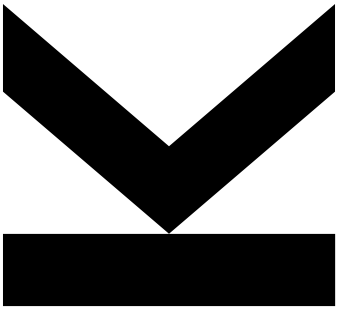
JOHANNES KEPLER
UNIVERSITÄT LINZ

PROJECT DOKUMENTATION



Team 2

Hannes Mayrhofer, Markus Lindner, Sara Kheribi, Milan Keta

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
jku.at
DVR 0093696

Inhalt

Ausführen der Applikation

Nachdem Sie das Maven-Projekt gebaut haben, können Sie die Applikation wie folgt ausführen:

1. Öffnen Sie ein Terminalfenster.
2. Wechseln Sie in das Verzeichnis, in dem sich die generierte JAR-Datei befindet. Dies ist typischerweise das target-Verzeichnis im Projektordner:

c**d Pfad/zum//Projekt/target**

3. Führen Sie die Applikation aus, indem Sie den folgenden Befehl eingeben:

Fahrtenbuch-1.0-SNAPSHOT.jar

Ersetzen Sie Pfad/zu/Ihrem/Projekt mit dem tatsächlichen Pfad zu Ihrem Projekt
und Fahrtenbuch-1.0-SNAPSHOT.jar mit dem Namen der JAR-Datei, die Maven erstellt hat.

Funktionalität der Applikation

1. Verwaltung von Fahrtdaten

- **Erfassung neuer Fahrten:** Man kann über ein spezielles Eingabeformular neue Fahrten hinzufügen. Dieses Formular erfasst relevante Informationen wie KFZ-Kennzeichen, Datum, Uhrzeit, gefahrene Kilometer und Kategorien.

- **Bearbeitung bestehender Fahrten:** Aus der Übersichtstabelle heraus können Benutzer eine Fahrt auswählen und über ein Dialogfenster bearbeiten. Änderungen können an verschiedenen Datenpunkten, wie Datum oder Kilometer, vorgenommen werden.
- **Löschen von Fahrten:** Benutzer haben die Möglichkeit, ausgewählte Fahrten zu löschen.

2. Anzeige und Filterung von Fahrtdaten

- **Übersichtliche Darstellung:** Die Hauptansicht zeigt eine Tabelle aller erfassten Fahrten. Diese Tabelle bietet eine klare und strukturierte Übersicht.
- **Filterfunktionen:** Benutzer können Fahrtdaten nach verschiedenen Kriterien wie Datum, Durchschnittsgeschwindigkeit und Kategorien filtern. Diese Funktionalität verbessert die Benutzerfreundlichkeit, insbesondere bei einer großen Anzahl von Fahrten.

3. Statistische Auswertungen

- **Grafische Darstellungen:** Die Applikation bietet die Möglichkeit, statistische Daten in Form von Balkendiagrammen zu visualisieren. Dies umfasst beispielsweise die Anzeige der gefahrenen Kilometer pro Monat oder Jahr.
- **Erweiterte Statistiken:** Neben den Basisstatistiken können Benutzer auch detaillierte statistische Auswertungen einsehen, die eine tiefere Analyse der Fahrtdaten ermöglichen.

4. Datenexport und -import

- **Speichern von Daten:** Die Applikation bietet eine Exportfunktion, mit der die erfassten Fahrtdaten in einer CSV-Datei gespeichert werden können. Dies ist nützlich für die Datensicherung und weitere Verarbeitung außerhalb der Applikation.
- **Laden von Daten:** Ebenso können Benutzer Daten aus einer CSV-Datei importieren, was den Datenaustausch mit anderen Systemen erleichtert.

4. Benutzeranpassungen und Einstellungen

- **Kategorienverwaltung:** Benutzer können Kategorien hinzufügen, bearbeiten oder entfernen. Dies ermöglicht eine individuelle Anpassung der Kategorisierung von Fahrten.
- **Anpassung von Einstellungen:** Es können Einstellungen wie den Speicherpfad für den Export angepasst werden, was zusätzliche Flexibilität bietet.

Architektur der Applikation

- Das Model besteht aus den Klassen Fahrtenbuch, Fahrt und FahrtStatus die die Kernlogik und die Datenstruktur der Anwendung darstellen. Diese Klassen sind für die Verwaltung der Fahrtdaten verantwortlich, einschließlich Operationen wie Speichern, Laden, Bearbeiten und Löschen von Einträgen sowie die Bereitstellung von statistischen Daten.

- Die View wird durch die Klasse FahrtenbuchUI repräsentiert, die die grafische Benutzeroberfläche (GUI) der Anwendung bildet. Diese Klasse nutzt JavaFX-UI-Komponenten, um die Daten in einer benutzerfreundlichen Form darzustellen. Dazu gehören Tabellenansichten für die Auflistung der Fahrten, Formulare für die Dateneingabe, Diagramme für statistische Darstellungen und verschiedene Steuerelemente wie Buttons und Menüs für die Interaktion mit der Anwendung.
- Der Controller ist in der FahrtenbuchUI-Klasse integriert und handhabt die Benutzerinteraktionen. Er reagiert auf Benutzereingaben, leitet die Anfragen an das Model weiter und aktualisiert die View basierend auf den Rückmeldungen und Datenänderungen. Diese Schicht sorgt dafür, dass die Benutzeroberfläche und das Datenmodell synchronisiert und konsistent bleiben.

Hauptkomponenten der Anwendung:

- **FahrtenbuchUI:** Ist eine zentrale Klasse in der Fahrtenbuch-Applikation, die für die Gestaltung und Interaktivität der grafischen Benutzeroberfläche (GUI) verantwortlich ist. Sie verwendet JavaFX zur Erstellung und Steuerung verschiedener Bildschirmelemente und ermöglicht Benutzerinteraktionen wie das Anzeigen, Erstellen, Bearbeiten und Löschen von Fahrten sowie das Anzeigen statistischer Daten.
- **Fahrtenbuch:** Diese Klasse ist für die Kernlogik und Datenhaltung für die Fahrten zu verwalten. Sie ist verantwortlich für die Speicherung, Filterung und Berechnung von Statistiken bezüglich der Fahrten.
- **Fahrt:** Eine Klasse, die die Datenstruktur für eine einzelne Fahrt darstellt, einschließlich Details wie Datum, Uhrzeit, gefahrene Kilometer und Kategorien.
- **FahrtStatus**: FahrtStatus ist ein enum, das verschiedene Zustände einer Fahrt repräsentiert. Als enum bietet es eine festgelegte Menge von Konstanten, die den Status einer Fahrt klar und konsistent definieren.

Klasse Fahrtenbuch UI:

Kernfunktionalitäten

- Anfangsbildschirm (Startfenster): Initialisiert und zeigt den Startbildschirm der Applikation an.
- Übersichtsbildschirm (Hauptbildschirm): Stellt eine Übersichtstabelle aller Fahrten dar und ermöglicht Benutzerinteraktionen wie das Filtern von Daten und das Öffnen weiterer Fenster für detaillierte Funktionen.
- Neue Fahrt anlegen: Erlaubt dem Benutzer, eine neue Fahrt mit allen relevanten Details anzulegen.
- Fahrt bearbeiten: Ermöglicht das Bearbeiten der Details einer ausgewählten Fahrt.
- Statistische Auswertungen anzeigen: Zeigt verschiedene statistische Ansichten, wie Jahresstatistik und erweiterte Kilometerstatistik.
- Export/Import von Daten: Implementiert Funktionen zum Speichern und Laden der Fahrtdaten in bzw. aus CSV-Dateien.
- Einstellungen verwalten: Bietet einen Einstellungsbildschirm, in dem Benutzer die Kategorien verwalten und Einstellungen wie den Speicherpfad anpassen können.

Wichtige Methoden

public void start(Stage primaryStage)

Diese Methode ist der Einstiegspunkt für die JavaFX-Anwendung. Sie initialisiert und zeigt den Startbildschirm der Applikation an.

- **Ablauf**:
  - Ein **StackPane** als Root-Element und ein **ProgressBar** für den Ladevorgang werden erstellt.
  - Ein Logo wird geladen und auf dem Bildschirm angezeigt.
  - Eine **FadeTransition** wird verwendet, um das Logo sanft einzublenden.
  - Nach Abschluss des Ladevorgangs wird das Hauptfenster (**overview**) aufgerufen.

private void overview(Stage primaryStage)

Erstellt und zeigt den Hauptbildschirm der Applikation, inklusive der Fahrtentabelle und verschiedener Steuerungselemente.

- **Ablauf**:
  - Erstellt eine **TableView**, um die Fahrtendaten anzuzeigen.
  - Initialisiert verschiedene Steuerelemente wie Buttons für das Hinzufügen, Bearbeiten und Löschen von Fahrten, sowie für den Zugriff auf Einstellungen und Statistiken.
  - Richtet Event-Handler für diese Steuerelemente ein, um entsprechende Aktionen auszulösen.
  - Verwendet **BorderPane** und **VBox**/**HBox** für das Layout.

private void neueFahrt(Stage primaryStage)

Öffnet ein Fenster, in dem der Benutzer eine neue Fahrt anlegen kann.

- **Ablauf**:
  - Zeigt ein Formular mit Eingabefeldern für Details wie KFZ-Kennzeichen, Datum, Uhrzeiten, gefahrene Kilometer und Kategorien.
  - Enthält einen 'Speichern'-Button, der die eingegebenen Daten validiert und an das Fahrtenbuch zur Speicherung weiterleitet.
  - Verwendet **TextField**, **DatePicker** und andere JavaFX-Komponenten für die Dateneingabe.

private void bearbeiteFahrt(Fahrt ausgewaehlteFahrt, Stage primaryStage)

Ermöglicht das Bearbeiten einer ausgewählten Fahrt.

- **Ablauf**:
  - Öffnet ein Dialogfenster, das mit den Details der ausgewählten Fahrt vorbefüllt ist.
  - Ermöglicht dem Benutzer, Änderungen vorzunehmen und diese zu speichern.
  - Implementiert eine Funktion zum Löschen der ausgewählten Fahrt.
  - Verwendet **Dialog**-Klasse für die Darstellung und **TextField**, **DatePicker** für die Eingabe.

private void switchToSettings(Stage primaryStage)

Zeigt ein Einstellungsfenster für die Verwaltung von Kategorien und Einstellungen wie dem Speicherpfad.

- **Ablauf**:
  - Stellt ein Benutzerinterface bereit, in dem Kategorien hinzugefügt, bearbeitet oder entfernt werden können.
  - Erlaubt dem Benutzer, den Speicherpfad für die Exportdateien festzulegen.
  - Nutzt **ListView** und **TextField** für die Anzeige und Bearbeitung der Kategorien sowie Eingabefelder für den Pfad.

private void initializeStatistikMenuButton()

- Diese Methode initialisiert einen Menü-Button für statistische Ansichten.
- Sie fügt Menüeinträge hinzu, die es dem Benutzer ermöglichen, verschiedene statistische Ansichten wie Jahresstatistik und erweiterte Statistik aufzurufen.
- Jeder Menüeintrag hat einen Event-Handler, der die entsprechende statistische Ansicht anzeigt.

private void initializeGrafikMenuButton()

- Ähnlich wie **initializeStatistikMenuButton**, aber für grafische Darstellungen der Statistiken.
- Fügt Menüeinträge hinzu für die Anzeige von Kilometern pro Monat und Jahr in Diagrammform.
- Jeder Eintrag löst eine Methode aus, die das entsprechende Diagramm erstellt und anzeigt.

void zeigeKilometerDiagramm()

- Erstellt und zeigt ein Balkendiagramm der gefahrenen Kilometer pro Monat und Kategorie.
- Nutzt die Daten aus **fahrtenbuch**, um die Diagramme zu generieren.
- Verwendet JavaFX-Komponenten wie **BarChart**, **CategoryAxis**, und **NumberAxis** für die Diagrammerstellung.

void zeigeJahresKilometerDiagramm()

- Ähnlich wie **zeigeKilometerDiagramm**, aber zeigt die Gesamtkilometer pro Jahr und Kategorie.
- Stellt die jährliche Fahrleistung in Balkendiagrammform dar.

**private TableView<Map.Entry<Integer, Map<String, Double>>>
erstelleJahresKilometerTableView(Set<String> kategorien)**

- Erstellt eine **TableView** für die Darstellung von Jahreskilometern pro Kategorie.
- Generiert Spalten für jedes Jahr und jede Kategorie basierend auf den bereitgestellten Daten.

void zeigeErweiterteKilometerStatistik()

- Zeigt eine detaillierte Statistik der gefahrenen Kilometer pro Monat und Kategorie in einer **TableView**.
- Nutzt **erstelleErweiterteKilometerTableView** für die Erstellung der Tabelle und aktualisiert sie mit aktuellen Daten.

**private TableView<Map.Entry<YearMonth, Map<String, Double>>>
erstelleErweiterteKilometerTableView(Set<String> kategorien)**

- Erstellt eine **TableView** für die detaillierte Darstellung von Kilometern pro Monat und Kategorie.

- Ähnlich wie **erstelleJahresKilometerTableView**, aber mit einer feineren Aufteilung auf Monatsebene.

private void aktualisiereErweiterteKilometerTabelle(TableView<Map.Entry<YearMonth, Map<String, Double>>> tableView)

- Aktualisiert die Tabelle der erweiterten Kilometerstatistik mit den neuesten Daten aus dem Fahrtenbuch.

void zeigeJahresKilometerStatistik()

- Zeigt eine Statistik der gefahrenen Kilometer pro Jahr und Kategorie in einer **TableView**.
- Überprüft, ob Daten vorhanden sind, und zeigt dann die Statistik in tabellarischer Form an.

void oeffneFilter(TableView fahrtenTabelle)

- Öffnet ein Dialogfenster zur Filterung von Fahrten nach verschiedenen Kriterien.
- Der Benutzer kann Fahrten nach Datum, Durchschnittsgeschwindigkeit und Kategorien filtern.
- Stellt eine dynamische Filterfunktion bereit, die die **TableView** mit den gefilterten Ergebnissen aktualisiert.

Klasse Fahrtenbuch:

1. Konstruktoren

- **Fahrtenbuch(List<String> kategorien, List<Fahrt> fahrten)**
  - **Zweck**: Erstellt ein Fahrtenbuch mit vorgegebenen Kategorien und Fahrten.
  - **Parameter**:
    - **kategorien**: Eine Liste der Kategorien.
    - **fahrten**: Eine Liste der Fahrten.
- **Fahrtenbuch()**
  - **Zweck**: Erstellt ein leeres Fahrtenbuch, nützlich beim ersten Start der Applikation.

2. Methoden zur Verwaltung von Fahrten

- **neueFahrt(...)**
  - **Zweck**: Fügt eine neue Fahrt hinzu.

    - **Parameter**: Detaillierte Informationen über die Fahrt wie KFZ-Kennzeichen, Datum, Zeiten, Kilometer, Fahrtstatus und Kategorien.
    - **Ausnahmen**: **IOException**, falls beim Speichern Probleme auftreten.
- **bearbeiteFahrt(...)**
    - **Zweck**: Ermöglicht die Bearbeitung einer bestehenden Fahrt.
    - **Parameter**: Ähnlich wie bei **neueFahrt**, jedoch mit dem Ziel der Aktualisierung.
- **loescheFahrten(...)**
    - **Zweck**: Löscht eine spezifische Fahrt basierend auf Schlüsselinformationen.
    - **Parameter**: KFZ-Kennzeichen, Datum und Abfahrtszeit.
- **planeZukuenftigeFahrten(...)**
    - **Zweck**: Plant wiederkehrende Fahrten.
    - **Parameter**: Daten für zukünftige Fahrten, KFZ-Kennzeichen, Abfahrtszeit und Kategorien.
    - **Ausnahmen**: **IOException** bei Speicherproblemen.

3. Methoden zur Datenabfrage und -aufbereitung

- **listeFahrten()**
    - **Zweck**: Gibt eine Liste aller erfassten Fahrten zurück.
- **berechneKilometerProMonat()**
    - **Zweck**: Berechnet die insgesamt gefahrenen Kilometer pro Monat.
    - **Rückgabe**: Eine **Map** von **YearMonth** zu **Double** (Kilometern).
- **berechneKilometerProMonatUndKategorie()**
    - **Zweck**: Berechnet die gefahrenen Kilometer pro Monat und Kategorie.
    - **Rückgabe**: Eine verschachtelte **Map** mit **YearMonth** und Kategorienamen zu Kilometern.
- **berechneKilometerProJahrUndKategorie()**
    - **Zweck**: Berechnet die gefahrenen Kilometer pro Jahr und Kategorie.
    - **Rückgabe**: Eine verschachtelte **Map** mit Jahreszahlen und Kategorienamen zu Kilometern.

4. Kategorienverwaltung und Filterung

- **addKategorie(String kategorie)**
  Fügt eine neue Kategorie hinzu.
  Parameter: Der Name der Kategorie.
- **getKategorien() und getKategorien(Boolean x)**
  Gibt eine Liste aller einzigartigen Kategorien zurück, entweder als **Set** oder **ObservableList**.
- **addKategories(Collection kategorienNeu)**
  Fügt eine Sammlung von Kategorien zur Liste hinzu.
- **Filtermethoden** (**filterByDate**, **filterByAvgVUnder**, **filterByAvgVOver**, **filterByKategorie**)
  Ermöglichen es, Fahrten nach verschiedenen Kriterien wie Datum, Durchschnittsgeschwindigkeit und Kategorien zu filtern.

5. Export und Import von Daten

- **exportFahrt()**
  Exportiert die Fahrtendaten in eine CSV-Datei.
  IOException bei Schreibproblemen.
- **importFahrt()**
  Importiert Fahrtendaten aus einer CSV-Datei.