**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ВТ**

Пояснительная записка к курсовой работе

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент гр. 9308 | Дементьев Д.П. |
| Преподаватель | Гречухин М. Н. |

Содержание

[Цель и задачи 3](#_Toc73829388)

[Техническое задание 4](#_Toc73829389)

[Введение 4](#_Toc73829390)

[Основание для разработки 4](#_Toc73829391)

[Назначение разработки 4](#_Toc73829392)

[Требования к программе 5](#_Toc73829393)

[Требования к функциональным характеристикам 5](#_Toc73829394)

[Требования к организации и форме представления выходных данных 5](#_Toc73829395)

[Требования к организации и форме представления входных данных 5](#_Toc73829396)

[Требования к надёжности 5](#_Toc73829397)

[Условия эксплуатации 5](#_Toc73829398)

[Требование к составу и параметрам технических средств 6](#_Toc73829399)

[Требования к информационной и программной совместимости 6](#_Toc73829400)

[Требования к программной документации 7](#_Toc73829401)

[Стадии и этапы разработки 7](#_Toc73829402)

[Порядок контроля и приемки 7](#_Toc73829403)

[Описание процесса проектирования ПК 8](#_Toc73829404)

[Описание вариантов использования ПК 8](#_Toc73829405)

[Создание прототипа интерфейса пользователя 10](#_Toc73829406)

[Построение диаграммы программных классов 15](#_Toc73829407)

[Описание классов 17](#_Toc73829408)

[Описание базы данных 36](#_Toc73829409)

[Руководство оператору 39](#_Toc73829410)

[Назначение программы 39](#_Toc73829411)

[Описание задачи 39](#_Toc73829412)

[Входные и выходные данные 39](#_Toc73829413)

[Запуск программы 39](#_Toc73829414)

[Заключение 44](#_Toc73829415)

[Приложение. Исходный текст программы 45](#_Toc73829416)

[Сущности 45](#_Toc73829417)

[Интерфейс 58](#_Toc73829418)

[Утилиты 88](#_Toc73829419)

# Цель и задачи

Целью курсового проектирования является закрепление и углубление теоретических знаний, приобретение практических навыков по проектированию и разработке программного обеспечения на объектно-ориентированном языке Java. В задачи курсового проектирования входят:

* изучение особенностей конкретной предметной области, относящейся к заданию на курсовой проект, и разработка технического задания на программный комплекс (ПК);
* объектно-ориентированное проектирование ПК с использованием языка UML;
* разработка ПК на объектно-ориентированном языке;
* написание программной документации

# Техническое задание

## Введение

В ходе работы потребуется разработать ПК для диспетчера автобусного парка. В ПК должна храниться информация о водителях, маршрутах и графике движения автобусов. Диспетчер автобусного парка может добавлять, изменять и удалять эту информацию. Ему могут потребоваться следующие сведения:

* + - * список водителей, их стаж работы и класс;
      * когда начинается или заканчивается движение автобуса на всех или отдельных маршрутах;
      * справка о графике движения автобусов и отчет о его нарушениях.

## Основание для разработки

Основанием для разработки ПК является курсовой проект по дисциплине «Объектно-ориентированное программирование».

## Назначение разработки

ПК разрабатывается в учебных целях для демонстрации полученных знаний и умений за прошедший период обучения.

# Требования к программе

## Требования к функциональным характеристикам

ПК должен обеспечивать выполнение следующих функций:

* просмотр, добавление, изменения базы данных (БД);
* выдача справочной информации согласно техническому заданию

ПК должен обеспечивать ведение и хранение следующих данных:

* информации об автобусах;
* информации о водителях;
* информации о маршрутах;
* информации о расписаниях;
* информации о нарушениях, связанных с сущностями описанными выше

## Требования к организации и форме представления выходных данных

Выходные данные должны быть представлены в виде таблицы, содержащей сведения указанных ранее данных.

## Требования к организации и форме представления входных данных

Ввод исходных данных должен осуществляться в режиме диалога через графический интерфейс ПК. Вводимые данные являются значениями характеристик объектов, хранимых в БД.

## Требования к надёжности

ПК должен устойчиво функционировать при соблюдении гарантии устойчивого функционирования операционной системы и системы управления базой данных. Должен быть обеспечен контроль входных данных на предмет соответствия предполагаемому типу. Для выходных данных должно быть обеспечено свободное пространство для записи на внешний носитель, также предпочтительна кодировка UTF-8 для ОС во избежание ошибок чтения/записи между ПК и операционной системой.

## Условия эксплуатации

Выполнение ПК своих функций корректным будет при наличии стабильного подключения к СУБД. Обслуживание ПК производится одной персоной, имеющей доступ к БД, особых требований к квалификации обслуживающей персоны не предполагается.

## Требование к составу и параметрам технических средств

Задача должна решаться на ПЭВМ с процессором Intel Core i3 2100 и выше, ОЗУ не менее 2 Гб, монитор, видеокарта 512 Мб, клавиатура, манипулятор типа "мышь".

В зависимости от места хранения СУБД непосредственно на внешнем носителе (жёстком диске) или на удаленном сервере требуется достаточное количество свободного пространства на носителе HDD не менее 64 Гб или стабильное подключение к сети соответственно.

## Требования к информационной и программной совместимости

Входная и выходная информация ПК должна быть удобна для визуального восприятия. ПК должен быть реализован на языке программирования Java и быть совместим с операционной системой Windows.

Обязательными требованиями при разработке кода ПК являются использование следующих конструкций языка Java:

* закрытые и открытые члены классов;
* наследование;
* конструкторы с параметрами;
* обработка исключительных ситуаций;
* динамическое создание объектов.

## Требования к программной документации

Документация должна быть представлена в следующем составе:

* описание процесса проектирования ПК
* руководство оператора
* исходный текст ПК

## Стадии и этапы разработки

* Разработка технического задания
* Описание вариантов использования ПК
* Создание прототипа интерфейса пользователя
* Разработка объектной модели ПК
* Построение диаграмм программных классов
* Описание поведения ПК
* Построение диаграмм действий

## Порядок контроля и приемки

В процессе приема работы устанавливается соответствие ПК и прилагаемой документации требованиям, обозначенным в техническом задании.

Разработанная ПК должна корректно реагировать на внештатные ситуации следующих типов:

* Некорректные входные данные
* Пресечение или, по крайней мере, минимизация случайных изменений в БД из-за человеческого фактора

Также ПК должна иметь интуитивно понятный интерфейс ПК на русском языке.

# Описание процесса проектирования ПК

## Описание вариантов использования ПК

Развернутое описание функциональных требований осуществляется на этапе проектирования комплекса. Для того чтобы детализировать требования, необходимо выделить процессы, происходящие в заданной предметной области. Описание таких процессов на UML выполняется в виде прецедентов (use case). Прецеденты являются сценарием или вариантом использования ПК при взаимодействии с внешней средой. Они являются продолжением описаний требований и функциональных спецификаций, указанных в техническом задании. Прецедент изображается в виде эллипса, в котором содержится имя прецедента. Название прецедента обязательно включает в себя глагол, выражающий суть выполняемой функции. С помощью прецедентов описывается функционирование ПК с точки зрения внешнего пользователя, который называется в UML актором (actor). Актор представляет собой любую внешнюю по отношению к моделируемой системе сущность (человек, программная система, устройство), которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Актор на диаграмме изображается пиктограммой в виде человечка, под которым указано его имя. Совокупность функций, реализуемых ПК, изображается в виде диаграммы (use case diagram). Для построения диаграммы необходимо определить акторов, прецеденты (функции) и взаимоотношение между акторами и прецедентами; между прецедентами, если один прецедент расширяет или использует другой. В языке UML для вариантов использования и действующих лиц поддерживается несколько типов связей. Это связи коммуникации (communication), использования (uses) и расширения (extends).

Связь коммуникации — это связь между прецедентом и актором. На языке UML связь коммуникации изображают в виде стрелки. Направление стрелки показывает, кто инициирует коммуникацию. При задании коммуникации необходимо указать данные, которые вводит или получает пользователь. Кроме данных на концах стрелки можно указать кратности отношения, которые характеризуют количество взаимодействующих между собой акторов и прецедентов. На диаграммах прецедентов наиболее распространенными являются две формы записи кратности 1 и 1 … \*. Первая форма записи означает, что один актор (прецедент) участвует во взаимодействии, а вторая форма записи, что один или несколько акторов (прецедентов) участвуют во взаимодействии.

Связь использования предполагает, что один прецедент всегда применяет функциональные возможности другого. С помощью таких связей структурируют прецеденты, показывая тем самым, какой прецедент является составной частью другого прецедента. Такой включаемый прецедент является абстрактным прецедентом в том смысле, что он не может исполняться независимо от других прецедентов, а лишь в их составе. Связь использования изображается с помощью стрелок и слова «uses» (использование). Направление стрелки указывает, какой прецедент используется для реализации функциональности другого прецедента.

Связь расширения задается в том случае, если необходимо показать родственные отношения между двумя прецедентами. Один из них является базовым, а другой – его расширением. Базовый прецедент не зависит от расширяющих прецедентов и способен функционировать без них. С другой стороны, расширяющие прецеденты без базового прецедента функционировать не могут. Связи расширения изображают в виде стрелки со словом «extends» (расширение), которая имеет направление от базового прецедента к расширяемому.

Диаграмма прецедентов представлена на рис. 1

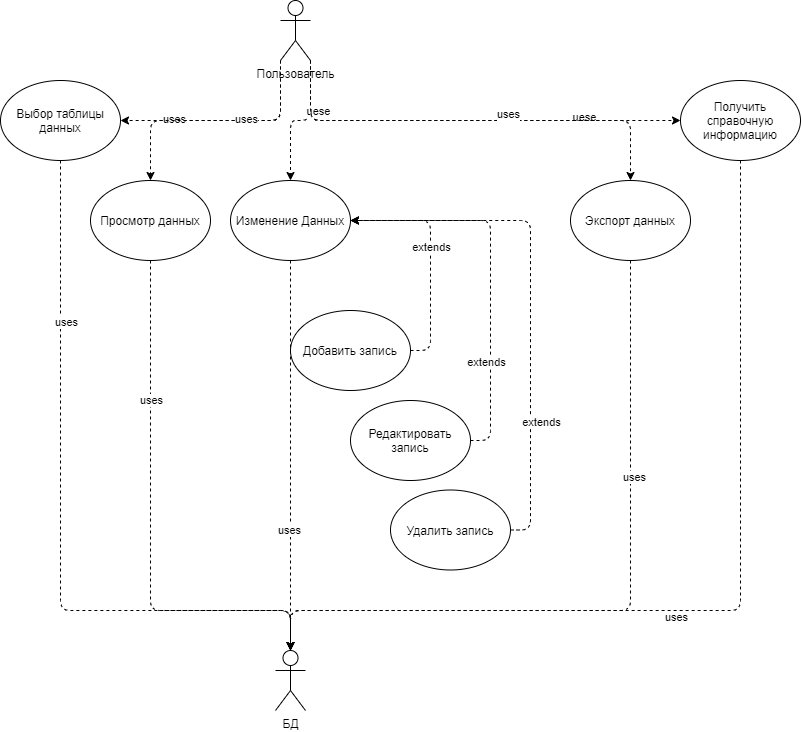


Рисунок 1. Диаграмма прецедентов

## Создание прототипа интерфейса пользователя

Интерфейс пользователя представлен в виде экранных форм и элементов управления. Их описание представлено в таблице 1.

Таблица 1. Экранные формы

|  |  |  |  |
| --- | --- | --- | --- |
| **Экранная форма** | **Элементы управления** | **Действия пользователя** | **Отклик системы** |
| Диспетчерская | Выпадающий список:  - «Выбор таблицы»  Кнопки:  - «Добавить запись»  - «Изменить запись»  - «Удалить запись»  - «Экспорт текущей таблицы»  - «Справка и нарушения»  Пункты выпадающего меню:  - Добавление/удаление нарушения для выбранной записи | Выбрать активную таблицу данных в выпадающем списке «Выбор таблицы» | Загрузка данных активной таблицы из БД в формате таблицы в ПК |
| Открыть окно «Добавление записи» (нажать кнопку «Добавить запись») | Открыть форму «Добавление записи» |
| Открыть окно «Редактирование записи» (нажать кнопку «Изменить запись») | Открыть форму «Редактирование записи» |
| Удаление выбранной(-ых) записи(-ей) (нажать кнопку «Удалить запись») | Открытие подтверждающего диалогового окна и последующее удаление записи из БД |
| Экспорт активной таблицы в формат PDF (нажать кнопку «Экспорт активной таблицы») | Экспорт таблицы БД в PDF, соответствующей активной таблице в ПК |
| Составление отчёта о маршрутах и выявленных нарушениях (нажать кнопку «Справка и нарушения») | Открытие формы «Справка и нарушения» |
| Добавление/удаление нарушения для выбранной записи (нажать кнопку выпадающего меню с соответствующим нарушением) | Фиксация нарушения или его исправления в БД |
| Добавление записи | Текстовые поля с информацией о характеристиках записи  Кнопка:  - «Добавить»  - «Составить расписание» (только для сущности расписание) | Заполнение текстовых полей, характеризующих добавляемый объект и нажатие кнопки «Добавить» | В случае корректных введённых данных, добавление новой записи в БД со значениями характеристик из заполненных текстовых полей |
| Редактирование записи | Текстовые поля с информацией о характеристиках записи  Кнопка:  - «Изменить»  - «Составить расписание» (только для сущности расписание) | Заполнение текстовых полей, характеризующих добавляемый объект и нажатие кнопки «Изменить» | В случае корректных введённых данных, изменение записи в БД с новыми значениями характеристик из заполненных текстовых полей.  Иначе выдача сообщения об ошибке |
| Составить расписание | Текстовые поля с информацией о недельном расписании  Кнопка:  - Составить | Заполнение текстовых полей с информацией о недельном расписании и нажатие кнопки «Составить» | В случае корректных введённых данных, формирование недельного расписания для маршрута из заполненных текстовых полей.  Иначе выдача сообщения об ошибке |
| Справка и нарушения | Кнопки:  - «Экспорт в HTML»  - «Обновить данные» | Экспорт справочной информации (нажатие кнопки «Экспорт в HTML») | Экспорт справочной информации в формате HTML |
| Обновление справочной информации, в соответствии с текущим состоянием БД (нажатие кнопки «Обновить данные») | Обновление справочной информации, в соответствии с текущим состоянием БД |

Примеры спроектированного пользовательского интерфейса представлены на рисунках 2, 3, 4, 5, 6, 7.

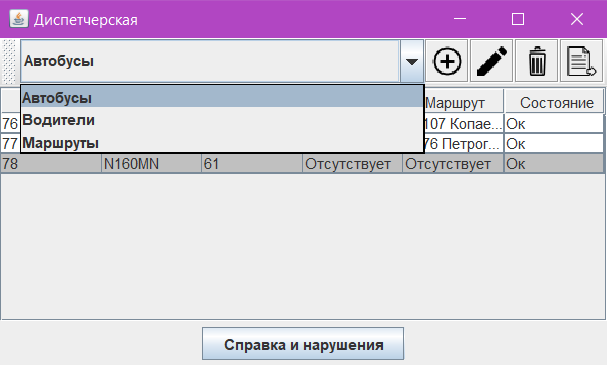


Рисунок 2. Диспетчерская

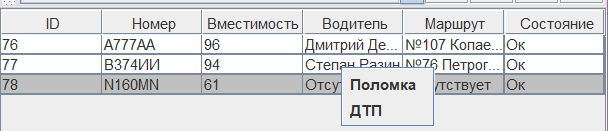


Рисунок 3. Выпадающее меню нарушений в форме Диспетчерская

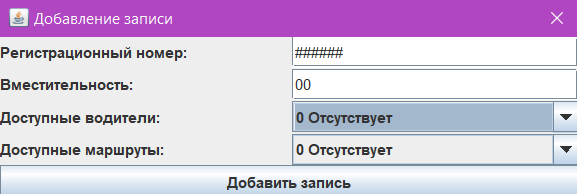


Рисунок 4. Добавление записи

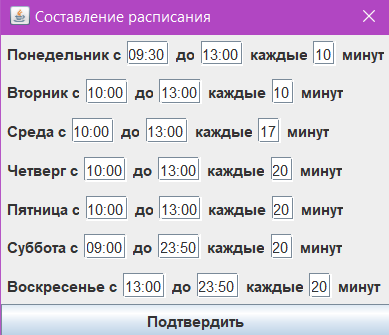


Рисунок 5. Составление расписания

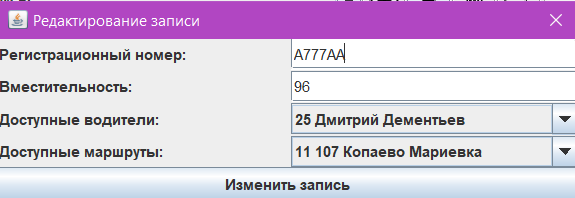


Рисунок 6. Редактирование записи

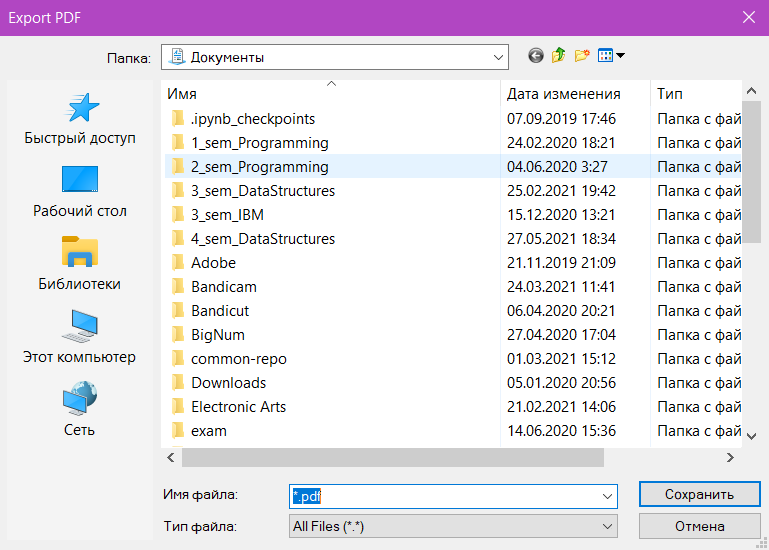


Рисунок 7. Экспорт в PDF

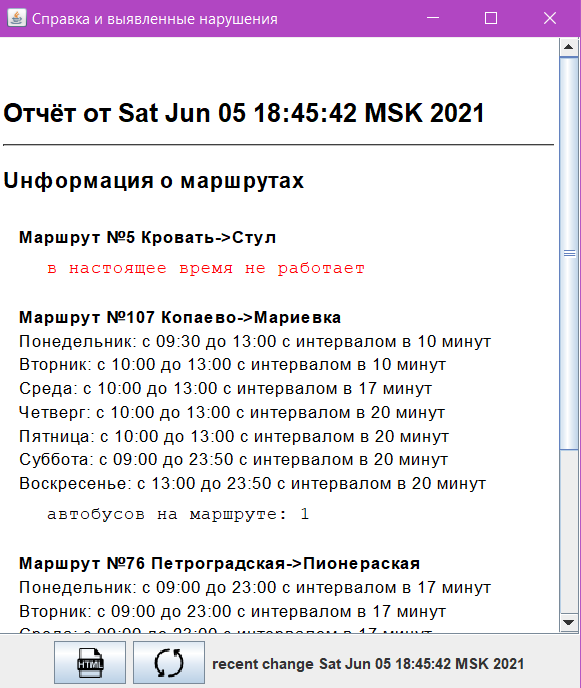


Рисунок 8. Справка и нарушения

## Построение диаграммы программных классов

Диаграмма классов (class diagram) строится на основе объектной модели. В описание класса указываются три раздела: имя класса, состав компонентов класса и методы класса. Графически класс изображается в виде прямоугольника. Соответствующая диаграмма представлена на рисунке 9.

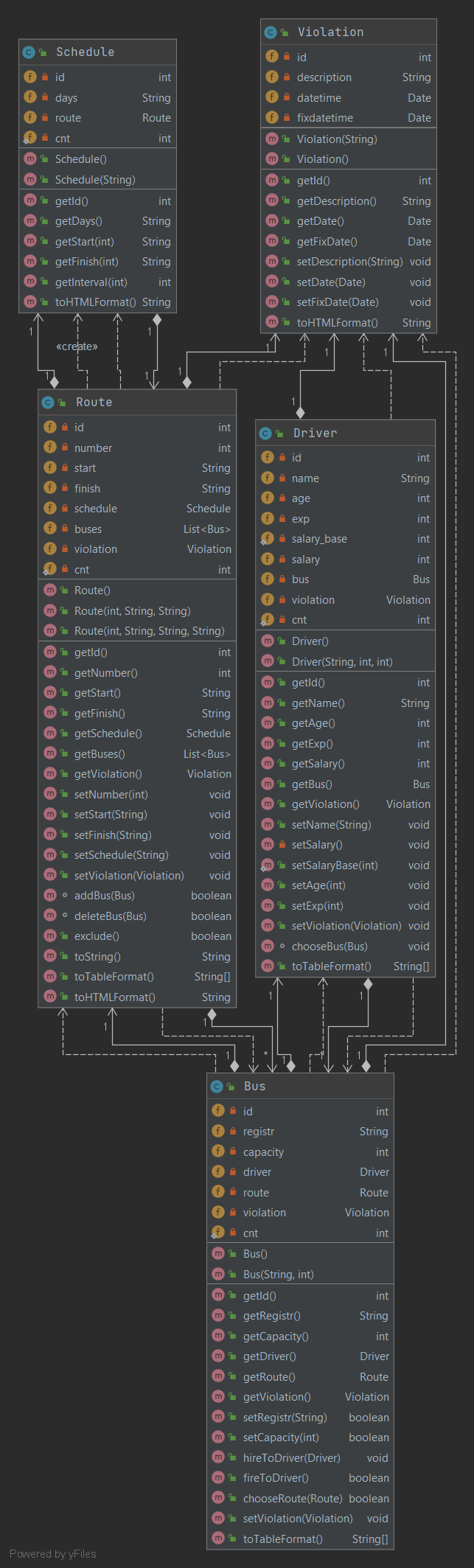


Рисунок 9. Диаграмма классов

# Описание классов

Описание классов будет приведено в формате Javadoc, также к пояснительной записке прилагается интерактивный html файл для более подробного ознакомления.

Bus.java

Класс, представляющий собой автобус

Описание конструкторов класса представлено на рисунке 10.0.

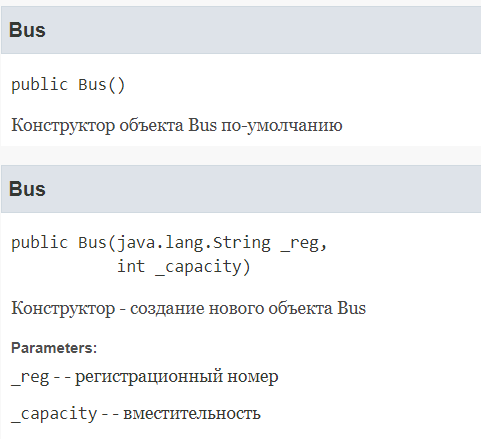


Рисунок 10.0. Конструкторы класса Bus

Описание полей класса представлено на рисунке 10.



Рисунок 10. Описание полей класса Bus

Описание полей класса представлено на рисунке 10.

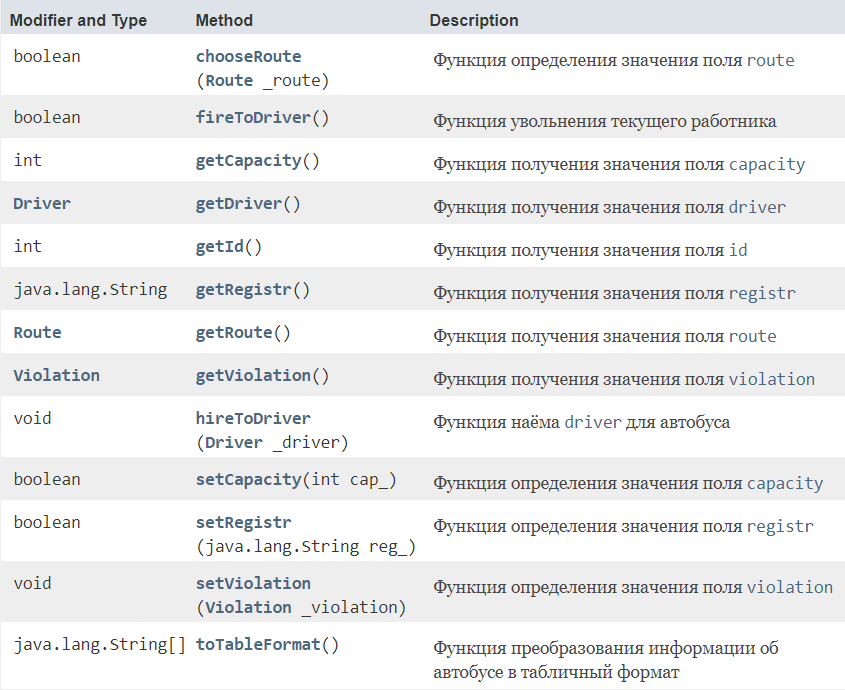


Рисунок 11. Описание методов класса Bus

Driver.java

Класс, представляющий собой водителя

Описание конструкторов класса представлено на рисунке 12.0.

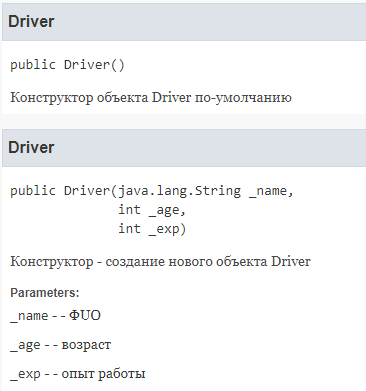


Рисунок 12.0. Конструкторы класса Driver

Описание полей класса представлено на рисунке 12.



Рисунок 12. Описание полей класса Driver

Описание полей класса представлено на рисунке 13.

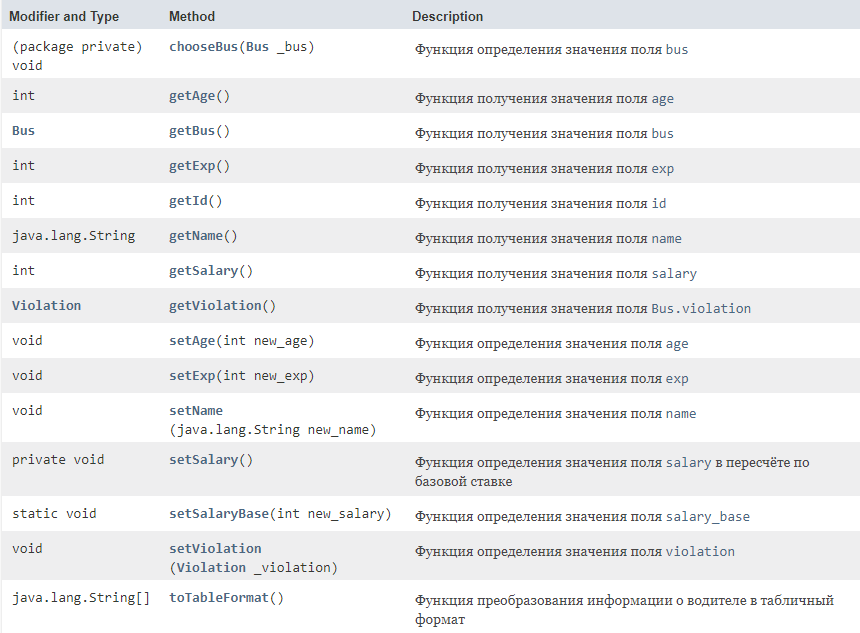


Рисунок 13. Описание методов класса Driver

Route.java

Класс, представляющий собой маршрут

Описание конструкторов класса представлено на рисунке 14.0.

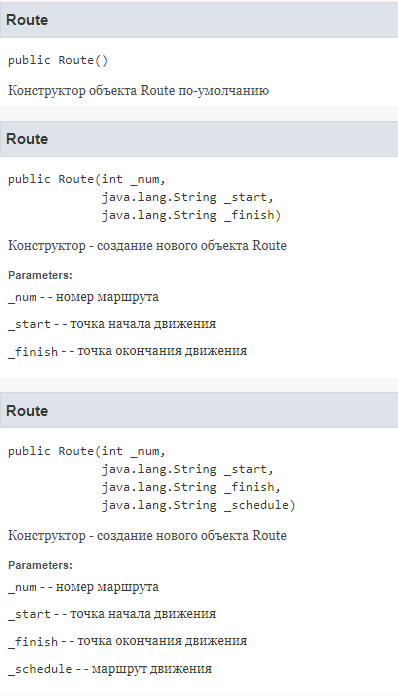


Рисунок 14.0. Конструкторы класса Route

Описание полей класса представлено на рисунке 14.



Рисунок 14. Описание полей класса Route

Описание полей класса представлено на рисунке 15.

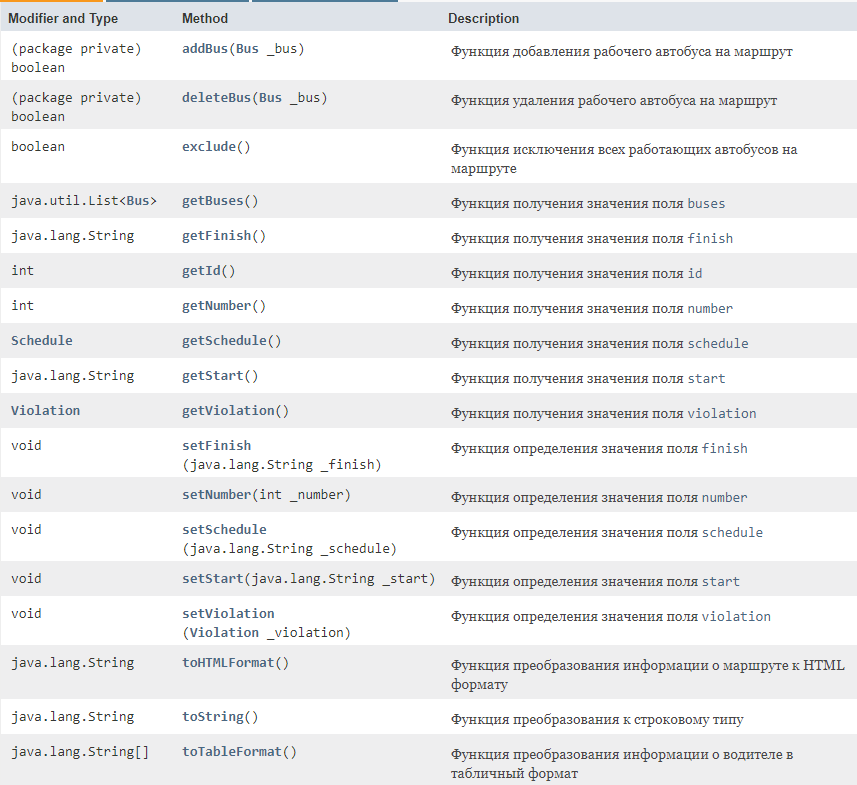


Рисунок 15. Описание методов класса Route

Schedule.java

Класс, представляющий собой расписание

Описание конструкторов класса представлено на рисунке 16.0.

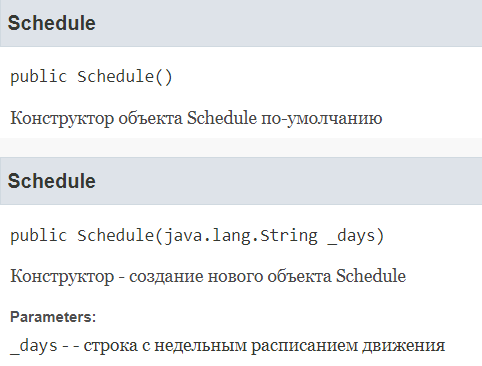


Рисунок 16.0. Конструкторы класса Schedule

Описание полей класса представлено на рисунке 16.



Рисунок 16. Описание полей класса Schedule

Описание полей класса представлено на рисунке 17.



Рисунок 17. Описание методов класса Schedule

Violation.java

Класс, представляющий собой нарушение

Описание конструкторов класса представлено на рисунке 18.0.

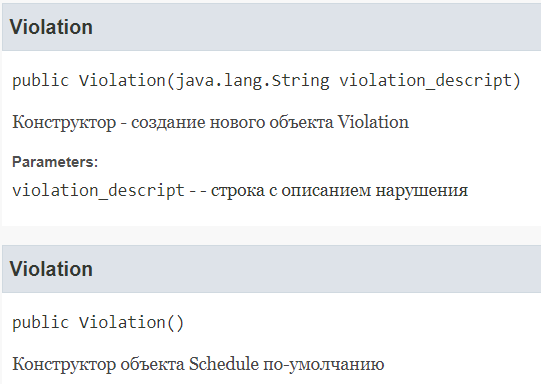


Рисунок 18.0. Конструкторы класса Violation

Описание полей класса представлено на рисунке 18.

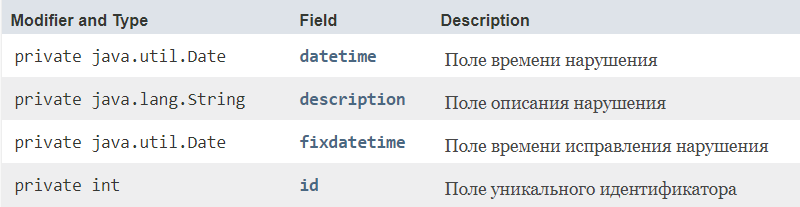


Рисунок 18. Описание полей класса Violation

Описание полей класса представлено на рисунке 19.

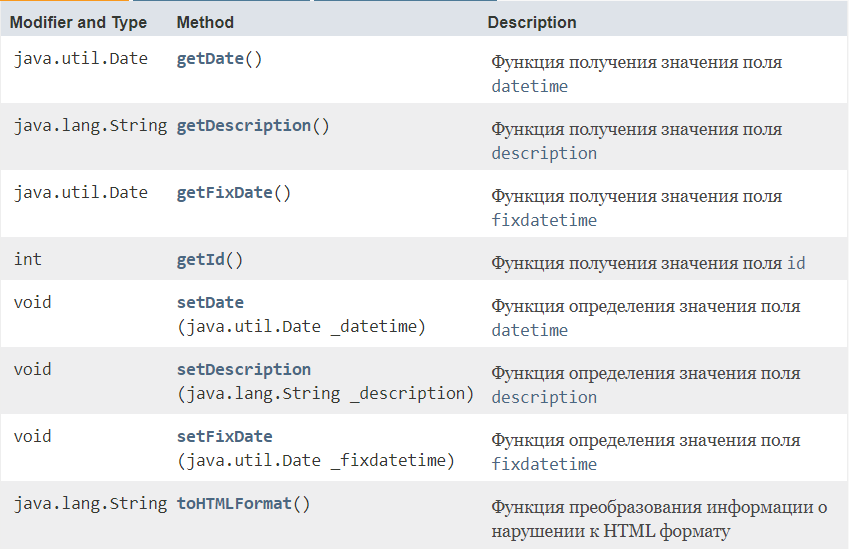


Рисунок 19. Описание методов класса Violation

App.java

Класс главного интерфейса ПК

Описание полей класса представлено на рисунке 20.

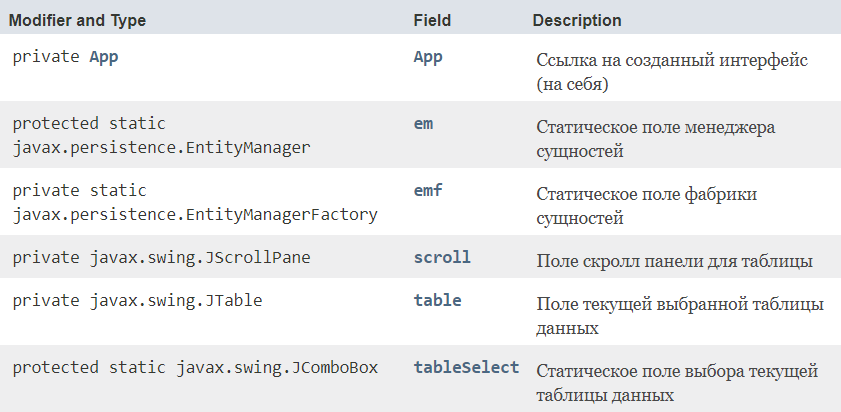


Рисунок 20. Описание полей класса App

Описание полей класса представлено на рисунке 21.

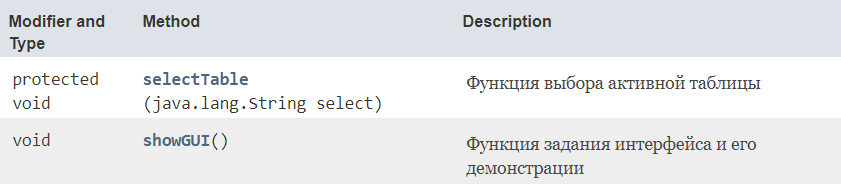


Рисунок 21. Описание методов класса App

CreateItemDialog.java

Класс окна для создания новых записей

Описание полей класса представлено на рисунке 22.

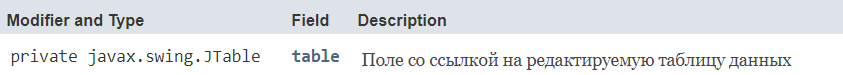


Рисунок 22. Описание полей класса CreateItemDialog

Описание полей класса представлено на рисунке 23.

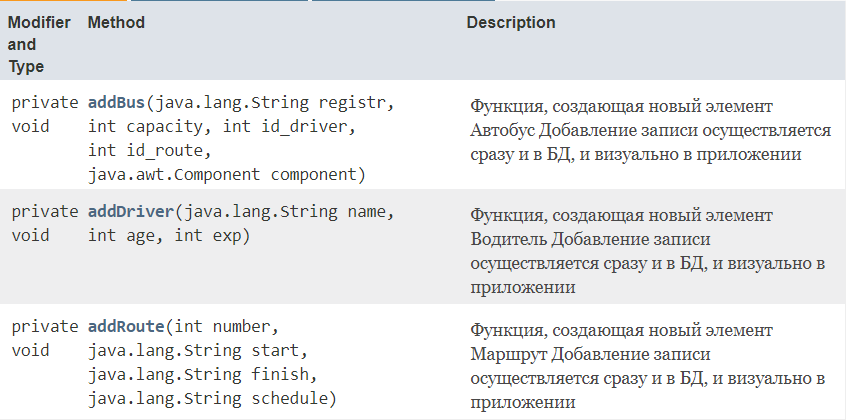


Рисунок 23. Описание методов класса CreateItemDialog

EditItemDialog.java

Класс окна для редактирования записей

Поля класса отсутствуют.

Описание методов класса представлено на рисунке 24.

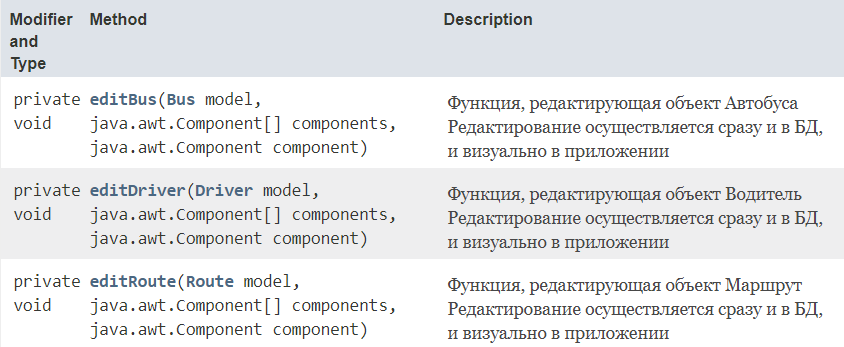


Рисунок 24. Описание методов класса EditItemDialog

EditPanel.java

Класс панели для редактирования объектов. Предназначена для добавления в уже созданное диалоговое окно parent.

Имеет только конструктор. Конструктор представлен на рисунке 25.

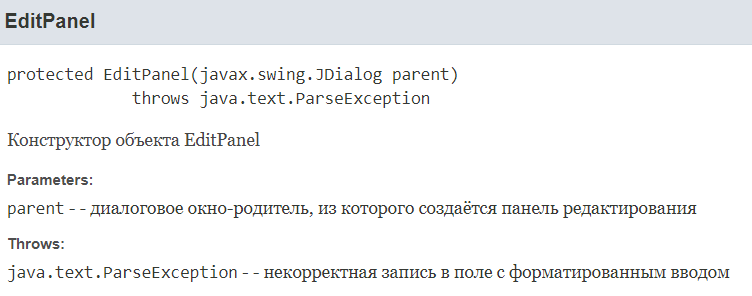


Рисунок 25. Конструктор класса EditPanel

HTMLExporter.java

Класс экспортера HTML

Имеет только конструктор. Конструктор представлен на рисунке 26.

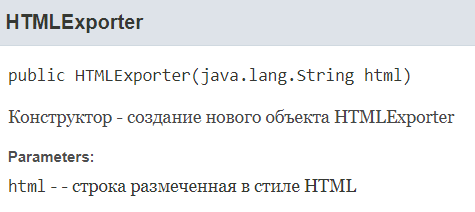


Рисунок 26. Конструктор класса HTMLExporter

PDFExporter.java

Класс экспортера PDF

Конструктор представлен на рисунке 27.1.

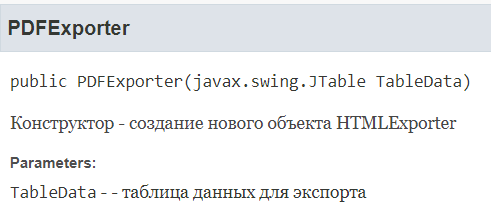


Рисунок 27. Конструктор класса PDFExporter

Конструктор представлен на рисунке 27.2.

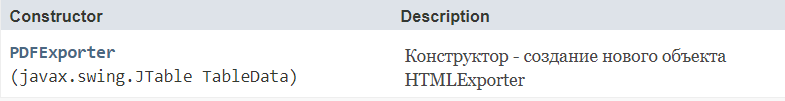


Рисунок 27. Методы класса PDFExporter

RemovalManager.java

Класс менеджера фиксации нарушений

Конструктор представлен на рисунке 28.

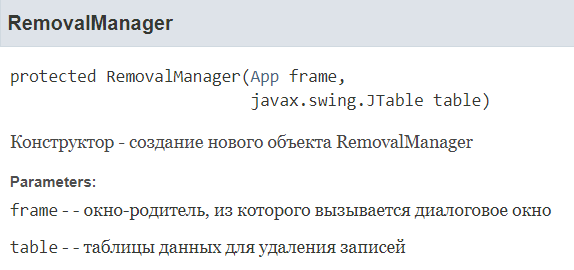


Рисунок 28. Конструктор класса RemovalManager

ReportThread.java

Класс потока для генерации отчёта по маршрутам и нарушениям

Описание полей класса представлено на рисунке 29.



Рисунок 29. Описание полей класса ReportThread

Описание полей класса представлено на рисунке 30.

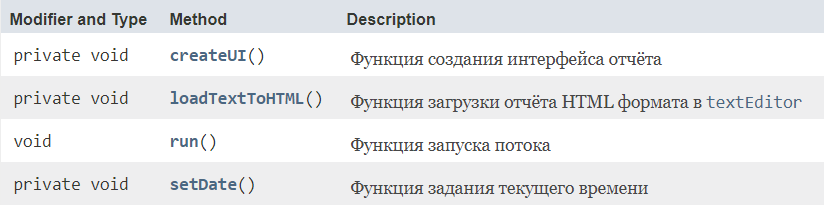


Рисунок 30. Описание методов класса ReportThread

ScheduleDialog.java

Класс диалогового окна для задания расписания

Имеет только конструктор. Конструктор представлен на рисунке 31.

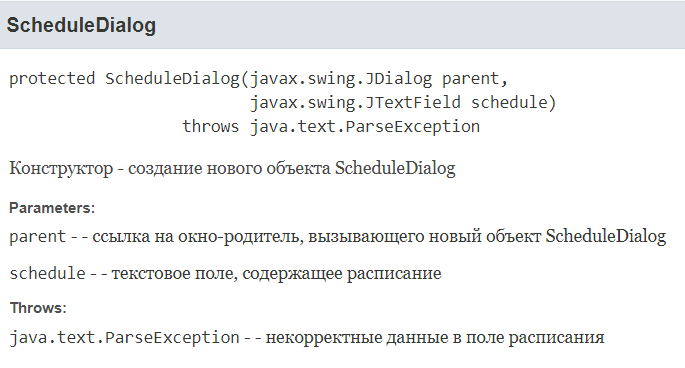


Рисунок 31. Конструктор класса ScheduleDialog

ScheduleViewer.java

Класс просмотра расписания маршрута

Имеет только конструктор. Конструктор представлен на рисунке 32.

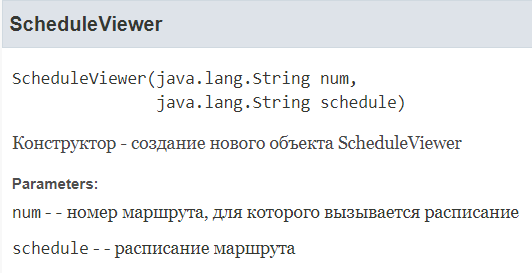


Рисунок 32. Конструктор класса ScheduleViewer

TablePopupMenu.java

Класс поп-ап меню для активной таблицы

Имеет только конструктор. Конструктор представлен на рисунке 33.

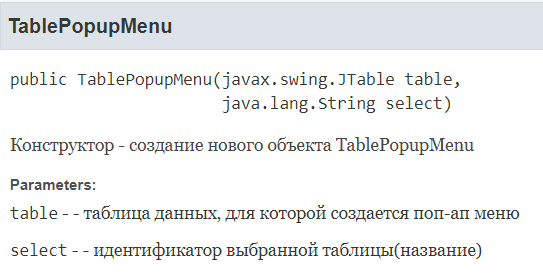


Рисунок 33. Конструктор класса TablePopupMenu

ViolationManager.java

Класс менеджера фиксации нарушений

Описание полей класса представлено на рисунке 34.

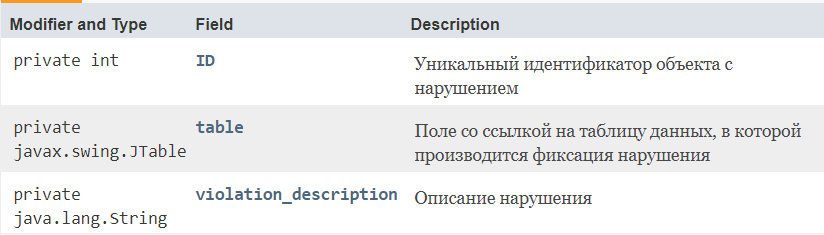


Рисунок 34. Описание полей класса ViolationManager

Описание полей класса представлено на рисунке 35.

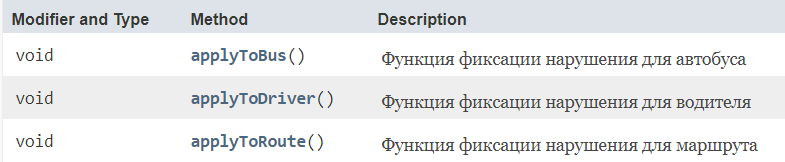


Рисунок 35. Описание методов класса ViolationManager

ViolationTableCellRender.java

Класс рендера таблицы данных

Описание полей класса представлено на рисунке 36.

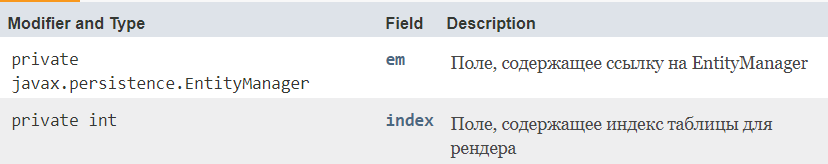


Рисунок 36. Описание полей класса ViolationTableCellRender

Описание полей класса представлено на рисунке 37.

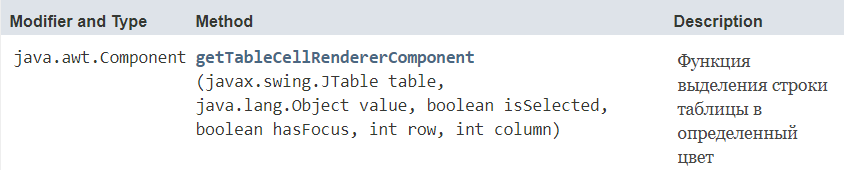


Рисунок 37. Описание методов класса ViolationTableCellRender

ValidationError.java

Класс ошибки валидности задаваемого значения

Описание полей класса представлено на рисунке 38.

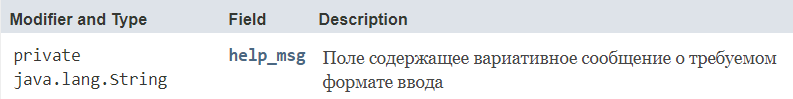


Рисунок 38. Описание полей класса ViolidationError

Описание полей класса представлено на рисунке 39.

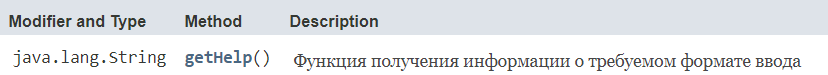


Рисунок 39. Описание методов класса ValidationError

WritingError.java

Класс ошибки добавления/редактирования записи

Имеет только конструктор. Описание конструктора приведено на рисунке 40.

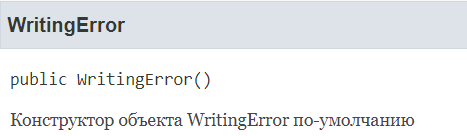


Рисунок 40. Конструктор класса WritingError

# Описание базы данных

В курсовом проекте была использована библиотека Hibernate для обращения к СУБД MySQL.

Была создана база данных controller, содержащей следующие таблицы:

* bus - хранит данных об автобусах
* driver - хранит данные о водителях
* route - хранит данные о маршрутах
* schedule – хранит данные о расписаниях
* violation – хранит данные о нарушениях

Структура таблиц в базе данных представлена на рисунках 41 и 42 соответственно.

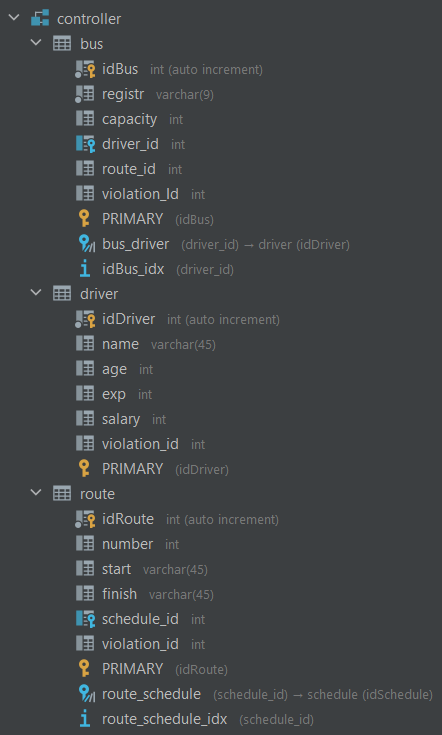


Рисунок 41. Структура БД

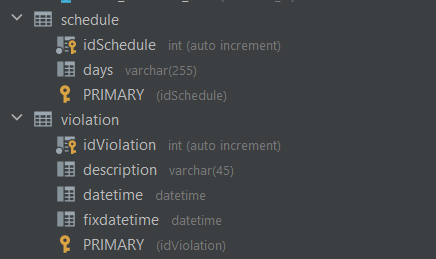


Рисунок 42. Структура БД (продолжение)

Листинг команд для создания базы данных приведён ниже.

CREATE DATABASE `controller` /\*!40100 DEFAULT CHARACTER SET utf8 \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;

CREATE TABLE `bus` (

`idBus` int NOT NULL AUTO\_INCREMENT,

`registr` varchar(9) NOT NULL,

`capacity` int DEFAULT NULL,

`driver\_id` int DEFAULT NULL,

`route\_id` int DEFAULT NULL,

`violation\_Id` int DEFAULT NULL,

PRIMARY KEY (`idBus`),

KEY `idBus\_idx` (`driver\_id`),

CONSTRAINT `bus\_driver` FOREIGN KEY (`driver\_id`) REFERENCES `driver` (`idDriver`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=85 DEFAULT CHARSET=utf8mb3;

CREATE TABLE `driver` (

`idDriver` int NOT NULL AUTO\_INCREMENT,

`name` varchar(45) DEFAULT NULL,

`age` int DEFAULT NULL,

`exp` int DEFAULT NULL,

`salary` int DEFAULT NULL,

`violation\_id` int DEFAULT NULL,

PRIMARY KEY (`idDriver`)

) ENGINE=InnoDB AUTO\_INCREMENT=30 DEFAULT CHARSET=utf8mb3;

CREATE TABLE `route` (

`idRoute` int NOT NULL AUTO\_INCREMENT,

`number` int DEFAULT NULL,

`start` varchar(45) DEFAULT NULL,

`finish` varchar(45) DEFAULT NULL,

`schedule\_id` int DEFAULT NULL,

`violation\_id` int DEFAULT NULL,

PRIMARY KEY (`idRoute`),

KEY `route\_schedule\_idx` (`schedule\_id`),

CONSTRAINT `route\_schedule` FOREIGN KEY (`schedule\_id`) REFERENCES `schedule` (`idSchedule`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=14 DEFAULT CHARSET=utf8mb3;

CREATE TABLE `schedule` (

`idSchedule` int NOT NULL AUTO\_INCREMENT,

`days` varchar(255) DEFAULT NULL,

PRIMARY KEY (`idSchedule`)

) ENGINE=InnoDB AUTO\_INCREMENT=20 DEFAULT CHARSET=utf8mb3;

CREATE TABLE `violation` (

`idViolation` int NOT NULL AUTO\_INCREMENT,

`description` varchar(45) DEFAULT NULL,

`datetime` datetime DEFAULT NULL,

`fixdatetime` datetime DEFAULT NULL,

PRIMARY KEY (`idViolation`)

) ENGINE=InnoDB AUTO\_INCREMENT=59 DEFAULT CHARSET=utf8mb3;

# Руководство оператору

## Назначение программы

ПК «Автобусная диспетчерская» предоставляет систему учета и администрированиях информации о функционировании автобусного парка.

В рамках ПК администратор может:

* добавлять, изменять или удалять:
  + информацию об автобусах;
  + информацию о водителях;
  + информацию о маршрутах;
  + информацию о расписаниях;
  + информацию о нарушениях
* получать отчёт о функционировании автобусного парка
* экспортировать данные, описанные в пунктах выше

## Описание задачи

ПК спроектирована для целей автоматизации отчётности по функционированию автобусного парка и ведению контроля текущей ситуации в работе маршрутов.

Диспетчер (администратор) может добавлять, редактировать и удалять любую информацию, хранимую в ПК. Также диспетчер может фиксировать любые нарушения, связанные с функционированием автобусного парка.

## Входные и выходные данные

Выходные данные должны быть представлены в виде таблицы, содержащей сведения всех записей выбранной таблицы с данными.

Ввод исходных данных должен осуществляться в режиме диалога. Вводимые данные являются значениями характеристик информационных объектов.

## Запуск программы

При запуске программы на экране появится окно – главное меню.

При запуске программы сразу происходит загрузка данных из БД для таблицы «Автобусы». Для загрузки остальных таблиц их требуется выбрать в выпадающем меню.

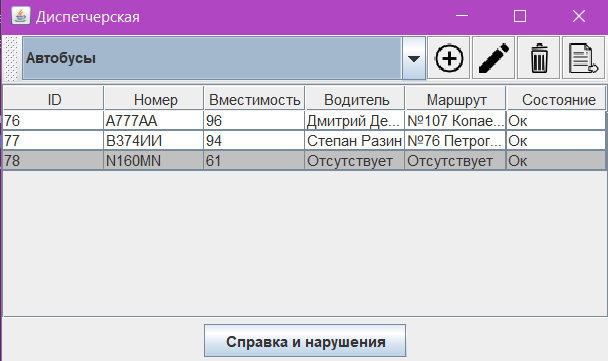


Рисунок 43. Главное меню Автобусной диспетчерской

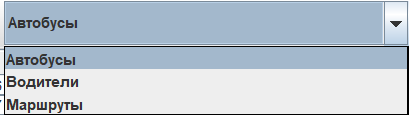


Рисунок 44. Выпадающее меню со списком имеющихся таблиц

Для всевозможных действий предназначены подсказки, активирующиеся при наведении на любой элемент управления.

Рассмотрим возможности работы с данными на примере таблицы Автобусы. Первой из рассматриваемых возможностей будет Добавление записи. Для этого требуется нажать на кнопку «Добавить запись» (иконка плюсика), после откроется диалоговое окно Добавления.

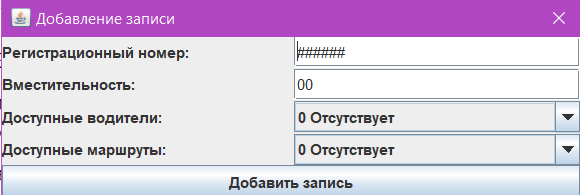


Рисунок 45. Диалоговое окно Добавления записи для Автобуса

Для избегания случайных или намеренных некорректных входных данных предусматривается маска сходных данных. Попробуем заполнить информационные поля.

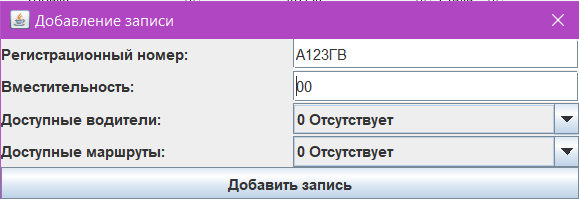


Рисунок 46. Заполненные информационные поля

При попытке добавить такую запись администратор получить сообщение об ошибке с описанием ожидаемых входных данных.

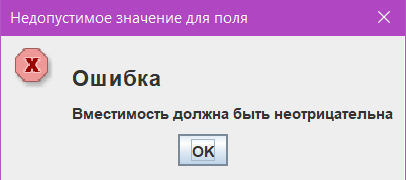
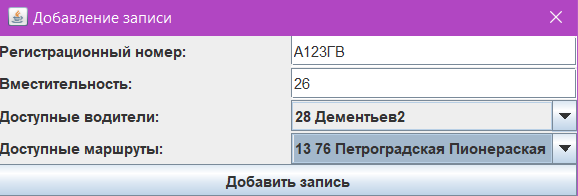


Рисунок 47. Сообщение о некорректных входных данных

Указав корректное значение указанного поля и выбрав из выпадающего списка интересующие пункты добавим запись.



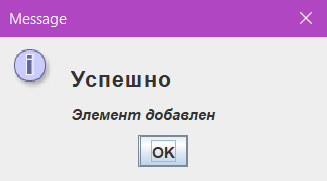


Рисунок 48. Корректные входные данные

Диалоговое окно Редактирования записей работает аналогично, интерфейс повторяет окно Добавления. Для редактирования записи требуется выбрать интересующую строку в таблице данных, иначе появится окно ошибки.

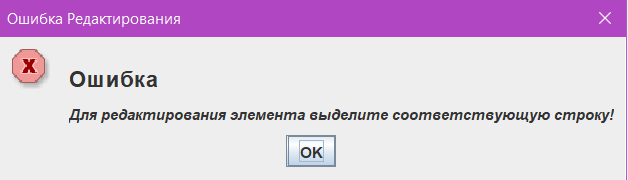


Рисунок 49. Ошибка редактирования

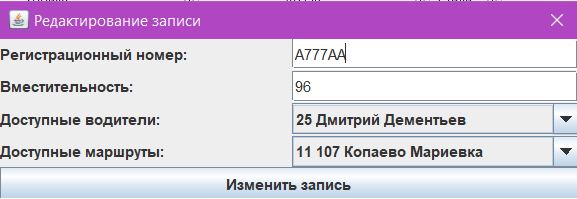


Рисунок 50. Диалоговое окно Редактирования

Для удаления элемента также требуется выбрать интересующую строку(-и) записи и нажать кнопку «Удалить запись» (иконка мусорного ведра), далее в появившемся диалоговом окне подтвердить действие.

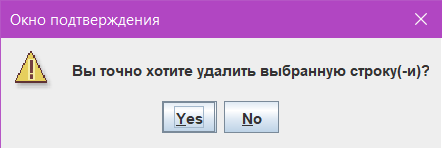


Рисунок 51. Подтверждение удаления

Для экспорта текущей таблицы данных требуется нажать кнопку «Экспорт в PDF» (иконка документа со стрелкой) и в открывшемся диалоговом окне выбрать путь для сохранения экспортированного файла.

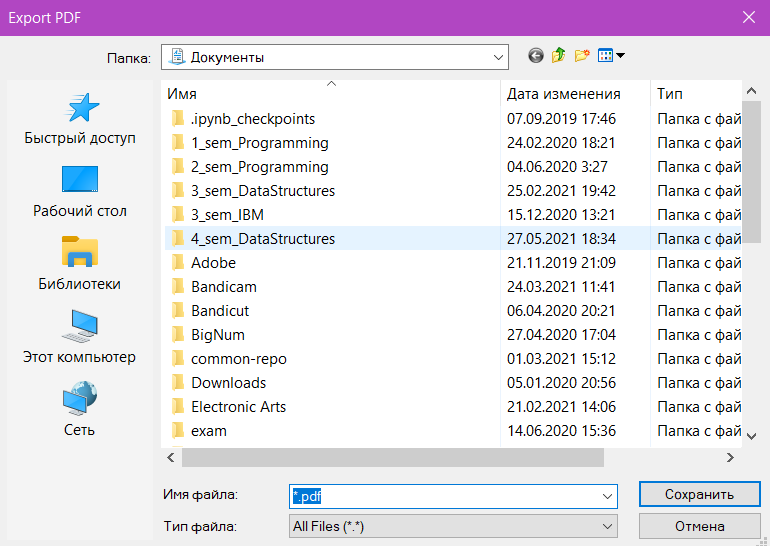


Рисунок 51. Диалоговое окно выбора директории для экспорта

Для генерации отчёта о функционировании Автобусного парка и выявленных нарушениях требуется нажать на кнопку «Справка и нарушения»

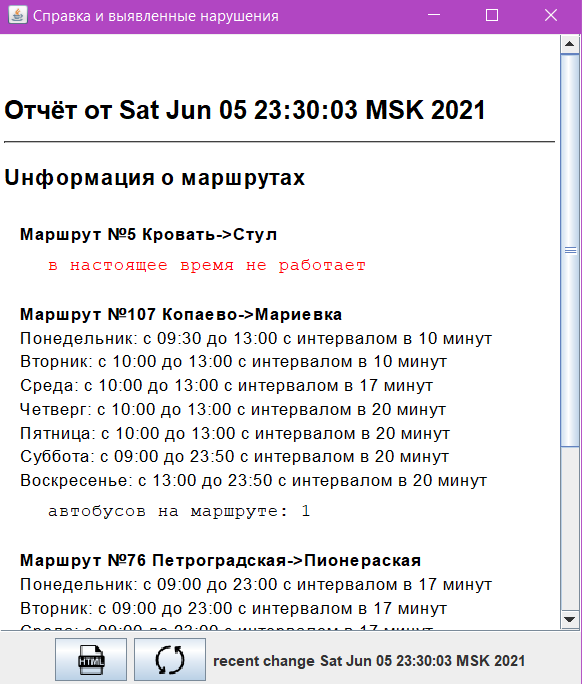


Рисунок 52. Окно сгенерированного отчёта

Окно может оставаться открытым во время действий в главном окне Диспетчерская, для обновления данных в актуальный момент требуется нажать кнопку «Обновить данные» (иконка стрелок). Также сгенерированный отчёт можно экспортировать по аналогии с экспортом PDF (выше по тексту).

# Заключение

В результате проделанной работы разработан ПК «Автобусная диспетчерская», предназначенный для администрирования и ведения отчётности Автобусного парка. В процессе проектирования ПК созданы прототип пользовательского графического интерфейса, диаграмма классов, руководство пользователю. Были закреплены теоретические знания, приобретены практические навыки по проектированию и разработке программного обеспечения на объектно-ориентированном языке Java.

# Приложение. Исходный текст программы

## Сущности

Bus.java

package entitys;  
  
import javax.persistence.\*;  
  
*/\*\*  
 \* Класс автобуса  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*@Entity // перевод класса к сущности БД  
@Table(name="controller.bus")  
public class Bus {  
 */\*\* Поле уникального идентификатора \*/* @Id  
 @Column(name="idBus")  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private int id;  
 */\*\* Поле регистрационного номера \*/* @Column(name="registr")  
 private String registr; // регистрационный номер  
 */\*\* Поле вместительности \*/* @Column(name="capacity")  
 private int capacity; // вместительность автобуса  
 */\*\* Поле со ссылкой на текущего водителя \*/* @OneToOne(optional = true, cascade = CascadeType.ALL)  
 @JoinColumn(name = "driver\_id", referencedColumnName = "idDriver")  
 private Driver driver; // ссылка на водителя  
 */\*\* Поле со ссылкой на текущий маршрут \*/* @ManyToOne(optional = true, cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "route\_id")  
 private Route route; // ссылка на маршрут  
 */\*\* Поле со ссылкой на имеющееся нарушение \*/* @OneToOne(optional = true, cascade = CascadeType.ALL)  
 @JoinColumn(name = "violation\_id")  
 private Violation violation; // нарушение  
 */\*\* Статическое поле счётчик автобусов \*/* private static int cnt = 0;  
 */\*\*  
 \* Конструктор объекта Bus по-умолчанию  
 \*/* public Bus() {  
 registr = "AAA" + cnt++;  
 capacity = 48;  
 driver = null;  
 route = null;  
 violation = null;  
 //System.out.println("create Bus №" + registr);  
 }  
 */\*\*  
 \* Конструктор - создание нового объекта Bus  
 \** ***@param*** *\_reg - регистрационный номер  
 \** ***@param*** *\_capacity - вместительность  
 \*/* public Bus(String \_reg, int \_capacity) {  
 cnt++;  
 registr = \_reg;  
 capacity = \_capacity;  
 driver = null;  
 route = null;  
 violation = null;  
 //System.out.println("create Bus №" + registr);  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Bus#id}  
 \** ***@return*** *возвращает уникальный идентификатор автобуса  
 \*/* public int getId() { return id; }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Bus#registr}  
 \** ***@return*** *возвращает регистрационный номер автобуса  
 \*/* public String getRegistr() { return registr; }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Bus#capacity}  
 \** ***@return*** *возвращает вместительность  
 \*/* public int getCapacity() { return capacity; }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Bus#driver}  
 \** ***@return*** *возвращает ссылку на водителя  
 \*/* public Driver getDriver() { return driver; }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Bus#route}  
 \** ***@return*** *возвращает ссылку на маршрут  
 \*/* public Route getRoute() { return route; }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Bus#violation}  
 \** ***@return*** *возвращает ссылку на нарушение  
 \*/* public Violation getViolation() { return violation; }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Bus#registr}  
 \** ***@param*** *reg\_ - новый регистрационный номер автобуса  
 \** ***@return*** *булево значение, означающее успешность изменения регистрационного номера  
 \*/* public boolean setRegistr(String reg\_) {  
 if (reg\_.length()>0 && reg\_.length()<7 && Character.isLetter(reg\_.charAt(0))) {  
 registr = reg\_; return true;  
 }  
 else return false;  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Bus#capacity}  
 \** ***@param*** *cap\_ - новая вместительность автобуса  
 \** ***@return*** *булево значение, означающее успешность изменения вместительность  
 \*/* public boolean setCapacity(int cap\_) {  
 if (cap\_ > 0) {  
 capacity = cap\_;  
 return true;  
 }  
 else return false;  
 }  
 */\*\*  
 \* Функция наёма {****@link*** *Bus#driver} для автобуса  
 \** ***@param*** *\_driver - ссылка на нанимаего водителя  
 \*/* public void hireToDriver(Driver \_driver) {  
 if (\_driver.getBus() != null) // если водитель работает на другом автобусе  
 \_driver.getBus().fireToDriver(); // то уволим его с того автобуса  
 if (driver != null) // если у автобуса уже есть водитель  
 this.fireToDriver();  
 driver = \_driver;  
 //System.out.println("hire " + driver.getName() + " on " + this.getRegistr()); // удалить потом  
 driver.chooseBus(this);  
 }  
 */\*\*  
 \* Функция увольнения текущего работника  
 \** ***@return*** *булево значение, означающее успешность увольнения водителя  
 \*/* public boolean fireToDriver() {  
 if (driver != null) {  
 driver.chooseBus(null);  
 //System.out.println("fire " + driver.getName() + " from " + this.getRegistr()); // удалить потом  
 driver = null;  
 if (route != null) {  
 route.deleteBus(this);  
 route = null;  
 }  
 return true;  
 }  
 else return false;  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Bus#route}  
 \** ***@param*** *\_route - ссылка на выбираемый маршрут  
 \** ***@return*** *булево значение, означающее успешность выбора нового маршрута  
 \*/* public boolean chooseRoute(Route \_route) {  
 if (\_route == null) {  
 //System.out.println("delete from path №" + route.getNumber() + " bus " + this.getRegistr()); // удалить потом  
 if (route != null) {  
 route.deleteBus(this);  
 }  
 route = null;  
 return true;  
 }  
 else {  
 if (driver != null) { // если у автобуса нет водителя  
 if (route != null) { // если автобус на другом маршруте  
 route.deleteBus(this);  
 }  
 route = \_route;  
 //System.out.println("choosen path №" + route.getNumber() + " for " + this.getRegistr()); // удалить потом  
 return route.addBus(this);  
 }  
 return false;  
 }  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Bus#violation}  
 \** ***@param*** *\_violation - ссылка на выявленное нарушение  
 \*/* public void setViolation(Violation \_violation) {  
 violation = \_violation;  
 // если замечено нарушение  
 if (violation != null) {  
 if (driver != null) {  
 fireToDriver();  
 }  
 if (route != null) {  
 chooseRoute(null);  
 }  
 }  
 }  
 */\*\*  
 \* Функция преобразования информации об автобусе в табличный формат  
 \** ***@return*** *список строк со значениями всех информационных полей автобуса  
 \*/* public String[] toTableFormat() {  
 String[] res = new String[] {String.valueOf(id), registr, String.valueOf(capacity), "Отсутствует", "Отсутствует", "Ок"};;  
 if (driver != null) res[3] = driver.getName();  
 if (route != null) res[4] = "№" + route.getNumber() + " " + route.getStart() + " - " + route.getFinish();  
 if (violation != null) res[5] = violation.getDescription();  
 return res;  
 }  
}

Driver.java

package entitys;  
  
import java.util.ArrayList;  
import javax.persistence.\*;  
  
*/\*\*  
 \* Класс водителя  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*@Entity // перевод класса к сущности БД  
@Table(name="controller.driver")  
public class Driver {  
 */\*\* Поле уникального идентификатора \*/* @Id  
 @Column(name="idDriver")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
 */\*\* Поле имени водителя \*/* @Column(name="name")  
 private String name;  
 */\*\* Поле возраста идентификатора \*/* @Column(name="age")  
 private int age;  
 */\*\* Поле опыта работы водителя \*/* @Column(name="exp")  
 private int exp;  
 */\*\* Статичное поле с базовой зарплатой \*/* private static int *salary\_base* = 21000; // базовая ставка для всех водителей  
 */\*\* Поле зарплаты водителя \*/* @Column(name="salary")  
 private int salary;  
 */\*\* Поле со ссылкой на рабочий автобус данного водителя \*/* @OneToOne(optional = true, mappedBy = "driver")  
 private Bus bus; // какой автобус водит  
 */\*\* Поле со ссылкой на имеющееся нарушение \*/* @OneToOne(optional = true, cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "violation\_id")  
 private Violation violation; // нарушение  
 */\*\* Статическое поле счётчик автобусов \*/* private static int *cnt* = 1;  
 */\*\*  
 \* Конструктор объекта Driver по-умолчанию  
 \*/* public Driver() {  
 name = "Anonymous\_" + *cnt*++;  
 age = 21;  
 exp = 0;  
 setSalary();  
 bus = null;  
 violation = null;  
 //System.out.println("create Driver " + name);  
 }  
 */\*\*  
 \* Конструктор - создание нового объекта Driver  
 \** ***@param*** *\_name - ФUО  
 \** ***@param*** *\_age - возраст  
 \** ***@param*** *\_exp - опыт работы  
 \*/* public Driver(String \_name, int \_age, int \_exp) {  
 *cnt*++;  
 name = \_name;  
 age = \_age;  
 exp = \_exp;  
 setSalary();  
 bus = null;  
 violation = null;  
 //System.out.println("create Driver " + name);  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Driver#id}  
 \** ***@return*** *возвращает уникальный идентификатор водителя  
 \*/* public int getId() { return id; }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Driver#name}  
 \** ***@return*** *возвращает ФUО водителя  
 \*/* public String getName() {  
 return name;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Driver#age}  
 \** ***@return*** *возвращает возраст водителя  
 \*/* public int getAge() {  
 return age;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Driver#exp}  
 \** ***@return*** *возвращает опыт работы водителя  
 \*/* public int getExp() {  
 return exp;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Driver#salary}  
 \** ***@return*** *возвращает зарплату водителя  
 \*/* public int getSalary() {  
 return salary;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Driver#bus}  
 \** ***@return*** *возвращает ссылку на рабочий автобус водителя  
 \*/* public Bus getBus() {  
 return bus;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Bus#violation}  
 \** ***@return*** *возвращает ссылку на нарушение  
 \*/* public Violation getViolation() { return violation; }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Driver#name}  
 \** ***@param*** *new\_name - новое ФUО водителя  
 \*/* public void setName(String new\_name) { name = new\_name; }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Driver#salary} в пересчёте по базовой ставке  
 \*/* private void setSalary() { if (exp<10) salary=*salary\_base*\*(10+exp)/10; else salary=*salary\_base*\*2; }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Driver#salary\_base}  
 \** ***@param*** *new\_salary - новая базовая ставка по зарплате  
 \*/* public static void setSalaryBase(int new\_salary) {  
 *salary\_base* = new\_salary;  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Driver#age}  
 \** ***@param*** *new\_age - новый возраст водителя  
 \*/* public void setAge(int new\_age) {  
 age = new\_age;  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Driver#exp}  
 \** ***@param*** *new\_exp - новый опыт работы водителя  
 \*/* public void setExp(int new\_exp) {  
 exp = new\_exp;  
 setSalary();  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Driver#violation}  
 \** ***@param*** *\_violation - ссылка на выявленное нарушение  
 \*/* public void setViolation(Violation \_violation) {  
 violation = \_violation;  
 // если замечено нарушение  
 if (violation != null) {  
 if (bus != null) {  
 bus.fireToDriver();  
 }  
 }  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Driver#bus}  
 \** ***@param*** *\_bus - ссылка на автобус  
 \*/* void chooseBus(Bus \_bus) {  
 bus = \_bus;  
 }  
 */\*\*  
 \* Функция преобразования информации о водителе в табличный формат  
 \** ***@return*** *список строк со значениями всех информационных полей водителя  
 \*/* public String[] toTableFormat() {  
 String[] res = new String[] {String.*valueOf*(id), name, String.*valueOf*(age), String.*valueOf*(exp), String.*valueOf*(salary), "Ок"};  
 if (violation != null) res[5] = violation.getDescription();  
 return res;  
 }  
}

Route.java

package entitys;  
  
import java.util.\*;  
import javax.persistence.\*;  
  
*/\*\*  
 \* Класс маршрута  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*@Entity // перевод класса к сущности БД  
@Table(name="controller.route")  
public class Route {  
 */\*\* Поле уникального идентификатора \*/* @Id  
 @Column(name="idRoute")  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private int id;  
 */\*\* Поле номера маршрута \*/* @Column(name="number")  
 private int number; // номер маршрута  
 */\*\* Поле локации начала движения на маршруте \*/* @Column(name="start")  
 private String start; // начало маршрута  
 */\*\* Поле локации окончания движения на маршруте \*/* @Column(name="finish")  
 private String finish; // конец маршрута  
 */\*\* Поле со ссылкой на текущее расписание движения маршрута \*/* @OneToOne(optional = false, cascade = CascadeType.ALL)  
 @JoinColumn(name = "schedule\_id", referencedColumnName = "idSchedule")  
 private Schedule schedule; // расписание движения по дням недели  
 */\*\* Поле массива ссылок на все автобусы, работающие на маршруте \*/* @OneToMany(mappedBy = "route", fetch = FetchType.EAGER, orphanRemoval = false)  
 private List<Bus> buses; // список автобусов на маршруте  
 */\*\* Поле со ссылкой на имеющееся нарушение \*/* @OneToOne(optional = true, cascade = CascadeType.ALL)  
 @JoinColumn(name = "violation\_id")  
 private Violation violation; // нарушение  
 */\*\* Статическое поле счётчик автобусов \*/* private static int cnt = 1;  
 */\*\*  
 \* Конструктор объекта Route по-умолчанию  
 \*/* public Route() {  
 number = cnt++;  
 start = "start";  
 finish = "finish";  
 buses = new ArrayList<>();  
 schedule = new Schedule();  
 violation = null;  
 //System.out.println("create Route №" + number + " " + start + "->" + finish);  
 }  
 */\*\*  
 \* Конструктор - создание нового объекта Route  
 \** ***@param*** *\_num - номер маршрута  
 \** ***@param*** *\_start - точка начала движения  
 \** ***@param*** *\_finish - точка окончания движения  
 \*/* public Route(int \_num, String \_start, String \_finish) {  
 cnt++;  
 number = \_num;  
 start = \_start;  
 finish = \_finish;  
 buses = new ArrayList<>();  
 schedule = new Schedule();  
 violation = null;  
 //System.out.println("create Route №" + number + " " + start + "->" + finish);  
 }  
 */\*\*  
 \* Конструктор - создание нового объекта Route  
 \** ***@param*** *\_num - номер маршрута  
 \** ***@param*** *\_start - точка начала движения  
 \** ***@param*** *\_finish - точка окончания движения  
 \** ***@param*** *\_schedule - маршрут движения  
 \*/* public Route(int \_num, String \_start, String \_finish, String \_schedule) {  
 cnt++;  
 number = \_num;  
 start = \_start;  
 finish = \_finish;  
 buses = new ArrayList<>();  
 schedule = new Schedule(\_schedule);  
 violation = null;  
 //System.out.println("create Route №" + number + " " + start + "->" + finish);  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Route#id}  
 \** ***@return*** *возвращает уникальный идентификатор маршрута  
 \*/* public int getId() { return id; }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Route#number}  
 \** ***@return*** *возвращает номер маршрута  
 \*/* public int getNumber() {  
 return number;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Route#start}  
 \** ***@return*** *возвращает точку начала движения  
 \*/* public String getStart() {  
 return start;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Route#finish}  
 \** ***@return*** *возвращает точку окончания движения  
 \*/* public String getFinish() {  
 return finish;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Route#schedule}  
 \** ***@return*** *возвращает ссылку на текущее расписание  
 \*/* public Schedule getSchedule() { return schedule; }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Route#buses}  
 \** ***@return*** *возвращает список ссылок на все автобусы на маршруте  
 \*/* public List<Bus> getBuses() {  
 return buses;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Route#violation}  
 \** ***@return*** *возвращает ссылку на нарушение  
 \*/* public Violation getViolation() { return violation; }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Route#number}  
 \** ***@param*** *\_number - новый номер маршрута  
 \*/* public void setNumber(int \_number) { number = \_number; }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Route#start}  
 \** ***@param*** *\_start - новая точка начала движения  
 \*/* public void setStart(String \_start) { start = \_start; }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Route#finish}  
 \** ***@param*** *\_finish- новая точка окончания движения  
 \*/* public void setFinish(String \_finish) { finish = \_finish; }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Route#schedule}  
 \** ***@param*** *\_schedule- ссылка на новое расписание движения  
 \*/* public void setSchedule(String \_schedule) {  
 schedule = new Schedule(\_schedule);  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Route#violation}  
 \** ***@param*** *\_violation - ссылка на выявленное нарушение  
 \*/* public void setViolation(Violation \_violation) {  
 violation = \_violation;  
 // если замечено нарушение  
 if (violation != null) {  
 exclude();  
 }  
 }  
 */\*\*  
 \* Функция добавления рабочего автобуса на маршрут  
 \** ***@param*** *\_bus - ссылка на добавляемый автобус  
 \** ***@return*** *булево значение, определяющее успешность добавления автобуса на маршрут  
 \*/* boolean addBus(Bus \_bus) {  
 if (\_bus != null) {  
 buses.add(\_bus);  
 //System.out.println("add " + \_bus.getRegistr() + " for path №" + this.getNumber()); // удалить потом  
 return true;  
 }  
 else return false;  
 }  
 */\*\*  
 \* Функция удаления рабочего автобуса на маршрут  
 \** ***@param*** *\_bus - ссылка на удаляемый автобус с маршрута  
 \** ***@return*** *булево значение, определяющее успешность удаления автобуса с маршрута  
 \*/* boolean deleteBus(Bus \_bus) {  
 if (\_bus != null) {  
 //System.out.println("delete " + \_bus.getRegistr() + " from path №" + this.getNumber()); // удалить потом  
 return buses.remove(\_bus);  
 }  
 else return false;  
 }  
 */\*\*  
 \* Функция исключения всех работающих автобусов на маршруте  
 \** ***@return*** *булево значение, определяющее успешность исключения автобусов с маршрута  
 \*/* public boolean exclude() {  
 if (buses.size() > 0) {  
 while (buses.size() != 0) {  
 buses.get(0).chooseRoute(null);  
 }  
 //System.out.println("exclude path №" + this.getNumber() + " from work"); // удалить потом  
 return true;  
 }  
 else return false;  
 }  
 */\*\*  
 \* Функция преобразования к строковому типу  
 \** ***@return*** *строка, содержащая всю информацию о маршруте  
 \*/* public String toString() {  
 return Integer.toString(id) + " " + Integer.toString(number) + " " + start + " " + finish;  
 }  
 */\*\*  
 \* Функция преобразования информации о водителе в табличный формат  
 \** ***@return*** *список строк со значениями всех информационных полей маршрута  
 \*/* public String[] toTableFormat() {  
 String[] res = new String[] {String.valueOf(id), String.valueOf(number), start, finish, schedule.getDays(), "Ок"};  
 if (violation != null) res[5] = violation.getDescription();  
 return res;  
 }  
 */\*\*  
 \* Функция преобразования информации о маршруте к HTML формату  
 \** ***@return*** *строка содержащая всю информацию о маршруте с разметкой формата HTML  
 \*/* public String toHTMLFormat() {  
 StringBuffer buff = new StringBuffer();  
 buff.append("<span><b>" + "Маршрут №" + getNumber() + ' ' + getStart() + "->" + getFinish() + "</span><br>");  
 if (buses.size() != 0) {  
 buff.append(schedule.toHTMLFormat());  
 buff.append("<pre> автобусов на маршруте: " + buses.size() + "</pre>");  
 }  
 else  
 buff.append("<pre style=\"color:red\"> в настоящее время не работает" + "</pre>");  
 return buff.toString();  
 }  
}

Schedule.java

package entitys;  
  
import javax.persistence.\*;  
  
*/\*\*  
 \* Класс расписания  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*@Entity // перевод класса к сущности БД  
@Table(name="schedule")  
public class Schedule {  
 */\*\* Поле уникального идентификатора \*/* @Id  
 @Column(name="idSchedule")  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private int id;  
 */\*\* Поле со строкой недельного расписания движения \*/* @Column(name="days")  
 private String days;  
 */\*\* Поле со ссылкой на маршрут \*/* @OneToOne(optional = true, mappedBy = "schedule", orphanRemoval = true)  
 private Route route;  
 */\*\* Статическое поле счётчик расписаний \*/* private static int cnt = 1;  
 */\*\*  
 \* Конструктор объекта Schedule по-умолчанию  
 \*/* public Schedule() {  
 days = new String();  
 for (int i=0; i<7; ++i) {  
 days += "9:00 21:00 " + cnt++ + ";";  
 }  
 //System.out.println("create Schedule №" + id);  
 }  
 */\*\*  
 \* Конструктор - создание нового объекта Schedule  
 \** ***@param*** *\_days - строка с недельным расписанием движения  
 \*/* public Schedule(String \_days) {  
 days = \_days;  
 //System.out.println("create Schedule №" + id);  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Schedule#id}  
 \** ***@return*** *уникальный идентификатор расписания  
 \*/* public int getId() { return id; }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Schedule#days}  
 \** ***@return*** *строка недельного расписания  
 \*/* public String getDays() { return days; }  
 */\*\*  
 \* Функция получения значения времени начала движения в определенный день  
 \** ***@param*** *day - порядковый номер дня недели  
 \** ***@return*** *строка с временем начала движения  
 \*/* public String getStart(int day) { return days.split(";")[day].split(" ")[0]; }  
 */\*\*  
 \* Функция получения значения времени окончания движения в определенный день  
 \** ***@param*** *day - порядковый номер дня недели  
 \** ***@return*** *строка с временем окончания движения  
 \*/* public String getFinish(int day) {  
 return days.split(";")[day].split(" ")[1];  
 }  
 */\*\*  
 \* Функция получения значения интервала движения в определенный день  
 \** ***@param*** *day - порядковый номер дня недели  
 \** ***@return*** *строка с интервалом движения  
 \*/* public int getInterval(int day) { return Integer.parseInt(days.split(";")[day].split(" ")[2]); }  
 */\*\*  
 \* Функция преобразования информации о расписании к HTML формату  
 \** ***@return*** *строка содержащая всю информацию о расписании с разметкой формата HTML  
 \*/* public String toHTMLFormat() {  
 StringBuffer buff = new StringBuffer();  
 buff.append("<span>");  
 for (int i=0; i<7; ++i) {  
 buff.append(Days.values()[i] + ": с " + getStart(i) + " до "  
 + getFinish(i) + " с интервалом в " + getInterval(i) + " минут<br>");  
 }  
 buff.append("</span>");  
 return buff.toString();  
 }  
}

Violation.java

package entitys;  
  
import javax.persistence.\*;  
import java.util.Date;  
  
*/\*\*  
 \* Класс нарушения  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*@Entity // перевод класса к сущности БД  
@Table(name="violation")  
public class Violation {  
 */\*\* Поле уникального идентификатора \*/* @Id  
 @Column(name="idViolation")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
 */\*\* Поле описания нарушения \*/* @Column(name="description")  
 private String description; // регистрационный номер  
 */\*\* Поле времени нарушения \*/* @Column(name="datetime")  
 private Date datetime; // время нарушения  
 */\*\* Поле времени исправления нарушения \*/* @Column(name="fixdatetime")  
 private Date fixdatetime; // время нарушения  
 */\*\*  
 \* Конструктор - создание нового объекта Violation  
 \** ***@param*** *violation\_descript - строка с описанием нарушения  
 \*/* public Violation(String violation\_descript) {  
 description = violation\_descript;  
 datetime = new Date();  
 //System.out.println("Проблемка " + ": " + description);  
 }  
 */\*\*  
 \* Конструктор объекта Schedule по-умолчанию  
 \*/* public Violation() {  
 description = "Нераспознанная ошибка";  
 datetime = new Date();  
 //System.out.println("Проблемка " + ": " + description);  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Violation#id}  
 \** ***@return*** *уникальный идентификатор нарушения  
 \*/* public int getId() {  
 return id;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Violation#description}  
 \** ***@return*** *описание нарушения  
 \*/* public String getDescription() {  
 return description;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Violation#datetime}  
 \** ***@return*** *время нарушения  
 \*/* public Date getDate() {  
 return datetime;  
 }  
 */\*\*  
 \* Функция получения значения поля {****@link*** *Violation#fixdatetime}  
 \** ***@return*** *время исправления нарушения  
 \*/* public Date getFixDate() {  
 return fixdatetime;  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Violation#description}  
 \** ***@param*** *\_description - новое описание нарушения  
 \*/* public void setDescription(String \_description) {  
 description = \_description;  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Violation#datetime}  
 \** ***@param*** *\_datetime - новое время нарушения  
 \*/* public void setDate(Date \_datetime) {  
 datetime = \_datetime;  
 }  
 */\*\*  
 \* Функция определения значения поля {****@link*** *Violation#fixdatetime}  
 \** ***@param*** *\_fixdatetime - новое время исправления нарушения  
 \*/* public void setFixDate(Date \_fixdatetime) {  
 fixdatetime = \_fixdatetime;  
 }  
 */\*\*  
 \* Функция преобразования информации о нарушении к HTML формату  
 \** ***@return*** *строка содержащая всю информацию о нарушении с разметкой формата HTML  
 \*/* public String toHTMLFormat() {  
 StringBuffer buff = new StringBuffer();  
 String date = "<a style=\"color:red\">["+getDate().toString()+"]</a>";  
 buff.append("<p>" + date + " " + getDescription() + "</p>");  
 return buff.toString();  
 }  
}

Days.java

package entitys;  
  
*/\*\*  
 \* Класс дней недели  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public enum Days { Понедельник, Вторник, Среда, Четверг, Пятница, Суббота, Воскресенье }

## Интерфейс

App.java

package app;  
  
import entitys.\*;  
  
import javax.persistence.\*;  
import javax.swing.\*;  
import javax.swing.table.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.text.ParseException;  
import java.util.List;  
*/\*\*  
 \* Класс главного интерфейса программы  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class App extends JFrame {  
 */\*\* Статическое поле фабрики сущностей \*/* private static final EntityManagerFactory *emf* = Persistence.*createEntityManagerFactory*("test\_persistence");  
 */\*\* Статическое поле менеджера сущностей \*/* protected static final EntityManager *em* = *emf*.createEntityManager();  
 */\*\* Ссылка на созданный интерфейс (на себя)\*/* final private App App;  
 */\*\* Статическое поле выбора текущей таблицы данных \*/* protected static JComboBox *tableSelect*;  
 */\*\* Поле скролл панели для таблицы \*/* private JScrollPane scroll;  
 */\*\* Поле текущей выбранной таблицы данных \*/* private JTable table;  
  
 */\*\*  
 \* Конструктор объекта App по-умолчанию  
 \*/* public App() {  
 super("Диспетчерская");  
 App = this;  
 showGUI();  
 }  
 */\*\*  
 \* Функция задания интерфейса и его демонстрации  
 \*/* public void showGUI() {  
// Создание окна  
 App.setSize(500, 300);  
 App.setLocation(100, 100);  
 App.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
// Создание кнопок и прикрепление иконок на панель инструментов  
 *tableSelect* = new JComboBox(new String[]{"Автобусы", "Водители",  
 "Маршруты"});  
 JButton add = new JButton(new ImageIcon("./src/main/resources/img/add.png")); //new ImageIcon("./src/main/resources/img/add.png")  
 JButton edit = new JButton(new ImageIcon("./src/main/resources/img/edit.png")); //new ImageIcon("./src/main/resources/img/edit.png")  
 JButton remove = new JButton(new ImageIcon("./src/main/resources/img/delete.png")); //new ImageIcon("./src/main/resources/img/delete.png")  
 JButton export = new JButton(new ImageIcon("./src/main/resources/img/export.png")); //new ImageIcon("./src/main/resources/img/export.png")  
  
// Добавление слушателей  
 ActionListener actionSelect = e -> {  
 JComboBox box = (JComboBox)e.getSource();  
 String item = (String)box.getSelectedItem();  
 selectTable(item);  
 };  
 *tableSelect*.addActionListener(actionSelect);  
  
 add.addActionListener(e -> {  
 try {  
 //createTableItem();  
 new CreateItemDialog(App, table);  
 } catch (ParseException parseException) {  
 parseException.printStackTrace();  
 }  
 });  
 edit.addActionListener(e -> {  
 try {  
 new EditItemDialog(App, table);  
 } catch (ParseException parseException) {  
 parseException.printStackTrace();  
 }  
 });  
 remove.addActionListener(e -> new RemovalManager(App, table));  
 export.addActionListener(e -> new PDFExporter(table));  
  
// Настройка подсказок для кнопок  
 *tableSelect*.setToolTipText("Выберите таблицу");  
 add.setToolTipText("Добавить запись");  
 edit.setToolTipText("Редактировать запись");  
 remove.setToolTipText("Удалить запись(-и)");  
 export.setToolTipText("Экспорт текущей таблицы в PDF");  
  
// Добавление кнопок на панель инструментов  
 JToolBar toolBar = new JToolBar("Панель инструментов");  
 toolBar.add(*tableSelect*);  
 toolBar.add(add);  
 toolBar.add(edit);  
 toolBar.add(remove);  
 toolBar.add(export);  
  
// Размещение панели инструментов  
 App.setLayout(new BorderLayout());  
 App.add(toolBar, BorderLayout.*NORTH*);  
  
// Создание таблицы с данными по-умолчанию  
 selectTable("Автобусы");  
  
// Подготовка компонентов поиска  
 JButton report = new JButton("Справка и нарушения");  
 report.addActionListener(e -> new ReportThread("report", *em*).start());  
  
// Добавление компонентов на панель  
 JPanel reportPanel = new JPanel();  
 reportPanel.add(report);  
  
// Размещение панели поиска внизу окна  
 App.add(reportPanel, BorderLayout.*SOUTH*);  
  
// Визуализация экранной формы  
 App.setVisible(true);  
 }  
 */\*\*  
 \* Функция выбора активной таблицы  
 \** ***@param*** *select - название выбираемой таблицы данных  
 \*/* protected void selectTable(String select) { // выбор отображаемой таблицы  
 if (scroll != null) App.remove(scroll); // для случая инициализации  
 String[] columns = null;  
 String[][] data = null;  
 // заполнение даными  
 switch (select) {  
 case ("Автобусы"):  
 columns = new String[] {"ID", "Номер", "Вместимость", "Водитель", "Маршрут", "Состояние"};  
 List<Bus> busList = *em*.createQuery("SELECT b FROM Bus b").getResultList();  
 data = new String[busList.size()][columns.length];  
 for (int i = 0; i < busList.size(); i++) {  
 data[i] = busList.get(i).toTableFormat();  
 }  
 break;  
 case ("Водители"):  
 columns = new String[] {"ID", "ФUО", "Возраст", "Опыт", "Зарплата", "Состояние"};  
 List<Driver> driverList = *em*.createQuery("SELECT d FROM Driver d").getResultList();  
 data = new String[driverList.size()][columns.length];  
 for (int i = 0; i<driverList.size(); i++) {  
 data[i] = driverList.get(i).toTableFormat();  
 }  
 break;  
 case ("Маршруты"):  
 columns = new String[] {"ID", "№", "Откуда", "Куда", "Расписание", "Состояние"};  
 List<Route> routeList = *em*.createQuery("SELECT r FROM Route r").getResultList();  
 data = new String[routeList.size()][columns.length];  
 for (int i = 0; i<routeList.size(); i++) {  
 data[i] = routeList.get(i).toTableFormat();  
 }  
 break;  
 }  
 DefaultTableModel model = new DefaultTableModel(data, columns) {  
 @Override // запрет на редактирование ячеек  
 public boolean isCellEditable(int row, int column) {  
 return false;  
 }  
 };  
 table = new JTable(model);  
 // настройка отрисовки таблицы  
 table.setDefaultRenderer(Object.class, new ViolationTableCellRender(*em*, *tableSelect*.getSelectedIndex()));  
 // Добавление попап меню для таблицы  
 table.setComponentPopupMenu(new TablePopupMenu(table, (String)*tableSelect*.getSelectedItem()));  
 // Добавление упорядочивания в таблице  
 table.setRowSorter(new TableRowSorter<>(model));  
 scroll = new JScrollPane(table);  
 // Размещение таблицы с данными  
 App.add(scroll, BorderLayout.*CENTER*);  
 // обновляем окно для корректного отображения новой таблицы  
 App.setVisible(true);  
 }  
}

CreateItemDialog.java

package app;  
  
import entitys.Bus;  
import entitys.Driver;  
import entitys.Route;  
import utils.ValidationError;  
import utils.WritingError;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.text.ParseException;  
*/\*\*  
 \* Класс окна для создания новых записей  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.2  
 \*/*public class CreateItemDialog extends JDialog {  
 */\*\* Поле со ссылкой на редактируемую таблицу данных \*/* private JTable table = null;  
 */\*\*  
 \* Конструктор объекта CreateItemDialog  
 \** ***@param*** *frame - фрейм-родитель, из которого создаётся Редактор Элементов Таблицы  
 \** ***@param*** *\_table - таблица данных для добавления новых записей  
 \** ***@throws*** *ParseException - некорректная запись в поле с форматированным вводом  
 \*/* protected CreateItemDialog(App frame, JTable \_table) throws ParseException {  
 super(frame, "Добавление записи", true);  
 this.setDefaultCloseOperation(*DISPOSE\_ON\_CLOSE*);  
 this.setLocation(150, 150);  
 // сохранение ссылки на таблицу для функций добавления  
 table = \_table;  
 // панель создания/редактирования элемента  
 JPanel editPanel = new EditPanel(this);  
 this.add(editPanel, BorderLayout.*CENTER*);  
 // кнопка добавления элемента  
 JButton confirm = new JButton("Добавить запись");  
 this.add(confirm, BorderLayout.*SOUTH*);  
 ActionListener actionAddItem = e -> {  
 try {  
 // получаем все компоненты панели ввода данных  
 Component[] components = editPanel.getComponents();  
 // в зависимости от выбранной таблицы получаем введенные данные и передаем в функцию создания новой записи  
 switch (App.*tableSelect*.getSelectedIndex()) {  
 // автобус  
 case 0:  
 String registr = ((JTextField) components[1]).getText();  
 int capacity = Integer.*parseInt*(((JTextField) components[3]).getText());  
 int id\_driver = Integer.*parseInt*(((String) ((JComboBox) components[5]).getSelectedItem()).split(" ")[0]);  
 int id\_route = Integer.*parseInt*(((String) ((JComboBox) components[7]).getSelectedItem()).split(" ")[0]);  
 addBus(registr, capacity, id\_driver, id\_route, this);  
 break;  
 // водитель  
 case 1:  
 String name = ((JTextField) components[1]).getText();  
 int age = Integer.*parseInt*(((JTextField) components[3]).getText());  
 int exp = Integer.*parseInt*(((JTextField) components[5]).getText());  
 addDriver(name, age, exp);  
 break;  
 // маршрут  
 case 2:  
 int number = Integer.*parseInt*(((JTextField) components[1]).getText());  
 String start = ((JTextField) components[3]).getText();  
 String finish = ((JTextField) components[5]).getText();  
 String schedule = ((JTextField) components[7]).getText();  
 addRoute(number, start, finish, schedule);  
 break;  
 }  
 // закрыть окно JDialog и вывести сообщение об успешном добавлении  
 JOptionPane.*showMessageDialog*(this, "<html><h2>Успешно</h2><i>Элемент добавлен</i>");  
 // обновим таблицу для отображения изменений  
 frame.selectTable((String) App.*tableSelect*.getSelectedItem());  
 // закроем диалоговое окно, т.к. оно выполнило свою функцию  
 this.dispose();  
 }  
 catch (WritingError exc) {  
 JOptionPane.*showMessageDialog*(this,"<html><h2>Ошибка</h2><i>Элемент не может быть добавлен, " +  
 "попробуйте снова</i>", exc.getMessage(), JOptionPane.*ERROR\_MESSAGE*);  
 }  
 catch (ValidationError exc) {  
 JOptionPane.*showMessageDialog*(this,"<html><h2>Ошибка</h2>"+exc.getHelp(),  
 exc.getMessage(), JOptionPane.*ERROR\_MESSAGE*);  
 }  
 };  
 confirm.addActionListener(actionAddItem);  
  
 this.pack();  
 this.setVisible(true);  
 }  
 */\*\*  
 \* Функция, создающая новый элемент Автобус  
 \* Добавление записи осуществляется сразу и в БД, и визуально в приложении  
 \** ***@param*** *registr - регистрационный номер автобуса  
 \** ***@param*** *capacity - вместительность автобуса  
 \** ***@param*** *id\_driver - уникальный идентификатор нанимаемого водителя  
 \** ***@param*** *id\_route - уникальный идентификатор выбираемого маршрута  
 \** ***@param*** *component - окно-родитель, из которого производится редактирование (EditItemDialog)  
 \** ***@throws*** *WritingError - ошибка при неудачном редактировании записи  
 \** ***@throws*** *ValidationError - ошибка некорректного ввода данных для информационных полей  
 \*/* private void addBus(String registr, int capacity, int id\_driver, int id\_route, Component component) throws WritingError, ValidationError {  
 if (!App.*em*.getTransaction().isActive()) App.*em*.getTransaction().begin();  
 Bus model = new Bus();  
 if (!model.setRegistr(registr)) throw new ValidationError("Формат рег.номера должен быть вида А777АА");  
 if (!model.setCapacity(capacity)) throw new ValidationError("Вместимость должна быть неотрицательна");  
 App.*em*.persist(model);  
 // Выбор водителя при добавлении  
 Driver driver;  
 if (id\_driver > 0) {  
 driver = App.*em*.find(Driver.class, id\_driver);  
 if (driver.getViolation() != null) throw new ValidationError(driver.getViolation().getDescription());  
 // если водитель работает на другом автобусе, то создаем подтверждающее диалоговое окно  
 if (driver.getBus() != null) {  
 int result = JOptionPane.*showConfirmDialog*(this, "Выбранный водитель уже работает на другом автобусе. Вы уверены?",  
 "Окно подтверждения", JOptionPane.*YES\_NO\_OPTION*, JOptionPane.*WARNING\_MESSAGE*);  
 if (result != JOptionPane.*YES\_OPTION*)  
 throw new WritingError();  
 }  
 model.hireToDriver(driver);  
 }  
 // Выбор маршрута при добавлении  
 Route route;  
 if (id\_route > 0 && id\_driver > 0) { // без водителя маршрут не выбрать!  
 route = App.*em*.find(Route.class, id\_route);  
 if (route.getViolation() != null) throw new ValidationError(route.getViolation().getDescription());  
 model.chooseRoute(route);  
 }  
 else if (id\_route > 0 && id\_driver <= 0) { // попытка выбрать маршрут без водителя  
 JOptionPane.*showMessageDialog*(component,"<html><h2>Ошибка</h2><i>Автобус без водителя " +  
 "выставлять на маршрут запрещено!</i>", "Ошибка добавления", JOptionPane.*ERROR\_MESSAGE*);  
 throw new WritingError();  
 }  
 App.*em*.getTransaction().commit();  
 DefaultTableModel dtm = (DefaultTableModel) table.getModel();  
 dtm.addRow(model.toTableFormat());  
 // обновляем окно для корректного отображения новой таблицы  
 dtm.fireTableDataChanged();  
 }  
 */\*\*  
 \* Функция, создающая новый элемент Водитель  
 \* Добавление записи осуществляется сразу и в БД, и визуально в приложении  
 \** ***@param*** *name - ФИО водителя  
 \** ***@param*** *age - возраст  
 \** ***@param*** *exp - опыт работы  
 \*/* private void addDriver(String name, int age, int exp) {  
 if (!App.*em*.getTransaction().isActive()) App.*em*.getTransaction().begin();  
 Driver model = new Driver(name, age, exp);  
 App.*em*.persist(model);  
 App.*em*.getTransaction().commit();  
 DefaultTableModel dtm = (DefaultTableModel) table.getModel();  
 dtm.addRow(model.toTableFormat());  
 // обновляем окно для корректного отображения новой таблицы  
 dtm.fireTableDataChanged();  
 }  
 */\*\*  
 \* Функция, создающая новый элемент Маршрут  
 \* Добавление записи осуществляется сразу и в БД, и визуально в приложении  
 \** ***@param*** *number - номер маршрута  
 \** ***@param*** *start - точка начала движения  
 \** ***@param*** *finish - точка окончания движения  
 \** ***@param*** *schedule - недельное расписание движения  
 \* формат задания расписания, где расписание дня недели выглядит так: 09:00 21:00;  
 \*/* private void addRoute(int number, String start, String finish, String schedule) {  
 if (!App.*em*.getTransaction().isActive()) App.*em*.getTransaction().begin();  
 Route model = new Route(number, start, finish, schedule);  
 App.*em*.persist(model);  
 App.*em*.getTransaction().commit();  
 // получаем модель таблицы и добавляем в неё новую запись  
 DefaultTableModel dtm = (DefaultTableModel) table.getModel();  
 dtm.addRow(model.toTableFormat());  
 // обновляем окно для корректного отображения новой таблицы  
 dtm.fireTableDataChanged();  
 }  
}

EditItemDialog.java

package app;  
  
import entitys.Bus;  
import entitys.Driver;  
import entitys.Route;  
import utils.ValidationError;  
import utils.WritingError;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.text.ParseException;  
*/\*\*  
 \* Класс окна для редактирования записей  
 \* Редактор Элементов Таблицы  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.2  
 \*/*public class EditItemDialog extends JDialog {  
 */\*\*  
 \* Конструктор объекта EditItemDialog  
 \** ***@param*** *frame - фрейм-родитель, из которого создаётся Редактор Элементов Таблицы  
 \** ***@param*** *\_table - таблица данных для редактирования  
 \** ***@throws*** *ParseException - некорректная запись в поле с форматированным вводом  
 \*/* protected EditItemDialog(App frame, JTable \_table) throws ParseException {  
 super(frame, "Редактирование записи", true);  
 DefaultTableModel dtm = (DefaultTableModel) \_table.getModel();  
 int rowIndex = \_table.getSelectedRow();  
 if (rowIndex > -1) {  
 this.setDefaultCloseOperation(*DISPOSE\_ON\_CLOSE*);  
 this.setLocation(150, 150);  
 // панель создания/редактирования элемента  
 JPanel editPanel = new EditPanel(this);  
 this.add(editPanel, BorderLayout.*CENTER*);  
 // получаем все компоненты панели ввода данных  
 Component[] components = editPanel.getComponents();  
 // кнопка редактирования элемента  
 JButton confirm = new JButton("Изменить запись");  
 this.add(confirm, BorderLayout.*SOUTH*);  
 // достаём ID редактируемого элемента  
 int ID = Integer.*parseInt*((String) dtm.getValueAt(rowIndex, 0));  
 if (!App.*em*.getTransaction().isActive()) App.*em*.getTransaction().begin();  
 // заполняем editPanel исходными данными  
 switch (App.*tableSelect*.getSelectedIndex()) {  
 // автобус  
 case 0:  
 Bus bus = App.*em*.find(Bus.class, ID);  
 ((JTextField) components[1]).setText(bus.getRegistr());  
 ((JTextField) components[3]).setText(Integer.*toString*(bus.getCapacity()));  
 if (bus.getDriver() != null)  
 ((JComboBox) components[5]).setSelectedItem(bus.getDriver().getId() + " " + bus.getDriver().getName());  
 if (bus.getRoute() != null)  
 ((JComboBox) components[7]).setSelectedItem(bus.getRoute().toString());  
 break;  
 // водитель  
 case 1:  
 Driver driver = App.*em*.find(Driver.class, ID);  
 ((JTextField) components[1]).setText(driver.getName());  
 ((JTextField) components[3]).setText(Integer.*toString*(driver.getAge()));  
 ((JTextField) components[5]).setText(Integer.*toString*(driver.getExp()));  
 break;  
 // маршрут  
 case 2:  
 Route route = App.*em*.find(Route.class, ID);  
 ((JTextField) components[1]).setText(Integer.*toString*(route.getNumber()));  
 ((JTextField) components[3]).setText(route.getStart());  
 ((JTextField) components[5]).setText(route.getFinish());  
 ((JTextField) components[7]).setText(route.getSchedule().getDays());  
 break;  
 }  
 // создаём слушатель для кнопки-подтверждения  
 ActionListener actionEditItem = e -> {  
 try {  
 // в зависимости от выбранной таблицы получаем введенные данные и передаем в функцию создания новой записи  
 switch (App.*tableSelect*.getSelectedIndex()) {  
 // автобус  
 case 0:  
 Bus bus = App.*em*.find(Bus.class, ID);  
 editBus(bus, components, this);  
 break;  
 // водитель  
 case 1:  
 Driver driver = App.*em*.find(Driver.class, ID);  
 editDriver(driver, components, this);  
 break;  
 // маршрут  
 case 2:  
 Route route = App.*em*.find(Route.class, ID);  
 editRoute(route, components, this);  
 break;  
 }  
 // закрыть окно JDialog и вывести сообщение об успешном добавлении  
 JOptionPane.*showMessageDialog*(this, "<html><h2>Успешно</h2><i>Элемент отредактирован</i>");  
 // обновим таблицу для отображения изменений  
 frame.selectTable((String) App.*tableSelect*.getSelectedItem());  
 // закроем диалоговое окно, т.к. оно выполнило свою функцию  
 this.dispose();  
 }  
 catch (WritingError exc) {  
 JOptionPane.*showMessageDialog*(this,"<html><h2>Ошибка</h2><i>Элемент не может быть изменён, " +  
 "попробуйте снова</i>", exc.getMessage(), JOptionPane.*ERROR\_MESSAGE*);  
 }  
 catch (ValidationError exc) {  
 JOptionPane.*showMessageDialog*(this,"<html><h2>Ошибка</h2>"+exc.getHelp(),  
 exc.getMessage(), JOptionPane.*ERROR\_MESSAGE*);  
 }  
 };  
 confirm.addActionListener(actionEditItem);  
  
 this.pack();  
 this.setVisible(true);  
 } else {  
 JOptionPane.*showMessageDialog*(frame, "<html><h2>Ошибка</h2><i>Для редактирования элемента " +  
 "выделите соответствующую строку!</i>", "Ошибка Редактирования", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
 */\*\*  
 \* Функция, редактирующая объект Автобуса  
 \* Редактирование осуществляется сразу и в БД, и визуально в приложении  
 \** ***@param*** *model - ссылка на автобус для редактирования  
 \** ***@param*** *components - компоненты с данными о новых введённых параметрах (компоненты EditPanel)  
 \** ***@param*** *component - окно-родитель, из которого производится редактирование (EditItemDialog)  
 \** ***@throws*** *WritingError - ошибка при неудачном редактировании записи  
 \** ***@throws*** *ValidationError - ошибка некорректного ввода данных для информационных полей  
 \*/* private void editBus(Bus model, Component[] components, Component component) throws WritingError, ValidationError {  
 // получаем введённые данные(если они были)  
 String registr = ((JTextField)components[1]).getText();  
 int capacity = Integer.*parseInt*( ((JTextField)components[3]).getText() );  
 int id\_driver = Integer.*parseInt*(((String)((JComboBox)components[5]).getSelectedItem()).split(" ")[0]);  
 int id\_route = Integer.*parseInt*(((String)((JComboBox)components[7]).getSelectedItem()).split(" ")[0]);  
  
 // изменяем модель  
 if (!model.setRegistr(registr)) throw new ValidationError("Формат рег.номера должен быть вида А777АА");  
 if (!model.setCapacity(capacity)) throw new ValidationError("Вместимость должна быть неотрицательна");  
 if (model.getViolation() != null) throw new ValidationError(model.getViolation().getDescription());  
 if (!App.*em*.getTransaction().isActive()) App.*em*.getTransaction().begin();  
 App.*em*.persist(model);  
 // Выбор(удаление) водителя при изменении  
 Driver newDriver = App.*em*.find(Driver.class, id\_driver);  
 if (newDriver != null) {  
 if (newDriver.getViolation() != null) throw new ValidationError(newDriver.getViolation().getDescription());  
 // если водитель работает на другом автобусе, то создаем подтверждающее диалоговое окно  
 if (newDriver.getBus() != null && newDriver.getBus() != model) {  
 int result = JOptionPane.*showConfirmDialog*(component, "Выбранный водитель уже работает на другом автобусе. Вы уверены?",  
 "Окно подтверждения", JOptionPane.*YES\_NO\_OPTION*, JOptionPane.*WARNING\_MESSAGE*);  
 if (result != JOptionPane.*YES\_OPTION*)  
 throw new WritingError();  
 }  
 model.hireToDriver(newDriver);  
 }  
 // иначе если требуется уволить водителя  
 else if (model.getDriver() != null) {  
 model.fireToDriver();  
 }  
 // Выбор(удаление) маршрута при изменении  
 Route newRoute = App.*em*.find(Route.class, id\_route);  
 if (newRoute != null) {  
 if (newRoute.getViolation() != null) throw new ValidationError(newRoute.getViolation().getDescription());  
 if ( id\_driver > 0 ) {  
 model.chooseRoute(newRoute);  
 }  
 else if (id\_driver == 0) {  
 JOptionPane.*showMessageDialog*(component,"<html><h2>Ошибка</h2><i>Автобус без водителя " +  
 "выставлять на маршрут запрещено!</i>", "Ошибка редактирования", JOptionPane.*ERROR\_MESSAGE*);  
 throw new WritingError();  
 }  
 }  
 else if (model.getRoute() != null) {  
 model.chooseRoute(null);  
 }  
 App.*em*.getTransaction().commit();  
 }  
 */\*\*  
 \* Функция, редактирующая объект Водитель  
 \* Редактирование осуществляется сразу и в БД, и визуально в приложении  
 \** ***@param*** *model - ссылка на водителя для редактирования  
 \** ***@param*** *components - компоненты с данными о новых введённых параметрах (компоненты EditPanel)  
 \** ***@param*** *component - окно-родитель, из которого производится редактирование (EditItemDialog)  
 \*/* private void editDriver(Driver model, Component[] components, Component component) {  
 // получаем введённые данные  
 String name = ((JTextField)components[1]).getText();  
 int age = Integer.*parseInt*( ((JTextField)components[3]).getText() );  
 int exp = Integer.*parseInt*( ((JTextField)components[5]).getText() );  
  
 // изменяем модель  
 model.setName(name);  
 model.setAge(age);  
 model.setExp(exp);  
  
 if (!App.*em*.getTransaction().isActive()) App.*em*.getTransaction().begin();  
 App.*em*.persist(model);  
 App.*em*.getTransaction().commit();  
 }  
 */\*\*  
 \* Функция, редактирующая объект Маршрут  
 \* Редактирование осуществляется сразу и в БД, и визуально в приложении  
 \** ***@param*** *model - ссылка на маршрут для редактирования  
 \** ***@param*** *components - компоненты с данными о новых введённых параметрах (компоненты EditPanel)  
 \** ***@param*** *component - окно-родитель, из которого производится редактирование (EditItemDialog)  
 \*/* private void editRoute(Route model, Component[] components, Component component) {  
 // получаем введённые данные  
 int number = Integer.*parseInt*( ((JTextField)components[1]).getText() );  
 String start = ((JTextField)components[3]).getText();  
 String finish = ((JTextField)components[5]).getText();  
 String schedule = ((JTextField)components[7]).getText();  
  
 // изменяем модель  
 model.setNumber(number);  
 model.setStart(start);  
 model.setFinish(finish);  
 model.setSchedule(schedule);  
  
 if (!App.*em*.getTransaction().isActive()) App.*em*.getTransaction().begin();  
 App.*em*.persist(model);  
 App.*em*.getTransaction().commit();  
 }  
}

RemovalManager.java

package app;  
  
import entitys.Bus;  
import entitys.Driver;  
import entitys.Route;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
*/\*\*  
 \* Класс менеджера удаления записей  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class RemovalManager {  
 */\*\*  
 \* Конструктор - создание нового объекта RemovalManager  
 \** ***@param*** *frame - окно-родитель, из которого вызывается диалоговое окно  
 \** ***@param*** *table - таблицы данных для удаления записей  
 \*/* protected RemovalManager(App frame, JTable table) {  
 DefaultTableModel dtm = (DefaultTableModel) table.getModel();  
 int indexes[] = table.getSelectedRows();  
 // достаем ID удаляемого объекта в выделенной строке  
 if (indexes.length > 0) {  
 // проверка, чтобы случайно не удалить  
 int result = JOptionPane.*showConfirmDialog*(frame, "Вы точно хотите удалить выбранную строку(-и)?",  
 "Окно подтверждения", JOptionPane.*YES\_NO\_OPTION*, JOptionPane.*WARNING\_MESSAGE*);  
 if (result == JOptionPane.*YES\_OPTION*) {  
 for (int rowIndex: indexes) {  
 // удаление элемента из БД и сразу визуально из таблицы  
 int ID = Integer.*parseInt*((String) dtm.getValueAt(rowIndex, 0));  
 if (!App.*em*.getTransaction().isActive()) App.*em*.getTransaction().begin();  
 // увольняем водителя(и если автобус был на маршруте, то он также удалится)  
 // снимаем все автобусы с маршрута, чтобы не удалить вместе  
 switch (App.*tableSelect*.getSelectedIndex()) {  
 // удаляем автобус  
 case 0:  
 Bus bus = App.*em*.find(Bus.class, ID);  
 bus.fireToDriver();  
 App.*em*.remove(bus);  
 break;  
 // удаляем водителя  
 case 1:  
 Driver driver = App.*em*.find(Driver.class, ID);  
 if (driver.getBus() != null)  
 driver.getBus().fireToDriver();  
 App.*em*.remove(driver);  
 break;  
 // удаляем маршрут  
 case 2:  
 Route route = App.*em*.find(Route.class, ID);  
 route.exclude();  
 App.*em*.remove(route);  
 break;  
 }  
 App.*em*.getTransaction().commit();  
 }  
 // удаляем визуально из таблицы в приложении  
 for (int i = indexes.length - 1; i>=0; --i) {  
 dtm.removeRow(indexes[i]);  
 }  
 dtm.fireTableDataChanged();  
 }  
 }  
 else {  
 JOptionPane.*showMessageDialog*(frame,"<html><h2>Ошибка</h2><i>Для удаления элемента " +  
 "выделите соответствующую строку!</i>", "Ошибка Удаления", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
}

EditPanel.java

package app;  
  
import entitys.Route;  
  
import javax.swing.\*;  
import javax.swing.text.MaskFormatter;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.text.ParseException;  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Класс панели для редактирования объектов  
 \* Предназанчена для добавления в уже созданное диалоговое окно parent  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.2  
 \*/*public class EditPanel extends JPanel{  
 */\*\*  
 \* Конструктор объекта EditPanel  
 \** ***@param*** *parent - диалоговое окно-родитель, из которого создаётся панель редактирования  
 \** ***@throws*** *ParseException - некорректная запись в поле с форматированным вводом  
 \*/* protected EditPanel(JDialog parent) throws ParseException {  
 super();  
 switch (App.*tableSelect*.getSelectedIndex()) {  
 case 0: // автобус  
 JLabel regLabel = new JLabel("Регистрационный номер:");  
 MaskFormatter regFormatter = new MaskFormatter("U###UU");  
 regFormatter.setPlaceholderCharacter('#');  
 JFormattedTextField registr = new JFormattedTextField(regFormatter);  
 JLabel capLabel = new JLabel("Вместительность:");  
 MaskFormatter capFormatter = new MaskFormatter("##");  
 capFormatter.setPlaceholderCharacter('0');  
 JFormattedTextField capacity = new JFormattedTextField(capFormatter);  
  
 JLabel driverLabel = new JLabel("Доступные водители:");  
 List<String> driverNameList = App.*em*.createQuery("SELECT name FROM Driver d WHERE violation=null").getResultList();  
 List<Integer> driverIDList = App.*em*.createQuery("SELECT id FROM Driver d WHERE violation=null").getResultList();  
 // добавим вариант отсутствия водителя при создании  
 driverNameList.add(0, "0 Отсутствует"); driverIDList.add(0, 0);  
 // допишем ID каждому из имен для избежания путаницы  
 for (int i=1; i<driverNameList.size(); i++) {  
 driverNameList.set(i, driverIDList.get(i).toString() + " " + driverNameList.get(i));  
 }  
 JComboBox driverSelect = new JComboBox(driverNameList.toArray());  
  
 JLabel routeLabel = new JLabel("Доступные маршруты:");  
 List<Route> routes = App.*em*.createQuery("SELECT r FROM Route r WHERE violation=null").getResultList();  
 // сформируем варианты выбора для JComboBox'а в формате "НомерМаршрута Старт Финиш"  
 List<String> routeBox = new ArrayList<>();  
 routeBox.add("0 Отсутствует");  
 for (Route route: routes) {  
 routeBox.add(route.toString());  
 }  
 JComboBox routeSelect = new JComboBox(routeBox.toArray());  
  
 this.setLayout(new GridLayout(4, 2, 5, 1));  
 // добавляем все элементы на панель в формате сетки 4х2  
 this.add(regLabel); this.add(registr);  
 this.add(capLabel); this.add(capacity);  
 this.add(driverLabel); this.add(driverSelect);  
 this.add(routeLabel); this.add(routeSelect);  
 break;  
 case 1: // водитель  
 JLabel nameLabel = new JLabel("ФUО");  
 JFormattedTextField name = new JFormattedTextField();  
 JLabel ageLabel = new JLabel("Возраст:");  
 MaskFormatter ageFormatter = new MaskFormatter("##");  
 ageFormatter.setPlaceholderCharacter('0');  
 JFormattedTextField age = new JFormattedTextField(ageFormatter);  
 JLabel expLabel = new JLabel("Стаж работы:");  
 JFormattedTextField exp = new JFormattedTextField(ageFormatter);  
  
 this.setLayout(new GridLayout(3, 2, 5, 1));  
 // добавляем все элементы на панель в формате сетки 3х2  
 this.add(nameLabel); this.add(name);  
 this.add(ageLabel); this.add(age);  
 this.add(expLabel); this.add(exp);  
 break;  
 case 2: // маршрут  
 JLabel numLabel = new JLabel("Номер маршрута:");  
 MaskFormatter numFormatter = new MaskFormatter("###");  
 numFormatter.setPlaceholderCharacter('0');  
 numFormatter.setValidCharacters("0123456789");  
 JFormattedTextField number = new JFormattedTextField(numFormatter);  
 JLabel startLabel = new JLabel("Начало маршрута: ");  
 JFormattedTextField start = new JFormattedTextField();  
 JLabel finishLabel = new JLabel("Конечная остановка: ");  
 JFormattedTextField finish = new JFormattedTextField();  
 JLabel scheduleLabel = new JLabel("Расписание: ");  
 JTextField schedule = new JTextField();  
 JLabel emptyLabel = new JLabel(" ");  
 JButton makeSchedule = new JButton("Составить расписание");  
 ActionListener actionMakeSchedule = e -> {  
 try {  
 new ScheduleDialog(parent, schedule);  
 } catch (ParseException parseException) {  
 parseException.printStackTrace();  
 }  
 };  
 makeSchedule.addActionListener(actionMakeSchedule);  
  
 this.setLayout(new GridLayout(5, 2, 5, 1));  
 // добавляем все элементы на панель в формате сетки 5x2  
 this.add(numLabel); this.add(number);  
 this.add(startLabel); this.add(start);  
 this.add(finishLabel); this.add(finish);  
 this.add(scheduleLabel); this.add(schedule);  
 this.add(emptyLabel); this.add(makeSchedule);  
 break;  
 }  
 }  
}

HTMLExporter.java

package app;  
  
import java.awt.FileDialog;  
import java.io.\*;  
  
import javax.swing.JFrame;  
  
*/\*\*  
 \* Класс экспортёра HTML  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class HTMLExporter extends JFrame {  
 */\*\*  
 \* Конструктор - создание нового объекта HTMLExporter  
 \** ***@param*** *html - строка размеченная в стиле HTML  
 \*/* public HTMLExporter(String html) {  
 FileDialog fileDial = new FileDialog(this, "Export HTML", FileDialog.*SAVE*);  
 fileDial.setFile("\*.html");  
 fileDial.setVisible(true);  
 if (fileDial.getFile() != null)  
 if (!fileDial.getFile().contains("."))  
 fileDial.setFile(fileDial.getFile()+".html");  
 String fileFullName = fileDial.getDirectory() + fileDial.getFile();  
 {  
 try {  
 FileWriter newHtmlFile = new FileWriter(fileFullName, false);  
 newHtmlFile.write(html);  
 newHtmlFile.flush();  
 newHtmlFile.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

PDFExporter.java

package app;  
  
import com.itextpdf.text.\*;  
import com.itextpdf.text.pdf.\*;  
import org.apache.log4j.Logger;  
  
import java.awt.FileDialog;  
import java.io.\*;  
import java.util.Date;  
  
import javax.swing.JFrame;  
import javax.swing.JTable;  
import javax.swing.table.DefaultTableModel;  
*/\*\*  
 \* Класс экспортёра PDF  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class PDFExporter extends JFrame {  
 */\*\* Статическое поле для логгера \*/* public static final Logger logger = Logger.getLogger(PDFExporter.class);  
 */\*\*  
 \* Конструктор - создание нового объекта HTMLExporter  
 \** ***@param*** *TableData - таблица данных для экспорта  
 \*/* public PDFExporter(JTable TableData)  
 {  
 FileDialog fileDial = new FileDialog(this, "Export PDF", FileDialog.SAVE);  
 fileDial.setFile("\*.pdf");  
 fileDial.setVisible(true);  
 if (fileDial.getFile() != null)  
 if (!fileDial.getFile().contains("."))  
 fileDial.setFile(fileDial.getFile()+".pdf");  
 String fileFullName = fileDial.getDirectory() + fileDial.getFile();  
  
 if(!fileFullName.equals("nullnull"))  
 {  
 Document document = new Document(PageSize.A4, 15, 15, 15, 15);  
 PdfPTable pdfTable = new PdfPTable(TableData.getColumnCount());  
  
 try {  
 PdfWriter.getInstance(document, new FileOutputStream(fileFullName));  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 logger.error("File Not Found", e);  
 } catch (DocumentException e) {  
 e.printStackTrace();  
 logger.error("IO exception occured", e);  
 }  
  
 Font font = FontFactory.getFont("/fonts/DejaVuSans.ttf", "cp1251", BaseFont.EMBEDDED, 10);  
  
 // add info  
 Paragraph info = new Paragraph("Report from " + new Date() + " \n\n");  
  
 // table header  
 for(int i =0; i < TableData.getColumnCount(); ++i) {  
 PdfPCell header = new PdfPCell();  
 header.setBackgroundColor(BaseColor.LIGHT\_GRAY);  
 header.setBorderWidth(2);  
 header.setPhrase(new Phrase(TableData.getModel().getColumnName(i), font));  
 pdfTable.addCell(header);  
 }  
  
 // table data  
 String data[][] = getTableData(TableData);  
 for(int i = 0; i < TableData.getRowCount(); i++){  
 for(int j =0; j < TableData.getColumnCount(); j++)  
 {  
 String dat = (data[i][j] == null? "" : data[i][j]);  
 pdfTable.addCell(new Phrase(dat, font));  
 }  
 }  
  
 document.open();  
  
 try {  
 document.add(info);  
 document.add(pdfTable);  
 logger.debug("File exported successfully");  
 } catch (DocumentException e) {  
 e.printStackTrace();  
 logger.error("Write Error", e);  
 }  
  
 document.close();  
 }  
 else {  
 logger.warn("User aborted file loading operation");  
 }  
 }  
 */\*\*  
 \* Функция получения данных из таблицы  
 \** ***@param*** *table - таблица данных  
 \** ***@return*** *данные исходной таблицы в формате двумерного массива строк  
 \*/* private String[][] getTableData(JTable table) {  
 DefaultTableModel dtm = (DefaultTableModel) table.getModel();  
 int nRow = dtm.getRowCount(), nCol = dtm.getColumnCount();  
 String[][] tableData = new String[nRow][nCol];  
 for (int i = 0; i < nRow; i++)  
 for (int j = 0; j < nCol; j++)  
 tableData[i][j] = (String) dtm.getValueAt(i, j);  
 return tableData;  
 }  
}

ReportThread.java

package app;  
  
import entitys.Route;  
import entitys.Violation;  
  
import javax.persistence.EntityManager;  
import javax.swing.\*;  
import java.awt.\*;  
import java.util.Date;  
import java.util.List;  
*/\*\*  
 \* Класс поток для генерации отчёта по маршрутам и нарушениям  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class ReportThread extends Thread{  
 */\*\* Поле со ссылкой на EntityManager \*/* final private EntityManager em;  
 */\*\* Поле текстового редактора для отображения отчёта \*/* final private JTextPane textEditor = new JTextPane();  
 */\*\* Поле для кнопки экспорта \*/* final private JButton export = new JButton(new ImageIcon("./src/main/resources/img/html.png"));  
 */\*\* Поле для кнопки обновления данных \*/* final private JButton refresh = new JButton(new ImageIcon("./src/main/resources/img/refresh.png"));  
 */\*\* Поле для указания даты последнего обновления \*/* final private JLabel date = new JLabel();  
 */\*\*  
 \* Конструктор - создание нового потока ReportThread  
 \** ***@param*** *name - имя потока  
 \** ***@param*** *\_em - ссылка на EntityManager  
 \*/* public ReportThread(String name, EntityManager \_em) {  
 super(name);  
 em = \_em;  
 }  
 */\*\*  
 \* Функция запуска потока  
 \*/* public void run() {  
 createUI();  
 synchronized (textEditor) {  
 // Загрузка документа в приложении  
 loadTextToHTML();  
 }  
 }  
 */\*\*  
 \* Функция создания интерфейса отчёта  
 \*/* private void createUI() {  
 JFrame frame = new JFrame("Справка и выявленные нарушения");  
 frame.setSize(400, 300);  
 frame.setLocation(100, 100);  
 frame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);  
 frame.setLayout(new BorderLayout());  
  
 // Создание редактора  
 //textEditor = new JTextPane();  
 textEditor.setEditable(false);  
 textEditor.setContentType("text/html");  
 // возможные действия с отчётом и информация  
 export.addActionListener(e -> {  
 synchronized (textEditor) {  
 // Запись отчёта в HTML файл  
 new HTMLExporter(textEditor.getText());  
 }  
 });  
 refresh.addActionListener(e -> {  
 synchronized (textEditor) {  
 // Загрузка документа в приложении  
 loadTextToHTML();  
 }  
 setDate();  
 });  
 export.setToolTipText("Экпорт в HTML");  
 refresh.setToolTipText("Обновить данные");  
 setDate();  
 JPanel actionPanel = new JPanel();  
 actionPanel.add(export);  
 actionPanel.add(refresh);  
 actionPanel.add(date);  
 // добавление в фрейм  
 frame.add(new JScrollPane(textEditor), BorderLayout.CENTER);  
 frame.add(actionPanel, BorderLayout.SOUTH);  
  
 //pack();  
 frame.setSize(480, 560);  
 frame.setVisible(true);  
 }  
 */\*\*  
 \* Функция задания текущего времени  
 \*/* private void setDate() {  
 date.setText("recent change " + new Date().toString());  
 }  
 */\*\*  
 \* Функция загрузки отчёта HTML формата в {****@link*** *ReportThread#textEditor}  
 \*/* private void loadTextToHTML() {  
 StringBuilder doc = new StringBuilder();  
 // начало HTML файла  
 doc.append("<html><body style=\"font-size:12px\" style=\"font-family: FreeSans\">");  
 doc.append("<head>" + "<meta http-equiv=\"content-type\" content=\"application/xhtml+xml; charset=UTF-8\"/>" + "</head>");  
 doc.append("<p style=\"margin: 5px\">");  
 // заголовок отчёта  
 doc.append("<h1><strong><span style=\"font-size:16px\">" + "Отчёт от " + new Date() + "</span></strong></h1><hr>");  
 // добавляем информацию о маршрутах  
 doc.append("<h3><strong><span style=\"font-size:14px\">" + "Uнформация о маршрутах" + "</span></strong></h3>");  
 java.util.List<Route> routeList = em.createQuery("SELECT r FROM Route r").getResultList();  
 for (int i = 0; i < routeList.size(); ++i) {  
 doc.append("<p style=\"margin-left: 10px\">");  
 doc.append(routeList.get(i).toHTMLFormat());  
 doc.append("</p>");  
 }  
 // информация о нарушениях  
 doc.append("<h3><strong><span style=\"font-size:14px\">" + "Выявленные нарушения" + "</span></strong></h3>");  
 List<Violation> violationList = em.createQuery("SELECT v FROM Violation v WHERE fixdatetime=null").getResultList();  
 for (int i = 0; i < violationList.size(); ++i) {  
 doc.append(violationList.get(i).toHTMLFormat());  
 }  
 doc.append("</p>");  
 // конец HTML файла  
 doc.append("</html></body>");  
 textEditor.setText(doc.toString());  
 textEditor.setCaretPosition(0);  
 }  
}

ScheduleDialog.java

package app;  
  
import entitys.Days;  
  
import javax.swing.\*;  
import javax.swing.text.MaskFormatter;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.text.ParseException;  
*/\*\*  
 \* Класс диалогового окна для задания расписания  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class ScheduleDialog extends JDialog {  
 */\*\*  
 \* Конструктор - создание нового объекта ScheduleDialog  
 \** ***@param*** *parent - ссылка на окно-родитель, вызывающего новый объект ScheduleDialog  
 \** ***@param*** *schedule - текстовое поле, содержащее расписание  
 \** ***@throws*** *ParseException - некорректные данные в поле расписания  
 \*/* protected ScheduleDialog(JDialog parent, JTextField schedule) throws ParseException {  
 super(parent, "Составление расписания", true);  
 this.setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);  
 this.setLocation(150, 150);  
  
 String[] initicalSchedule = schedule.getText().split(";");  
 JPanel schedulePanel = new JPanel();  
 schedulePanel.setLayout(new GridLayout(7, 1, 5, 1));  
 // добавляем все элементы на панель в формате сетки 7x1 и заполняем(если данные уже есть)  
 for (int i=0; i<7; i++) {  
 JPanel day = new JPanel();  
 day.setLayout(new FlowLayout(FlowLayout.LEFT));  
 JLabel dayFrom = new JLabel(Days.values()[i] + " с"); // добавляем подпись дня недели  
 // Определение маски и содание поля ввода времени телефона  
 MaskFormatter timeFormatter = new MaskFormatter("##:##");  
 timeFormatter.setPlaceholderCharacter('0');  
 JFormattedTextField from = new JFormattedTextField(timeFormatter); // ввод времени начала работы  
 JLabel toLabel = new JLabel("до"); // добавляем подпись дня недели  
 JFormattedTextField to = new JFormattedTextField(timeFormatter); // ввод времени окончания работы  
 JLabel intervalLabel = new JLabel("каждые"); // добавляем подпись дня недели  
 MaskFormatter intervalFormatter = new MaskFormatter("##");  
 intervalFormatter.setPlaceholderCharacter('0');  
 JFormattedTextField interval = new JFormattedTextField(intervalFormatter); // форматированный ввод интервала  
 JLabel minLabel = new JLabel("минут"); // добавляем подпись дня недели  
 // Добавляем все на панель  
 day.add(dayFrom); day.add(from); day.add(toLabel); day.add(to);  
 day.add(intervalLabel); day.add(interval); day.add(minLabel);  
 // заполняем поля для случая с редактированием записи  
 if (initicalSchedule.length == 7) {  
 from.setText(initicalSchedule[i].split(" ")[0]);  
 to.setText(initicalSchedule[i].split(" ")[1]);  
 interval.setText(initicalSchedule[i].split(" ")[2]);  
 }  
 schedulePanel.add(day);  
 }  
 this.add(schedulePanel, BorderLayout.CENTER);  
 // добавляем кнопку подтверждения и слушателя к ней  
 JButton confirm = new JButton("Подтвердить");  
 ActionListener actionConfirmSchedule = e -> {  
 // задание расписания по умолчанию  
 String str = "";  
 boolean validData = true; // проверка корректности введённых данных  
 // распаковали 7 дней  
 Component[] daysComponents = schedulePanel.getComponents();  
 for (int i=0; i<7; i++) {  
 // распакуем каждый из дней  
 Component[] dayComponents = ((JPanel) daysComponents[i]).getComponents();  
 // преобразуем все введённые данные в единую строку принятого формата  
 String from = ((JFormattedTextField)dayComponents[1]).getText(),  
 to = ((JFormattedTextField)dayComponents[3]).getText(),  
 interval = ((JFormattedTextField)dayComponents[5]).getText();  
 str += from + " " + to + " " + interval + ";";  
 // если время начала позже времени окончания или указанное время некорректно или интервал <3 минут  
 if (from.compareTo(to) >= 0 || from.compareTo("23:59") > 0 ||  
 to.compareTo("23:59") > 0 || interval.compareTo("03") < 0) {  
 validData = false;  
 }  
 }  
 if (validData == true) {  
 // закрыть окно JDialog и вывести сообщение об успешном добавлении  
 JOptionPane.showMessageDialog(this,"<html><h2>Успешно</h2><i>Расписание сформировано!</i>");  
 // выставляем составленное расписание  
 schedule.setText(str);  
 // закроем диалоговое окно, т.к. оно выполнило свою функцию  
 this.dispose();  
 }  
 else {  
 str = "09:00 21:00 5;09:00 21:00 5;09:00 21:00 5;09:00 21:00 5;09:00 21:00 5;09:00 21:00 5;09:00 21:00 5;";  
 JOptionPane.showMessageDialog(this,"<html><h2>Ошибка</h2><i>Расписание составлено некорректно, " +  
 "попробуйте снова</i>", "Ошибка формирования", JOptionPane.ERROR\_MESSAGE);  
 }  
 };  
 confirm.addActionListener(actionConfirmSchedule);  
 this.add(confirm, BorderLayout.SOUTH);  
  
 this.pack();  
 this.setVisible(true);  
 }  
}

ScheduleViewer.java

package app;  
  
import entitys.\*;  
  
import javax.swing.\*;  
import javax.swing.event.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.util.ArrayList;  
*/\*\*  
 \* Класс поп-ап меню для таблицы  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class TablePopupMenu extends JPopupMenu {  
 */\*\*  
 \* Конструктор - создание нового объекта TablePopupMenu  
 \** ***@param*** *table - таблица данных, для которой создается поп-ап меню  
 \** ***@param*** *select - идентификатор выбранной таблицы(название)  
 \*/* public TablePopupMenu(JTable table, String select) {  
 super();  
 // автоматически выбрать строку, в которой был сделан щелчок правой кнопкой мыши  
 JPopupMenu p = this; // сохранение ссылки для последующей работы  
 this.addPopupMenuListener(new PopupMenuListener() {  
 @Override  
 public void popupMenuWillBecomeVisible(PopupMenuEvent e) {  
 SwingUtilities.invokeLater(new Runnable() {  
 @Override  
 public void run() {  
 int rowAtPoint = table.rowAtPoint(SwingUtilities.convertPoint(p, new Point(0, 0), table));  
 if (rowAtPoint > -1) {  
 table.setRowSelectionInterval(rowAtPoint, rowAtPoint);  
 }  
 }  
 });  
 }  
 @Override  
 public void popupMenuWillBecomeInvisible(PopupMenuEvent e) {  
 // TODO Auto-generated method stub  
 //System.out.println("i will be back");  
 }  
 @Override  
 public void popupMenuCanceled(PopupMenuEvent e) {  
 // TODO Auto-generated method stub  
 //System.out.println("good bye");  
 }  
 });  
  
 // добавление пунктов "Справка" в PopupMenu  
 switch (select) {  
 case "Автобусы":  
 break;  
 case "Водители":  
 break;  
 case "Маршруты":  
 JMenuItem viewSchedule = new JMenuItem("Расписание");  
 viewSchedule.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String num = table.getValueAt(table.getSelectedRow(), 1).toString(),  
 schedule = table.getValueAt(table.getSelectedRow(), 4).toString();  
 new ScheduleViewer(num, schedule);  
 }  
 });  
 this.add(viewSchedule);  
 break;  
 }  
 // добавление пунктов "Нарушений" в PopupMenu  
 ArrayList<JMenuItem> problems\_list = new ArrayList<>();  
 switch (select) {  
 case "Автобусы":  
 problems\_list.add(new JMenuItem("Поломка"));  
 problems\_list.add(new JMenuItem("ДТП"));  
 break;  
 case "Водители":  
 problems\_list.add(new JMenuItem("Прогул"));  
 problems\_list.add(new JMenuItem("Болезнь"));  
 break;  
 case "Маршруты":  
 problems\_list.add(new JMenuItem("Ремонт"));  
 problems\_list.add(new JMenuItem("Крупное ДТП"));  
 break;  
 }  
 if (problems\_list.size() > 0) {  
 for (JMenuItem problem: problems\_list) {  
 this.add(problem);  
 problem.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 // создаём менеджер нарушений  
 ViolationManager vm = new ViolationManager(table, Integer.parseInt((String) table.getValueAt(table.getSelectedRow(), 0)), problem.getText());  
 // вызываем фиксацию нарушения  
 switch (select) {  
 case "Автобусы":  
 vm.applyToBus();  
 break;  
 case "Водители":  
 vm.applyToDriver();  
 break;  
 case "Маршруты":  
 vm.applyToRoute();  
 break;  
 }  
 }  
 });  
 }  
 }  
 }  
}  
*/\*\*  
 \* Класс просмотра расписания маршрута  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*class ScheduleViewer extends JFrame {  
 */\*\*  
 \* Конструктор - создание нового объекта ScheduleViewer  
 \** ***@param*** *num - номер маршрута, для которого вызывается расписание  
 \** ***@param*** *schedule - расписание маршрута  
 \*/* ScheduleViewer(String num, String schedule) {  
 super("Маршрут №" + num);  
 setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);  
 setLocation(150, 150);  
  
 String[] initicalSchedule = schedule.split(";");  
  
 JPanel schedulePanel = new JPanel();  
 schedulePanel.setLayout(new GridLayout(7, 1, 5, 1));  
 // добавляем все элементы на панель в формате сетки 7x1 и заполняем(если данные уже есть)  
 for (int i=0; i<7; i++) {  
 JPanel day = new JPanel();  
 day.setLayout(new FlowLayout(FlowLayout.LEFT));  
 JLabel dayFrom = new JLabel(Days.values()[i] + " с"); // добавляем подпись дня недели  
 // Определение маски и содание поля ввода времени телефона  
 JTextField from = new JTextField(); // ввод времени начала работы  
 JLabel toLabel = new JLabel("до"); // добавляем подпись дня недели  
 JTextField to = new JTextField(); // ввод времени окончания работы  
 JLabel intervalLabel = new JLabel("каждые"); // добавляем подпись дня недели  
 JTextField interval = new JTextField(); // форматированный ввод интервала  
 JLabel minLabel = new JLabel("минут"); // добавляем подпись дня недели  
 // Добавляем все на панель  
 day.add(dayFrom); day.add(from); day.add(toLabel); day.add(to);  
 day.add(intervalLabel); day.add(interval); day.add(minLabel);  
 // заполняем поля для случая с редактированием записи  
 from.setText(initicalSchedule[i].split(" ")[0]);  
 to.setText(initicalSchedule[i].split(" ")[1]);  
 interval.setText(initicalSchedule[i].split(" ")[2]);  
 schedulePanel.add(day);  
 }  
 add(schedulePanel, BorderLayout.CENTER);  
  
 pack();  
 setVisible(true);  
 }  
}

TablePopupMenu.java

package app;  
  
import entitys.\*;  
  
import javax.swing.\*;  
import javax.swing.event.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.util.ArrayList;  
*/\*\*  
 \* Класс поп-ап меню для таблицы  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class TablePopupMenu extends JPopupMenu {  
 */\*\*  
 \* Конструктор - создание нового объекта TablePopupMenu  
 \** ***@param*** *table - таблица данных, для которой создается поп-ап меню  
 \** ***@param*** *select - идентификатор выбранной таблицы(название)  
 \*/* public TablePopupMenu(JTable table, String select) {  
 super();  
 // автоматически выбрать строку, в которой был сделан щелчок правой кнопкой мыши  
 JPopupMenu p = this; // сохранение ссылки для последующей работы  
 this.addPopupMenuListener(new PopupMenuListener() {  
 @Override  
 public void popupMenuWillBecomeVisible(PopupMenuEvent e) {  
 SwingUtilities.*invokeLater*(new Runnable() {  
 @Override  
 public void run() {  
 int rowAtPoint = table.rowAtPoint(SwingUtilities.*convertPoint*(p, new Point(0, 0), table));  
 if (rowAtPoint > -1) {  
 table.setRowSelectionInterval(rowAtPoint, rowAtPoint);  
 }  
 }  
 });  
 }  
 @Override  
 public void popupMenuWillBecomeInvisible(PopupMenuEvent e) {  
 // *TODO Auto-generated method stub* //System.out.println("i will be back");  
 }  
 @Override  
 public void popupMenuCanceled(PopupMenuEvent e) {  
 // *TODO Auto-generated method stub* //System.out.println("good bye");  
 }  
 });  
  
 // добавление пунктов "Справка" в PopupMenu  
 switch (select) {  
 case "Автобусы":  
 break;  
 case "Водители":  
 break;  
 case "Маршруты":  
 JMenuItem viewSchedule = new JMenuItem("Расписание");  
 viewSchedule.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String num = table.getValueAt(table.getSelectedRow(), 1).toString(),  
 schedule = table.getValueAt(table.getSelectedRow(), 4).toString();  
 new ScheduleViewer(num, schedule);  
 }  
 });  
 this.add(viewSchedule);  
 break;  
 }  
 // добавление пунктов "Нарушений" в PopupMenu  
 ArrayList<JMenuItem> problems\_list = new ArrayList<>();  
 switch (select) {  
 case "Автобусы":  
 problems\_list.add(new JMenuItem("Поломка"));  
 problems\_list.add(new JMenuItem("ДТП"));  
 break;  
 case "Водители":  
 problems\_list.add(new JMenuItem("Прогул"));  
 problems\_list.add(new JMenuItem("Болезнь"));  
 break;  
 case "Маршруты":  
 problems\_list.add(new JMenuItem("Ремонт"));  
 problems\_list.add(new JMenuItem("Крупное ДТП"));  
 break;  
 }  
 if (problems\_list.size() > 0) {  
 for (JMenuItem problem: problems\_list) {  
 this.add(problem);  
 problem.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 // создаём менеджер нарушений  
 ViolationManager vm = new ViolationManager(table, Integer.*parseInt*((String) table.getValueAt(table.getSelectedRow(), 0)), problem.getText());  
 // вызываем фиксацию нарушения  
 switch (select) {  
 case "Автобусы":  
 vm.applyToBus();  
 break;  
 case "Водители":  
 vm.applyToDriver();  
 break;  
 case "Маршруты":  
 vm.applyToRoute();  
 break;  
 }  
 }  
 });  
 }  
 }  
 }  
}  
*/\*\*  
 \* Класс просмотра расписания маршрута  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*class ScheduleViewer extends JFrame {  
 */\*\*  
 \* Конструктор - создание нового объекта ScheduleViewer  
 \** ***@param*** *num - номер маршрута, для которого вызывается расписание  
 \** ***@param*** *schedule - расписание маршрута  
 \*/* ScheduleViewer(String num, String schedule) {  
 super("Маршрут №" + num);  
 setDefaultCloseOperation(*DISPOSE\_ON\_CLOSE*);  
 setLocation(150, 150);  
  
 String[] initicalSchedule = schedule.split(";");  
  
 JPanel schedulePanel = new JPanel();  
 schedulePanel.setLayout(new GridLayout(7, 1, 5, 1));  
 // добавляем все элементы на панель в формате сетки 7x1 и заполняем(если данные уже есть)  
 for (int i=0; i<7; i++) {  
 JPanel day = new JPanel();  
 day.setLayout(new FlowLayout(FlowLayout.*LEFT*));  
 JLabel dayFrom = new JLabel(Days.*values*()[i] + " с"); // добавляем подпись дня недели  
 // Определение маски и содание поля ввода времени телефона  
 JTextField from = new JTextField(); // ввод времени начала работы  
 JLabel toLabel = new JLabel("до"); // добавляем подпись дня недели  
 JTextField to = new JTextField(); // ввод времени окончания работы  
 JLabel intervalLabel = new JLabel("каждые"); // добавляем подпись дня недели  
 JTextField interval = new JTextField(); // форматированный ввод интервала  
 JLabel minLabel = new JLabel("минут"); // добавляем подпись дня недели  
 // Добавляем все на панель  
 day.add(dayFrom); day.add(from); day.add(toLabel); day.add(to);  
 day.add(intervalLabel); day.add(interval); day.add(minLabel);  
 // заполняем поля для случая с редактированием записи  
 from.setText(initicalSchedule[i].split(" ")[0]);  
 to.setText(initicalSchedule[i].split(" ")[1]);  
 interval.setText(initicalSchedule[i].split(" ")[2]);  
 schedulePanel.add(day);  
 }  
 add(schedulePanel, BorderLayout.*CENTER*);  
  
 pack();  
 setVisible(true);  
 }  
}

ViolationManager.java

package app;  
  
import entitys.Bus;  
import entitys.Driver;  
import entitys.Route;  
import entitys.Violation;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.util.Date;  
*/\*\*  
 \* Класс менеджера фиксации нарушений  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class ViolationManager {  
 */\*\* Поле со ссылкой на таблицу данных, в которой производится фиксация нарушения \*/* private final JTable table;  
 */\*\* Уникальный идентификатор объекта с нарушением \*/* private final int ID;  
 */\*\* Описание нарушения \*/* private String violation\_description;  
 */\*\*  
 \* Конструктор - создание нового объекта ViolationManager  
 \** ***@param*** *\_table - ссылка на таблицу, где фиксируется нарушение  
 \** ***@param*** *\_ID - уникальный идентификатор объекта с нарушением  
 \** ***@param*** *\_description - описание нарушения  
 \*/* public ViolationManager(JTable \_table, int \_ID, String \_description) {  
 table = \_table;  
 ID = \_ID;  
 violation\_description = \_description;  
 }  
  
 */\*\*  
 \* Функция фиксации нарушения для автобуса  
 \*/* public void applyToBus() {  
 if (!App.*em*.getTransaction().isActive()) App.*em*.getTransaction().begin();  
  
 entitys.Bus model = App.*em*.find(Bus.class, ID);  
 violation\_description = violation\_description + ": " + model.getRegistr();  
 Violation violation = null;  
 if (model.getViolation() != null) {  
 Violation fix = App.em.find(Violation.class, model.getViolation().getId());  
 fix.setFixDate(new Date());  
 if (model.getViolation().getDescription().equals(violation\_description)) violation\_description = "";  
 }  
 if (violation\_description.length() > 0) {  
 violation = new Violation(violation\_description);  
 }  
 model.setViolation(violation);  
  
 App.em.getTransaction().commit();  
 // обновляем данные в таблице приложения  
 DefaultTableModel dtm = ((DefaultTableModel) table.getModel());  
 dtm.setValueAt(violation\_description.length() > 0 ? violation\_description : "Ок", table.getSelectedRow(), table.getColumnCount()-1);  
 dtm.fireTableDataChanged();  
 }  
 */\*\*  
 \* Функция фиксации нарушения для водителя  
 \*/* public void applyToDriver() {  
 if (!App.em.getTransaction().isActive()) App.em.getTransaction().begin();  
  
 Driver model = App.em.find(Driver.class, ID);  
 violation\_description = violation\_description + ": " + model.getName();  
 Violation violation = null;  
 if (model.getViolation() != null) {  
 Violation fix = App.em.find(Violation.class, model.getViolation().getId());  
 fix.setFixDate(new Date());  
 if (model.getViolation().getDescription().equals(violation\_description)) violation\_description = "";  
 }  
 if (violation\_description.length() > 0) {  
 violation = new Violation(violation\_description);  
 }  
 model.setViolation(violation);  
  
 App.em.getTransaction().commit();  
 // обновляем данные в таблице приложения  
 DefaultTableModel dtm = ((DefaultTableModel) table.getModel());  
 dtm.setValueAt(violation\_description.length() > 0 ? violation\_description : "Ок", table.getSelectedRow(), table.getColumnCount()-1);  
 dtm.fireTableDataChanged();  
 }  
 */\*\*  
 \* Функция фиксации нарушения для маршрута  
 \*/* public void applyToRoute() {  
 if (!App.em.getTransaction().isActive()) App.em.getTransaction().begin();  
  
 Route model = App.em.find(Route.class, ID);  
 violation\_description = violation\_description + ": №" + model.getNumber();  
 Violation violation = null;  
 if (model.getViolation() != null) {  
 Violation fix = App.em.find(Violation.class, model.getViolation().getId());  
 fix.setFixDate(new Date());  
 if (model.getViolation().getDescription().equals(violation\_description)) violation\_description = "";  
 }  
 if (violation\_description.length() > 0) {  
 violation = new Violation(violation\_description);  
 }  
 model.setViolation(violation);  
  
 App.em.getTransaction().commit();  
 // обновляем данные в таблице приложения  
 DefaultTableModel dtm = ((DefaultTableModel) table.getModel());  
 dtm.setValueAt(violation\_description.length() > 0 ? violation\_description : "Ок", table.getSelectedRow(), table.getColumnCount()-1);  
 dtm.fireTableDataChanged();  
 }  
}

ViolationTableCellRender.java

package app;  
  
import entitys.\*;  
  
import javax.persistence.EntityManager;  
import javax.swing.\*;  
import javax.swing.table.DefaultTableCellRenderer;  
import java.awt.\*;  
*/\*\*  
 \* Класс рендера таблицы данных  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class ViolationTableCellRender extends DefaultTableCellRenderer {  
 */\*\* Поле, содержащее ссылку на EntityManager \*/* private EntityManager em;  
 */\*\* Поле, содержащее индекс таблицы для рендера \*/* private int index; // для какой из таблиц вызван рендер  
 */\*\*  
 \* Конструктор - создание нового объекта ViolationTableCellRender  
 \** ***@param*** *\_em - ссылка на EntityManager  
 \** ***@param*** *\_index - индекс таблицы для рендера  
 \*/* public ViolationTableCellRender(EntityManager \_em, int \_index) {  
 super();  
 em = \_em;  
 index = \_index;  
 }  
 */\*\*  
 \* Функция выделения строки таблицы в определенный цвет  
 \** ***@param*** *table - таблица  
 \** ***@param*** *value - значение ячейки  
 \** ***@param*** *isSelected - выбор ячейки  
 \** ***@param*** *hasFocus - наведение на ячейку  
 \** ***@param*** *row - номер строки  
 \** ***@param*** *column - номер столбца  
 \*/* @Override  
 public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column)  
 {  
 Component c = super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);  
 switch (index) {  
 case 0: // автобусы  
 Bus bus = em.find(Bus.class, Integer.parseInt((String) table.getValueAt(row, 0)));  
 if (bus.getViolation() != null) {  
 for (int i = 0; i < table.getColumnCount(); i++) {  
 c = super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, i);  
 c.setBackground(Color.RED);  
 }  
 }  
 else if (bus.getDriver() == null || bus.getRoute() == null) c.setBackground(Color.LIGHT\_GRAY);  
 else c.setBackground(Color.WHITE);  
 break;  
 case 1: // водители  
 Driver driver = em.find(Driver.class, Integer.parseInt((String) table.getValueAt(row, 0)));  
 if (driver.getViolation() != null) {  
 for (int i = 0; i < table.getColumnCount(); i++) {  
 c = super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, i);  
 c.setBackground(Color.RED);  
 }  
 }  
 else c.setBackground(Color.WHITE);  
 break;  
 case 2: // маршруты  
 Route route = em.find(Route.class, Integer.parseInt((String) table.getValueAt(row, 0)));  
 if (route.getViolation() != null) {  
 for (int i = 0; i < table.getColumnCount(); i++) {  
 c = super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, i);  
 c.setBackground(Color.RED);  
 }  
 }  
 else c.setBackground(Color.WHITE);  
 break;  
 }  
 return c;  
 }  
}

## Утилиты

WritingError.java

package utils;  
*/\*\*  
 \* Класс ошибки добавления/редактирования записи  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class WritingError extends Exception {  
 */\*\*  
 \* Конструктор объекта WritingError по-умолчанию  
 \*/* public WritingError() {  
 super("Ошибка добавления(редактирования) записи");  
 }  
}

ValodationError.java

package utils;  
  
*/\*\*  
 \* Класс ошибки валидности зазаваемого значения  
 \** ***@author*** *Дмитрий Дементьев 9308  
 \** ***@version*** *0.1  
 \*/*public class ValidationError extends Exception {  
 */\*\* Поле содержащее вариативное сообщение о требуемом формате ввода \*/* private String help\_msg;  
 */\*\*  
 \* Конструктор объекта ValidationError по-умолчанию  
 \*/* public ValidationError() {  
 super("Недопустимое значение для поля");  
 help\_msg = "Некорректное значение поля";  
 }  
 */\*\*  
 \* Конструктор - создание нового объекта ValidationError  
 \** ***@param*** *msg - вариативное сообщение о требуемом формате ввода  
 \*/* public ValidationError(String msg) {  
 super("Недопустимое значение для поля");  
 help\_msg = msg;  
 }  
 */\*\*  
 \* Функция получения информации о требуемом формате ввода  
 \** ***@return*** *сообщение о требуемом формате ввода  
 \*/* public String getHelp() {  
 return help\_msg;  
 }  
}