Name: _____

Abiturprüfung 2020

Informatik, Leistungskurs

Aufgabenstellung:

Ein Eisenbahnunternehmen möchte mit einer Software die Sitzplatzreservierungen für die Waggons von Zügen verwalten.

Die Sitzplätze eines Waggons sind fortlaufend nummeriert, beginnend mit der Nummer 0. Alle Sitzplätze sind in Vierer-Sitzgruppen angeordnet; jeweils vier Plätze mit fortlaufenden Platznummern gehören zu einer Sitzgruppe.

In Abbildung 1 wird ein Ausschnitt eines Zuges mit zwei Waggons dargestellt. „Waggon 15“ hat 16 Sitzplätze, „Waggon 21“ hat 24 Sitzplätze. Sitzgruppen sind mit einem Rahmen dargestellt: z. B. die Plätze 12, 13, 14 und 15 bilden in beiden Waggons jeweils eine Sitzgruppe. Reservierte Plätze sind grau markiert.

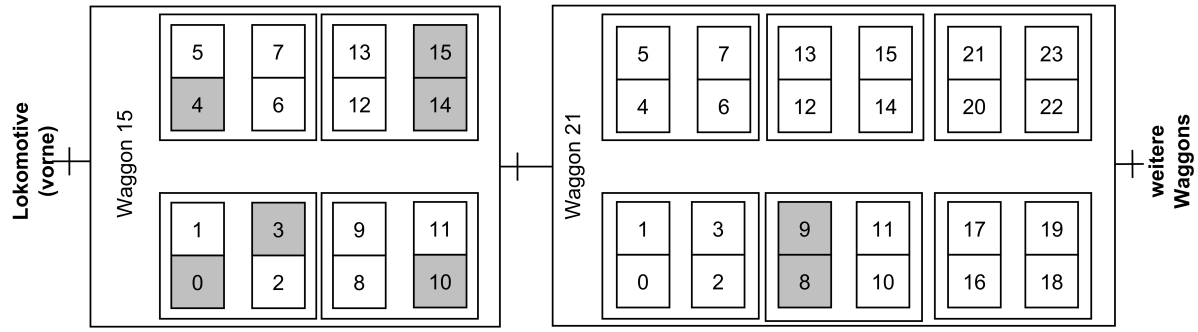


Abbildung 1: Zwei Waggons eines Beispielzuges mit Sitzplatznummern

Eine erste Modellierung für die Verwaltung der Sitzplatzreservierungen ist im folgenden Implementationsdiagramm dargestellt. Die Dokumentationen der Klassen befinden sich im Anhang.

Name: _____

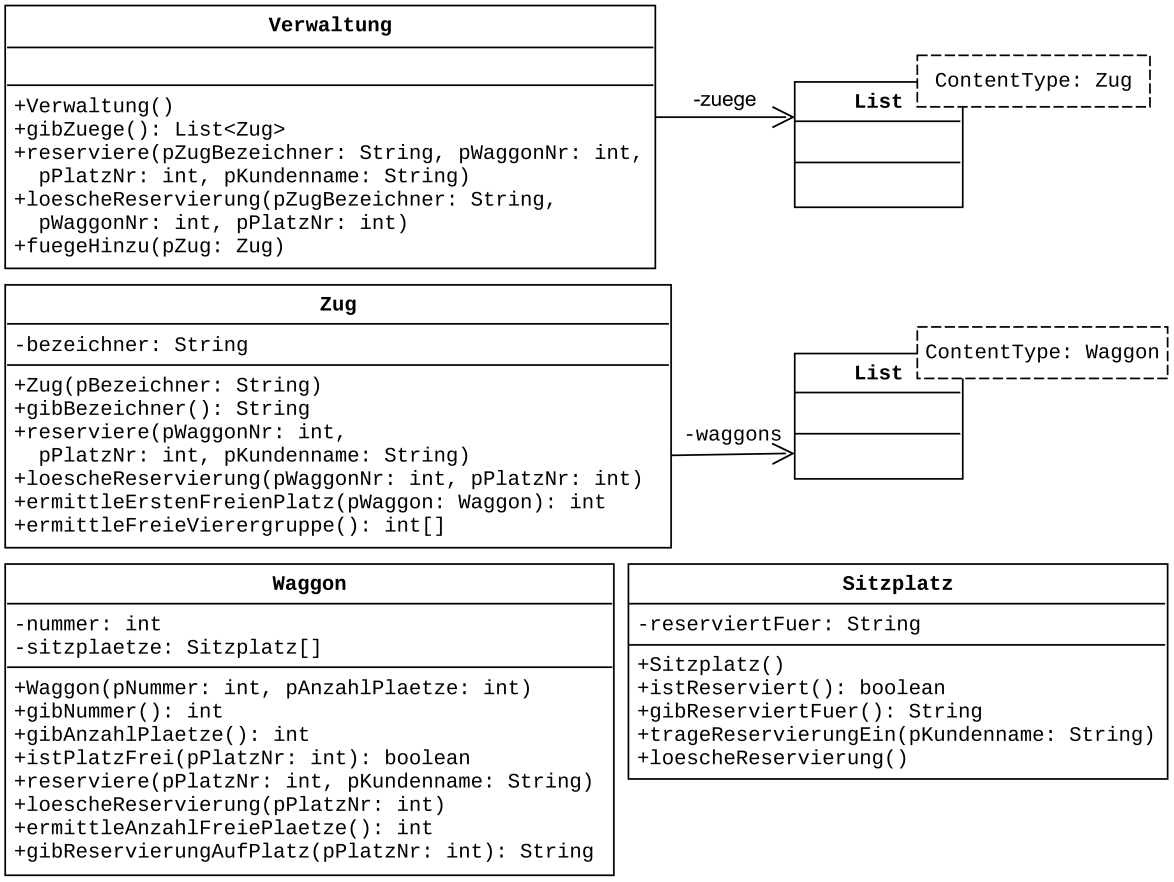


Abbildung 2: Teilmodellierung als Implementationsdiagramm

a) *Erläutern Sie mit Bezug auf Abbildung 2 die Beziehungen zwischen den Klassen* `Verwaltung`, `Zug`, `Waggon` *und* `Sitzplatz`.

Erläutern Sie im Sachkontext, welche Argumente dafür sprechen, die Datensammlung für die Züge als lineare Liste und die Datensammlung für die Sitzplätze als Feld (Array) zu realisieren.

(7 Punkte)

In einem Objekt der Klasse `Waggon` hat das Feld `sitzplaetze` so viele Elemente wie durch den Parameter des Konstruktors `pAnzahlPlaetze` angegeben wird. Die Sitzplatznummer entspricht dem Index im Feld `sitzplaetze`. Wenn es eine Reservierung für einen Sitzplatz gibt, so wird der Kundenname im entsprechenden Objekt der Klasse `Sitzplatz` eingetragen.

Name: _____

b) Die Klasse Zug erhält zu Optimierungszwecken eine zusätzliche Methode:

```
 1 public void wasMacheIch(Waggon pWaggon) {
 2   waggons.toFirst();
 3   for (int j = 0; j < pWaggon.gibAnzahlPlaetze(); j++) {
 4     if (!pWaggon.istPlatzFrei(j)){
 5       while (waggons.hasAccess()
 6           && (pWaggon == waggons.getContent()
 7               || waggons.getContent().
 8                   ermittleAnzahlFreiePlaetze() == 0)) {
 9         waggons.next();
10       }
11       if (waggons.hasAccess()) {
12         Waggon aktW = waggons.getContent();
13         int i = ermittleErstenFreienPlatz(aktW);
14         aktW.reserviere(i,
15             pWaggon.gibReservierungAufPlatz(j));
16         pWaggon.loescheReservierung(j);
17       }
18     }
19   }
20 }
```

Ermitteln Sie im Sachzusammenhang, inwieweit sich der in Abbildung 1 dargestellte Zug durch den Aufruf der Methode `wasMacheIch` *mit dem in der Abbildung 1 als „Waggon 21“ dargestellten Waggonobjekt als Parameter verändert.*

Erläutern Sie die Funktionsweise der Methode `wasMacheIch` *für die Zeilen 5 bis 10 und für die Zeilen 11 bis 17.*

Erläutern Sie die Aufgabe der Methode `wasMacheIch` *im Sachzusammenhang.*

(12 Punkte)

Name: _____

c) Um Gruppen von Reisenden zu ermöglichen, nahe beieinander zu sitzen, soll es möglich sein, Vierer-Sitzgruppen zu reservieren. Die Methode `ermittleFreieVierergruppe` der Klasse `Zug` soll einen Waggon des Zuges suchen, in dem es eine Vierer-Sitzgruppe gibt, bei der alle vier Plätze noch nicht reserviert sind. Als Rückgabe soll sie ein Feld (Array) aus zwei Zahlen liefern: die Nummer des gefundenen Waggons und die kleinste Platznummer in der freien Vierer-Sitzgruppe.

Für den Zug in Abbildung 1 würde im „Waggon 15“ keine freie Vierer-Sitzgruppe gefunden. Im „Waggon 21“ würde die Vierer-Sitzgruppe mit den Plätzen Nr. 0 bis 3 gefunden und somit die beiden Zahlen 21 (für Waggon 21) und 0 (für die kleinste Platznummer der Vierer-Sitzgruppe) im Feld `[21, 0]` zurückgegeben.

Wenn es in keinem der Waggons eine freie Vierer-Sitzgruppe gibt, so soll die Methode zweimal den Wert `-1` im Feld `[-1, -1]` zurückgeben.

Entwickeln und erläutern Sie ein algorithmisches Verfahren für die Methode.

Implementieren Sie die Methode mit dem folgenden Methodenkopf:

```
public int[] ermittleFreieVierergruppe()
```

(15 Punkte)

d) Das Modell soll um folgende Anforderungen erweitert werden:

> In Zukunft soll die Verwaltung nicht nur Reservierungen vornehmen, sondern auch noch Kunden verwalten können. In der bisherigen Modellierung wurden die Kunden nur über die Namen identifiziert. Nun sollen zu den Kunden nicht nur die Namen, sondern auch die Vornamen und eine Kundennummer gespeichert werden. Die Verwaltung soll Zugriff auf alle Kundendaten haben, um den Kundenstamm pflegen zu können. Die Verwaltung soll alle Kundendaten auf Anfrage zurückliefern können.

Modellieren Sie die oben genannten Anforderungen als Erweiterung der Modellierung aus Abbildung 2 und stellen Sie dabei nur die Veränderungen und Erweiterungen in Form eines Implementationsdiagramms dar.

Erläutern Sie zu allen Klassen, ob und welche Veränderungen durchgeführt werden müssen.

(12 Punkte)

Name: _____

e) Der Praktikant Thomas schlägt eine alternative Modellierung vor:

> „Jedes Sitzplatzobjekt hat eine Referenz auf sein Waggonobjekt, und jedes Waggonobjekt hat eine Referenz auf sein Zugobjekt, die Referenz auf das Feld aus Sitzplätzen entfällt. Dafür verwaltet die Klasse `Verwaltung` außer der Zugliste auch alle Sitzplatzobjekte in einer Liste."

Das findet die Praktikantin Johanna nicht sehr praktisch. Sie behauptet, dass man dann deutlich mehr Arbeitsschritte benötigt, um z. B. die Anzahl der nicht reservierten Plätze in einem bestimmten Waggon zu ermitteln.

Beurteilen Sie den alternativen Modellierungsvorschlag von Thomas im Hinblick auf Johannas Behauptung.

(4 Punkte)

Zugelassene Hilfsmittel:

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung

Name: _____

Anhang

Dokumentationen der verwendeten Klassen

Die Klasse `Sitzplatz`

Objekte der Klasse **`Sitzplatz`** verwalten je einen Sitzplatz in einem Zugwaggon. Sitzplätze können z. B. reserviert werden.

Auszug aus der Dokumentation der Klasse `Sitzplatz`

Konstruktor **`Sitzplatz()`**
Ein Sitzplatzobjekt wird initialisiert. Das Sitzplatzobjekt ist zunächst nicht reserviert.

Anfrage **`boolean istReserviert()`**
Die Anfrage liefert `true`, wenn der Sitzplatz reserviert ist, ansonsten liefert sie `false`.

Anfrage **`String gibReserviertFuer()`**
Die Anfrage liefert den Namen der Person, die diesen Sitzplatz reserviert hat. Gibt es keine Reservierung, wird `null` zurückgegeben.

Auftrag **`void trageReservierungEin(String pKundenname)`**
Der Auftrag bewirkt, dass dieser Sitzplatz für die Person mit dem Namen `pKundenname` reserviert ist. Ist bereits eine Reservierung eingetragen, so passiert nichts.

Auftrag **`void loescheReservierung()`**
Der Auftrag löscht eine ggf. vorhandene Reservierung für diesen Sitzplatz.

Name: _____

Die Klasse `Waggon`

Objekte der Klasse **`Waggon`** verwalten die Sitzplätze eines Waggons. Waggons haben immer eine durch vier teilbare Anzahl an Sitzplätzen. Die Sitzplätze sind beginnend mit 0 nummeriert.

Auszug aus der Dokumentation der Klasse `Waggon`

Konstruktor **`Waggon(int pNummer, int pAnzahlPlaetze)`**
Ein Waggonobjekt mit der Nummer `pNummer` und mit `pAnzahlPlaetze` Sitzplätzen wird initialisiert. Zu Anfang ist kein Sitzplatz reserviert.

Anfrage **`int gibNummer()`**
Die Anfrage liefert die Nummer des Waggons.

Anfrage **`int gibAnzahlPlaetze()`**
Die Anfrage liefert die Anzahl der Sitzplätze des Waggons.

Anfrage **`boolean istPlatzFrei(int pPlatzNr)`**
Die Anfrage liefert `true`, wenn es den Sitzplatz mit der Nummer `pPlatzNr` im Waggon gibt und er nicht reserviert ist, ansonsten liefert sie `false`.

Auftrag **`void reserviere(int pPlatzNr, String pKundenname)`**
Der Auftrag trägt eine Reservierung für die Person mit dem Namen `pKundenname` auf den Sitzplatz mit der Nummer `pPlatzNr` ein. Wenn es die gewünschte Sitzplatznummer im Waggon nicht gibt oder der Platz bereits reserviert ist, so passiert nichts.

Auftrag **`void loescheReservierung(int pPlatzNr)`**
Der Auftrag löscht die Reservierung auf dem Sitzplatz mit der Nummer `pPlatzNr`. Wenn es in diesem Waggon keinen Platz mit der Platznummer `pPlatzNr` gibt, so passiert nichts.

Anfrage **`int ermittleAnzahlFreiePlaetze()`**
Die Anfrage liefert die Anzahl der Sitzplätze ohne Reservierung.

Anfrage **`String gibReservierungAufPlatz(int pPlatzNr)`**
Die Anfrage liefert den Kundennamen, der auf dem Platz mit der Nummer `pPlatzNr` eingetragen ist. Wenn es im diesem Waggon keinen Platz mit der Platznummer `pPlatzNr` gibt oder der Platz nicht reserviert ist, wird `null` zurückgegeben.

Name: _____

Die Klasse Zug

Ein Objekt der Klasse **`Zug`** verwaltet seine Waggons.

Auszug aus der Dokumentation der Klasse Zug

Konstruktor **`Zug(String pBezeichner)`**
Ein Zugobjekt mit der Bezeichnung `pBezeichner` wird initialisiert.
Das Zugobjekt erzeugt und verwaltet eine Liste von Waggons.

Anfrage **`String gibBezeichner()`**
Die Anfrage liefert die Bezeichnung des Zugs.

Auftrag **`void reserviere(int pWaggonNr, int pPlatzNr, String pKundenname)`**
Der Auftrag trägt eine Reservierung für den Waggon mit der Nummer `pWaggonNr` auf dem Sitzplatz mit der Nummer `pPlatzNr` für die Person mit dem Namen `pKundenname` ein. Wenn es im Zug keinen Waggon mit der angegebenen Waggonnummer gibt oder die Platznummer im Waggon nicht vorhanden ist oder der Platz bereits reserviert war, so passiert nichts.

Auftrag **`void loescheReservierung(int pWaggonNr, int pPlatzNr)`**
Der Auftrag löscht die Reservierung im Waggon mit der Nummer `pWaggonNr` auf dem Sitzplatz mit der Nummer `pPlatzNr`. Wenn es im Zug keinen Waggon mit der angegebenen Waggonnummer gibt oder die Platznummer im Waggon nicht vorhanden ist, so passiert nichts.

Anfrage **`int ermittleErstenFreienPlatz(Waggon pWaggon)`**
Die Anfrage ermittelt die Platznummer des ersten freien Sitzplatzes im Waggon `pWaggon` und gibt sie zurück. Gibt es keinen freien Sitzplatz, so wird `-1` zurückgegeben.

Anfrage **`int[] ermittleFreieVierergruppe()`**
Die Anfrage ist in Teilaufgabe c) zu implementieren.

Name: _____

Die Klasse `Verwaltung`

Ein Objekt der Klasse **`Verwaltung`** verwaltet Züge.

Auszug aus der Dokumentation der Klasse `Verwaltung`

Konstruktor **`Verwaltung()`**
Ein Verwaltungsobjekt wird initialisiert.

Anfrage **`List<Zug> gibZuege()`**
Die Anfrage liefert eine Liste aller Züge.

Auftrag
```
void reserviere(String pZugBezeichner,
      int pWaggonNr, int pPlatzNr, String pKundenname)
```
Der Auftrag trägt eine Reservierung für die Person mit dem Namen `pKundenname` auf den Sitzplatz mit der Nummer `pPlatzNr` im Waggon mit der Nummer `pWaggonNr` im Zug mit der Bezeichnung `pZugBezeichner` ein. Wenn es den Zug mit der genannten Bezeichnung nicht gibt oder der Zug keinen Waggon mit der Nummer `pWaggonNr` besitzt oder es keinen Sitzplatz mit der Nummer `pPlatzNr` im Waggon gibt oder der Platz bereits reserviert war, so passiert nichts.

Auftrag
```
void loescheReservierung(String pZugBezeichner,
                         int pWaggonNr, int pPlatzNr)
```
Der Auftrag löscht die Reservierung im Zug mit der Bezeichnung `pZugBezeichner` im Waggon mit der Nummer `pWaggonNr` auf dem Sitzplatz mit der Nummer `pPlatzNr`. Wenn es den Zug mit der genannten Bezeichnung nicht gibt oder der Zug keinen Waggon mit der Nummer `pWaggonNr` besitzt oder es keinen Sitzplatz mit der Nummer `pPlatzNr` im Waggon gibt, so passiert nichts.

Auftrag **`void fuegeHinzu(Zug pZug)`**
Der Auftrag hängt den Zug `pZug` hinten an die Liste der Züge an.

Name: _____

Die generische Klasse List<ContentType>

Objekte der generischen Klasse **`List`** verwalten beliebig viele, linear angeordnete Objekte vom Typ **`ContentType`**. Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt. Das erste oder das letzte Objekt einer Liste können durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden.

Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt werden.

Dokumentation der Klasse `List<ContentType>`

Konstruktor **`List()`**

Eine leere Liste wird erzeugt. Objekte, die in dieser Liste verwaltet werden, müssen vom Typ `ContentType` sein.

Anfrage **`boolean isEmpty()`**

Die Anfrage liefert den Wert `true`, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert `false`.

Anfrage **`boolean hasAccess()`**

Die Anfrage liefert den Wert `true`, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert `false`.

Auftrag **`void next()`**

Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt, d. h., `hasAccess()` liefert den Wert `false`.

Auftrag **`void toFirst()`**

Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Auftrag **`void toLast()`**

Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Name: _____

Anfrage **`ContentType getContent()`**

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt zurückgegeben. Andernfalls (`hasAccess() == false`) gibt die Anfrage den Wert `null` zurück.

Auftrag **`void setContent(ContentType pContent)`**

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`) und `pContent` ungleich `null` ist, wird das aktuelle Objekt durch `pContent` ersetzt. Sonst bleibt die Liste unverändert.

Auftrag **`void append(ContentType pContent)`**

Ein neues Objekt `pContent` wird am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (`hasAccess() == false`).
Falls `pContent` gleich `null` ist, bleibt die Liste unverändert.

Auftrag **`void insert(ContentType pContent)`**

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird ein neues Objekt `pContent` vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert.
Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (`hasAccess() == false`), wird `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt.
Falls es kein aktuelles Objekt gibt (`hasAccess() == false`) und die Liste nicht leer ist oder `pContent == null` ist, bleibt die Liste unverändert.

Auftrag **`void concat(List<ContentType> pList)`**

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls es sich bei der Liste und `pList` um dasselbe Objekt handelt, `pList == null` oder eine leere Liste ist, bleibt die Liste unverändert.

Auftrag **`void remove()`**

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr (`hasAccess() == false`). Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (`hasAccess() == false`), bleibt die Liste unverändert.

Unterlagen für die Lehrkraft

Abiturprüfung 2020

Informatik, Leistungskurs

1. Aufgabenart

Modellierung, Implementation und Analyse kontextbezogener Problemstellungen mit Schwerpunkt auf den Inhaltsfeldern Daten und ihre Strukturierung und Algorithmen

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2020

Die Aufgaben weisen vielfältige Bezüge zu den Kompetenzerwartungen und Inhaltsfeldern des Kernlehrplans bzw. zu den in den Vorgaben ausgewiesenen Fokussierungen auf.
Im Folgenden wird auf Bezüge von zentraler Bedeutung hingewiesen.

1. *Inhaltsfelder und inhaltliche Schwerpunkte*
   Daten und ihre Strukturierung
   - Objekte und Klassen
     - Entwurfsdiagramme und Implementationsdiagramme
     - Lineare Strukturen
       Array bis zweidimensional
       Lineare Liste (Klasse `List`*)*

   Algorithmen
   - Analyse, Entwurf und Implementierung von Algorithmen
   - Algorithmen in ausgewählten informatischen Kontexten

   Formale Sprachen und Automaten
   - Syntax und Semantik einer Programmiersprache
     - Java

2. *Medien/Materialien*
   - entfällt

[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

5. Zugelassene Hilfsmittel

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung

6. Modelllösungen

Die jeweilige Modelllösung stellt eine mögliche Lösung bzw. Lösungsskizze dar. Der gewählte Lösungsansatz und -weg der Schülerinnen und Schüler muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden mit entsprechender Punktzahl bewertet (Bewertungsbogen: Zeile „Sachlich richtige Lösungsalternative zur Modelllösung“).

Teilaufgabe a)

Ein Objekt der Klasse `Verwaltung` verwaltet seine Züge in einer linearen Liste mit Objekten der Klasse `Zug`. Diese Züge verwalten ihre Waggons jeweils in einer weiteren linearen Liste, welche Objekte vom Typ `Waggon` enthält. Jedes Objekt der Klasse `Waggon` hat ein Feld von Objekten der Klasse `Sitzplatz`, in dem alle Sitzplatzobjekte gespeichert werden, die zu Sitzplätzen dieses Waggons gehören.

Für die Verwaltung der Züge wird eine Liste `zuege` gewählt, weil jederzeit Züge hinzugefügt werden können, die Anzahl der Züge unbekannt ist und die Datensammlung daher dynamisch sein sollte. Die Verwaltung der Sitzplätze in der Klasse `Waggon` ist hingegen als Feld modelliert, weil sich die Anzahl der Sitzplätze eines Waggons zur Laufzeit nicht ändert und man auf die Sitzplätze über die Sitzplatznummern direkt per Index zugreifen kann.

Teilaufgabe b)

Der Sitzplatz Nummer 1 im „Waggon 15“ wird reserviert mit dem Kundennamen, der zuvor im „Waggon 21“ auf Platz Nummer 8 als Reservierung eingetragen war. Der Sitzplatz Nummer 2 im „Waggon 15“ wird reserviert mit dem Kundennamen, der zuvor im „Waggon 21“ auf Platz Nummer 9 eingetragen war. Die Plätze 8 und 9 im „Waggon 21“ sind nach dem Methodenaufruf wieder frei.

Zeilen 5 bis 10: In der Waggonliste des Zuges wird nach einem Waggon gesucht, der nicht der Waggon `pWaggon` ist und der noch freie Plätze hat.

Zeilen 11 bis 17: Wenn ein solcher Waggon gefunden werden konnte, wird die Platznummer des ersten freien Sitzplatzes dieses Waggons in der lokalen Variablen gespeichert (Zeile 13). Der Kundenname, der in `pWaggon` als Reservierung eingetragen war, wird auf dem zuvor gespeicherten Sitzplatz eingetragen (Zeilen 14 - 15). Die Reservierung in `pWaggon` wird anschließend gelöscht (Zeile 16).

Aufgabe der Methode `wasMacheIch`: Die Methode bucht so viele Reservierungen wie möglich aus dem Waggon `pWaggon` auf andere Waggons dieses Zuges um. Dabei werden der Reihe nach alle freien Plätze in den Waggons des Zuges von vorne bis hinten gefüllt.

Teilaufgabe c)

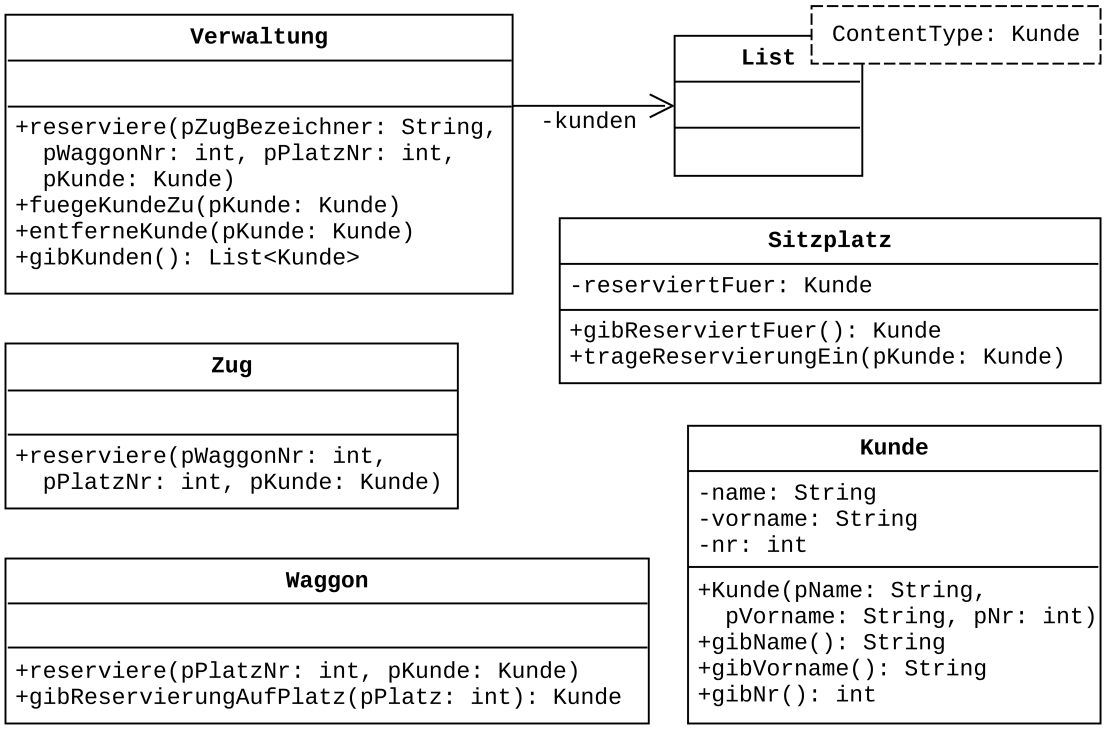
Ein algorithmisches Verfahren:
Man muss die Waggonliste des Zuges systematisch durchsuchen. Für jeden Waggon muss man, mit dem Sitzplatz `0` beginnend, in einer Schleife jeden vierten Sitzplatz untersuchen. Für jeden der untersuchten Sitzplätze ist zu prüfen, ob dieser Sitzplatz und die drei folgenden Sitzplätze nicht reserviert sind. Sind alle vier Sitzplätze noch frei, so hat man eine freie Vierergruppe gefunden und speichert im Rückgabefeld die Waggonnummer und die Nummer des ersten Sitzplatzes der Vierergruppe. Ansonsten muss man – mit einem Viererschritt – den ersten Sitzplatz der nächsten Vierer-Sitzgruppe analog untersuchen. Sobald die Sitzplatznummern der zu untersuchenden Sitzplätze größer sind als die Platzanzahl im Waggon, kann man die Suche abbrechen. Dann gibt es keine freie Vierer-Sitzgruppe in diesem Waggon, und man muss den nächsten Waggon der Waggonliste analog untersuchen. Wenn in der ganzen Waggonliste keine freie Vierer-Sitzgruppe gefunden wurde, so wird das mit dem Inhalt `[-1, -1]` initialisierte Feld zurückgegeben. Ansonsten wird das mit der Waggonnummer und der Sitzplatznummer der gefundenen Vierer-Sitzgruppe gefüllte Feld zurückgegeben.

Implementierte Methode:

```
public int[] ermittleFreieVierergruppe() {
  boolean gefunden = false;
  int[] rueckgabe = new int[2];
  rueckgabe[0] = -1;
  rueckgabe[1] = -1;
  waggons.toFirst();
  while (waggons.hasAccess() && !gefunden) {
    int platznummer = 0;
    Waggon aktWaggon = waggons.getContent();
    while (platznummer <= aktWaggon.gibAnzahlPlaetze() - 4
           && !gefunden) {
      if (aktWaggon.istPlatzFrei(platznummer)
          && aktWaggon.istPlatzFrei(platznummer + 1)
          && aktWaggon.istPlatzFrei(platznummer + 2)
          && aktWaggon.istPlatzFrei(platznummer + 3)) {
        gefunden = true;
        rueckgabe[0] = aktWaggon.gibNummer();
        rueckgabe[1] = platznummer;
      }
      platznummer = platznummer + 4;
    }
    waggons.next();
  }
  return rueckgabe;
}
```

Teilaufgabe d)

Nur die Änderungen sind im folgenden Implementationsdiagramm notiert:



Für die Erweiterung muss man eine Klasse `Kunde` modellieren mit Attributen vom Typ `String` für den Namen und den Vornamen und einem ganzzahligen Attribut für die Kundennummer. Die Klasse erhält einen Konstruktor mit den beiden Zeichenketten für Vor- und Nachname und einer ganzzahligen Nummer als Parametern zum Initialisieren eines neuen Kundenobjektes.

Die Reservierungen auf den Sitzplätzen werden mit Kundenobjektreferenzen eingetragen und verwaltet (statt mit den Kundennamen).

In den Klassen `Waggon`, `Zug` und `Verwaltung` sollen Reservierungen vorgenommen werden: Die Parameterlisten für die entsprechenden Methoden `reserviere` müssen dafür ebenfalls von Kundennamen zu Kundenobjektreferenzen verändert werden.

In der Klasse `Waggon` muss auch die Anfrage `gibReservierungAufPlatz` so geändert werden, dass statt einer Zeichenkette eine Objektreferenz zurückgegeben wird.

Die Klasse `Verwaltung` soll laut Anforderung alle Kunden verwalten können. Da der Kundenstamm sich zur Laufzeit ändern kann, wird dafür die dynamische Datenstruktur der linearen Liste gewählt. Die Klasse `Verwaltung` erhält zusätzlich eine Methode, um eine Liste aller Kunden zurückzugeben.

Teilaufgabe e)

Bei der alternativen Modellierung muss man alle Sitzplätze aller Züge, also die gesamte Sitzplatzliste der Verwaltung, durchsuchen, um die Sitzplätze herauszufiltern, die im gewünschten Waggon liegen und die keine Reservierung haben.

Da die Summe der Sitzplätze in allen Waggons größer ist als die Anzahl der Sitzplätze in einem Waggon, ist der Aufwand bei der alternativen Modellierung größer. Daher ist die alternative Modellierung im Hinblick auf das genannte Problem abzulehnen.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	erläutert die Beziehungen zwischen den Klassen.	3			
2	erläutert im Sachkontext, welche Argumente dafür sprechen, die Datensammlung für die Züge als Liste und die Datensammlung für die Sitzplätze als Feld zu realisieren.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (7) …… ……					
	Summe Teilaufgabe a)	**7**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	ermittelt im Sachzusammenhang, inwieweit sich der dargestellte Zug durch den Aufruf der Methode mit dem in Abbildung 1 als „Waggon 21“ dargestellten Waggonobjekt als Parameter verändert.	3			
2	erläutert die Funktionsweise für die Zeilen 5 bis 10 und für die Zeilen 11 bis 17.	6			
3	erläutert die Aufgabe der Methode im Sachzusammenhang.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe b)	**12**			

[2] EK = Erstkorrektur; ZK = Zweitkorrektur; DK = Drittkorrektur

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt und erläutert ein algorithmisches Verfahren für die Methode.	6			
2	implementiert die Methode.	9			
Sachlich richtige Lösungsalternative zur Modelllösung: (15) …… ……					
	Summe Teilaufgabe c)	**15**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	modelliert die genannten Anforderungen als Erweiterung der Modellierung und stellt nur die Veränderungen und Erweiterungen als Implementationsdiagramm dar.	6			
2	erläutert zu allen Klassen, ob und welche Veränderungen durchgeführt werden müssen.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe d)	**12**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	beurteilt den alternativen Modellierungsvorschlag im Hinblick auf die Behauptung.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (4) …… ……					
	Summe Teilaufgabe e)	**4**			

	Summe insgesamt	**50**			

Festlegung der Gesamtnote (Bitte nur bei der letzten bearbeiteten Aufgabe ausfüllen.)

	Lösungsqualität			
	maximal erreichbare Punktzahl	EK	ZK	DK
Übertrag der Punktsumme aus der ersten bearbeiteten Aufgabe	**50**			
Übertrag der Punktsumme aus der zweiten bearbeiteten Aufgabe	**50**			
Übertrag der Punktsumme aus der dritten bearbeiteten Aufgabe	**50**			
Punktzahl der gesamten Prüfungsleistung	**150**			
aus der Punktsumme resultierende Note gemäß nachfolgender Tabelle				
Note ggf. unter Absenkung um bis zu zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOSt				
Paraphe				

Berechnung der Endnote nach Anlage 4 der Abiturverfügung auf der Grundlage von § 34 APO-GOSt

Die Klausur wird abschließend mit der Note _____ (____ Punkte) bewertet.

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

Für die Zuordnung der Notenstufen zu den Punktzahlen ist folgende Tabelle zu verwenden:

Note	**Punkte**	**Erreichte Punktzahl**
sehr gut plus	15	150 – 143
sehr gut	14	142 – 135
sehr gut minus	13	134 – 128
gut plus	12	127 – 120
gut	11	119 – 113
gut minus	10	112 – 105
befriedigend plus	9	104 – 98
befriedigend	8	97 – 90
befriedigend minus	7	89 – 83
ausreichend plus	6	82 – 75
ausreichend	5	74 – 68
ausreichend minus	4	67 – 60
mangelhaft plus	3	59 – 50
mangelhaft	2	49 – 41
mangelhaft minus	1	40 – 30
ungenügend	0	29 – 0