Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ПРОГРАММА УПРАВЛЕНИЯ РАСПИСАНИЕМ ПОЕЗДОВ

БГУИР КП 1-40 02 01 410 ПЗ

Студент Косяк И.М.

Руководитель Марзалюк А.В.

МИНСК 2022

Учреждение образования

«Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_\_Б. В. Никульшин

(подпись)

–"–––––––––––––––––2022г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Косяку Ивану Михайловичу*

1. Тема проекта *Программа управления расписанием поездов*

2. Срок сдачи студентом законченного проекта *15 декабря 2022 г.*

3. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение. 1. Обзор литературы 2. Структурное проектирование. 3. Функциональное проектирование. 4. Руководство пользователя. Заключение. Список использованных источников. Приложение А. Приложение Б. Приложение В. Приложение Г. Приложение Д. Приложение Е.*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1. Скриншоты работы программы*

*2.**Диаграмма классов*

*3.**Блок-схема алгоритма void List<T>::push(T&)*

*4. Блок-схема алгоритма void AddCargoForm::on\_submit\_clicked()*

*5. Ведомость документов*

6. Консультант по проекту *Марзалюк А.В.*

7. Дата выдачи задания *16 сентября 2022 г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*разделы 1 к 15 октября 2022 г. – 20 %;*

*разделы 2, 3 к 15 ноября 2022 г. – 40 %;*

*разделы 4 к 30 ноября 2022 г. – 20 %;*

*оформление пояснительной записки и графического материала к 15 декабря 2022 г. – 20 %*

*Защита курсового проекта с 19 декабря 2022 г. по 30 декабря 2022 г.*

РУКОВОДИТЕЛЬ– ––––––––– *Марзалюк А.В.*

(подпись)

Задание принял к исполнению –Х *Косяк И.М.*

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ5

1. ОБЗОР ЛИТЕРАТУРЫ6

2. СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ9

3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ15

4. РУКОВОДСВО ПОЛЬЗОВАТЕЛЯ19

ЗАКЛЮЧЕНИЕ21

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 22

ПРИЛОЖЕНИЕ А 23

ПРИЛОЖЕНИЕ Б 58

ПРИЛОЖЕНИЕ В 67

ПРИЛОЖЕНИЕ Г 68

ПРИЛОЖЕНИЕ Д 69

ПРИЛОЖЕНИЕ Е 70

­

**ВВЕДЕНИЕ**

Сегодня чем больше развиваются технологии, тем доступнее становятся разнообразные устройства, упрощающие жизнь людей.

Деятельность железнодорожных предприятий связана с постоянным контролем всех направлений работы и поэтому нуждается в использовании инструментов автоматизированного программного обеспечения, благодаря которым управление компанией станет одновременно более качественным и менее трудоёмким.

Любой транспорт – это время, и оно является ключевым критерием как для людей, непосредственно работающих на железнодорожном предприятии, так и для обычных пассажиров. Руководствуясь этими данными, приложение должно быть комфортным для всех категорий пользователей. Актуальность разработанного приложения обусловлена необходимостью адаптации обычных граждан к использованию железнодорожного транспорта, а также улучшением качества обслуживания и расширением использования данного средства передвижения. Для самих же работников данной отрасли использование приложения будет способствовать упрощению создания расписания и экономии времени при обслуживании поезда.

Целью данной курсовой работы является разработка удобного приложения для преодоления данных проблем.

Приложение используется для управления рейсами железнодорожного транспорта. Добавлены два вида поездов: пассажирские и грузовые. Для пассажира в приложении имеются следующие функции: покупка, возврат, хранение билетов. Для администрации: самостоятельное добавление, удаление, хранение рейсов с выставлением даты, цены, класса, номера, количества свободных мест, пункта назначения, грузоподъёмности.

**1 ОБЗОР ЛИТЕРАТУРЫ**

* 1. **Основные термины**
* Интерфейс – совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя и компьютерной системы
* Слот – это функция, вызываемая в ответ на определенный сигнал.
* Объектно-ориентированное программирование — расширение структурного программирования, в котором основными концепциями являются понятия классов и объектов.
* Класс – это абстракция описывающая методы, свойства, ещё не существующих объектов.
* Объект – конкретное представление абстракции, имеющее свои свойства и методы.
* Методы – это функции, которые могут выполнять какие-либо действия над данными класса.
* Наследование – это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса родителя присваиваются классу-потомку.
  1. **Фреймворк Qt**

Для реализации программы будет использовался фреймворк Qt[2], основанный на языке программирования C++. Для реализации программной логики используется язык C++ по причине высокой производительности и поддержки объектно-ориентированного программирования.

Qt обладает большим количеством классов, которые могут использоваться в процессе разработки программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами, предназначенными для работы с базами данных и XML- документами, удобной системой коммуникации между виджетами приложения с помощью сигналов, слотов и понятной документацией[3], позволяющей за короткий промежуток времени разобраться в устройстве фреймворка. Приложение предполагает дальнейшее использование широким кругом людей, начиная от покупателей билетов и заканчивая работниками железнодорожной отрасли. Им важно удобство и понятность совершаемых в приложении действий, чтобы допустить как можно меньше ошибок и просчётов. поэтому важным преимуществом Qt будет возможность сделать простой и понятный интерфейс, в сравнении с консольным приложением, которое может вызвать трудности в использовании.

* 1. **Обзор аналогов**

Аналогами данной программы пользуются по всему миру в крупных городах. Многие из этих систем автоматизированы и используются вместе с современными технологиями. Различные программы имеют отличный от других интерфейс и функции, которые предназначены для того, чтобы пользователям, равно как и администраторам, было легче контролировать расписание поездов.

Одним из белорусских аналогов является мобильное приложение “БЧ.Мой поезд”[1], логотип которого изображён на рисунке 1.3.1



Рисунок 1.3.1 - логотип мобильного приложения “БЧ.Мой поезд”.

Его функционал более широк, по сравнению с разрабатываемой программой:

* Планирование поездки: получение актуальной информации о расписании поездов и наличии свободных мест в режиме реального времени
* Оформление электронных проездных документов (билетов): оформление билетов всего за несколько шагов
* Личный кабинет: удобное хранение личной информации и билетов
* Управление заказами: контроль заказов в мобильном приложении
* Справочник пассажира: использование необходимой справочной информации
* Онлайн-табло: получение информации о том, с какого пути отправится или куда прибудет поезд
* Языки: русский, белорусский, английский (определяется системными настройками мобильного устройства)

**2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ**

Структурное проектирование понимается как методология построения алгоритмов, программ и систем, в основе которой лежит выявление структуры задачи, определение составляющих компонентов и выделение связей между ними.

Работа программы разбита на четыре основных блока: авторизация пользователя, чтение данных из файла, работа с данными из файла и запись данных в файл. Все блоки между собой связаны.

Блок авторизации пользователя служит для записи и загрузки информации о пользователях. Данный блок представляет каталог, где хранятся файлы для хранения данных. В этом каталоге присутствуют 3 текстовых файла: users.txt, user\_nameP.txt, user\_nameC.txt, в которых хранятся данные об авторизовавшихся пользователях и купленных ими билетах на пассажирские и грузовые поезда. Далее эти данные загружаются в программу, что позволяет пользователю взаимодействовать с ними, используя пользовательский интерфейс.

Блоки чтения данных из файла и работа с данными из файла содержат 2 текстовых файла, в которых хранится информация о грузовых и пассажирских поездах(cargo.txt, passenger.txt). Блоки позволяют просматривать доступные для пользователя поезда, а также производить их добавление и удаление. Когда информация уже загружена из файла, пользователь может взаимодействовать с данными. При добавлении информация, введённая пользователем, проверяется на корректность ввода.

Блок записи данных в файл обеспечивает запись и распределение купленных пользователями билетов по файлам, созданным для каждого пользователя по отдельности. Этот блок предназначен для анализа данных, которые были загружены в файл или были введены пользователем. Этот блок представлен в виде набора элементов пользовательского интерфейса, который связан с блоком авторизации пользователя и блоком работы с данными. Если пользователь собирается приобрести билет, то данная информация начинает обрабатываться. В случае нехватки свободных мест на заявленный поезд, выскочит предупреждающее окно, которое уведомит пользователя об отсутствии билетов.

После всех произведенных действий данные сохраняются в файлы. При возникновении потребности данные в файлах можно изменять в них самих.

Код программы был разбит на файлы для удобства использования и понятности при прочтении.

Описание содержания файлов программы, которые не затрагивают графическую составляющую, приведено ниже:

**2.1. Cargo.h**

Содержит класс Cargo для грузовых поездов, его приватные поля и публичные методы.

Поля:

-timetable: bool\* – расписание поезда

-carryingCapacity: int – грузоподъёмность поезда

Методы:

+Cargo(int, int, int, int, int, bool\*): void – конструктор класса

**+**getCarryingCapacity(): int – геттер для грузоподъёмности

+setCarryingCapacity(int): void – сеттер для грузоподъёмности

+getTimetable(): bool\* – геттер для расписания

+setTimetable(bool\*) – сеттер для расписания

**2.2. clickException.h**

Содержит шаблонный класс ClickException, который унаследован от шаблонного класса Exception, публичные методы.

Методы:

+ClickException(Except, T\*, QString): void – конструктор класса

+show(): void – метод класса для отображения ошибки, унаследованный от класса Exception

**2.3. Exception.h**

Содержит шаблонный класс Exception, класс перечислений Except.

В классе перечислений Except содержатся коды ошибок, а в Exception защищённые поля и методы.

Поля:

#code: Except – код ошибки

#message: QString – сообщение об ошибке

Методы:

#show: void – виртуальный метод класса для отображения ошибки

**2.3. File.h**

Содержит класс File для работы с файлами и его публичные методы.

Методы:

+openCargo(List<Cargo>\*, const char\*) – метод класса для считывания грузовых рейсов с файла

+saveCargo(List<Cargo>\*, const char\*) – метод класса для сохранения грузовых рейсов в файл

+openPassenger(List<Passenger>\*, const char\*) – метод класса для считывания пассажирских рейсов с файла

+savePassenger(List<Passenger>\*, const char\*) – метод класса для сохранения пассажирских рейсов в файл

+openTrip(List<Trip>\*, const char\*) – метод класса для считывания билетов пассажира с файла

+saveTrip(List<Trip>\*, const char\*) – метод класса для сохранения билетов пассажира в файл

+openUser(List<QString>\*, const char\*) – метод класса для считывания пользователей с файла

+saveUser(List<QString>\*, const char\*) – метод класса для сохранения пользователей в файл

**2.4. List.h**

Содержит шаблонный класс контейнера List, шаблонную структуру Node узла контейнера и шаблонный класс Iterator итератора.

Класс List содержит приватные поля и публичные методы, перегруженный оператор.

Поля:

-head: Node<T>\* – указатель на голову списка

-tail: Node<T>\* – указатель на хвост списка

-amount: int – количество элементов списка

Методы и перегрузки операторов:

+List(): void – конструктор класса

+~List(): void – деструктор класса

+size: int –метод класса для получения количества элементов списка

+push(T&): void –метод класса для добавления элемента

+pop(): T –метод класса для удаления элемента

+operator[](int): T& – перегрузка оператора [] для обращения к элементу списка по индексу

+begin(): Node<T>\* –метод для получения головы списка

+end(): Node<T>\* –метод для получения хвоста списка

Структура Node содержит публичные поля.

Поля:

+data: T – поле структуры для хранения данных

+previous: Node<T>\* – указатель на предыдущий элемент

+next: Node<T>\* – указатель на следующий элемент

Класс Iterator содержит приватные поля и публичные методы, перегруженные операторы.

Поля:

-ptr: Node<T>\* – указатель на элемент списка

-index: int – индекс элемента списка

Методы и перегрузки операторов:

+Iterator(): void – конструктор класса

+getPtr: Node<T>\* – геттер для получения указателя на элемент списка

+setPtr(Node<T>\*): void – сеттер для установки указателя на элемент списка

+operator\*(): T& – перегрузка оператора \*

+operator++(): Iterator& – перегрузка оператора ++

+operator--(): Iterator& – перегрузка оператора --

+operator==(const Iterator&): bool – перегрузка оператора ==

+operator!=(const Iterator&): bool – перегрузка оператора !=

+friend operator+(Iterator<T>, int): Iterator<T> – перегрузка оператора +

+friend operator-(Iterator<T>, int): Iterator<T> – перегрузка оператора –

**2.5. Passenger.h**

Содержит класс Passenger для пассажира, его приватные поля и публичные методы.

Поля:

-timetable: bool\* - расписание поезда

-classOfTickets: int – тип билета

Методы:

+Passenger(): void – конструктор класса

+getTimetable(): bool\* – геттер для получения расписания

+setTimetable(bool\*): void – сеттер для установки расписания

+getClassOfTickets(): int – геттер для получения типа билета

+setClassOfTickets(int): void – сеттер для установки типа билета

**2.6. Tickets.h**

Содержит класс Tickets для билетов на поезд, его приватные поля и публичные методы.

Поля:

-id: int – номер рейса

-price: int – цена билета

Методы:

+Tickets(): void – конструктор класса

+getId(): int – геттер для получения номера рейса

+setId(int): void – сеттер для установки номера рейса

+getPrice(): int – геттер для получения цены билета

+setPrice(int): void – сеттер для установки цены билета

**2.7. Trip.h**

Содержит класс Trip для рейсов, его приватные поля и публичные методы.

Поля:

-destination: int – пункт назначения

-numberOfFreeSeats: int – количество свободных мест

Методы:

+Trip(): void – конструктор класса

+getDestination(): int – метод класса для получения пункта назначения

+setDestination(int): void – метод класса для установки пункта назначения

+getNumberOfFreeSeats(): int – метод класса для получения количества свободных мест

+setNumberOfFreeSeats(int): void – метод класса для установки количества свободных мест

Более подробно с классами, методами и полями можно ознакомиться, обратившись к обязательному Приложению А(листинг кода).

# **3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

Для лучшего понимания кода ниже представлены алгоритм по шагам для некоторых методов, диаграмма прецедентов и диаграмма классов в обязательном приложении В.

**3.1 Диаграмма прецедентов**

Диаграмма прецедентов – это тип диаграммы UML, который часто используется для анализа различных систем. Она позволяет визуализировать различные типы ролей в системе и то, как эти роли взаимодействуют с системой[4]. Диаграмма прецедентов для данного курсового проекта представлена на рисунке 3.3.1.

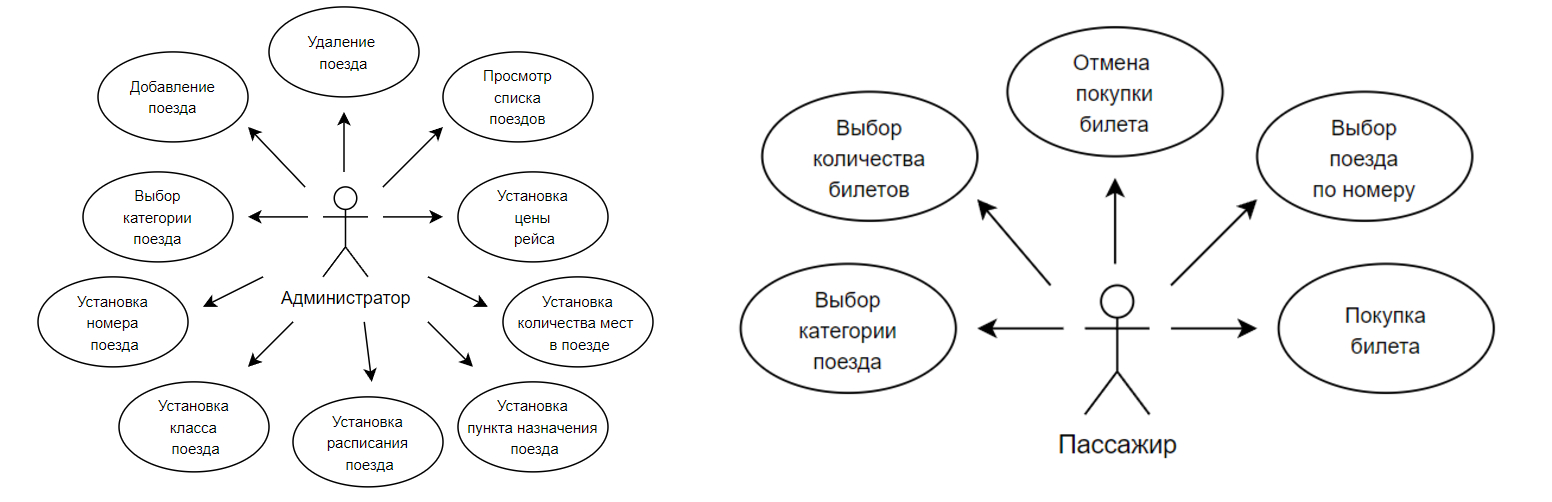


Рисунок 3.1.1 – UML-диаграмма прецедентов

**3.2 Метод void AdminMenuWindow::on\_add\_clicked().**

Этот метод служит для открытия формы с добавлением поезда.

Шаг 1. Если type равно 1, перейти на Шаг 2, иначе на Шаг 5;

Шаг 2. Создать указатель на объект класса AddCargoForm под названием w;

Шаг 3. Выделить память под w и передать в конструктор AddCargoForm указатель на список грузовых поездов listC;

Шаг 4. Сделать виджет видимым;

Шаг 5. Создать указатель на объект класса AddPassengerForm под названием w;

Шаг 6. Выделить память под w и передать в конструктор AddPassengerForm указатель на список пассажирских поездов listP;

Шаг 7. Сделать виджет видимым;

Шаг 8. Конец метода.

**3.3 Метод void AdminMenuWindow::on\_Look\_clicked().**

Этот метод служит для просмотра списка всех поездов.

Шаг 1. Если type равно 1, перейти на Шаг 2, иначе на Шаг 5;

Шаг 2. Создать указатель на объект класса ShowCargoTableWindow под названием w;

Шаг 3. Выделить память под w и передать в конструктор ShowCargoTableWindow указатель на список грузовых поездов listC;

Шаг 4. Сделать виджет видимым, перейти к Шагу 8;

Шаг 5. Создать указатель на объект класса ShowPassengerTableWindow под названием w;

Шаг 6. Выделить память под w и передать в конструктор ShowPassengerTableWindow указатель на список пассажирских поездов listP;

Шаг 7. Сделать виджет видимым;

Шаг 8. Конец метода.

**3.4 Метод void Cargo::setTimetable(bool\* timetable).**

Этот метод служит для установки расписания поездов.

Шаг 1. Выделить память под поле класса Cargo timetable (this->timetable) для 7 элементов типа bool;

Шаг 2. Начать цикл for int i = 0; i < 7; i++;

Шаг 3. Присвоить значение переданной переменной timetable[i] нашему полю класса this->timetable[i];

Шаг 4. Если i < 7, перейти на Шаг 3, иначе на Шаг 5;

Шаг 5. Конец метода.

**3.5 Метод void MainWindow::on\_passengerB\_clicked().**

Этот метод служит для входа в программу в качестве пассажира.

Шаг 1. Если в поле имени пользователя была введена пустая строка, перейти на Шаг 2, иначе на Шаг 3;

Шаг 2. Создать исключение ClickException;

Шаг 3. Полю класса MainWindow id присвоить введённое имя пользователя;

Шаг 4. Создать переменную типа List<QString>\* list;

Шаг 5. Выделить память под переменную list;

Шаг 6. Вызвать метод File::openUser(list, “users.txt”);

Шаг 7. Создать переменную типа bool flag;

Шаг 8. Присвоить flag значение false;

Шаг 9. Цикл for Iterator it = list->begin; it!=list->end + 1; it++;

Шаг 10. Если it.getPtr() равно nullptr, перейти на Шаг 14, иначе на Шаг 11;

Шаг 11. Если (\*it) равно id, перейти на Шаг 12, иначе на Шаг 13;

Шаг 12. Присвоить переменной flag значение true;

Шаг 13. Если it!=list->end + 1, перейти на Шаг 10, иначе на Шаг 14;

Шаг 14. Если flag равно false, перейти на Шаг 15, иначе на Шаг 21;

Шаг 15. Создать переменную str из введённого имени пользователя и “P.txt”;

Шаг 16. Создать и закрыть файл с названием, которое хранится в str;

Шаг 17. Присвоить переменной str значение введённого имени пользователя и “С.txt”;

Шаг 18. Создать и закрыть файл с названием, которое хранится в str;

Шаг 19. С помощью метода push занести имя пользователя в переменную list;

Шаг 20. Вызвать метод File::saveUser(list, “users.txt”);

Шаг 21. Создать переменную типа MenuWindow\* w и выделить память под неё, передавая в конструктор значение цифры 2 и переменной id (new MenuWindow(2, id));

Шаг 22. Сделать виджет видимым;

Шаг 23. Закрыть окно MainWindow;

Шаг 24. Конец метода.

**3.6 Метод T List<T>::pop().**

Этот метод служит для удаления элемента из списка.

Шаг 1. Если указатель на голову head равен nullptr, перейти на Шаг 2, иначе на Шаг 3;

Шаг 2. Вернуть пустой шаблонный объект класса T и перейти на Шаг 10;

Шаг 3. Создать локального объекта data, который принимает значение head->data и указатель node на голову списка;

Шаг 4. Если head не равно tail, перейти на Шаг 5, иначе на Шаг 6;

Шаг 5. Значению head присвоить значение head->next, а head->previous присвоить nullptr;

Шаг 6. Значениям head и tail присвоить nullptr;

Шаг 7. Очистить динамическую память, на которую указывает node;

Шаг 8. Уменьшить amount на 1;

Шаг 9. Вернуть data;

Шаг 10. Конец метода.

# **4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Ниже будут даны описания последовательностей действий, необходимых для успешной работы с программой.

После запуска программы открывается окно с главным меню. Если нужно выполнить вход в программу как пассажир, нужно ввести логин пользователя с помощью поля ввода текста и нажать на кнопку “Войти как пассажир”. Если же планируется вход в качестве администратора, нужно нажать на кнопку “Войти как администратор”. В случае нажатия кнопки “Выход” произойдёт выход из приложения.

После входа в качестве пассажира либо администратора для продолжения работы необходимо выбрать тип поездов и нажать соответствующую кнопку, если планируется вернуться обратно в главное меню, то необходимо нажать кнопку “Назад”.

В режиме администратора после выбора типа поездов откроется окно с выбором действий. При нажатии кнопки “Добавить поезд” откроется форма с добавлением нового поезда. После её корректного заполнения и нажатия кнопки “Добавить” будет выведено сообщение в отдельном окне об успешном добавлении, поезд будет добавлен, форма добавления закроется, и останется предыдущее окно. Если во время заполнения формы возникла необходимость вернуться на предыдущее окно, необходимо нажать кнопку “Назад”. При нажатии кнопки “Удалить поезд” будет удалён последний добавленный поезд, если же поезда отсутствуют, будет выведено сообщение об отсутствии поездов. При нажатии кнопки “Просмотреть список поездов” будет открыто окно со списком всех поездов. Чтобы вернуться назад из этого окна, нужно нажать кнопку “Назад”.

В режиме пассажира после выбора типа поездов откроется окно с выбором действий. При нажатии кнопки “Покупка билета” откроется форма со списком поездов ранее выбранного типа и поля “Номер поезда:”, “Количество билетов:”. После её корректного заполнения и нажатия кнопки “Подтвердить” будет выведено сообщение в отдельном окне об успешной покупке билетов. Для возвращения назад из этого окна, нужно нажать кнопку “Назад”. При нажатии кнопки “Отмена последнего действия”, будет выведено сообщение об отмене покупки билетов, которые были куплены в последний раз, билеты будут возвращены. При отсутствии ранее купленных билетов будет выведено сообщение о невозможности отмены последнего действия.

# **ЗАКЛЮЧЕНИЕ**

В ходе данного курсового проекта было изучено объектно-ориентированное программирование на примере языка C++, а также фреймворк Qt. В конечном итоге была разработана программа для управления расписанием поездов, которая позволит улучшить жизнь и удовлетворить потребности людей, которые были описаны в введении к данной курсовой работе.

Однако данный проект может быть усовершенствован в одном из следующих направлений:

* Улучшение пользовательского интерфейса (корректировка неточностей отображения, подбор цветовой гаммы, создание нескольких цветовых тем)
* добавление новых функций для просмотра, изменения, поиска, сортировки рейсов
* более защищённый вход в программу в качестве пассажира

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] БЧ. Мой поезд [Электронный ресурс]. – Режим доступа:  
https://pass.rw.by/ru/info/mobile\_app. – [Дата доступа] – 10.10.2022

[2] Development Framework for Cross-platform Applications – Qt [Электронный ресурс]. – Режим доступа: https://www.qt.io/product/framework. – [Дата доступа] – 10.10.2022

[3] Справочная документация по Qt [Электронный ресурс]. –   
Режим доступа: http://doc.crossplatform.ru/qt/4.5.0. – [Дата доступа] – 10.10.2022

[4] Учебное пособие по диаграмме прецедентов [Электронный ресурс]. – Режим доступа: https://creately.com/blog/ru/диаграмма/учебное-пособие-по-диаграмма-прецеде. – [Дата доступа] – 10.10.2022

[5] Бьерн Страуструп. Язык программирования С++/ Б.Страуструп; специальное издание. Пер. с англ. — СПб.: BHV, 2008. – 1098 с.

ПРИЛОЖЕНИЕ А

(Обязательное)

Листинг кода

\\ файл “addcargoform.h”

#ifndef ADDCARGOFORM\_H

#define ADDCARGOFORM\_H

#include <QWidget>

#include <QIntValidator>

#include "adminmenuwindow.h"

namespace Ui

{

class AddCargoForm;

}

class AddCargoForm : public QWidget // Класс для добавления грузового поезда

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

explicit AddCargoForm(List<Cargo>\* list); // Конструктор класса

~AddCargoForm(); // Деструктор класса

private slots:

void on\_submit\_clicked(); // Функция-слот для подтверждения добавления грузового поезда

void on\_back\_clicked(); // Функция-слот для возврата назад

private:

Ui::AddCargoForm \*ui; // Указатель на объект класса

List<Cargo>\* list; // Указатель на список грузовых поездов

};

#endif

\\ файл “addpassengerform.h”

#ifndef ADDPASSENGERFORM\_H

#define ADDPASSENGERFORM\_H

#include <QWidget>

#include <QIntValidator>

#include "adminmenuwindow.h"

namespace Ui

{

class AddPassengerForm;

}

class AddPassengerForm : public QWidget // Класс для добавления пассажирского поезда

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

explicit AddPassengerForm(List<Passenger>\* list); // Конструктор класса

~AddPassengerForm(); // Деструктор класса

private slots:

void on\_submit\_clicked(); // Функция-слот для подтверждения добавления грузового поезда

void on\_back\_clicked(); // Функция-слот для возврата назад

private:

Ui::AddPassengerForm \*ui; // Указатель на объект класса

List<Passenger>\* list; // Указатель на список пассажирских поездов

};

#endif

\\ файл “adminmenuwindow.h”

#ifndef ADMINMENUWINDOW\_H

#define ADMINMENUWINDOW\_H

#include <QMainWindow>

#include "File.h"

#include "List.h"

#include "Cargo.h"

#include "Passenger.h"

#include "menuwindow.h"

#include <QMessageBox>

#include "addpassengerform.h"

#include "addcargoform.h"

namespace Ui

{

class AdminMenuWindow;

}

class AdminMenuWindow : public QMainWindow // Класс для меню администратора

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

explicit AdminMenuWindow(int type = 0); // Конструктор класса

~AdminMenuWindow(); // Деструктор класса

private slots:

void on\_back\_clicked(); // Функция-слот для возврата назад

void on\_delete\_2\_clicked(); // Функция-слот для удаления последнего рейса

void on\_add\_clicked(); // Функция-слот для добавления рейсов

void on\_Look\_clicked(); // Функция-слот для просмотра рейсов

private:

Ui::AdminMenuWindow \*ui; // Указатель на объект класса

int type; // Поле класса типа int для выбора типа поездов для операций с ними (пассажирские/грузовые)

List<Passenger>\* listP; // Указатель на список пассажирских поездов

List<Cargo>\* listC; // Указатель на список грузовых поездов

};

#endif

\\ файл “Cargo.h”

#ifndef CARGO\_H

#define CARGO\_H

#include "Trip.h"

class Cargo : public Trip // Класс для грузовых поездов

{

private:

int carryingCapacity; // Поле класса типа int для грузоподъёмности

bool\* timetable; // Поле класса типа bool\* для расписания

public:

explicit Cargo (int id = 0, int price = 0, int destination = 0, int numberOfFreeSeats = 0, // Конструктор класса

int carryingCapacity = 0, bool\* timetable = nullptr);

int getCarryingCapacity() const; // Метод класса для получения грузоподъёмности

void setCarryingCapacity(int carryingCapacity); // Метод класса для установки грузоподъёмности

bool \*getTimetable() const; // Метод класса для получения расписания

void setTimetable(bool \*timetable); // Метод класса для установки расписания

};

#endif

\\ файл “clickException.h”

#ifndef CLICKEXCEPTION\_H

#define CLICKEXCEPTION\_H

#include "Exception.h"

template <class T>

class ClickException : public Exception // Шаблонный класс для исключений при кликах на кнопки в программе (отправление форм, отмена, удаление)

{

T\* window; // Поле шаблонного класса типа T\* для окна

public:

ClickException<T>(Except code, T\* window, QString message); // Конструктор шаблонного класса

void show() override; // Метод класса для отображения ошибки

};

template <class T>

ClickException<T>::ClickException(Except code, T\* window, QString message)

{

this->code = code;

this->window = window;

this->message = message;

}

template <class T>

void ClickException<T>::show()

{

QMessageBox::warning(window, "Ошибка " + QString::number((int)code), message);

}

#endif

\\ файл “Exception.h”

#ifndef EXCEPTION\_H

#define EXCEPTION\_H

#include <QString>

#include <QMessageBox>

enum class Except // Класс для кодов ошибок

{

emptyLines = 100, // Код ошибки при незаполненном поле

inabilityCancelPurchase = 101, // Код ошибки при невозможности отмены

emptyList = 102, // Код ошибки при пустом контейнере

zeroFreeSeats = 103, // Код ошибки при отсутствии свободных мест

zeroPrice = 104, // Код ошибки при неправильном вводе цены

zeroDays = 105, // Код ошибки при неправильном вводе расписания

zeroCapacity = 106, // Код ошибки при неправильном вводе грузоподъёмности

zeroId = 107, // Код ошибки при неправильном вводе номера рейса

zeroTickets = 108 // Код ошибки при неправильном вводе количества билетов

};

class Exception // Класс для исключений

{

protected:

Except code; // Поле класса типа Except для кодов ошибок

QString message; // Поле класса типа QString для сообщения об ошибке

virtual void show() = 0; // Виртуальный метод класса для показа ошибки

};

#endif

\\ файл “File.h”

#ifndef FILE\_H

#define FILE\_H

#include "List.h"

#include "Cargo.h"

#include "Passenger.h"

#include <fstream>

#include <QString>

#include <iomanip>

using namespace std;

class File { // Класс для работы с файлами

public:

static void openCargo(List<Cargo> \*list, const char \*file) { // Метод класса для считывания грузовых рейсов с файла

ifstream in;

in.open(file);

if (!in.is\_open()) {

return;

}

int val1, val2, val3, val4, val5;

while(EOF) {

bool \*val6 = new bool[7];

in >> val1;

if (!in) {

break;

}

in >> val2 >> val3 >> val4 >> val5;

for (int i = 0; i < 7; i++) {

in >> val6[i];

}

Cargo cargo(val1, val2, val3, val4, val5, val6);

list->push(cargo);

}

in.close();

}

static void saveCargo(List<Cargo> \*list, const char \*file) { // Метод класса для сохранения грузовых рейсов в файл

ofstream on;

on.open(file, std::ios\_base::out);

if (!on.is\_open()) {

return;

}

for (Iterator it = list->begin(); it != list->end() + 1; ++it) {

if(it.getPtr() == nullptr)

{

break;

}

on << setw(20) << (\*it).getId() << setw(20) << (\*it).getPrice() <<

setw(20) << (\*it).getDestination() << setw(20) << (\*it).getNumberOfFreeSeats() <<

setw(20) << (\*it).getCarryingCapacity();

for (int j = 0; j < 7; j++) {

on << " " << (\*it).getTimetable()[j];

}

on << std::endl;

}

on << "%";

on.close();

}

static void openPassenger(List<Passenger> \*list, const char \*file) { // Метод класса для считывания пассажирских рейсов с файла

ifstream in;

in.open(file);

if (!in.is\_open()) {

return;

}

int val1, val2, val3, val4, val5;

while(EOF) {

bool \*val6 = new bool[7];

in >> val1;

if (!in) {

break;

}

in >> val2 >> val3 >> val4;

for (int i = 0; i < 7; i++) {

in >> val6[i];

}

in >> val5;

Passenger passenger(val1, val2, val3, val4, val5, val6);

list->push(passenger);

}

in.close();

}

static void savePassenger(List<Passenger> \*list, const char \*file) { // Метод класса для сохранения пассажирских рейсов в файл

ofstream on;

on.open(file, std::ios\_base::out);

if (!on.is\_open()) {

return;

}

for (Iterator it = list->begin(); it != list->end() + 1; ++it) {

if(it.getPtr() == nullptr)

{

break;

}

on << setw(20) << (\*it).getId() << setw(20) << (\*it).getPrice() <<

setw(20) << (\*it).getDestination() << setw(20) << (\*it).getNumberOfFreeSeats();

for (int j = 0; j < 7; j++) {

on << " " << (\*it).getTimetable()[j];

}

on << setw(20) << (\*it).getClassOfTickets();

on << std::endl;

}

on << "%";

on.close();

}

static void openTrip(List<Trip> \*list, const char \*file) { // Метод класса для считывания билетов пассажира с файла

ifstream in;

in.open(file);

if (!in.is\_open()) {

return;

}

int val1, val2, val3, val4;

while(EOF) {

in >> val1;

if (!in) {

break;

}

in >> val2 >> val3 >> val4;

Trip trip(val1, val2, val3, val4);

list->push(trip);

}

in.close();

}

static void saveTrip(List<Trip> \*list, const char \*file) { // Метод класса для сохранения билетов пассажира в файл

ofstream on;

on.open(file, std::ios\_base::out);

if (!on.is\_open()) {

return;

}

for (Iterator it = list->end(); it != list->begin() - 1; --it)

{

if(it.getPtr() == nullptr)

{

break;

}

on << setw(20) << (\*it).getId() << setw(20) << (\*it).getPrice() <<

setw(20) << (\*it).getDestination() << setw(20) << (\*it).getNumberOfFreeSeats();

on << std::endl;

}

on << "%";

on.close();

}

static void openUser(List<QString> \*list, const char \*file) { // Метод класса для считывания пользователей с файла

ifstream in;

in.open(file);

if (!in.is\_open()) {

return;

}

string val;

while(EOF) {

in >> val;

if (!in || val == "%") {

break;

}

in.get();

QString std = QString::fromStdString(val);

list->push(std);

}

in.close();

}

static void saveUser(List<QString> \*list, const char \*file) { // Метод класса для сохранения пользователей в файл

ofstream on;

on.open(file, std::ios\_base::out);

if (!on.is\_open()) {

return;

}

for (int i = 0; i < list->size(); i++) {

on << list->operator[](i).toStdString();

on << std::endl;

}

on << "%";

on.close();

}

};

#endif

\\ файл “List.h”

#ifndef LIST\_H

#define LIST\_H

template <class T>

struct Node // Шаблонная структура узла контейнера List

{

T data; // Поле структуры типа T для хранения данных

Node<T>\* previous; // Поле структуры типа Node<T>\* для указателя на предыдущий элемент

Node<T>\* next; // Поле структуры типа Node<T>\* для указателя на следующий элемент

};

template <class T>

class List // Шаблонный класс для контейнера List (список)

{

private:

Node<T> \*head = nullptr; // Указатель на голову списка

Node<T> \*tail = nullptr; // Указатель на хвост списка

int amount = 0; // Количество элементов списка

public:

List() {}; // Конструктор класса

~List(); // Деструктор класса

int size(); // Шаблонный метод класса для получения количества элементов списка

void push(T&); // Шаблонный метод класса для добавления элемента

T pop(); // Шаблонный метод класса для удаления элемента

T& operator[](int); // Перегрузка оператора [] для обращения к элементу списка по индексу

Node<T>\* begin(); // Шаблонный метод для получения головы списка

Node<T>\* end(); // Шаблонный метод для получения хвоста списка

};

template<class T>

void List<T>::push(T& obj)

{

if (head == nullptr)

{

head = new Node<T>;

head->data = obj;

head->next = nullptr;

head->previous = nullptr;

tail = head;

amount++;

}

else

{

Node<T> \*node = new Node<T>;

node->data = obj;

node->next = head;

node->previous = nullptr;

head->previous = node;

head = node;

amount++;

}

}

template<class T>

int List<T>::size(){

return this->amount;

}

template<class T>

T List<T>::pop()

{

if (!(head)) return T();

T data = head->data;

Node<T>\* node = head;

if (head != tail)

{

head = head->next;

head->previous = nullptr;

}

else

{

head = tail = nullptr;

}

delete node;

amount--;

return data;

}

template<class T>

T& List<T>::operator[](int num)

{

Node<T> \*curr = head;

if (num < 0 || num >= amount) return curr->data;

for (int i = 0; i < num; i++)

{

curr = curr->next;

}

return curr->data;

}

template<class T>

Node<T>\* List<T>::begin()

{

return head;

}

template<class T>

Node<T>\* List<T>::end()

{

return tail;

}

template<class T>

List<T>::~List()

{

while (this->head)

{

this->pop();

}

}

template <class T>

class Iterator // Шаблонный класс для итератора

{

Node<T>\* ptr; // Поле класса типа Node<T>\* для указателя на элемент списка

int index = 0; // Поле класса типа int для индекса

public:

Iterator (Node<T>\* ptr); // Конструктор класса

Node<T>\* getPtr(); // Метод класса для получения указателя на элемент списка

void setPtr(Node<T>\* newPtr); // Метод класса для установки указателя на элемент списка

T& operator\*() const; // Перегрузка оператора \*

Iterator& operator++(); // Перегрузка оператора ++

Iterator& operator--(); // Перегрузка оператора --

bool operator==(const Iterator& iter) const; // Перегрузка оператора ==

bool operator!=(const Iterator& iter) const; // Перегрузка оператора !=

friend Iterator<T> operator+(Iterator<T> iter, int shift) // Перегрузка оператора +

{

for (int i = 0; i < shift; i++)

++iter;

return iter;

}

friend Iterator<T> operator-(Iterator<T> iter, int shift) // Перегрузка оператора -

{

for (int i = 0; i < shift; i++)

--iter;

return iter;

}

};

template <class T>

Iterator<T>::Iterator(Node<T> \*ptr): ptr(ptr){}

template<class T>

Node<T>\* Iterator<T>::getPtr()

{

return ptr;

}

template<class T>

void Iterator<T>::setPtr(Node<T> \*newPtr)

{

ptr = newPtr;

}

template <class T>

bool Iterator<T>::operator==(const Iterator& iter) const

{

return ptr == iter.ptr;

}

template <class T>

bool Iterator<T>::operator!=(const Iterator& iter) const

{

return ptr != iter.ptr;

}

template <class T>

T& Iterator<T>::operator\*() const

{

return ptr->data;

}

template <class T>

Iterator<T>& Iterator<T>::operator++()

{

if (!ptr)

{

return \*this;

}

ptr=ptr->next;

return \*this;

}

template <class T>

Iterator<T>& Iterator<T>::operator--()

{

ptr= ptr->previous;

return \*this;

}

#endif

\\ файл “mainwindow.h”

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "menuwindow.h"

namespace Ui

{

class MainWindow;

}

class MainWindow : public QMainWindow // Класс для главного окна программы

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

MainWindow(QWidget \*parent = nullptr); // Конструктор класса

~MainWindow(); // Деструктор класса

private slots:

void on\_exit\_clicked(); // Функция-слот для выхода из программы

void on\_passengerB\_clicked(); // Функция-слот для входа пользователя

void on\_admin\_clicked(); // Функция-слот для входа администратора

private:

Ui::MainWindow \*ui; // Указатель на объект класса

QString id; // Поле класса типа QString для логина пассажира

};

#endif

\\ файл “menuwindow.h”

#ifndef MENUWINDOW\_H

#define MENUWINDOW\_H

#include <QMainWindow>

#include "mainwindow.h"

#include "adminmenuwindow.h"

#include "passengermenuwindow.h"

namespace Ui

{

class MenuWindow;

}

class MenuWindow : public QMainWindow // Класс для меню программы (выбор между пассажирскими и грузовыми поездами)

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

explicit MenuWindow(int variant = 0, QString id = ""); // Конструктор класса

~MenuWindow(); // Деструктор класса

private slots:

void on\_back\_clicked(); // Функция-слот для возврата назад

void on\_cargo\_clicked(); // Функция-слот для перехода к грузовым поездам

void on\_passenger\_clicked(); // Функция-слот для перехода к пассажирским поездам

private:

Ui::MenuWindow \*ui; // Указатель на объект класса

int variant; // Поле класса типа int для отображения меню разного типа (пассажирские/грузовые)

QString id; // Поле класса типа QString для логина пассажира

};

#endif

\\ файл “Passenger.h”

#ifndef PASSENGER\_H

#define PASSENGER\_H

#include "Trip.h"

class Passenger: public Trip // Класс для пассажира

{

private:

bool\* timetable; // Поле класса типа bool\* для расписания

int classOfTickets; // Поле класса типа int для типа билетов пассажиров

public:

explicit Passenger (int id = 0, int price = 0, int destination = 0, int numberOfFreeSeats = 0, // Конструктор класса

int classOfTickets = 0, bool\* timetable=nullptr);

bool\* getTimetable() const; // Метод класса для получения расписания

void setTimetable(bool\* timetable); // Метод класса для установки расписания

int getClassOfTickets() const; // Метод класса для получения типа билетов

void setClassOfTickets(int classOfTickets); // Метод класса для установки типа билетов

};

#endif

\\ файл “passengermenuwindow.h”

#ifndef PASSENGERMENUWINDOW\_H

#define PASSENGERMENUWINDOW\_H

#include <QMainWindow>

#include "menuwindow.h"

namespace Ui

{

class PassengerMenuWindow;

}

class PassengerMenuWindow : public QMainWindow // Класс для меню пассажира (покупка билета, отмена последнего действия)

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

explicit PassengerMenuWindow(int variant = 0, QString id = ""); // Конструктор класса

~PassengerMenuWindow(); // Деструктор класса

private slots:

void on\_back\_clicked(); // Функция-слот для возврата назад

void on\_takeTicket\_clicked(); // Функция-слот для покупки билета

void on\_cancel\_clicked(); // Функция-слот для отмены последнего действия

private:

Ui::PassengerMenuWindow \*ui; // Указатель на объект класса

int variant; // Поле класса типа int для отображения формы покупки разного типа (пассажирские/грузовые)

QString id; // Поле класса типа QString для логина пассажира

List<Passenger>\* listP; // Указатель на список пассажирских поездов

List<Cargo>\* listC; // Указатель на список грузовых поездов

};

#endif

\\ файл “showcargotablewindow.h”

#ifndef SHOWCARGOTABLEWINDOW\_H

#define SHOWCARGOTABLEWINDOW\_H

#include <QWidget>

#include "adminmenuwindow.h"

namespace Ui

{

class ShowCargoTableWindow;

}

class ShowCargoTableWindow : public QWidget // Класс для показа таблицы грузовых поездов

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

explicit ShowCargoTableWindow(List<Cargo>\*list); // Конструктор класса

~ShowCargoTableWindow(); // Деструктор класса

private slots:

void on\_back\_clicked(); // Функция-слот для возврата назад

private:

Ui::ShowCargoTableWindow \*ui; // Указатель на объект класса

};

#endif

\\ файл “showpassengertablewindow.h”

#ifndef SHOWPASSENGERTABLEWINDOW\_H

#define SHOWPASSENGERTABLEWINDOW\_H

#include <QWidget>

#include "adminmenuwindow.h"

namespace Ui

{

class ShowPassengerTableWindow;

}

class ShowPassengerTableWindow : public QWidget // Класс для показа таблицы пассажирских поездов

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

explicit ShowPassengerTableWindow(List<Passenger>\*list); // Конструктор класса

~ShowPassengerTableWindow(); // Деструктор класса

private slots:

void on\_back\_clicked(); // Функция-слот для возврата назад

private:

Ui::ShowPassengerTableWindow \*ui; // Указатель на объект класса

};

#endif

\\ файл “takecargoticketform.h”

#ifndef TAKECARGOTICKETFORM\_H

#define TAKECARGOTICKETFORM\_H

#include <QWidget>

#include "adminmenuwindow.h"

namespace Ui

{

class TakeCargoTicketForm;

}

class TakeCargoTicketForm : public QWidget // Класс для покупки билетов на грузовой поезд

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

explicit TakeCargoTicketForm(List<Cargo>\*list = nullptr, QString id = ""); // Конструктор класса

~TakeCargoTicketForm(); // Деструктор класса

private slots:

void on\_back\_clicked(); // Функция-слот для возврата назад

void on\_submit\_clicked(); // Функция-слот для подтверждения покупки билета грузового поезда

private:

Ui::TakeCargoTicketForm \*ui; // Указатель на объект класса

List<Cargo> \*list; // Указатель на список грузовых поездов

QString id; // Поле класса типа QString для логина пассажира

};

#endif

\\ файл “takepassengerticketform.h”

#ifndef TAKEPASSENGERTICKETFORM\_H

#define TAKEPASSENGERTICKETFORM\_H

#include <QWidget>

#include "adminmenuwindow.h"

namespace Ui

{

class TakePassengerTicketForm;

}

class TakePassengerTicketForm : public QWidget // Класс для покупки билетов на пассажирский поезд

{

Q\_OBJECT // Базовый класс для всех объектов Qt

public:

explicit TakePassengerTicketForm(List<Passenger>\*list = nullptr, QString id = ""); // Конструктор класса

~TakePassengerTicketForm(); // Деструктор класса

private slots:

void on\_back\_clicked(); // Функция-слот для возврата назад

void on\_submit\_clicked(); // Функция-слот для подтверждения покупки билета грузового поезда

private:

Ui::TakePassengerTicketForm \*ui; // Указатель на объект класса

List<Passenger>\*list; // Указатель на список пассажирских поездов

QString id; // Поле класса типа QString для логина пассажира

};

#endif

\\ файл “Tickets.h”

#ifndef TICKETS\_H

#define TICKETS\_H

class Tickets // Класс для билетов на поезд

{

protected:

int id; // Поле класса типа int для номера рейса

int price; // Поле класса типа int для цены рейса

public:

explicit Tickets(int id = 0, int price = 0); // Конструктор класса

int getId() const; // Метод класса для получения номера рейса

void setId(int id); // Метод класса для установки номера рейса

int getPrice() const; // Метод класса для получения цены рейса

void setPrice(int price); // Метод класса для установки цены рейса

};

#endif

\\ файл “Trip.h”

#ifndef TRIP\_H

#define TRIP\_H

#include "Tickets.h"

class Trip : public Tickets // Класс для рейсов

{

protected:

int destination; // Поле класса типа int для пункта назначения

int numberOfFreeSeats; // Поле класса типа int для количества свободных мест

public:

explicit Trip (int id = 0, int price = 0, int destination = 0, int numberOfFreeSeats = 0); // Конструктор класса

int getDestination() const; // Метод класса для получения пункта назначения

void setDestination(int destination); // Метод класса для установки пункта назначения

int getNumberOfFreeSeats() const; // Метод класса для получения количества свободных мест

void setNumberOfFreeSeats(int numberOfFreeSeats); // Метод класса для установки количества свободных мест

};

#endif

\\ файл “addcargoform.cpp”

#include "addcargoform.h"

#include "ui\_addcargoform.h"

#include "clickException.h"

AddCargoForm::AddCargoForm(List<Cargo>\*list) : list(list), ui(new Ui::AddCargoForm)

{

ui->setupUi(this);

ui->destination->setMinimum(1);

ui->destination->setMaximum(5);

}

AddCargoForm::~AddCargoForm()

{

delete ui;

}

void AddCargoForm::on\_submit\_clicked()

{

try{

bool \*timetable = new bool[7];

if (ui->monday->isChecked() == false &&

ui->tuesday->isChecked() == false &&

ui->wednesday->isChecked() == false &&

ui->thurtday->isChecked() == false &&

ui->friday->isChecked() == false &&

ui->saturday->isChecked() == false &&

ui->sunday->isChecked() == false)

{

throw ClickException(Except::zeroDays, this, "Поезд должен ходить хотя бы в один из дней недели!");

return;

}

timetable[0] = ui->monday->isChecked();

timetable[1] = ui->tuesday->isChecked();

timetable[2] = ui->wednesday->isChecked();

timetable[3] = ui->thurtday->isChecked();

timetable[4] = ui->friday->isChecked();

timetable[5] = ui->saturday->isChecked();

timetable[6] = ui->sunday->isChecked();

if (ui->numberOfSeats->text().toInt() == 0)

{

throw ClickException(Except::zeroFreeSeats, this, "Введите правильно количество свободных мест(более 0)!");

return;

}

int freeSeats = ui->numberOfSeats->text().toInt();

if (ui->id->text().toInt() == 0)

{

throw ClickException(Except::zeroId, this, "Введите правильно номер поезда(более 0)!");

return;

}

int id = ui->id->text().toInt();

int destination = ui->destination->text().toInt();

if (ui->capacity->text().toInt() == 0)

{

throw ClickException(Except::zeroCapacity, this, "Введите правильную грузоподъёмность поезда(более 0)!");

return;

}

int capacity = ui->capacity->text().toInt();

if (ui->price->text().toInt() == 0)

{

throw ClickException(Except::zeroPrice, this, "Введите правильную цену для рейса(более 0)!");

return;

}

int price = ui->price->text().toInt();

Cargo \*cargo = new Cargo(id, price, destination, freeSeats, capacity, timetable);

list->push(\*cargo);

File::saveCargo(list, "cargo.txt");

QMessageBox::information(nullptr, "Добавление грузового поезда", "Грузовой поезд успешно добавлен!");

this->close();

delete this;

}

catch(ClickException<AddCargoForm> &obj)

{

obj.show();

return;

}

}

void AddCargoForm::on\_back\_clicked()

{

this->close();

delete this;

}

\\ файл “addpassengerform.cpp”

#include "addpassengerform.h"

#include "ui\_addpassengerform.h"

#include "clickException.h"

AddPassengerForm::AddPassengerForm(List<Passenger>\* list) : list(list), ui(new Ui::AddPassengerForm)

{

ui->setupUi(this);

ui->destination->setMinimum(1);

ui->destination->setMaximum(5);

ui->class\_2->setMinimum(1);

ui->class\_2->setMaximum(3);

}

AddPassengerForm::~AddPassengerForm()

{

delete ui;

}

void AddPassengerForm::on\_submit\_clicked()

{

try

{

bool\*timetable = new bool[7];

if (ui->monday->isChecked() == false &&

ui->tuesday->isChecked() == false &&

ui->wednesday->isChecked() == false &&

ui->thurtday->isChecked() == false &&

ui->friday->isChecked() == false &&

ui->saturday->isChecked() == false &&

ui->sunday->isChecked() == false)

{

throw ClickException(Except::zeroDays, this, "Поезд должен ходить хотя бы в один из дней недели!");

return;

}

timetable[0] = ui->monday->isChecked();

timetable[1] = ui->tuesday->isChecked();

timetable[2] = ui->wednesday->isChecked();

timetable[3] = ui->thurtday->isChecked();

timetable[4] = ui->friday->isChecked();

timetable[5] = ui->saturday->isChecked();

timetable[6] = ui->sunday->isChecked();

if (ui->numberOfSeats->text().toInt() == 0)

{

throw ClickException(Except::zeroFreeSeats, this, "Введите правильно количество свободных мест(более 0)!");

return;

}

int freeSeats = ui->numberOfSeats->text().toInt();

int id = ui->id->text().toInt();

if (ui->id->text().toInt() == 0)

{

throw ClickException(Except::zeroId, this, "Введите правильно номер поезда(более 0)!");

return;

}

int destination = ui->destination->text().toInt();

int Class = ui->class\_2->text().toInt();

if (ui->price->text().toInt() == 0)

{

throw ClickException(Except::zeroPrice, this, "Введите правильную цену для рейса(более 0)!");

return;

}

int price = ui->price->text().toInt();

Passenger \*passenger = new Passenger(id, price, destination, freeSeats, Class, timetable);

list->push(\*passenger);

File::savePassenger(list, "passenger.txt");

QMessageBox::information(nullptr, "Добавление пассажирского поезда", "Пассажирский поезд успешно добавлен!");

this->close();

delete this;

}

catch(ClickException<AddPassengerForm> &obj)

{

obj.show();

return;

}

}

void AddPassengerForm::on\_back\_clicked()

{

this->close();

delete this;

}

\\ файл “adminmenuwindow.cpp”

#include "adminmenuwindow.h"

#include "ui\_adminmenuwindow.h"

#include "showcargotablewindow.h"

#include "showpassengertablewindow.h"

#include "clickException.h"

AdminMenuWindow::AdminMenuWindow(int type) : ui(new Ui::AdminMenuWindow), type(type)

{

ui->setupUi(this);

listC = new List<Cargo>();

listP = new List<Passenger>();

File::openCargo(listC, "cargo.txt");

File::openPassenger(listP, "passenger.txt");

}

AdminMenuWindow::~AdminMenuWindow()

{

delete ui;

}

void AdminMenuWindow::on\_back\_clicked()

{

File::saveCargo(listC, "cargo.txt");

File::savePassenger(listP, "passenger.txt");

MenuWindow \*w = new MenuWindow(1);

w->setVisible(true);

this->close();

delete this;

}

void AdminMenuWindow::on\_delete\_2\_clicked()

{

if (type == 1)

{

try

{

if (listC->size() < 1)

{

throw ClickException(Except::emptyList, this, "Поезда отсутствуют!");

}

else

{

listC->pop();

File::saveCargo(listC, "cargo.txt");

QMessageBox::information(nullptr, "Удаление грузового поезда", "Вы удалили поезд.");

}

}

catch(ClickException<AdminMenuWindow> &obj)

{

obj.show();

return;

}

}

else

{

try

{

if (listP->size() < 1)

{

throw ClickException(Except::emptyList, this, "Поезда отсутствуют!");

}

else

{

listP->pop();

File::savePassenger(listP, "passenger.txt");

QMessageBox::information(nullptr, "Удаление пассажирского поезда", "Вы удалили поезд.");

}

}

catch(ClickException<AdminMenuWindow> &obj)

{

obj.show();

return;

}

}

}

void AdminMenuWindow::on\_add\_clicked()

{

if (type == 1)

{

AddCargoForm \*w = new AddCargoForm(listC);

w->setVisible(true);

}

else

{

AddPassengerForm\* w = new AddPassengerForm(listP);

w->setVisible(true);

}

}

void AdminMenuWindow::on\_Look\_clicked()

{

if (type == 1)

{

ShowCargoTableWindow \*w = new ShowCargoTableWindow(listC);

w->setVisible(true);

}

else

{

ShowPassengerTableWindow\*w = new ShowPassengerTableWindow(listP);

w->setVisible(true);

}

}

\\ файл “Cargo.cpp”

#include "Cargo.h"

Cargo::Cargo(int id, int price, int destination, int numberOfFreeSeats, int carryingCapacity,

bool\* timetable) : Trip(id, price, destination, numberOfFreeSeats), carryingCapacity(carryingCapacity), timetable(timetable) {}

int Cargo::getCarryingCapacity() const

{

return carryingCapacity;

}

void Cargo::setCarryingCapacity(int carryingCapacity)

{

Cargo::carryingCapacity = carryingCapacity;

}

bool \*Cargo::getTimetable() const

{

return timetable;

}

void Cargo::setTimetable(bool \*timetable)

{

this->timetable = new bool[7];

for (int i = 0; i < 7; i++)

{

this->timetable[i] = timetable[i];

}

}

\\ файл “main.cpp”

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow \*w = new MainWindow;

w->show();

return a.exec();

}

\\ файл “mainwindow.cpp”

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "clickException.h"

MainWindow::MainWindow(QWidget \*parent) : QMainWindow(parent), ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_exit\_clicked()

{

QApplication::exit(0);

}

void MainWindow::on\_passengerB\_clicked()

{

try

{

if(ui->passenger->text() == "")

{

throw ClickException(Except::emptyLines, this, "Введите имя пользователя!");

}

id = ui->passenger->text();

List<QString> \* list = new List<QString>();

File::openUser(list, "users.txt");

bool flag = false;

for (Iterator it = list->begin(); it != list->end() + 1; ++it)

{

if(it.getPtr() == nullptr)

{

break;

}

if ((\*it) == id)

{

flag = true;

}

}

if (!flag)

{

string str = id.toStdString() + "P.txt";

ofstream of(str.c\_str());

of.close();

str = id.toStdString() + "C.txt";

ofstream on(str.c\_str());

on.close();

list->push(id);

File::saveUser(list, "users.txt");

}

MenuWindow \*w = new MenuWindow(2, id);

w->setVisible(true);

this->close();

delete this;

}

catch(ClickException<MainWindow> &obj)

{

obj.show();

return;

}

}

void MainWindow::on\_admin\_clicked()

{

id = "";

MenuWindow \*w = new MenuWindow(1, "");

w->setVisible(true);

this->close();

delete this;

}

\\ файл “menuwindow.cpp”

#include "menuwindow.h"

#include "ui\_menuwindow.h"

MenuWindow::MenuWindow(int variant, QString id) : variant(variant), id(id), ui(new Ui::MenuWindow)

{

ui->setupUi(this);

}

MenuWindow::~MenuWindow()

{

delete ui;

}

void MenuWindow::on\_back\_clicked()

{

MainWindow \*w = new MainWindow;

w->setVisible(true);

this->close();

delete this;

}

void MenuWindow::on\_cargo\_clicked()

{

if (variant == 1)

{

AdminMenuWindow \*w = new AdminMenuWindow(1);

w->setVisible(true);

}

else

{

PassengerMenuWindow \*w = new PassengerMenuWindow(1, id);

w->setVisible(true);

}

this->close();

delete this;

}

void MenuWindow::on\_passenger\_clicked()

{

if (variant == 1)

{

AdminMenuWindow \*w = new AdminMenuWindow(2);

w->setVisible(true);

}

else

{

PassengerMenuWindow \*w = new PassengerMenuWindow(2, id);

w->setVisible(true);

}

this->close();

delete this;

}

\\ файл “Passenger.cpp”

#include "Passenger.h"

Passenger::Passenger(int id, int price, int destination, int numberOfFreeSeats,

int classOfTickets, bool\* timetable) : Trip(id, price, destination, numberOfFreeSeats), classOfTickets(classOfTickets), timetable(timetable) {}

bool\* Passenger::getTimetable() const

{

return timetable;

}

void Passenger::setTimetable(bool\* timetable)

{

this->timetable = new bool[7];

for (int i = 0; i < 7; i++)

{

this->timetable[i] = timetable[i];

}

}

int Passenger::getClassOfTickets() const

{

return classOfTickets;

}

void Passenger::setClassOfTickets(int classOfTickets)

{

Passenger::classOfTickets = classOfTickets;

}

\\ файл “passengermenuwindow.cpp”

#include "passengermenuwindow.h"

#include "ui\_passengermenuwindow.h"

#include "takepassengerticketform.h"

#include "takecargoticketform.h"

#include "clickException.h"

PassengerMenuWindow::PassengerMenuWindow(int variant, QString id) : ui(new Ui::PassengerMenuWindow), variant(variant), id(id)

{

ui->setupUi(this);

listC = new List<Cargo>();

listP = new List<Passenger>();

File::openCargo(listC, "cargo.txt");

File::openPassenger(listP, "passenger.txt");

}

PassengerMenuWindow::~PassengerMenuWindow()

{

delete ui;

}

void PassengerMenuWindow::on\_back\_clicked()

{

MenuWindow \*w = new MenuWindow(2,id);

w->setVisible(true);

this->close();

delete this;

}

void PassengerMenuWindow::on\_takeTicket\_clicked()

{

if (variant != 1)

{

TakePassengerTicketForm\*w = new TakePassengerTicketForm(listP, id);

w->setVisible(true);

}

else

{

TakeCargoTicketForm\*w = new TakeCargoTicketForm(listC, id);

w->setVisible(true);

}

}

void PassengerMenuWindow::on\_cancel\_clicked()

{

if (variant != 1)

{

try

{

List<Trip> \*trips = new List<Trip>();

string str = id.toStdString() + "P.txt";

File::openTrip(trips, str.c\_str());

if (trips->size() < 1)

{

throw ClickException(Except::inabilityCancelPurchase, this, "Невозможно отменить покупку!");

return;

}

Trip trip = trips->pop();

File::saveTrip(trips, str.c\_str());

QMessageBox::information(nullptr, "Отмена покупки", "Вы отменили покупку билета!");

for (Iterator it = listP->begin(); it != listP->end() + 1; ++it)

{

if(it.getPtr() == nullptr)

{

break;

}

if ((\*it).getId() == trip.getId())

{

(\*it).setNumberOfFreeSeats((\*it).getNumberOfFreeSeats() + trip.getNumberOfFreeSeats());

File::savePassenger(listP, "passenger.txt");

}

}

}

catch(ClickException<PassengerMenuWindow> &obj)

{

obj.show();

return;

}

}

else

{

try

{

List<Trip> \*trips = new List<Trip>();

string str = id.toStdString() + "C.txt";

File::openTrip(trips, str.c\_str());

if (trips->size() < 1)

{

throw ClickException(Except::inabilityCancelPurchase, this, "Невозможно отменить покупку!");

return;

}

Trip trip = trips->pop();

File::saveTrip(trips, str.c\_str());

QMessageBox::information(nullptr, "Отмена покупки", "Вы отменили покупку билета!");

for (Iterator it = listC->begin(); it != listC->end() + 1; ++it)

{

if(it.getPtr() == nullptr)

{

break;

}

if ((\*it).getId() == trip.getId())

{

(\*it).setNumberOfFreeSeats((\*it).getNumberOfFreeSeats() + trip.getNumberOfFreeSeats());

File::saveCargo(listC, "cargo.txt");

}

}

}

catch(ClickException<PassengerMenuWindow> &obj)

{

obj.show();

return;

}

}

}

\\ файл “showcargotablewindow.cpp”

#include "showcargotablewindow.h"

#include "ui\_showcargotablewindow.h"

ShowCargoTableWindow::ShowCargoTableWindow(List<Cargo>\* list) : ui(new Ui::ShowCargoTableWindow)

{

ui->setupUi(this);

ui->tableWidget->setColumnCount(12);

QTableWidgetItem \*w = new QTableWidgetItem("Номер поезда");

ui->tableWidget->setHorizontalHeaderItem(0, w);

ui->tableWidget->setHorizontalHeaderItem(1, new QTableWidgetItem("Вместимость груза"));

ui->tableWidget->setHorizontalHeaderItem(2, new QTableWidgetItem("Количество свободных мест"));

ui->tableWidget->setHorizontalHeaderItem(3, new QTableWidgetItem("Цена"));

ui->tableWidget->setHorizontalHeaderItem(4, new QTableWidgetItem("Пункт назначения"));

ui->tableWidget->setHorizontalHeaderItem(5, new QTableWidgetItem("Пн"));

ui->tableWidget->setHorizontalHeaderItem(6, new QTableWidgetItem("Вт"));

ui->tableWidget->setHorizontalHeaderItem(7, new QTableWidgetItem("Ср"));

ui->tableWidget->setHorizontalHeaderItem(8, new QTableWidgetItem("Чт"));

ui->tableWidget->setHorizontalHeaderItem(9, new QTableWidgetItem("Пт"));

ui->tableWidget->setHorizontalHeaderItem(10, new QTableWidgetItem("Сб"));

ui->tableWidget->setHorizontalHeaderItem(11, new QTableWidgetItem("Вс"));

ui->tableWidget->setShowGrid(true);

int i = 0;

for (Iterator it = list->begin(); it != list->end() + 1; ++it)

{

if(it.getPtr() == nullptr)

{

break;

}

ui->tableWidget->insertRow(i);

ui->tableWidget->setItem(i, 0, new QTableWidgetItem(QString::number((\*it).getId())));

ui->tableWidget->setItem(i, 1, new QTableWidgetItem(QString::number((\*it).getCarryingCapacity())));

ui->tableWidget->setItem(i, 2, new QTableWidgetItem(QString::number((\*it).getNumberOfFreeSeats())));

ui->tableWidget->setItem(i, 3, new QTableWidgetItem(QString::number((\*it).getPrice())));

QString str;

switch (list->operator[](i).getDestination())

{

case 1:

{

str = "Россия";

break;

}

case 2:

{

str = "Китай";

break;

}

case 3:

{

str = "Польша";

break;

}

case 4:

{

str = "Украина";

break;

}

case 5:

{

str = "Другая страна";

break;

}

}

ui->tableWidget->setItem(i, 4, new QTableWidgetItem(str));

for (int j = 0; j < 7; j++)

{

switch((\*it).getTimetable()[j])

{

case 0:

{

ui->tableWidget->setItem(i, 5 + j, new QTableWidgetItem("-"));

break;

}

case 1:

{

ui->tableWidget->setItem(i, 5 + j, new QTableWidgetItem("+"));

}

}

}

i++;

}

ui->tableWidget->resizeColumnsToContents();

}

ShowCargoTableWindow::~ShowCargoTableWindow()

{

delete ui;

}

void ShowCargoTableWindow::on\_back\_clicked()

{

this->close();

delete this;

}

\\ файл “showpassengertablewindow.cpp”

#include "showpassengertablewindow.h"

#include "ui\_showpassengertablewindow.h"

ShowPassengerTableWindow::ShowPassengerTableWindow(List<Passenger>\* list) : ui(new Ui::ShowPassengerTableWindow)

{

ui->setupUi(this);

ui->tableWidget->setColumnCount(12);

QTableWidgetItem \*w = new QTableWidgetItem("Номер поезда");

ui->tableWidget->setHorizontalHeaderItem(0, w);

ui->tableWidget->setHorizontalHeaderItem(1, new QTableWidgetItem("Тип места"));

ui->tableWidget->setHorizontalHeaderItem(2, new QTableWidgetItem("Количество свободных мест"));

ui->tableWidget->setHorizontalHeaderItem(3, new QTableWidgetItem("Цена"));

ui->tableWidget->setHorizontalHeaderItem(4, new QTableWidgetItem("Пункт назначения"));

ui->tableWidget->setHorizontalHeaderItem(5, new QTableWidgetItem("Пн"));

ui->tableWidget->setHorizontalHeaderItem(6, new QTableWidgetItem("Вт"));

ui->tableWidget->setHorizontalHeaderItem(7, new QTableWidgetItem("Ср"));

ui->tableWidget->setHorizontalHeaderItem(8, new QTableWidgetItem("Чт"));

ui->tableWidget->setHorizontalHeaderItem(9, new QTableWidgetItem("Пт"));

ui->tableWidget->setHorizontalHeaderItem(10, new QTableWidgetItem("Сб"));

ui->tableWidget->setHorizontalHeaderItem(11, new QTableWidgetItem("Вс"));

ui->tableWidget->setShowGrid(true);

int i = 0;

for (Iterator it = list->begin(); it != list->end() + 1; ++it)

{

if(it.getPtr() == nullptr)

{

break;

}

ui->tableWidget->insertRow(i);

ui->tableWidget->setItem(i, 0, new QTableWidgetItem(QString::number((\*it).getId())));

QString type;

switch ((\*it).getClassOfTickets())

{

case 1:

{

type = "Бизнес";

break;

}

case 2:

{

type = "Купе";

break;

}

case 3:

{

type = "Плацкарт";

break;

}

}

ui->tableWidget->setItem(i, 1, new QTableWidgetItem(type));

ui->tableWidget->setItem(i, 2, new QTableWidgetItem(QString::number((\*it).getNumberOfFreeSeats())));

ui->tableWidget->setItem(i, 3, new QTableWidgetItem(QString::number((\*it).getPrice())));

QString str;

switch ((\*it).getDestination())

{

case 1:

{

str = "Россия";

break;

}

case 2:

{

str = "Китай";

break;

}

case 3:

{

str = "Польша";

break;

}

case 4:

{

str = "Украина";

break;

}

case 5:

{

str = "Другая страна";

break;

}

}

ui->tableWidget->setItem(i, 4, new QTableWidgetItem(str));

for (int j = 0; j < 7; j++)

{

switch((\*it).getTimetable()[j])

{

case 0:

{

ui->tableWidget->setItem(i, 5 + j, new QTableWidgetItem("-"));

break;

}

case 1: {

ui->tableWidget->setItem(i, 5 + j, new QTableWidgetItem("+"));

}

}

}

i++;

}

ui->tableWidget->resizeColumnsToContents();

}

ShowPassengerTableWindow::~ShowPassengerTableWindow()

{

delete ui;

}

void ShowPassengerTableWindow::on\_back\_clicked()

{

this->close();

delete this;

}

\\ файл “takecargoticketform.cpp”

#include "takecargoticketform.h"

#include "ui\_takecargoticketform.h"

#include "clickException.h"

TakeCargoTicketForm::TakeCargoTicketForm(List<Cargo>\*list, QString id) : ui(new Ui::TakeCargoTicketForm), list(list), id(id)

{

ui->setupUi(this);

ui->tableWidget->setColumnCount(12);

QTableWidgetItem \*w = new QTableWidgetItem("Номер поезда");

ui->tableWidget->setHorizontalHeaderItem(0, w);

ui->tableWidget->setHorizontalHeaderItem(1, new QTableWidgetItem("Вместимость груза"));

ui->tableWidget->setHorizontalHeaderItem(2, new QTableWidgetItem("Количество свободных мест"));

ui->tableWidget->setHorizontalHeaderItem(3, new QTableWidgetItem("Цена"));

ui->tableWidget->setHorizontalHeaderItem(4, new QTableWidgetItem("Пункт назначения"));

ui->tableWidget->setHorizontalHeaderItem(5, new QTableWidgetItem("Пн"));

ui->tableWidget->setHorizontalHeaderItem(6, new QTableWidgetItem("Вт"));

ui->tableWidget->setHorizontalHeaderItem(7, new QTableWidgetItem("Ср"));

ui->tableWidget->setHorizontalHeaderItem(8, new QTableWidgetItem("Чт"));

ui->tableWidget->setHorizontalHeaderItem(9, new QTableWidgetItem("Пт"));

ui->tableWidget->setHorizontalHeaderItem(10, new QTableWidgetItem("Сб"));

ui->tableWidget->setHorizontalHeaderItem(11, new QTableWidgetItem("Вс"));

ui->tableWidget->setShowGrid(true);

int i = 0;

for (Iterator it = list->begin(); it != list->end() + 1; ++it)

{

if(it.getPtr() == nullptr)

{

break;

}

ui->tableWidget->insertRow(i);

ui->tableWidget->setItem(i, 0, new QTableWidgetItem(QString::number((\*it).getId())));

ui->tableWidget->setItem(i, 1, new QTableWidgetItem(QString::number((\*it).getCarryingCapacity())));

ui->tableWidget->setItem(i, 2, new QTableWidgetItem(QString::number((\*it).getNumberOfFreeSeats())));

ui->tableWidget->setItem(i, 3, new QTableWidgetItem(QString::number((\*it).getPrice())));

QString str;

switch ((\*it).getDestination())

{

case 1:

{

str = "Россия";

break;

}

case 2:

{

str = "Китай";

break;

}

case 3:

{

str = "Польша";

break;

}

case 4:

{

str = "Украина";

break;

}

case 5:

{

str = "Другая страна";

break;

}

}

ui->tableWidget->setItem(i, 4, new QTableWidgetItem(str));

for (int j = 0; j < 7; j++)

{

switch((\*it).getTimetable()[j])

{

case 0:

{

ui->tableWidget->setItem(i, 5 + j, new QTableWidgetItem("-"));

break;

}

case 1:

{

ui->tableWidget->setItem(i, 5 + j, new QTableWidgetItem("+"));

}

}

}

i++;

}

ui->tableWidget->resizeColumnsToContents();

ui->number->setMinimum(0);

ui->spinBox->setMinimum(1);

ui->spinBox->setMaximum(list->size());

}

TakeCargoTicketForm::~TakeCargoTicketForm()

{

delete ui;

}

void TakeCargoTicketForm::on\_back\_clicked()

{

this->close();

delete this;

}

void TakeCargoTicketForm::on\_submit\_clicked()

{

try

{

int number = ui->spinBox->text().toInt() - 1;

if(ui->number->text().toInt() == 0 && list->size() != 0)

{

throw ClickException(Except::zeroTickets, this, "Введите верное количество билетов (более 0)!");

return;

}

if(list->size() == 0)

{

throw ClickException(Except::emptyList, this, "Поезда отсутствуют!");

return;

}

if (list->operator[](number).getNumberOfFreeSeats() - ui->number->text().toInt() < 0)

{

throw ClickException(Except::zeroFreeSeats, this, "Все места заняты!");

return;

}

if (list->operator[](number).getNumberOfFreeSeats() == 0)

{

throw ClickException(Except::zeroFreeSeats, this, "Все места заняты!");

return;

}

List<Trip> \*trips = new List<Trip>();

string str = id.toStdString() + "C.txt";

File::openTrip(trips, str.c\_str());

Trip trip(list->operator[](number).getId(), list->operator[](number).getPrice(), list->operator[](number).getDestination(), ui->number->text().toInt());

trips->push(trip);

File::saveTrip(trips, str.c\_str());

QMessageBox::information(nullptr, "Взятие билета", "Вы успешно купили " + ui->number->text() + " билетов");

list->operator[](number).setNumberOfFreeSeats(list->operator[](number).getNumberOfFreeSeats() - ui->number->text().toInt());

ui->tableWidget->setItem(number, 2, new QTableWidgetItem(QString::number(list->operator[](number).getNumberOfFreeSeats())));

File::saveCargo(list, "cargo.txt");

}

catch(ClickException<TakeCargoTicketForm> &obj)

{

obj.show();

return;

}

}

\\ файл “takepassengerticketform.cpp”

#include "takepassengerticketform.h"

#include "ui\_takepassengerticketform.h"

#include "clickException.h"

TakePassengerTicketForm::TakePassengerTicketForm(List<Passenger> \*list, QString id) : list(list), id(id), ui(new Ui::TakePassengerTicketForm)

{

ui->setupUi(this);

ui->tableWidget->setColumnCount(12);

QTableWidgetItem \*w = new QTableWidgetItem("Номер поезда");

ui->tableWidget->setHorizontalHeaderItem(0, w);

ui->tableWidget->setHorizontalHeaderItem(1, new QTableWidgetItem("Тип места"));

ui->tableWidget->setHorizontalHeaderItem(2, new QTableWidgetItem("Количество свободных мест"));

ui->tableWidget->setHorizontalHeaderItem(3, new QTableWidgetItem("Цена"));

ui->tableWidget->setHorizontalHeaderItem(4, new QTableWidgetItem("Пункт назначения"));

ui->tableWidget->setHorizontalHeaderItem(5, new QTableWidgetItem("Пн"));

ui->tableWidget->setHorizontalHeaderItem(6, new QTableWidgetItem("Вт"));

ui->tableWidget->setHorizontalHeaderItem(7, new QTableWidgetItem("Ср"));

ui->tableWidget->setHorizontalHeaderItem(8, new QTableWidgetItem("Чт"));

ui->tableWidget->setHorizontalHeaderItem(9, new QTableWidgetItem("Пт"));

ui->tableWidget->setHorizontalHeaderItem(10, new QTableWidgetItem("Сб"));

ui->tableWidget->setHorizontalHeaderItem(11, new QTableWidgetItem("Вс"));

ui->number->setMinimum(0);

ui->tableWidget->setShowGrid(true);

int i = 0;

for (Iterator it = list->begin(); it != list->end() + 1; ++it)

{

if(it.getPtr() == nullptr)

{

break;

}

ui->tableWidget->insertRow(i);

ui->tableWidget->setItem(i, 0, new QTableWidgetItem(QString::number((\*it).getId())));

QString type;

switch ((\*it).getClassOfTickets())

{

case 1:

{

type = "Бизнес";

break;

}

case 2:

{

type = "Купе";

break;

}

case 3:

{

type = "Плацкарт";

break;

}

}

ui->tableWidget->setItem(i, 1, new QTableWidgetItem(type));

ui->tableWidget->setItem(i, 2, new QTableWidgetItem(QString::number((\*it).getNumberOfFreeSeats())));

ui->tableWidget->setItem(i, 3, new QTableWidgetItem(QString::number((\*it).getPrice())));

QString str;

switch ((\*it).getDestination())

{

case 1:

{

str = "Россия";

break;

}

case 2:

{

str = "Китай";

break;

}

case 3:

{

str = "Польша";

break;

}

case 4:

{

str = "Украина";

break;

}

case 5:

{

str = "Другая страна";

break;

}

}

ui->tableWidget->setItem(i, 4, new QTableWidgetItem(str));

for (int j = 0; j < 7; j++)

{

switch((\*it).getTimetable()[j])

{

case 0:

{

ui->tableWidget->setItem(i, 5 + j, new QTableWidgetItem("-"));

break;

}

case 1:

{

ui->tableWidget->setItem(i, 5 + j, new QTableWidgetItem("+"));

}

}

}

i++;

}

ui->tableWidget->resizeColumnsToContents();

ui->spinBox->setMinimum(1);

ui->spinBox->setMaximum(list->size());

}

TakePassengerTicketForm::~TakePassengerTicketForm()

{

delete ui;

}

void TakePassengerTicketForm::on\_submit\_clicked()

{

try

{

int number = ui->spinBox->text().toInt() - 1;

if(ui->number->text().toInt() == 0 && list->size() != 0)

{

throw ClickException(Except::zeroTickets, this, "Введите верное количество билетов (более 0)!");

return;

}

if(list->size() == 0)

{

throw ClickException(Except::emptyList, this, "Поезда отсутствуют!");

return;

}

if (list->operator[](number).getNumberOfFreeSeats() - ui->number->text().toInt() < 0)

{

throw ClickException(Except::zeroFreeSeats, this, "Все места заняты!");

return;

}

if (list->operator[](number).getNumberOfFreeSeats() == 0)

{

throw ClickException(Except::zeroFreeSeats, this, "Все места заняты!");

return;

}

List<Trip> \*trips = new List<Trip>();

string str = id.toStdString() + "P.txt";

File::openTrip(trips, str.c\_str());

Trip trip(list->operator[](number).getId(), list->operator[](number).getPrice(), list->operator[](number).getDestination(), ui->number->text().toInt());

trips->push(trip);

File::saveTrip(trips, str.c\_str());

QMessageBox::information(nullptr, "Взятие билета", "Вы успешно купили " + ui->number->text() + " билетов");

list->operator[](number).setNumberOfFreeSeats(list->operator[](number).getNumberOfFreeSeats() - ui->number->text().toInt());

ui->tableWidget->setItem(number, 2, new QTableWidgetItem(QString::number(list->operator[](number).getNumberOfFreeSeats())));

File::savePassenger(list, "passenger.txt");

}

catch(ClickException<TakePassengerTicketForm> &obj)

{

obj.show();

return;

}

}

void TakePassengerTicketForm::on\_back\_clicked()

{

this->close();

delete this;

}

\\ файл “Tickets.cpp”

#include "Tickets.h"

Tickets::Tickets(int id, int price) : id(id), price(price) {}

int Tickets::getId() const

{

return id;

}

void Tickets::setId(int id)

{

Tickets::id = id;

}

int Tickets::getPrice() const

{

return price;

}

void Tickets::setPrice(int price)

{

Tickets::price = price;

}

\\ файл “Trip.cpp”

#include "Trip.h"

Trip::Trip(int id, int price, int destination , int numberOfFreeSeats) : Tickets(id, price), destination(destination), numberOfFreeSeats(numberOfFreeSeats) {}

int Trip::getDestination() const

{

return destination;

}

void Trip::setDestination(int destination)

{

Trip::destination = destination;

}

int Trip::getNumberOfFreeSeats() const

{

return numberOfFreeSeats;

}

void Trip::setNumberOfFreeSeats(int numberOfFreeSeats)

{

Trip::numberOfFreeSeats = numberOfFreeSeats;

}

# ПРИЛОЖЕНИЕ Б

(Обязательное)

Скриншоты работы программы

Рисунок Б.1 представляет собой главное меню программы.

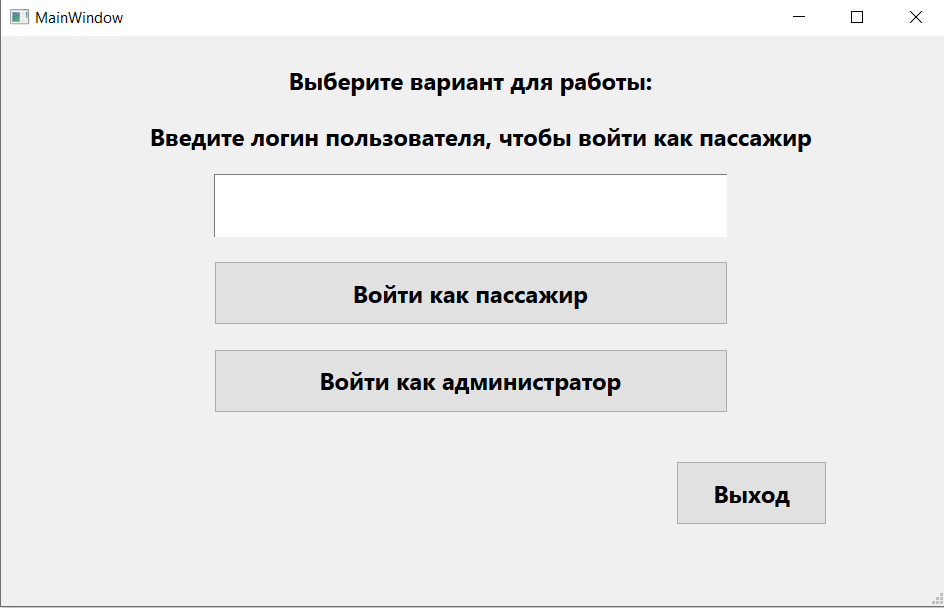


Рисунок Б.1 - главное меню программы.

На рисунке Б.2 изображено окно выбора типа поездов для работы.

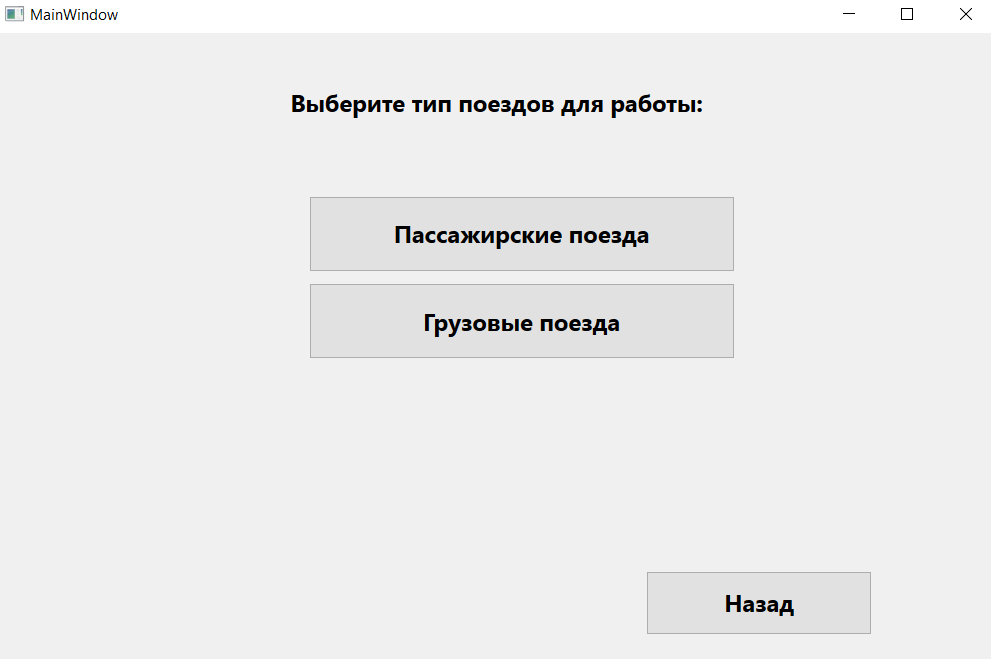


Рисунок Б.2 - окно выбора типа поездов для работы.

Рисунок Б.3 представляет собой окно выбора действий с поездами в режиме администратора.

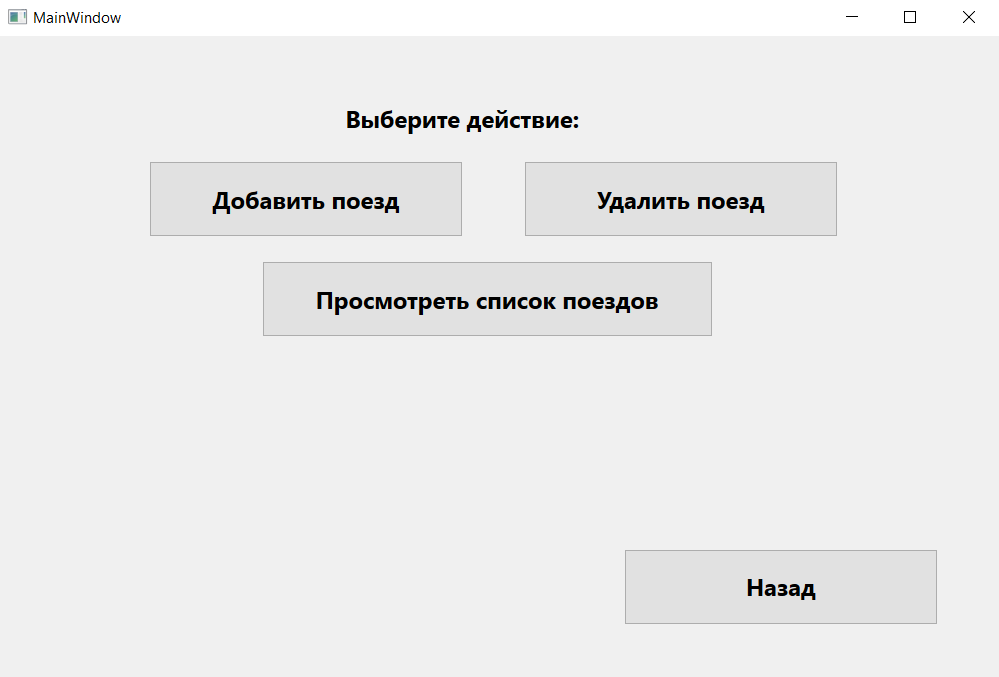


Рисунок Б.3 - окно выбора действий с поездами в режиме администратора.

На рисунке Б.4 изображена форма добавления грузового поезда.

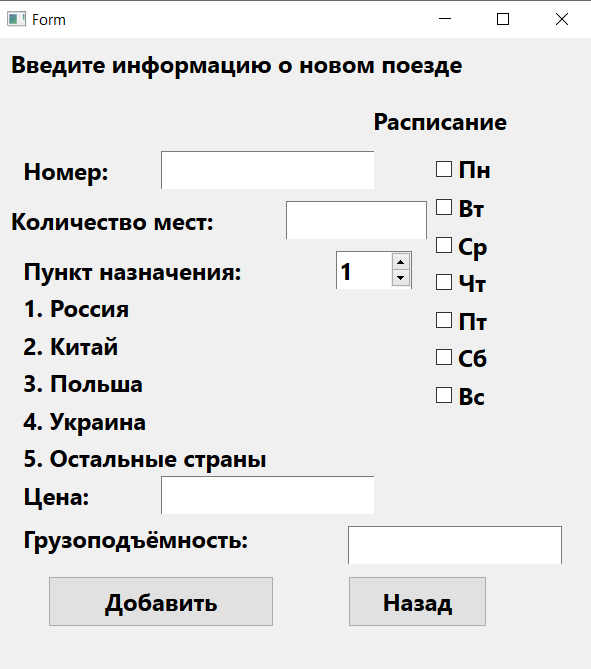


Рисунок Б.4 - форма добавления грузового поезда.

На рисунке Б.5 изображена форма добавления пассажирского поезда.

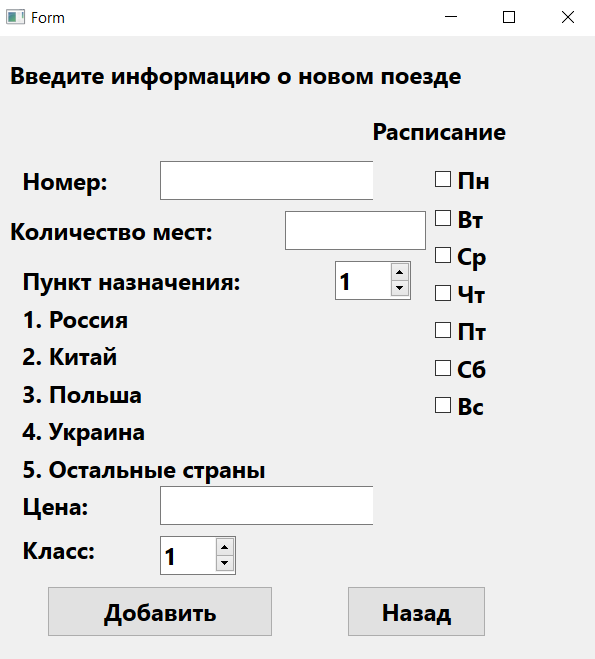


Рисунок Б.5 - форма добавления пассажирского поезда.

Рисунок Б.6 представляет собой список грузовых поездов.

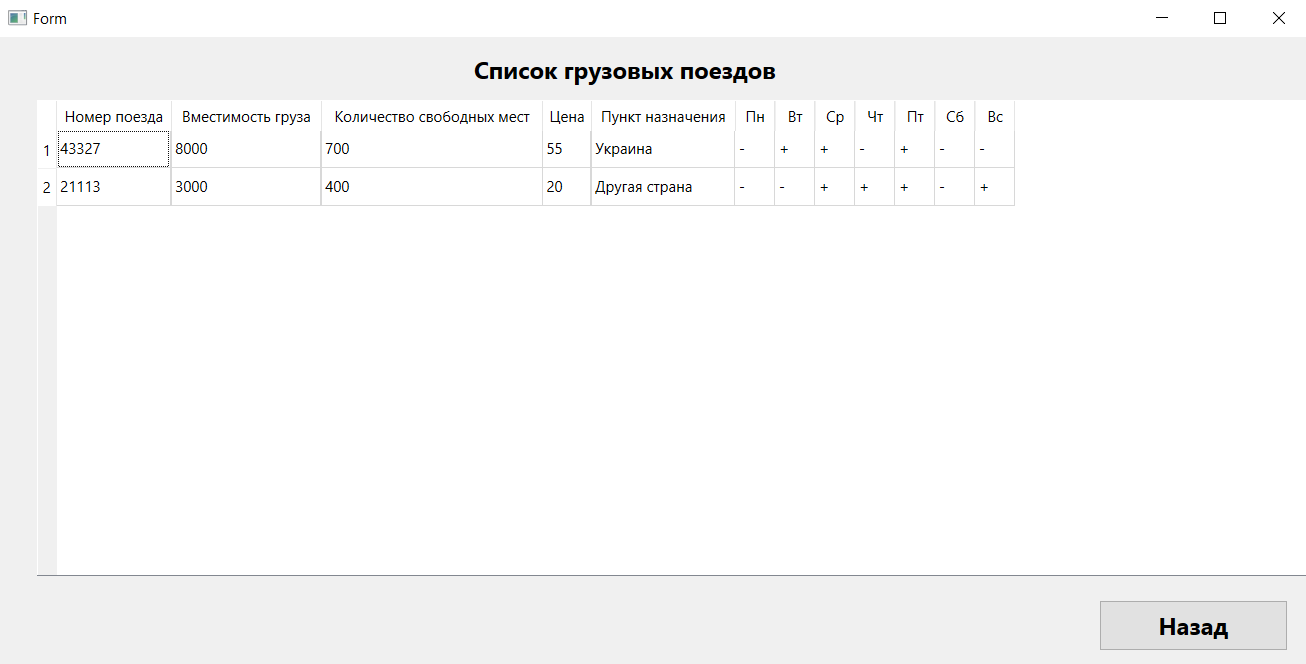


Рисунок Б.6 – список грузовых поездов.

Рисунок Б.7 представляет собой список пассажирских поездов.

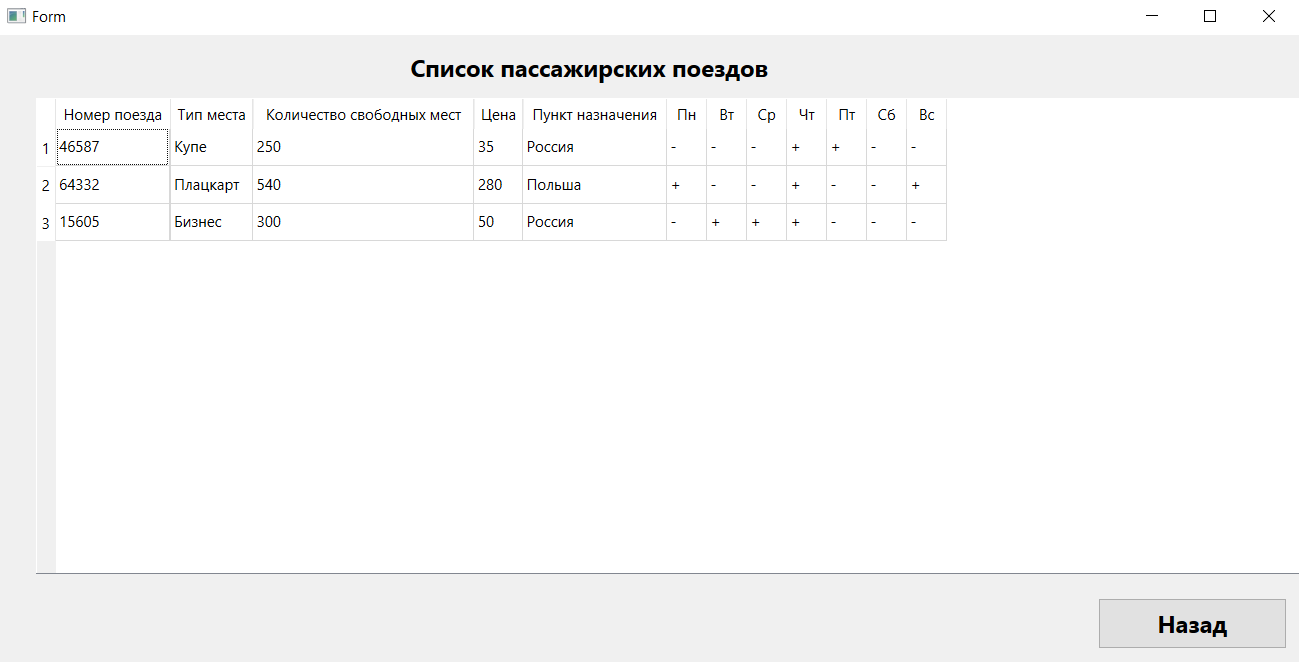


Рисунок Б.7 – список пассажирских поездов.

Рисунок Б.8 представляет собой окно выбора действий с поездами в режиме пассажира.

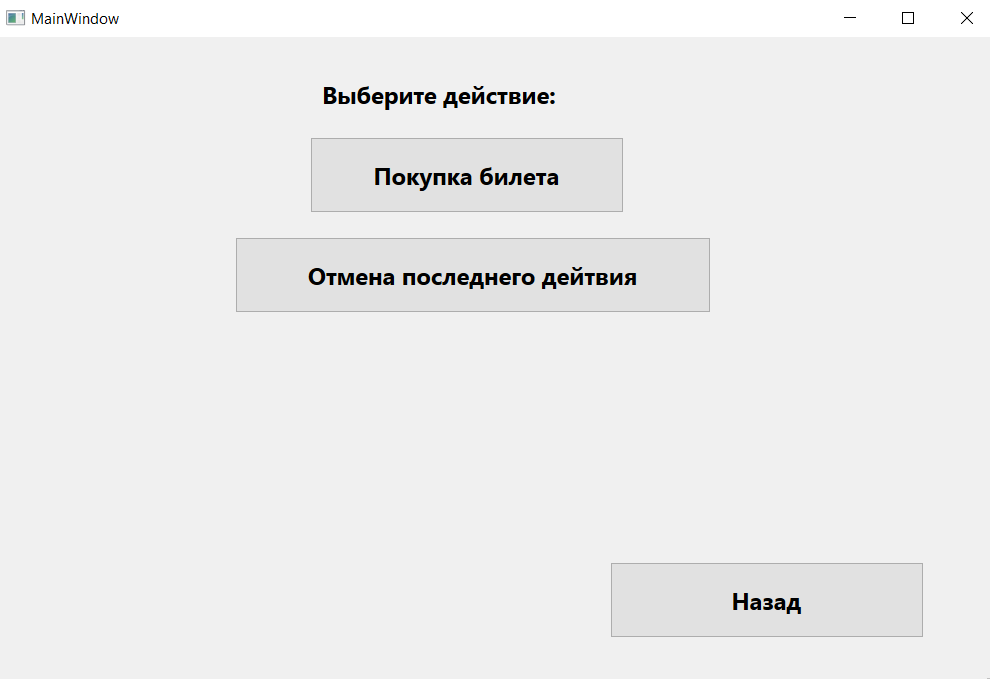


Рисунок Б.8 - окно выбора действий с поездами в режиме пассажира.

На рисунке Б.9 изображено окно покупки билетов на грузовой поезд.

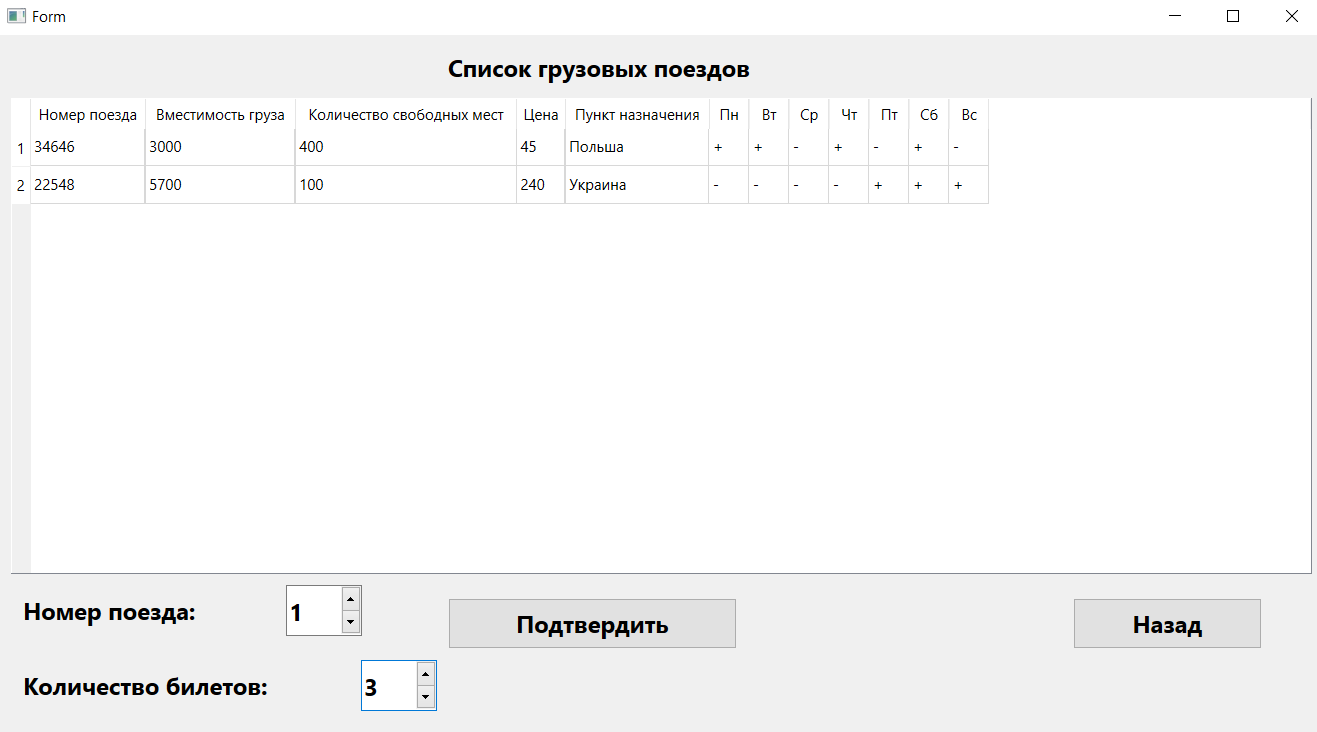


Рисунок Б.9 - окно покупки билетов на грузовой поезд.

На рисунке Б.10 изображено окно покупки билетов на пассажирский поезд.

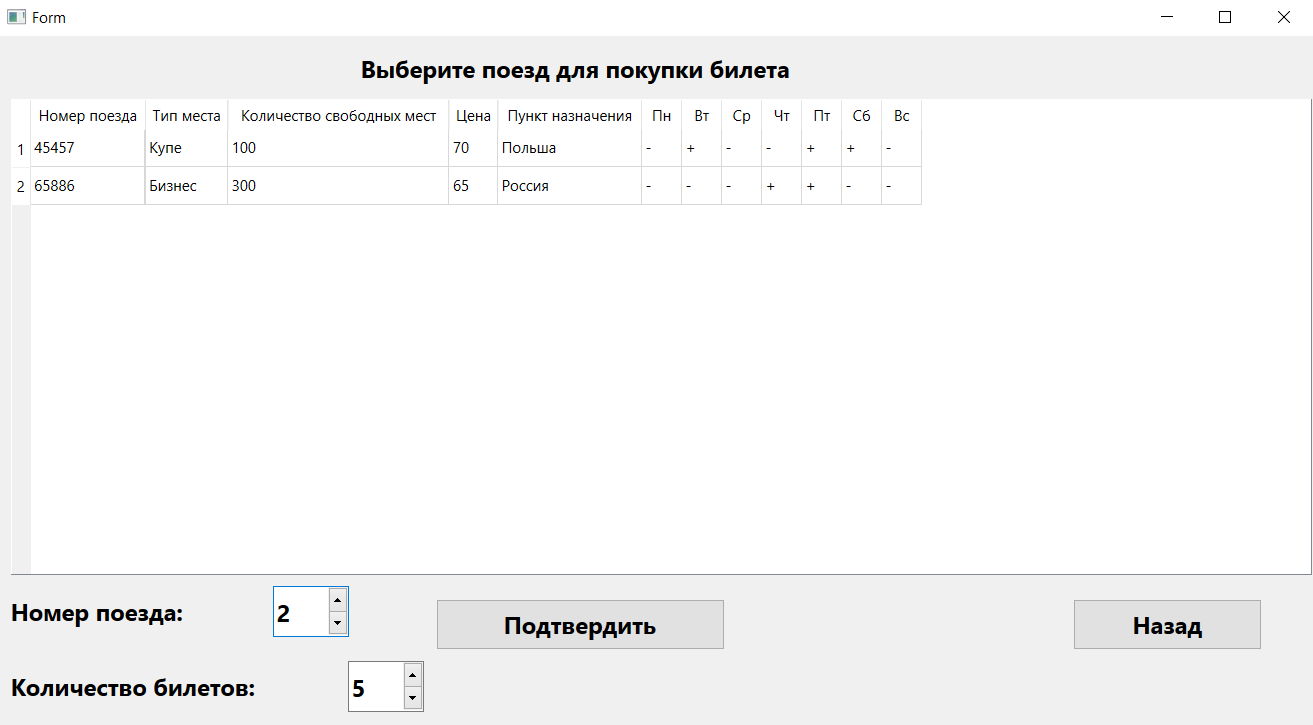


Рисунок Б.10 - окно покупки билетов на пассажирский поезд

На рисунках Б.11, Б.12, Б.13, Б.14, Б.15 изображены сообщения об выполнении удачных действий в программе.

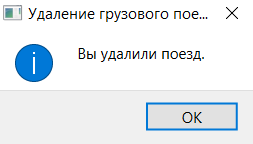


Рисунок Б.11 – удаление поезда

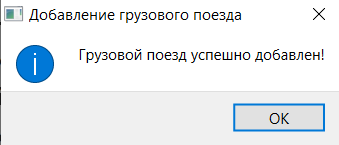


Рисунок Б.12 – добавление грузового поезда

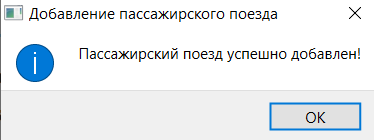


Рисунок Б.13 – добавление пассажирского поезда

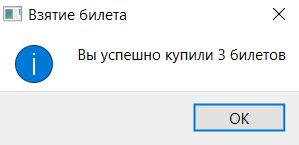


Рисунок Б.14 – покупка билетов

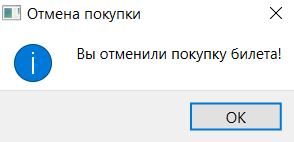


Рисунок Б.15 – отмена покупки билетов

На рисунках Б.16, Б.17, Б.18, Б.19, Б.20, Б.21, Б.22, Б.23, Б.24 изображены сообщения об ошибках.

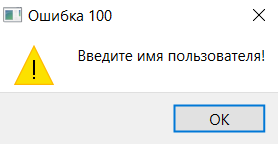


Рисунок Б.16 – ошибка ввода имени пользователя

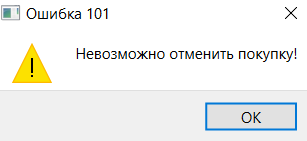


Рисунок Б.17 – ошибка отмены покупки

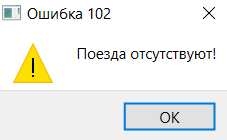


Рисунок Б.18 – ошибка удаления поездов

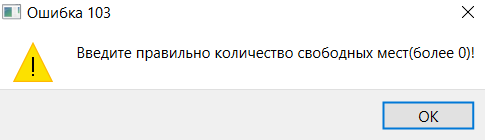


Рисунок Б.19 – ошибка ввода количества свободных мест

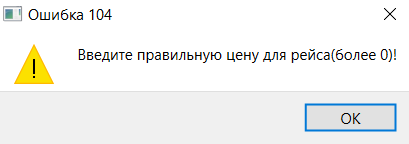


Рисунок Б.20 – ошибка ввода цены рейса

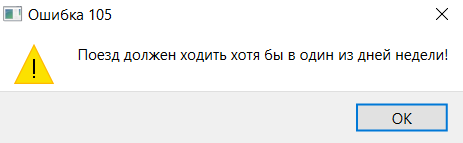


Рисунок Б.21 – ошибка ввода расписания поезда

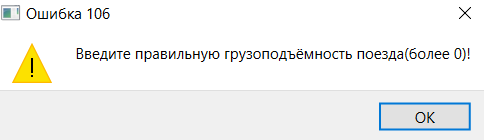


Рисунок Б.22 – ошибка ввода грузоподъёмности поезда

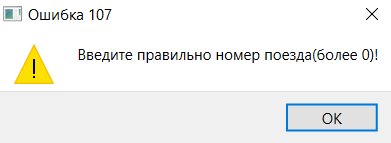


Рисунок Б.23 – ошибка ввода номера поезда

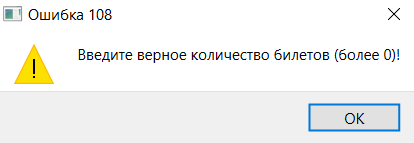


Рисунок Б.24 – ошибка ввода количества билетов

На рисунках Б.25, Б.26, Б.27, Б.28, Б.29 изображены текстовые файлы, которые используются в процессе работы программы.

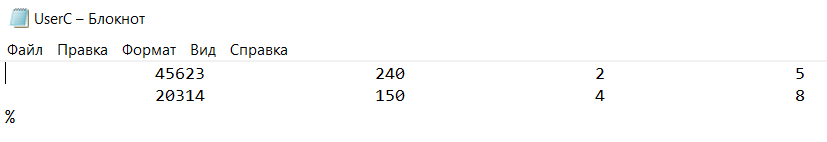


Рисунок Б.25 – файл с билетами пользователя на грузовые рейсы



Рисунок Б.26 – файл с билетами пользователя на пассажирские рейсы



Рисунок Б.27 – файл со списком грузовых рейсов

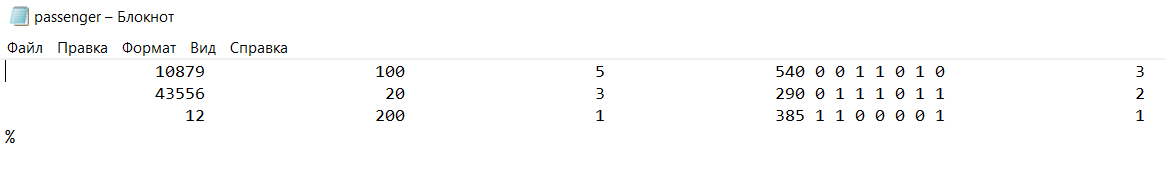


Рисунок Б.28 – файл со списком пассажирских рейсов

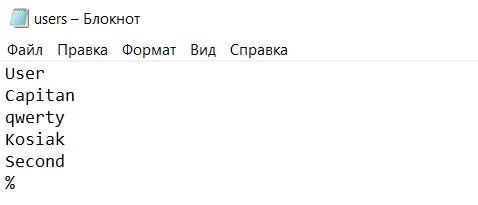


Рисунок Б.29 – файл со списком всех пользователей

# 

# ПРИЛОЖЕНИЕ В

(Обязательное)

Диаграмма классов

# ПРИЛОЖЕНИЕ Г

(Обязательное)

Блок-схема алгоритма void List<T>::push(T&)

# ПРИЛОЖЕНИЕ Д

(Обязательное)

Блок-схема алгоритма void AddCargoForm::on\_submit\_clicked()

# ПРИЛОЖЕНИЕ Е

(Обязательное)

Ведомость документов