Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный технический университет”
Кафедра ИИТ

Отчёт
По лабораторной работе №10
По дисциплине СПП

Выполнил
Студент группы ПО-3
3-го курса
Кулинкович И. Т.

Проверил
Крощенко А. А.

Брест 2021

Лабораторная работа №10

ВАРИАНТ 13

На основе БД, разработанной в лабораторной работе No9, реализовать многооконное приложение-клиент, позволяющее выполнять основные операции над таблицей в БД (добавление, удаление, модификацию данных).

Основные требования к приложению:
• Для отображения выбирать таблицу с внешними ключами;
• Осуществлять вывод основных данных в табличном представлении;
• При выводе краткого представления записи в таблице (т.е. если выводятся не все поля), по щелчку мышкой на запись осуществлять вывод всех полей в подготовленные компоненты на форме;
• Для всех полей, представленных внешними ключами, выводить их текстовое представление
из связанных таблиц (например, таблица-справочник «Времена года» содержит два поля – идентификатор и название сезона, в связанной таблице «Месяц года» есть внешний ключ на таблицу «Времена года»; в этом случае при выводе таблицы «Месяц года» нужно выводить название сезона, а не его идентификатор);
• При выводе предусмотреть упорядочивание по столбцу;
• Реализовать простейший фильтр данных по одному-двум полям;
• При добавлении новых данных в таблицу использовать дополнительное окно для ввода;
• При модификации данных можно использовать ту же форму, что и для добавления, но с внесенными актуальными значениями полей;
• При добавлении/модификации выводить варианты значений полей с внешним ключом с помощью выпадающего списка;
• При удалении данных осуществлять удаление записи, на которой в данных момент находится
фокус.

13) База данных *«Формула-1»*

Для разработки выбираем базу данных SQLite и библиотеку sqlite-jdbc.

Код программы

live.ilyusha.DBProcess

```
package live.ilyusha;

import java.sql.*;

public class DBProcess {
    private static final String url = "jdbc:sqlite:sample.db";
    public Connection con;
    private static Statement stmt;
    private static ResultSet rs;
    public Savepoint point;

    public DBProcess() {
        try {
            con = DriverManager.getConnection(url);
            stmt = con.createStatement();
        } catch (SQLException sqlEx) {
            sqlEx.printStackTrace();
        }
    }

    public void reconnect() {
```

```
        try {
            con.close();
            con = DriverManager.getConnection(url);
            stmt = con.createStatement();
        } catch (SQLException _) {}
    }

    public void setup() throws SQLException {
        update("create table car ( car_id integer not null constraint car_pk
primary key autoincrement, make varchar(40), name varchar(40) )");
        update("create unique index car_car_id_uindex on car (car_id)");
        update("create table race_location ( race_location_id integer not
null constraint race_location_pk primary key autoincrement, name
varchar(100), country varchar(100) )");
        update("create table race ( race_id integer not null constraint
races_pk primary key autoincrement, winner_id integer references racer
(racer_id), race_location_id integer references race_location )");
        update("create unique index races_race_id_uindex on race (race_id)");
        update("create unique index race_location_race_location_id_uindex on
race_location (race_location_id)");
        update("create table racer ( racer_id integer not null constraint
racer_pk primary key autoincrement, name varchar(100), birth_year integer,
race_id integer references race, car_id integer references car )");
        update("create unique index racer_racer_id_uindex on racer
(racer_id)");
        update("create table sponsorship ( sponsorship_id integer not null
constraint sponsorship_pk primary key autoincrement, company_name
varchar(40), pay integer, race_id integer references race )");
        update("create unique index sponsorship_sponsorship_id_uindex on
sponsorship (sponsorship_id)");

        // Fill with test data
        insert("insert or ignore into car (car_id, make, name) values (0,
\"Toyota\", \"Supra\")");
        insert("insert or ignore into race_location (race_location_id, name,
country) values (0, \"Nürburgring\", \"Germany\")");
        insert("insert or ignore into racer (racer_id, name, birth_year,
race_id, car_id) values (0, \"Valtteri Bottas\", 1989, 0, 0)");
        insert("insert or ignore into race (race_id, winner_id,
race_location_id) values (0, 0, 0)");
        insert("insert or ignore into sponsorship (sponsorship_id,
company_name, pay, race_id) values (0, \"Nissan\", 2900000, 0)");
        insert("insert or ignore into car (car_id, make, name) values (1,
\"Bmw\", \"M4\")");
        insert("insert or ignore into racer (racer_id, name, birth_year,
race_id, car_id) values (1, \"Lewis Hamilton\", 1985, 0, 1)");
        insert("insert or ignore into race_location (race_location_id, name,
country) values (1, \"Adelaide Street\", \"Australia\")");
        insert("insert or ignore into race (race_id, winner_id,
race_location_id) values (1, 1, 1)");
    }

    public void insert(String query) throws SQLException {
        stmt.executeUpdate(query);
    }

    public ResultSet select(String query) throws SQLException {
        rs = stmt.executeQuery(query);
        return rs;
    }
```

```
    public void delete(String query) throws SQLException {
        stmt.executeUpdate(query);
    }

    public void update(String query) throws SQLException {
        stmt.executeUpdate(query);
    }
}
```

live.ilyusha.Main

```
package live.ilyusha;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

import java.io.File;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        DBProcess db = new DBProcess();

        Stage stage = new Stage();
        final Button resetButton = new Button("Reset table");
        resetButton.setMinWidth(200);
        resetButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                try {
                    new File("sample.db").delete();
                    Thread.sleep(1000);
                    db.reconnect();
                    db.setup();
                } catch (SQLException | InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        final Button racesButton = new Button("Show table");
        racesButton.setMinWidth(200);
        racesButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                try {
                    stage.close();
                    RacesView racesView = new RacesView();
                    racesView.start(stage);
                } catch (SQLException e) {
                    e.printStackTrace();
```

```
                }
            }
        });
        VBox root = new VBox();
        root.setPadding(new Insets(5));
        root.setSpacing(20);
        root.setAlignment(Pos.CENTER);
        root.getChildren().addAll(resetButton, racesButton);
        primaryStage.setTitle("Formula 1");
        Scene scene = new Scene(root, 300, 140);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

live.ilyusha.AddRaceForm

```
package live.ilyusha;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class AddRaceForm extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) throws SQLException {
        List<Location> locations = new ArrayList<>();
        List<Racer> racers = new ArrayList<>();
        List<String> locationsNames = new ArrayList<>();
        List<String> racersNames = new ArrayList<>();
        DBProcess db = new DBProcess();
        ResultSet locationsResult = db.select("select * from race_location");
        while (locationsResult.next()) {
            locations.add(new
Location(locationsResult.getInt("race_location_id"),
locationsResult.getString("name"), locationsResult.getString("country")));
        }
```

```
        ResultSet racersResult = db.select("select racer.racer_id as racerId,
racer.name as racerName, racer.birth_year as racerBirthYear, car.make as
carMake, car.name as carName from racer inner join car on racer.car_id =
car.car_id");
        while (racersResult.next()) {
            racers.add(new Racer(
                    racersResult.getInt("racerId"),
                    racersResult.getString("racerName"),
                    racersResult.getInt("racerBirthYear"),
                    new Car(
                        racersResult.getString("carMake"),
                        racersResult.getString("carName")
                    )
            ));
        }
        for (Location location : locations) {
            locationsNames.add(location.name);
        }
        for (Racer racer : racers) {
            racersNames.add(racer.name);
        }
        ObservableList<String> locationsList =
FXCollections.observableArrayList(locationsNames);
        ObservableList<String> racersList =
FXCollections.observableArrayList(racersNames);
        final ComboBox<String> addLocation = new
ComboBox<String>(locationsList);
        addLocation.setMinWidth(250);
        addLocation.setMaxWidth(250);
        addLocation.setValue("Choose a location");
        final ComboBox<String> addRacer = new ComboBox<String>(racersList);
        addRacer.setMinWidth(250);
        addRacer.setMaxWidth(250);
        addRacer.setValue("Choose the race winner");
        final Button addButton = new Button("Add");
        addButton.setMinWidth(250);
        addButton.setMaxWidth(250);
        addButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                if (!addLocation.getValue().equals("Choose a location") &&
                        !addRacer.getValue().equals("Choose the race
winner")) {
                    try {
                        String locationName = addLocation.getValue();
                        String racerName = addRacer.getValue();
                        Integer locationId = locations.stream().filter(x ->
x.name == locationName).findAny().map(x -> x.id).orElse(0);
                        Integer racerId = racers.stream().filter(x -> x.name
== racerName).findAny().map(x -> x.id).orElse(0);
                        db.insert("insert into race (winner_id,
race_location_id)" +
                                "values (" + racerId + ", " + locationId +
")");
                        primaryStage.close();

                    } catch (SQLException ex) {
                        ex.printStackTrace();
                    }
                }
```

```
            }
        });
        VBox root = new VBox();
        root.setPadding(new Insets(5));
        root.setSpacing(10);
        root.setAlignment(Pos.CENTER);
        root.getChildren().addAll(addLocation, addRacer, addButton);
        primaryStage.setTitle("Add Race");
        Scene scene = new Scene(root, 270, 150);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

live.ilyusha.Car

```
package live.ilyusha;

public class Car {
    String make;
    String name;

    public Car(String make, String name) {
        this.make = make;
        this.name = name;
    }
}
```

live.ilyusha.Location

```
package live.ilyusha;

public class Location {
    Integer id;
    String name;
    String country;

    public Location(int id, String name, String country) {
        this.id = id;
        this.name = name;
        this.country = country;
    }
}
```

live.ilyusha.Race

```
package live.ilyusha;

public class Race {
    Integer id;
    Racer winner;
    Location location;

    public Race(Integer id, Racer winner, Location location) {
        this.id = id;
        this.winner = winner;
        this.location = location;
    }
}
```

live.ilyusha.Racer

```
package live.ilyusha;

public class Racer {
    Integer id;
    String name;
    Integer birthYear;
    Car car;

    public Racer(int id, String name, Integer birthYear, Car car) {
        this.id = id;
        this.name = name;
        this.birthYear = birthYear;
        this.car = car;
    }
}
```

live.ilyusha.RacesView

```
package live.ilyusha;

import javafx.application.Application;
import javafx.beans.property.SimpleObjectProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.collections.transformation.FilteredList;
import javafx.collections.transformation.SortedList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class RacesView extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) throws SQLException {
        DBProcess db = new DBProcess();
        ResultSet racesResult = db.select("select " +
                "race.race_id as raceId, racer.racer_id as racerId,
racer.name as racerName, racer.birth_year as racerBirthYear, " +
```

```
                "race_location.race_location_id as locationId,
race_location.name as locationName, race_location.country as locationCountry,
" +
                "car.make as carMake, car.name as carName from race " +
                "inner join racer on race.winner_id = racer.racer_id " +
                "inner join car on racer.car_id = car.car_id " +
                "inner join race_location on race.race_location_id =
race_location.race_location_id");

        List<Race> races = new ArrayList<>();
        while (racesResult.next()) {
            races.add(new Race(
                racesResult.getInt("raceId"),
                new Racer(racesResult.getInt("racerId"),
racesResult.getString("racerName"), racesResult.getInt("racerBirthYear"), new
Car(
                    racesResult.getString("carMake"),
racesResult.getString("carName")
                )),
                new Location(racesResult.getInt("locationId"),
racesResult.getString("locationName"),
racesResult.getString("locationCountry"))
            ));
        }
        ObservableList<Race> racesList =
FXCollections.observableArrayList(races);
        TableView<Race> table = new TableView<Race>(racesList);
        table.setMinWidth(480);
        table.setEditable(false);

        TableColumn<Race, Integer> columnId = new TableColumn<Race,
Integer>("Race ID");
        columnId.setCellValueFactory(new
Callback<TableColumn.CellDataFeatures<Race, Integer>,
ObservableValue<Integer>>() {
            @Override
            public ObservableValue<Integer>
call(TableColumn.CellDataFeatures<Race, Integer> param) {
                return new
SimpleObjectProperty<Integer>(param.getValue().id);
            }
        });

        TableColumn<Race, String> columnWinner = new TableColumn<Race,
String>("Race winner name");
        columnWinner.setCellValueFactory(new
Callback<TableColumn.CellDataFeatures<Race, String>,
ObservableValue<String>>() {
            @Override
            public ObservableValue<String>
call(TableColumn.CellDataFeatures<Race, String> param) {
                return new
SimpleObjectProperty<String>(param.getValue().winner.name);
            }
        });

        TableColumn<Race, String> columnWinnerCar = new TableColumn<Race,
String>("Race winner's car");
```

```
        columnWinnerCar.setCellValueFactory(new
Callback<TableColumn.CellDataFeatures<Race, String>,
ObservableValue<String>>() {
            @Override
            public ObservableValue<String>
call(TableColumn.CellDataFeatures<Race, String> param) {
                return new
SimpleObjectProperty<String>(param.getValue().winner.car.make + " " +
param.getValue().winner.car.name);
            }
        });

       TableColumn<Race, String> columnLocation = new TableColumn<Race,
String>("Race location");
        columnLocation.setCellValueFactory(new
Callback<TableColumn.CellDataFeatures<Race, String>,
ObservableValue<String>>() {
            @Override
            public ObservableValue<String>
call(TableColumn.CellDataFeatures<Race, String> param) {
                return new
SimpleObjectProperty<String>(param.getValue().location.name + ", " +
param.getValue().location.country);
            }
        });
        table.getColumns().addAll(columnId, columnWinner, columnWinnerCar,
columnLocation);

        final Race[] race = new Race[1];
        TableView.TableViewSelectionModel<Race> selectionModel =
table.getSelectionModel();
        selectionModel.selectedItemProperty().addListener(new
ChangeListener<Race>() {
            @Override
            public void changed(ObservableValue<? extends Race>
observableValue, Race oldRace, Race
                    newRace) {
                if (newRace != null) {
                    race[0] = newRace;
                }
            }
        });
        Stage updateStage = new Stage();
        final Button updateButton = new Button("Update");
        updateButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                try {
                    updateStage.close();
                    UpdateRaceForm updateRace = new UpdateRaceForm(race[0]);
                    updateRace.start(updateStage);
                } catch (NullPointerException | SQLException e) {
                    try {
                        start(primaryStage);
                    } catch (SQLException ex) {
                        ex.printStackTrace();
                    }
                }
            }
        });
```

```
        Stage addStage = new Stage();
        final Button addButton = new Button("Add");
        addButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                try {
                    addStage.close();
                    AddRaceForm addRace = new AddRaceForm();
                    addRace.start(addStage);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        });
        final Button deleteButton = new Button("Delete");
        deleteButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                if (table.getSelectionModel().getSelectedItem() != null) {
                    Race selectedItem =
table.getSelectionModel().getSelectedItem();
                    try {
                        db.delete("delete from race where race_id == " +
selectedItem.id);
                        start(primaryStage);
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        final Button refreshButton = new Button("Refresh");
        refreshButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                try {
                    start(primaryStage);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        });
        FilteredList<Race> filteredData = new FilteredList<>(racesList, p ->
true);
        final TextField winnerFilter = new TextField();
        winnerFilter.setPromptText("Winner filter");
        winnerFilter.setMinWidth(250);
        winnerFilter.textProperty().addListener((observable, oldValue,
newValue) -> {
            filteredData.setPredicate(currentRace -> {
                if (newValue == null || newValue.isEmpty()) {
                    return true;
                }
                String lowerCaseFilter = newValue.toLowerCase();
                return
currentRace.winner.name.toLowerCase().contains(newValue);
            });
        });
        SortedList<Race> sortedData = new SortedList<>(filteredData);
        sortedData.comparatorProperty().bind(table.comparatorProperty());
```

```
        table.setItems(sortedData);
        VBox root = new VBox();
        HBox deleteBox = new HBox();
        HBox addBox = new HBox();
        root.setPadding(new Insets(5));
        root.setSpacing(10);
        root.getChildren().addAll(deleteBox, table, addBox);
        deleteBox.setPadding(new Insets(5));
        deleteBox.setSpacing(10);
        deleteBox.getChildren().addAll(deleteButton, refreshButton);
        addBox.setPadding(new Insets(5));
        addBox.setSpacing(10);
        addBox.getChildren().addAll(winnerFilter, addButton, updateButton);
        primaryStage.setTitle("Races Table");
        Scene scene = new Scene(root, 550, 500);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

live.ilyusha.UpdateRaceForm

```
package live.ilyusha;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class UpdateRaceForm extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    public Race race;

    public UpdateRaceForm(Race updateRace) {
        this.race = updateRace;
    }

    @Override
    public void start(Stage primaryStage) throws SQLException {
        List<Location> locations = new ArrayList<>();
        List<Racer> racers = new ArrayList<>();
        List<String> locationsNames = new ArrayList<>();
        List<String> racersNames = new ArrayList<>();
```

```
        DBProcess db = new DBProcess();
        ResultSet locationsResult = db.select("select * from race_location");
        while (locationsResult.next()) {
            locations.add(new
Location(locationsResult.getInt("race_location_id"),
locationsResult.getString("name"), locationsResult.getString("country")));
        }
        ResultSet racersResult = db.select("select racer.racer_id as racerId,
racer.name as racerName, racer.birth_year as racerBirthYear, car.make as
carMake, car.name as carName from racer inner join car on racer.car_id =
car.car_id");
        while (racersResult.next()) {
            racers.add(new Racer(
                    racersResult.getInt("racerId"),
                    racersResult.getString("racerName"),
                    racersResult.getInt("racerBirthYear"),
                    new Car(
                            racersResult.getString("carMake"),
                            racersResult.getString("carName")
                    )
            ));
        }
        for (Location location : locations) {
            locationsNames.add(location.name);
        }
        for (Racer racer : racers) {
            racersNames.add(racer.name);
        }
        ObservableList<String> locationsList =
FXCollections.observableArrayList(locationsNames);
        ObservableList<String> racersList =
FXCollections.observableArrayList(racersNames);
        final ComboBox<String> updateLocation = new
ComboBox<String>(locationsList);
        updateLocation.setMinWidth(250);
        updateLocation.setMaxWidth(250);
        updateLocation.setValue(race.location.name);
        final ComboBox<String> updateRacer = new
ComboBox<String>(racersList);
        updateRacer.setMinWidth(250);
        updateRacer.setMaxWidth(250);
        updateRacer.setValue(race.winner.name);
        final Button updateButton = new Button("Update");
        updateButton.setMinWidth(250);
        updateButton.setMaxWidth(250);
        updateButton.setOnAction(
                new EventHandler<ActionEvent>() {
                    @Override
                    public void handle(ActionEvent e) {
                        try {
                            String locationName = updateLocation.getValue();
                            String racerName = updateRacer.getValue();
                            Integer locationId = locations.stream().filter(x
-> x.name == locationName).findAny().map(x -> x.id).orElse(0);
                            Integer racerId = racers.stream().filter(x ->
x.name == racerName).findAny().map(x -> x.id).orElse(0);
                            db.update("update race set " +
                                    "winner_id = " + racerId + ",
race_location_id = " + locationId + " where " +
                                    "race_id = " + race.id);
```

```
                        primaryStage.close();
                    } catch (SQLException ex) {
                        ex.printStackTrace();
                    }
                }
            }
        );
        VBox root = new VBox();
        root.setPadding(new Insets(5));
        root.setSpacing(10);
        root.setAlignment(Pos.CENTER);
        root.getChildren().addAll(updateLocation, updateRacer, updateButton);
        primaryStage.setTitle("Update Race");
        Scene scene = new Scene(root, 270, 150);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

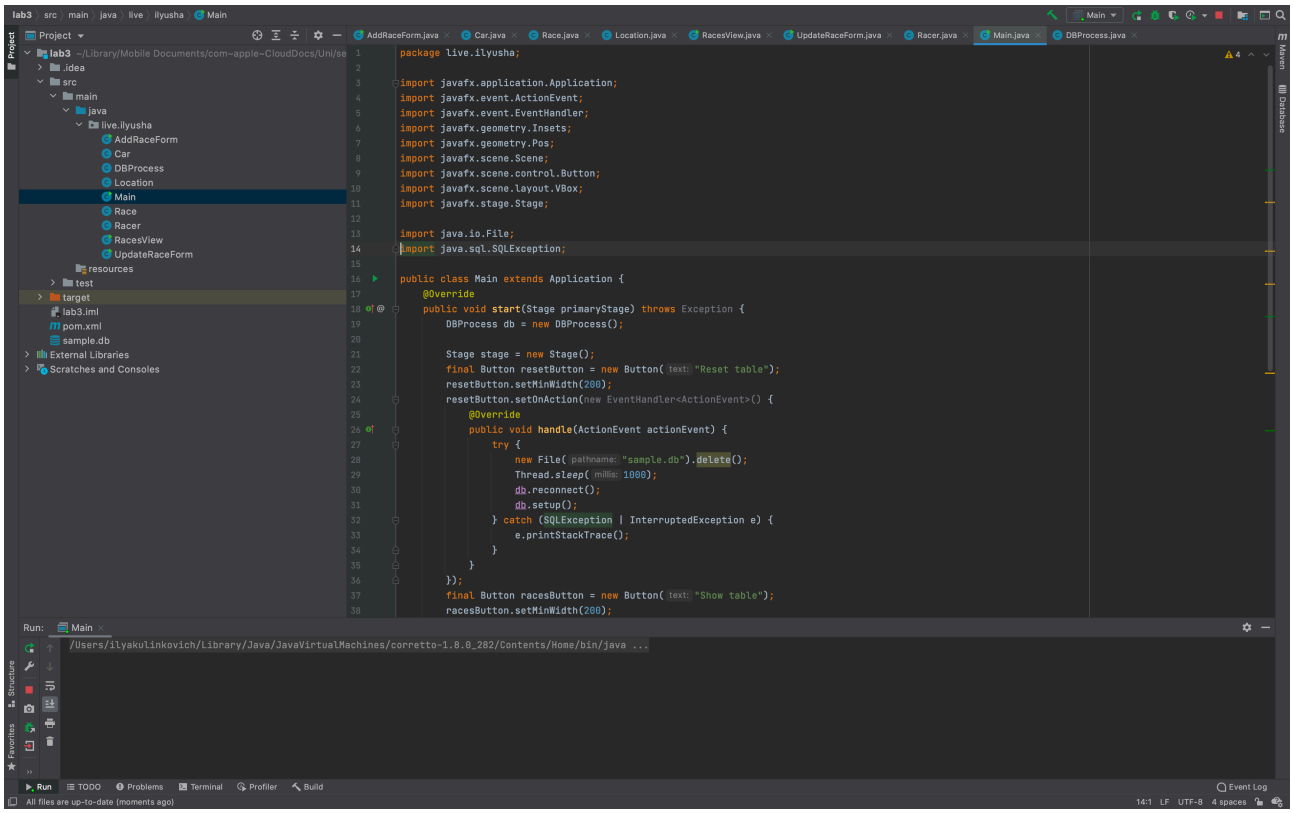
Спецификация ввода

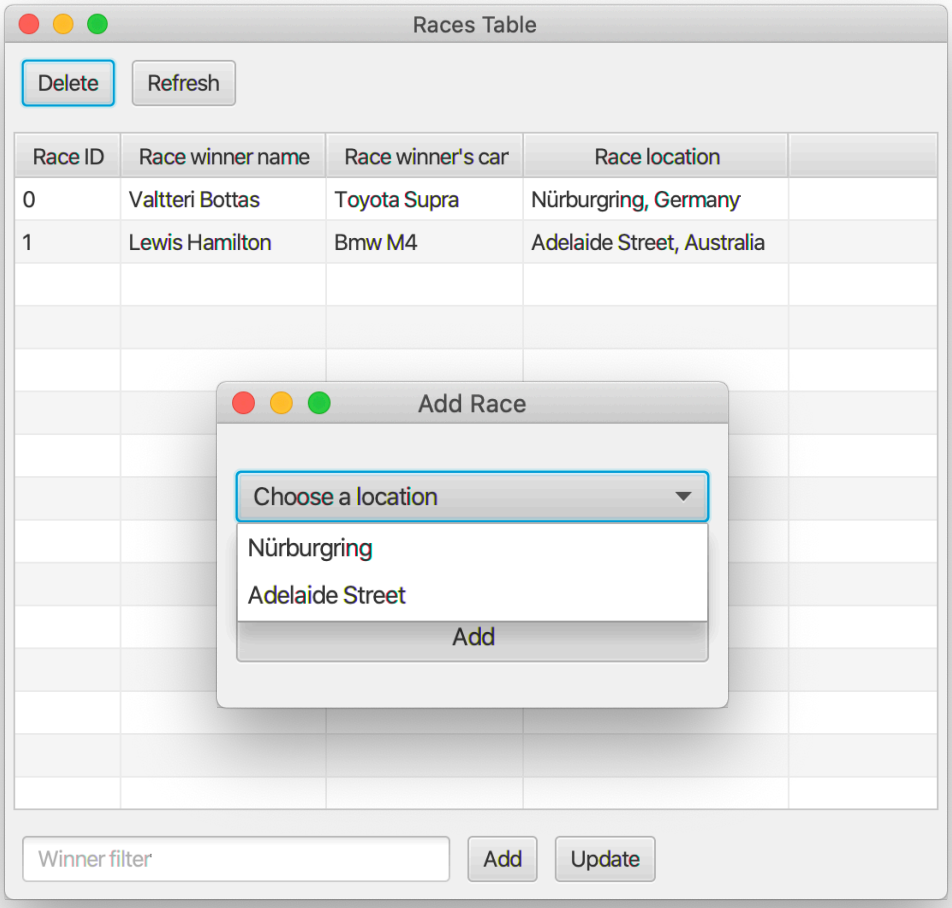
Данные не вводятся при запуске приложения, приложение предоставляет ввод различных данных в рамках своего функционала для хранения в базе данных

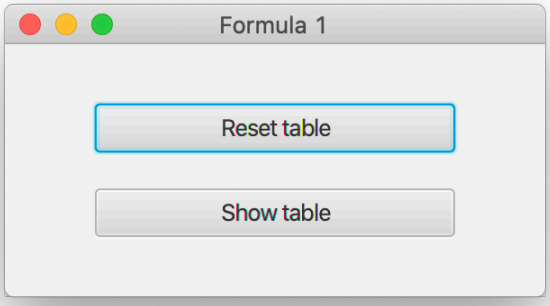
Спецификация вывода

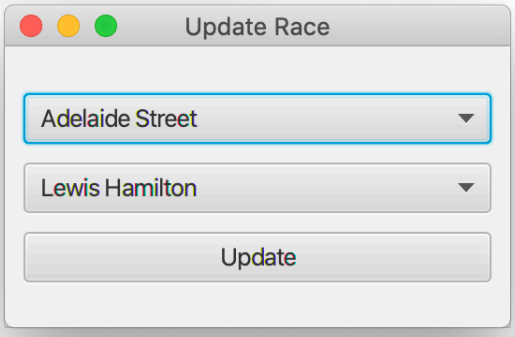
Графическое приложение

Рисунки с результатами работы программы









Вывод

В данной лабораторной работе я приобрел практические навыки разработки многооконных приложений на JavaFX для работы с базами данных.