Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Architecture-driven Requirements Engineering (ARE)
https://sdqweb.ipd.kit.edu/wiki/Programmieren
programmieren-vorlesung@ipd.kit.edu

Programmieren – Wintersemester 2019/20

Abschlussaufgabe 1 20 Punkte
Version 1.7

Ausgabe: 10.02.2020, ca. 13:00 Uhr
Praktomat: 24.02.2020, 13:00 Uhr
Abgabefrist: 10.03.2020, 06:00 Uhr

Abgabehinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe dieses Blattes notwendig ist. Der Praktomat wird Ihre Abgabe zurückweisen, falls eine der nachfolgenden Regeln verletzt ist. Eine zurückgewiesene Abgabe wird automatisch mit null Punkten bewertet. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie keine Elemente der Java-Bibliotheken, ausgenommen Elemente der Pakete `java.lang`, `java.util`, `java.util.regex`, `java.util.function` und `java.util.stream`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 120 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute.
- Nutzen Sie nicht das `default`-Package.
- `System.exit()` und `Runtime.exit()` dürfen nicht verwendet werden.
- Halten Sie die Regeln zur Javadoc-Dokumentation ein.
- Halten Sie auch alle anderen Checkstyle-Regeln an.

Bearbeitungshinweise

Diese Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe, jedoch wird der Praktomat Ihre Abgabe **nicht** zurückweisen, falls eine der nachfolgenden Regeln verletzt ist.

- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im ILIAS-Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

Plagiat

Es werden nur selbstständig erarbeitete Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen wie beispielsweise der Lösungsvorschläge des Übungsbetriebes, ist ein Täuschungsversuch und führt zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden und alles was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden. Für weitere Ausführungen sei auf die Einverständniserklärung (Disclaimer) verwiesen.

Checkstyle

Der Praktomat überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen der Praktomat die Abgabe zurückweist, da diese Regel verpflichtend einzuhalten ist. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki (ILIAS) beschreibt, wie das funktioniert.

Terminal-Klasse

Laden Sie für diese Aufgabe die Terminal-Klasse herunter und platzieren Sie diese unbedingt im Paket `edu.kit.informatik`. Die Methode `Terminal.readLine()` liest eine Benutzereingabe von der Konsole und ersetzt `System.in`. Die Methode `Terminal.printLine()` schreibt eine Ausgabe auf die Konsole und ersetzt `System.out`. Verwenden Sie für jegliche Konsoleneingabe oder Konsolenausgabe die Terminal-Klasse. Verwenden Sie in keinem Fall `System.in` oder `System.out`. Fehlermeldungen werden ausschließlich über die Terminal-Klasse ausgegeben und müssen aus technischen Gründen unbedingt mit `Error,␣` beginnen. Dies kann unter anderem bequem über die Terminal.printError-Methode erfolgen. Laden Sie die Terminal-Klasse **niemals** zusammen mit Ihrer Abgabe hoch.

Modelleisenbahnsimulation (20 Punkte)

In dieser Abschlussaufgabe sollen Sie einen Modelleisenbahnsimulator entwickeln. Die Abschlussaufgabe besteht aus zwei voneinander abhängenden Teilaufgaben. Die erste Teilaufgabe ist die Modellierung des Streckennetzes und des Rollmaterials (Wagen, Lokomotiven, Triebzüge). Die zweite Teilaufgabe ist die Fahrsimulation.

Information

Die grafische Repräsentation von dem Rollmaterial basiert teilweise auf dem `sl`-Programm [a]. Für die Dateien gilt das Copyright von Toyoda Masashi. Zur einfacheren Verwendung stellen wir zusätzlich die Textdatei mit den einzelnen Elementen zur Verfügung (representation.txt).

[a] `https://github.com/mtoyoda/sl`

Gleismaterial

Das Gleisnetz wird über Gleise dargestellt. Vereinfacht werden Gleise durch zwei bzw. drei Punkte im zweidimensionalen Raum dargestellt. Ein Punkt ist ein Tupel aus zwei 32-Bit-Ganzzahlen und repräsentiert eine Koordinate. Alle Gleise besitzen einen eindeutigen Identifikator beginnend bei 1. Dieser Identifikator ist nur eindeutig für Gleise (siehe Beispielablauf). Der Identifikator ist eine 32-bit-Ganzzahl und wird aufsteigend vergeben.

Eine Strecke besteht aus mehreren Gleisen. Dabei gibt es zwei Gleistypen: *normale Gleise* und *Gleisweichen* (Weichen). Normale Gleise werden durch zwei Punkte im Raum beschrieben, den Start- und Endpunkt. Die Koordinaten können beliebig im Raum liegen, mit der Einschränkung, dass Gleisteile immer waagerecht oder senkrecht sind. Dabei ist nur eine Veränderung in x-Richtung waagerecht und eine Veränderung nur in y-Richtung senkrecht. Außerdem muss mit Ausnahme vom ersten Gleis ein Start- oder Endpunk immer an einem Start- bzw. Endpunkt eines vorhanden Gleis anliegen. An einem Punkt an einem Gleis kann immer nur ein anderes Gleis (normales Gleis oder Gleisweiche) angeschlossen werden. Jedes Gleis besitzt eine Länge[1], welche Abhängig von ihrer Start- und Endkoordinate ist. Zur Vereinfachung wird die Länge immer als abgerundete positive Ganzzahl[2] angegeben.

Gleisweichen dienen dazu eine Strecke in zwei Teilstrecken aufzuspalten. Sie werden durch drei Koordinaten beschrieben, einen Startpunkt und zwei Endpunkten. Gleisweichen besitzen zusätzlich die Eigenschaft, die aktuelle Gleisweichenstellung zu speichern. Die aktuelle Gleisweichenstellung entscheidet die Befahrbarkeit einer Gleisweiche. Die Länge einer Gleisweiche ergibt sich in Abhängigkeit ihrer aktuellen Gleisweichenstellung.

Abbildung 0.1 illustriert ein mögliches Streckennetz. Die schwarzen Punkte zeigen jeweilige Start- und Endpunkte für Gleismaterial. Gleiseweichen sind durch das Label „Weiche X“ markiert, wobei X ein Platzhalter für eine ganze positive Zahl ist. Die aktuelle Gleiseweichenstellung wird durch die schwarze Linie angezeigt. Die rote Linie zeigt die alternative Auswahlmöglichkeit an. Ein Zug

[1] Hinweis: Betrag eines Vektors

[2] Wählen Sie hier einen passenden Datentyp, achten Sie dabei insbesondere auf die Abhängigkeit zu Koordinaten

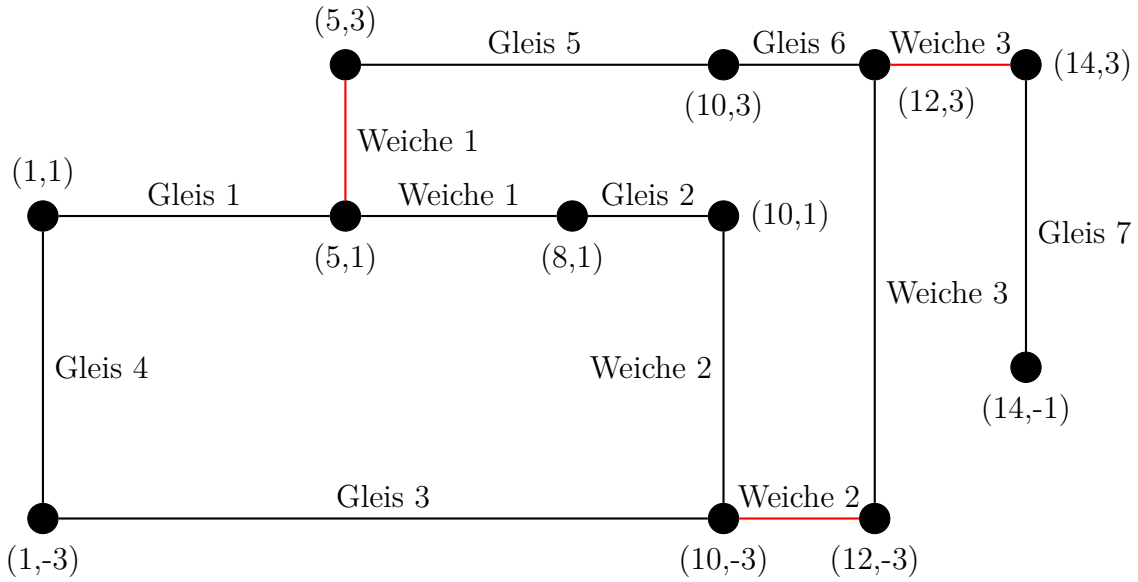


Abbildung 0.1: Beispiel einer einfachen Strecke mit Gleisweichen und Gleisen

kann nur auf der schwarzen Linie die Strecke befahren. Dies ist auch das Streckennetz, welches im Beispielablauf verwendet wird.

Gleise können sich nicht kreuzen, d.h. wenn Gleise als Geraden gesehen werden, dürfen sie sich nicht schneiden. Abbildung 0.2 zeigt eine verbotene Gleiskreuzung. Auch ist es nicht erlaubt Gleise außerhalb von Start- und Endpunkten zu verbinden. Dies wäre in dieser Aufgabe auch ein Kreuzen von Gleisen. Zudem kann einem Start- oder Endpunkt nur genau ein anderes Gleis anliegen.

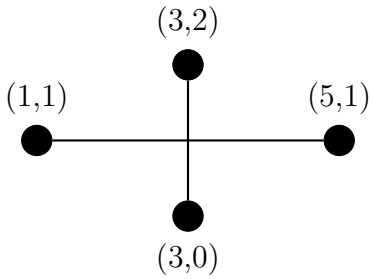
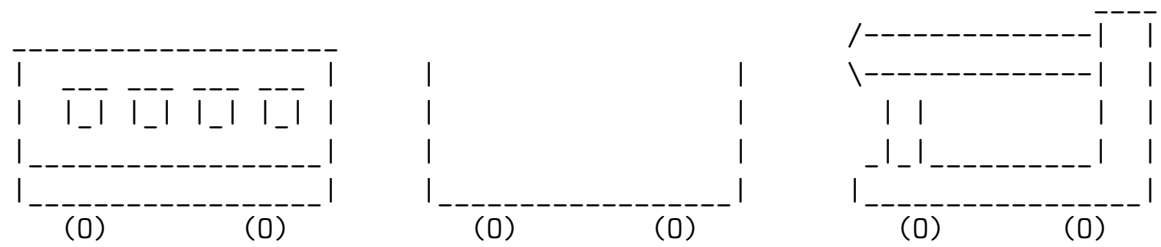


Abbildung 0.2: Nicht erlaubtes Kreuzen von Gleisen

Rollmaterial

Das Rollmaterial beschreibt im Allgemeinen alle auf Schienen fahrenden Gegenstände, z.B. einen Eisenbahnwagen. Aus Gründen der Einfachheit nehmen wir an, dass es nur folgende Typen gibt: Wagen, Lokomotiven, Triebzüge. Allen Typen gemeinsam ist, dass sie ~~einen Namen und~~ eine Länge besitzen. Lokomotiven und Triebzüge besitzen zusätzlich noch einen Namen, z.B. könnte eine Lokomotive den Namen Emma und die Länge 1 haben.



(a) Personenwagen (b) Güterwagen (c) Spezialwagen

Abbildung 0.3: Grafische Repräsentation von Wagen

Wagen

Zur Vereinfachung gibt es hier nur die Personen-, Güter-, Spezialwagen als Wagentyp. Der Personenwagen dient zum Transport von Personen, der Güterwagen zum Transport von Gütern und Spezialwagen sind z.B. ein Kranwagen oder ein Feuerwehrwagen. Die verschiedenen Wagentypen werden wie in Abbildung 0.3 grafisch repräsentiert.

Alle Wagen besitzen einen eindeutigen 32-bit-ganzzahligen aufsteigenden Identifikator beginnend bei 1. Dieser Identifikator ist nur eindeutig für Wagen (siehe Beispielablauf).

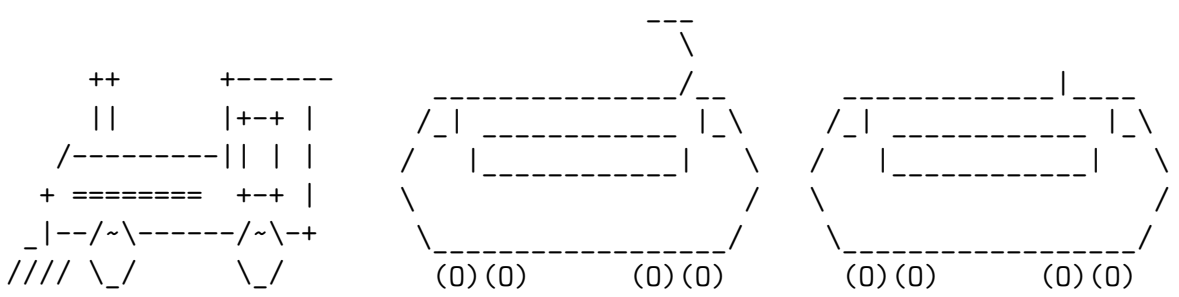
Jeder Wagen besitzt mindestens eine, aber maximal zwei Kupplungen. Diese befinden sich immer am Anfang und Ende eines Wagens. Auch darf es auf einer Seite (Anfang, Ende) maximal eine Kupplung geben. Diese Kupplungen ermöglichen die Komposition von Rollmaterial (siehe Komposition von Rollmaterial)

Lokomotive

Es gibt drei verschiedene Untertypen von Lokomotiven: Elektrolokomotiven, Diesellokomotiven und Dampflokomotiven. Alle Lokomotiven besitzen eine eindeutige ID. Diese ID setzt sich aus der Baureihe und dem Namen der Lokomotive zusammen. Diese werden durch einen Bindestrich getrennt, beispielsweise ergeben die Baureihe „103“ und der Name „118“ die ID „103-118“. Sowohl der Name als auch die Baureihe können Buchstaben und Zahlen enthalten. Wobei die Baureihe nicht die Zeichenkette „W“ sein darf. Jede Lokomotive besitzt mindestens eine, aber maximal zwei Kupplungen. Diese befinden sich immer am Anfang und Ende einer Lokomotive. Auch darf es auf einer Seite (Anfang, Ende) maximal eine Kupplung geben. Diese Kupplungen ermöglichen die Komposition von Rollmaterial (siehe Komposition von Rollmaterial). Abbildung 0.4 zeigt die grafische Repräsentation von Lokomotiven.

Triebzug

Alle Triebzüge besitzen eine Baureihe. Die Triebzug-ID setzt sich nach den gleichen Regeln, wie bei den Lokomotiven zusammen. Aus diesem Grund teilen sich Lokomotiven und Triebzüge den gleichen ID-Raum. Ein Triebzug besitzt eine spezielle Art von Kupplung und kann damit nur mit der gleichen Baureihe von Triebzügen komponiert werden (siehe Komposition von Rollmaterial). Ansonsten



(a) Dampflokomotive (b) Elektrolokomotive (c) Diesellokomotive

Abbildung 0.4: Grafische Repräsentation von Lokomotiven

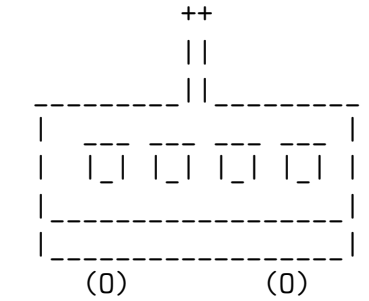


Abbildung 0.5: Grafische Repräsentation von Triebzügen

gelten die gleichen Regeln wie bei Lokomotiven. Abbildung 0.5 zeigt die grafische Repräsentation von Triebzügen.

Komposition von Rollmaterial

Rollmaterial kann zu einem Zug komponiert werden. Bei der Komposition gilt es folgende Einschränkungen zu beachten:

- Am Anfang oder Ende eines validen Zug muss immer mindestens eine Lokomotive/Triebzug sein.
- Bei der Komposition muss immer beachtet werden, ob das Rollmaterial an der gewünschten Kompositionsstelle eine passende Kupplung besitzt.
- Das Rollmaterial, welches komponiert wird, ist noch nicht in einem anderen Zug eingesetzt.

Alle Züge besitzen eine eindeutige ID beginnend bei 1. Die ID ist eine 32-Bit-Ganzzahl und wird aufsteigend vergeben. Diese ID ist nur eindeutig für Züge (siehe Beispielablauf). Ein Zug kann auch grafisch auf der Konsole ausgegeben werden. Dabei werden die einzelnen Repräsentationen des Rollmaterial entsprechen ihrer Position im Zug konkateniert. Die einzelnen Repräsentation sind durch ein Leerzeichen getrennt. Abbildung 0.6 zeigt eine Ausgabe für einen Zug mit einer Dampflokomotive und zwei Personenwagen. Abbildung 0.7 zeigt ein weiteres Zug-Beispiel mit einer Elektrolokomotive und einem Personenwagen.

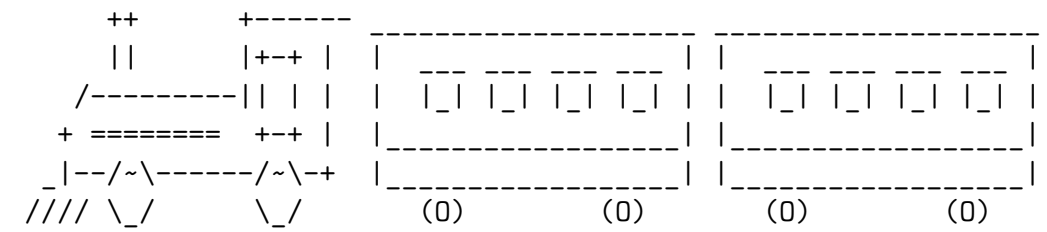


Abbildung 0.6: Beispielrepräsentation von einem Zug mit Dampflokomotive

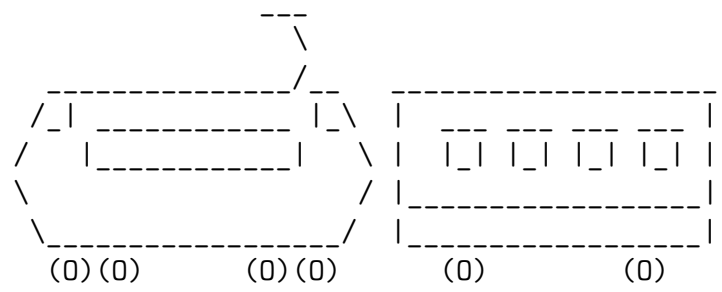


Abbildung 0.7: Beispielrepräsentation von einem Zug mit Elektrolokomotive

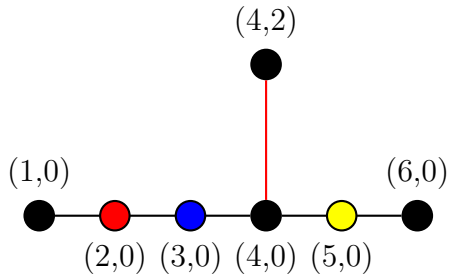
Fahrbetrieb

Um den Fahrbetrieb einer Modelleisenbahn zu simulieren, muss als erstes ein Zug auf ein Gleis gesetzt werden. Dabei wird immer der ganze Zug komplett auf das Gleis gesetzt. Beim Aufsetzen muss ein Richtungsvektor angegeben werden. Dieser gibt die initiale Fahrrichtung des Zuges an. Der Richtungsvektor entspricht dem Richtungsvektor des Gleises oder ist genau der entgegensetze Vektor. Beim Aufgleisen muss beachtet werden, dass die benutzten Gleise noch frei sind, d.h. kein anderer Zug dort steht und es dort überhaupt ein Gleis gibt. Dabei spielt natürlich auch die Länge eines Zuges eine Rolle. Züge können auf Strecken mit Gleiseweichen erst aufgesetzt werden, wenn die Gleiseweichenstellung gesetzt wurde. Außerdem können Züge nur auf den Abschnitt gesetzt werden, der die aktuelle Gleisweichenstellung ist. Falls Gleiseweichen auf denen ein Zug bereits steht, geschaltet werden, entgleist der daraufstehende Zug.

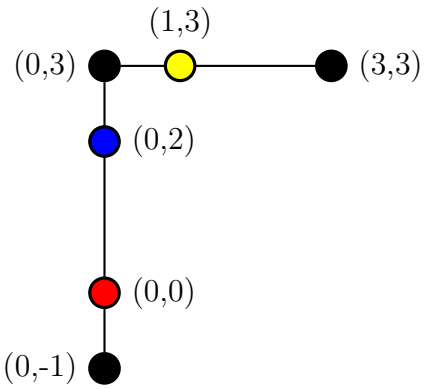
Wenn Züge auf dem Gleis stehen können sie sich immer nur auf dem Gleis fortbewegen. Dabei bewegen sie sich initial in Richtung des Richtungsvektors fort und folgen dem Schienenverlauf. Bevor sich ein Zug bewegen kann, muss für alle Gleisweichen eine Gleisweichenstellung gesetzt werden.

Wie auch bei kleinen Modelleisenbahnanlagen üblich fahren hier alle Züge, die auf der Anlage stehen. Die Geschwindigkeit ist für alle Züge gleich. Im Allgemeinen gilt, dass der Zug bei waagerechten Gleisen sich um n x-Einheiten bewegt und bei vertikalen Gleisen um n y-Einheiten bewegt. Die Simulation hier wird in Schritten berechnet, dazu wird bei jedem Schritt $n \in \mathbb{Z}_{16-Bit}$ übergeben. Zur Verdeutlichung hier ein Beispiel in Abbildung 0.8. Die Punkte sind durch ein Tupel aus Ganzzahlen angegeben. Der Kopf eines Zuges (roter Punkt) steht auf (2,0) auf dem Gleis von (1,0) nach (4,0) und bewegt sich Richtung Punkt (4,0). Bei Punkt (4,0) ist eine Gleisweiche mit den Endpunkten (4,2) und (6,0) angeschlossen. Die aktuelle Gleisweichenstellung ist auf (6,0). Nach einem Schritt mit $n = 1$ würde der Zugkopf (blauer Punkt) bei (3,0) stehen (siehe Abbildung 0.8a). Nach einem weiteren Schritt mit $n = 2$ würde der Zugkopf (gelber Punkt) bei (5,0) stehen. Im zweiten Beispiel (siehe Abbildung 0.8b) steht der Kopf des Zuges (roter Punkt) auf (0,0) auf dem Gleis von (0,3)

nach (0,-1) und bewegt sich in Richtung Punkt (0,3). Nach einem Schritt mit $n = 2$ würde der Zugkopf (blauer Punkt) bei (0,2) stehen. In einem weiteren Schritt mit $n = 2$ würde der Zugkopf (gelber Punkt) bei (1,3) stehen.



(a) Fahrtsimulation mit Weiche



(b) Fahrtsimulation mit Kurve

Abbildung 0.8: Bewegungsrichtung Züge

Während der Simulation kann es zu Zusammenstößen von verschieden Zügen kommen. Ein Zusammenstoß tritt hierbei auf, wenn zwei Züge auf dem gleichen Gleis oder Position stehen. Bei der Überprüfung gehören die Start-/Endpunkte immer zu dem Gleis auf dem der der Rest des Zuges steht. Zum Beispiel Gleis 1 (G1) von (10,1) nach (20,1) und Gleis 2 (G2) von (20,1) nach (30,1). Wenn nun ein Zug mit Länge 4 auf (20,1) steht und Richtung (30,1) fährt, dann wäre G1 belegt und G2 noch frei. Kommt es zu einem Zusammenstoß entgleisen die involvierten Züge.

Züge können über das Gleis hinaus fahren, d.h falls das Gleis endet und es kein Anschlussgleis gibt, z.b. wie bei Abbildung 0.1 (Gleis 7) entgleist der Zug.

Wenn Züge entgleisen, werden Sie vom Gleis genommen. Sie bleiben jedoch in der Zugdatenbank bestehen. Sie können danach wieder aufgesetzt werden.

Interaktive Benutzerschnittstelle

Nach dem Start nimmt Ihr Programm über die Konsole mittels `Terminal.readLine()` verschiedene Arten von Befehlen entgegen, die im Folgenden näher spezifiziert werden. Nach Abarbeitung eines Befehls wartet Ihr Programm auf weitere Befehle, bis das Programm irgendwann durch den `exit`-Befehl beendet wird.

Achten Sie darauf, dass durch Ausführung der folgenden Befehle die zuvor definierten Grundlagen und Bedingungen nicht verletzt werden und geben Sie in diesen Fällen immer eine aussagekräftige Fehlermeldung aus. Entspricht eine Eingabe nicht dem vorgegebenen Format, dann ist immer eine Fehlermeldung auszugeben. Danach soll das Programm auf die nächste Eingabe warten. Bei Fehlermeldungen dürfen Sie den Text frei wählen, er sollte jedoch sinnvoll sein. Jede Fehlermeldung muss aber mit `Error,` beginnen und darf keine Zeilenumbrüche enthalten.

Da wir automatische Tests Ihrer interaktiven Benutzerschnittstelle durchführen, müssen die Ausgaben exakt den Vorgaben entsprechen. Insbesondere sollen sowohl Klein- und Großbuchstaben als

auch die Leerzeichen und Zeilenumbrüche genau übereinstimmen. Geben Sie auch keine zusätzlichen Informationen aus. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären.

Beachten Sie, dass bei der Beschreibung der Eingabe- und Ausgabeformate die Wörter zwischen spitzen Klammen (`<` und `>`) für Platzhalter stehen, welche bei der konkreten Ein- und Ausgabe durch Werte ersetzt werden. Diese eigentlichen Werte enthalten bei der Ein- und Ausgabe keine Klammern. Vergleichen Sie hierzu auch die jeweiligen Beispielabläufe.

Beachten Sie auch, dass bei den folgenden Beispielabläufen die Eingabezeilen mit dem `>`-Zeichen gefolgt von einem Leerzeichen eingeleitet werden. Diese beiden Zeichen sind ausdrücklich kein Bestandteil des eingegebenen Befehls, sondern dienen nur der Unterscheidung zwischen Ein- und Ausgabezeilen.

Wenn Listen ausgegeben werden sollen, werden diese immer Zeilenweise ausgegeben, es sei den ein Befehl schreibt dies explizit anders fest. Für eindeutige Identifikatoren und IDs, welche aus genau einer 32-Bit-Ganzzahl bestehen gilt, dass sie immer aufsteigend beginnend bei 1 vergeben werden. D.h. die erste ID wäre 1 und die zweite 2. Außerdem gilt, dass immer die nächste freie ID gewählt wird. Als Beispiel, wenn z.B. die IDs 1,3,4,5 vergeben sind, wird als nächste ID 2 gewählt.

Hinweis: Kreuzen von Gleisen
Sie können davon ausgehen, dass der Benutzer keine Invalide-Eingaben bezüglich dem Kreuzen von Gleisen tätig, d.h. Sie müssen nicht auf kreuzende Gleise prüfen.

Punkteingabe

Ein Punkt ist ein Tupel aus zwei 32-Bit-Ganzzahlen und beschreibt eine Koordinate in einem 2D-Raum. Das Tupel ist von „runden“ Klammern umschlossen. Die beiden Ganzzahlen werden durch ein Komma getrennt. Die erste 32-Bit-Ganzzahl gibt die x-Koordinate ein und die zweite 32-Bit-Ganzzahl die y-Koordinate.

Eingabeformat

```
(<x-Koordinate>,<y-Koordinate>)
```

Der `add track`-Befehl

Fügt ein normales Gleis hinzu. Dabei wird ein Startpunkt `<startpoint>` und ein Endpunkt `<endpoint>` gewählt. Wenn das Gleis nicht das erste Gleis ist muss immer einer der beiden Punkte auf einem Start- oder Endpunkt eines anderen Gleis liegen. Gleise können nur waagerecht oder senkrecht sein.

Eingabeformat

```
add track <startpoint> -> <endpoint>
```

Ausgabeformat

Im Erfolgsfall wird die eindeutige ID des Gleises ausgegeben. Im Fehlerfall (z.B., kein Anschluss an ein bisheriges Gleis) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `add switch`-Befehl

Fügt eine Gleisweiche hinzu. Dabei werden ein Startpunkt und zwei Endpunkte gewählt. Die einzelnen Gleiseweichenstrecken müssen immer waagerecht oder senkrecht sein. Wenn das Gleis nicht das erste Gleis ist, muss immer einer der drei Punkte auf einem Start- oder Endpunkt eines anderen Gleise liegen.

Eingabeformat

```
add switch <startpoint> -> <endpoint1>,<endpoint2>
```

Ausgabeformat

Im Erfolgsfall wird die eindeutige ID des Gleises ausgegeben. Im Fehlerfall (z.B., kein Anschluss an ein bisheriges Gleis) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `delete track`-Befehl

Löscht ein beliebiges Gleis (normales Gleis, Gleisweiche). Das zu löschende Gleis wird über die `<trackID>` ausgewählt. Beim Löschen muss geprüft werden, dass nach dem Entfernen keine zwei unabhängigen Strecken existieren, d.h. dass alle Gleise verbunden sind und es keine Gleisstücke gibt, die nicht verbunden sind[3]. Außerdem darf kein Zug auf dem Gleis stehen.

Eingabeformat

```
delete track <trackID>
```

Ausgabeformat

Im Erfolgsfall wird `OK` ausgegeben. Im Fehlerfall (z.B., `<trackID>` nicht gefunden) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

[3] Hinweis: Lösbar z.B. mit Tiefensuche über Start- und Endknoten

Der `list tracks`-Befehl

Gibt zeilenweise eine Liste mit allen Gleisen (Gleiseweichen und normale Gleise) aus. Die Ausgabe beinhaltet den Typ, die eindeutige Gleis-ID `<trackID>`, den Startpunkt, den Endpunkt und die Länge. Bei Gleisweichen wird die Länge nur ausgegeben, falls die Gleiseweichenstellung schon gesetzt wurde. In diesem Fall ist die Länge, dann die Länge des Vektors vom Startpunkt zum passenden Endpunkt der aktuellen Gleiseweichenstellung. Die Liste ist nach der Gleis-ID aufsteigend sortiert.

Eingabeformat

`list tracks`

Ausgabeformat

Für normale Gleise: `t <trackID> <startpoint> -> <endpoint> <length>`
Für Gleisweichen: `s <trackID> <startpoint> -> <endpoint1>,<endpoint2> <length>`
`s` oder `t` beschreiben den Gleistyp. `<trackID>` ist eine 32-Bit-ganzzahlige ID. Falls noch kein Gleis existiert wird `No track exists` ausgegeben. Im Fehlerfall (z.B., zusätzliche Parameter) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Beispiel: `list tracks`-Befehl

```
> list tracks
t 1 (1,1) -> (5,1) 54
s 2 (5,1) -> (8,1),(5,3)
```

Der `set switch`-Befehl

Wählt für eine Gleisweiche mit der `<trackID>` die Gleisweichenstellung die den Startpunkt der Gleiseweiche mit dem gewählten Endpunkt (`<point>`) verbindet.

Eingabeformat

`set switch <trackID> position <point>`

Ausgabeformat

Im Erfolgsfall wird `OK` ausgegeben. Im Fehlerfall (z.B., Punkt nicht vorhanden oder Gleis-ID nicht gefunden) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `create engine`-Befehl

Erstellt eine Lokomotive. `<engineType>` beschreibt den Typ der Lokomotive (`electrical` , `steam` , `diesel`). Die Baureihe wird über den Platzhalter `<class>` gesetzt. `<name>` ist der Name der Lokomotive. `<length>` gibt die Länge einer Lokomotive als 32-Bit-Ganzzahl an. `<couplingFront>` und `<couplingBack>` sind Wahrheitswerte, die angeben ob eine Kupplung vorne und oder hinten vorhanden ist. Als Wahrheitswert kann `true` oder `false` gesetzt werden. Dabei gibt `true` an, dass eine Kupplung vorhanden ist. Eine Lokomotive kann nur hinzugefügt werden, falls noch keine Lokomotive oder Triebzug mit der gleichen ID existiert.

Eingabeformat

```
create engine <engineType> <class> <name> <length> <couplingFront> <couplingBack>
```

Ausgabeformat

Im Erfolgsfall wird die eindeutige ID der Lokomotive ausgegeben. Im Fehlerfall (z.B., falsches Eingabeformat oder ID schon vorhanden) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `list engines`-Befehl

Gibt zeilenweise eine Liste mit allen Lokomotiven aus. Die Liste ist lexikografisch nach der der Lok-ID sortiert.

Eingabeformat

```
list engines
```

Ausgabeformat

`<trainID> <engineType> <class> <name> <length> <couplingFront> <couplingBack>`
`<trainID>` ist die ID des Zugs zu dem die Lokomotive gehört. Falls noch keine gesetzt ist wird `none` ausgegeben. `<engineType>` identifiziert den Type der Lokomotive. Dabei steht `e` für elektrisch `s` für Dampf und `d` für Diesel. Alle anderen Platzhalter haben die selbe Bedeutung wie beim `create engine` -Befehl. Falls noch keine Lokomotive existiert wird `No engine exists` ausgegeben.

Beispiel: `list engines`-Befehl

```
> list engines
none s T3 Emma 1 false true
1 e 103 118 1 true true
```

Der `create coach`-Befehl

Erstellt einen Wagen. `<coachType>` beschreibt den Typ des Wagens. Es gibt folgende Typen: `passenger`, `freight`, `special`. Die drei letzten Parameter sind analog zum `create engine`-Befehl.

Eingabeformat

```
create coach <coachType> <length> <couplingFront> <couplingBack>
```

Ausgabeformat

Im Erfolgsfall wird die eindeutige Wagen-ID ausgegeben. Im Fehlerfall (z.B., falsches Eingabeformat) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `list coaches`-Befehl

Gibt zeilenweise eine Liste mit allen Wagen aus. Die List ist aufsteigend nach der Wagen-ID sortiert.

Eingabeformat

```
list coaches
```

Ausgabeformat

```
<coachID> <trainID> <coachType> <length> <couplingFront> <couplingBack>
```

`<coachID>` ist die ID des Wagens. `<trainID>` ist die ID des Zuges zu dem der Wagen gehört. Falls noch keine gesetzt ist wird `none` ausgegeben. `<coachType>` identifiziert den Type des Wagen. Dabei steht `p` für Personenwagen `f` für Güterwagen und `s` für Spezialwagen. Alle anderen Platzhalter haben die selbe Bedeutung wie beim `list engine`-Befehl. Falls noch kein Wagen existiert wird `No coach exists` ausgegeben.

Beispiel: `list coaches`-Befehl

```
> list coaches
1 none ep 1 true true
2 none p 1 true false
```

Der `create train-set`-Befehl

Erstellt einen Triebzug. Die weiteren Parameter und Fehlerfälle sind beim `create engine`-Befehl beschrieben

Eingabeformat

`create train-set <class> <name> <length> <couplingFront> <couplingBack>`

Ausgabeformat

Im Erfolgsfall wird ~~OK~~ die Triebzug-ID ausgegeben. Im Fehlerfall (z.B., falsches Eingabeformat) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `list train-sets`-Befehl

Gibt zeilenweise eine Liste mit allen Triebzügen aus. Die Liste ist lexikografisch nach der Triebzug ID geordnet.

Eingabeformat

`list train-sets`

Ausgabeformat

`<trainID> <class> <name> <length> <couplingFront> <couplingBack>`
Alle Platzhalter haben die selbe Bedeutung wie beim `list engines`-Befehl. Falls noch kein Triebzug existiert wird `No train-set exists` ausgegeben.

Der `delete rolling stock`-Befehl

Löscht ein Rollmaterial. Das Rollmaterial wird anhand seiner eindeutigen ID identifiziert. Für einen Wagen wird der ID ein `W` vorgestellt. Rollmaterial kann nur gelöscht werden, falls es in keinem Zug verwendet wird.

Eingabeformat

```
delete rolling stock <id>
```

Ausgabeformat

Im Erfolgsfall wird `OK` ausgegeben. Im Fehlerfall (z.B., falsches Eingabeformat) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `add train`-Befehl

Fügt dem Zug mit der 32-bit-ganzzahligen `<trainID>` Rollmaterial hinzu. Falls der Zug mit der ID noch nicht existiert wird er automatisch angelegt. Die ID muss der nächsten freien ID entsprechen. Die `<rollingStockID>` ist entweder die ID eines Triebzugs, Lokomotive oder eines Wagens. Bei Wagens wird ein `W` der ID vorausgestellt. Der Zug muss valide bezüglich der Kupplung sein, aber nicht valide bezüglich der Lokomotiven. Das Rollmaterial wird der Reihe nach hinzugefügt, d.h. das erste einem Zug hinzugefügt Rollmaterial ist der Kopf des Zuges (vorne) und das letzte hinzugefügte Rollmaterial stellt den Schluss des Zuges da. Steht der Zug bereits auf einem Gleis kann kein Rollmaterial hinzugefügt werden.

Eingabeformat

```
add train <trainID> <rollingStockID>
```

Ausgabeformat

Im Erfolgsfall wird `<type> <stockID> added to <trainID>` ausgegeben. `<type>` ist der Typ. Folgende Werte können angenommen werden `electrical engine`, `steam engine`, `diesel engine`, `train-set`, `passenger coach`, `freight coach`, `special coach`. `<stockID>` ist die ID des Rollmaterials, wobei bei Wagons ein W davorgestellt wird. `<trainID>` ist die ID des Zuges. Im Fehlerfall (z.B., falsche Kupplung) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `delete train`-Befehl

Löscht den Zug mit der `<ID>`. Falls der Zug sich auf der Strecke befindet, wird er auch von der Strecke entfernt.

Eingabeformat

```
delete train <ID>
```

Ausgabeformat

Im Erfolgsfall wird `OK` ausgegeben. Im Fehlerfall (z.B., falsches Eingabeformat) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `list trains`-Befehl

Gibt zeilenweise eine Liste mit allen Zügen aus. Die Liste ist aufsteigend nach der Zug-ID sortiert.

Eingabeformat

```
list trains
```

Ausgabeformat

`<trainID> <members>`
Die `<trainID>` ist die aktuelle ID des Zuges und Members ist eine Liste der IDs des verwendeten Rollmaterials. Wobei die einzelnen IDs durch Leerzeichen getrennt ausgegeben werden. Der ID eines Wagens wird ein W vorausgestellt um zu kennzeichnen, dass es sich um einen Wagen handelt. z.B. `1 103-118 W1` für den Zug 1 mit der Lokomotive 103-118 und dem Wagen 1.

Der `show train`-Befehl

Der Befehl gibt die grafische Repräsentation des Zuges aus. `trainID` ist dabei die ID des Zuges.

Eingabeformat

```
show train <trainID>
```

Ausgabeformat

Im Erfolgsfall wird die grafische Repräsentation für jedes Rollmaterial ausgegeben. Im Fehlerfall (z.B., Zug nicht vorhanden) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben. (siehe Beispielablauf)

Der `put train`-Befehl

Stellt einen validen (siehe Komposition von Rollmaterial) Zug auf ein Gleis an der Position `<point>`. Die initiale Bewegungsrichtung wird über die Ganzzahlen[4] `<x>,<y>` bestimmt, die als Richtungsvektoren dienen. Beim Aufgleisen ist zu beachten, dass der Zug valide ist und das noch kein anderer Zug auf den benötigten Gleisen steht (Länge des Zuges beachten) und das der Richtungsvektor zu der Richtung des Gleises beim Aufgleispunkt entspricht (oder dem entgegengesetzten).

Eingabeformat

```
put train <trainID> at <point> in direction <x>,<y>
```

Ausgabeformat

Im Erfolgsfall wird `OK` ausgegeben. Im Fehlerfall (z.B., falsches Eingabeformat, oder invalider Zug) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `step`-Befehl

Lässt alle Züge um `<speed>`-Einheiten fahren. `<speed>` ist eine 16-Bit-Ganzzahl. Anschließend wird zeilenweise die Position von allen Zugköpfen (`<headPoint>`), die auf einem Gleis stehen ausgegeben. Dies erfolgt aufsteigend sortiert nach der Zug-ID. Falls es während des Fahrbetriebs zu einem Zusammenstoß kam werden die involvierten Züge ausgegeben. Für die Beschreibung der Fahrt und des Zusammenstoß-Verhalten siehe Fahrbetrieb. Züge die nicht von dem Zusammenstoß betroffen sind fahren normal weiter.

Eingabeformat

```
step <speed>
```

Ausgabeformat

`Train <trainID> at <headPoint>`
Zeilenweise Ausgabe von Zügen. Falls noch kein Zug auf der Strecke steht, wird `OK` ausgegeben. Im Fall eines Zusammenstoßes/Entgleisens wird `Crash of train <trains>` ausgegeben. Wobei `<trains>` eine Liste der Zug-IDs ist, die mit Kommata getrennt sind (aufsteigend sortiert). Falls es zu mehreren Zusammenstoßen/Entgleisen kommt, wird dies zeilenweise ausgegeben. Dabei ist zu beachten, dass die Züge sich nicht überschneiden dürfen, d.h. wenn Zug 1 einen Zusammenstoß mit Zug 3 hat und Zug 3 einen mit Zug 4 würde dies als ein Zusammenstoß gelten und die Ausgabe würde `Crash of train 1,3,4` lauten. Falls es mehrere unabhängige Zusammenstöße gibt, wird

[4] Wählen Sie hier einen passenden Datentyp, achten Sie dabei insbesondere auf die Abhängigkeit zu Koordinaten

die Ausgabe sortiert nach der kleinsten Zug-ID von jedem Zusammenstoß ausgeben. Das gleiche gilt falls es zu einer Mischung von Zusammenstößen und normaler Ausgabe kommt. Zum Beispiel Zug 11 und Zug 12 haben einen Unfall und Zug 9 und Zug 22 auch. Jedoch Zug 10 und 8 nicht, dann würde die Ausgabe ohne Berücksichtigung der Position lauten:

Beispiel: Zusammenstoß von mehreren Zügen

```
> step 2
Train 8 at (0,0)
Crash of train 9,22
Train 10 at (20,3)
Crash of train 11,12
```

Der `exit`-Befehl

Der parameterlose Befehl ermöglicht es, das laufende Programm vollständig zu beenden. Beachten Sie, dass hierfür keine Methoden wie `System.exit()` oder `Runtime.exit()` verwendet werden dürfen.

Eingabeformat

`exit`

Ausgabeformat

Im Erfolgsfall findet keine Ausgabe statt. Im Fehlerfall (z.B. bei einem falsch spezifizierten Eingabeformat) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Beispielablauf

Beispielablauf: Teil 1 von 3

```
> add track (1,1) -> (5,1)
1
> add track (10,10) -> (10,11)
Error, track not connected to other track
> list tracks
t 1 (1,1) -> (5,1) 4
> add switch (5,1) -> (8,1),(5,3)
2
> add track (5,3) -> (8,1)
Error, creation not possible wrong position
> add track (10,1) -> (8,1)
3
> add switch (10,-3) -> (10,1),(12,-3)
4
> add track (10,-3) -> (1,-3)
5
> add track (1,-3) -> (1,1)
6
> add track (5,3) -> (10,3)
7
> add track (10,3) -> (12,3)
8
> add switch (12,3) -> (12,-3),(14,3)
9
> add track (14,-1) -> (14,3)
10
> create engine steam T3 Emma 1 false true
T3-Emma
> list engines
none s T3 Emma 1 false true
> create engine electrical 103 118 3 true true
103-118
```

Beispielablauf: Teil 2 von 3

```
> list engines
none e 103 118 3 true true
none s T3 Emma 1 false true
> delete rolling stock 3
Error, rolling stock with ID 03 not found
> delete rolling stock 103-118
OK
> create coach passenger 1 true true
1
> create coach passenger 1 true true
2
> list coaches
1 none p 1 true true
2 none p 1 true true
> add train 1 W1
passenger coach W1 added to train 1
> list trains
1 W1
> show train 01
____________________
|  ___ ___ ___ ___ |
|  |_| |_| |_| |_| |
|__________________|
|__________________|
   (O)        (O)
> delete train 1
OK
> list trains
No train exists
> add train 1 T3-Emma
steam engine T3-Emma added to train 1
> add train 1 W1
passenger coach W1 added to train 1
> add train 1 W2
passenger coach W2 added to train 1
> list trains
1 T3-Emma W1 W2
```

Beispielablauf: Teil 3 von 3

```
> show train 01
     ++      +------ ____________________ ____________________
     ||      |+-+ |  |  ___ ___ ___ ___ | |  ___ ___ ___ ___ |
   /---------|| | |  |  |_| |_| |_| |_| | |  |_| |_| |_| |_| |
  + ========  +-+ |  |____________________| |____________________|
 _|--/~\------/~\-+  |____________________| |____________________|
//// \_/      \_/       (O)        (O)       (O)        (O)
> list engines
1 s T3 Emma 1 false true
> create train-set 403 145 4 true true
403-145
> add train 2 403-145
train-set 403-145 added to train 2
> set switch 4 position (10,1)
OK
> step 1
Error, position of switches not set
> set switch 2 position (8,1)
OK
> set switch 9 position (12,-3)
OK
> step 1
OK
> put train 1 at (1,1) in direction 1,0
OK
> put train 2 at (10,-2) in direction 0,1
OK
> step 2
Train 1 at (3,1)
Train 2 at (10,0)
> step -2
Train 1 at (1,1)
Train 2 at (10,-2)
> step 2
Train 1 at (3,1)
Train 2 at (10,0)
> step 3
Train 1 at (6,1)
Train 2 at (8,1)
> step 1
Crash of train 1,2
> put train 1 at (1,1) in direction 0,-1
OK
> exit
```