One of the most important parts of our program was create new trip/tour.

In order to create new tour user has to specify dates, times, departure, destination, distance, bus type and possibly extra services and stops. At the end price is specified. Tour is created by clicking on button create tour.

package main.Controller;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXMLLoader;  
import javafx.fxml.Initializable;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.input.KeyEvent;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import main.Main;  
import main.Model.\*;  
  
import java.io.IOException;  
import java.net.URL;  
import java.util.Date;  
import java.util.ResourceBundle;  
  
*/\*\*  
 \* Class that manages trips.  
 \*  
 \** ***@author*** *IT-1Y-A16 Group 1  
 \*/*public class TripController extends Controller implements Initializable {  
  
  
 public TextField fieldStartTime;  
 public TextField fieldEndTime;  
 public ComboBox fieldDestination;  
 public ComboBox fieldDeparture;  
 public TextField fieldDistance;  
 public TextField fieldPrice;  
 public DatePicker startDatePicker;  
 public DatePicker endDatePicker;  
 public CheckBox checkPrivateTrip;  
 public Button CreateTourBtn;  
 public ListView busListview;  
 public ChoiceBox busType;  
 public ComboBox stopName;  
 public Button addStopBtn;  
 public Button removeStopBtn;  
 public ListView stopsList;  
 public TextField stopTimeField;  
 public ListView chauffeurList;  
 public CheckBox foodCheckBox;  
 public CheckBox accommodationCheckBox;  
 public CheckBox ticketCheckBox;  
  
 //Add customer  
 public TextField fieldCustomerName;  
 public TextField fieldCustomerCompany;  
 public TextField fieldCustomerAddress;  
 public TextField fieldCustomerEmail;  
 public TextField fieldCustomerPhone;  
 public ListView customerList;  
 public Button saveCustomerBtn;  
 public Label tourLabel;  
 private Customer customer = null;  
 private TripController mainController;  
  
 //list for stops  
 private DestinationList stops = new DestinationList();  
  
 private Trip oldTrip;  
  
  
 @Override  
 public void initialize(URL location, ResourceBundle resources) {  
  
 if (fieldStartTime != null) {  
 loadBusList();  
 loadChauffeurList();  
  
 ObservableList<Destination> destinationItems = FXCollections.*observableArrayList*();  
 destinationItems.addAll(DataHandler.*getDestinationList*().getArrayDestination());  
 fieldDestination.setItems(destinationItems);  
 fieldDeparture.setItems(destinationItems);  
 stopName.setItems(destinationItems);  
 }  
  
 if (customerList != null) loadCustomerList();  
  
 }  
  
 public void getDataChoice(ActionEvent actionEvent) {  
 loadBusList();  
 loadChauffeurList();  
 }  
  
 public void getDataFromField(KeyEvent keyEvent) {  
 loadBusList();  
 loadChauffeurList();  
 }  
  
 public void callCustomerList(KeyEvent keyEvent) {  
 loadCustomerList();  
 }  
  
 private void loadChauffeurList() {  
 ChauffeurList chauffeurs;  
  
 chauffeurs = DataHandler.*getChauffeurList*().copy();  
 chauffeurs.removeAllVicars();  
  
 if (validateEmptyField(fieldDistance) && validateNumberField(fieldDistance)) {  
 chauffeurs = chauffeurs.getAllByPrefferedDistance(Integer.*parseInt*(fieldDistance.getText()));  
 }  
 chauffeurs = chauffeurs.getAllByPrefferedBus(busType.getValue().toString());  
  
 for (Chauffeur chauffeur : DataHandler.*getChauffeurList*().getAllVicars().getArrayChauffeur()) {  
 chauffeurs.add(chauffeur);  
 }  
  
 if (startDatePicker.getValue() != null && endDatePicker.getValue() != null && validateTimeField(fieldStartTime) && validateTimeField(fieldEndTime)) {  
 String[] lineToken = fieldStartTime.getText().split(":");  
 int hours = Integer.*parseInt*(lineToken[0]);  
 int minutes = Integer.*parseInt*(lineToken[1]);  
 Date dateStart = new Date(startDatePicker.getValue().getYear() - 1900, startDatePicker.getValue().getMonthValue(), startDatePicker.getValue().getDayOfMonth(), hours, minutes);  
 lineToken = fieldEndTime.getText().split(":");  
 hours = Integer.*parseInt*(lineToken[0]);  
 minutes = Integer.*parseInt*(lineToken[1]);  
 Date dateEnd = new Date(endDatePicker.getValue().getYear() - 1900, endDatePicker.getValue().getMonthValue(), endDatePicker.getValue().getDayOfMonth(), hours, minutes);  
 chauffeurs = chauffeurs.getAvailable(dateStart, dateEnd);  
 }  
  
 chauffeurList.getSelectionModel().setSelectionMode(SelectionMode.*SINGLE*);  
 ObservableList<Chauffeur> items = FXCollections.*observableArrayList*();  
 if (chauffeurs.getSize() != 0) {  
 items.addAll(chauffeurs.getArrayChauffeur());  
 }  
 chauffeurList.setItems(items);  
 }  
  
 private void loadBusList() {  
  
 BusList buses;  
  
 if (busType.getValue().equals("Mini Bus"))  
 buses = new BusList(DataHandler.*getBusList*().searchByType("main.Model.MiniBus"));  
 else if (busType.getValue().equals("Party Bus"))  
 buses = new BusList(DataHandler.*getBusList*().searchByType("main.Model.PartyBus"));  
 else if (busType.getValue().equals("Luxury Bus"))  
 buses = new BusList(DataHandler.*getBusList*().searchByType("main.Model.LuxuryBus"));  
 else buses = new BusList(DataHandler.*getBusList*().searchByType("main.Model.ClassicBus"));  
  
 if (startDatePicker.getValue() != null && endDatePicker.getValue() != null && validateTimeField(fieldStartTime) && validateTimeField(fieldEndTime)) {  
 String[] lineToken = fieldStartTime.getText().split(":");  
 int hours = Integer.*parseInt*(lineToken[0]);  
 int minutes = Integer.*parseInt*(lineToken[1]);  
 Date dateStart = new Date(startDatePicker.getValue().getYear() - 1900, startDatePicker.getValue().getMonthValue(), startDatePicker.getValue().getDayOfMonth(), hours, minutes);  
 lineToken = fieldEndTime.getText().split(":");  
 hours = Integer.*parseInt*(lineToken[0]);  
 minutes = Integer.*parseInt*(lineToken[1]);  
 Date dateEnd = new Date(endDatePicker.getValue().getYear() - 1900, endDatePicker.getValue().getMonthValue(), endDatePicker.getValue().getDayOfMonth(), hours, minutes);  
 buses = buses.getAvailable(dateStart, dateEnd);  
 }  
  
 busListview.getSelectionModel().setSelectionMode(SelectionMode.*SINGLE*);  
 ObservableList<Bus> items = FXCollections.*observableArrayList*();  
 items.addAll(buses.getArrayBuses());  
 busListview.setItems(items);  
  
  
 }  
  
 private void loadCustomerList() {  
 CustomerList customers = DataHandler.*getCustomerList*();  
  
 if (validateEmptyField(fieldCustomerName))  
 if (customers.findAllByName(fieldCustomerName.getText()) != null)  
 customers = customers.findAllByName(fieldCustomerName.getText());  
 if (validateEmptyField(fieldCustomerPhone))  
 if (customers.findAllByPhone(fieldCustomerPhone.getText()) != null)  
 customers = customers.findAllByPhone(fieldCustomerPhone.getText());  
 if (validateEmptyField(fieldCustomerCompany))  
 if (customers.findAllByCompanyName(fieldCustomerCompany.getText()) != null)  
 customers = customers.findAllByCompanyName(fieldCustomerCompany.getText());  
  
 ObservableList<Customer> customerItems = FXCollections.*observableArrayList*();  
  
 if (customers.getSize() != 0) {  
 customerItems.addAll(customers.getArrayCustomer());  
 } else {  
 customerItems.addAll(DataHandler.*getCustomerList*().getArrayCustomer());  
 }  
  
 customerList.setItems(customerItems);  
 }  
  
 private void loadStops() {  
 ObservableList<Destination> destinationItems = FXCollections.*observableArrayList*();  
 for (Destination destination : stops.getArrayDestination()) {  
 destinationItems.add(destination);  
 }  
 stopsList.setItems(destinationItems);  
 }  
  
 public void handleStops(ActionEvent actionEvent) {  
 if (actionEvent.getSource() == addStopBtn && validateNumberField(stopTimeField)) {  
 stops.add(new Destination(stopName.getValue().toString(), stopTimeField.getText()));  
 }  
  
 if (actionEvent.getSource() == removeStopBtn && stopsList.getSelectionModel().getSelectedItem() != null) {  
 String[] lineToken = stopsList.getSelectionModel().getSelectedItem().toString().split(", ");  
 String stopNameTemp = lineToken[0].trim();  
 stops.removeDestination(stops.findByName(stopNameTemp));  
 }  
 loadStops();  
 }  
  
 public void openCustomerView(ActionEvent actionEvent) {  
 if (checkPrivateTrip != null) {  
 Stage stage = Main.*stage*;  
 if (checkPrivateTrip.isSelected()) {  
 Stage window = new Stage();  
 Parent root = null;  
 try {  
 FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("../View/addCustomerData.fxml"));  
 root = fxmlLoader.load();  
 TripController CustomerViewController = fxmlLoader.getController();  
 CustomerViewController.addMainController(this);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 window.initModality(Modality.*APPLICATION\_MODAL*);  
 window.setTitle("Add Customer Data");  
 window.setMinWidth(600);  
 window.setMinHeight(400);  
 window.setResizable(false);  
  
 Scene scene = new Scene(root != null ? root : null);  
 window.setScene(scene);  
 window.show();  
 }  
 }  
  
 }  
  
 public void addMainController(TripController tripController) {  
 mainController = tripController;  
 }  
  
 public void addCustomerData(ActionEvent actionEvent) {  
  
 String alert = "There are some mistakes: ";  
 int length = alert.length();  
  
 if (!validateEmptyField(fieldCustomerName)) alert += "Name, ";  
 if (!validateEmptyField(fieldCustomerAddress)) alert += "Address, ";  
 if (!validateEmptyField(fieldCustomerEmail) || !validateEmail(fieldCustomerEmail))  
 alert += "Email, ";  
 if (!validateEmptyField(fieldCustomerPhone) || !validateLength(fieldCustomerPhone, 8)) alert += "Phone, ";  
  
 if (length == alert.length()) {  
 //save it DataHandler. .....  
 boolean isCompany = validateEmptyField(fieldCustomerCompany);  
  
 if (!isCompany) {  
 DataHandler.*getCustomerList*().add(new Customer(fieldCustomerName.getText(), fieldCustomerAddress.getText(), fieldCustomerEmail.getText(), fieldCustomerPhone.getText()));  
 } else if (isCompany) {  
 DataHandler.*getCustomerList*().add(new Customer(fieldCustomerName.getText(), fieldCustomerAddress.getText(), fieldCustomerEmail.getText(), fieldCustomerPhone.getText(), isCompany, fieldCustomerCompany.getText()));  
 }  
 DataHandler.*save*();  
 successdisplay("Success", "Customer was created.");  
 loadCustomerList();  
 } else {  
 //alert  
 alertdisplay("Wrong Input", alert);  
 }  
 }  
  
 public void chooseCustomer(ActionEvent actionEvent) {  
  
 if (customerList.getSelectionModel().getSelectedItem() != null) {  
 mainController.saveCustomerToMain((Customer) customerList.getSelectionModel().getSelectedItem());  
  
 Stage stage = (Stage) saveCustomerBtn.getScene().getWindow();  
 stage.close();  
 } else {  
 alertdisplay("No customer", "Please choose one Customer");  
 }  
 }  
  
 public void saveCustomerToMain(Customer customer) {  
 this.customer = customer;  
 }  
  
 public void setEditData(Trip trip) {  
 menu.setVisible(false);  
 tourLabel.setText("Edit Trip");  
 CreateTourBtn.setText("Edit");  
  
 oldTrip = trip;  
  
 fieldStartTime.setText(trip.getTimeStart());  
 fieldEndTime.setText(trip.getTimeEnd());  
 startDatePicker.setValue(trip.getDateStart());  
 endDatePicker.setValue(trip.getDateEnd());  
 fieldDestination.setValue(trip.getDestination().toString());  
 fieldDeparture.setValue(trip.getPickUpPoint().toString());  
 fieldDistance.setText(Integer.*toString*(trip.getDistance()));  
 fieldPrice.setText(Integer.*toString*(trip.getPrice()));  
 checkPrivateTrip.setSelected(trip.isPrivate());  
 foodCheckBox.setSelected(trip.isFood());  
 accommodationCheckBox.setSelected(trip.isAccommodation());  
 ticketCheckBox.setSelected(trip.isTickets());  
  
 busListview.getSelectionModel().select(trip.getBus());  
 busType.setValue(trip.getBus().getBusType());  
  
 if (trip.getStops() != null) {  
 stops = trip.getStops();  
 loadStops();  
 }  
  
 chauffeurList.getSelectionModel().select(trip.getChauffeur());  
  
 if (trip.getChauffeur() != null) {  
 chauffeurList.getSelectionModel().select(trip.getCustomer());  
 }  
 }  
  
 public void createTour(ActionEvent actionEvent) throws IOException {  
  
  
 String alert = "There are some mistakes: ";  
 int length = alert.length();  
  
 if (!validateEmptyDate(startDatePicker)) alert += "Start Date, ";  
 if (!validateEmptyDate(endDatePicker) || validateAdultDate(endDatePicker)) alert += "End Date, ";  
 if (!validateEmptyField(fieldStartTime) || !validateTimeField(fieldStartTime)) alert += "Start time, ";  
 if (!validateEmptyField(fieldEndTime) || !validateTimeField(fieldEndTime)) alert += "End time, ";  
 if (!validateEmptyField(fieldDistance) || !validateNumberField(fieldDistance)) alert += "Distance, ";  
 if (!validateEmptyField(fieldPrice) || !validateNumberField(fieldPrice)) alert += "Price, ";  
 if (!validateEmptyCombo(fieldDestination)) alert += "Destination, ";  
 if (!validateEmptyCombo(fieldDeparture)) alert += "Departure, ";  
 if (busListview.getSelectionModel().getSelectedItem() == null) alert += "Bus, ";  
 if (chauffeurList.getSelectionModel().getSelectedItem() == null) alert += "Chauffeur, ";  
  
 if (length == alert.length()) {  
 //save it DataHandler. .....  
 Bus bus = (Bus) busListview.getSelectionModel().getSelectedItem();  
 Chauffeur chauffeur = (Chauffeur) chauffeurList.getSelectionModel().getSelectedItem();  
  
 Destination pickUp;  
 Destination destination;  
  
 if (DataHandler.*getDestinationList*().findByName(fieldDeparture.getValue().toString()) != null) {  
 pickUp = DataHandler.*getDestinationList*().findByName(fieldDeparture.getValue().toString());  
 } else {  
 pickUp = new Destination(fieldDeparture.getValue().toString());  
 DataHandler.*getDestinationList*().add(pickUp);  
 }  
  
 if (DataHandler.*getDestinationList*().findByName(fieldDestination.getValue().toString()) != null) {  
 destination = DataHandler.*getDestinationList*().findByName(fieldDestination.getValue().toString());  
 } else {  
 destination = new Destination(fieldDestination.getValue().toString());  
 DataHandler.*getDestinationList*().add(destination);  
 }  
  
 int distance = Integer.*parseInt*(fieldDistance.getText());  
  
 Trip trip = new Trip(bus, chauffeur, pickUp, destination, distance, startDatePicker.getValue(), fieldStartTime.getText(), endDatePicker.getValue(), fieldEndTime.getText(), Integer.*parseInt*(fieldPrice.getText()));  
  
 if (stops != null) {  
 trip.setStops(stops);  
 }  
 if (checkPrivateTrip.isSelected() && customer != null) {  
 trip.setCustomer(customer);  
 }  
  
  
 if (foodCheckBox.isSelected()) {  
 trip.setFood(true);  
 }  
 if (accommodationCheckBox.isSelected()) {  
 trip.setAccommodation(true);  
 }  
 if (ticketCheckBox.isSelected()) {  
 trip.setTickets(true);  
 }  
  
 if (oldTrip != null) {  
 DataHandler.*getTrips*().remove(oldTrip);  
 }  
  
 DataHandler.*getTrips*().add(trip);  
 if (oldTrip != null) {  
 successdisplay("Edited", "Trip was edited.");  
  
 Stage stage = (Stage) accommodationCheckBox.getScene().getWindow();  
 stage.close();  
 } else {  
 successdisplay("Created", "Trip was created.");  
 }  
 DataHandler.*save*();  
 Parent root = FXMLLoader.*load*(getClass().getResource("../View/mainScreen.fxml"));  
 Scene scene = new Scene(root);  
 Main.*stage*.setScene(scene);  
 Main.*stage*.show();  
  
 } else {  
 //alert  
 alertdisplay("Wrong Input", alert);  
 }  
 }  
}

Class TripController is responsible for creating tours. Button create tour calls method called createTour. In the method all data from gui is verified, if there are some mistakes the alert is shown. Class Controller is responsible for verification of data. If there were no mistakes all data is stored in variables and then trip is created. If stops were included method calls setStops method from trip. Method createTour is also responsible for editing existing trips. If private vas ticked new window for customer will appear and after the trip becomes privet = bus and chauffeur. At the end of method the data is saved.

package main.Controller;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXMLLoader;  
import javafx.fxml.Initializable;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.input.MouseEvent;  
import main.Main;  
import main.Model.DataHandler;  
import main.Model.Trip;  
import main.Model.TripList;  
  
import java.io.IOException;  
import java.net.URL;  
import java.time.LocalDate;  
import java.time.Period;  
import java.util.ResourceBundle;  
  
*/\*\*  
 \* Class that manages data and files.  
 \*  
 \** ***@author*** *IT-1Y-A16 Group 1  
 \*/*public class Controller implements Initializable {  
  
 public MenuBar menu;  
 public MenuItem homeHome;  
 public MenuItem homeTour;  
 public MenuItem homeReserve;  
 public MenuItem homeSearch;  
 public MenuItem homeBus;  
 public MenuItem homeBusAdd;  
 public MenuItem homeDriver;  
 public MenuItem homeDriverAdd;  
  
 //main screen  
 public Button createTour;  
 public Button mkReservation;  
 public Button findTrip;  
 //tripList  
 public ListView tripList;  
  
 //bus list  
 public Button addBusView;  
  
 //chauffeur list  
 public Button addChauffeur;  
  
 @Override  
 public void initialize(URL location, ResourceBundle resources) {  
 if (tripList != null) {  
 showList();  
 }  
 }  
  
 */\*\*  
 \* Loads and displays list of trips on the listview.  
 \*/* private void showList() {  
 TripList trips = DataHandler.*getTrips*();  
 trips.sort();  
 ObservableList<Trip> data = FXCollections.*observableArrayList*();  
 for (int i = 0; i < trips.getSize(); i++) {  
 if (trips.getArrayTrip().get(i).getDateStart().isEqual(LocalDate.*now*())) data.add(trips.get(i));  
 if (trips.getArrayTrip().get(i).getDateStart().isAfter(LocalDate.*now*())) data.add(trips.get(i));  
 }  
 tripList.setItems(data);  
 }  
  
 */\*\*  
 \* Changes view in GUI.  
 \*/* public void changeView(MouseEvent mouseEvent) throws IOException {  
  
 Parent root = null;  
  
 //main  
 if ((mouseEvent.getSource() == createTour)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/createTour.fxml"));  
 } else if ((mouseEvent.getSource() == mkReservation)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/makeReservation.fxml"));  
 } else if ((mouseEvent.getSource() == findTrip)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/search.fxml"));  
 }  
  
 //bus view  
 else if ((mouseEvent.getSource() == addBusView)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/addBus.fxml"));  
 }  
  
 //chauffeur view  
 else if ((mouseEvent.getSource() == addChauffeur)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/AddChauffeur.fxml"));  
 }  
  
  
 if (root != null) {  
 Scene scene = new Scene(root);  
  
 Main.*stage*.setScene(scene);  
 Main.*stage*.show();  
 }  
 }  
  
 */\*\*  
 \* Changes view in GUI through menu bar.  
 \*/* public void changeViewMenu(ActionEvent actionEvent) throws IOException {  
  
 Parent root = null;  
  
 if ((actionEvent.getSource() == homeHome)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/mainScreen.fxml"));  
 } else if ((actionEvent.getSource() == homeTour)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/createTour.fxml"));  
 } else if ((actionEvent.getSource() == homeBus)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/busList.fxml"));  
 } else if ((actionEvent.getSource() == homeBusAdd)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/addBus.fxml"));  
 } else if ((actionEvent.getSource() == homeReserve)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/makeReservation.fxml"));  
 } else if ((actionEvent.getSource() == homeSearch)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/search.fxml"));  
 } else if ((actionEvent.getSource() == homeDriver)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/chauffeurList.fxml"));  
 } else if ((actionEvent.getSource() == homeDriverAdd)) {  
 root = FXMLLoader.*load*(getClass().getResource("../View/addChauffeur.fxml"));  
 }  
  
 if (root != null) {  
 Scene scene = new Scene(root);  
 Main.*stage*.setScene(scene);  
 Main.*stage*.show();  
 }  
  
 }  
  
 */\*\*  
 \* Validates if textfied is not empty.  
 \*/* protected boolean validateEmptyField(TextField textField) {  
 return !textField.getText().isEmpty();  
  
 }  
  
 */\*\*  
 \* Validates if textfied contains only numbers.  
 \*/* protected boolean validateNumberField(TextField textField) {  
 return textField.getText().matches("[0-9]+");  
  
 }  
  
 */\*\*  
 \* Validates if textfied contain double number.  
 \*/* protected boolean validateDoubleNumberField(TextField textField) {  
 return textField.getText().matches("[0-9]+.[0-9]+");  
  
 }  
  
 */\*\*  
 \* Validates if textfied contains registration plate, two letters and three numbers.  
 \*/* protected boolean validateNumberPlate(TextField textField) {  
 return textField.getText().matches("[A-Z]{2}[0-9]{5}");  
  
 }  
  
 */\*\*  
 \* Validates if textfield text has given length.  
 \*/* protected boolean validateLength(TextField textField, int length) {  
 if (length < 1) length = length \* (-1);  
 return textField.getText().length() == length;  
  
 }  
  
 */\*\*  
 \* Validates if textfied contains time.  
 \*/* protected boolean validateTimeField(TextField textField) {  
 return textField.getText().matches("([01]?[0-9]|2[0-3]):[0-5][0-9]");  
  
 }  
  
 */\*\*  
 \* Validates if datepicker is empty.  
 \*/* protected boolean validateEmptyDate(DatePicker datePicker) {  
 return datePicker.getValue() != null;  
  
 }  
  
 */\*\*  
 \* Validates if person is adult.  
 \*/* protected boolean validateAdultDate(DatePicker datePicker) {  
 LocalDate birthDate = datePicker.getValue();  
 LocalDate now = LocalDate.*now*();  
 if (birthDate != null) {  
 int age = Period.*between*(birthDate, now).getYears();  
 return (age >= 18);  
 } else {  
 return false;  
 }  
  
 }  
  
 */\*\*  
 \* Validates if combobox is empty.  
 \*/* protected boolean validateEmptyCombo(ComboBox comboBox) {  
 return comboBox.getValue() != null;  
  
 }  
  
 */\*\*  
 \* Validates if email is valid.  
 \*/* protected boolean validateEmail(TextField textField) {  
 return textField.getText().matches("(?:[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*|\"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])\*\")@(?:(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]\*[a-z0-9]:(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)\\])");  
  
 }  
  
 */\*\*  
 \* Displays alert.  
 \*  
 \** ***@param*** *title title for alert  
 \** ***@param*** *message alert's message  
 \*/* protected void alertdisplay(String title, String message) {  
  
 Alert alert = new Alert(Alert.AlertType.*WARNING*);  
 alert.setTitle(title);  
 alert.setHeaderText(null);  
 alert.setContentText(message);  
 alert.showAndWait();  
 }  
  
 */\*\*  
 \* Displays success alert.  
 \*  
 \** ***@param*** *title tittle for mesage  
 \** ***@param*** *message to show  
 \*/* protected void successdisplay(String title, String message) {  
  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle(title);  
 alert.setHeaderText(null);  
 alert.setContentText(message);  
 alert.showAndWait();  
 }  
}

Class Controller is responsible for validation of data all around the gui, and more.

There is also method in Chauffeur and Bus class that checks if its available.

package main.Model;  
  
import java.io.Serializable;  
import java.util.ArrayList;  
import java.util.Date;  
  
*/\*\*  
 \* Class which represents a list of chauffeurs.  
 \*  
 \** ***@author*** *IT-1Y-A16 Group 1  
 \*/*public class ChauffeurList implements Serializable {  
  
 private ArrayList<Chauffeur> chauffeurs;  
  
 */\*\*  
 \* Constructs a list of chauffeurs.  
 \*/* public ChauffeurList() {  
 chauffeurs = new ArrayList<>();  
 }  
  
 */\*\*  
 \** ***@return*** *size of chauffeur list  
 \*/* public int getSize() {  
 return chauffeurs.size();  
 }  
  
 */\*\*  
 \* Adds a given chauffeur to the list.  
 \*  
 \** ***@param*** *chauffeur chauffeur to add  
 \*/* public void add(Chauffeur chauffeur) {  
 chauffeurs.add(chauffeur);  
 }  
  
 */\*\*  
 \* Removes given chauffeur from the list.  
 \*  
 \** ***@param*** *chauffeur chauffeur to remove  
 \*/* public void removeChauffeur(Chauffeur chauffeur) {  
 for (int i = 0; i < chauffeurs.size(); i++) {  
 if (chauffeurs.get(i).equals(chauffeur))  
 chauffeurs.remove(chauffeur);  
 }  
 }  
  
 */\*\*  
 \* Finds chauffeur at the given index.  
 \*  
 \** ***@param*** *index index to look at  
 \** ***@return*** *chauffeur at given index  
 \*/* public Chauffeur getChauffeurByIndex(int index) {  
 return chauffeurs.get(index);  
 }  
  
 */\*\*  
 \** ***@return*** *arraylist of all chauffeurs in the list  
 \*/* public ArrayList<Chauffeur> getArrayChauffeur() {  
 return chauffeurs;  
 }  
  
 */\*\*  
 \* Finds chauffeur with the given name.  
 \*  
 \** ***@param*** *name name to look by  
 \** ***@return*** *chauuffeur with given name  
 \*/* public Chauffeur getByName(String name) {  
 for (Chauffeur chauffeur : chauffeurs) {  
 if (chauffeur.getName().equals(name))  
 return chauffeur;  
 }  
 return null;  
 }  
  
 */\*\*  
 \* Checks if list contains given chauffeur.  
 \*  
 \** ***@param*** *chauffeur chauffeur to check  
 \** ***@return*** *true if list contains given chauffeur  
 \*/* private boolean contains(Chauffeur chauffeur) {  
 return chauffeurs.contains(chauffeur);  
 }  
  
 */\*\*  
 \* Finds all chauffeurs with given preferred distance.  
 \*  
 \** ***@param*** *prefferedDistance distance to look by  
 \** ***@return*** *ChauffeurList of all chauffeurs with given preferred distance  
 \*/* public ChauffeurList getAllByPrefferedDistance(int prefferedDistance) {  
 ChauffeurList result = new ChauffeurList();  
 for (Chauffeur chauffeur : chauffeurs) {  
 if (!chauffeur.isVikar()) {  
 if ((chauffeur.getPreferredDistance() != null) && !(chauffeur.getPreferredDistance().isEmpty())) {  
 for (int j = 0; j < chauffeur.getPreferredDistance().size(); j++) {  
 if (chauffeur.getPreferredDistance().get(j) > prefferedDistance) {  
 if (!result.contains(chauffeur))  
 result.add(chauffeur);  
 }  
 }  
 } else if (chauffeur.getPreferredDistance().isEmpty()) {  
 result.add(chauffeur);  
 }  
 }  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Finds all chauffeurs with given preferred bustype.  
 \*  
 \** ***@param*** *busType bustype to look by  
 \** ***@return*** *ChauffeurList of all chauffeurs with given preferred bustype  
 \*/* public ChauffeurList getAllByPrefferedBus(String busType) {  
 ChauffeurList result = new ChauffeurList();  
 for (Chauffeur chauffeur : chauffeurs) {  
 ArrayList<String> prefferedBuses = chauffeur.getArrayBusTypes();  
 if (!chauffeur.isVikar()) {  
 if (!prefferedBuses.isEmpty()) {  
 for (String prefferedBuse : prefferedBuses) {  
 if (busType.equals(prefferedBuse)) {  
 if (!result.contains(chauffeur))  
 result.add(chauffeur);  
 }  
 }  
 } else  
 result.add(chauffeur);  
 }  
 }  
  
 return result;  
 }  
  
 */\*\*  
 \** ***@return*** *ChauffeurList of all with vicar contract  
 \*/* public ChauffeurList getAllVicars() {  
 ChauffeurList result = new ChauffeurList();  
 for (Chauffeur chauffeur : chauffeurs) {  
 if (chauffeur.isVikar()) {  
 result.add(chauffeur);  
 }  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Removes all vicars in the list.  
 \*/* public void removeAllVicars() {  
 for (int i = 0; i < chauffeurs.size(); i++) {  
 if (chauffeurs.get(i).isVikar()) {  
 chauffeurs.remove(chauffeurs.get(i));  
 }  
 }  
 }  
  
 */\*\*  
 \** ***@return*** *List converted to String  
 \*/* public String toString() {  
 String s = "";  
 for (int i = 0; i < chauffeurs.size(); i++) {  
 s += chauffeurs.get(i);  
 if (i < chauffeurs.size() - 1)  
 s += "\n";  
 }  
 return s;  
 }  
  
 */\*\*  
 \* Finds all chauffeurs who are available in the given date interval.  
 \*  
 \** ***@param*** *from start of date interval  
 \** ***@param*** *to end of date interval  
 \** ***@return*** *ChauffeurList of all available in the given date interval  
 \*/* public ChauffeurList getAvailable(Date from, Date to) {  
 ArrayList<Chauffeur> inTrips = new ArrayList<>();  
 ChauffeurList result = new ChauffeurList();  
 TripList trips = DataHandler.*getTrips*();  
 for (int i = 0; i < trips.getArrayTrip().size(); i++) {  
 for (Chauffeur chauffeur : chauffeurs) {  
 if (chauffeur.equals(trips.getArrayTrip().get(i).getChauffeur())) {  
 if (!inTrips.contains(chauffeur))  
 inTrips.add(chauffeur);  
 if (((from.before(trips.getArrayTrip().get(i).getDateObjStart())) && (to.before(trips.getArrayTrip().get(i).getDateObjStart())))  
 || ((from.after(trips.getArrayTrip().get(i).getDateObjEnd())) && (to.after(trips.getArrayTrip().get(i).getDateObjEnd())))) {  
 result.add(chauffeur);  
 }  
 }  
 }  
 }  
 for (Chauffeur chauffeur : chauffeurs) {  
 if (!inTrips.contains(chauffeur))  
 result.add(chauffeur);  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Copies this list.  
 \*  
 \** ***@return*** *Copy of this ChauffeurList  
 \*/* public ChauffeurList copy() {  
 ChauffeurList chauffeurList = new ChauffeurList();  
 for (Chauffeur chauffeur : chauffeurs) {  
 chauffeurList.add(chauffeur);  
 }  
 return chauffeurList;  
 }  
  
}

After inserting dates in the gui. Method getAvailable is called, which will find all chaufers and that are free in given date interval. After inserting distance, method getAllByPrefferedDistance returns all chauffeurs who prefer given distance or have no preferred. After choosing bus type method getAllByPrefferedBus will find chauffeurs who prefer specified bus or have no preferred. Also only buses of given type are shown. This is done by method loadBusList in TripController which also calls getAvailable method from BusList class. Adding stops to trip is handled by method handleStops.