MVC-Konzept

MVC steht für Model-View-Controller. Es gibt eine Möglichkeit an, wie man ein größeres Programm aufbauen kann. Dazu wird das Programm in die drei Bereiche aufgeteilt:

Model Zum Model gehören die Klassen, die die Daten enthalten. Diese bilden das Modell, auf dem das Programm basiert.

View Zur View gehören alle Klassen, die sich um die Ausgabe kümmern. Dieses kann z. B. in der Textkonsole, in einer graphischen Oberfläche oder auf dem Drucker geschehen.

Controller Mit den Controllerklassen werden alle benötigten Methoden bereit gestellt, die für die Kommunikation zwischen View und Model benötigt werden. Sie sorgen dabei auch, z. B. für das Einlesen der Daten und Erzeugen der Model-Objekte.

Vorteile

Der Programmcode lässt sich besser überblicken. Außerdem können verschiedene Personen besser unterschiedliche Teile übernehmen, so dass sich ein Programm gut kooperativ erstellen lässt.

Zusätzlich lassen sich einzelne Teile besser austauschen. Hier ist als Beispiel zu nennen, dass man ein Programm für die Textkonsole und eine graphische Oberfläche schreiben kann. Dazu muss in einem solchen Fall nur die View ausgetauscht werden.

Beispiel für ein MVC-Programm

Als Beispiel soll hier ein Programm zur Verwaltung eines Stellwerks herangezogen werden. Zu den Model-Klassen gehören z. B. Abstellgleis, Zug, Ringlokschuppen, Weiche und Drehscheibe. Die Ausgabe und Eingabe (Steuerung) gehört mit zu den View-Klassen. Dieses kann mit einem graphischen Userinterface geschehen. Die Ausgabe sollte aber auch über die Signalanlagen, die Eingabe über Knöpfe/Schalter, sowie Signalgeber erfolgen.

Aufgabe: Geben Sie an, welche Methoden die verschiedenen View-Klassen benötigen, damit der Controller mit ihnen kommunizieren kann. Hinweis: Überlegen Sie z. B., was der View-Klasse für Weichen mitgeteilt werden muss, dass eine Weiche ihren Status gewechselt hat. Dafür wird eine Methode mit entsprechenden Parametern benötigt.
Lösung: Klasse WeicheView: aenderung(weichenNr, neueRichtung) wurdeUeberfahren(weichenNr, zug) Klasse GleisView: zugbelegung(gleisNr, zug) status(gleisNr, status)

Aufgabe: Erstelle eine Klassenkarte, die im Model ein Gleis repräsentiert.

Gleis
gleislaenge: int abstellgleis: boolean status: String vorgaenger: Gleis nachfolger: Gleis zug: Zug pos: Position
Gleis() istBesetzt(): boolean gibZug(): Zug setzeZug(zug: Zug) gibStatus(): String setzeNachfolger(nachfolger: Gleis) gibNachfolger(): Gleis setzeVorgaenger(vorgaenger: Gleis) gibVorgaenger(): Gleis

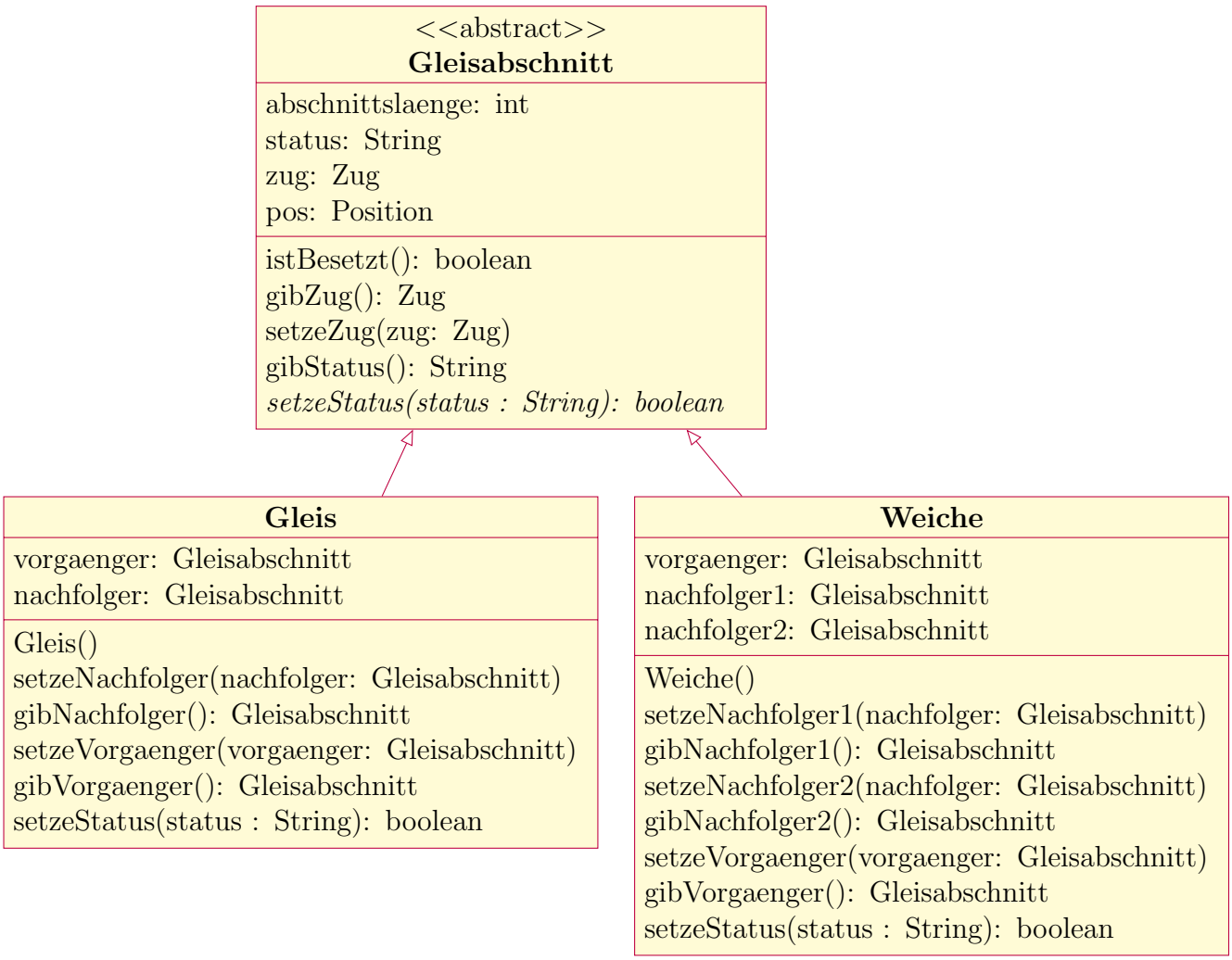
Abstrakte Klassen

Es gibt verschiedene Typen von Gleisen mit sehr ähnlichen Eigenschaften, wie z. B. ein einfaches Gleis, eine Weiche oder eine Drehscheibe. Die gleichen Eigenschaften lassen sich in einer gemeinsamen Elternklasse zusammenfassen, die hier als Gleisabschnitt bezeichnet wird.

Da wir kein Gleisabschnittsobjekt erzeugen wollen, wird diese Klasse als abstrakte Klasse definiert. Dadurch wird verhindert, dass davon Objekte erstellt werden, da die Programmiersprachen dieses nicht zulassen. In Java wird eine abstrakte Klasse durch das Schlüsselwort abstract gekennzeichnet:

```
public abstract class Gleisabschnitt {
        ...
}
```

In dieser Klasse sollten die Attribute nicht mit private geschützt werden, sondern mit protected. Dadurch können die Kindklassen auf diese Attribute direkt zugreifen. Ein direkter Zugriff von außen ist aber weiterhin nicht möglich.



In diesem Beispiel ist auch die Methode setzeStatus in der Klasse Gleisabschnitt als abstrakt deklariert worden. Dieses hat den Hintergrund, dass das Setzen des Status direkt vom Gleisabschnittstyp abhängt. Eine Weiche muss z. B. zusätzlich zu dem Status, ob sie belegt ist noch speichern, in welche Richtung sie zeigt. Außerdem sollte beim Setzen immer überprüft werden, ob der angegebene Status für diesen Typ passt. Deshalb soll die Methode nicht bereits in der Klasse Gleisabschnitt implementiert werden. Alle Kindklassen müssen diese Klasse aber enthalten. Daher gibt man in der Elternklasse diese Methode nur mit ihrem Kopf an:

```
public abstract boolean setzeStatus(String status);
```

Java fordert so nun ein, dass jede Kindklasse diese Methode implementiert oder wieder als abstrakte Klasse definiert ist.

Interfaces

Der Blick wird nun auf die View des Programms geworfen. Hier erfolgt die Eingabe und die Ausgabe des Programms. Die Eingabe kann dabei über verschiedene Wege passieren: Eine graphische Oberfläche (GUI = Graphic User Interface), die direkte Eingabe von Befehlen, über entfernten Netzwerkzugriff oder auch durch die Sensoren im Gleisbett. Dem Controller soll es aber aber egal sein, wie diese Eingaben geschehen und implementiert sind, da er unabhängig davon sein soll.

Aufgabe: Schreiben Sie Köpfe der Methoden für einen Eingabeteil einer View auf, die der Controller ansprechen kann. Lösung: