Rangierbahnhof

Anmerkungen

- grundlegendes Anforderungsniveau
- vorgesehene Bearbeitungszeit: 90 min

Aufgabe

Auf einem Rangierbahnhof werden Güterzüge in Waggons zerlegt und diese zu neuen Zügen zusammengestellt. Zum Rangieren verwendet man einen Ablaufberg. Die Rangierlokomotive kann einen auf dem Rangiergleis stehenden Zug auf den Ablaufberg ziehen. Auf dem Ablaufberg werden die Waggons der Reihe nach entkuppelt. Sie rollen dann eigenständig das Gefälle hinab und gelangen gemäß der jeweiligen Weichenstellung auf die vor Beginn des Rangiervorgangs leeren Rangiergleise.

1. Für die Modellierung des Rangiervorgangs besteht ein Güterzug aus keinem, einem oder mehreren Waggons und hat keine Lokomotive. Wir beschränken uns auf Züge mit maximal zwölf Waggons. Von jedem Waggon sind seine Start- und Zielbahnhofsnummer zu speichern.

1.1 Entwerfen Sie für einen Güterzug ein UML-Klassendiagramm mit den beiden Klassen *Zug* und *Waggon*. Sehen Sie in der Klasse *Waggon* Attribute und Methoden so vor, dass die im Text genannten Informationen modelliert und später abgefragt werden können. Bei der Modellierung der Klasse *Zug* sind auch Material 1 und aus Material 2 die in der Methode *WaggonsAbstossen*() verwendeten Methoden sowie die sich daraus ergebenden Attribute zu berücksichtigen.

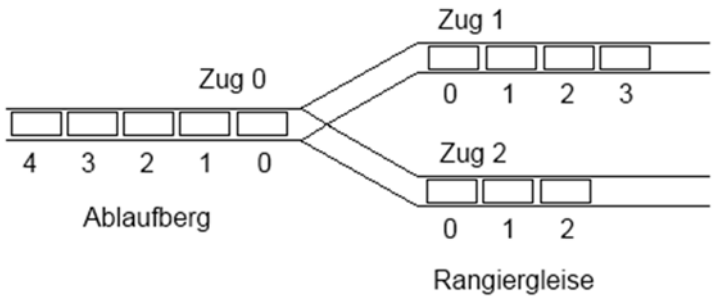
Begründen Sie die von Ihnen ausgewählte Klassenbeziehung.

1.2 Beschreiben Sie die beiden Konstruktoren der Klasse *Zug* in Material 1.

2.1 Das An- und Abkuppeln von Waggons kann beim Rangieren nur am Zugende stattfinden. Es wird davon ausgegangen, dass der letzte Waggon an der Position 0 im Feld gespeichert ist.

Implementieren Sie die Methode *ankuppeln(Waggon einWaggon)* der Klasse *Zug*, um einen Waggon an einen Zug anzukuppeln.

2.2 Die Methode *sucheMaxZiel()* bestimmt die größte Zielbahnhofsnummer der Waggons eines Zuges. Implementieren Sie *sucheMaxZiel()*.



3. Die Klasse *Rangierbahnhof* in Material 2 besitzt als Attribute die drei Züge, die in obiger Grafik dargestellt sind. *Zug0* ist dabei der zu rangierende Güterzug, dessen Waggons mit Hilfe der zwei anfangs leeren Züge *Zug1* und *Zug2* umrangiert werden sollen.

3.1 Analysieren Sie die beiden in Material 2 dargestellten Methoden *WaggonsAbstossen()* und *ZugAufDenAblaufberg(Zug einZug)*.

3.2 Analysieren und erklären Sie unter Einbezug von Material 2 den Algorithmus *Rangieren*, der in Material 3 als Struktogramm dargestellt ist.

4. Erläutern Sie unter Berücksichtigung aller Aufgabenteile, warum die Verwendung zweier Konstruktoren innerhalb der Klasse *Zug* sinnvoll ist.

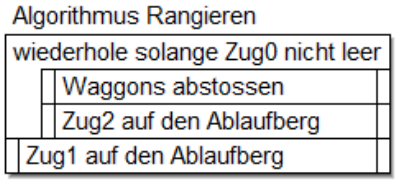
Material 1

```
public class Zug {
  private int Anzahl;
  private int MaxZahl = 12;
  private Waggon[] Waggons = new Waggon[MaxZahl];

  public Zug() {
    for (int i = 0; i < MaxZahl; i++)
      Waggons[i] = null;
    Anzahl = 0;
  }

  public Zug(int Start)  { // ein Testzug
    Waggons[ 0] = new Waggon(Start, 4);
    Waggons[ 1] = new Waggon(Start, 6);
    Waggons[ 2] = new Waggon(Start, 7);
    Waggons[ 3] = new Waggon(Start, 5);
    Waggons[ 4] = new Waggon(Start, 1);
    Waggons[ 5] = new Waggon(Start, 8);
    Waggons[ 6] = new Waggon(Start, 6);
    Waggons[ 7] = new Waggon(Start, 2);
    Waggons[ 8] = new Waggon(Start, 9);
    Waggons[ 9] = new Waggon(Start, 1);
    Waggons[10] = new Waggon(Start, 7);
    Waggons[11] = new Waggon(Start, 6);
    Anzahl = MaxZahl;
  }

  public int sucheMaxZiel() {
    // zu implementieren in Aufgabe 2.2
  }
  ...
}
```

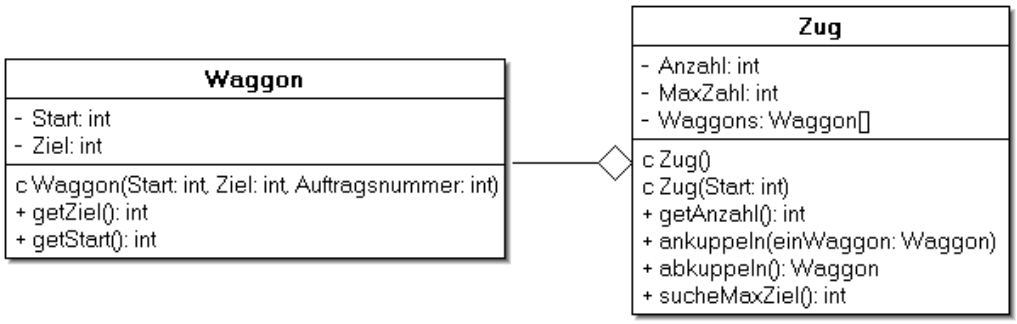
Material 2

```
public class Rangierbahnhof {
  private Zug Zug0;
  private Zug Zug1;
  private Zug Zug2;

  public void WaggonsAbstossen() {
    Waggon einWaggon;
    int MaxZiel = Zug0.sucheMaxZiel();
    int AnzahlWaggons = Zug0.getAnzahl();
    for (int i = 0; i < AnzahlWaggons; i++) {
      einWaggon = Zug0.abkuppeln();
      if (einWaggon.getZiel() < MaxZiel)
        Zug2.ankuppeln(einWaggon);
      else
        Zug1.ankuppeln(einWaggon);
    }
  }

  public void ZugAufDenAblaufberg(Zug einZug) {
    int AnzahlWaggons = einZug.getAnzahl();
    for (int i = 0; i < AnzahlWaggons; i++)
      Zug0.ankuppeln(einZug.abkuppeln());
  }

  ...
}
```

Material 3



Lösungshinweise

Aufg.	erwartete Leistungen
1.1	 Da ein Zug aus keinem, einem oder mehreren Waggons besteht, haben die beiden Klassen *Waggon* und *Zug* eine Aggregationsbeziehung.
1.2	Der erste Konstruktor hat keinen Parameter. Mittels einer for-Schleife wird jedes Element des Feldes *Waggons* mit *null* initialisiert. Die Anzahl der Waggons wird auf 0 gesetzt. Der zweite Konstruktor hat den Parameter *Start* vom Datentyp *int*. Durch Aufrufe des Konstruktors der Klasse *Waggon* mit dem Parameter *Start* sowie den angegebenen Werten für *Ziel* werden zwölf Waggons erzeugt und Verweise darauf in den Elementen des Feldes gespeichert. Die Anzahl der Waggons wird auf *MaxZahl* gesetzt.

2.1	`public void ankuppeln(Waggon einWaggon) {`
	`  if (Anzahl < MaxZahl) {`
	`  for (int i = Anzahl; i > 0; i--)`
	`  Waggons[i] = Waggons[i-1];`
	`  Waggons[0] = einWaggon;`
	`  Anzahl++;`
	`  }`
	`}`

2.2	`public int sucheMaxZiel() {`
	`  int MaxZiel = 0;`
	`  for (int i = 0; i < Anzahl; i++)`
	`  if (MaxZiel < Waggons[i].getZiel())`
	`  MaxZiel = Waggons[i].getZiel();`
	`  return MaxZiel;`
	`}`

3.1	Die Methode *WaggonsAbstossen()* hat keinen Parameter. Eine lokale Variable *einWaggon* vom Datentyp *Waggon* wird deklariert. Die lokale Variable *MaxZiel* erhält mit *Zug0.sucheMaxZiel()* die aktuell größte Zielbahnhofsnummer von *Zug0* als Wert und die lokale Variable *AnzahlWaggons* mit *Zug0.getAnzahl()* die Anzahl der Waggons von *Zug0*. In der for-Schleife wird bei jedem Schleifendurchlauf ein Waggon von *Zug0* abgekuppelt und in *einWaggon* zwischengespeichert. In einer bedingten Anweisung wird geprüft, ob die Zielbahnhofsnummer des

Aufg.	erwartete Leistungen
	Waggons kleiner als *MaxZiel ist.* Trifft dies zu, so wird der Waggon an *Zug2* angekuppelt, ansonsten an *Zug1*. Die Methode *WaggonsAbstoßen()* kuppelt die Waggons von *Zug0* an die Züge *Zug1* und *Zug2* an. Waggons mit der aktuell maximalen Zielbahnhofsnummer von *Zug0* werden an *Zug1* angekuppelt, die anderen an *Zug2*. *Die Methode ZugAufDenBerg()* wird mit dem Parameter *einZug* vom Datentyp *Zug* aufgerufen. Die lokale Variable *AnzahlWaggons erhält mit einZug.getAnzahl()* die Anzahl der Waggons von *einZug*. In der for-Schleife werden alle Waggons von *einZug* abgekuppelt und an *Zug0* angekuppelt. Wenn die Schleife beendet ist, hat *einZug* keine Waggons mehr und alle Waggons befinden sich in *Zug0* auf dem Ablaufberg.
3.2	Der Algorithmus beginnt mit einer Schleife, die terminiert wenn *Zug0* leer ist. Solange *Zug0* noch Waggons hat, werden die Waggons mit der aktuell größten Zielbahnhofsnummer mittels *WaggonsAbstossen()* an *Zug1* angekuppelt, alle anderen Waggons an *Zug2*. Letztere werden anschließend wieder an *Zug0* angekuppelt. Im nächsten Durchlauf wird die größte Zielbahnhofsnummer der verbliebenen Waggons bestimmt und ebenso verfahren. Der Vorgang ist abgeschlossen, wenn alle Waggons an *Zug1* angekuppelt worden sind und somit *Zug0* leer ist. Zum Schluss werden alle Waggons wieder von *Zug1* an *Zug0* angekuppelt. Die Waggons werden durch den Rangier-Algorithmus nach der Zielbahnhofsnummer sortiert.
4	Mit dem ersten Konstruktor kann man einen leeren Zug erzeugen. Dieser Konstruktor wird für die Erzeugung der anfangs leeren Züge *Zug1* und *Zug2* benötigt. Der zweite Konstruktor erzeugt einen vollständigen Güterzug mit zwölf Waggons. Mit ihm wird *Zug0* erzeugt, damit man einen zu rangierenden Zug simulieren kann.

Quelle: Hessisches Kultusministerium, Landesabitur Informatik Grundkurs, 2013
Lösungshinweise bearbeitet

Zuordnung zu den Prozess-, Inhalts- und Anforderungsbereichen

Aufg.	Prozessbereiche					Inhaltsbereiche					Bewertungseinheiten in Anforderungsbereichen		
	MI	BB	SV	KK	DI	ID	AL	SA	IS	IMG	I	II	III
1.1	X	X			X				X		5	7	
1.2				X			X				3	3	
2.1	X						X				2	4	2
2.2	X						X					4	4
3.1				X	X		X				4	7	1
3.2				X	X		X				4	4	2
4		X					X					1	3
Summe 60											18	30	12