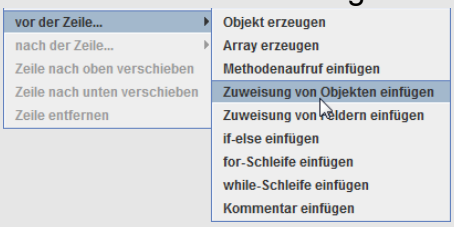
setzeBild()
Der Angestellte soll auf der Bühne mit folgendem Bild repräsentiert werden: *Angestellter.png*

auszahlen()
Der Verdienst des Angestellten ergibt sich aus einem Festgehalt von 4000 €. Die Methode zeigt den Verdienst in dem Feld *fAuszahlen* an.

setzeGehalt(g:Float)	
g	Das Gehalt eines Angestellten.
Das Gehalt, das mit dem Parameter *g* der Methode übergeben wird soll im Attribut *gehalt* gespeichert werden.	

<u>Objekte zuweisen:</u>

Mit dem Befehl *Zuweisen von Objekten* können in Hooplu Objekte einer Variablen einer anderen Variablen zugewiesen werden:



© 2015 Henrik Horstmann

hohleGehalt():Float
Die Methode gibt das im Attribut *gehalt* gespeicherte Gehalt zurück.

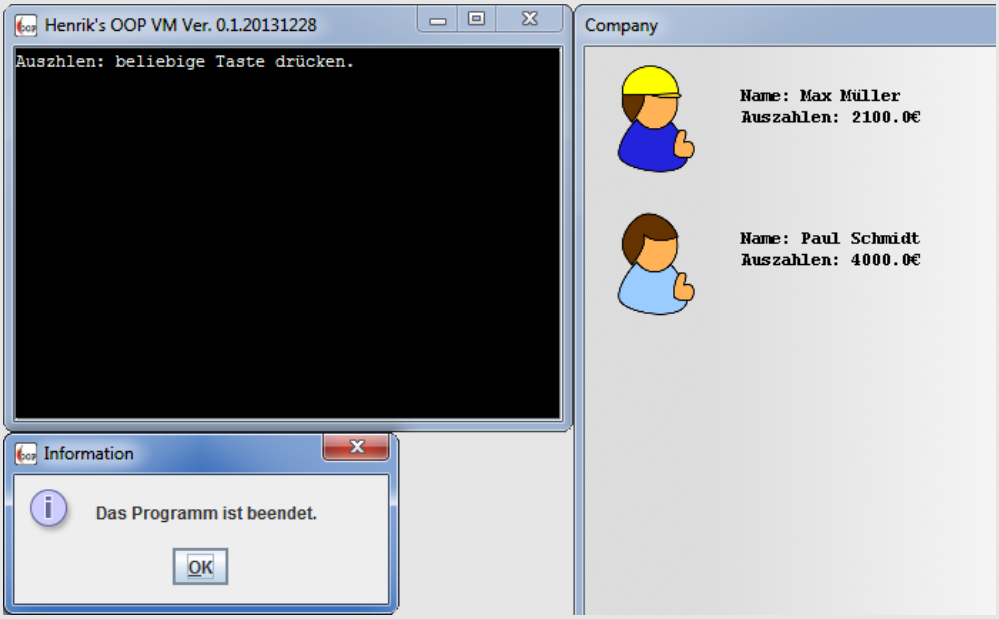
Aufgabe 6:

Erstellen Sie eine Klasse *Angestellter* gemäß der obigen Beschreibung.

Hier geht es zur Lösung auf Seite 33.

Aufgabe 7:

Erweitern Sie die *main* Methode der Klasse *Program*, so dass auch der Angestellte "Paul Schmidt" auf der *Stage* an Positiopn (50,150) erscheint. Nach dem Betätigen einer Taste, soll er ein Gehalt von 4000 € erhalten. Das Ergebnis soll wie folgt aussehen:



Hier geht es zur Lösung auf Seite 34.

Beschreibung der Klasse Manager

Manager()
Der Konstruktor hat keine Aufgaben.

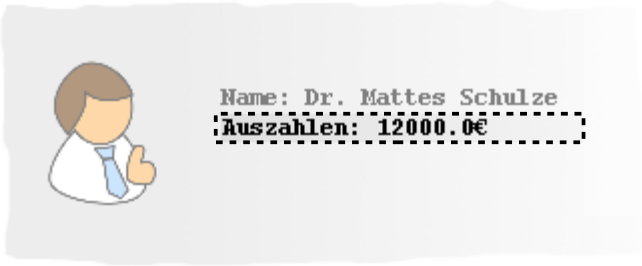
setzeBild()
Der Manager soll auf der Bühne mit folgendem Bild repräsentiert werden: *Manager.png*

auszahlen(prozent:Float,gewinn:Float)	
prozent	Prozentualer Anteil vom Gewinn des Unternehmens.
gewinn	Gewinn des Unternehmens.

Berechnet den Verdienst des Managers. Dieser besteht aus dem Gehalt eines Angestellten zuzüglich einem prozentualem Anteil vom Gewinn. Der Verdienst berechnet sich wie folgt:

$$verdienst = gehalt + \frac{gewinn \cdot prozent}{100}$$

Anschließend wird der Verdienst im Feld *fAuszahlen* angezeigt.



Hinweis:

Beachten Sie, dass die Klasse *Manager* von der Klasse *Angestellter* und **nicht** von der Klasse *Mitarbeiter* abgeleitete wird (siehe UML Diagramm auf Seite 12)

Aufgabe 8:

Erstellen Sie eine Klasse *Manager* gemäß der obigen Beschreibung.

Hinweis:

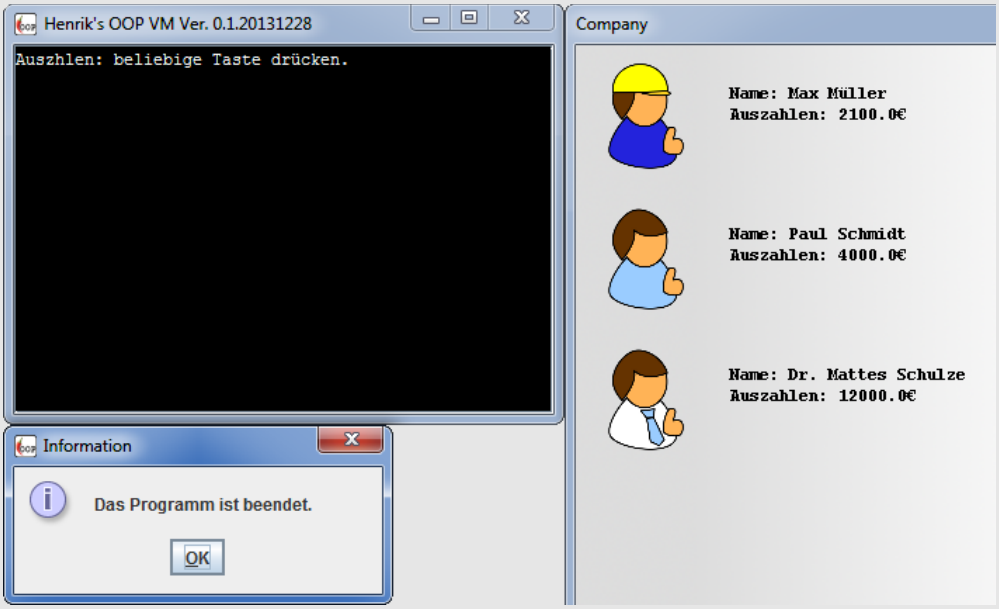
Beachten Sie, dass das Gehalt mittels der Methode *hohleGehalt()* der Superklasse *Angestellter* erfragt werden muss.

Hier geht es zur Lösung auf Seite 35.

Aufgabe 9:

Erweitern Sie die *main* Methode der Klasse *Program*, so dass auch der Manager "Dr. Mattes Schulze" auf der *Stage* an Positiopn (50,250) erscheint. Sein Grundgehalt beträgt 10.000 €. Er erhält zusätzlich 2% von dem Gewinn. Der Gewinn beträgt 100.000 €.

Das Ergebnis soll wie folgt aussehen:



Hier geht es zur Lösung auf Seite 36.

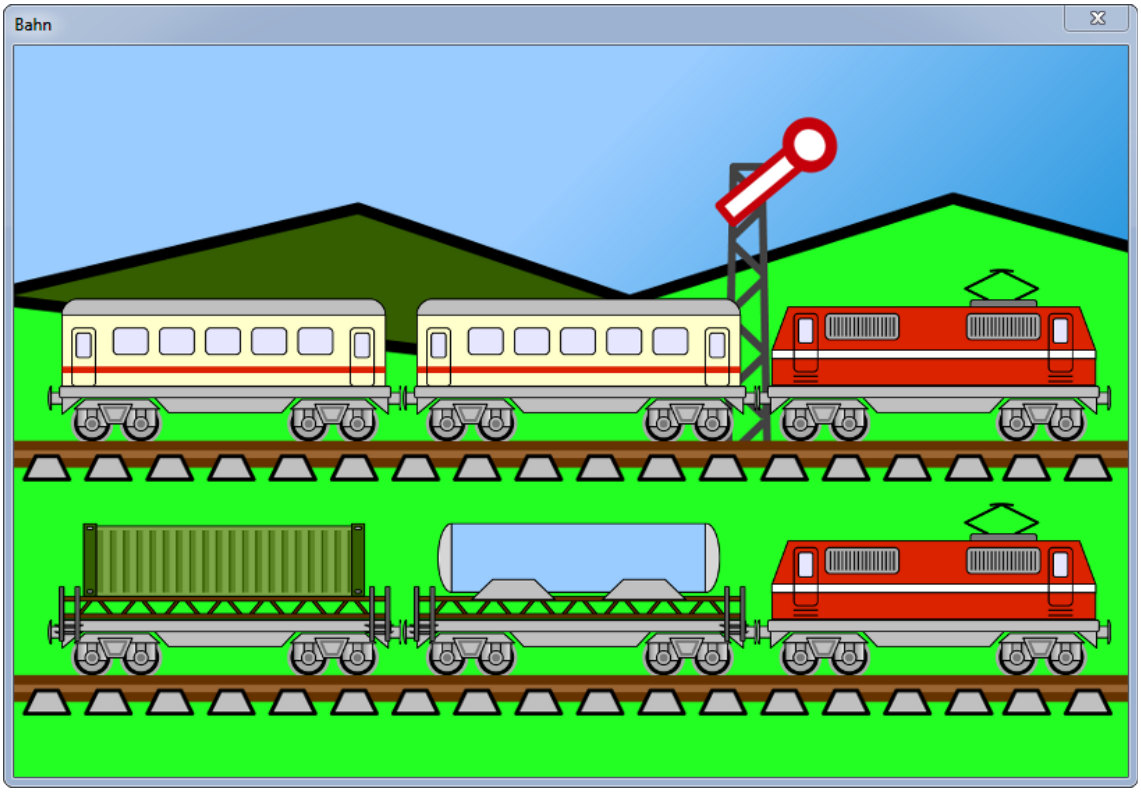
4.8 Übung

Aufgabe 10:

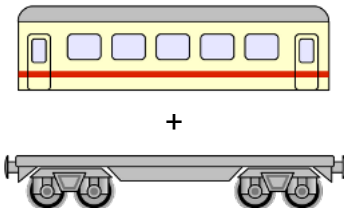
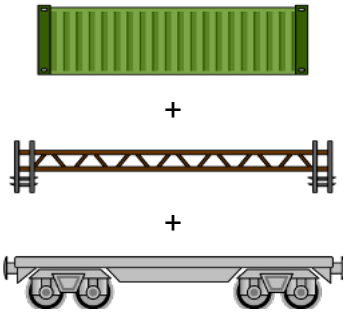
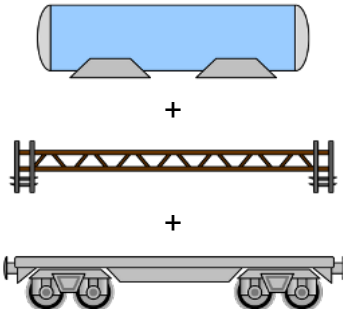
Platzieren Sie die restlichen Mitarbeiter aus der Tabelle von Seite 2 auf der *Stage*.

5 Überschreiben von Methoden

In diesem Kapitel soll folgende *Stage* realisiert werden:



Dabei geht es im speziellen um die Abbildung der Wagons als Klassen:

Personenwagon	Containerwagon	Tankwagon
		

Die Tabelle zeigt die gemeinsame Bauteile der unterschiedlichen Wagons an.

Aufgabe 11:

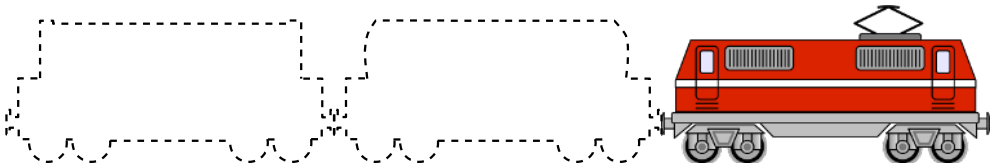
Erstellen Sie ein UML-Diagramm, das die Klassen für die unterschiedlichen Wagons enthält. Nutzen Sie dabei das Konzept der Vererbung optimal aus. Attribute und Methoden sollen die Klassen noch nicht enthalten.

Hier geht es zur Lösung auf Seite 37.

5.1 Der Zug

Der Zug inklusive Lok wird durch eine separate Klasse repräsentiert:

Zug
cx:Integer cy:Integer s:Stage
Zug(stage:Stage, xp:Integer, yp:Integer) anhaengen(wagon:Wagon)



Mit der Methode *anhaengen* werden die einzelnen Wagons an den Zug angehängt. Für den Zug hat es keine Bedeutung, ob es sich dabei um einen Personen, Container oder Tankwagon handelt.

5.2 Der Wagon

Jeder Wagon hat ein Fahrgestell mit Rädern.

Daher bildet die Klasse *Wagon* die Basis für alle spezialisierten Wagons.

Wagon
fahrgestell:StageObject
Wagon() create(stage:Stage, xp:Integer, yp:Integer)

Beschreibung der Klasse Wagon

Wagon()
Der Konstruktor hat keine Aufgaben.

create(stage:Stage, xp:Integer, yp:Integer)	
stage	Die "Bühne", auf der der Wagon platziert werden soll.
xp	Horizontale Position des Wagons auf der "Bühne".
yp	Vertikale Position des Wagons auf der "Bühne".
Die Methode erzeugt ein neues Objekt der Klasse *StageObject* und weist es dem Attribut *fahrgestell* zu. Die *StageObject* soll dass Bild des Fahrgestells (Fahrgestell.png) laden und entsprechend der Koordinaten (*xp*, *yp*) positionieren. Anschließend soll *fahrgestell* der "Bühne" *stage* hinzugefügt werden.	

Aufgabe 12:

Öffnen Sie das Projekt *Bahn.oop*.

Fügen Sie in der Klasse *Wagon* in der Methode *create* Code ein, so dass die Methode sich entsprechend der Beschreibung verhält. Dabei soll der Wagon mit einer Skallierung von 100 % dargestellt werden.

Hier geht es zur Lösung auf Seite 38.

Aufgabe 13:

Erweitern Sie die *main* Methode der Klasse *Program* um die Variablen *zug1* vom Typ *Zug* und *w1*, *w2* vom Typ *Wagon*.

Erzeugen Sie entsprechende Objekte und weisen Sie diese den Variablen zu. Der Zug soll auf die Koordinaten (660,222) positioniert werden.

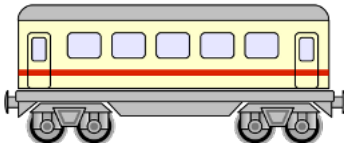
Hängen Sie die beiden Wagons *w1* und *w2* an den Zug *zug1* an.

Testen Sie Ihr Programm.

Hier geht es zur Lösung auf Seite 39.

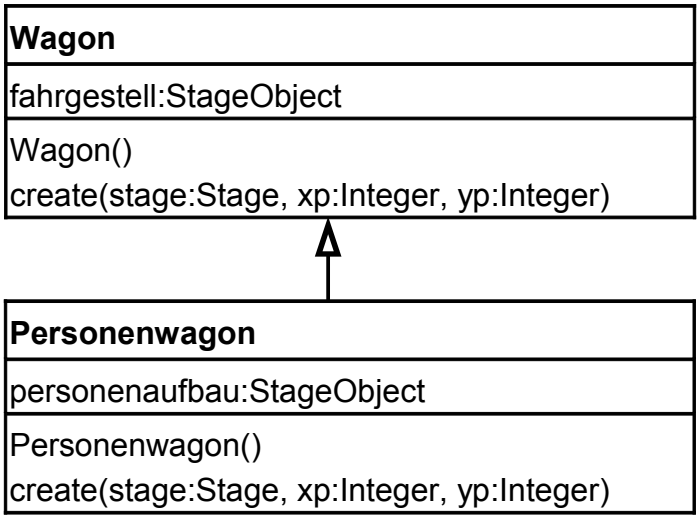
© 2015 Henrik Horstmann

5.3 Der Personenwagon



Der Personenwagon unterscheidet sich vom Wagon dadurch, dass für ihn zusätzlich ein Aufbau für Personen gezeichnet werden muss.

Wagon kann somit als Basisklasse für die Klasse Personenwagon verwendet werden:



5.4 Überschreiben von Methoden

Im UML-Diagramm hat sowohl die Basisklasse, als auch die abgeleitet Klasse eine Methode

create(stage:Stage, xp:Integer, yp:Integer)

Überschreiben von Methoden:

Sind Name und Parameterliste einer Methode in Basisklasse und abgeleiteter Klasse gleich, so **überschreibt** die Methode in der abgeleiteten Klasse die Methode in der Basisklasse.

Eine Parameterliste ist gleich, wenn alle Parameter vom gleichen Typ sind.

Beschreibung der Klasse Personenwagon

Personenwagon()
Der Konstruktor hat keine Aufgaben.

create(stage:Stage, xp:Integer, yp:Integer)	
stage	Die "Bühne", auf der der Wagon platziert werden soll.
xp	Horizontale Position des Wagons auf der "Bühne".
yp	Vertikale Position des Wagons auf der "Bühne".
Die Methode ruft als erste die *create* Methode der Basisklasse auf um das Fahrgestell auf der "Bühne" zu platzieren. Danach erzeugt sie ein neues Objekt der Klasse *StageObject* und weist es dem Attribut *personenaufbau* zu. Die *StageObject* soll dass Bild des Personenaufbaus (Personenaufbau.png) laden und entsprechend der Koordinaten (*xp*, *yp*) positionieren. Anschließend soll *personenaufbau* der "Bühne" *stage* hinzugefügt werden.	

Methoden der Basisklasse werden mit `super.methodenName` aufgerufen.

Aufgabe 14:

Erstellen Sie eine Klasse *Personenwagon* entsprechend obiger Beschreibung.

Hier geht es zur Lösung auf Seite 40.

5.5 Ohne Kenntnis einen Personenwagon verwenden

Im Programm kann nun ein Personenwagon an den Zug angehängt werden:

Anhängen eines Personenwagon:

In der Klasse *Program* in der Methode *main* muss nun lediglich dem Wagon *pw1* eine Personenwagon zugeordnet werden. Dazu ist der Mauszeiger in der Zeile, in der das Objekt für *pw1* erzeugt wird auf auf den Klassennamen zu positionieren:

```
zug1 = new Zug (stage,660,222);
wg1 = new Wagon ();
wg2 = new Wagon ();
zug1.anhaengen (wg1);
```

In der Dropdown-Liste erscheinen nun neben der Klasse *Wagon* auch alle von *Wagon* abgeleiteten Klassen:

```
zug1 = new Zug (stage,660,222);
wg1 = new Wagon ();
wg2 = new
zug1.anhaengen (wg1);
```

Wagon
Personenwagon

Da ein Personenwagon angehängt werden soll, ist die Klasse *Personenwagon* auszuwählen:

```
zug1 = new Zug (stage,660,222);
wg1 = new Personenwagon ();
wg2 = new Wagon ();
zug1.anhaengen (wg1);
```

Hinweis:

Obwohl die Variable *zug1* vom Typ *Wagon* ist, kann ihr ein Objekt vom Typ *Personenwagon* zugewiesen werden. Dies liegt daran, dass die Klasse *Personenwagon* von der Klasse *Wagon* abgeleitet ist und somit ebenfalls ein *Wagon* ist.

Aufgabe 15:

Erzeugen Sie in der Methode *main* der Klasse *Program* ein Objekt vom Typ *Personenwagon* und weisen Sie es der Variablen *w1* zu.

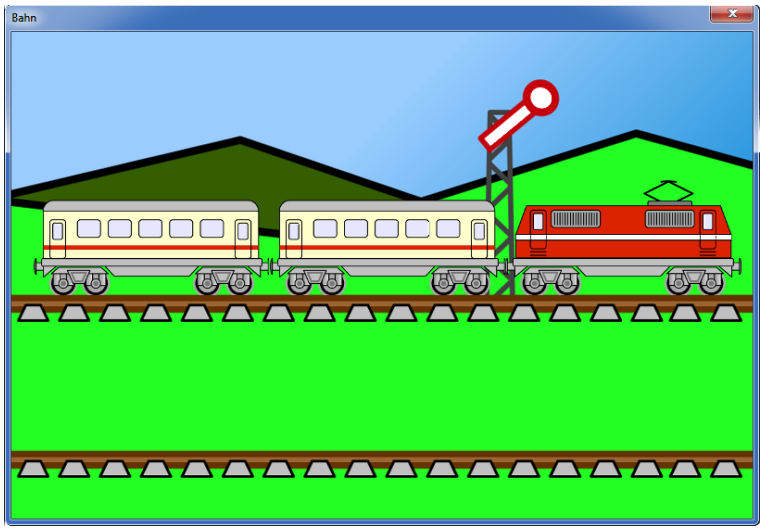
Testen Sie Ihr Programm.

Hier geht es zur Lösung auf Seite 41.

6 Übungen

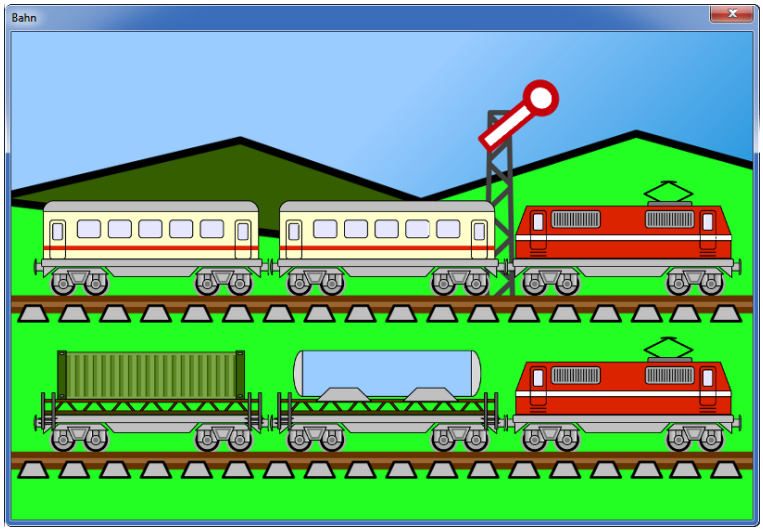
6.1 Ein vollständiger Personenzug

Ändern Sie das Projekt *Bahn.opp* so ab, dass auf dem Gleis ein vollständiger Personenzug steht:



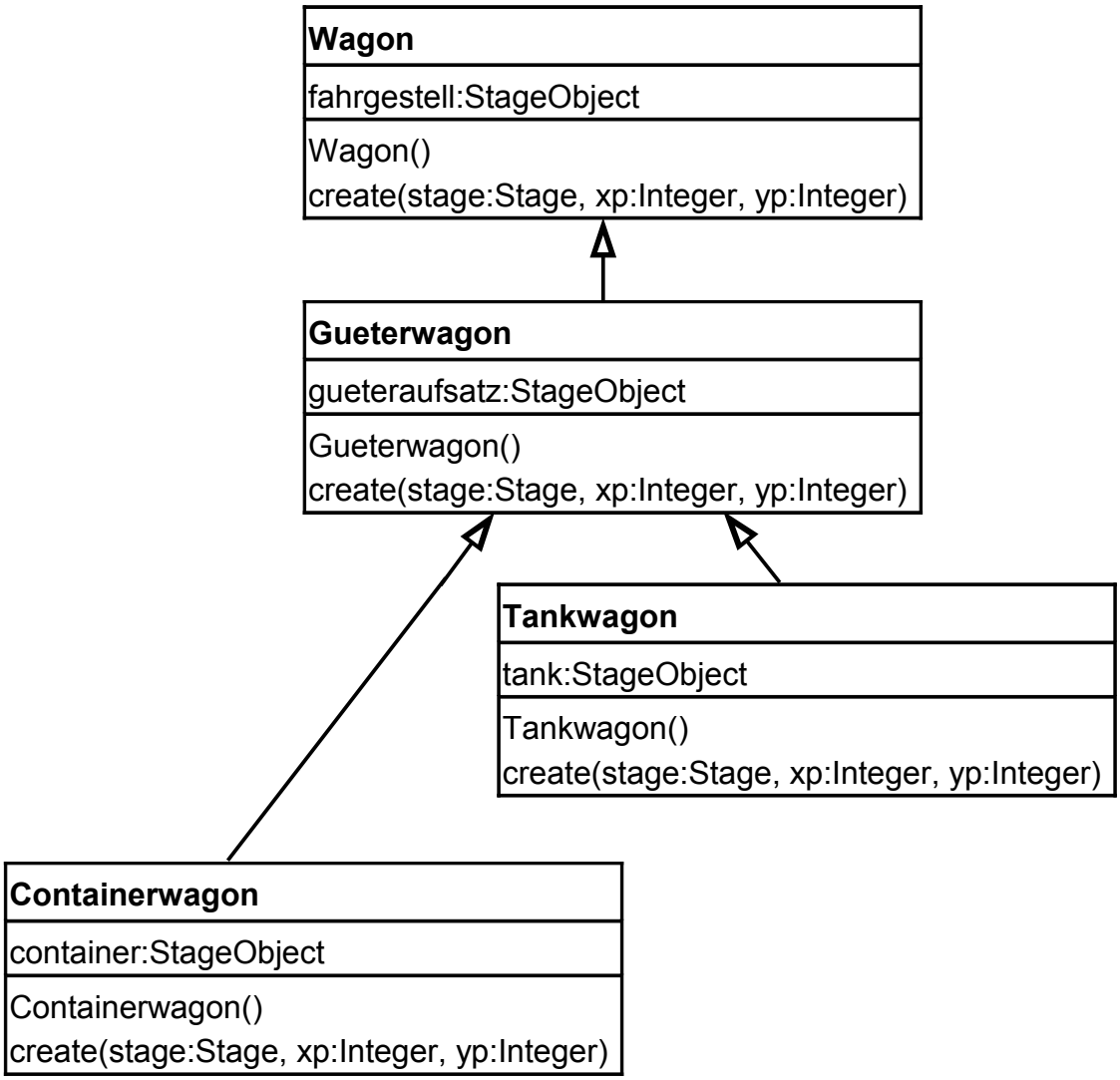
6.2 Noch mehr Züge

Erweitern Sie das Projekt *Bahn.oop* um einen Güterzug auf dem zweiten Gleis:



Der zweite Zug wird an der Position (660,390) platziert.

Dazu sollen die folgenden Klassen implementiert werden:



Beschreibung der Klasse Gueterwagon

Gueterwagon()
Der Konstruktor hat keine Aufgaben.

create(stage:Stage, xp:Integer, yp:Integer)	
stage	Die "Bühne", auf der der Wagon platziert werden soll.
xp	Horizontale Position des Wagons auf der "Bühne".
yp	Vertikale Position des Wagons auf der "Bühne".
Die Methode ruft als erste die *create* Methode der Basisklasse auf um das Fahrgestell auf der "Bühne" zu platzieren. Danach erzeugt sie ein neues Objekt der Klasse *StageObject* und weist es dem Attribut *gueteraufsatz* zu. Die *StageObject* soll dass Bild des Güterausatz (Gueteraufsatz.png) laden und entsprechend der Koordinaten (*xp*, *yp*) positionieren. Anschließend soll *gueteraufsatz* der "Bühne" *stage* hinzugefügt werden.	

Beschreibung der Klasse Tankwagon

Tankwagon()
Der Konstruktor hat keine Aufgaben.

create(stage:Stage, xp:Integer, yp:Integer)	
stage	Die "Bühne", auf der der Wagon platziert werden soll.
xp	Horizontale Position des Wagons auf der "Bühne".
yp	Vertikale Position des Wagons auf der "Bühne".
Die Methode ruft als erste die *create* Methode der Basisklasse auf um einen Güterwagon auf der "Bühne" zu platzieren. Danach erzeugt sie ein neues Objekt der Klasse *StageObject* und weist es dem Attribut *tank* zu. Die *StageObject* soll dass Bild des Tanks (Tank.png) laden und entsprechend der Koordinaten (*xp*, *yp*) positionieren. Anschließend soll *tank* der "Bühne" *stage* hinzugefügt werden.	

Beschreibung der Klasse Containerwagon

Containerwagon()
Der Konstruktor hat keine Aufgaben.

© 2015 Henrik Horstmann

create(stage:Stage, xp:Integer, yp:Integer)	
stage	Die "Bühne", auf der der Wagon platziert werden soll.
xp	Horizontale Position des Wagons auf der "Bühne".
yp	Vertikale Position des Wagons auf der "Bühne".
Die Methode ruft als erste die *create* Methode der Basisklasse auf um einen Güterwagon auf der "Bühne" zu platzieren. Danach erzeugt sie ein neues Objekt der Klasse *StageObject* und weist es dem Attribut *container* zu. Die *StageObject* soll dass Bild des Containers (Container.png) laden und entsprechend der Koordinaten (*xp*, *yp*) positionieren. Anschließend soll *container* der "Bühne" *stage* hinzugefügt werden.	

ENDE

7 Lösungen zu den Aufgaben

7.1 Lösung zu Aufgabe 1

```
public class Mitarbeiter
{
   // Anfang Attribute
   private StageObject fAuszahlen;
   private StageObject bild;
   private StageObject fName;
   // Ende Attribute


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

   }

   public void auszahlen (Float betrag)
   {
      // Anfang Variablen
      String str;
      // Ende Variablen

      str = new String("Auszahlen: ");
      str.append(betrag);
      str.append("€");
      fAuszahlen.setText(str,12);
   }

   public void setzeNamen (String name)
   {
      // Anfang Variablen
      String str;
      // Ende Variablen

      str = new String("Name: ");
      str.append(name);
      fName.setText(str,12);
   }

   public void setzeBild (String bildName)
   {
      // Anfang Variablen
      // Ende Variablen

      bild.addSprite("Bild",bildName);
   }
```

```
   public void zurStageHinzufuegen (Stage stage,Integer xp,Integer yp)
   {
      // Anfang Variablen
      Integer cx;
      String str;
      Integer cy;
      // Ende Variablen

      bild = new StageObject(xp,yp,50);
      stage.addStageObject(bild);
      cx = xp.add(60);
      cy = yp.sub(20);
      fName = new StageObject(cx,cy,100);
      fName.setTextAlignLeft();
      stage.addStageObject(fName);
      cy = yp.sub(5);
      fAuszahlen = new StageObject(cx,cy,100);
      fAuszahlen.setTextAlignLeft();
      stage.addStageObject(fAuszahlen);
   }
}
```

Zurück zur Aufgabe auf Seite 7.

7.2 Lösung zu Aufgabe 2

```
public class Facharbeiter extends Mitarbeiter
{
   // Anfang Attribute
   // Ende Attribute


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

      super();
   }

   public void setzeBild ()
   {
      // Anfang Variablen
      // Ende Variablen

      super.setzeBild("Facharbeiter.png");
   }
}
```

Zurück zur Aufgabe auf Seite 10.

7.3 Lösung zu Aufgabe 3

```
public class Facharbeiter extends Mitarbeiter
{
   // Anfang Attribute
   // Ende Attribute


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

      super();
   }

   public void setzeBild ()
   {
      // Anfang Variablen
      // Ende Variablen

      super.setzeBild("Facharbeiter.png");
   }

   public void auszahlen (Integer stunden,Float stundenlohn)
   {
      // Anfang Variablen
      Float verdienst;
      // Ende Variablen

      verdienst = stundenlohn.mul(stunden);
      super.auszahlen(verdienst);
   }
}
```

Zurück zur Aufgabe auf Seite 10.

7.4 Lösung zu Aufgabe 4

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


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

   }

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

      stage = new Stage("Company",640,480);
      stage.loadBackground("BG.GW.png");
      fa = new Facharbeiter();
      fa.zurStageHinzufuegen(stage,50,50);
      fa.setzeBild();
      fa.setzeNamen("Max Müller");
      stage.start();
   }
}
```

Zurück zur Aufgabe auf Seite 10.

7.5 Lösung zu Aufgabe 5

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Stage stage;
      Facharbeiter fa;
      Computer comp;
      // Ende Variablen

      stage = new Stage("Company",640,480);
      stage.loadBackground("BG.GW.png");
      fa = new Facharbeiter();
      fa.zurStageHinzufuegen(stage,50,50);
      fa.setzeBild();
      fa.setzeNamen("Max Müller");
      stage.start();
      comp = new Computer();
      comp.print("Auszahlen: beliebige Taste drücken.");
      stage.waitUntilKeyPressed();
      fa.auszahlen(140,15.0);
   }
}
```

Zurück zur Aufgabe auf Seite 11.

7.6 Lösung zu Aufgabe 6

```
public class Angestellter extends Mitarbeiter
{
   // Anfang Attribute
   Float gehalt;
   // Ende Attribute

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

   public void setzeBild ()
   {
      // Anfang Variablen
      // Ende Variablen

      super.setzeBild("Angestellter.png");
   }

   public void auszahlen ()
   {
      // Anfang Variablen
      // Ende Variablen

      super.auszahlen(gehalt);
   }

   public void setzeGehalt (Float g)
   {
      // Anfang Variablen
      // Ende Variablen

      gehalt = g;
   }

   public Float holeGehalt ()
   {
      // Anfang Variablen
      // Ende Variablen

      return gehalt;
   }
}
```

Zurück zur Aufgabe auf Seite 14.

7.7 Lösung zu Aufgabe 7

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Computer comp;
      Stage stage;
      Facharbeiter fa;
      Angestellter a;
      // Ende Variablen

      stage = new Stage("Company",640,480);
      stage.loadBackground("BG.GW.png");
      fa = new Facharbeiter();
      fa.zurStageHinzufuegen(stage,50,50);
      fa.setzeBild();
      fa.setzeNamen("Max Müller");
      a = new Angestellter();
      a.zurStageHinzufuegen(stage,50,150);
      a.setzeBild();
      a.setzeNamen("Paul Schmidt");
      a.setzeGehalt(4000.0);
      stage.start();
      comp = new Computer();
      comp.print("Auszhlen: beliebige Taste drücken.");
      stage.waitUntilKeyPressed();
      fa.auszahlen(140,15.0);
      a.auszahlen();
   }
}
```

Zurück zur Aufgabe auf Seite 14.

7.8 Lösung zu Aufgabe 8

```
public class Manager extends Angestellter
{
   // Anfang Attribute
   // Ende Attribute


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

      super();
   }

   public void setzeBild ()
   {
      // Anfang Variablen
      // Ende Variablen

      super.setzeBild("Manager.png");
   }

   public void auszahlen (Float prozent,Float gewinn)
   {
      // Anfang Variablen
      Float verdienst;
      Float bonus;
      // Ende Variablen

      prozent = prozent.div(100.0);
      bonus = gewinn.mul(prozent);
      verdienst = super.holeGehalt();
      verdienst = verdienst.add(bonus);
      super.auszahlen(verdienst);
   }
}
```

Zurück zur Aufgabe auf Seite 16.

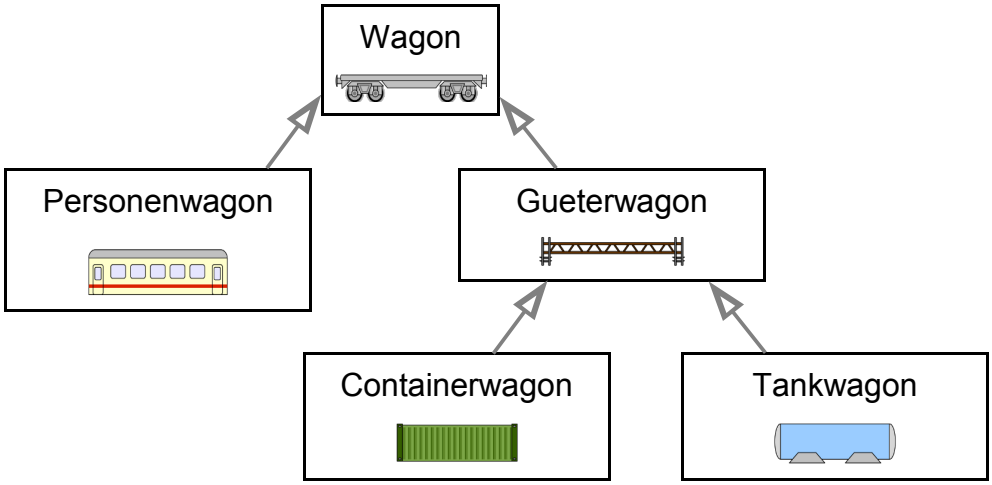
7.9 Lösung zu Aufgabe 9

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Computer comp;
      Stage stage;
      Facharbeiter fa;
      Angestellter a;
      Manager m;
      // Ende Variablen

      stage = new Stage("Company",640,480);
      stage.loadBackground("BG.GW.png");
      fa = new Facharbeiter();
      fa.zurStageHinzufuegen(stage,50,50);
      fa.setzeBild();
      fa.setzeNamen("Max Müller");
      a = new Angestellter();
      a.zurStageHinzufuegen(stage,50,150);
      a.setzeBild();
      a.setzeNamen("Paul Schmidt");
      a.setzeGehalt(4000.0);
      m = new Manager();
      m.zurStageHinzufuegen(stage,50,250);
      m.setzeBild();
      m.setzeNamen("Dr. Mattes Schulze");
      m.setzeGehalt(10000.0);
      stage.start();
      comp = new Computer();
      comp.print("Auszhlen: beliebige Taste drücken.");
      stage.waitUntilKeyPressed();
      fa.auszahlen(140,15.0);
      a.auszahlen();
      m.auszahlen(2.0,100000.0);
   }
}
```

Zurück zur Aufgabe auf Seite 16.

7.10 Lösung zu Aufgabe 11

UML Diagramm



Zurück zur Aufgabe auf Seite 17.

7.11 Lösung zu Aufgabe 12

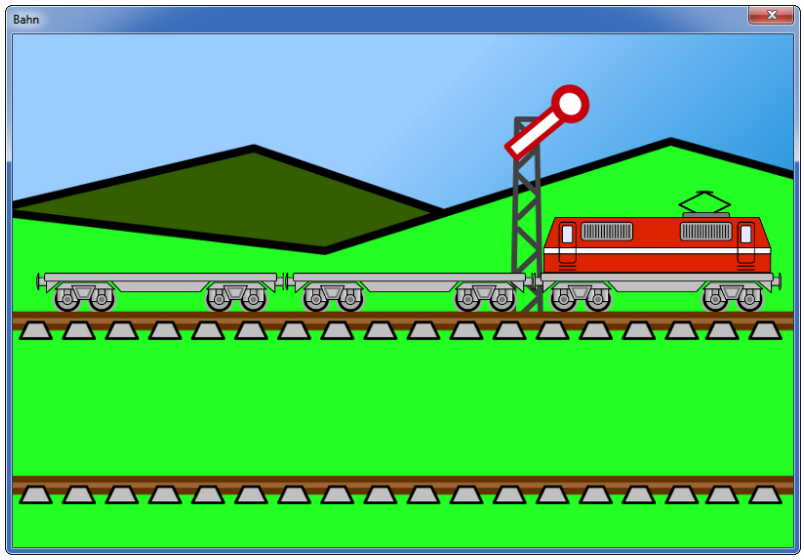
```
public class Wagon
{
   // Anfang Attribute
   private StageObject fahrgestell;
   // Ende Attribute


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

   }

   public void create (Stage stage,Integer xp,Integer yp)
   {
      // Anfang Variablen
      // Ende Variablen

      fahrgestell = new StageObject("Fahrgestell",
                                    xp,yp,"Fahrgestell.png",100);
      stage.addStageObject(fahrgestell);
   }
}
```

Zurück zur Aufgabe auf Seite 19.

7.12 Lösung zu Aufgabe 13

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Stage stage;
      Zug zug1;
      Wagon wg2;
      Wagon wg1;
      // Ende Variablen

      stage = new Stage("Bahn",800,525);
      stage.loadBackground("Bahn.png");
      stage.start();
      zug1 = new Zug(stage,660,222);
      wg1 = new Wagon();
      wg2 = new Wagon();
      zug1.anhaengen(wg1);
      zug1.anhaengen(wg2);
   }
}
```

Das Ergebnis sollte wie folgt aussehen:



Zurück zur Aufgabe auf Seite 19.

7.13 Lösung zu Aufgabe 14

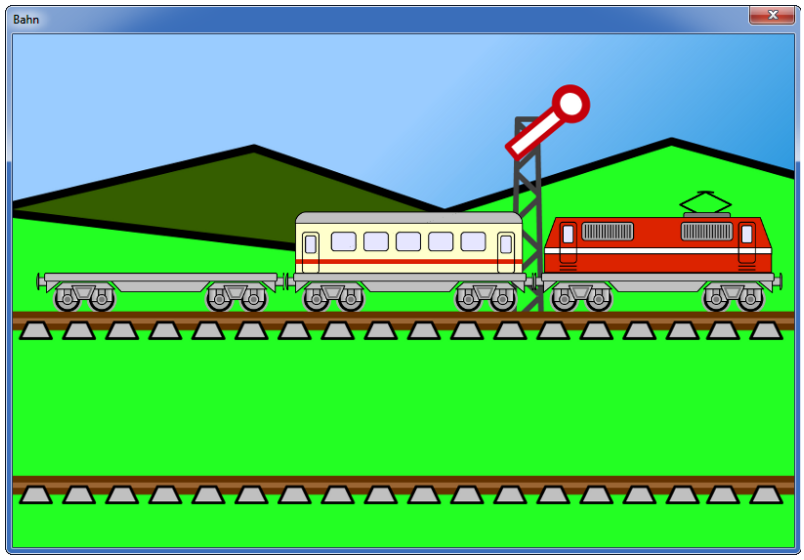
```
public class Personenwagon extends Wagon
{
   // Anfang Attribute
   private StageObject personenaufbau;
   // Ende Attribute


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

      super();
   }

   public void create (Stage stage,Integer xp,Integer yp)
   {
      // Anfang Variablen
      // Ende Variablen

      super.create(stage,xp,yp);
      personenaufbau = new StageObject("Personenaufbau",
                                       xp,yp,
                                       "Personenaufbau.png",100);
      stage.addStageObject(personenaufbau);
   }
}
```

Zurück zur Aufgabe auf Seite 21.

7.14 Lösung zu Aufgabe 15

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


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

   }

   public void main ()
   {
      // Anfang Variablen
      Stage stage;
      Zug zug1;
      Wagon wg2;
      Wagon wg1;
      // Ende Variablen

      stage = new Stage("Bahn",800,525);
      stage.loadBackground("Bahn.png");
      stage.start();
      zug1 = new Zug(stage,660,222);
      wg1 = new Personenwagon();
      wg2 = new Wagon();
      zug1.anhaengen(wg1);
      zug1.anhaengen(wg2);
   }
}
```

Das Ergebnis sollte wie folgt aussehen:



Zurück zur Aufgabe auf Seite 22.

7.15 Lösung zu Aufgabe 16

Zurück zur Aufgabe auf Seite 19.