DB vasion

DER BETREIBER

Hinweis zur Bewertung:

- 1/4 der Punkte (25%) wird nach Funktionstests Ihrer Lösung vergeben.
- 3/4 der Punkte (75%) werden entsprechend des in den Teilaufgaben angegebenen Schlüssels auf Basis des Quellcodes vergeben.

Deko-Bild erzeugt mit Adobe Firefly 2, Prompt: „Fast white train moving on rails; clock in the background“. Stile: Kunst, Schichtpapier

DHBW Karlsruhe, Vorlesung Programmieren I+II
Probe-Programmentwurf
Bearbeitungszeit: 120 Minuten

Aufgabe

Die Situation ist sicher bekannt: Sie stehen am Gleis und warten darauf, dass Ihr Zug zu der Zeit einfährt, die vorher vom Beförderungsunternehmen („Der Betreiber“, kurz: DB) im Fahrplan kommuniziert wurde. Es soll jedoch schon vorgekommen sein, dass die erlebte Realität von dieser Information – auch kurzfristig in der App – abwich.

Daher soll ein Experiment durchgeführt werden: Ist es evtl. ressourcenschonender und auch nicht weniger genau, wenn die Fahrten nicht mehr per Monitoring in großen Server-Farmen analysiert und dann angezeigt würden, sondern die Pünktlichkeit an den einzelnen Halten einfach vor Fahrtantritt einmalig ausgewürfelt wird. Studien haben außerdem gezeigt, dass von den Kunden die Anzeige von Gründen bei Verspätungen als positiv wahrgenommen wird, weshalb diese ebenso zufällig zugewiesen werden. Realisieren Sie dies in Form der Java-Anwendung **`DBVasion`**!

Teilaufgabe a) **[17%]**

Das Wichtigste in jedem Fahrplan ist die Erfassung von Zeiten. Schreiben Sie hierfür eine Klasse **`Time`**, welche die aktuelle Zeit als Stunde (`hour`) und Minute (`minute`) speichern kann. Für die spätere Anzeige implementieren Sie die `toString()`-Methode und geben dort die Zeit in der Form `hh:mm` (also bspw. `08:15`) als Zeichenkette zurück.

Der Konstruktor nimmt die Werte für `hour` und `minute` in *exakt dieser Reihenfolge* entgegen (vgl. Verwendung in der bereitgestellten Klasse `DBVasion`) und speichert diese. Zusätzlich ist jedoch eine Plausibilitätsprüfung vorzunehmen: Die Werte für die Stunde dürfen nur im Bereich 0 bis 23, für die Minute im Bereich 0 bis 59 (jeweils inklusive) liegen. Ist dies nicht der Fall ist eine selbst zu schreibende, sprechende **`DBException`** zu werfen, deren Nachricht auf den *gültigen* Wertebereich hinweist!

Um verschiedene Halte auf einer Fahrt zu klassifizieren, soll der *komplexe Aufzählungstyp* **`StopStatus`** umgesetzt werden. Neben der eigentlichen Konstante besetzt der Status eine Farbe für den Text (`textColor`) sowie den Hintergrund (`backgroundColor`), für die spätere Anzeige in einer grafischen Oberfläche.

Folgende Statustypen sollen implementiert werden:

StopStatus	Text-Farbe	Hintergrund-Farbe
`ON_SCHEDULE`	`Color.BLACK`	`new Color(0, 150, 0)`
`REASONABLE`	`Color.BLACK`	`new Color(150, 150, 255)`
`DELAYED`	`Color.WHITE`	`Color.RED`

Um den Status für einen konkreten Halt zu erhalten, realisieren Sie zusätzlich die *statische* Methode

```
public static StopStatus get(int delay)
```

welche bei einer Verspätung von 0 oder weniger Minuten `StopStatus.ON_SCHEDULE`, bei weniger als 6 Minuten `StopStatus.REASONABLE` und `StopStatus.DELAYED` in jedem anderen Fall zurückgibt.

Hinweis: Die Verwendung der Text- und Hintergrundfarbe sowie der statischen `get`-Methode können Sie in der Methode `setDelay` der bereitgestellten Klasse `StopComponent` sehen.

Teilaufgabe b) **[12%]**

Erstellen Sie nun eine Klasse **`Stop`** zur Repräsentation eines Halts während der Fahrt. Ein Halt besteht aus dem Namen des Bahnhofs (`name`), der Ankunftszeit (`arrival` vom Typ `Time`), der Aufenthaltsdauer (`stayPeriod`) sowie der Gleisnummer (`stage`).

Schreiben Sie einen Konstruktor, der die Attribute in *exakt dieser Reihenfolge* (vgl. Verwendung in bereitgestellter Klasse `DBVasion`) entgegennimmt und in den Instanz-Attributen speichert.

Um eine Zugverbindung als Ganze abbilden zu können, implementieren Sie nun die Klasse **`TrainConnection`**. Eine Zugverbindung besteht aus einem Namen (`name`), dem Start- und Endbahnhof (`start`, `end`), einem Wahrheitswert, der aussagt, ob die Verbindung zum Regionalverkehr gehört (`regional`), dem Preis für ein reguläres Ticket (`price`) sowie einer Liste von Halten (`stops`).

Der Konstruktor nimmt die ersten fünf der Attribute in *exakt dieser Reihenfolge* (vgl. Verwendung in bereitgestellter Klasse `DBVasion`) entgegen und speichert diese in den Instanz-Attributen. Die Halte werden nachträglich mit der Set-Methode für die `stops`-Instanz-Variable gesetzt (vgl. Verwendung in bereitgestellter Klasse `DBVasion`).

Teilaufgabe c) **[8%]**

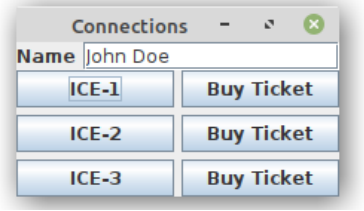
Ändern Sie die `loadConnections`-Methode der Klasse `DBVasion` so ab, dass die Verbindungen aus der bereitgestellten Textdatei `connections.txt` zeilenweise eingelesen werden und *anstatt* der bereitgestellten Testdaten verwendet werden. Für das Parsen der Textzeilen in `TrainConnection`-Objekte nutzen Sie die zur Verfügung gestellte Methode `parseConnection` aus der Klasse `DBVasion`!

Ändern Sie außerdem **zusätzlich** die `loadTrainStops`-Methode der Klasse `DBVasion` so ab, dass die Halte für den übergebenen Verbindungsnamen aus einer der bereitgestellten Text-Dateien geladen wird. Der zu verwendende Dateiname ist in der Variable `filename` vordefiniert (bspw. `ICE-509.txt` für den Zug mit dem Namen `ICE-509`).

Lesen Sie die Datei ebenso zeilenweise ein und verwenden Sie diese *anstatt* der bereitgestellten Testdaten. Für das Parsen der Textzeilen in `Stop`-Objekte nutzen Sie die zur Verfügung gestellte Methode `parseStop` aus der Klasse `DBVasion`!

Sollten beim Lesen Fehler auftreten sind diese abzufangen, müssen aber nicht weiter behandelt werden.

Teilaufgabe d) **[18%]**



Schreiben Sie eine Klasse **`ConnectionSelectionTerm`**, welche eine grafische Nutzeroberfläche zur Auswahl einer verfügbaren Zugverbindung ermöglicht. Der Konstruktor von `ConnectionSelectionTerm` nimmt eine Liste von Zugverbindungen entgegen (vgl. Verwendung in bereitgestellter Klasse `DBVasion`).

Der Titel des Fensters ist auf „Connections“ zu setzen. Im oberen Bereich ist ein Feld für die Eingabe des Namens samt Beschriftung vorzusehen. Für jede dem Konstruktor übergebene Verbindung sollen darunter zeilenweise zwei Buttons erstellt werden. Einen mit dem Verbindungsnamen als Text, einen mit der Aufschrift „Buy Ticket“.

Sollte die Verbindung Teil des Regionalverkehrs sein (vgl. *Teilaufgabe b)*), ist dem ersten der beiden Buttons zusätzlich ein entsprechendes Icon zu setzen (Hinweise beachten, Ansicht siehe Deckblatt).

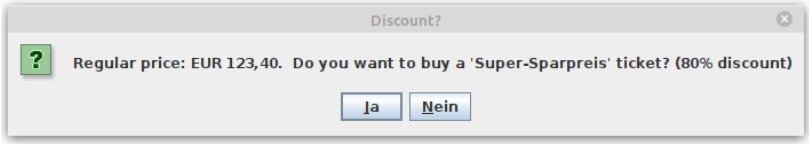
Hinweis: Ein Icon können Sie auf einen `JButton` mittels `button.setIcon(icon)` setzen. Die `ImageIcon`-Instanz wird Ihnen von `DBVasion.createDLTicketIcon` bereitgestellt.

Wird ein Verbindungsbutton ausgewählt ist („normalerweise“, Ausnahme s.u.) für die entsprechende Verbindung ein neues Fenster mit den Verbindungsinformationen (welches Sie in *Teilaufgabe d)* erstellen werden) anzuzeigen.

Aus aktuellem Anlass soll in 70% der Fälle das Fenster jedoch nicht erscheinen und stattdessen eine Meldung angezeigt werden, dass die Verbindung leider entfällt.

Wird ein Button zum Kaufen eines Tickets ausgewählt, soll zunächst geprüft werden, ob ein Name eingegeben wurde. Ist dies nicht der Fall, soll ein Dialog darüber informieren. Um die Kunden für unser Transportmittel zu begeistern, gibt es die Möglichkeit ein Ticket zum Super-Sparpreis mit 80% Rabatt (quasi ein „Lo(c)k-Angebot“!) angeboten.

Fragen Sie in einem Dialog, ob ein solches Ticket erworben werden soll. Neben dem Rabatt-Angebot von 80% muss auch der reguläre Preis der Verbindung angezeigt werden (Hinweise beachten!)



Abhängig von Dialog-Auswahl berechnen Sie den endgültigen Preis für das Ticket und geben diesen in einer Meldung zusammen mit dem Namen der Verbindung, dem Namen der reisenden Person sowie der Ticket-Art (Super-Sparpreis bzw. Regular) aus!

Hinweis: Die Rabatt-Abfrage kann mit `boolean discount = JOptionPane.showConfirmDialog(parent, msg, "Discount?", JOptionPane.YES_NO_OPTION)==JOptionPane.YES_OPTION` realisiert werden..

Hinweis: Sie müssen die Gleitkomma-Werte zur Vereinfachung nicht auf zwei Nachkommastellen runden

Hinweis: Diese Funktionalität wird in Teilaufgabe f) noch erweitert.

Teilaufgabe e) **[9%]**

Erstellen Sie mittels der Klasse **`ConnectionTerm`** nun eine grafische Oberfläche, welche Einzelheiten zu einer Verbindung anzeigt. Diese Verbindung wird dem Konstruktor beim Erzeugen des Objekts übergeben.

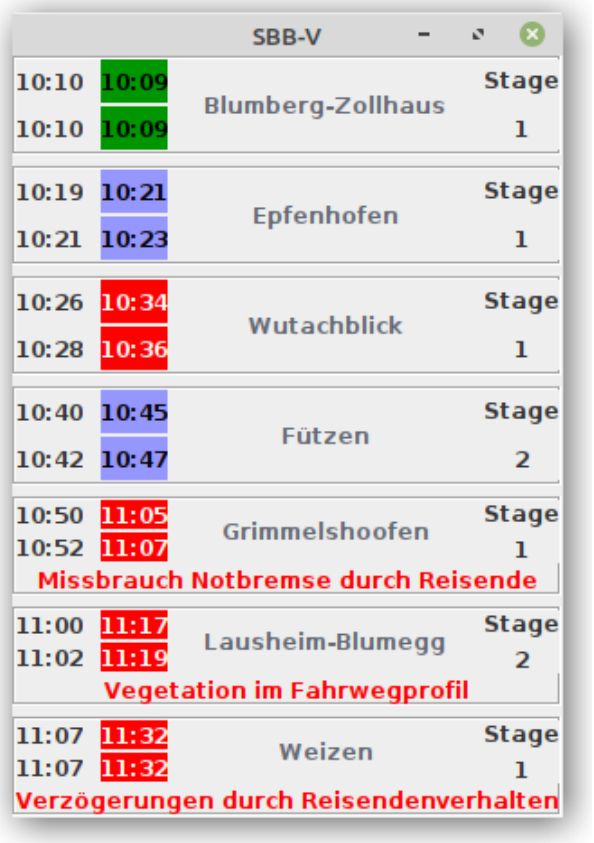
Der Titel des Fensters ist auf den Verbindungsnamen zu setzen. Für jeden Halt ist untereinander die entsprechende Information zu Abfahrtszeiten, Name und Gleis anzuzeigen.

Nutzen Sie hierfür die bereitgestellt Klasse `StopComponent` (siehe USB-Stick), welche die grafische Oberfläche für einen Halt bereits realisiert.

Berechnen Sie ebenfalls die Verspätungen für die Fahrt. Es beginnt immer pünktlich (Verspätung von 0 Minuten) und fügen Sie dann für jeden Halt eine *zufällige* Verspätung zwischen -5 (auch Aufholen ist möglich!) und 10 Minuten (jeweils inklusive) hinzu und übergeben den jeweils aktuellen Wert der Komponente für den aktuellen Halt.

Sollte bei einem Halt eine Verspätung von *insgesamt* mehr als 10 Minuten vorhanden sein, setzen Sie noch eine zufälligen Verspätungsgrund. Der Text kann mit `stopComp.setReason(text)` gesetzt werden, ein Array mit allen verfügbaren Gründen ist unter `StopComponent.DELAY_REASONS` verfügbar.

Hinweis: Die Funktionalität wird in Teilaufgabe g) noch erweitert!



Teilaufgabe f) **[5%]**

Aktualisieren Sie nun Ihre Implementierung für den Kauf von Tickets aus *Teilaufgabe c)*.

Die Nachricht für den Ticket-Kauf, die bisher nur am Ende als Dialog angezeigt wurde soll nun noch in eine Datei mit dem Namen „`tickets.txt`“ geschrieben werden. Sollte die Datei bereits existieren sind weitere Inhalte an das Ende der Datei anzuhängen.

Fehler beim Schreiben der Datei sind abzufangen, müssen aber nicht weiter behandelt werden.

Teilaufgabe g) **[6%]**

Abschließend soll nun eine Fahrt entsprechend simuliert werden. Standardmäßig sind in einer `StopComponent` sämtliche Haltebahnhöfe grau dargestellt, d.h. nicht erreicht worden. Wird die Komponente auf „Stop erreicht“ gesetzt (Aufruf von `stopComponent.setStopReached(true)`), wird die Schrift schwarz.

Starten Sie einen *nebenläufigen* Programmteil, welcher vom ersten bis zum letzten Halt nach und nach den Zustand der entsprechenden Komponente als erreicht setzt. Vor jedem Halt ist dabei eine zufällige Zeit zwischen zwei und fünf Sekunden zu warten, bevor er auf „erreicht“ gesetzt wird. Die Nebenläufigkeit endet, sobald alle Halte abgearbeitet wurden.

Allgemeine Hinweise

Starten
Starten Sie die Anwendung mit der gegebenen Klasse `DBVasion` (siehe USB-Stick).

Schließen eines Fensters
Beim Schließen eines Fensters soll die komplette Anwendung beendet werden.

Sichtbarkeit von Instanz-Attributen
Sämtliche Instanz-Attribute sind als privat zu definieren und von außerhalb der Klasse ggf. mittels Getter- und/oder Setter-Methoden zu verwenden.