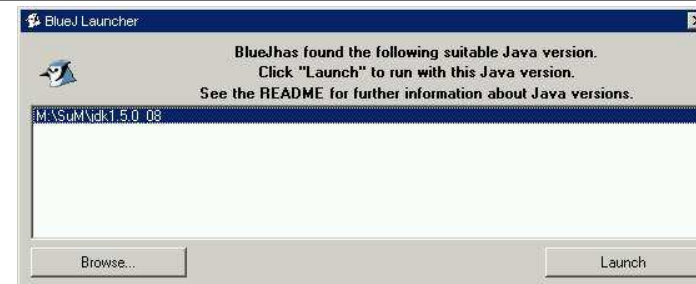


Start Projekt

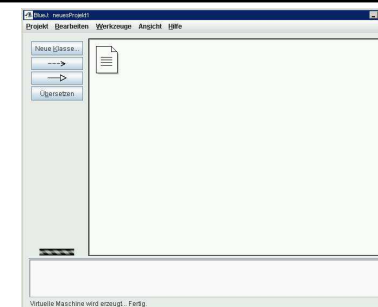
Starte BlueJ
(Select Virtual Machine)



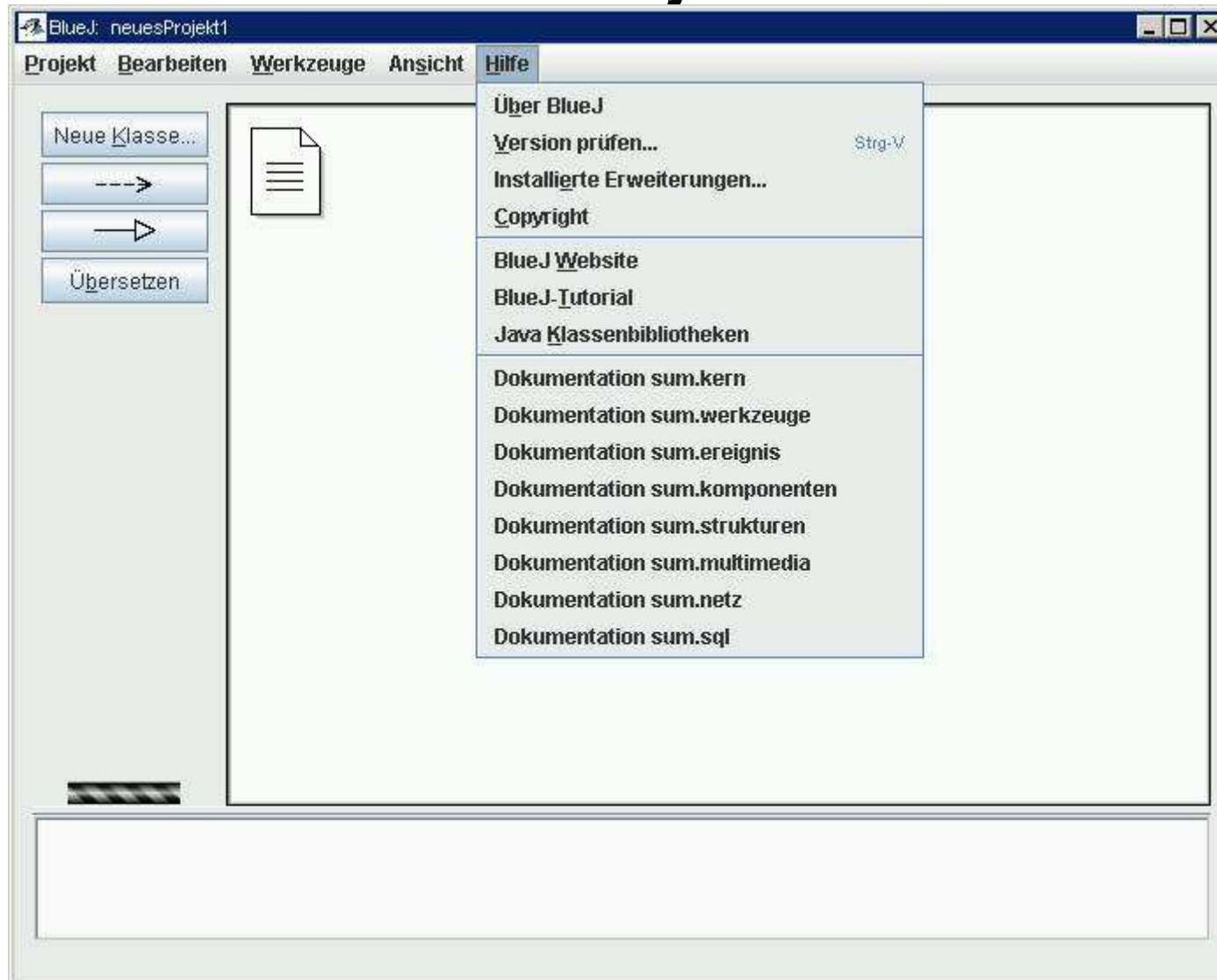
Launch the suitable
Java version
(wichtiges Detail!)



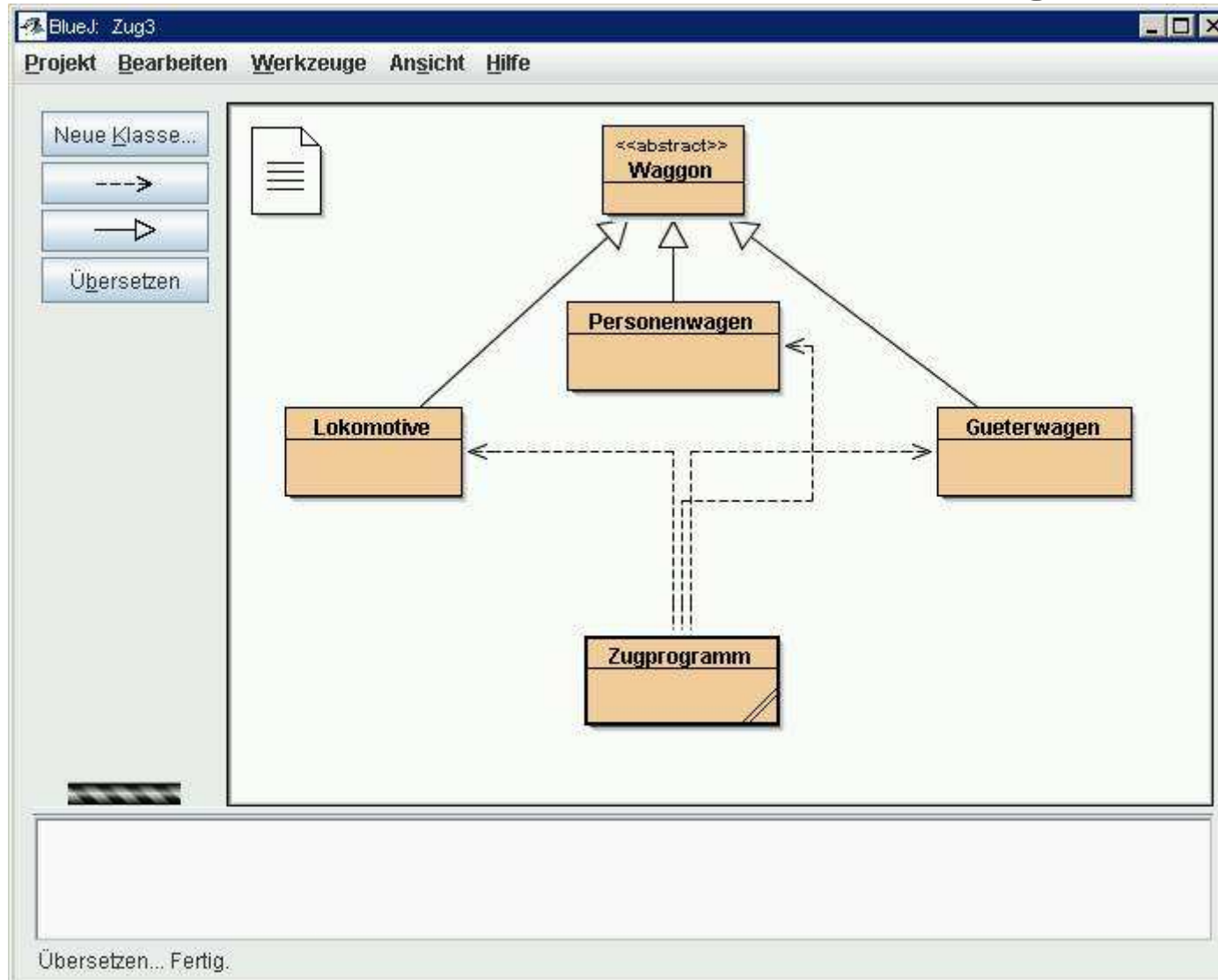
Anzeige BlueJ-Projekt-
Verwaltung



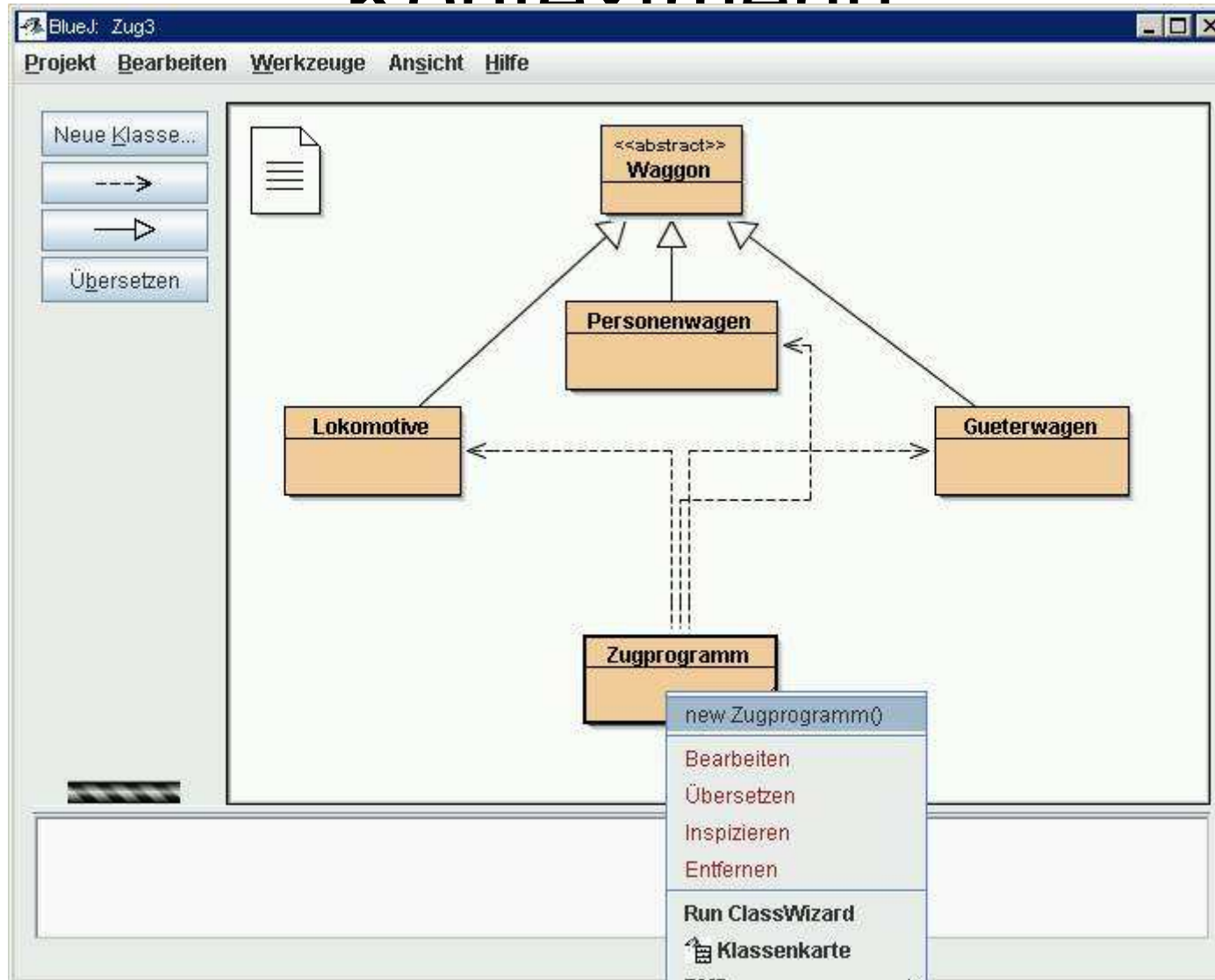
Ansicht BlueJ-Projekt-Verwaltung



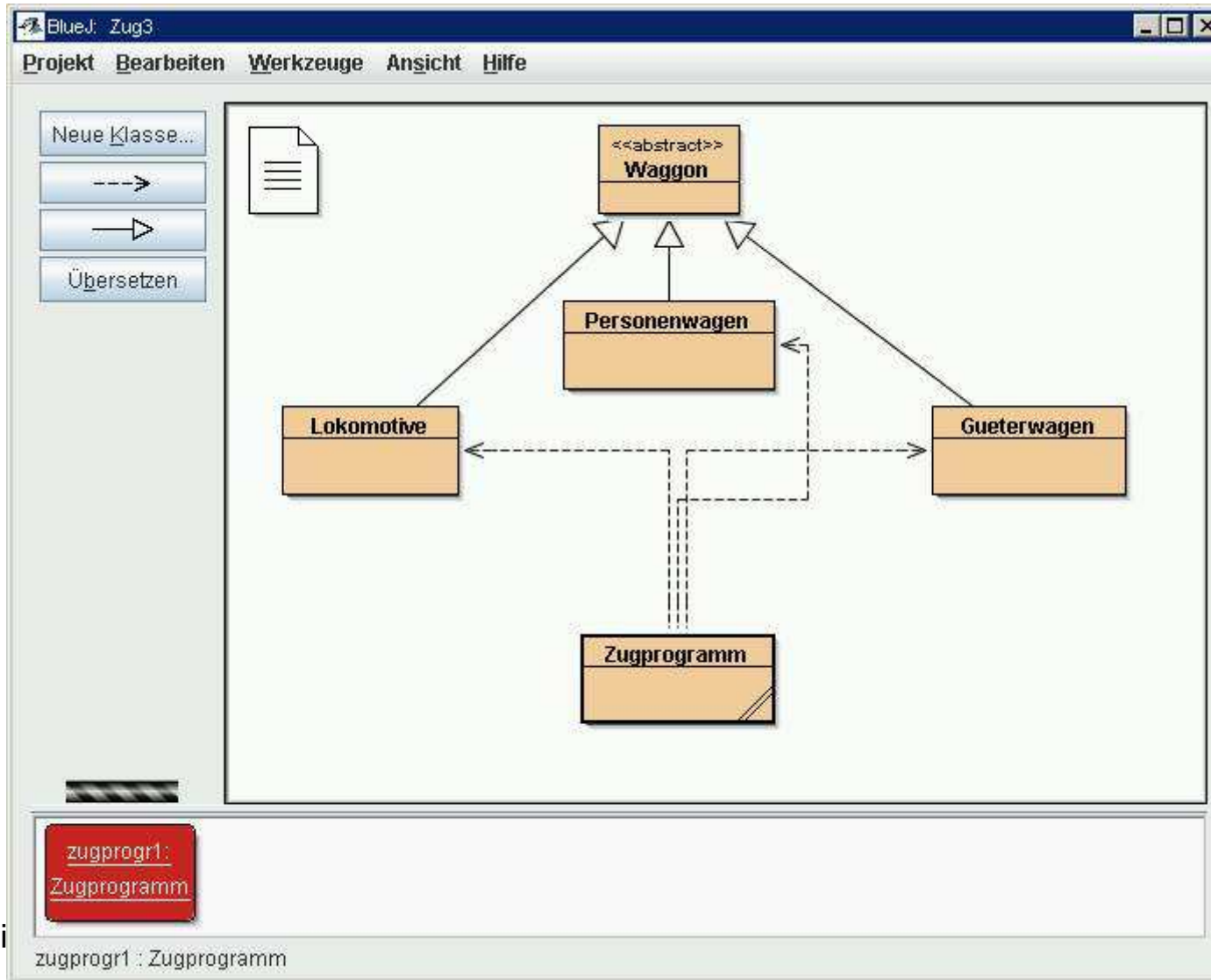
Generiere Klassen / Diagramm



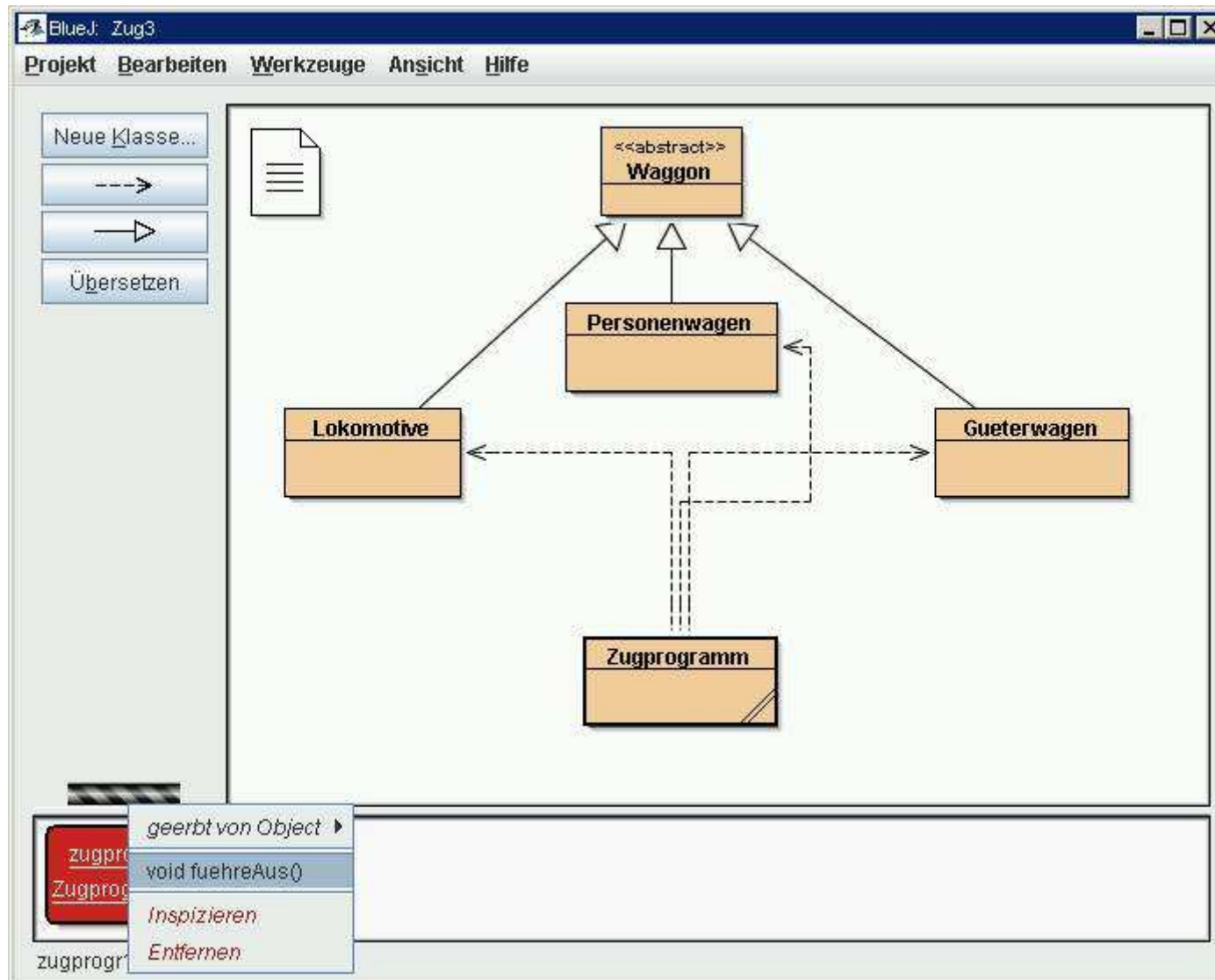
Instanziierung Startklasse per Kontextmenü



Instanz erzeugt



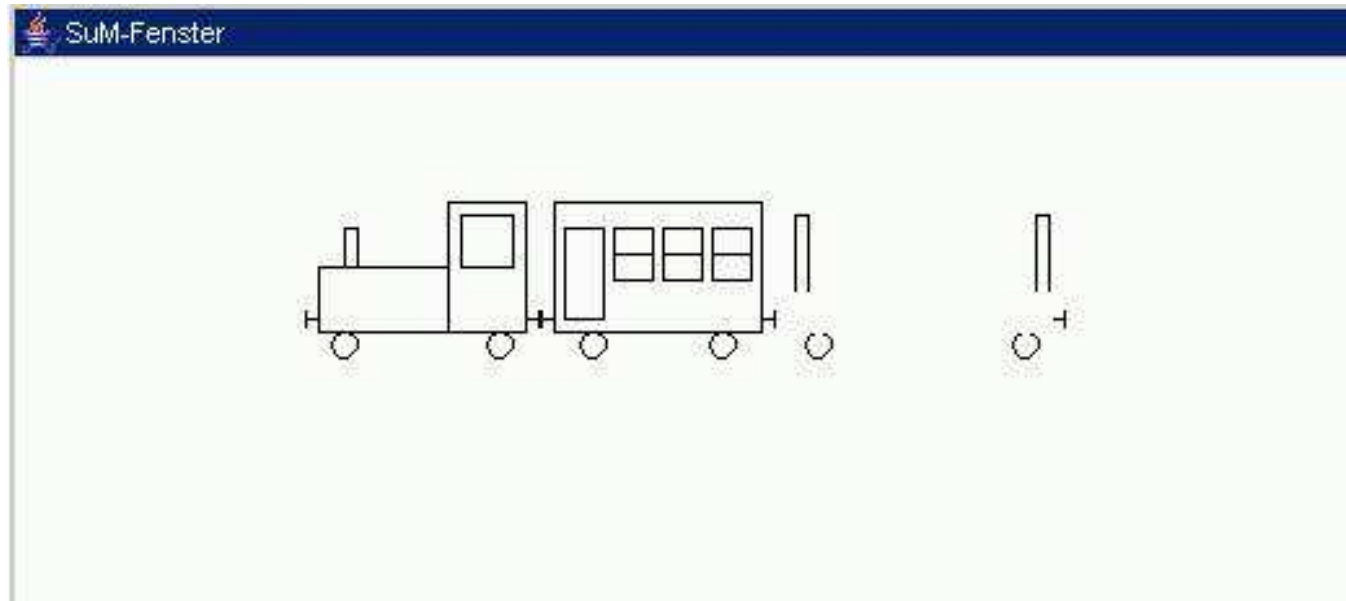
Kontextmenü: Startmethode ausführen



(anspruchsvolles)

Beispiel in Ausführung

(abstrakte Klasse)



Buchtext zum Beispiel (Kap. 9 abstrakte Klassen II)

Kapitel 9 Abstrakte Klassen II

9

In diesem Kapitel sollen Sie lernen:

- wie man mit Klassen- und Beziehungsdiagrammen ein Programm entwirft
- was lokale Variable sind
- wie man eine Verkettung von Objekten konstruiert
- wie Objekte einer Verkettung Nachrichten weiterleiten

In diesem Kapitel sollen Sie den objektorientierten Entwurf unter Benutzung einer abstrakten Klasse üben. Auf dem Bildschirm soll ein Zug gezeichnet werden, der langsam nach links fährt. Der Zug soll aus Lokomotive, Personen- und Güterwagen bestehen.



Abbildung 9.1:
ein Zug

9.1 Entwurf

Man erkennt sofort, dass alle drei Objekte gemeinsame Dienste zur Verfügung stellen müssen. Alle drei Objekte benötigen einen Stift, um sich zu zeichnen.

Es empfiehlt sich eine abstrakte Oberklasse `Waggon` mit den Unterklassen `Lokomotive`, `Personenwagen` und `Guetterwagen` zu entwickeln.



Abbildung 9.2:
abstrakte Oberklasse Waggon

Übung 9.1 Welche Dienste werden benötigt, welche davon kann man in die abstrakte Oberklasse verlagern?

Übung 9.2 Welche Attribute werden benötigt, welche davon kann man in die abstrakte Oberklasse verlagern?

Der einzigen Dienste der Unterklassen neben dem Konstruktor sind `zeichne` und `laenge`. Damit die Waggons aneinander passen, wenn sie von verschiedenen Arbeitsgruppen erstellt werden, müssen ein paar Vereinbarungen zum Zeichnen getroffen werden:

- Die Räder haben einen Radius von 5 Punkten.
- Der Boden des Waggons sitzt auf den Rädern.

- Der Puffer ist 4 Punkte vom Boden entfernt, hat eine Breite von 5 und eine Höhe von (zweimal) 3 Punkten.
- Die Position eines Waggons ist die Mitte des linken Puffers.

Die meisten Dienste stellt die abstrakte Oberklasse `Waggon` zur Verfügung. Die Dienste `hPosition` und `vPosition` sind notwendig, um die Position der nächsten Waggons zu ermitteln. Der Dienst `stift` ist notwendig, um die private Bezugsklasse `stift` an die Unterklassen weiterzureichen.

Als Attribute kommen die Position und die Länge des Waggons in Frage. Wie immer verwaltet der Stift die Position. Im Konstruktor der Unterklassen wird die Länge an die abstrakte Oberklasse übermittleit. Da der Dienst `setzeLaenge` der Oberklasse nicht nach außen sichtbar sein soll, er wird nur von den Unterklassen benutzt, wird seine Sichtbarkeit auf `protected` beschränkt. Dem Konstruktor eines Waggons wird die Position als Parameter übergeben.

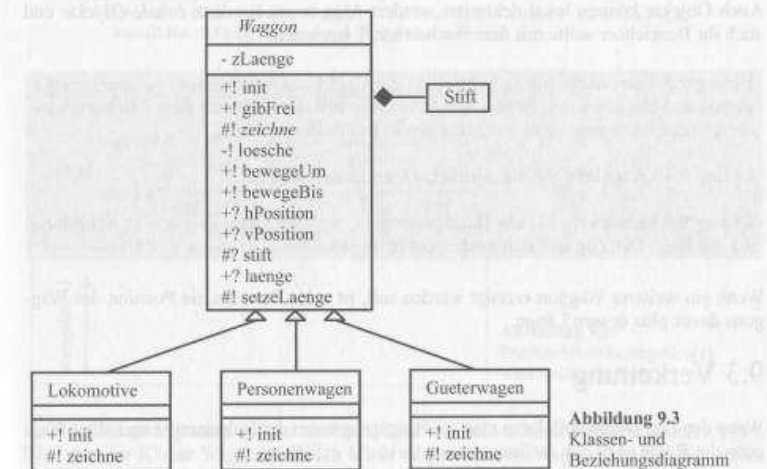
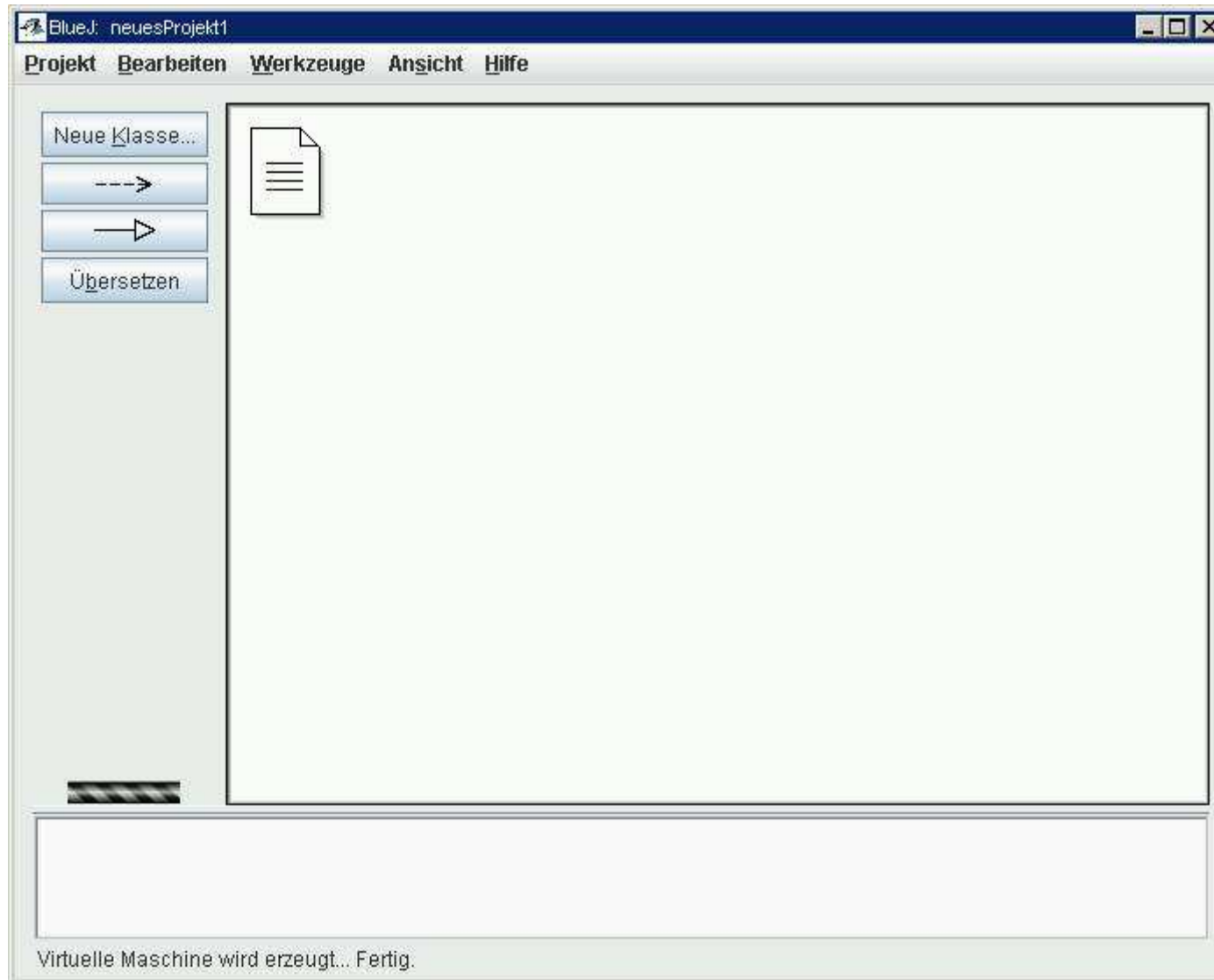


Abbildung 9.3
Klassen- und
Beziehungsdiagramm

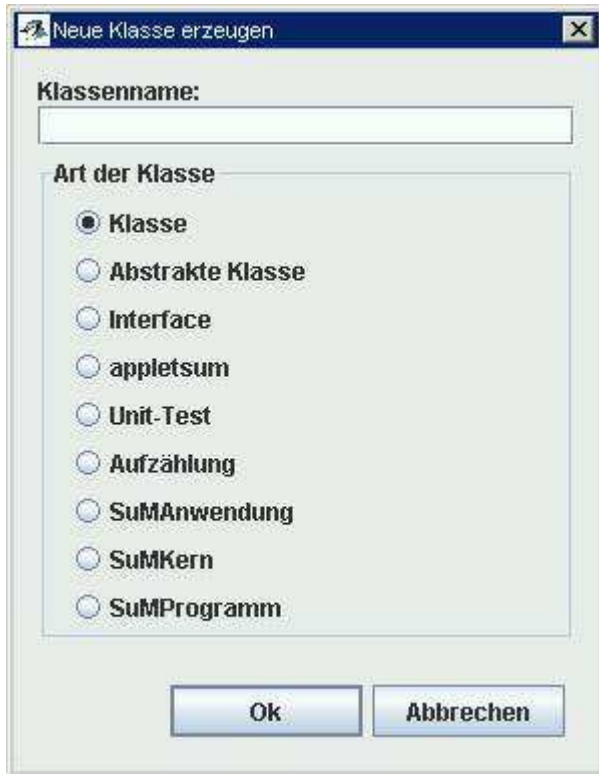
9.2 Lokale Variable

Wenn ein konkreter `Waggon` gezeichnet werden soll, so tritt ein Problem auf: Der `Stift` soll nach dem Zeichnen wieder an die Ausgangsposition zurückkehren. Dazu muss die Ausgangsposition des Stifts verwahrt werden. Es gibt in Java die Möglichkeit innerhalb eines Dienstes Variable zu deklarieren. Die Variablen existieren nur während der Laufzeit dieses Dienstes. Sie können auch nur innerhalb dieses Dienstes angesprochen werden. Man bezeichnet sie als lokale Variable und lässt ihren Bezeichner mit dem Buchstaben `'l'` beginnen. Es ist ein guter Stil, wenn lokale Variable zu Beginn eines Dienstes deklariert werden, so wie Attribute zu Beginn der Klassendefinition deklariert werden. Attribute bezeichnet man oft auch als globale Variable, da sie von allen Diensten einer Klasse angesprochen werden können. Der Vorteil von lokalen Variablen liegt darin, dass bei arbeitsteiliger Entwicklung von Diensten einer Klasse in unterschiedlichen Diensten

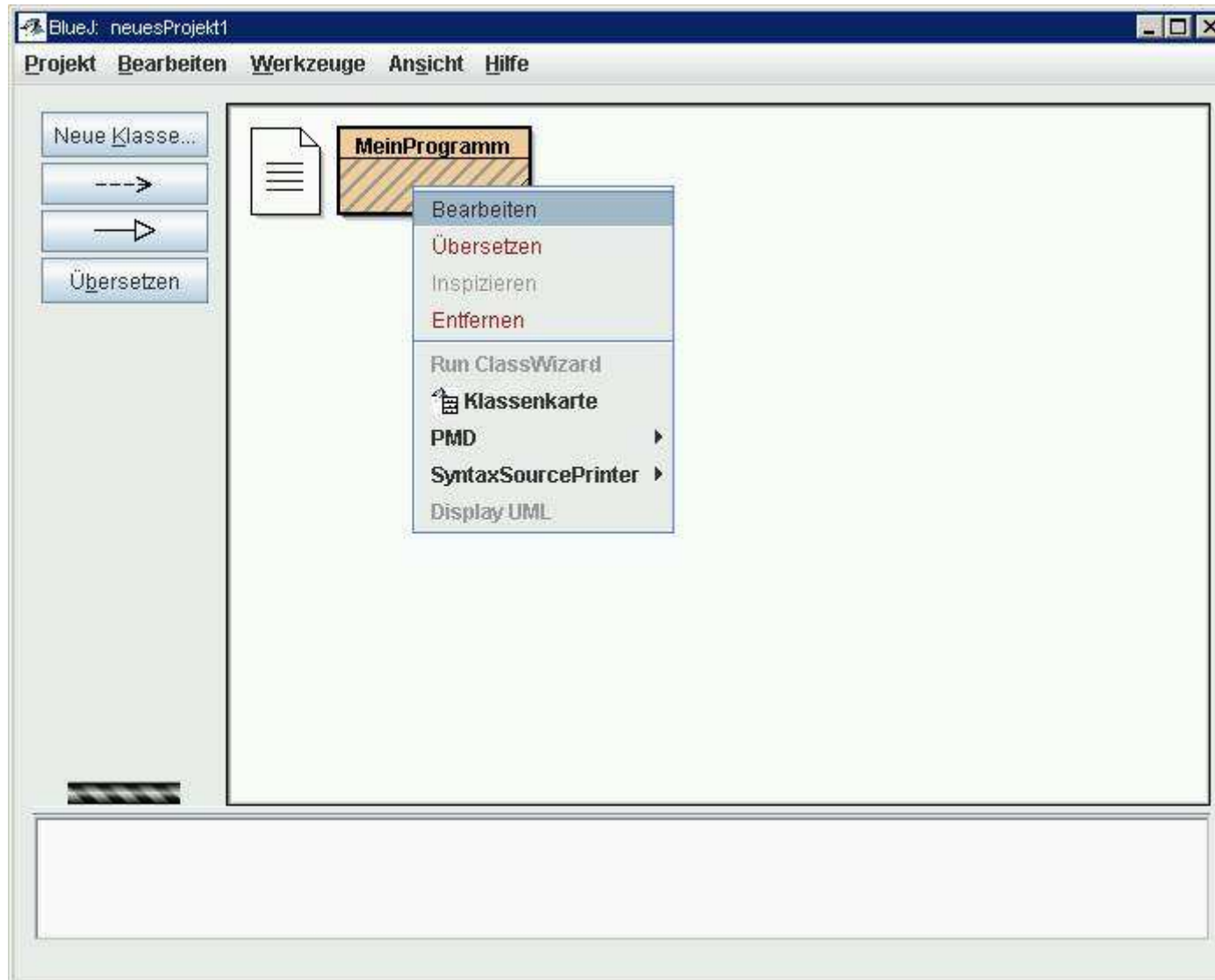
Eigene Entwicklungsarbeit: neue Klasse(n)



Vorgabe wählen



Vorgaben bearbeiten



Klasse im Editor

```
1  
2 import sum.kern.*;  
3 /**  
4  * @author  
5  * @version  
6  */  
7 public class MeinProgramm  
8 {  
9     // Objekte  
10    Bildschirm derBildschirm;  
11    Stift meinStift;  
12  
13    // Konstruktor  
14    public MeinProgramm()  
15    {  
16        derBildschirm = new Bildschirm();  
17        meinStift = new Stift();  
18    }  
19  
20    // Dienste  
21    public void fuehreAus()  
22    {  
23        // Aktionsteil  
24        meinStift.bewegeBis(100, 100);  
25        meinStift.schreibeText("Hallo Welt");  
26  
27        // Aufräumen  
28        meinStift.gibFrei();  
29        derBildschirm.gibFrei();  
30    }  
31 }
```

gespeichert