




Stufe 5 - Beliebige Züge, ereignisgesteuert

Ziele	<p>Die Schülerinnen und Schüler erstellen ein Programm, mit dem man beliebige Züge zusammenstellen kann und das durch Tastatur- und Mausereignisse gesteuert wird und damit die Ereignisanwendung vorbereitet. Außerdem üben sie die Anwendung boolescher Schalter und vertiefen das Weiterreichen von Nachrichten in einer Nachrichtenkette.</p>
Aufgabenstellung	<p>Die Schülergruppe diskutiert ein komfortables Programm, um beliebige Züge zu animieren. Eine mögliche Funktionsbeschreibung ist:</p> <p>Der Zug besteht zu Beginn nur aus einer Lokomotive. Durch Drücken der Taste mit dem entsprechenden Anfangsbuchstaben können beliebige Waggon angehängt werden. Durch einen Mausklick wird die Animation gestartet bzw. angehalten. Das Programm wird durch Drücken der Taste "e" beendet.</p>
Das Ankoppeln eines beliebigen Waggon	<p>Der Knackpunkt dieser Stufe ist die Fähigkeit des Programms, beliebige Waggon an den Zug ankoppeln zu können. Zunächst einmal muss den Schülerinnen und Schülern klar werden, dass man jeweils nur ein neues Exemplar der entsprechenden Klasse erzeugen muss und es hinten anhängen kann. Es kann also für das Anhängen von 5 neuen Waggon fünfmal ein und dieselbe Variable verwendet werden (in den Programmiersprachen Java und Delphi benötigt man sogar keine Variable, da man anonyme Objekte benutzen kann) und es müssen nicht etwa fünf Variablen deklariert werden.</p> <p>Hauptproblem ist aber das Anhängen des jeweiligen neuen Waggon an das Zugende. Eine Lösung ist es, sich den letzten Waggon in einer Variablen zu merken. Die Schülerinnen und Schüler können aber auch an ihre spielerischen Erfahrungen mit der Nachrichtenkette denken. Dann reicht jeder Waggon die Nachricht <code>koppelAn</code> an seinen Nachfolger weiter, bis schließlich das Zugende erreicht ist.</p>
Klasse Zug	<p>Entsprechend wird auch die <code>gibFrei</code>-Nachricht in der Nachrichtenkette weitergereicht. Nur muss hierbei zuerst der Nachfolger diese Nachricht erhalten, bevor der Waggon selbst zerstört wird.</p> <p>Gerade die Lösung des Ankoppel-Problems mit dem Merken des letzten Waggon legt die Verwendung einer Klasse <code>Zug</code> nahe. Diese könnte sich dann die Lokomotive und den letzten Waggon in Form einer Hat- und einer Kennt-Beziehung merken.</p> <p>Tatsächlich lässt sich die Zug-Animation mit Hilfe einer solchen Klasse gut realisieren. Nur reichen fast alle Dienste dieser Klasse lediglich die empfangene Nachricht an die Lokomotive weiter, die sie dann an den Rest des Zuges weitergibt.</p> <p>Unterrichtserfahrungen haben außerdem gezeigt, dass die zusätzliche Klasse, die mit einer weiteren Abstraktion verbunden ist (es handelt sich praktisch um die Klasse <code>Liste</code>), oft nur von besonders leistungsstarken Schülerinnen und Schülern wirklich durchschaut wird, während andere Probleme bekommen. So erscheint es besser, die Idee der Liste in diesem Projekt nur vorzubereiten und erst in einem späteren Projekt zu vertiefen.</p>
Implementation der abstrakten Klasse Waggon	<p>Deshalb wird in diesem Durchgang auf die Klasse <code>Zug</code> verzichtet. Dennoch ist ein alternativer Unterrichtsweg mit dieser Klasse natürlich denkbar.</p> <p>Es müssen nun also die Methoden <code>koppelAn</code> und <code>gibFrei</code> mit einer Nachrichtenkette implementiert werden. Die Dokumentation der Klasse <code>Waggon</code> ändert sich geringfügig.</p>
 Metasprache  Quelltext Maus-Ereignisse	<p>Die Klasse <code>Waggon</code> kann dann in der Metasprache etwa so aussehen.</p> <p>Nun müssen noch Überlegungen zum Hauptprogramm durchgeführt werden.</p>
Erzeugen der neuen Waggon	<p>Die Fahrt des Zuges soll mit Mausklicken beginnen und enden. Dazu reicht es also nicht, wenn geprüft wird, ob die Maus gedrückt ist, sondern es muss gewartet werden, bis der Mausknopf losgelassen wird; dann wird entsprechend reagiert.</p> <p>Damit wird das MausLos-Ereignis vorbereitet.</p> <p>Eine Besonderheit ist das Erzeugen der neuen Waggon. Es kann nicht für jeden Waggon eine eigene Variable angelegt werden, weil man dann keinen beliebigen Zug zusammenstellen könnte. Da sich der jeweils alte letzte Waggon den neuen letzten nach dem Anhängen merkt, ist es auch nicht nötig, dass irgendein Waggon außer der Lokomotive in einer Variablen des Hauptprogramms dauerhaft gespeichert wird.</p>
 Quelltext	<p>Tatsächlich genügt zum Anhängen des Waggon in den meisten objektorientierten Programmiersprachen eine einzige Variable. In der Metasprache sieht das z. B. so aus:</p> <pre>neuerWaggon := neu(Personenwagen) neuerWaggon.init(0,0) meineLokomotive.koppelAn(neuerWaggon)</pre> <p>Dabei ist die Variable <code>neuerWaggon</code> vom Typ <code>Waggon</code>, also von der abstrakten Oberklasse. Durch die Polymorphie ist es möglich, dieser Variablen ein Exemplar einer beliebigen Unterklasse, also etwa auch einen Güterwagen zuzuweisen. Sobald ein <code>neuerWaggon</code> angekoppelt ist, ist der Zugriff durch die Verkettung des</p>

Zuges möglich; es ist also kein Problem, dass dieselbe Variable `neuerWaggon` danach einen anderen neuen speziellen Waggon zugewiesen bekommt.

In manchen Programmiersprachen, z. B. Object Pascal ist die Erzeugung eines neuen Exemplars einer Unterklasse in einer Variablen der Oberklasse nicht möglich. Hier muss man für jeden Waggon typ eine eigene Variable verwenden.

In modernen objektorientierten Programmiersprachen wie Java und Delphi kann man durch die Verwendung von *anonymen Objekten* sogar ganz auf eine Hilfsvariable verzichten. Das sieht dann etwa so aus:

```
meineLokomotive.koppelAn(neu (Personenwagen(0,0))
```

Eine Folge des Steuerns der Zug-Animation mit dem Mausklick ist es, dass das Programm sich in einer booleschen Variable den Zustand "fährt" merken muss und in der Schleife entsprechend reagiert.

Eine weitere Anwendung dieses Verfahrens lernen die Schülerinnen und Schüler mit einer Variable kennen, in der sich das Programm merkt, ob es beendet wird. Sie wird bei Druck der Taste "e" auf `wahr` gesetzt.

Es entsteht nun ein Hauptprogramm mit einer typischen Ereignis-Schleife. Es kann dann in der Metasprache etwa [so](#) aussehen.

Boolesche Schalter

Implementation des Hauptprogramms

 [Metasprache](#)

 [Quelltext](#)

Dauerhafte Animation

Nun folgen zwei Ergänzungen, die man als Übung oder zur inneren Differenzierung verwenden kann.

Leider ist die Zug-Animation recht kurz. Irgendwann ist er durchgefahren und verschwunden. Da wäre es sinnvoll, wenn der Zug, nachdem er aus dem linken Bildrand verschwindet, am rechten wieder hereinfährt. Das geht natürlich nur dann, wenn der Zug kürzer als die Bildschirmbreite ist.

Eine Möglichkeit der Realisierung ist es, dass ein Waggon am Ende der `bewegeUm`-Methode überprüft, ob seine Position negativ ist. In diesem Fall muss er sich um eine Bildschirmbreite nach rechts bewegen. Interessant ist, dass diese Bewegung durch einen rekursiven Aufruf von `bewegeUm` leicht geschehen kann; es ist dabei aber nicht nötig, die Rekursion zu thematisieren. Diese Lösung würde im Endeffekt trotz der Realisierung in der Oberklasse nur die Lokomotive betreffen, die dann die anderen Waggonen automatisch nach sich zieht.

Dazu müssen die Waggonen aber den Bildschirm bzw. zumindest dessen Breite kennen. Es muss also bei der Initialisierung ein weiterer Parameter übergeben werden.

Eine weitere sinnvolle Ergänzung ist die Möglichkeit, den Zug während der Animation zu beschleunigen oder bremsen. U. U. haben die Schülerinnen das schon bei der ersten Entwicklung des Hauptprogramms vorgeschlagen und realisiert.

Dazu muss die Klasse `Waggon` um das Attribut der Geschwindigkeit erweitert werden. Dieses erhält bei der Initialisierung einen Standardwert. Mit einem Dienst `beschleunige` kann man die Geschwindigkeit durch Multiplikation mit einem Faktor ändern; d. h., dass mit einem Faktor < 1 auch das Bremsen ermöglicht wird. Auch diese Nachricht reicht sich in einer Kette weiter.

Schließlich muss die Klasse `Waggon` noch um den Dienst `fahre` ergänzt werden, der den Waggon um seine Geschwindigkeit bewegt.

Im Hauptprogramm kann man das Beschleunigen und Bremsen etwa den Tasten "+" und "-" zuordnen. Zum Schluss ein Überblick über die "Komplettversion" des Zugs:

Letzte Fassungen der [Waggon-Dokumentation](#), [Implementation der Waggon-Klasse](#) und des [Hauptprogramms](#) in der Metasprache.

Dokumentation und Implementation

Quelltexte:

 [Waggon](#)

 [Hauptprogramm](#)