СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5

1 обзор методов и алгоритмов решения поставленной задачи 6

1.1 Обзор языка программирования 6

1.2 Обзор среды разработки 7

1.3 Обзор фреймворка Qt 7

2 СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ФАЙЛОВ 9

3 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА 10

3.1 Описание классов данных 10

3.2 Описание классов обработки данных 13

3.3 Описание классов пользовательского интерфейса (окон) 13

4 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ 16

4.1 Алгоритм функции *void MainWindow::findClient()* 16

4.2 Алгоритм функции *void BaseAutoShow::uploadECars() const* 17

4.2 Алгоритм функций *void MainWindow::updateModelLine(QString*

*mark)* и *void BaseAutoShow::downloadServices()* 18

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 18

ЗАКЛЮЧЕНИЕ 27

СПИСОК ЛИТЕРАТУРЫ 28

ПРИЛОЖЕНИЕ А (обязательное) Диаграмма классов 29

ПРИЛОЖЕНИЕ Б (обязательное) Схема алгоритма функции

обновления элемента выбора модели 30

ПРИЛОЖЕНИЕ В (обязательное) Схема алгоритма функции чтения

списка сервисов из бинарного файла 31

ПРИЛОЖЕНИЕ Г (обязательное) Листинг кода 32

ВВЕДЕНИЕ

В настоящее время C++ является одним из языков программирования, наиболее чётко представляющих собой основные парадигмы программирования. Он поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование.

Целью настоящей курсовой работы является создание прикладного программного обеспечения (далее ПО). Это вероятно самая огромная область программирования, так как каждый человек использует прикладное ПО при работе с компьютером (например, когда слушает музыку, смотрит фильмы, печатает документы или ищет информацию в интернете). Для разработки прикладного ПО разработано множество библиотек и фреймворков. Прежде всего, стандартная библиотека языка C++, но помимо её существует множество других: SFML, SDL, Silicon, Qt и другие. В ходе курсовой работы графический интерфейс пользователя осуществляется при помощи средств фреймворка Qt, предоставляющего широкий спектр возможностей для создания прикладного ПО.

1 обзор методов и алгоритмов решения поставленной задачи

1.1 Обзор языка программирования

С++ – это компилируемый, статически типизированный язык программирования общего назначения.

Как уже отмечалось во введении, C++ поддерживает следующие парадигмы программирования:

– процедурное программирование;

– объектно-ориентированное программирование;

– обобщённое программирование.

Принцип ООП, взятый за основу в настоящей работе, основан на трёх основных концепциях: инкапсуляции, наследовании, полиморфизме. Программа в таком случае представлена в виде совокупности объектов, каждый из которых является экземпляром отдельного класса, а классы в совокупности образуют иерархию наследования. Действия программы представляются как свойства какого-либо объекта, либо как результат взаимодействия нескольких объектов одного или разных типов.

Язык C++ имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков, что делает его одним из самых мощных средств программирования, однако тем самым повышая его сложность. Но именно в связи с этим язык C++ является отличной базой для изучения программирования, основных парадигм и алгоритмов. Также в дальнейшем это способствует быстрому изучению программистом других языков, в частности C-подобных (Java, C# и других).

Язык C++ является популярным средством программирования, даже не смотря на то, что в последнее время лидирующие позиции по популярности перед C++ заняли такие языки программирования, как JavaScript, Java, PHP, Python, C#. Лидерство этих языков легко объясняется: они проще. Однако это отнюдь не является главным критерием выбора языка программирования. При помощи языка C++ можно разрабатывать огромный спектр продуктов: начиная от простых прикладных программ или компьютерных игр, заканчивая полноценными операционными системами или системами управления серверами. И, как правило, при разработке более крупных и важных проектов, где необходима надёжность и защищённость (например, при разработке программного или системного обеспечения для космической техники) программисты оставляю однозначный выбор за языком C++.

1.2 Обзор среды разработки

Qt Creator – это кроссплатформенная свободная интегрированная среда разработки, работающая с языками C и C++. Разработана компанией Qt Development Frameworks для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика GDB и визуальные средства разработки интерфейса. Для Windows-версий комплектуется компилятором, заголовочными и объектными файлами MinGW.

Qt Creator также включает в себя Qt Assistant – справочную систему Qt, упрощающую работу с документацией по библиотеке, а также позволяющую создавать кроссплатформенную справку для разрабатываемого на основе Qt программного обеспечения.

Основная задача Qt Creator — упростить разработку приложения с помощью фреймворка Qt на разных платформах. Поэтому среди возможностей, присущих любой среде разработки, есть и специфичные, такие как отладка приложений на QML и отображение в отладчике данных из контейнеров Qt, встроенный дизайнер интерфейсов как на QML, так и на QtWidgets.

1.3 Обзор фреймворка Qt

Qt – это кроссплатформенный фреймворк для разработки ПО на языке программирования C++. Qt позволяет запускать написанное с его помощью ПО на большинстве современных операционных систем путём простой компиляции программы для каждой из систем без изменения исходного кода программы. Фреймворк включает в себя библиотеки со всеми основными классами, которые могут потребоваться при разработке прикладного ПО, в частности при разработке элементов графического интерфейса, именно поэтому данный фреймворк был выбран для выполнения курсовой работы.

Отличительной особенностью фреймворка является использование метаобъектной системы – предварительной системы обработки кода.

Метаобъектная система – часть ядра фреймворка Qt для поддержки в С++ таких возможностей, как сигналы и слоты для коммуникации между объектами в режиме реального времени и динамических свойств системы. Метаобъектная система содержит:

– класс *QObject*;

– макрос *Q\_OBJECT*;

– утилиту *moc* (метаобъектный компилятор).

*QObject* – это базовый класс для всех Qt-классов. Макрос Q\_*OBJECT* используется для включения метаобъектных функций в классах и на этапе компиляции работает как препроцессор, который преобразует применения макроса в исходный код C++.

Таким образом, использование метаобъектной системы упрощает реализацию цикла обработки сообщений всех объектов. Связывая между собой разные объекты через определенные методы, присущие этим объектам (сигналы и слоты).

Сигнал ­– это метод объекта класса *QObject*, который вызывается в реальном времени в результате какого-либо непосредственного внешнего воздействия на объект (например, нажатие пользователь на кнопку, изменение текста в строке поиска и т. д.). Сигналы представляют собой методы класса, но в отличие от методов их не нужно определять, и они не могут быть вызваны как обычная функция. Для того, чтобы сигнал вызвался, он должен быть "выработан" (это реализуется при помощи ключевого слова *emit*) во время выполнения определённой логически связанной с данным сигналом функции. Например, сигнал *void valueChanged(int)* логично "вырабатывать" при вызове метода *void setValue(int)*. Сигнал во время вызова может отправлять информацию (как аргументы функции) о текущем состоянии объекта, свойством которого он является. Этот аргумент в последующем будет передан вызванному данным сигналом слоту. Сигналы по умолчанию являются константными методами класса (указывать при объявлении это не требуется), так как лишь передают информацию о состоянии.

Слот – это метод объекта класса *QObject*, который производит непосредственное действие в реальном времени над самим объектом (например, закрытия окна, очистка содержимого, обработка информации и т. д.) при определённых условиях (как правило, при вызове сигнала). Слот представляет собой обычный метод класса: он может иметь любое количество параметров, может также быть напрямую вызван, но в отличие от обычного метода он ещё может быть вызван определённым сигналом. Для того чтобы слот был вызван неким сигналом, он должен быть "связан" с ним, в таком случае, когда будет вызван связанный с данным слотом сигнал, то тогда будет вызван и сам слот. Слоты, как правило, являются не константными методами, так как они обычно производят обработку данных.

2 СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ФАЙЛОВ

Структура файлов представлена бинарными файлами двух видов:

– один файл типа *.bin*, в котором хранятся данные о последнем присвоенном ID для четырёх типов данных;

– шесть файлов типа *.dat*, в которых хранятся список марок авто, список клиентов, список заказов, список автосервисов и два списка авто.

Рассмотрим структуру каждого файла по отдельности.

Файл *idlast.bin* хранит всегда только четыре 32-битных целых числа (*int*). Если при первом запуске программы файл отсутствует, то он создаётся и заполняется четырьмя целыми числами со значением минус один.

Файл *markslist.dat* хранит в себе данные о марках авто. В начало файла записывается 32-битное число с общим количеством марок. Каждая марка записывается как целое число, являющееся уникальным идентификатором марки, и строка типа *UString* (собственная разработка) с названием марки.

Файл *serviceslist.dat* хранит в себе данные об автосервисах. В данном файле количество сервисов не записывается, так как сервисы считываются для каждой марки отдельно. Каждый сервис записывается как одно целое число с идентификатором марки, к которой относится сервис и две строки типа *UString* (название и адрес).

Файл *clientslist.dat* хранит в себе данные о клиентах. В начало файла записывается число с общим количеством клиентов. Каждый клиент записывается как целое число, являющееся идентификатором клиента, три строки типа *UString* (фамилия, имя, отчество), три целых числа (день, месяц, год рождения), две строки *UString* (номер паспорта, адрес).

Файл *orderslist.dat* хранит данные о заказах. Заказы, как и автосервисы, считываются в зависимости от выбранного клиента, поэтому в файл количество заказов не записывается. Каждый заказ хранится в виде шести чисел целого типа (уникальный идентификатор заказа, идентификатор клиента, идентификатор покупки, день, месяц и год покупки).

Файл *ecarslist.dat* хранит данные об электромобилях. Считывание производится в зависимости от выбранной марки. Каждый автомобиль хранится в виде двух целых чисел (идентификатор модели, идентификатор марки), строки *UString* (название модели), далее записывается информация о кузове автомобиля: строка *UString* (тип кузова) и целое число (масса кузова), после этого записывается информации о двигателе (для данной машины это электродвигатель): три целых числа (мощность, ёмкость и КПД), далее записывается информация о трансмиссии: булева переменная (указывает, является ли коробка переключения передач автоматической или нет) и целое число (количество ступеней трансмиссии), далее записывается три целых числа (год выпуска, цена и количество на складе).

Файл *fcarslist.dat* хранит данные об автомобилях с двигателями внутреннего сгорания. Файл имеет ту же структуру, что и файл *ecarslist.dat* за исключением того, что запись информации о двигателе имеет следующую структуру: два целых числа (мощность и количество цилиндров) и 64-битное число с плавающей точкой (*double*) (объём камеры сгорания двигателя).

В случае первого запуска программы файлы, если они отсутствуют, создаются пустыми. Файл *idlast.bin* по умолчанию заполняется четырьмя целыми числами со значением минус один. Файлы *markslist.dat* и *clientslist.dat* заполняются одним целым числом со значением ноль. Остальные файлы создаются полностью пустыми.

3 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА

Классы программы можно разбить на различные блоки: классы данных, классы обработки данных, классы пользовательского интерфейса (окон). Рассмотрим каждую группу, и классы, которые в неё входят. Каждый класс будем рассматривать с точки зрения его ключевых параметров. Полную информацию о каждом классе можно найти на диаграмме классов (см. приложение А) или в листинге кода (см. приложение Г).

3.1 Описание классов данных

Группа данных включает в себя следующие классы и структуры:

1 Структура *Date* хранит дату. Структура имеет три поля типа *int*: день, месяц и год. Для структуры перегружены операторы записи и чтения из бинарного файла и операторы сравнения.

2 Класс *UString* – класс строки (собственная разработка). Класс состоит двух полей типа *int*, хранящих длину строки и размер выделенной памяти для строки, и поля char \*, которое хранит саму строку. Класс имеет все основные функции для работы со строкой (подробно можно посмотреть на диаграмме классов или листинге кода), но акцентируем внимание на методах *char \* get\_cstr() const* и *QString getQString() const*. Данные методы преобразуют строку типа *UString* к обычной строке (char \*) и к строке типа *QString*, которая используется фреймворком Qt по умолчанию. Также класс *UString* имеет перегруженные арифметические операторы для простой работы со строками, операторы сравнения и операторы записи и чтения из бинарного файла.

3 Класс *cpt::CarBody* – класс кузова автомобиля. Класс состоит из поля типа *UString*, хранящего тип кузова автомобиля, и поля типа *int* с информаций о массе. Класс имеет перегруженные операторы записи и чтения из бинарного файла. Также класс принадлежит пространству имён *cpt*, которое содержит классы частей автомобиля.

4 Класс *cpt::CarTransmission* – класс трансмиссии автомобиля. Класс имеет поле типа *bool*, которое указывает, является ли коробка переключения передач автоматической (значение *true* – автоматическая, значение *false* – механическая), и поле типа *int* с количеством ступеней трансмиссии. Класс имеет перегруженные операторы записи и чтения из бинарного файла. Класс принадлежит пространству имён *cpt*.

5 Класс *cpt::CarEngine* – класс двигателя автомобиля. Класс имеет поле типа *int*, хранящее мощность двигателя. Класс имеет перегруженные операторы записи и чтения из бинарного файла. Класс принадлежит пространству имён *cpt*. Также класс является базовым для классов *cpt::ElectroEngine* и *cpt::FuelEngine*.

6 Класс *cpt::ElectroEngine* – класс электродвигателя. Класс наследует от класса *cpt::CarEngine* данные о мощности (поле типа *int*), дополняя его данными о ёмкости и КПД двигателя (два поля типа *int*). Имеет перегруженные операторы записи и чтения из бинарного файла и принадлежит пространству имён *cpt*.

7 Класс *cpt::FuelEngine* – класс двигателя внутреннего сгорания. Класс наследует от класса *cpt::CarEngine* данные о мощности (поле типа *int*), дополняя его данными о количестве цилиндров (поле типа *int*) и о объёме двигателя (поле типа *double*). Имеет перегруженные операторы записи и чтения из бинарного файла и принадлежит пространству имён *cpt*.

8 Класс *AutoMark\_ID* – класс индексации марок авто. Класс содержит одно статическое поле типа *int* которое хранит информацию о предыдущем использованном уникальном идентификаторе. Является базовым для класса *AutoMark*.

9 Класс *AutoMark* – класс марки автомобиля. Наследуется от класса *AutoMark\_ID*. Имеет поле типа *int*, хранящее уникальный идентификатор марки, и поля типа *UString*, хранящее название марки. Уникальный идентификатор присваивается увеличенной на единицу статической переменной класса *AutoMark\_ID*. Класс имеет операторы сравнения и записи и чтения из бинарного файла.

10 Класс *AutoService* – класс автосервиса. Имеет поле типа *int*, хранящее уникальный идентификатор марки авто, к которой принадлежит сервис, и два поля типа *UString*, хранящие название сервиса и адрес, по которому он расположен.

11 Класс *Human* – класс человека. Имеет три поля типа *UString*, хранящие фамилию, имя и отчество, и поле типа *Date*, хранящее дату рождения.

12 Класс *Client\_ID* – класс индексации клиентов. Аналогичен классу *AutoMark\_ID*: хранит информацию о предыдущем использованном уникальном идентификаторе. Является базовым для класса *Client*.

13 Класс *Client* – класс клиента. Наследуется от классов *Client\_ID* и *Human*. Помимо унаследованных данных имеет поле типа *int*  с уникальным идентификатором и два поля типа *UString* c адресом проживания и номером паспорта.

14 Класс *Order\_ID* – класс индексации заказов. Аналогичен классам *Client\_ID* и *AutoMark\_ID*: хранит информацию о предыдущем использованном уникальном идентификаторе. Является базовым для класса *Order*.

15 Класс *Order* – класс заказа. Имеет три поля типа *int*, хранящие уникальные идентификаторы заказа, клиента и автомобиля, и поле типа *Date*, хранящее дату оформления заказа.

16 Класс *Car\_ID* – класс индексации автомобилей. Аналогичен классам *Client\_ID*, *AutoMark\_ID* и *Order\_ID*: хранит информацию о предыдущем использованном уникальном идентификаторе. Является базовым для класса *Car*.

17 Класс *Car<EngineType>* – класс автомобилей. Является шаблонным классам, так как данные о двигателях внутреннего сгорания и электродвигателях имеют разные размеры. Так как двигатель может быть только двух типов, то на параметры шаблона наложены ограничения при помощи compile-time проверки (реализовано при помощи *static\_assert*). Класс имеет пять полей типа *int*, хранящие уникальные идентификаторы конкретной модели и марки, год выпуска, цену и количество на складе, поле типа *UString*, хранящее название модели. Класс агрегирует в себе данные о кузове автомобиля (поле типа *cpt::CarBody \**), о трансмиссии (поле типа *cpt::CarTransmission \**) и о двигателе (поле типа *cpt::ElectroEngine* \* или *cpt::FuelEngine* \*). Класс имеет операторы сравнения и записи и чтения из бинарного файла.

3.2 Описание классов обработки данных

Рассмотрим группу классов, производящих обработку данных:

1 Класс *BaseAutoShow* – основной класс обработки данных. В классе производится обработка специализированных данных: объектов классов *AutoMark*, *AutoService*, *Client*, *Order*, *Car<cpt::ElectroEngine>* и *Car<cpt::FuelEngine>* – их запись и чтение из файлов, добавление и удаление и другие необходимые методы. Данный класс является функциональным по отношению к данным, и его следует выбирать как базовый для класса интерфейса. Таким образом можно разделить реализацию интерфейса и функционала, что является одним из принципов ООП.

2 Класс *UFile* – базовый класс файла. Наследуется от *std::fstream*. Обеспечивает открытие файла в обоих режимах: текстовом и бинарном.

3 Класс *UBinFile* – класс бинарного файла. Наследуется от *UFile*. Обеспечивает безопасное открытие только в режиме бинарного файла. Для упрощения работы для класса перегружены операторы побитового сдвига (*operator <<()* и *operator >>()*) в качестве операторов записи и чтения из файла для основных базовых типов.

4 Класс *MyException* – базовый класс исключений (собственная разработка). Является базовым классов для всех исключений. Имеет поле типа *UString*, которое хранит информацию о произошедшей ошибке, и виртуальный метод *const QString what() const*, служащий для вывода информации об ошибке. Методы класса специфицированы как безопасные и сами по себе не могут генерировать исключительные ситуации.

5 Класс *FstreamException* – класс исключений, связанный с ошибками при работе с файлами. Наследуется от класса *MyException*.

6 Класс *ContainerException* – класс исключений, связанный с ошибками при работе с контейнерами. Наследуется от класса *MyException*.

3.3 Описание классов пользовательского интерфейса (окон)

Группа классов окон содержит множество окон, реализующих взаимодействие пользователя с основным функциональным классом *BaseAutoShow*. Интерфейс представлен в виде одного основного окна и нескольких модальных окон, каждое из которых представляет собой взаимодействие пользователя с определённой функцией. В отличие от предыдущих групп, данные классы реализованы при помощи функционала фреймворка Qt. Рассмотрим каждый класс данной группы:

1 Класс (окно) *MainWindow* – класс главного окна. Наследуется от абстрактного класса *QMainWindow* – класса, необходимого для создания главного окна в фреймворке Qt. Так же наследуется от класса *BaseAutoShow* для объединения реализации и интерфейса. В классе реализовано множество методов для простого взаимодействия пользователя с интерфейсом, а так же методов для вызова модальных окон, представляющих интерфейс для определённых функций класса *BaseAutoShow* (например, добавление или отображение списка). Обратим особое внимание на деструктор данного класса: в нём выполняется только очистка динамических структур данных имеющихся в классе, но нет удаления динамических объектов, унаследованных от класса *QObject*, это объясняется тем, что все объекты окна связываются через указатели на так называемого "родителя" и "наследника". И при вызове деструктора объекта *QObject* указатели на "наследника" поочерёдно удаляются. Таким образом, нет необходимости удалять объекты, унаследованные от класса *QObject*, к другим же динамическим данным это не относится (например, *std::vector* или *std::list*). Данное окно представлено в виде трёх вкладок ("Авто", "Клиенты", "Сервисы"), каждая из которых осуществляет работу с моделями автомобилей, клиентами и заказами, марками и сервисами соответственно. В частности в самом главном окне реализуется интерфейс поиска модели, клиента и сервисов определённой марки, удаление марки и клиента, а так же отмена этих удалений. Является родительским для всех модальных окон, кроме *AddServiceWindow*.

2 Класс (окно) *AddAutoWindow* – класс окна для добавления новой модели автомобиля. Наследуется от класса *QDialog* – класса для создания модального окна. Следует пояснить, что модальное окно – это такое окно, которое при появлении блокирует окно, их которого оно было вызвано, до тех пор, пока оно не будет закрыто. Данное окно является интерфейсом для добавления новой модели авто, состоит окно из множества полей, необходимых для заполнения (если не все поля заполнены, то сохранение невозможно) информацией об автомобиле. Следует отметить, что при вводе в поле "Марка" отсутствующей марки, она будет добавлена как новая.

3 Класс (окно) *AddCarAmountWindow* – класс окна для добавления количества модели на складе. Наследуется от *QDialog* и является модальным. Позволяет изменять количество определённой модели на складе. Представляет собой одно поле – добавочное количество.

4 Класс (окно) *AddClientWindow* – класс окна для добавления нового клиента. Наследуется от *QDialog* и является модальным. Является интерфейсом для добавления нового клиента. Как и интерфейс добавления новой модели, представляет собой множество полей необходимых для заполнения информацией о клиенте.

5 Класс (окно) *DateInput* – класс окна для поиска по дате. Наследуется от *QDialog* и является модальным. Вызывается при поиске клиента при условии, если найдено несколько клиентов с указанным именем, для продолжения поиска следует уточнить дату рождения клиента. Представляет собой одно необходимое для заполнения поле.

6 Класс (окно) *AddOrderWindow* – класс окна для добавления заказа. Наследуется от *QDialog* и является модальным. Доступно для вызова только при выборе существующего клиента. Представляет собой поля для выбора покупаемого автомобиля, заказ будет оформлен на выбранного клиента.

7 Класс (окно) *OrdersListWindow* – класс окна для отображения заказов выбранного клиента. Наследуется от *QDialog* и является модальным. Доступно для вызова только при выборе существующего клиента. Представляет собой таблицу, отображающую информацию обо всех заказах выбранного клиента.

8 Класс (окно) *AddMarkWindow* – класс окна для добавления марки авто. Наследуется от *QDialog* и является модальным. Представляет собой поле для ввода названия новой марки. Также новая марка может быть добавлена в окне *AddAutoWindow*.

9 Класс (окно) *AddServiceWindow* – класс окна для добавления автосервиса. Наследуется от *QDialog* и является модальным. Представляет собой поля, необходимые для заполнения информацией об автосервисе.

10 Класс (окно) *ServicesListWindow* – класс окна для отображения сервисов определённо марки, и их удаления. Наследуется от *QDialog* и является модальным. Отображает список автосервисов выбранной марки в виде таблицы. Позволяет их удалять и реализует отмену удаления. Является родительским для модального окна *AddServiceWindow*.

11 Класс (окно) *AboutWindow* – класс окна, выводящего краткую информацию о приложении. Наследуется от *QDialog* и является модальным.

4 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

Основными алгоритмами в программе являются алгоритмы поисков и работы с файлами. Поэтому рассмотрим эти алгоритмы. Так же в программе часто используется проверка полей на заполнение, но они слишком просты и не нуждаются в рассмотрении.

4.1 Алгоритм функции *void MainWindow::findClient()*

Рассмотрим алгоритм по шагам (см. таблицу 4.1) функции *void MainWindow::findClient()* функция не принимает аргументов, так как использует данные класса.

Таблица *4.1* – Алгоритм функции поиска клиента

|  |  |
| --- | --- |
| Шаг | Описание действий |
| 1 | Начало. |
| 2 | Объявление переменной *temp* типа *Human*, инициализируемой именем из *line\_edit.* |
| 3 | Объявление итератора *it* для *vector<Client>* и инициализация  *vector<Client>::iterator it = std::find(clients.begin(), clients.end(), dynamic\_cast<const Client&>(temp));* |
| 4 | Если *std::count(clients.begin(), clients.end(),static \_cast<const Client&>(temp))==0*, то переход к шагу 15. |
| 5 | Если *std::count(clients.begin(), clients.end(),static \_cast<const Client&>(temp))>1*, то переход к следующему шагу, иначе к шагу ­14. |
| 6 | Объявление переменной *temp\_date* типа *Date*. |
| 7 | Вызов окна *DateInput* и передача ему *temp\_date.* |
| 8 | *temp.setBirthday(temp\_date);* |
| 9 | Начло цикла 1 поиска по дате. |
| 10 | Если *it->getBirthday() == temp.getBirthday()*, то переход к шагу 14. |
| 11 | *it = std::find(it + 1, clients.end(), static\_cast<const Client&>(temp));* |
| 12 | Если *it == clients.end()*, то переход к шагу 15. |
| 13 | Конец цикла 1 поиска по дате. |
| 14 | Вывод информации о клиенте *\*it*. Переход к шагу 17. |
| 15 | *status\_bar->showMessage(QString("Клиент не найден!"), 5000);* |
| 16 | *current\_client = –1;* |
| 17 | Выход. |

4.2 Алгоритм функции *void BaseAutoShow::uploadECars() const*

Рассмотрим ещё функцию *void BaseAutoShow::uploadECars() const*. Её алгоритм по шагам представлен в таблице 4.2:

Таблица *4.2* – Алгоритм по шагам функции записи элементов электромобилей в файл

|  |  |
| --- | --- |
| Шаг | Описание действий |
| 1 | Начало. |
| 2 | Открытие файла *UBinFile file(ecars\_file.get\_cstr(), ios\_base::in)* для чтения. |
| 3 | Объявление вектора *vector<Car<cpt::ElectroEngine>> out\_ecars;* |
| 4 | Если *!file.is\_open()==true || is\_all\_cars!=0*, то переход к шагу\_. |
| 5 | Запуск цикла 1 чтения из файла. |
| 6 | Если *file.eof()==true*, то переход к шагу 12. |
| 7 | *Car<cpt::ElectroEngine> temp;* |
| 8 | *file >> temp;* |
| 9 | Если *temp.getMarkID() == current\_mark*, то переход на следующую итерацию цикла. |
| 10 | *out\_ecars.push\_back(temp);* |
| 11 | Конец цикла 1 чтения из файла. |
| 12 | Закрытие файла *file.close();* |
| 13 | Вставка текущих моделей *out\_ecars.insert(out\_ecars.end(), ecars.cbegin(), ecars.cend());* |
| 14 | Сортировка вектора *sort(out\_ecars.begin(), out\_ecars.end());* |
| 15 | Открытие файла *file.open(ecars\_file.get\_cstr(), ios\_base::out)* для записи. |
| 16 | Если *!file.is\_open()* то переход к шагу 23. |
| 17 | Объявление переменной и инициализация *int i = 0;* |
| 18 | Запуск цикла 2 записи в файл. |
| 19 | Если i >= *out\_ecars.size()*, то переход к шагу 22. |
| 20 | *file << out\_ecars[i++];* |
| 21 | Конец цикла 2 записи в файл. |
| 22 | Закрытие файла *file.close();* |
| 23 | Выход. |

4.2 Алгоритм функций *void MainWindow::updateModelLine(QString mark)* и *void BaseAutoShow::downloadServices()*

Рассмотрим ещё две функции в виде схемы алгоритма:

– функцию *void MainWindow::updateModelLine(QString mark)* (см. приложение Б);

– функцию *void BaseAutoShow::downloadServices()* (см. приложение В).

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Минимальные системные требования:

– операционная система: Windows 7 или Windows Vista;

– процессор: частота не менее 500 МГц;

– оперативная память: не менее 128 Мб;

– место на диске: не менее 32 Мб;

– дополнительно: размер приложений текста и других элементов не должен превышать 100%.

Запуск программы производится при помощи исполняемого файла *auto\_show.exe*. При запуске появляется главное окно с открытой вкладкой "Авто":

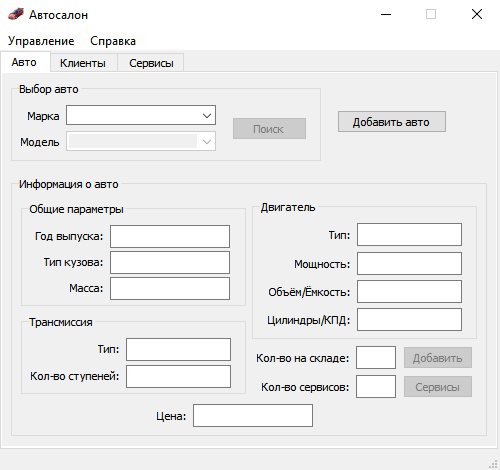


Рисунок 5.1 – Главное окно с активной вкладкой "Авто"

Вкладка "Авто" на главном окне позволяет просматривать сведения о конкретном автомобиле при вводе названия марки и модели и нажатии клавиши "Поиск":

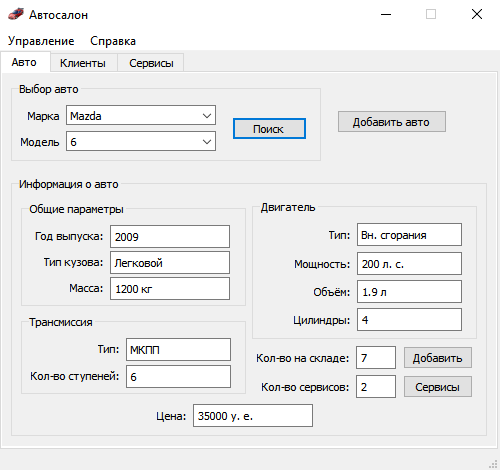
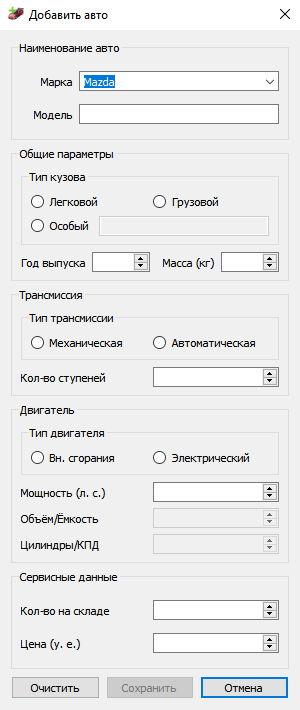
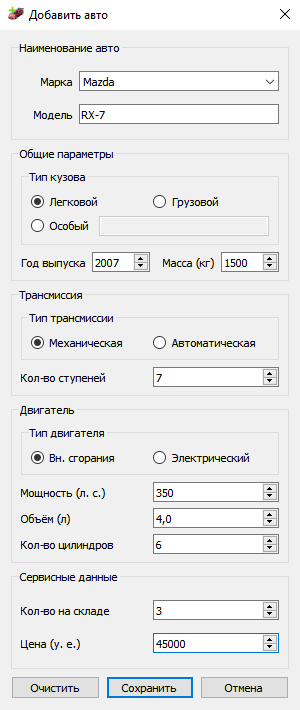


Рисунок 5.2 – Главное окно с результатами поиска авто

При нажатии клавиши добавить появляется диалоговое окно, в поля которого нужно внести информацию о новой модели автомобиля, так же по умолчанию выбирается введённая в главном окне марка. При вызове окна главное окно блокируется и недоступно для изменения. А кнопка сохранения становится доступна только в случае, если все поля для данных заполнены:



а)



б)

а – незаполненная форма; б – заполненная форма

Рисунок 5.3 – Диалоговое окно добавления авто

Для выбранного автомобиля можно также вывести список сервисов (если он имеется), для этого необходимо нажать кнопку "Сервисы":

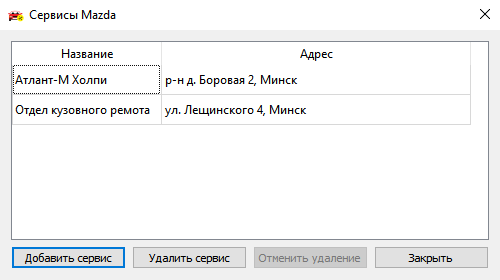


Рисунок 5.4 – Список сервисов

По нажатию кнопки "Добавить" можно увеличить количество автомобилей данной модели на складе:

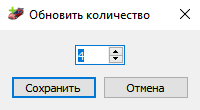


Рисунок 5.5 – Диалоговое окно "Обновит количество"

Вкладка "Клиенты" главного окна позволяет просматривать информацию о клиенте, добавлять новых клиентов, оформлять на них заказы и просматривать заказы:

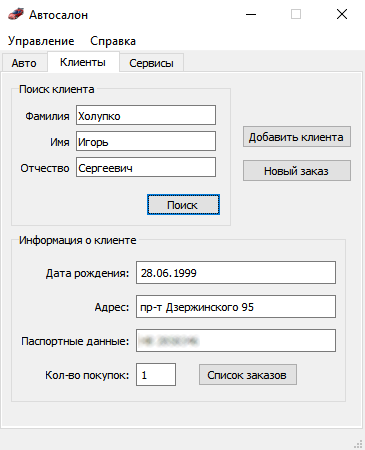


Рисунок 5.6 – Главное окно с активной вкладкой "Клиенты" и результатом поиска

При нажатии на кнопку "Добавить клиента" появляется диалоговое окно, поля которого необходимо заполнить для добавления клиента, поля "Адрес" и "Паспортные данные" для заполнения не обязательны. Окно "Добавить клиента" блокирует главное окно:

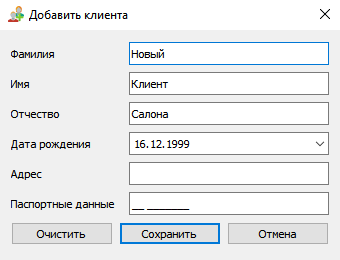


Рисунок 5.7 – Диалоговое окно добавления клиента

При нажатии клавиши "Новый заказ" будет вызвано диалоговое окно, где необходимо выбрать модель покупаемого выбранным клиентом автомобиля. Сохранение возможно только тогда, когда выбрана существующая модель. После оформления заказа от общего количество на складе проданной модели будет уменьшено. Данное окно блокирует главное:

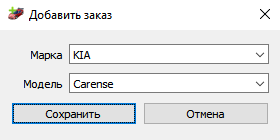


Рисунок 5.8 – Окно оформления заказа

Отображение списка заказов производится нажатием на кнопку "Список заказов", если они имеются:

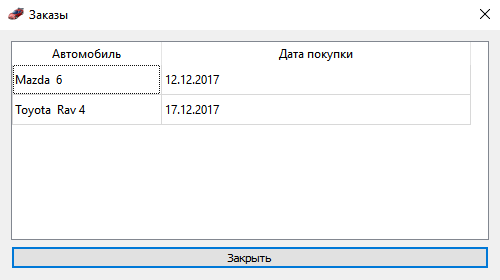


Рисунок 5.9 – Список заказов

Вкладка "Сервисы" позволяет искать автосервисы выбранной марки авто. Ещё возможность добавить новую марку:

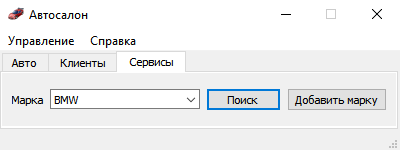


Рисунок 5.10 – Главное окно с активной вкладкой "Сервисы"

При нажатии на кнопку "Добавить марку" открывается диалоговое окно, где необходимо ввести название новой марки. Аналогично предыдущим частям обязательно нужно ввести название:

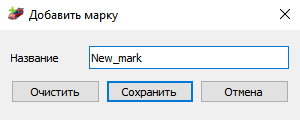


Рисунок 5.11 – Диалоговое окно "Добавить марку"

При нажатии кнопки "Поиск", если марка имеется в списке, отображается окно со списком сервисов (см. рисунок 5.4). Чтобы удалить сервис необходимо выделить один из сервисов, нажав либо на клетку с названием или адресом и нажать кнопку "Удалить сервис". Кнопка "Отменить удаление" производит отмену предыдущего удаления. Возможно отменить все удаления до закрытия окна. При нажатии кнопки "Добавить сервис" отображается диалоговое окно для добавления нового сервиса. Для сохранения необходимо заполнить все поля:

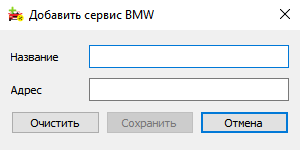


Рисунок 5.12 – Окно добавления нового сервиса

В главном окне присутствует панель меню:

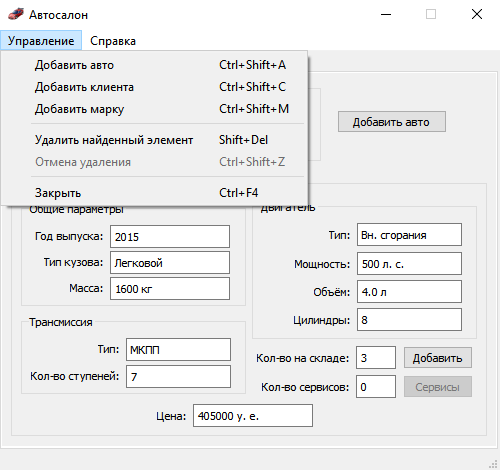


Рисунок 5.13 – Главное окно с отображение пунктов раздела меню "Управление"

Как видно из рисунка данный раздел меню содержит такие команды, как "Добавить авто", "Добавить клиента" и "Добавить марку", которые действуют аналогично одноимённым кнопкам, описанным выше. Пункт меню "Удалить найденный элемент" удаляет найденную модель, если в главном окне активна вкладка "Авто", найденного клиента и все его заказы, если в главном окне активна вкладка "Клиенты", и найденную марку и все её сервисы, если активна вкладка "Сервисы", но только, если марка не содержит моделей. Все действия удаления, произведённые в главном окне программы во время текущего сеанса программы независимо от того, переключались ли вкладки или нет, могут быть отменены при помощи пункта меню "Отмена удаления". Пункт меню "Закрыть" завершает текущий сеанс программы. Все пункты меню могут быть вызваны при помощи сочетаний клавиш, указанных напротив заголовков пунктов меню.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был разработан программный модуль "Автосалон". Приложение имеет простой функционал и не требует особых навыков. При необходимости есть возможность воспользоваться руководством пользователя. В ходе разработки и тестирования программы было проведено исследование возможных вариантов ошибок, как программных, так и пользовательских, и их устранение.

В ходе разработке был изучен фреймворк Qt, связанная с ним среда разработки Qt Creator (версии 4.4.1 – 4.5.0) и множество основных его функций. Изучены основные принципы ООП и возможности языка C++. В перспективе есть возможность расширить программу, добавив в неё дополнительный функционал и проведя возможную оптимизацию средствами нового стандарта C++20, которая не могла быть реализована в связи с огромными временными затратами.

Так же основа интерфейса программы может быть перенесена для других проектов, где используется иная функциональная основа.

Реализованное приложение может быть интересно различным компаниям, занимающимся учётом и продажей автомобилей. В перспективе может быть разработана и внедрена система работы приложения в локальной сети предприятия, система авторизации персонала такая, что для каждого сотрудника будут доступны определённые возможности, связанные с его должностью, способ хранения и обмена данных между сотрудниками.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Луцик, Ю. А. Объектно-ориентированное программирование на языке С++ : учебное пособие / Ю. А. Луцик, А. М. Ковальчук, И. В. Лукьянова. – Мн. : БГУИР, 2003 – 203 с.
2. Шилдт, Г. Самоучитель C++ : Пер. с англ. – 3-е изд. – СПб. : БХВ-Петербург, 2006. – 688 с.
3. Standard C++ Library Reference [Электронный ресурс]. – Режим доступа : http://www.cplusplus.com/reference/.
4. C++ Reference [Электронный ресурс]. – Режим доступа : http://en.cppreference.com/w/.
5. Qt 5.10 Documentation [Электронный ресурс]. – Режим доступа : http://doc.qt.io/qt-5/
6. Дейтел, Х. М. Как программировать на С++ / Х. М. Дейтел, П.Д. Дейтел : пер. с англ. – Мск. : Бином, 2007.
7. Доманов, А. Т. Стандарт предприятия. Дипломные проекты (работы) : общие требования / А. Т. Доманов, Н. И. Сороко. – Мн. : БГУИР, 2013.

ПРИЛОЖЕНИЕ А

*(обязательное)*

Диаграмма классов

ПРИЛОЖЕНИЕ Б

*(обязательное)*

Схема алгоритма функции обновления элемента выбора модели

ПРИЛОЖЕНИЕ В

*(обязательное)*

Схема алгоритма функции чтения списка сервисов из бинарного файла

ПРИЛОЖЕНИЕ Г

*(обязательное)*

Листинг кода

//old\_projects/date.hpp

#pragma once

class UBinFile;

struct Date {

int day, month, year;

Date(const int& day = 0, const int& month = 0, const int& year = 0) :

day(day), month(month), year(year) {}

Date(const Date& obj) : day(obj.day),month(obj.month),year(obj.year){}

public:

void setDate(Date date) {

day = date.day;

month = date.month;

year = date.year;

}

};

bool operator==(const Date& left, const Date& right);

bool operator<(const Date& left, const Date& right);

bool operator>(const Date& left, const Date& right);

UBinFile& operator <<(UBinFile& file, const Date& obj);

UBinFile& operator >>(UBinFile& file, Date& obj);

//old\_projects/date.cpp

#include "old\_projects/date.hpp"

#include "old\_projects/ubinfile.hpp"

bool operator==(const Date &left, const Date &right) {

return (left.day == right.day && left.month == right.month &&

left.year == right.year);

}

bool operator<(const Date &left, const Date &right) {

int a = (left.year \* 100 + left.month) \* 100 + left.day;

int b = (right.year \* 100 + right.month) \* 100 + right.day;

return a < b;

}

bool operator>(const Date &left, const Date &right) {

int a = (left.year \* 100 + left.month) \* 100 + left.day;

int b = (right.year \* 100 + right.month) \* 100 + right.day;

return a > b;

}

UBinFile& operator <<(UBinFile& file, const Date& obj) {

file << obj.day;

file << obj.month;

file << obj.year;

return file;

}

UBinFile& operator >>(UBinFile& file, Date& obj) {

file >> obj.day;

file >> obj.month;

file >> obj.year;

return file;

}

//old\_projects/ustring.hpp

#pragma once

#include <iostream>

class QString;

class UBinFile;

class UString {

int size, capacity;

char\* str;

public:

//Конструкторы

UString();

UString(const int& reserve\_size);

UString(const char\* string);

UString(const UString& object\_to\_copy);

//Деструктор

~UString();

//Фукции-члены

void resize(const int& add\_odd\_size);

void resize(const int& add\_odd\_size, const char& fill);

int getLenth() const;

int getReservedCapacity() const;

bool empty() const;

void clear();

UString& append(const char\* string);

UString& append(const UString& string);

UString& insert(const int& position, const char\* string);

UString& insert(const int& position, const UString& string);

void push\_back(const char& c);

char pop\_back();

char \* get\_cstr() const;

QString getQString() const;

//Операторы-члены

UString& operator =(const UString& right);

UString& operator =(const char\* right);

UString& operator +=(const UString& right);

UString& operator +=(const char\* right);

UString& operator ++();

UString operator ++(int);

UString& operator --();

UString operator --(int);

UString operator ()(const int& a, const int& b);

char& operator [](const int& a);

//Дружественные операторы

friend UString operator +(const UString& left, const UString& right);

friend UString operator +(const UString& left, const char\* right);

friend UString operator +(const char\* left, const UString& right);

friend bool operator ==(const UString& left, const UString& right);

friend bool operator ==(const UString& left, const char\* right);

friend bool operator ==(const char\* left, const UString& right);

friend bool operator !=(const UString& left, const UString& right);

friend bool operator !=(const UString& left, const char\* right);

friend bool operator !=(const char\* left, const UString& right);

friend bool operator >(const UString& left, const UString& right);

friend bool operator <(const UString& left, const UString& right);

//перегрузки потоков

friend std::istream& operator>>(std::istream& stream, UString& obj);

friend std::ostream& operator<<(std::ostream& stream,const UString& obj);

friend std::istream& getline(std::istream& stream, UString& obj);

friend UBinFile& operator>>(UBinFile& file, UString& obj);

friend UBinFile& operator<<(UBinFile& file, const UString& obj);

};

//Non-member functions

int strlenth(const char\* str);

//old\_projects/ustring.cpp

#include "old\_projects/ustring.hpp"

#include "old\_projects/containerexception.hpp"

#include "old\_projects/ubinfile.hpp"

#include <QString>

#include <fstream>

//Public constructors

UString::UString() {

size = capacity = 0;

str = nullptr;

}

UString::UString(const int& reserve\_size) {

size = 0;

capacity = reserve\_size;

str = new char[capacity+1]();

}

UString::UString(const char\* string) {

size = capacity = strlenth(string);

str = new char[capacity+1]();

memcpy(str, string, (capacity+1)\*sizeof(char));

}

UString::UString(const UString& object\_to\_copy) {

size = capacity = object\_to\_copy.size;

str = new char[capacity+1];

memcpy(str, object\_to\_copy.str, (capacity+1)\*sizeof(char));

}

//Destructor

UString::~UString() {

if(str != nullptr)

delete [] str;

}

//Public member functions

void UString::resize(const int& add\_odd\_size) {

capacity += add\_odd\_size;

if(capacity == add\_odd\_size)

str = new char[capacity+1]();

else {

str = (char\*)realloc(str, (capacity+1)\*sizeof(char));

str[capacity] = 0;

}

}

void UString::resize(const int& add\_odd\_size, const char& fill) {

capacity += add\_odd\_size;

if(capacity == add\_odd\_size)

str = new char[capacity+1];

else {

str = (char\*)realloc(str, (capacity+1)\*sizeof(char));

int temp\_size = size;

while(size < capacity)

str[size++] = fill;

if(fill == 0)

size = temp\_size;

str[capacity] = 0;

}

}

int UString::getLenth() const {

return size;

}

int UString::getReservedCapacity() const {

return capacity;

}

bool UString::empty() const {

return !size;

}

void UString::clear() {

if(capacity == 0)

return;

this->~UString();

str = nullptr;

size = 0;

capacity = 0;

}

UString& UString::append(const char\* string) {

const int temp\_size = strlenth(string);

if(temp\_size == 0)

return \*this;

if(capacity < size+temp\_size)

resize(temp\_size);

memcpy(str+size, string, temp\_size\*sizeof(char));

size += temp\_size;

return \*this;

}

UString& UString::append(const UString& string) {

const int temp\_size = string.size;

if(temp\_size == 0)

return \*this;

if(capacity < size+temp\_size)

resize(temp\_size);

memcpy(str+size, string.str, temp\_size\*sizeof(char));

size += temp\_size;

return \*this;

}

UString& UString::insert(const int& position, const char\* string) {

const int temp\_size = strlenth(string);

if(temp\_size == 0)

return \*this;

size += temp\_size;

if(capacity < size+temp\_size)

resize(temp\_size);

for(int i = size; i-temp\_size >= position; i--)

str[i] = str[i-temp\_size];

memcpy(str+position, string, temp\_size\*sizeof(char));

return \*this;

}

UString& UString::insert(const int& position, const UString& string) {

const int temp\_size = string.size;

if(temp\_size == 0)

return \*this;

size += temp\_size;

if(capacity < size+temp\_size)

resize(temp\_size);

for(int i = size; i-temp\_size >= position; i--)

str[i] = str[i-temp\_size];

memcpy(str+position, string.str, temp\_size\*sizeof(char));

return \*this;

}

void UString::push\_back(const char& c) {

if(size == capacity)

resize(1);

str[size++] = c;

}

char UString::pop\_back() {

const char temp = str[--size];

resize(-1);

return temp;

}

char \* UString::get\_cstr() const {

char \* out\_str = new char[size+1];

memcpy(out\_str, str, (size+1)\*sizeof(char));

return out\_str;

}

QString UString::getQString() const {

return QString(str);

}

//Member operators

UString& UString::operator =(const UString& right) {

if(this == &right)

return \*this;

clear();

append(right);

return \*this;

}

UString& UString::operator =(const char\* right) {

if(\*this == right)

return \*this;

clear();

append(right);

return \*this;

}

UString& UString::operator +=(const UString& right) {

return this->append(right);

}

UString& UString::operator +=(const char\* right) {

return this->append(right);

}

UString& UString::operator ++() {

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

++str[i];

return \*this;

}

UString UString::operator ++(int) {

UString temp(\*this);

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

++str[i];

return temp;

}

UString& UString::operator --() {

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

--str[i];

return \*this;

}

UString UString::operator --(int) {

UString temp(\*this);

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

--str[i];

return temp;

}

UString UString::operator ()(const int& a, const int& b) {

if(a >= size || a < 0 || b >= size || b < 0 || a > b)

throw ContainerException("UString::operator() - неверное значение

индекса");

UString result(b-a+1);

result.size = result.capacity;

memcpy(result.str, str+a, (b-a+1)\*sizeof(char));

return result;

}

char& UString::operator [](const int& a) {

if(a >= size || a < 0) throw ContainerException("UString::operator[] -

неверное значение индекса");

return str[a];

}

//Non-member operators

UString operator +(const UString& left, const UString& right) {

UString temp(left);

return temp.append(right);

}

UString operator +(const UString& left, const char\* right) {

UString temp(left);

return temp.append(right);

}

UString operator +(const char\* left, const UString& right) {

UString temp(left);

return temp.append(right);

}

bool operator ==(const UString& left, const UString& right) {

if(left.size != right.size)

return false;

for(int i = 0; i < left.size; i++)

if(left.str[i] != right.str[i])

return false;

return true;

}

bool operator ==(const UString& left, const char\* right) {

int temp\_size = strlenth(right);

if(left.size != temp\_size)

return false;

for(int i = 0; i < temp\_size; i++)

if(left.str[i] != right[i])

return false;

return true;

}

bool operator ==(const char\* left, const UString& right) {

int temp\_size = strlenth(left);

if(right.size != temp\_size)

return false;

for(int i = 0; i < temp\_size; i++)

if(right.str[i] != left[i])

return false;

return true;

}

bool operator !=(const UString& left, const UString& right) {

if(left.size != right.size)

return true;

for(int i = 0; i < left.size; i++)

if(left.str[i] != right.str[i])

return true;

return false;

}

bool operator !=(const UString& left, const char\* right) {

int temp\_size = strlenth(right);

if(left.size != temp\_size)

return true;

for(int i = 0; i < temp\_size; i++)

if(left.str[i] != right[i])

return true;

return false;

}

bool operator !=(const char\* left, const UString& right) {

int temp\_size = strlenth(left);

if(right.size != temp\_size)

return true;

for(int i = 0; i < temp\_size; i++)

if(right.str[i] != left[i])

return true;

return false;

}

bool operator >(const UString& left, const UString& right) {

for(int i = 0; i < left.getLenth() && i < right.getLenth(); i++) {

if(left.str[i] > right.str[i]) return true;

else if(left.str[i] < right.str[i]) return false;

}

if(left.getLenth() < right.getLenth()) return true;

return false;

}

bool operator <(const UString& left, const UString& right) {

for(int i = 0; i < left.getLenth() && i < right.getLenth(); i++) {

if(left.str[i] > right.str[i]) return false;

else if(left.str[i] < right.str[i]) return true;

}

if(left.getLenth() > right.getLenth()) return true;

return false;

}

using namespace std;

ostream& operator <<(ostream& stream, const UString& obj) {

if(!obj.empty())

stream << obj.str;

return stream;

}

istream& operator >>(istream& stream, UString& obj) {

return getline(stream, obj);

}

//Non-member functions

istream& getline(istream& stream, UString& obj) {

const int temp\_size = 3;

if(obj.capacity < temp\_size)

obj.resize(temp\_size);

stream.getline(obj.str, temp\_size+1);

stream.clear();

obj.size = (int)stream.gcount()-1;

int buffer\_size = (int)stream.rdbuf()->in\_avail();

if(buffer\_size != 0 ) {

obj.resize(buffer\_size-1);

stream.getline(obj.str+temp\_size, buffer\_size);

stream.clear(ios\_base::goodbit);

stream.sync();

obj.size += buffer\_size;

}

return stream;

}

UBinFile& operator>>(UBinFile& file, UString& obj) {

file >> obj.size;

obj.resize(obj.size);

file.read(obj.str, obj.size \* sizeof(\*obj.str));

return file;

}

UBinFile& operator<<(UBinFile& file, const UString& obj) {

file << obj.size;

file.write(obj.str, obj.size \* sizeof(\*obj.str));

return file;

}

int strlenth(const char\* str) {

if(str == nullptr)

return 0;

int i = 0;

while(str[i++] != 0);

return i-1;

}

//auto\_details/carbody.hpp

#pragma once

#include "old\_projects/ustring.hpp"

namespace cpt {

class CarBody;

}

UBinFile& operator <<(UBinFile& file, const cpt::CarBody& obj);

UBinFile& operator >>(UBinFile& file, cpt::CarBody& obj);

namespace cpt {

class CarBody {

UString type;

int weight;

public:

CarBody(const UString &type = 0, const int &weight = 0);

CarBody& operator =(const CarBody& right);

UString getType() const;

int getWeight() const;

void setType(const UString &new\_type);

void setWeight(const int &new\_weight);

friend UBinFile& ::operator <<(UBinFile& file, const CarBody& obj);

friend UBinFile& ::operator >>(UBinFile& file, CarBody& obj);

};

}

//auto\_details/carbody.cpp

#include "carbody.hpp"

#include "old\_projects/ubinfile.hpp"

using namespace cpt;

CarBody::CarBody(const UString &type, const int &weight)

: type(type), weight(weight) {}

CarBody& CarBody::operator =(const CarBody& right) {

if(this != &right) {

type = right.type;

weight = right.weight;

}

return \*this;

}

UString CarBody::getType() const {

return type;

}

int CarBody::getWeight() const {

return weight;

}

void CarBody::setType(const UString &new\_type) {

type = new\_type;

}

void CarBody::setWeight(const int &new\_weight) {

weight = new\_weight;

}

UBinFile& operator <<(UBinFile& file, const CarBody& obj) {

file << obj.type;

file << obj.weight;

return file;

}

UBinFile& operator >>(UBinFile& file, CarBody& obj) {

file >> obj.type;

file >> obj.weight;

return file;

}

//auto\_details/cartransmission.hpp

#pragma once

class UBinFile;

namespace cpt {

class CarTransmission;

}

UBinFile& operator <<(UBinFile& file, const cpt::CarTransmission& obj);

UBinFile& operator >>(UBinFile& file, cpt::CarTransmission& obj);

namespace cpt {

class CarTransmission {

bool is\_auto;

int stages\_amount;

public:

CarTransmission(const bool &is\_auto = 0, const int &stages\_amount = 0);

CarTransmission& operator =(const CarTransmission& right);

int getStagesAmount();

bool isAuto();

void setAuto(const bool &is\_auto);

void setStagesAmount(const int &new\_stages\_amouunt);

friend UBinFile& ::operator <<(UBinFile& file, const CarTransmission& obj);

friend UBinFile& ::operator >>(UBinFile& file, CarTransmission& obj);

};

}

//auto\_details/cartransmission.cpp

#include "cartransmission.hpp"

#include "old\_projects/ubinfile.hpp"

using namespace cpt;

CarTransmission::CarTransmission(const bool &is\_auto, const int

&stages\_amount)

: is\_auto(is\_auto), stages\_amount(stages\_amount) {}

CarTransmission& CarTransmission::operator =(const CarTransmission& right) {

if(this != &right) {

is\_auto = right.is\_auto;

stages\_amount = right.stages\_amount;

}

return \*this;

}

int CarTransmission::getStagesAmount() {

return stages\_amount;

}

bool CarTransmission::isAuto() {

return is\_auto;

}

void CarTransmission::setAuto(const bool &is\_auto) {

this->is\_auto = is\_auto;

}

void CarTransmission::setStagesAmount(const int &new\_stages\_amouunt) {

stages\_amount = new\_stages\_amouunt;

}

UBinFile& operator <<(UBinFile& file, const CarTransmission& obj) {

file << obj.is\_auto;

file << obj.stages\_amount;

return file;

}

UBinFile& operator >>(UBinFile& file, CarTransmission& obj) {

file >> obj.is\_auto;

file >> obj.stages\_amount;

return file;

}

//auto\_details/carengine.hpp

#pragma once

class UBinFile;

namespace cpt {

class CarEngine;

}

UBinFile& operator <<(UBinFile& file, const cpt::CarEngine& obj);

UBinFile& operator >>(UBinFile& file, cpt::CarEngine& obj);

namespace cpt {

class CarEngine {

int power;

public:

CarEngine(const int &power = 0);

CarEngine& operator =(const CarEngine& right);

int getPower() const;

void setPower(const int& new\_power);

friend UBinFile& ::operator <<(UBinFile& file,const CarEngine& obj);

friend UBinFile& ::operator >>(UBinFile& file, CarEngine& obj);

};

}

//auto\_details/carengine.cpp

#include "carengine.hpp"

#include "old\_projects/ubinfile.hpp"

using namespace cpt;

CarEngine::CarEngine(const int &power) : power(power) {}

CarEngine& CarEngine::operator =(const CarEngine& right) {

if(this != &right)

power = right.power;

return \*this;

}

int CarEngine::getPower() const {

return power;

}

void CarEngine::setPower(const int &new\_power) {

power = new\_power;

}

UBinFile& operator <<(UBinFile& file, const CarEngine& obj) {

file << obj.power;

return file;

}

UBinFile& operator >>(UBinFile& file, CarEngine& obj) {

file >> obj.power;

return file;

}

//auto\_details/electroengine.hpp

#pragma once

#include "auto\_details/carengine.hpp"

class UBinFile;

namespace cpt {

class ElectroEngine;

}

UBinFile& operator <<(UBinFile& file, const cpt::ElectroEngine& obj);

UBinFile& operator >>(UBinFile& file, cpt::ElectroEngine& obj);

namespace cpt {

class ElectroEngine : public CarEngine {

int capacity, efficiency;

public:

ElectroEngine(const int &power = 0, const int &capacity = 0, const

int &efficiency = 0);

ElectroEngine& operator =(const ElectroEngine& right);

int getCapacity() const;

int getEfficiency() const;

void setCapacity(const int &new\_capacity);

void setEfficiency(const int &new\_efficiency);

friend UBinFile& ::operator <<(UBinFile& file,

const ElectroEngine& obj);

friend UBinFile& ::operator >>(UBinFile& file, ElectroEngine& obj);

};

}

//auto\_details/electroengine.cpp

#include "electroengine.hpp"

#include "old\_projects/ubinfile.hpp"

using namespace cpt;

ElectroEngine::ElectroEngine(const int &power, const int &capacity, const int

&efficiency)

: CarEngine(power), capacity(capacity), efficiency(efficiency) {}

ElectroEngine& ElectroEngine::operator =(const ElectroEngine& right) {

if(this != &right) {

CarEngine::operator =(dynamic\_cast<const CarEngine&>(right));

capacity = right.capacity;

efficiency = right.efficiency;

}

return \*this;

}

int ElectroEngine::getCapacity() const {

return capacity;

}

int ElectroEngine::getEfficiency() const {

return efficiency;

}

void ElectroEngine::setCapacity(const int &new\_capacity) {

capacity = new\_capacity;

}

void ElectroEngine::setEfficiency(const int &new\_efficiency) {

efficiency = new\_efficiency;

}

UBinFile& operator <<(UBinFile& file, const ElectroEngine& obj) {

file << dynamic\_cast<const CarEngine&>(obj);

file << obj.capacity;

file << obj.efficiency;

return file;

}

UBinFile& operator >>(UBinFile& file, ElectroEngine& obj) {

file >> dynamic\_cast<CarEngine&>(obj);

file >> obj.capacity;

file >> obj.efficiency;

return file;

}

//auto\_details/fuelengine.hpp

#pragma once

#include "auto\_details/carengine.hpp"

class UBinFile;

namespace cpt {

class FuelEngine;

}

UBinFile& operator <<(UBinFile& file, const cpt::FuelEngine& obj);

UBinFile& operator >>(UBinFile& file, cpt::FuelEngine& obj);

namespace cpt {

class FuelEngine : public CarEngine {

int cylinders\_amount;

double volume;

public:

FuelEngine(const int &power = 0, const int &cylinders\_amount = 0,\

const double &volume = 0);

FuelEngine& operator =(const FuelEngine& right);

int getCylindersAmount() const;

double getVolume() const;

void setCylindersAmount(const int &new\_cylinders\_amount);

void setVolume(const int &new\_volume);

friend UBinFile& ::operator <<(UBinFile& file,

const FuelEngine& obj);

friend UBinFile& ::operator >>(UBinFile& file, FuelEngine& obj);

};

}

//auto\_details/fuelengine.cpp

#include "fuelengine.hpp"

#include "old\_projects/ubinfile.hpp"

using namespace cpt;

FuelEngine::FuelEngine(const int &power, const int &cylinders\_amount, const

double &volume)

: CarEngine(power), cylinders\_amount(cylinders\_amount), volume(volume) {}

FuelEngine& FuelEngine::operator =(const FuelEngine& right) {

if(this != &right) {

CarEngine::operator =(dynamic\_cast<const CarEngine&>(right));

cylinders\_amount = right.cylinders\_amount;

volume = right.volume;

}

return \*this;

}

int FuelEngine::getCylindersAmount() const {

return cylinders\_amount;

}

double FuelEngine::getVolume() const {

return volume;

}

void FuelEngine::setCylindersAmount(const int &new\_cylinders\_amount) {

cylinders\_amount = new\_cylinders\_amount;

}

void FuelEngine::setVolume(const int &new\_volume) {

volume = new\_volume;

}

UBinFile& operator <<(UBinFile& file, const FuelEngine& obj) {

file << dynamic\_cast<const CarEngine&>(obj);

file << obj.cylinders\_amount;

file << obj.volume;

return file;

}

UBinFile& operator >>(UBinFile& file, FuelEngine& obj) {

file >> dynamic\_cast<CarEngine&>(obj);

file >> obj.cylinders\_amount;

file >> obj.volume;

return file;

}

//automark.hpp

#pragma once

#include "old\_projects/ustring.hpp"

class UBinFile;

class AutoMark\_ID {

protected:

friend class BaseAutoShow;

AutoMark\_ID();

static int next\_ID;

};

class AutoMark : private AutoMark\_ID

{

int ID;

UString mark\_name;

public:

AutoMark();

AutoMark(const AutoMark& obj);

explicit AutoMark(const UString &name);

AutoMark& operator =(const AutoMark& right);

int getID() const;

UString getName() const;

bool operator <(const AutoMark& right) const;

bool operator ==(const AutoMark& right) const;

friend UBinFile& operator <<(UBinFile& file, const AutoMark& obj);

friend UBinFile& operator >>(UBinFile& file, AutoMark& obj);

};

//automark.cpp

#include "automark.hpp"

#include "old\_projects/ubinfile.hpp"

AutoMark\_ID::AutoMark\_ID() {}

int AutoMark\_ID::next\_ID = -1;

AutoMark::AutoMark() : ID(-1), mark\_name(0) {}

AutoMark::AutoMark(const AutoMark &obj) : ID(obj.ID), mark\_name(obj.mark\_name) {}

AutoMark::AutoMark(const UString &name) : ID(++next\_ID), mark\_name(name) {}

AutoMark& AutoMark::operator =(const AutoMark& right) {

if(this != &right) {

ID = right.ID;

mark\_name = right.mark\_name;

}

return \*this;

}

int AutoMark::getID() const {

return ID;

}

UString AutoMark::getName() const {

return mark\_name;

}

bool AutoMark::operator <(const AutoMark& right) const {

return mark\_name < right.mark\_name;

}

bool AutoMark::operator ==(const AutoMark& right) const {

return mark\_name == right.mark\_name;

}

UBinFile& operator <<(UBinFile& file, const AutoMark& obj) {

file << obj.ID;

file << obj.mark\_name;

return file;

}

UBinFile& operator >>(UBinFile& file, AutoMark& obj) {

file >> obj.ID;

file >> obj.mark\_name;

return file;

}

//autoservice.hpp

#pragma once

#include "old\_projects/ustring.hpp"

class UBinFile;

class AutoService

{

int mark\_ID;

UString service\_name;

UString address;

public:

AutoService(const AutoService& obj);

AutoService(const int& mark\_ID = -1, const UString &name = 0, const

UString &address = 0);

AutoService& operator =(const AutoService& right);

int getMarkID() const;

UString getAddress() const;

UString getName() const;

bool operator <(const AutoService& right) const;

bool operator ==(const AutoService& right) const;

friend UBinFile& operator <<(UBinFile& file, const AutoService& obj);

friend UBinFile& operator >>(UBinFile& file, AutoService& obj);

};

//autoservice.cpp

#include "autoservice.hpp"

#include "old\_projects/ubinfile.hpp"

AutoService::AutoService(const AutoService &obj)

: mark\_ID(obj.mark\_ID), service\_name(obj.service\_name),

address(obj.address) {}

AutoService::AutoService(const int& mark\_ID, const UString &name, const

UString &address)

: mark\_ID(mark\_ID), service\_name(name), address(address) {}

AutoService& AutoService::operator =(const AutoService& right) {

if(this != &right) {

mark\_ID = right.mark\_ID;

service\_name = right.service\_name;

address = right.address;

}

return \*this;

}

int AutoService::getMarkID() const {

return mark\_ID;

}

UString AutoService::getAddress() const {

return address;

}

UString AutoService::getName() const {

return service\_name;

}

bool AutoService::operator <(const AutoService& right) const {

return service\_name < right.service\_name;

}

bool AutoService::operator ==(const AutoService& right) const {

return service\_name == right.service\_name;

}

UBinFile& operator <<(UBinFile& file, const AutoService& obj) {

file << obj.mark\_ID;

file << obj.service\_name;

file << obj.address;

return file;

}

UBinFile& operator >>(UBinFile& file, AutoService& obj) {

file >> obj.mark\_ID;

file >> obj.service\_name;

file >> obj.address;

return file;

}

//human.hpp

#pragma once

#include "old\_projects/ustring.hpp"

#include "old\_projects/date.hpp"

class UBinFile;

class Human

{

UString surname, firstname, secondname;

Date birthday;

public:

Human(const Human& obj);

Human(const UString& surname = 0, const UString& firstname = 0,

const UString& secondname = 0, const Date& birthday = Date());

Human& operator =(const Human& right);

Date getBirthday() const;

UString getSurname() const;

UString getFirstname() const;

UString getSecondname() const;

void setBirthday(const Date &new\_birthday);

void setSurname(const UString &new\_surname);

void setFirstname(const UString &new\_firstname);

void setSecondname(const UString &new\_secondname);

bool operator <(const Human& right) const;

bool operator ==(const Human& right) const;

friend UBinFile& operator <<(UBinFile& file, const Human& obj);

friend UBinFile& operator >>(UBinFile& file, Human& obj);

};

//human.cpp

#include "human.hpp"

#include "old\_projects/ubinfile.hpp"

Human::Human(const Human &obj)

: surname(obj.surname), firstname(obj.firstname),

secondname(obj.secondname), birthday(obj.birthday) {}

Human::Human(const UString &surname, const UString &firstname,

const UString &secondname, const Date &birthday)

: surname(surname), firstname(firstname),

secondname(secondname), birthday(birthday) {}

Human& Human::operator =(const Human& right) {

if(this != &right) {

surname = right.surname;

firstname = right.firstname;

secondname = right.secondname;

birthday = right.birthday;

}

return \*this;

}

Date Human::getBirthday() const {

return birthday;

}

UString Human::getSurname() const {

return surname;

}

UString Human::getFirstname() const {

return firstname;

}

UString Human::getSecondname() const {

return secondname;

}

void Human::setBirthday(const Date &new\_birthday) {

birthday = new\_birthday;

}

void Human::setSurname(const UString &new\_surname) {

surname = new\_surname;

}

void Human::setFirstname(const UString &new\_firstname) {

firstname = new\_firstname;

}

void Human::setSecondname(const UString &new\_secondname) {

secondname = new\_secondname;

}

bool Human::operator <(const Human& right) const {

return surname < right.surname ||

(surname == right.surname && firstname < right.firstname) ||

(surname == right.surname && firstname == right.firstname &&

secondname < right.secondname) ||

(surname == right.surname && firstname == right.firstname &&

secondname == right.secondname && birthday < right.birthday);

}

bool Human::operator ==(const Human& right) const {

return surname == right.surname &&

firstname == right.firstname &&

secondname == right.secondname;

}

UBinFile& operator <<(UBinFile& file, const Human& obj) {

file << obj.surname;

file << obj.firstname;

file << obj.secondname;

file << obj.birthday;

return file;

}

UBinFile& operator >>(UBinFile& file, Human& obj) {

file >> obj.surname;

file >> obj.firstname;

file >> obj.secondname;

file >> obj.birthday;

return file;

}

//client.hpp

#pragma once

#include "human.hpp"

class Client\_ID {

protected:

friend class BaseAutoShow;

Client\_ID();

static int next\_ID;

};

class Client : public Human, private Client\_ID {

int ID;

UString passport;

UString address;

public:

Client();

Client(const Client& obj);

Client(const UString &surname, const UString &firstname,

const UString &secondname, const Date &birthday,

const UString &passport, const UString &address);

Client& operator =(const Client& right);

int getID() const;

UString getPassport() const;

UString getAddress() const;

void setPassport(const UString& new\_passport);

void setAddress(const UString& new\_address);

bool operator <(const Client& right) const;

bool operator ==(const Client& right) const;

friend UBinFile& operator <<(UBinFile& file, const Client& obj);

friend UBinFile& operator >>(UBinFile& file, Client& obj);

};

//client.cpp

#include "client.hpp"

#include "old\_projects/ubinfile.hpp"

Client\_ID::**Client\_ID**() {}

int Client\_ID::next\_ID = -1;

Client::**Client**() : Human(), ID(-1), passport(0), address(0) {}

Client::**Client**(const Client &obj)

: Human(dynamic\_cast<const Human&>(obj)), ID(obj.ID),

passport(obj.passport), address(obj.address) {}

Client::**Client**(const UString &surname, const UString &firstname,

const UString &secondname, const Date &birthday,

const UString &passport, const UString &address)

: Human(surname, firstname, secondname, birthday), ID(++next\_ID),

passport(passport), address(address) {}

Client& Client::**operator** =(const Client& right) {

if(this != &right) {

Human::operator =(dynamic\_cast<const Human&>(right));

ID = right.ID;

passport = right.passport;

address = right.address;

}

return \*this;

}

int Client::**getID**() const {

return ID;

}

UString Client::**getPassport**() const {

return passport;

}

UString Client::**getAddress**() const {

return address;

}

void Client::**setPassport**(const UString& new\_passport) {

passport = new\_passport;

}

void Client::**setAddress**(const UString& new\_address) {

address = new\_address;

}

bool Client::**operator** <(const Client& right) const {

return Human::operator <(right);

}

bool Client::**operator** ==(const Client& right) const {

return Human::operator ==(right);

}

UBinFile& **operator** <<(UBinFile& file, const Client& obj) {

file << obj.ID;

file << dynamic\_cast<const Human&>(obj);

file << obj.passport;

file << obj.address;

return file;

}

UBinFile& **operator** >>(UBinFile& file, Client& obj) {

file >> obj.ID;

file >> dynamic\_cast<Human&>(obj);

file >> obj.passport;

file >> obj.address;

return file;

}

//order.hpp

#pragma once

#include "old\_projects/date.hpp"

class Order\_ID {

protected:

friend class BaseAutoShow;

Order\_ID();

static int next\_ID;

};

class Order : private Order\_ID

{

int ID;

int Client\_ID;

int Car\_ID;

Date purchase\_date;

public:

Order();

Order(const Order& obj);

Order(const int &client, const int &car, const Date &purchase\_date);

void setClientID(const int& client\_ID);

int getCarID() const;

int getClientID() const;

Date getPurchaseDate() const;

void setPurchaseDate(const Date& new\_purchase\_date);

bool operator <(const Order& right) const;

bool operator ==(const Order& right) const;

friend UBinFile& operator <<(UBinFile& file, const Order& obj);

friend UBinFile& operator >>(UBinFile& file, Order& obj);

};

//order.cpp

#include "order.hpp"

#include "old\_projects/ubinfile.hpp"

Order\_ID::Order\_ID() {}

int Order\_ID::next\_ID = -1;

Order::Order() : ID(-1), Client\_ID(0), Car\_ID(0), purchase\_date(Date()) {}

Order::Order(const Order &obj)

: ID(obj.ID), Client\_ID(obj.Client\_ID), Car\_ID(obj.Car\_ID),

purchase\_date(obj.purchase\_date) {}

Order::Order(const int &client, const int &car, const Date &purchase\_date)

: ID(++next\_ID), Client\_ID(client), Car\_ID(car),

purchase\_date(purchase\_date) {}

void Order::setClientID(const int &client\_ID) {

Client\_ID = client\_ID;

}

int Order::getCarID() const {

return Car\_ID;

}

int Order::getClientID() const {

return Client\_ID;

}

Date Order::getPurchaseDate() const {

return purchase\_date;

}

void Order::setPurchaseDate(const Date& new\_purchase\_date) {

purchase\_date = new\_purchase\_date;

}

bool Order::operator <(const Order& right) const {

return purchase\_date < right.purchase\_date;

}

bool Order::operator ==(const Order& right) const {

return purchase\_date == right.purchase\_date;

}

UBinFile& operator <<(UBinFile& file, const Order& obj) {

file << obj.ID;

file << obj.Client\_ID;

file << obj.Car\_ID;

file << obj.purchase\_date;

return file;

}

UBinFile& operator >>(UBinFile& file, Order& obj) {

file >> obj.ID;

file >> obj.Client\_ID;

file >> obj.Car\_ID;

file >> obj.purchase\_date;

return file;

}

//car\_id.hpp

#pragma once

class Car\_ID {

protected:

friend class BaseAutoShow;

Car\_ID();

static int next\_ID;

};

//car\_id.cpp

#include "car\_id.hpp"

Car\_ID::Car\_ID() {}

int Car\_ID::next\_ID = -1;

//car.hpp

#pragma once

#include "auto\_details/carbody.hpp"

#include "auto\_details/electroengine.hpp"

#include "auto\_details/fuelengine.hpp"

#include "auto\_details/cartransmission.hpp"

#include "automark.hpp"

#include "car\_id.hpp"

class UBinFile;

template <class EngineType>

class Car : private Car\_ID {

int ID;

int mark\_ID;

UString model;

cpt::CarBody \* body;

EngineType \* engine;

cpt::CarTransmission \* transmission;

int year;

int price;

int amount;

public:

Car();

Car(const Car<EngineType>& obj);

Car(const int &mark\_ID, const UString &model, const cpt::CarBody &body,

const EngineType &engine, const cpt::CarTransmission &transmission,

const int &year, const int &price, const int &amount);

~Car();

Car<EngineType>& operator =(const Car<EngineType>& right);

int getID() const;

int getMarkID() const;

UString getModel() const;

int getYear() const;

int getPrice() const;

int getAmount() const;

cpt::CarBody \* getBody() const;

EngineType \* getEngine() const;

cpt::CarTransmission \* getTransmission() const;

void setModel(const UString& new\_model);

void setYear(const int& new\_year);

void setPrice(const int& new\_price);

void setAmount(const int& new\_amount);

void setBody(const cpt::CarBody& new\_body);

void setEngine(const EngineType& new\_engine);

void setTransmission(const cpt::CarTransmission& new\_transmission);

bool operator <(const Car<EngineType>& right) const;

bool operator ==(const Car<EngineType>& right) const;

template<class ET>

friend UBinFile& operator <<(UBinFile& file, const Car<ET>& obj);

template<class ET>

friend UBinFile& operator >>(UBinFile& file, Car<ET>& obj);

};

#include "car.cpp"

//car.cpp

#pragma once

#include "car.hpp"

using namespace cpt;

template <class EngineType>

Car<EngineType>::Car()

: ID(-1), mark\_ID(-1), model(0), body(new CarBody),

engine(new EngineType), transmission(new CarTransmission), year(0) {

static\_assert(std::is\_same<EngineType, ElectroEngine>::value ||

std::is\_same<EngineType, FuelEngine>::value,

"EngineType should be ElectroEngine or FuelEngine!");

}

template <class EngineType>

Car<EngineType>::Car(const Car<EngineType>& obj)

: ID(obj.ID), mark\_ID(obj.mark\_ID), model(obj.model),

body(new CarBody(\*obj.body)), engine(new EngineType(\*obj.engine)),

transmission(new CarTransmission(\*obj.transmission)), year(obj.year),

price(obj.price), amount(obj.amount) {}

template <class EngineType>

Car<EngineType>::Car(const int &mark\_ID, const UString &model,

const CarBody &body, const EngineType &engine,

const CarTransmission &transmission, const int &year,

const int &price, const int &amount)

: ID(++next\_ID), mark\_ID(mark\_ID), model(model), body(new CarBody(body)),

engine(new EngineType(engine)),

transmission(new CarTransmission(transmission)), year(year),

price(price), amount(amount) {

static\_assert(std::is\_same<EngineType, ElectroEngine>::value ||

std::is\_same<EngineType, FuelEngine>::value,

"EngineType should be ElectroEngine or FuelEngine!");

}

template <class EngineType>

Car<EngineType>::~Car() {

if(body != nullptr) delete body;

body = nullptr;

if(engine != nullptr) delete engine;

engine = nullptr;

if(transmission != nullptr) delete transmission;

transmission = nullptr;

}

template <class EngineType>

Car<EngineType>& Car<EngineType>::operator =(const Car<EngineType>& right) {

if(this != &right) {

ID = right.ID;

mark\_ID = right.mark\_ID;

model = right.model;

body = new CarBody(\*right.body);

engine = new EngineType(\*right.engine);

transmission = new CarTransmission(\*right.transmission);

year = right.year;

price = right.price;

amount = right.amount;

}

return \*this;

}

template <class EngineType>

int Car<EngineType>::getID() const {

return ID;

}

template <class EngineType>

int Car<EngineType>::getMarkID() const {

return mark\_ID;

}

template <class EngineType>

int Car<EngineType>::getYear() const {

return year;

}

template <class EngineType>

int Car<EngineType>::getPrice() const {

return price;

}

template <class EngineType>

int Car<EngineType>::getAmount() const {

return amount;

}

template <class EngineType>

UString Car<EngineType>::getModel() const {

return model;

}

template <class EngineType>

CarBody \* Car<EngineType>::getBody() const {

return body;

}

template <class EngineType>

EngineType \* Car<EngineType>::getEngine() const {

return engine;

}

template <class EngineType>

CarTransmission \* Car<EngineType>::getTransmission() const {

return transmission;

}

template <class EngineType>

void Car<EngineType>::setModel(const UString& new\_model) {

model = new\_model;

}

template <class EngineType>

void Car<EngineType>::setYear(const int& new\_year) {

year = new\_year;

}

template <class EngineType>

void Car<EngineType>::setPrice(const int& new\_price) {

price = new\_price;

}

template <class EngineType>

void Car<EngineType>::setAmount(const int& new\_amount) {

amount = new\_amount;

}

template <class EngineType>

void Car<EngineType>::setBody(const CarBody& new\_body) {

\*body = new\_body;

}

template <class EngineType>

void Car<EngineType>::setEngine(const EngineType& new\_engine) {

\*engine = new\_engine;

}

template <class EngineType>

void Car<EngineType>::setTransmission(const CarTransmission& new\_transmission) {

\*transmission = new\_transmission;

}

template <class EngineType>

bool Car<EngineType>::operator <(const Car<EngineType>& right) const {

return model < right.model;

}

template <class EngineType>

bool Car<EngineType>::operator ==(const Car<EngineType>& right) const {

return model == right.model;

}

template<class ET>

UBinFile& operator <<(UBinFile& file, const Car<ET>& obj) {

file << obj.ID;

file << obj.mark\_ID;

file << obj.model;

file << \*obj.body;

file << \*obj.engine;

file << \*obj.transmission;

file << obj.year;

file << obj.price;

file << obj.amount;

return file;

}

template<class ET>

UBinFile& operator >>(UBinFile& file, Car<ET>& obj) {

file >> obj.ID;

file >> obj.mark\_ID;

file >> obj.model;

if(obj.body != nullptr) delete obj.body;

obj.body = new CarBody;

file >> \*obj.body;

if(obj.engine != nullptr) delete obj.engine;

obj.engine = new ET;

file >> \*obj.engine;

if(obj.transmission != nullptr) delete obj.transmission;

obj.transmission = new CarTransmission;

file >> \*obj.transmission;

file >> obj.year;

file >> obj.price;

file >> obj.amount;

return file;

}

//baseautoshow.hpp

#pragma once

#include <vector>

#include "automark.hpp"

#include "autoservice.hpp"

#include "car.hpp"

#include "client.hpp"

#include "order.hpp"

class BaseAutoShow {

private:

bool is\_all\_cars;

protected:

const static UString marks\_file;

const static UString clients\_file;

const static UString orders\_file;

const static UString services\_file;

const static UString ecars\_file;

const static UString fcars\_file;

const static UString ids\_file;

int current\_mark;

int current\_client;

std::vector<AutoMark> marks;

std::vector<Client> clients;

std::vector<Order> orders;

std::vector<AutoService> services;

std::vector<Car<cpt::ElectroEngine>> ecars;

std::vector<Car<cpt::FuelEngine>> fcars;

public:

BaseAutoShow();

~BaseAutoShow();

bool downloadMarks();

bool downloadClients();

void downloadOrders();

void downloadServices();

void downloadCars();

void downloadAllCars();

void uploadMarks() const;

void uploadClients() const;

void uploadOrders() const;

void uploadServices() const;

void uploadECars() const;

void uploadFCars() const;

void uploadAllCars() const;

};

//baseautoshow.cpp

#include <algorithm>

#include "baseautoshow.hpp"

#include "old\_projects/ustring.hpp"

#include "old\_projects/ubinfile.hpp"

#include "old\_projects/fstreamexception.hpp"

using namespace std;

const UString BaseAutoShow::marks\_file = "markslist.dat";

const UString BaseAutoShow::clients\_file = "clientslist.dat";

const UString BaseAutoShow::orders\_file = "orderslist.dat";

const UString BaseAutoShow::services\_file = "serviceslist.dat";

const UString BaseAutoShow::ecars\_file = "ecarslist.dat";

const UString BaseAutoShow::fcars\_file = "fcarslist.dat";

const UString BaseAutoShow::ids\_file = "idslast.bin";

BaseAutoShow::BaseAutoShow()

: current\_mark(-1), current\_client(-1), marks(0), clients(0), orders(0),

services(0), ecars(0), fcars(0) {

UBinFile file(ids\_file.get\_cstr(), ios\_base::in);

if(file.is\_open()) {

file >> AutoMark\_ID::next\_ID;

file >> Car\_ID::next\_ID;

file >> Client\_ID::next\_ID;

file >> Order\_ID::next\_ID;

file.close();

}

}

BaseAutoShow::~BaseAutoShow() {

marks.clear();

services.clear();

ecars.clear();

fcars.clear();

clients.clear();

orders.clear();

UBinFile file(ids\_file.get\_cstr(), ios\_base::out);

file << AutoMark\_ID::next\_ID;

file << Car\_ID::next\_ID;

file << Client\_ID::next\_ID;

file << Order\_ID::next\_ID;

file.close();

}

bool BaseAutoShow::downloadMarks() {

UBinFile file(marks\_file.get\_cstr(), ios\_base::in);

if(!file.is\_open()) return false;

if(file.rdbuf()->in\_avail() < 4)

throw FstreamException("файл повреждён!");

marks.clear();

int size;

file >> size;

marks.resize(size);

for(int i = 0; i < size; ++i) file >> marks[i];

file.close();

sort(marks.begin(), marks.end());

return true;

}

bool BaseAutoShow::downloadClients() {

UBinFile file(clients\_file.get\_cstr(), ios\_base::in);

if(!file.is\_open()) return false;

if(file.rdbuf()->in\_avail() < 4)

throw FstreamException("файл повреждён!");

clients.clear();

int size;

file >> size;

clients.resize(size);

for(int i = 0; i < size; ++i) file >> clients[i];

file.close();

return true;

}

void BaseAutoShow::downloadOrders() {

UBinFile file(orders\_file.get\_cstr(), ios\_base::in);

if(!file.is\_open()) return;

if(file.rdbuf()->in\_avail() < 4)

throw FstreamException("файл повреждён!");

orders.clear();

int size;

file >> size;

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

Order temp;

file >> temp;

if(temp.getClientID() == current\_client)

orders.push\_back(temp);

}

file.close();

sort(orders.begin(), orders.end());

}

void BaseAutoShow::downloadServices() {

if(current\_mark < 0) {

services.clear();

return;

}

UBinFile file(services\_file.get\_cstr(), ios\_base::in);

if(!file.is\_open()) return;

services.clear();

for( ; file.rdbuf()->in\_avail() > 0; ) {

AutoService temp;

file >> temp;

if(temp.getMarkID() == current\_mark)

services.push\_back(temp);

}

file.close();

sort(services.begin(), services.end());

}

void BaseAutoShow::downloadCars() {

if(current\_mark < 0) {

ecars.clear();

fcars.clear();

return;

}

UBinFile file(ecars\_file.get\_cstr(), ios\_base::in);

if(file.is\_open()) {

ecars.clear();

for( ; file.rdbuf()->in\_avail() > 0; ) {

Car<cpt::ElectroEngine> temp;

file >> temp;

if(temp.getMarkID() == current\_mark)

ecars.push\_back(temp);

}

file.close();

}

file.open(fcars\_file.get\_cstr(), ios\_base::in);

if(file.is\_open()) {

fcars.clear();

for( ; file.rdbuf()->in\_avail() > 0; ) {

Car<cpt::FuelEngine> temp;

file >> temp;

if(temp.getMarkID() == current\_mark)

fcars.push\_back(temp);

}

file.close();

}

is\_all\_cars = false;

}

void BaseAutoShow::downloadAllCars() {

UBinFile file(ecars\_file.get\_cstr(), ios\_base::in);

if(file.is\_open()) {

ecars.clear();

for( ; file.rdbuf()->in\_avail() > 0; ) {

Car<cpt::ElectroEngine> temp;

file >> temp;

ecars.push\_back(temp);

}

file.close();

}

file.open(fcars\_file.get\_cstr(), ios\_base::in);

if(file.is\_open()) {

fcars.clear();

for( ; file.rdbuf()->in\_avail() > 0; ) {

Car<cpt::FuelEngine> temp;

file >> temp;

fcars.push\_back(temp);

}

file.close();

}

is\_all\_cars = true;

}

void BaseAutoShow::uploadMarks() const {

UBinFile file(marks\_file.get\_cstr(), ios\_base::out);

if(!file.is\_open()) return;

int size = marks.size();

file << size;

for(int i = 0; i < size; ++i) file << marks[i];

file.close();

}

void BaseAutoShow::uploadClients() const {

UBinFile file(clients\_file.get\_cstr(), ios\_base::out);

if(!file.is\_open()) return;

int size = clients.size();

file << size;

for(int i = 0; i < size; ++i) file << clients[i];

file.close();

}

void BaseAutoShow::uploadOrders() const {

UBinFile file(orders\_file.get\_cstr(), ios\_base::in);

vector<Order> out\_orders;

if(file.is\_open()) {

int size;

file >> size;

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

Order temp;

file >> temp;

if(temp.getClientID() != current\_client)

out\_orders.push\_back(temp);

}

file.close();

}

out\_orders.insert(out\_orders.end(), orders.cbegin(), orders.cend());

sort(out\_orders.begin(), out\_orders.end());

file.open(orders\_file.get\_cstr(), ios\_base::out);

if(!file.is\_open()) return;

int size = out\_orders.size();

file << size;

for(int i = 0; i < size; ++i) file << out\_orders[i];

file.close();

}

void BaseAutoShow::uploadServices() const {

UBinFile file(services\_file.get\_cstr(), ios\_base::in);

vector<AutoService> out\_services;

if(file.is\_open()) {

for( ; file.rdbuf()->in\_avail() > 0; ) {

AutoService temp;

file >> temp;

if(temp.getMarkID() != current\_mark)

out\_services.push\_back(temp);

}

file.close();

}

out\_services.insert(out\_services.end(), services.cbegin(),

services.cend());

sort(out\_services.begin(), out\_services.end());

file.open(services\_file.get\_cstr(), ios\_base::out);

if(!file.is\_open()) return;

int size = out\_services.size();

for(int i = 0; i < size; ++i) file << out\_services[i];

file.close();

}

void BaseAutoShow::uploadECars() const {

UBinFile file(ecars\_file.get\_cstr(), ios\_base::in);

vector<Car<cpt::ElectroEngine>> out\_ecars;

if(file.is\_open() && !is\_all\_cars) {

for( ; file.rdbuf()->in\_avail() > 0; ) {

Car<cpt::ElectroEngine> temp;

file >> temp;

if(temp.getMarkID() != current\_mark)

out\_ecars.push\_back(temp);

}

file.close();

}

out\_ecars.insert(out\_ecars.end(), ecars.cbegin(), ecars.cend());

sort(out\_ecars.begin(), out\_ecars.end());

file.open(ecars\_file.get\_cstr(), ios\_base::out);

if(!file.is\_open()) return;

int size = out\_ecars.size();

for(int i = 0; i < size; ++i) file << out\_ecars[i];

file.close();

}

void BaseAutoShow::uploadFCars() const {

UBinFile file(fcars\_file.get\_cstr(), ios\_base::in);

vector<Car<cpt::FuelEngine>> out\_fcars;

if(file.is\_open() && !is\_all\_cars) {

for( ; file.rdbuf()->in\_avail() > 0; ) {

Car<cpt::FuelEngine> temp;

file >> temp;

if(temp.getMarkID() != current\_mark)

out\_fcars.push\_back(temp);

}

file.close();

}

out\_fcars.insert(out\_fcars.end(), fcars.cbegin(), fcars.cend());

sort(out\_fcars.begin(), out\_fcars.end());

file.open(fcars\_file.get\_cstr(), ios\_base::out);

if(!file.is\_open()) return;

int size = out\_fcars.size();

for(int i = 0; i < size; ++i) file << out\_fcars[i];

file.close();

}

void BaseAutoShow::uploadAllCars() const {

if(!is\_all\_cars) {

uploadECars();

uploadFCars();

return;

}

UBinFile file(ecars\_file.get\_cstr(), ios\_base::out);

if(!file.is\_open()){}

int size = ecars.size();

for(int i = 0; i < size; ++i) file << ecars[i];

file.close();

file.open(fcars\_file.get\_cstr(), ios\_base::out);

if(!file.is\_open()){}

size = fcars.size();

for(int i = 0; i < size; ++i) file << fcars[i];

file.close();

}

//old\_projects/ufile.hpp

#pragma once

#include <fstream>

class UFile : public std::fstream {

public:

UFile();

explicit UFile(const char\* filename, std::ios\_base::openmode mode =

std::ios\_base::in | std::ios\_base::out);

template <class T>

std::ostream& operator <<(const T &obj) {

return std::fstream::operator <<(obj);

}

template <class T>

std::istream& operator >>(T &obj) {

return std::fstream::operator >>(obj);

}

};

//old\_projects/ufile.cpp

#include "old\_projects/ufile.hpp"

UFile::UFile() : std::fstream() {}

UFile::UFile(const char\* filename, std::ios\_base::openmode mode)

: std::fstream(filename, mode) {}

//old\_projects/ubinfile.hpp

#pragma once

#include "/old\_projects/ufile.hpp"

class UBinFile : public UFile {

public:

UBinFile();

explicit UBinFile(const char\* filename, std::ios\_base::openmode mode = std::ios\_base::in | std::ios\_base::out);

void open(const char\* filename, std::ios\_base::openmode mode = std::ios\_base::in | std::ios\_base::out);

UBinFile& operator <<(const bool &obj);

UBinFile& operator >>(bool &obj);

UBinFile& operator <<(const int &obj);

UBinFile& operator >>(int &obj);

UBinFile& operator <<(const double &obj);

UBinFile& operator >>(double &obj);

UBinFile& operator <<(const float &obj);

UBinFile& operator >>(float &obj);

};

//old\_projects/ubinfile.cpp

#include "old\_projects/ubinfile.hpp"

UBinFile::UBinFile() : UFile() {}

UBinFile::UBinFile(const char \*filename, std::ios\_base::openmode mode)

: UFile() {

open(filename, mode);

}

void UBinFile::open(const char \*filename, std::ios\_base::openmode mode) {

mode |= binary;

UFile::open(filename, mode);

}

UBinFile& UBinFile::operator <<(const bool &obj) {

write(reinterpret\_cast<const char\*>(&obj), sizeof(obj));

return \*this;

}

UBinFile& UBinFile::operator >>(bool &obj) {

read(reinterpret\_cast<char\*>(&obj), sizeof(obj));

return \*this;

}

UBinFile& UBinFile::operator <<(const int &obj) {

write(reinterpret\_cast<const char\*>(&obj), sizeof(obj));

return \*this;

}

UBinFile& UBinFile::operator >>(int &obj) {

read(reinterpret\_cast<char\*>(&obj), sizeof(obj));

return \*this;

}

UBinFile& UBinFile::operator <<(const double &obj) {

write(reinterpret\_cast<const char\*>(&obj), sizeof(obj));

return \*this;

}

UBinFile& UBinFile::operator >>(double &obj) {

read(reinterpret\_cast<char\*>(&obj), sizeof(obj));

return \*this;

}

UBinFile& UBinFile::operator <<(const float &obj) {

write(reinterpret\_cast<const char\*>(&obj), sizeof(obj));

return \*this;

}

UBinFile& UBinFile::operator >>(float &obj) {

read(reinterpret\_cast<char\*>(&obj), sizeof(obj));

return \*this;

}

//old\_projects/myexception.hpp

#pragma once

#include "old\_projects/ustring.hpp"

class MyException {

protected:

UString \_WhatException;

public:

MyException() throw();

MyException(const UString& what) throw();

virtual const UString what() const;

};

//old\_projects/myexception.cpp

#include "old\_projects/myexception.hpp"

MyException::MyException() throw()

: \_WhatException("Неизвестное исключение!") {}

MyException::MyException(const UString& what) throw()

: \_WhatException(what) {}

const UString MyException::what() const {

return \_WhatException;

}

//old\_projects/fstreamexception.hpp

#pragma once

#include "old\_projects/myexception.hpp"

class FstreamException : public MyException {

public:

FstreamException();

FstreamException(const UString& what);

};

//old\_projects/fstreamexception.hpp

#include "old\_projects/fstreamexception.hpp"

FstreamException::FstreamException()

: MyException("Неизвестное исключение в файловом потоке!") {}

FstreamException::FstreamException(const UString& what)

: MyException("Исключение в файловом потоке: " + what) {}

//old\_projects/containerexception.hpp

#pragma once

#include "old\_projects/myexception.hpp"

class ContainerException : public MyException {

public:

ContainerException();

ContainerException(const UString& what);

};

//old\_projects/containerexception.cpp

#include "old\_projects/containerexception.hpp"

ContainerException::ContainerException()

: MyException("Неизвестное исключение в контейнере!") {}

ContainerException::ContainerException(const UString& what)

: MyException("Исключение в контейнере: " + what) {}

//mainwindow.hpp

#pragma once

#include <QMainWindow>

#include <QTabWidget>

#include <QWidget>

#include <QMenu>

#include <QStatusBar>

#include <QComboBox>

#include <QLineEdit>

#include <QPushButton>

#include <QLabel>

#include <QTextBrowser>

#include <QGroupBox>

#include <QCloseEvent>

#include <stack>

#include "baseautoshow.hpp"

class MainWindow : public QMainWindow, private BaseAutoShow

{

Q\_OBJECT

QAction \* delete\_element, \* undo\_action;

QTabWidget \* centralZone;

QWidget \* window[3];

QMenu \* menu[2];

QStatusBar \* status\_bar;

QComboBox \* combo\_box[3];

QLineEdit \* line\_edit[3];

QPushButton \* button[10];

QLabel \* label[22];

QTextBrowser \* text\_browser[16];

QGroupBox \* group\_box[7];

std::stack<int> undo\_stack;

std::stack<Car<cpt::ElectroEngine>> deleted\_ecars;

std::stack<Car<cpt::FuelEngine>> deleted\_fcars;

std::stack<AutoMark> deleted\_marks;

std::stack<std::vector<AutoService>> deleted\_services;

std::stack<Client> deleted\_clients;

std::stack<std::vector<Order>> deleted\_orders;

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private:

void marksFromVectorToComboBox();

void modelsFromVectorToComboBox();

protected slots:

void closeEvent(QCloseEvent \*event);

private slots:

void updateModelLine(QString mark);

void updateModelSearchButton(QString model);

void updateClientSearchButton();

void findModel();

void findMarkServices();

void findClient();

void clearModelInfo();

void clearClientInfo();

void callAddAutoWindow();

void callAddCarAmount();

void callAddClientWindow();

void callAddMarkWindow();

void callAddOrder();

void callAbout();

void showServices();

void showOrders();

void changeWindowSets(int i);

void addMarkElement(QString new\_mark\_name);

void addMarkElement(AutoMark new\_mark);

void addEModelElement(Car<cpt::ElectroEngine> new\_model);

void addFModelElement(Car<cpt::FuelEngine> new\_model);

void addClientElement(Client new\_client);

void addOrderElement(Order new\_order);

void deleteCarElement();

void deleteClientElement();

void deleteMarkElement();

void undoLastAction();

signals:

void deleteActions();

};

//mainwindow.cpp

#include <iostream>

#include <QMenuBar>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QAction>

#include "mainwindow.hpp"

#include "addautowindow.hpp"

#include "addcaramountwindow.hpp"

#include "addclientwindow.hpp"

#include "addorderwindow.hpp"

#include "addmarkwindow.hpp"

#include "aboutwindow.hpp"

#include "orderslistwindow.hpp"

#include "serviceslistwindow.hpp"

#include "dateinput.hpp"

#include "old\_projects/ubinfile.hpp"

#include "old\_projects/fstreamexception.hpp"

using namespace std;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent), BaseAutoShow() {

setWindowFlags(Qt::WindowCloseButtonHint | Qt::WindowMinimizeButtonHint);

setWindowTitle(QString("Автосалон"));

setWindowIcon(QIcon(":/resources/icon.ico"));

setFixedSize(500, 440);

menu[0] = menuBar()->addMenu(QString("Управление"));

{

QAction \* add\_auto = new QAction(QString("Добавить авто"), this);

{

add\_auto->setShortcut(QKeySequence(QString("Ctrl+Shift+A")));

connect(add\_auto, SIGNAL(triggered(bool)), SLOT(callAddAutoWindow()));

connect(this, SIGNAL(deleteActions()), add\_auto, SLOT(deleteLater()));

}

QAction \* add\_client = new QAction(QString("Добавить клиента"), this);

{

add\_client->setShortcut(QKeySequence(QString("Ctrl+Shift+C")));

connect(add\_client, SIGNAL(triggered(bool)), SLOT(callAddClientWindow()));

connect(this, SIGNAL(deleteActions()), add\_client, SLOT(deleteLater()));

}

QAction \* add\_mark = new QAction(QString("Добавить марку"), this);

{

add\_mark->setShortcut(QKeySequence(QString("Ctrl+Shift+M")));

connect(add\_mark, SIGNAL(triggered(bool)), SLOT(callAddMarkWindow()));

connect(this, SIGNAL(deleteActions()), add\_mark, SLOT(deleteLater()));

}

delete\_element = new QAction(QString("Удалить найденный элемент"), this);

{

delete\_element->setShortcut(QKeySequence(QString("Shift+delete")));

delete\_element->setEnabled(false);

connect(delete\_element, SIGNAL(triggered(bool)),

SLOT(deleteCarElement()));

connect(this, SIGNAL(deleteActions()), delete\_element,

SLOT(deleteLater()));

}

undo\_action = new QAction(QString("Отмена удаления"), this);

{

undo\_action->setShortcut(QKeySequence(QString("Ctrl+Shift+Z")));

undo\_action->setEnabled(false);

connect(undo\_action, SIGNAL(triggered(bool)), SLOT(undoLastAction()));

}

QAction \* close = new QAction(QString("Закрыть"), this);

{

close->setShortcut(QKeySequence::Close);

connect(close, SIGNAL(triggered(bool)), SLOT(close()));

connect(this, SIGNAL(deleteActions()), close, SLOT(deleteLater()));

}

menu[0]->addAction(add\_auto);

menu[0]->addAction(add\_client);

menu[0]->addAction(add\_mark);

menu[0]->addSeparator();

menu[0]->addAction(delete\_element);

menu[0]->addAction(undo\_action);

menu[0]->addSeparator();

menu[0]->addAction(close);

}

menu[1] = menuBar()->addMenu(QString("Справка"));

{

QAction \* help = new QAction(QString("Окно справки"), this);

{

help->setShortcut(QKeySequence::HelpContents);

help->setEnabled(false);

}

QAction \* info = new QAction(QString("О программе"), this);

{

connect(info, SIGNAL(triggered(bool)), SLOT(callAbout()));

}

menu[1]->addAction(help);

menu[1]->addSeparator();

menu[1]->addAction(info);

}

status\_bar = statusBar();

QVBoxLayout \* window\_layout\_1 = new QVBoxLayout;

{

QHBoxLayout \* layout\_input = new QHBoxLayout;

{

group\_box[0] = new QGroupBox(QString("Выбор авто"));

{

QHBoxLayout \* searching\_layout = new QHBoxLayout;

{

QVBoxLayout \* input\_layout = new QVBoxLayout;

{

QHBoxLayout \* mark\_input\_layout = new QHBoxLayout;

{

label[0] = new QLabel(QString("Марка"));

{

label[0]->setFixedHeight(20);

}

combo\_box[0] = new QComboBox;

{

combo\_box[0]->setFixedSize(150, 20);

combo\_box[0]->setEditable(true);

combo\_box[0]->setMaxVisibleItems(10);

combo\_box[0]->setValidator(new QRegExpValidator(QRegExp(QString(

"^\\S{1}(\\s{0,1}\\S{1,}){0,}$"))));

connect(combo\_box[0], SIGNAL(currentTextChanged(QString)),

SLOT(updateModelLine(QString)));

}

mark\_input\_layout->addWidget(label[0], 45, Qt::AlignRight);

mark\_input\_layout->addWidget(combo\_box[0], 150, Qt::AlignLeft);

}

QHBoxLayout \* model\_input\_layout = new QHBoxLayout;

{

label[1] = new QLabel(QString("Модель"));

{

label[1]->setFixedHeight(20);

}

combo\_box[1] = new QComboBox;

{

combo\_box[1]->setFixedSize(150, 20);

combo\_box[1]->setEditable(true);

combo\_box[1]->setMaxVisibleItems(10);

combo\_box[1]->setEnabled(false);

combo\_box[1]->setValidator(new QRegExpValidator(QRegExp(QString(

"^\\S{1}(\\s{0,1}\\S{1,}){0,}$"))));

connect(combo\_box[1], SIGNAL(currentTextChanged(QString)),

SLOT(updateModelSearchButton(QString)));

}

model\_input\_layout->addWidget(label[1], 45, Qt::AlignRight);

model\_input\_layout->addWidget(combo\_box[1], 150, Qt::AlignLeft);

}

input\_layout->addLayout(mark\_input\_layout, 20);

input\_layout->addLayout(model\_input\_layout, 20);

}

button[0] = new QPushButton(QString("Поиск"));

{

button[0]->setEnabled(false);

button[0]->setFixedSize(75, 23);

connect(button[0], SIGNAL(clicked(bool)), SLOT(findModel()));

}

searching\_layout->addLayout(input\_layout, 150);

searching\_layout->addSpacing(10);

searching\_layout->addWidget(button[0], 75, Qt::AlignLeft);

}

group\_box[0]->setFixedSize(310, 80);

group\_box[0]->setLayout(searching\_layout);

}

button[1] = new QPushButton(QString("Добавить авто"));

{

button[1]->setFixedSize(110, 23);

connect(button[1], SIGNAL(clicked(bool)), SLOT(callAddAutoWindow()));

}

layout\_input->addWidget(group\_box[0], 310, Qt::AlignTop);

layout\_input->addSpacing(10);

layout\_input->addWidget(button[1], 90);

}

group\_box[1] = new QGroupBox(QString("Информация о авто"));

{

QVBoxLayout \* all\_info\_layout = new QVBoxLayout;

{

QHBoxLayout \* car\_info\_layout = new QHBoxLayout;

{

QVBoxLayout \* info\_groups\_1 = new QVBoxLayout;

{

group\_box[2] = new QGroupBox(QString("Общие параметры"));

{

QVBoxLayout \* group\_box\_layout = new QVBoxLayout;

QHBoxLayout \* layouts[3];

label[2] = new QLabel(QString("Год выпуска:"));

label[3] = new QLabel(QString("Тип кузова:"));

label[4] = new QLabel(QString("Масса:"));

for(int i = 2; i < 5; i++) {

label[i]->setFixedHeight(20);

text\_browser[i-2] = new QTextBrowser;

text\_browser[i-2]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[i-2]->setVerticalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[i-2]->setFixedSize(120, 23);

layouts[i-2] = new QHBoxLayout;

layouts[i-2]->addWidget(label[i], 70, Qt::AlignRight);

layouts[i-2]->addWidget(text\_browser[i-2], 120, Qt::AlignLeft);

group\_box\_layout->addLayout(layouts[i-2], 40);

}

group\_box[2]->setFixedSize(225, 105);

group\_box[2]->setLayout(group\_box\_layout);

}

group\_box[4] = new QGroupBox(QString("Трансмиссия"));

{

QVBoxLayout \* group\_box\_layout = new QVBoxLayout;

QHBoxLayout \* layouts[2];

label[9] = new QLabel(QString("Тип:"));

label[10] = new QLabel(QString("Кол-во ступеней:"));

for(int i = 9; i < 11; i++) {

label[i]->setFixedHeight(20);

text\_browser[i-2] = new QTextBrowser;

text\_browser[i-2]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[i-2]->setVerticalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[i-2]->setFixedSize(105, 23);

layouts[i-5] = new QHBoxLayout;

layouts[i-5]->addWidget(label[i], 85, Qt::AlignRight);

layouts[i-5]->addWidget(text\_browser[i-2], 105, Qt::AlignLeft);

group\_box\_layout->addLayout(layouts[i-5], 40);

}

group\_box[4]->setFixedSize(225, 80);

group\_box[4]->setLayout(group\_box\_layout);

}

info\_groups\_1->addWidget(group\_box[2]);

info\_groups\_1->addWidget(group\_box[4]);

}

QVBoxLayout \* info\_groups\_2 = new QVBoxLayout;

{

group\_box[3] = new QGroupBox(QString("Двигатель"));

{

QVBoxLayout \* group\_box\_layout = new QVBoxLayout;

QHBoxLayout \* layouts[4];

label[5] = new QLabel(QString("Тип:"));

label[6] = new QLabel(QString("Мощность:"));

label[7] = new QLabel(QString("Объём/Ёмкость:"));

label[8] = new QLabel(QString("Цилиндры/КПД:"));

for(int i = 5; i < 9; i++) {

label[i]->setFixedHeight(20);

text\_browser[i-2] = new QTextBrowser;

text\_browser[i-2]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[i-2]->setVerticalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[i-2]->setFixedSize(105, 23);

layouts[i-5] = new QHBoxLayout;

layouts[i-5]->addWidget(label[i], 85, Qt::AlignRight);

layouts[i-5]->addWidget(text\_browser[i-2], 105, Qt::AlignLeft);

group\_box\_layout->addLayout(layouts[i-5], 40);

}

group\_box[3]->setFixedSize(225, 140);

group\_box[3]->setLayout(group\_box\_layout);

}

QVBoxLayout \* info\_groups\_3 = new QVBoxLayout;

{

QHBoxLayout \* storeged\_amount = new QHBoxLayout;

{

label[11] = new QLabel(QString("Кол-во на складе:"));

{

label[11]->setFixedHeight(20);

}

text\_browser[9] = new QTextBrowser;

{

text\_browser[9]->setFixedSize(40, 23);

text\_browser[9]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[9]->setVerticalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

}

button[2] = new QPushButton(QString("Добавить"));

{

button[2]->setFixedWidth(70);

button[2]->setEnabled(false);

connect(button[2], SIGNAL(clicked(bool)),

SLOT(callAddCarAmount()));

}

storeged\_amount->addSpacing(5);

storeged\_amount->addWidget(label[11], 90, Qt::AlignRight);

storeged\_amount->addWidget(text\_browser[9], 40, Qt::AlignLeft);

storeged\_amount->addWidget(button[2], 72, Qt::AlignLeft);

}

QHBoxLayout \* services\_amount = new QHBoxLayout;

{

label[12] = new QLabel(QString("Кол-во сервисов:"));

{

label[12]->setFixedHeight(20);

}

text\_browser[10] = new QTextBrowser;

{

text\_browser[10]->setFixedSize(40, 23);

text\_browser[10]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[10]->setVerticalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

}

button[3] = new QPushButton(QString("Сервисы"));

{

button[3]->setFixedWidth(70);

button[3]->setEnabled(false);

connect(button[3], SIGNAL(clicked(bool)), SLOT(showServices()));

}

services\_amount->addSpacing(5);

services\_amount->addWidget(label[12], 90, Qt::AlignRight);

services\_amount->addWidget(text\_browser[10], 40, Qt::AlignLeft);

services\_amount->addWidget(button[3], 72, Qt::AlignLeft);

}

info\_groups\_3->addLayout(storeged\_amount);

info\_groups\_3->setAlignment(storeged\_amount, Qt::AlignLeft);

info\_groups\_3->addLayout(services\_amount);

}

info\_groups\_2->addWidget(group\_box[3]);

info\_groups\_2->addLayout(info\_groups\_3);

}

car\_info\_layout->addLayout(info\_groups\_1);

car\_info\_layout->addLayout(info\_groups\_2);

}

QHBoxLayout \* car\_price = new QHBoxLayout;

{

label[13] = new QLabel(QString("Цена:"));

{

label[13]->setFixedHeight(20);

}

text\_browser[11] = new QTextBrowser;

{

text\_browser[11]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[11]->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

text\_browser[11]->setFixedSize(120, 23);

}

car\_price->addWidget(label[13], 70, Qt::AlignRight);

car\_price->addWidget(text\_browser[11], 120, Qt::AlignLeft);

}

all\_info\_layout->addLayout(car\_info\_layout);

all\_info\_layout->addLayout(car\_price);

}

group\_box[1]->setLayout(all\_info\_layout);

}

window\_layout\_1->addLayout(layout\_input, 405);

window\_layout\_1->setAlignment(layout\_input, Qt::AlignTop);

window\_layout\_1->addWidget(group\_box[1]);

}

QVBoxLayout \* window\_layout\_2 = new QVBoxLayout;

{

QHBoxLayout \* input\_layout = new QHBoxLayout;

{

group\_box[5] = new QGroupBox(QString("Поиск клиента"));

{

QVBoxLayout \* client\_search = new QVBoxLayout;

{

QHBoxLayout \* surname\_layout = new QHBoxLayout;

{

label[14] = new QLabel(QString("Фамилия"));

{

label[14]->setFixedHeight(20);

}

line\_edit[0] = new QLineEdit;

{

line\_edit[0]->setFixedSize(140, 20);

line\_edit[0]->setValidator(new QRegExpValidator(QRegExp(QString(

"^(\\b([A-Z]{1}[a-z]{0,})\\b)((\\b([A-Z]?[a-z]{0,})\\b|

\\b(-{1})\\b){2})$|^(\\b([А-Я]{1}[а-я]{0,})\\b)

((\\b([А-Я]?[а-я]{0,})\\b|\\b(-{1})\\b){2})$"))));

connect(line\_edit[0], SIGNAL(textChanged(QString)), SLOT(updateClientSearchButton()));

}

surname\_layout->addWidget(label[14], 50, Qt::AlignRight);

surname\_layout->addWidget(line\_edit[0], 150, Qt::AlignLeft);

}

QHBoxLayout \* firstname\_layout = new QHBoxLayout;

{

label[15] = new QLabel(QString("Имя"));

{

label[15]->setFixedHeight(20);

}

line\_edit[1] = new QLineEdit;

{

line\_edit[1]->setFixedSize(140, 20);

line\_edit[1]->setValidator(new QRegExpValidator(QRegExp(QString(

"^(\\b([A-Z]{1}[a-z]{0,})\\b)$|^(\\b([А-Я|Ё]{1}

[а-я|ё]{0,})\\b)$"))));

connect(line\_edit[1], SIGNAL(textChanged(QString)),

SLOT(updateClientSearchButton()));

}

firstname\_layout->addWidget(label[15], 50, Qt::AlignRight);

firstname\_layout->addWidget(line\_edit[1], 150, Qt::AlignLeft);

}

QHBoxLayout \* secondname\_layout = new QHBoxLayout;

{

label[16] = new QLabel(QString("Отчество"));

{

label[16]->setFixedHeight(20);

}

line\_edit[2] = new QLineEdit;

{

line\_edit[2]->setFixedSize(140, 20);

line\_edit[2]->setValidator(line\_edit[1]->validator());

connect(line\_edit[2], SIGNAL(textChanged(QString)),

SLOT(updateClientSearchButton()));

}

secondname\_layout->addWidget(label[16], 50, Qt::AlignRight);

secondname\_layout->addWidget(line\_edit[2], 150, Qt::AlignLeft);

}

button[4] = new QPushButton(QString("Поиск"));

{

button[4]->setEnabled(false);

button[4]->setFixedSize(75, 23);

connect(button[4], SIGNAL(clicked(bool)), SLOT(findClient()));

}

client\_search->addLayout(surname\_layout);

client\_search->addLayout(firstname\_layout);

client\_search->addLayout(secondname\_layout);

client\_search->addSpacing(10);

client\_search->addWidget(button[4], 75, Qt::AlignRight);

}

group\_box[5]->setFixedSize(220, 145);

group\_box[5]->setLayout(client\_search);

}

QVBoxLayout \* buttons\_layout = new QVBoxLayout;

{

button[5] = new QPushButton(QString("Добавить клиента"));

{

button[5]->setFixedSize(110, 23);

connect(button[5], SIGNAL(clicked(bool)), SLOT(callAddClientWindow()));

}

button[6] = new QPushButton(QString("Новый заказ"));

{

button[6]->setEnabled(false);

button[6]->setFixedSize(110, 23);

connect(button[6], SIGNAL(clicked(bool)), SLOT(callAddOrder()));

}

buttons\_layout->addSpacing(38);

buttons\_layout->addWidget(button[5], 23, Qt::AlignRight);

buttons\_layout->addWidget(button[6], 23, Qt::AlignRight);

buttons\_layout->addSpacing(38);

}

input\_layout->addWidget(group\_box[5]);

input\_layout->addSpacing(10);

input\_layout->addLayout(buttons\_layout);

}

group\_box[6] = new QGroupBox(QString("Информация о клиенте"));

{

QVBoxLayout \* client\_info = new QVBoxLayout;

{

QHBoxLayout \* birthday\_date = new QHBoxLayout;

{

label[17] = new QLabel(QString("Дата рождения:"));

{

label[17]->setFixedHeight(20);

}

text\_browser[12] = new QTextBrowser;

{

text\_browser[12]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[12]->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

text\_browser[12]->setFixedSize(200, 23);

}

birthday\_date->addWidget(label[17], 200, Qt::AlignRight);

birthday\_date->addWidget(text\_browser[12], 200, Qt::AlignLeft);

}

QHBoxLayout \* address = new QHBoxLayout;

{

label[18] = new QLabel(QString("Адрес:"));

{

label[18]->setFixedHeight(20);

}

text\_browser[13] = new QTextBrowser;

{

text\_browser[13]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[13]->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

text\_browser[13]->setFixedSize(200, 23);

}

address->addWidget(label[18], 110, Qt::AlignRight);

address->addWidget(text\_browser[13], 200, Qt::AlignLeft);

}

QHBoxLayout \* passport = new QHBoxLayout;

{

label[19] = new QLabel(QString("Паспортные данные:"));

{

label[19]->setFixedHeight(20);

}

text\_browser[14] = new QTextBrowser;

{

text\_browser[14]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[14]->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

text\_browser[14]->setFixedSize(200, 23);

}

passport->addWidget(label[19], 110, Qt::AlignRight);

passport->addWidget(text\_browser[14], 200, Qt::AlignLeft);

}

QHBoxLayout \* purchases = new QHBoxLayout;

{

label[20] = new QLabel(QString("Кол-во покупок:"));

{

label[20]->setFixedHeight(20);

}

text\_browser[15] = new QTextBrowser;

{

text\_browser[15]->setHorizontalScrollBarPolicy(

Qt::ScrollBarAlwaysOff);

text\_browser[15]->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

text\_browser[15]->setFixedSize(40, 23);

}

button[7] = new QPushButton(QString("Список заказов"));

{

button[7]->setEnabled(false);

button[7]->setFixedWidth(100);

connect(button[7], SIGNAL(clicked(bool)), SLOT(showOrders()));

}

purchases->addWidget(label[20], 79, Qt::AlignRight);

purchases->addWidget(text\_browser[15], 40, Qt::AlignLeft);

purchases->addWidget(button[7], 100, Qt::AlignLeft);

}

client\_info->addLayout(birthday\_date);

client\_info->addLayout(address);

client\_info->addLayout(passport);

client\_info->addLayout(purchases);

}

group\_box[6]->setFixedSize(335, 170);

group\_box[6]->setLayout(client\_info);

}

window\_layout\_2->addLayout(input\_layout);

window\_layout\_2->setAlignment(input\_layout, Qt::AlignTop);

window\_layout\_2->addWidget(group\_box[6], 180, Qt::AlignTop);

}

QHBoxLayout \* window\_layout\_3 = new QHBoxLayout;

{

label[21] = new QLabel(QString("Марка"));

{

label[21]->setFixedHeight(20);

}

combo\_box[2] = new QComboBox;

{

combo\_box[2]->setFixedSize(150, 20);

combo\_box[2]->setEditable(true);

combo\_box[2]->setMaxVisibleItems(10);

combo\_box[2]->setValidator(new QRegExpValidator(QRegExp(QString(

"^\\S{1}(\\s{0,1}\\S{1,}){0,}$"))));

connect(combo\_box[2], SIGNAL(editTextChanged(QString)),

SLOT(updateModelLine(QString)));

}

button[8] = new QPushButton(QString("Поиск"));

{

button[8]->setEnabled(false);

button[8]->setFixedSize(75, 23);

connect(button[8], SIGNAL(clicked(bool)), SLOT(findMarkServices()));

}

button[9] = new QPushButton(QString("Добавить марку"));

{

button[9]->setFixedSize(100, 23);

connect(button[9], SIGNAL(clicked(bool)), SLOT(callAddMarkWindow()));

}

window\_layout\_3->addWidget(label[21], 45, Qt::AlignRight);

window\_layout\_3->addWidget(combo\_box[2], 150, Qt::AlignLeft);

window\_layout\_3->addWidget(button[8]);

window\_layout\_3->addWidget(button[9]);

}

for(int i = 0; i < 3; i++)

{

window[i] = new QWidget;

window[i]->setAutoFillBackground(true);

}

window[0]->setLayout(window\_layout\_1);

window[1]->setLayout(window\_layout\_2);

window[2]->setLayout(window\_layout\_3);

centralZone = new QTabWidget;

{

centralZone->addTab(window[0], QString("Авто"));

centralZone->addTab(window[1], QString("Клиенты"));

centralZone->addTab(window[2], QString("Сервисы"));

connect(centralZone, SIGNAL(currentChanged(int)), SLOT(changeWindowSets(int)));

}

setCentralWidget(centralZone);

try {

if(!downloadMarks()) {

UBinFile file(marks\_file.get\_cstr(), std::ios\_base::out);

file << int(0);

file.close();

file.open(services\_file.get\_cstr(), std::ios\_base::out);

file.close();

file.open(ecars\_file.get\_cstr(), std::ios\_base::out);

file.close();

file.open(fcars\_file.get\_cstr(), std::ios\_base::out);

file.close();

}

if(!downloadClients()) {

UBinFile file(clients\_file.get\_cstr(), std::ios\_base::out);

file << int(0);

file.close();

file.open(orders\_file.get\_cstr(), std::ios\_base::out);

file << int(0);

file.close();

}

}

catch(FstreamException& ex) {

status\_bar->showMessage(ex.what().getQString());

}

marksFromVectorToComboBox();

}

MainWindow::~MainWindow()

{

while(!deleted\_clients.empty())

deleted\_clients.pop();

while(!deleted\_ecars.empty())

deleted\_ecars.pop();

while(!deleted\_fcars.empty())

deleted\_fcars.pop();

while(!deleted\_marks.empty())

deleted\_marks.pop();

while(!deleted\_orders.empty())

deleted\_orders.pop();

while(!deleted\_services.empty())

deleted\_services.pop();

}

void MainWindow::marksFromVectorToComboBox() {

for(int i = 0; i < 3; i+= 2) {

int prev\_ind = combo\_box[i]->currentIndex();

combo\_box[i]->clear();

for(unsigned int j = 0; j < marks.size(); ++j)

combo\_box[i]->addItem(marks[j].getName().getQString(), QVariant(marks[j].getID()));

combo\_box[i]->setCurrentIndex(prev\_ind);

}

}

void MainWindow::modelsFromVectorToComboBox() {

combo\_box[1]->clear();

unsigned int j = 0, k = 0;

while(j < ecars.size() && k < fcars.size())

if(ecars[j].getModel() < fcars[k].getModel()) {

combo\_box[1]->addItem(ecars[j].getModel().getQString(), QVariant(QStringList({QString::number(1), QString::number(j)})));

++j;

}

else {

combo\_box[1]->addItem(fcars[k].getModel().getQString(), QVariant(QStringList({QString::number(2), QString::number(k)})));

++k;

}

while(j < ecars.size()) {

combo\_box[1]->addItem(ecars[j].getModel().getQString(), QVariant(QStringList({QString::number(1), QString::number(j)})));

++j;

}

while(k < fcars.size()) {

combo\_box[1]->addItem(fcars[k].getModel().getQString(), QVariant(QStringList({QString::number(2), QString::number(k)})));

++k;

}

combo\_box[1]->setCurrentIndex(-1);

}

void MainWindow::updateModelLine(QString mark) {

switch(centralZone->currentIndex()) {

case 0:

if(mark.isEmpty()) combo\_box[1]->clearEditText();

combo\_box[1]->setEnabled(!mark.isEmpty());

combo\_box[1]->clear();

if(combo\_box[0]->currentText() == combo\_box[0]->itemText(combo\_box[0]->currentIndex())) {

if(current\_mark != combo\_box[0]->currentData().toInt()) {

current\_mark = combo\_box[0]->currentData().toInt();

downloadCars();

downloadServices();

}

modelsFromVectorToComboBox();

}

combo\_box[1]->setCurrentIndex(-1);

break;

case 2:

button[8]->setEnabled(!mark.isEmpty());

button[8]->setDefault(!mark.isEmpty());

if(combo\_box[2]->currentText() == combo\_box[2]->itemText(combo\_box[2]->currentIndex())) {

if(current\_mark != combo\_box[2]->currentData().toInt()) {

current\_mark = combo\_box[2]->currentData().toInt();

downloadServices();

}

delete\_element->setEnabled(true);

}

else delete\_element->setEnabled(false);

break;

}

}

void MainWindow::updateModelSearchButton(QString model) {

button[0]->setEnabled(!model.isEmpty());

button[0]->setDefault(!model.isEmpty());

if(!text\_browser[0]->toPlainText().isEmpty()) clearModelInfo();

}

void MainWindow::updateClientSearchButton() {

bool check = !(line\_edit[0]->displayText().isEmpty() ||

line\_edit[1]->displayText().isEmpty() ||

line\_edit[2]->displayText().isEmpty());

button[4]->setEnabled(check);

button[4]->setDefault(check);

if(!text\_browser[12]->toPlainText().isEmpty()) clearClientInfo();

}

void MainWindow::findModel() {

if(combo\_box[1]->currentData().toStringList().empty()) {

int ind = combo\_box[0]->findText(combo\_box[0]->currentText());

if(ind < 0) {

status\_bar->showMessage("Марка не найдена!");

return;

}

QString model = combo\_box[1]->currentText();

combo\_box[0]->setCurrentIndex(ind);

updateModelLine(combo\_box[0]->currentText());

combo\_box[1]->setCurrentText(model);

combo\_box[1]->setCurrentIndex(combo\_box[1]->findText(model));

}

if(combo\_box[1]->currentIndex() < 0 ||

combo\_box[1]->currentText() !=

combo\_box[1]->itemText(combo\_box[1]->currentIndex()))

status\_bar->showMessage(QString("Модель не найдена!"), 5000);

else if(combo\_box[1]->currentData().toStringList().front().toInt() == 1) {

int ind = combo\_box[1]->currentData().toStringList().back().toInt();

text\_browser[0]->setText(QString::number(ecars[ind].getYear()));

text\_browser[1]->setText(ecars[ind].getBody()->getType().getQString());

text\_browser[2]->setText(QString::number(

ecars[ind].getBody()->getWeight()) + QString(" кг"));

text\_browser[3]->setText(QString("Электрический"));

text\_browser[4]->setText(QString::number(

ecars[ind].getEngine()->getPower()) + QString(" л. с."));

text\_browser[5]->setText(QString::number(

ecars[ind].getEngine()->getCapacity()) + QString(" кА"));

text\_browser[6]->setText(QString::number(

ecars[ind].getEngine()->getEfficiency()) + QString(" %"));

text\_browser[7]->setText(ecars[ind].getTransmission()->isAuto() ?

QString("АКПП") : QString("МКПП"));

text\_browser[8]->setText(QString::number(

ecars[ind].getTransmission()->getStagesAmount()));

text\_browser[9]->setText(QString::number(ecars[ind].getAmount()));

text\_browser[10]->setText(QString::number(services.size()));

text\_browser[11]->setText(QString::number(

ecars[ind].getPrice()) + QString(" у. е."));

label[7]->setText(QString("Ёмкость:"));

label[8]->setText(QString("КПД:"));

button[2]->setEnabled(true);

button[3]->setEnabled(services.size());

delete\_element->setEnabled(true);

}

else if(combo\_box[1]->currentData().toStringList().front().toInt() == 2) {

int ind = combo\_box[1]->currentData().toStringList().back().toInt();

text\_browser[0]->setText(QString::number(fcars[ind].getYear()));

text\_browser[1]->setText(fcars[ind].getBody()->getType().getQString());

text\_browser[2]->setText(QString::number(

fcars[ind].getBody()->getWeight()) + QString(" кг"));

text\_browser[3]->setText(QString("Вн. сгорания"));

text\_browser[4]->setText(QString::number(

fcars[ind].getEngine()->getPower()) + QString(" л. с."));

text\_browser[5]->setText(QString::number(

fcars[ind].getEngine()->getVolume(), 'f', 1) + QString(" л"));

text\_browser[6]->setText(QString::number(

fcars[ind].getEngine()->getCylindersAmount()));

text\_browser[7]->setText(fcars[ind].getTransmission()->isAuto() ?

QString("АКПП") : QString("МКПП"));

text\_browser[8]->setText(QString::number(

fcars[ind].getTransmission()->getStagesAmount()));

text\_browser[9]->setText(QString::number(fcars[ind].getAmount()));

text\_browser[10]->setText(QString::number(services.size()));

text\_browser[11]->setText(QString::number(

fcars[ind].getPrice()) + QString(" у. е."));

label[7]->setText(QString("Объём:"));

label[8]->setText(QString("Цилиндры:"));

button[2]->setEnabled(true);

button[3]->setEnabled(services.size());

delete\_element->setEnabled(true);

}

}

void MainWindow::findClient() {

Human temp(UString(line\_edit[0]->text().toStdString().c\_str()),

UString(line\_edit[1]->text().toStdString().c\_str()),

UString(line\_edit[2]->text().toStdString().c\_str()));

vector<Client>::iterator it = std::find(clients.begin(), clients.end(),

static\_cast<const Client&>(temp));

int count = std::count(clients.begin(), clients.end(),

static\_cast<const Client&>(temp));

if(count == 0) {

status\_bar->showMessage(QString("Клиент не найден!"), 5000);

current\_client = -1;

return;

}

else if(count > 1) {

status\_bar->showMessage(QString(

"Несколько вариантов, уточните дату рождения!"));

static Date temp\_date;

static DateInput \* d = new DateInput(this, temp\_date);

temp\_date = Date(0, 0, 0);

d->show();

this->setEnabled(false);

d->setEnabled(true);

d->exec();

this->setEnabled(true);

if(temp\_date == Date(0, 0, 0)) {

status\_bar->clearMessage();

return;

}

temp.setBirthday(temp\_date);

while(!(temp.getBirthday() == it->getBirthday())) {

it = std::find(it + 1, clients.end(), static\_cast<const Client&>(temp));

if(it == clients.end()) {

status\_bar->showMessage(QString("Дата рождения не найдена!"), 5000);

current\_client = -1;

return;

}

}

}

current\_client = it->getID();

QString day(QString::number(it->getBirthday().day));

if(it->getBirthday().day - 10 < 0) day = QString::number(0) + day;

QString month(QString::number(it->getBirthday().month));

if(it->getBirthday().month - 10 < 0) month = QString::number(0) + month;

text\_browser[12]->setText(day + '.' + month + '.'

+ QString::number(it->getBirthday().year));

text\_browser[13]->setText(it->getAddress().getQString());

text\_browser[14]->setText(it->getPassport().getQString());

downloadOrders();

text\_browser[15]->setText(QString::number(orders.size()));

status\_bar->clearMessage();

button[6]->setEnabled(true);

button[7]->setEnabled(orders.size());

delete\_element->setEnabled(true);

}

void MainWindow::findMarkServices() {

if(combo\_box[2]->currentText() !=

combo\_box[2]->itemText(combo\_box[2]->currentIndex())) {

status\_bar->showMessage(QString("Марка не найдена!"), 5000);

return;

}

if(services.size() != 0)

if(services[1].getMarkID() == current\_mark);

else downloadServices();

else downloadServices();

showServices();

}

void MainWindow::clearModelInfo() {

label[7]->setText(QString("Объём/Ёмкость:"));

label[8]->setText(QString("Цилиндры/КПД:"));

for(int i = 0; i < 12; ++i) text\_browser[i]->clear();

button[2]->setEnabled(false);

button[3]->setEnabled(false);

delete\_element->setEnabled(false);

}

void MainWindow::clearClientInfo() {

for(int i = 12; i < 16; ++i) text\_browser[i]->clear();

button[7]->setEnabled(false);

button[6]->setEnabled(false);

delete\_element->setEnabled(false);

current\_client = -1;

}

void MainWindow::callAddAutoWindow() {

static AddAutoWindow \* a = new AddAutoWindow(this);

static QMetaObject::Connection a\_c1 =

connect(a, SIGNAL(saveClickedElectroCar(Car<cpt::ElectroEngine>)),

SLOT(addEModelElement(Car<cpt::ElectroEngine>)));

static QMetaObject::Connection a\_c2 =

connect(a, SIGNAL(saveClickedFuelCar(Car<cpt::FuelEngine>)),

SLOT(addFModelElement(Car<cpt::FuelEngine>)));

static QMetaObject::Connection a\_c3 =

connect(a, SIGNAL(saveClickedMark(AutoMark)),

SLOT(addMarkElement(AutoMark)));

a->uploadMarks(combo\_box[0]);

a->show();

this->setEnabled(false);

a->setEnabled(true);

a->exec();

this->setEnabled(true);

}

void MainWindow::callAddCarAmount() {

static int add\_amount;

static AddCarAmountWindow \* ca = new AddCarAmountWindow(this, add\_amount=0);

ca->show();

this->setEnabled(false);

ca->setEnabled(true);

ca->exec();

if(add\_amount != 0) {

int ind = combo\_box[1]->currentData().toStringList().back().toInt();

if(combo\_box[1]->currentData().toStringList().front().toInt() == 1) {

ecars[ind].setAmount(ecars[ind].getAmount() + add\_amount);

uploadECars();

}

if(combo\_box[1]->currentData().toStringList().front().toInt() == 2) {

fcars[ind].setAmount(fcars[ind].getAmount() + add\_amount);

uploadFCars();

}

findModel();

}

this->setEnabled(true);

}

void MainWindow::callAddClientWindow() {

static AddClientWindow \* c = new AddClientWindow(this);

static QMetaObject::Connection c\_c = connect(c, SIGNAL(saveClicked(Client)), SLOT(addClientElement(Client)));

c->show();

this->setEnabled(false);

c->setEnabled(true);

c->exec();

this->setEnabled(true);

}

void MainWindow::callAddMarkWindow() {

static AddMarkWindow \* m = new AddMarkWindow(this);

static QMetaObject::Connection m\_c =

connect(m, SIGNAL(saveClicked(QString)), SLOT(addMarkElement(QString)));

if(combo\_box[2]->currentText() !=

combo\_box[2]->itemText(combo\_box[2]->currentIndex()) ||

combo\_box[2]->currentIndex() < 0)

m->setMarkName(combo\_box[2]->currentText());

m->show();

this->setEnabled(false);

m->setEnabled(true);

m->exec();

this->setEnabled(true);

}

void MainWindow::callAddOrder() {

static AddOrderWindow \* o = new AddOrderWindow(this);

static QMetaObject::Connection o\_c =

connect(o, SIGNAL(saveClicked(Order)), SLOT(addOrderElement(Order)));

o->uploadMarks(combo\_box[0]);

downloadAllCars();

o->uploadModels(ecars, fcars);

o->show();

this->setEnabled(false);

o->setEnabled(true);

o->exec();

downloadCars();

this->setEnabled(true);

}

void MainWindow::callAbout() {

static AboutWindow \* a = new AboutWindow(this);

a->show();

this->setEnabled(false);

a->setEnabled(true);

a->exec();

this->setEnabled(true);

}

void MainWindow::showServices() {

static ServicesListWindow \* s = new ServicesListWindow(this);

int ind = centralZone->currentIndex();

unsigned int prev\_size = services.size();

s->setWindowTitle(QString("Сервисы " + combo\_box[ind]->currentText()));

s->uploadServices(services, current\_mark);

s->show();

this->setEnabled(false);

s->setEnabled(true);

s->exec();

if(prev\_size != services.size()) uploadServices();

if(ind == 0) findModel();

this->setEnabled(true);

}

void MainWindow::showOrders() {

static OrdersListWindow \* o = new OrdersListWindow(this);

downloadAllCars();

o->uploadOrders(orders, ecars, fcars, marks);

o->show();

this->setEnabled(false);

o->setEnabled(true);

o->exec();

downloadCars();

this->setEnabled(true);

}

void MainWindow::closeEvent(QCloseEvent \*event) {

emit deleteActions();

event->accept();

}

void MainWindow::changeWindowSets(int i) {

switch(i) {

case 0:

setFixedSize(500, 440);

button[0]->setDefault(true);

current\_mark = combo\_box[0]->currentData().toInt();

downloadCars();

delete\_element->disconnect();

connect(delete\_element, SIGNAL(triggered(bool)), SLOT(deleteCarElement()));

connect(this, SIGNAL(deleteActions()), delete\_element,SLOT(deleteLater()));

delete\_element->setEnabled(!text\_browser[0]->toPlainText().isEmpty());

break;

case 1:

setFixedSize(365, 420);

button[4]->setDefault(true);

delete\_element->disconnect();

connect(delete\_element, SIGNAL(triggered(bool)),

SLOT(deleteClientElement()));

connect(this, SIGNAL(deleteActions()), delete\_element,SLOT(deleteLater()));

delete\_element->setEnabled(!text\_browser[12]->toPlainText().isEmpty());

break;

case 2:

setFixedSize(400, 120);

button[8]->setDefault(true);

current\_mark = combo\_box[2]->currentData().toInt();

delete\_element->disconnect();

connect(delete\_element, SIGNAL(triggered(bool)),SLOT(deleteMarkElement()));

connect(this, SIGNAL(deleteActions()), delete\_element,SLOT(deleteLater()));

delete\_element->setEnabled(

(combo\_box[2]->itemText(combo\_box[2]->currentIndex()) ==

combo\_box[2]->currentText()) && combo\_box[2]->currentIndex() > -1);

break;

}

}

void MainWindow::addMarkElement(QString new\_mark\_name) {

{

AutoMark new\_mark(UString(new\_mark\_name.toStdString().c\_str()));

if(marks.end() != std::find(marks.cbegin(), marks.cend(), new\_mark)) {

status\_bar->showMessage(QString("Такая марка уже есть!"), 5000);

return;

}

marks.insert(marks.end(), new\_mark);

}

sort(marks.begin(), marks.end());

uploadMarks();

marksFromVectorToComboBox();

combo\_box[0]->setCurrentIndex(-1);

combo\_box[2]->setCurrentIndex(-1);

}

void MainWindow::addMarkElement(AutoMark new\_mark) {

marks.insert(marks.end(), new\_mark);

sort(marks.begin(), marks.end());

uploadMarks();

marksFromVectorToComboBox();

combo\_box[0]->setCurrentIndex(-1);

combo\_box[2]->setCurrentIndex(-1);

}

void MainWindow::addEModelElement(Car<ElectroEngine> new\_model) {

int prev\_mark = current\_mark;

if(new\_model.getMarkID() != current\_mark) {

current\_mark = new\_model.getMarkID();

downloadCars();

}

ecars.insert(ecars.end(), new\_model);

sort(ecars.begin(), ecars.end());

uploadECars();

current\_mark = prev\_mark;

downloadCars();

modelsFromVectorToComboBox();

QString prev\_model = combo\_box[1]->currentText();

modelsFromVectorToComboBox();

if(!prev\_model.isEmpty()) {

combo\_box[1]->setCurrentIndex(combo\_box[1]->findText(prev\_model));

findModel();

}

}

void MainWindow::addFModelElement(Car<FuelEngine> new\_model) {

int prev\_mark = current\_mark;

if(new\_model.getMarkID() != current\_mark) {

current\_mark = new\_model.getMarkID();

downloadCars();

}

fcars.insert(fcars.end(), new\_model);

sort(fcars.begin(), fcars.end());

uploadFCars();

current\_mark = prev\_mark;

downloadCars();

QString prev\_model = combo\_box[1]->currentText();

modelsFromVectorToComboBox();

if(!prev\_model.isEmpty()) {

combo\_box[1]->setCurrentIndex(combo\_box[1]->findText(prev\_model));

findModel();

}

}

void MainWindow::addClientElement(Client new\_client) {

vector<Client>::iterator it = std::find(clients.begin(), clients.end(), new\_client);

if(it != clients.end()) {

if(it->getBirthday() == new\_client.getBirthday()) {

status\_bar->showMessage(QString("Такой клиент уже есть! Добавить невозможно!"), 5000);

return;

}

}

clients.insert(clients.end(), new\_client);

uploadClients();

}

void MainWindow::addOrderElement(Order new\_order) {

if(new\_order.getCarID() < 0) {

status\_bar->showMessage(QString("Данной модели нет на складе!"), 5000);

return;

}

new\_order.setClientID(current\_client);

orders.insert(orders.end(), new\_order);

sort(orders.begin(), orders.end());

uploadAllCars();

uploadOrders();

findClient();

}

void MainWindow::deleteCarElement() {

int ind = combo\_box[1]->currentData().toStringList().back().toInt();

if(combo\_box[1]->currentText() ==

combo\_box[1]->itemText(combo\_box[1]->currentIndex())) {

if(combo\_box[1]->currentData().toStringList().front().toInt() == 1) {

deleted\_ecars.push(\*(ecars.begin() + ind));

ecars.erase(ecars.begin() + ind);

undo\_stack.push(0);

undo\_action->setEnabled(true);

uploadECars();

}

else if(combo\_box[1]->currentData().toStringList().front().toInt() == 2) {

deleted\_fcars.push(\*(fcars.begin() + ind));

fcars.erase(fcars.begin() + ind);

undo\_stack.push(1);

undo\_action->setEnabled(true);

uploadFCars();

}

clearModelInfo();

modelsFromVectorToComboBox();

}

}

void MainWindow::deleteClientElement() {

vector<Client>::iterator it = clients.begin();

while(it->getID() != current\_client) ++it;

deleted\_clients.push(\*it);

clients.erase(it);

deleted\_orders.push(orders);

orders.clear();

undo\_stack.push(2);

undo\_action->setEnabled(true);

uploadOrders();

uploadClients();

clearClientInfo();

for(int i = 0; i < 3; ++i) line\_edit[i]->clear();

}

void MainWindow::deleteMarkElement() {

if(combo\_box[2]->currentText().isEmpty()) {

status\_bar->showMessage(QString("Введите название марки"), 5000);

return;

}

int ind = combo\_box[2]->currentIndex();

downloadCars();

if(ecars.size() + fcars.size() > 0) {

status\_bar->showMessage(QString("Для удаления марка не должна содержать моделей"), 5000);

return;

}

deleted\_marks.push(\*(marks.begin() + ind));

marks.erase(marks.begin() + ind);

deleted\_services.push(services);

services.clear();

undo\_stack.push(3);

undo\_action->setEnabled(true);

uploadMarks();

uploadServices();

marksFromVectorToComboBox();

current\_mark = -1;

combo\_box[0]->setCurrentText(QString());

centralZone->setCurrentIndex(0);

updateModelLine(QString());

centralZone->setCurrentIndex(2);

combo\_box[2]->setCurrentIndex(-1);

}

void MainWindow::undoLastAction() {

switch(undo\_stack.top()) {

case 0:

addEModelElement(deleted\_ecars.top());

combo\_box[0]->setCurrentIndex(combo\_box[0]->findData(

QVariant(deleted\_ecars.top().getMarkID())));

combo\_box[1]->setCurrentIndex(combo\_box[1]->findText(

deleted\_ecars.top().getModel().getQString()));

deleted\_ecars.pop();

centralZone->setCurrentIndex(0);

break;

case 1:

addFModelElement(deleted\_fcars.top());

combo\_box[0]->setCurrentIndex(combo\_box[0]->findData(

QVariant(deleted\_fcars.top().getMarkID())));

combo\_box[1]->setCurrentIndex(combo\_box[1]->findText(

deleted\_fcars.top().getModel().getQString()));

deleted\_fcars.pop();

centralZone->setCurrentIndex(0);

break;

case 2:

addClientElement(deleted\_clients.top());

line\_edit[0]->setText(deleted\_clients.top().getSurname().getQString());

line\_edit[1]->setText(deleted\_clients.top().getFirstname().getQString());

line\_edit[2]->setText(deleted\_clients.top().getSecondname().getQString());

current\_client = deleted\_clients.top().getID();

orders = deleted\_orders.top();

uploadOrders();

deleted\_clients.pop();

deleted\_orders.pop();

centralZone->setCurrentIndex(1);

break;

case 3:

addMarkElement(deleted\_marks.top());

combo\_box[2]->setCurrentIndex(combo\_box[2]->findData(

QVariant(deleted\_marks.top().getID())));

services = deleted\_services.top();

uploadServices();

deleted\_marks.pop();

deleted\_services.pop();

current\_mark = -1;

combo\_box[0]->setCurrentText(QString());

centralZone->setCurrentIndex(0);

updateModelLine(QString());

centralZone->setCurrentIndex(2);

break;

}

undo\_stack.pop();

if(undo\_stack.empty()) undo\_action->setEnabled(false);

}

//addautowindow.hpp

#pragma once

#include <QDialog>

#include <QGroupBox>

#include <QLabel>

#include <QLineEdit>

#include <QComboBox>

#include <QPushButton>

#include <QRadioButton>

#include <QSpinBox>

#include <QDoubleSpinBox>

#include "car.hpp"

#include "automark.hpp"

class AddAutoWindow : public QDialog

{

Q\_OBJECT

QGroupBox \* group\_box[8];

QLabel \* label[10];

QComboBox \* combo\_box;

QLineEdit \* line\_edit[2];

QRadioButton \* radio\_button[7];

QSpinBox \* spin\_box[7];

QDoubleSpinBox \* dspin\_box;

QPushButton \* button[3];

public:

AddAutoWindow(QWidget \*parent);

~AddAutoWindow();

void uploadMarks(QComboBox \*parent);

public slots:

void updateModelLine(QString mark);

void updateSaveButton();

void updateBodyTypeGroup();

void updateEngineGroup();

void saveAuto();

void clear();

bool close();

void reject();

signals:

void saveClickedMark(AutoMark mark\_name);

void saveClickedElectroCar(Car<cpt::ElectroEngine> new\_model);

void saveClickedFuelCar(Car<cpt::FuelEngine> new\_model);

};

//addautowindow.cpp

AddAutoWindow::AddAutoWindow(QWidget \*parent)

: QDialog(parent)

{

setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);

setWindowTitle(QString("Добавить авто"));

setWindowIcon(QIcon(":/resources/addcar\_icon.png"));

setFixedSize(300, 680);

QVBoxLayout \* result\_layout = new QVBoxLayout;

{

group\_box[0] = new QGroupBox(QString("Наименование авто"));

{

QVBoxLayout \* auto\_name = new QVBoxLayout;

{

QHBoxLayout \* mark\_input = new QHBoxLayout;

{

label[0] = new QLabel(QString("Марка"));

{

label[0]->setFixedHeight(20);

}

combo\_box = new QComboBox;

{

combo\_box->setFixedSize(200, 20);

combo\_box->setEditable(true);

combo\_box->setMaxVisibleItems(10);

connect(combo\_box, SIGNAL(currentTextChanged(QString)),

SLOT(updateModelLine(QString)));

}

mark\_input->addWidget(label[0], 70, Qt::AlignRight);

mark\_input->addWidget(combo\_box);

}

QHBoxLayout \* model\_input = new QHBoxLayout;

{

label[1] = new QLabel(QString("Модель"));

{

label[1]->setFixedHeight(20);

}

line\_edit[0] = new QLineEdit;

{

line\_edit[0]->setFixedSize(200, 20);

line\_edit[0]->setEnabled(false);

connect(line\_edit[0], SIGNAL(textChanged(QString)),

SLOT(updateSaveButton()));

}

model\_input->addWidget(label[1], 70, Qt::AlignRight);

model\_input->addWidget(line\_edit[0]);

}

auto\_name->addLayout(mark\_input);

auto\_name->addLayout(model\_input);

}

group\_box[0]->setLayout(auto\_name);

}

group\_box[1] = new QGroupBox(QString("Общие параметры"));

{

QVBoxLayout \* params = new QVBoxLayout;

{

group\_box[2] = new QGroupBox(QString("Тип кузова"));

{

QVBoxLayout \* type\_choosing = new QVBoxLayout;

{

QHBoxLayout \* common\_types = new QHBoxLayout;

{

radio\_button[0] = new QRadioButton(QString("Легковой"));

{

connect(radio\_button[0], SIGNAL(clicked(bool)),

SLOT(updateBodyTypeGroup()));

connect(radio\_button[0], SIGNAL(clicked(bool)),

SLOT(updateSaveButton()));

}

radio\_button[1] = new QRadioButton(QString("Грузовой"));

{

connect(radio\_button[1], SIGNAL(clicked(bool)),

SLOT(updateBodyTypeGroup()));

connect(radio\_button[1], SIGNAL(clicked(bool)),

SLOT(updateSaveButton()));

}

common\_types->addWidget(radio\_button[0]);

common\_types->addWidget(radio\_button[1]);

}

QHBoxLayout \* special\_type = new QHBoxLayout;

{

radio\_button[2] = new QRadioButton(QString("Особый"));

{

connect(radio\_button[2], SIGNAL(clicked(bool)),

SLOT(updateBodyTypeGroup()));

connect(radio\_button[2], SIGNAL(clicked(bool)),

SLOT(updateSaveButton()));

}

line\_edit[1] = new QLineEdit;

{

line\_edit[1]->setEnabled(false);

connect(line\_edit[1], SIGNAL(textChanged(QString)),

SLOT(updateSaveButton()));

}

special\_type->addWidget(radio\_button[2]);

special\_type->addWidget(line\_edit[1]);

}

type\_choosing->addLayout(common\_types);

type\_choosing->addLayout(special\_type);

}

group\_box[2]->setLayout(type\_choosing);

}

QHBoxLayout \* other\_params = new QHBoxLayout;

{

label[2] = new QLabel(QString("Год выпуска"));

spin\_box[0] = new QSpinBox;

{

time\_t TIME = time(nullptr);

int max\_year = 1900 + localtime(&TIME)->tm\_year;

int min\_year = max\_year - 10;

spin\_box[0]->setMinimum(min\_year);

spin\_box[0]->setMaximum(max\_year);

connect(spin\_box[0], SIGNAL(valueChanged(int)),

SLOT(updateSaveButton()));

connect(spin\_box[0], SIGNAL(editingFinished()),

SLOT(updateSaveButton()));

}

label[3] = new QLabel(QString("Масса (кг)"));

spin\_box[1] = new QSpinBox;

{

spin\_box[1]->setMinimum(400);

spin\_box[1]->setMaximum(5000);

spin\_box[1]->setSingleStep(100);

connect(spin\_box[1], SIGNAL(valueChanged(int)),

SLOT(updateSaveButton()));

connect(spin\_box[1], SIGNAL(editingFinished()),

SLOT(updateSaveButton()));

}

other\_params->addWidget(label[2]);

other\_params->addWidget(spin\_box[0]);

other\_params->addWidget(label[3]);

other\_params->setAlignment(label[3], Qt::AlignRight);

other\_params->addWidget(spin\_box[1]);

}

params->addWidget(group\_box[2]);

params->addLayout(other\_params);

}

group\_box[1]->setLayout(params);

}

group\_box[3] = new QGroupBox(QString("Трансмиссия"));

{

QVBoxLayout \* transmission\_layout = new QVBoxLayout;

{

group\_box[4] = new QGroupBox(QString("Тип трансмиссии"));

{

QHBoxLayout \* type\_choosing = new QHBoxLayout;

{

radio\_button[3] = new QRadioButton(QString("Механическая"));

{

connect(radio\_button[3], SIGNAL(clicked(bool)),

SLOT(updateSaveButton()));

}

radio\_button[4] = new QRadioButton(QString("Автоматическая"));

{

connect(radio\_button[4], SIGNAL(clicked(bool)),

SLOT(updateSaveButton()));

}

type\_choosing->addWidget(radio\_button[3]);

type\_choosing->addWidget(radio\_button[4]);

}

group\_box[4]->setLayout(type\_choosing);

}

QHBoxLayout \* steps\_count = new QHBoxLayout;

{

label[4] = new QLabel(QString("Кол-во ступеней"));

spin\_box[2] = new QSpinBox;

{

spin\_box[2]->setMinimum(4);

spin\_box[2]->setMaximum(8);

connect(spin\_box[2], SIGNAL(valueChanged(int)),

SLOT(updateSaveButton()));

connect(spin\_box[2], SIGNAL(editingFinished()),

SLOT(updateSaveButton()));

}

steps\_count->addWidget(label[4]);

steps\_count->addWidget(spin\_box[2]);

}

transmission\_layout->addWidget(group\_box[4]);

transmission\_layout->addLayout(steps\_count);

}

group\_box[3]->setLayout(transmission\_layout);

}

group\_box[5] = new QGroupBox(QString("Двигатель"));

{

QVBoxLayout \* engine\_layout = new QVBoxLayout;

{

group\_box[6] = new QGroupBox(QString("Тип двигателя"));

{

QHBoxLayout \* engine\_type = new QHBoxLayout;

{

radio\_button[5] = new QRadioButton("Вн. сгорания");

{

connect(radio\_button[5], SIGNAL(clicked(bool)),

SLOT(updateEngineGroup()));

connect(radio\_button[5], SIGNAL(clicked(bool)),

SLOT(updateSaveButton()));

}

radio\_button[6] = new QRadioButton("Электрический");

{

connect(radio\_button[6], SIGNAL(clicked(bool)),

SLOT(updateEngineGroup()));

connect(radio\_button[6], SIGNAL(clicked(bool)),

SLOT(updateSaveButton()));

}

engine\_type->addWidget(radio\_button[5]);

engine\_type->addWidget(radio\_button[6]);

}

group\_box[6]->setLayout(engine\_type);

}

QHBoxLayout \* engine\_power = new QHBoxLayout;

{

label[5] = new QLabel(QString("Мощность (л. с.)"));

spin\_box[3] = new QSpinBox;

{

spin\_box[3]->setMinimum(50);

spin\_box[3]->setMaximum(600);

spin\_box[3]->setSingleStep(50);

connect(spin\_box[3], SIGNAL(valueChanged(int)),

SLOT(updateSaveButton()));

connect(spin\_box[3], SIGNAL(editingFinished()),

SLOT(updateSaveButton()));

}

engine\_power->addWidget(label[5]);

engine\_power->addWidget(spin\_box[3]);

}

QHBoxLayout \* capacity\_layout = new QHBoxLayout;

{

label[6] = new QLabel(QString("Объём/Ёмкость"));

dspin\_box = new QDoubleSpinBox;

{

dspin\_box->setEnabled(false);

dspin\_box->setDecimals(1);

connect(dspin\_box, SIGNAL(valueChanged(double)),

SLOT(updateSaveButton()));

connect(dspin\_box, SIGNAL(editingFinished()),

SLOT(updateSaveButton()));

}

capacity\_layout->addWidget(label[6]);

capacity\_layout->addWidget(dspin\_box);

}

QHBoxLayout \* powers\_count = new QHBoxLayout;

{

label[7] = new QLabel(QString("Цилиндры/КПД"));

spin\_box[4] = new QSpinBox;

{

spin\_box[4]->setEnabled(false);

connect(spin\_box[4], SIGNAL(valueChanged(int)),

SLOT(updateSaveButton()));

connect(spin\_box[4], SIGNAL(editingFinished()),

SLOT(updateSaveButton()));

}

powers\_count->addWidget(label[7]);

powers\_count->addWidget(spin\_box[4]);

}

engine\_layout->addWidget(group\_box[6]);

engine\_layout->addLayout(engine\_power);

engine\_layout->addLayout(capacity\_layout);

engine\_layout->addLayout(powers\_count);

}

group\_box[5]->setLayout(engine\_layout);

}

group\_box[7] = new QGroupBox(QString("Сервисные данные"));

{

QVBoxLayout \* service\_info = new QVBoxLayout;

{

QHBoxLayout \* units\_amount = new QHBoxLayout;

{

label[8] = new QLabel(QString("Кол-во на складе"));

spin\_box[5] = new QSpinBox;

{

spin\_box[5]->setMinimum(0);

spin\_box[5]->setMaximum(1000);

connect(spin\_box[5], SIGNAL(valueChanged(int)),

SLOT(updateSaveButton()));

connect(spin\_box[5], SIGNAL(editingFinished()),

SLOT(updateSaveButton()));

}

units\_amount->addWidget(label[8]);

units\_amount->addWidget(spin\_box[5]);

}

QHBoxLayout \* price = new QHBoxLayout;

{

label[9] = new QLabel(QString("Цена (у. е.)"));

spin\_box[6] = new QSpinBox;

{

spin\_box[6]->setMinimum(5000);

spin\_box[6]->setMaximum(2000000);

spin\_box[6]->setSingleStep(100);

connect(spin\_box[6], SIGNAL(valueChanged(int)),

SLOT(updateSaveButton()));

connect(spin\_box[6], SIGNAL(editingFinished()),

SLOT(updateSaveButton()));

}

price->addWidget(label[9]);

price->addWidget(spin\_box[6]);

}

service\_info->addLayout(units\_amount);

service\_info->addLayout(price);

}

group\_box[7]->setLayout(service\_info);

}

QHBoxLayout \* buttons\_layout = new QHBoxLayout;

{

button[0] = new QPushButton(QString("Очистить"));

{

connect(button[0], SIGNAL(clicked(bool)), SLOT(clear()));

}

button[1] = new QPushButton(QString("Сохранить"));

{

button[1]->setEnabled(false);

connect(button[1], SIGNAL(clicked(bool)), SLOT(saveAuto()));

}

button[2] = new QPushButton(QString("Отмена"));

{

button[2]->setDefault(true);

connect(button[2], SIGNAL(clicked(bool)), SLOT(close()));

}

buttons\_layout->addWidget(button[0]);

buttons\_layout->addWidget(button[1]);

buttons\_layout->addWidget(button[2]);

}

result\_layout->addWidget(group\_box[0]);

result\_layout->addWidget(group\_box[1]);

result\_layout->addWidget(group\_box[3]);

result\_layout->addWidget(group\_box[5]);

result\_layout->addWidget(group\_box[7]);

result\_layout->addLayout(buttons\_layout);

}

setLayout(result\_layout);

for(int i = 0; i < 7; i++) spin\_box[i]->clear();

dspin\_box->clear();

}

AddAutoWindow::~AddAutoWindow()

{

}

void AddAutoWindow::uploadMarks(QComboBox \*parent) {

combo\_box->clear();

int amount = parent->count();

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

combo\_box->addItem(parent->itemText(i), parent->itemData(i));

combo\_box->setCurrentIndex(parent->currentIndex());

combo\_box->setCurrentText(parent->currentText());

}

void AddAutoWindow::updateModelLine(QString mark) {

if(mark.isEmpty()) line\_edit[0]->clear();

line\_edit[0]->setEnabled(!mark.isEmpty());

}

void AddAutoWindow::updateSaveButton() {

bool spins\_check = true;

for(int i = 0; i < 7 &&

(spins\_check = !spin\_box[i]->cleanText().isEmpty()); i++);

spins\_check = spins\_check && !dspin\_box->cleanText().isEmpty();

spins\_check = (!line\_edit[0]->displayText().isEmpty() && spins\_check &&

(radio\_button[0]->isChecked() || radio\_button[1]->isChecked()

|| !line\_edit[1]->displayText().isEmpty()) &&

(radio\_button[3]->isChecked() || radio\_button[4]->isChecked())

&& (radio\_button[5]->isChecked() || radio\_button[6]->isChecked()));

button[1]->setEnabled(spins\_check);

button[1]->setDefault(spins\_check);

button[2]->setDefault(!spins\_check);

}

void AddAutoWindow::updateBodyTypeGroup() {

if(!radio\_button[2]->isChecked()) line\_edit[1]->clear();

line\_edit[1]->setEnabled(radio\_button[2]->isChecked());

}

void AddAutoWindow::updateEngineGroup() {

if(radio\_button[5]->isChecked()) {

label[6]->setText(QString("Объём (л)"));

dspin\_box->setEnabled(true);

dspin\_box->setMinimum(0.5);

dspin\_box->setMaximum(7.5);

dspin\_box->setSingleStep(0.5);

label[7]->setText(QString("Кол-во цилиндров"));

spin\_box[4]->setEnabled(true);

spin\_box[4]->setMinimum(1);

spin\_box[4]->setMaximum(12);

spin\_box[4]->setSingleStep(1);

}

else if(radio\_button[6]->isChecked()) {

label[6]->setText(QString("Ёмкость (кА)"));

dspin\_box->setEnabled(true);

dspin\_box->setMinimum(0.5);

dspin\_box->setMaximum(20.0);

dspin\_box->setSingleStep(0.5);

label[7]->setText(QString("КПД (%)"));

spin\_box[4]->setEnabled(true);

spin\_box[4]->setMinimum(40);

spin\_box[4]->setMaximum(98);

spin\_box[4]->setSingleStep(5);

}

else {

label[6]->setText(QString("Объём/Ёмкость"));

label[7]->setText(QString("Цилиндры/КПД"));

dspin\_box->setEnabled(false);

spin\_box[4]->setEnabled(false);

}

dspin\_box->setValue(0.0);

dspin\_box->clear();

spin\_box[4]->setValue(0);

spin\_box[4]->clear();

}

void AddAutoWindow::saveAuto() {

int mark\_ID = combo\_box->currentData().toInt();

UString body;

if(combo\_box->currentText() !=

combo\_box->itemText(combo\_box->currentIndex()) ||

combo\_box->currentIndex() < 0) {

QString name = combo\_box->currentText();

if(name[name.size()-1] == ' ') name.remove(name.size()-1, 1);

AutoMark temp(UString(name.toStdString().c\_str()));

mark\_ID = temp.getID();

emit saveClickedMark(temp);

}

if(radio\_button[0]->isChecked())

body = radio\_button[0]->text().toStdString().c\_str();

else if(radio\_button[1]->isChecked())

body = radio\_button[1]->text().toStdString().c\_str();

else

body = line\_edit[1]->text().toStdString().c\_str();

if(radio\_button[6]->isChecked()) {

emit saveClickedElectroCar(Car<cpt::ElectroEngine>(mark\_ID,

UString(line\_edit[0]->text().toStdString().c\_str()),

cpt::CarBody(body, spin\_box[1]->value()),

cpt::ElectroEngine(spin\_box[3]->value(),

static\_cast<int>(dspin\_box->value()\*1000),

spin\_box[4]->value()),

cpt::CarTransmission(radio\_button[4]->isChecked(),

spin\_box[2]->value()),

spin\_box[0]->value(), spin\_box[6]->value(),

spin\_box[5]->value()));

}

else {

emit saveClickedFuelCar(Car<cpt::FuelEngine>(mark\_ID,

UString(line\_edit[0]->text().toStdString().c\_str()),

cpt::CarBody(body, spin\_box[1]->value()),

cpt::FuelEngine(spin\_box[3]->value(),

spin\_box[4]->value(),

dspin\_box->value()),

cpt::CarTransmission(radio\_button[4]->isChecked(),

spin\_box[2]->value()),

spin\_box[0]->value(),

spin\_box[6]->value(),

spin\_box[5]->value()));

}

close();

}

void AddAutoWindow::clear() {

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

radio\_button[i]->setAutoExclusive(false);

radio\_button[i]->setChecked(false);

radio\_button[i]->setAutoExclusive(true);

spin\_box[i]->setValue(0);

spin\_box[i]->clear();

}

for(int i = 0; i < 2; i++) line\_edit[i]->clear();

line\_edit[1]->setEnabled(false);

combo\_box->setCurrentIndex(-1);

updateEngineGroup();

}

bool AddAutoWindow::close() {

clear();

return QWidget::close();

}

void AddAutoWindow::reject() {

clear();

QDialog::reject();

}

//addcaramountwindow.hpp

#pragma once

#include <QDialog>

#include <QSpinBox>

#include <QPushButton>

class AddCarAmountWindow : public QDialog

{

Q\_OBJECT

QSpinBox \* spin\_box;

QPushButton \* button[2];

int \* amount;

public:

AddCarAmountWindow(QWidget \*parent, int& amount);

~AddCarAmountWindow();

public slots:

void updateSaveButton();

void saveAmount();

void clear();

bool close();

void reject();

};

//addcaramountwindow.cpp

#include "addcaramountwindow.hpp"

#include <QHBoxLayout>

#include <QVBoxLayout>

AddCarAmountWindow::AddCarAmountWindow(QWidget \*parent, int &amount)

: QDialog(parent), amount(&amount)

{

setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);

setWindowTitle(QString("Обновить количество"));

setWindowIcon(QIcon(":/resources/addcar\_icon.png"));

setFixedSize(200, 80);

QVBoxLayout \* central\_layout = new QVBoxLayout;

{

spin\_box = new QSpinBox;

{

spin\_box->setMinimum(0);

spin\_box->setSingleStep(1);

spin\_box->setFixedWidth(50);

spin\_box->clear();

connect(spin\_box, SIGNAL(valueChanged(int)),

SLOT(updateSaveButton()));

}

QHBoxLayout \* buttons\_layout = new QHBoxLayout;

{

button[0] = new QPushButton(QString("Сохранить"));

{

button[0]->setEnabled(false);

connect(button[0], SIGNAL(clicked(bool)),

SLOT(saveAmount()));

}

button[1] = new QPushButton(QString("Отмена"));

{

button[1]->setDefault(true);

connect(button[1], SIGNAL(clicked(bool)), SLOT(close()));

}

buttons\_layout->addWidget(button[0]);

buttons\_layout->addWidget(button[1]);

}

central\_layout->addWidget(spin\_box, 50, Qt::AlignCenter);

central\_layout->addLayout(buttons\_layout);

}

setLayout(central\_layout);

}

AddCarAmountWindow::~AddCarAmountWindow()

{

}

void AddCarAmountWindow::updateSaveButton() {

bool check = spin\_box->value();

button[0]->setEnabled(check);

button[0]->setDefault(check);

button[1]->setDefault(!check);

}

void AddCarAmountWindow::saveAmount() {

\*amount = spin\_box->value();

close();

}

void AddCarAmountWindow::clear() {

spin\_box->clear();

button[0]->setEnabled(false);

}

bool AddCarAmountWindow::close() {

clear();

return QWidget::close();

}

void AddCarAmountWindow::reject() {

clear();

QDialog::reject();

}

//addclientwindow.hpp

#pragma once

#include <QDialog>

#include <QLabel>

#include <QLineEdit>

#include <QDateEdit>

#include <QPushButton>

#include "client.hpp"

class AddClientWindow : public QDialog

{

Q\_OBJECT

QLabel \* label[6];

QLineEdit \* line\_edit[5];

QDateEdit \* date\_edit;

QPushButton \* button[3];

public:

AddClientWindow(QWidget \*parent);

~AddClientWindow();

public slots:

void updateSaveButton();

void saveClient();

void clear();

bool close();

void reject();

signals:

void saveClicked(Client new\_client);

};

//addclientwindow.cpp

#include "addclientwindow.hpp"

#include <QIcon>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <time.h>

AddClientWindow::AddClientWindow(QWidget \*parent)

: QDialog(parent)

{

setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);

setWindowTitle(QString("Добавить клиента"));

setWindowIcon(QIcon(":/resources/adduser\_icon.png"));

setFixedSize(340, 230);

QVBoxLayout \* central\_layout = new QVBoxLayout;

{

QHBoxLayout \* surname\_layout = new QHBoxLayout;

{

label[0] = new QLabel(QString("Фамилия"));

line\_edit[0] = new QLineEdit;

{

line\_edit[0]->setFixedSize(200, 23);

line\_edit[0]->setValidator(

new QRegExpValidator(QRegExp((QString(

"^(\\b([A-Z]{1}[a-z]{0,})\\b)((\\b([A-Z]?[a-z]{0,})\\b|

\\b(-{1})\\b){2})$|^(\\b([А-Я]{1}[а-я]{0,})\\b)

((\\b([А-Я]?[а-я]{0,})\\b|\\b(-{1})\\b){2})$"))), this));

connect(line\_edit[0], SIGNAL(textChanged(QString)),

SLOT(updateSaveButton()));

}

surname\_layout->addWidget(label[0]);

surname\_layout->addWidget(line\_edit[0]);

}

QHBoxLayout \* firstname\_layout = new QHBoxLayout;

{

label[1] = new QLabel(QString("Имя"));

line\_edit[1] = new QLineEdit;

{

line\_edit[1]->setFixedSize(200, 23);

line\_edit[1]->setValidator(

new QRegExpValidator(QRegExp((QString(

"^(\\b([A-Z]{1}[a-z]{0,})\\b)$|^(\\b([А-Я|Ё]{1}

[а-я|ё]{0,})\\b)$"))), this));

connect(line\_edit[1], SIGNAL(textChanged(QString)),

SLOT(updateSaveButton()));

}

firstname\_layout->addWidget(label[1]);

firstname\_layout->addWidget(line\_edit[1]);

}

QHBoxLayout \* secondname\_layout = new QHBoxLayout;

{

label[2] = new QLabel(QString("Отчество"));

line\_edit[2] = new QLineEdit;

{

line\_edit[2]->setFixedSize(200, 23);

line\_edit[2]->setValidator(line\_edit[1]->validator());

connect(line\_edit[2], SIGNAL(textChanged(QString)),

SLOT(updateSaveButton()));

}

secondname\_layout->addWidget(label[2]);

secondname\_layout->addWidget(line\_edit[2]);

}

QHBoxLayout \* birthday\_layout = new QHBoxLayout;

{

label[3] = new QLabel(QString("Дата рождения"));

date\_edit = new QDateEdit;

{

time\_t TIME = time(nullptr);

int max\_day = localtime(&TIME)->tm\_mday - 1;

int max\_month = localtime(&TIME)->tm\_mon + 1;

int max\_year = 1900 + localtime(&TIME)->tm\_year - 18;

date\_edit->setMaximumDate(QDate(max\_year, max\_month,

max\_day));

date\_edit->setMinimumDate(QDate(max\_year - 100, max\_month,

max\_day));

date\_edit->setFixedSize(200, 23);

date\_edit->setCalendarPopup(true);

}

birthday\_layout->addWidget(label[3]);

birthday\_layout->addWidget(date\_edit);

}

QHBoxLayout \* address\_layout = new QHBoxLayout;

{

label[4] = new QLabel(QString("Адрес"));

line\_edit[3] = new QLineEdit;

{

line\_edit[3]->setFixedSize(200, 23);

}

address\_layout->addWidget(label[4]);

address\_layout->addWidget(line\_edit[3]);

}

QHBoxLayout \* passport\_layout = new QHBoxLayout;

{

label[5] = new QLabel(QString("Паспортные данные"));

line\_edit[4] = new QLineEdit;

{

line\_edit[4]->setFixedSize(200, 23);

line\_edit[4]->setInputMask(QString(">AA 9999999;\_"));

}

passport\_layout->addWidget(label[5]);

passport\_layout->addWidget(line\_edit[4]);

}

QHBoxLayout \* buttons\_layout = new QHBoxLayout;

{

button[0] = new QPushButton(QString("Очистить"));

button[1] = new QPushButton(QString("Сохранить"));

button[2] = new QPushButton(QString("Отмена"));

buttons\_layout->addWidget(button[0]);

{

connect(button[0], SIGNAL(clicked(bool)), SLOT(clear()));

}

buttons\_layout->addWidget(button[1]);

{

button[1]->setEnabled(false);

connect(button[1], SIGNAL(clicked(bool)),

SLOT(saveClient()));

}

buttons\_layout->addWidget(button[2]);

{

button[2]->setDefault(true);

connect(button[2], SIGNAL(clicked(bool)), SLOT(close()));

}

}

central\_layout->addLayout(surname\_layout);

central\_layout->addLayout(firstname\_layout);

central\_layout->addLayout(secondname\_layout);

central\_layout->addLayout(birthday\_layout);

central\_layout->addLayout(address\_layout);

central\_layout->addLayout(passport\_layout);

central\_layout->addLayout(buttons\_layout);

}

setLayout(central\_layout);

}

AddClientWindow::~AddClientWindow()

{

}

void AddClientWindow::updateSaveButton() {

bool check = !(line\_edit[0]->displayText().isEmpty() ||

line\_edit[1]->displayText().isEmpty() ||

line\_edit[2]->displayText().isEmpty());

button[1]->setEnabled(check);

button[1]->setDefault(check);

button[2]->setDefault(!check);

}

void AddClientWindow::saveClient() {

emit saveClicked(Client(

UString(line\_edit[0]->text().toStdString().c\_str()),

UString(line\_edit[1]->text().toStdString().c\_str()),

UString(line\_edit[2]->text().toStdString().c\_str()),

Date(date\_edit->date().day(),

date\_edit->date().month(),

date\_edit->date().year()),

UString(line\_edit[4]->text().toStdString().c\_str()),

UString(line\_edit[3]->text().toStdString().c\_str())));

close();

}

void AddClientWindow::clear() {

for(int i = 0; i < 5; i++) line\_edit[i]->clear();

date\_edit->setDate(date\_edit->maximumDate());

}

bool AddClientWindow::close() {

clear();

return QWidget::close();

}

void AddClientWindow::reject() {

clear();

QDialog::reject();

}

//dateinput.hpp

#pragma once

#include "addmarkwindow.hpp"

#include "old\_projects/date.hpp"

#include <QDateEdit>

class DateInput : public AddMarkWindow

{

Q\_OBJECT

QDateEdit \* date\_edit;

Date \* date;

public:

DateInput(QWidget \*parent, Date& date);

~DateInput();

public slots:

void saveDate();

void clear();

};

//dateinput.cpp

#include "dateinput.hpp"

#include <QIcon>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QRegExpValidator>

#include <time.h>

DateInput::DateInput(QWidget \*parent, Date &date)

: AddMarkWindow(parent), date(&date)

{

setWindowTitle(QString("Поиск по дате"));

setWindowIcon(QIcon(":/resources/user\_icon.png"));

label->setText("Дата рождения");

button[1]->setEnabled(true);

button[1]->setDefault(true);

disconnect(button[1], SIGNAL(clicked(bool)), this, SLOT(saveMark()));

connect(button[1], SIGNAL(clicked(bool)), SLOT(saveDate()));

date\_edit = new QDateEdit;

{

time\_t TIME = time(nullptr);

int max\_day = localtime(&TIME)->tm\_mday - 1;

int max\_month = localtime(&TIME)->tm\_mon + 1;

int max\_year = 1900 + localtime(&TIME)->tm\_year - 18;

date\_edit->setMaximumDate(QDate(max\_year, max\_month, max\_day));

date\_edit->setMinimumDate(QDate(1900, max\_month, max\_day));

date\_edit->setFixedSize(200, 23);

date\_edit->setCalendarPopup(true);

}

layout()->replaceWidget(line\_edit, date\_edit);

layout()->removeWidget(button[0]);

disconnect(line\_edit, SIGNAL(textChanged(QString)), this,

SLOT(updateSaveButton()));

disconnect(button[0], SIGNAL(clicked(bool)), this, SLOT(clear()));

line\_edit->close();

button[0]->close();

delete line\_edit;

delete button[0];

}

DateInput::~DateInput()

{

}

void DateInput::clear() {

date\_edit->setDate(date\_edit->maximumDate());

}

void DateInput::saveDate() {

\*date = Date(date\_edit->date().day(), date\_edit->date().month(), date\_edit->date().year());

close();

}

//addorderwindow.hpp

#pragma once

#include <QDialog>

#include <QLabel>

#include <QComboBox>

#include <QPushButton>

#include <vector>

#include "car.hpp"

#include "order.hpp"

class AddOrderWindow : public QDialog

{

Q\_OBJECT

QLabel \* label[2];

QComboBox \* combo\_box[2];

QPushButton \* button[2];

std::vector<Car<cpt::ElectroEngine>> \* ecars;

std::vector<Car<cpt::FuelEngine>> \* fcars;

public:

AddOrderWindow(QWidget \*parent);

~AddOrderWindow();

void uploadMarks(QComboBox \*parent);

void uploadModels(std::vector<Car<cpt::ElectroEngine>> &ecars, std::vector<Car<cpt::FuelEngine>> &fcars);

public slots:

void updateModelLine();

void updateSaveButton();

void saveOrder();

void clear();

bool close();

void reject();

signals:

void saveClicked(Order new\_order);

};

//addorderwindow.cpp

#include "addorderwindow.hpp"

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QList>

#include <time.h>

AddOrderWindow::AddOrderWindow(QWidget \*parent)

: QDialog(parent), ecars(nullptr), fcars(nullptr)

{

setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);

setWindowTitle(QString("Добавить заказ"));

setWindowIcon(QIcon(":/resources/addcar\_icon.png"));

setFixedSize(280, 110);

QVBoxLayout \* central\_layout = new QVBoxLayout;

{

QHBoxLayout \* mark\_input = new QHBoxLayout;

{

label[0] = new QLabel(QString("Марка"));

{

label[0]->setFixedHeight(20);

}

combo\_box[0] = new QComboBox;

{

combo\_box[0]->setFixedSize(200, 20);

combo\_box[0]->setEditable(true);

combo\_box[0]->setMaxVisibleItems(10);

connect(combo\_box[0], SIGNAL(currentTextChanged(QString)),

SLOT(updateModelLine()));

}

mark\_input->addWidget(label[0], 70, Qt::AlignRight);

mark\_input->addWidget(combo\_box[0]);

}

QHBoxLayout \* model\_input = new QHBoxLayout;

{

label[1] = new QLabel(QString("Модель"));

{

label[1]->setFixedHeight(20);

}

combo\_box[1] = new QComboBox;

{

combo\_box[1]->setFixedSize(200, 20);

combo\_box[1]->setEditable(true);

combo\_box[1]->setMaxVisibleItems(10);

combo\_box[1]->setEnabled(false);

connect(combo\_box[1], SIGNAL(currentTextChanged(QString)),

SLOT(updateSaveButton()));

}

model\_input->addWidget(label[1], 70, Qt::AlignRight);

model\_input->addWidget(combo\_box[1]);

}

QHBoxLayout \* buttons\_layout = new QHBoxLayout;

{

button[0] = new QPushButton(QString("Сохранить"));

{

button[0]->setEnabled(false);

connect(button[0], SIGNAL(clicked(bool)),

SLOT(saveOrder()));

}

button[1] = new QPushButton(QString("Отмена"));

{

button[1]->setDefault(true);

connect(button[1], SIGNAL(clicked(bool)), SLOT(close()));

}

buttons\_layout->addWidget(button[0]);

buttons\_layout->addWidget(button[1]);

}

central\_layout->addLayout(mark\_input);

central\_layout->addLayout(model\_input);

central\_layout->addLayout(buttons\_layout);

}

setLayout(central\_layout);

}

AddOrderWindow::~AddOrderWindow()

{

}

void AddOrderWindow::uploadMarks(QComboBox \*parent) {

combo\_box[0]->clear();

int amount = parent->count();

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

combo\_box[0]->addItem(parent->itemText(i), parent->itemData(i));

combo\_box[0]->setCurrentIndex(-1);

}

void AddOrderWindow::uploadModels(std::vector<Car<ElectroEngine>> &ecars,

std::vector<Car<FuelEngine>> &fcars) {

this->ecars = &ecars;

this->fcars = &fcars;

}

void AddOrderWindow::updateModelLine() {

if(combo\_box[0]->currentIndex() > -1 && combo\_box[0]->currentText() ==

combo\_box[0]->itemText(combo\_box[0]->currentIndex()) &&

ecars != nullptr && fcars != nullptr) {

combo\_box[1]->clear();

unsigned int j = 0, k = 0;

while(j < ecars->size() && k < fcars->size()) {

while(j < ecars->size() && ecars->at(j).getMarkID() != combo\_box[0]->currentData().toInt()) ++j;

while(k < fcars->size() && fcars->at(k).getMarkID() != combo\_box[0]->currentData().toInt()) ++k;

if(j >= ecars->size() || k >= fcars->size()) break;

if(ecars->at(j).getModel() < fcars->at(k).getModel()) {

combo\_box[1]->addItem(ecars->at(j).getModel().getQString(),

QVariant(QList<QVariant>(

{ecars->at(j).getID(), j, 1})));

++j;

}

else {

combo\_box[1]->addItem(fcars->at(k).getModel().getQString(), QVariant(QList<QVariant>(

{fcars->at(k).getID(), k, 2})));

++k;

}

}

while(j < ecars->size()) {

if(ecars->at(j).getMarkID() ==

combo\_box[0]->currentData().toInt())

combo\_box[1]->addItem(ecars->at(j).getModel().getQString(),

QVariant(QList<QVariant>(

{ecars->at(j).getID(), j, 1})));

++j;

}

while(k < fcars->size()) {

if(fcars->at(k).getMarkID() ==

combo\_box[0]->currentData().toInt())

combo\_box[1]->addItem(fcars->at(k).getModel().getQString(),

QVariant(QList<QVariant>(

{fcars->at(k).getID(), k, 2})));

++k;

}

combo\_box[1]->setCurrentIndex(-1);

combo\_box[1]->setEnabled(true);

}

else {

combo\_box[1]->clear();

combo\_box[1]->setEnabled(false);

}

}

void AddOrderWindow::updateSaveButton() {

button[0]->setEnabled(combo\_box[1]->currentText() ==

combo\_box[1]->itemText(combo\_box[1]->currentIndex()) &&

combo\_box[1]->currentIndex() > -1);

}

void AddOrderWindow::saveOrder() {

if(combo\_box[1]->currentData().toList().at(2).toInt() == 1) {

if(ecars->at(

combo\_box[1]->currentData().toList().at(1).toInt()).getAmount()

== 0) {

emit saveClicked(Order(-1, -1, Date(-1, -1, -1)));

close();

return;

}

ecars->at(

combo\_box[1]->currentData().toList().at(1).toInt()).setAmount(

ecars->at(

combo\_box[1]->currentData().toList().at(1).toInt()).getAmount()-1);

}

else if(combo\_box[1]->currentData().toList().at(2).toInt() == 2) {

if(

fcars->at(combo\_box[1]->currentData().toList().at(1).toInt()).getAmount()

== 0) {

emit saveClicked(Order(-1, -1, Date(-1, -1, -1)));

close();

return;

}

fcars->at(

combo\_box[1]->currentData().toList().at(1).toInt()).setAmount(

fcars->at(

combo\_box[1]->currentData().toList().at(1).toInt()).getAmount()-1);

}

time\_t TIME = time(nullptr);

int day = localtime(&TIME)->tm\_mday;

int month = localtime(&TIME)->tm\_mon + 1;

int year = 1900 + localtime(&TIME)->tm\_year;

emit saveClicked(Order(-1,

combo\_box[1]->currentData().toList().at(0).toInt(),

Date(day, month, year)));

close();

}

void AddOrderWindow::clear() {

combo\_box[0]->clear();

combo\_box[0]->setCurrentIndex(-1);

combo\_box[1]->clear();

combo\_box[1]->setCurrentIndex(-1);

}

bool AddOrderWindow::close() {

clear();

return QWidget::close();

}

void AddOrderWindow::reject() {

clear();

QDialog::reject();

}

//orderslistwindow.hpp

#pragma once

#include <QDialog>

#include <QTableWidget>

#include <QPushButton>

#include "order.hpp"

#include "car.hpp"

#include "automark.hpp"

class OrdersListWindow : public QDialog

{

Q\_OBJECT

QTableWidget \* table;

QPushButton \* button;

public:

OrdersListWindow(QWidget \*parent);

~OrdersListWindow();

void uploadOrders(const std::vector<Order> &orders,

const std::vector<Car<cpt::ElectroEngine>> &ecars,

const std::vector<Car<cpt::FuelEngine>> &fcars,

const std::vector<AutoMark> &marks);

public slots:

void clear();

bool close();

void reject();

};

//orderslistwindow.cpp

#include "orderslistwindow.hpp"

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QHeaderView>

OrdersListWindow::OrdersListWindow(QWidget \*parent)

: QDialog(parent)

{

setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);

setWindowTitle(QString("Заказы"));

setWindowIcon(QIcon(":/resources/icon.ico"));

setFixedSize(500, 250);

QVBoxLayout \* central\_layout = new QVBoxLayout;

{

table = new QTableWidget;

{

table->setRowCount(0);

table->setColumnCount(2);

table->setColumnWidth(0, 150);

table->setColumnWidth(1, 309);

table->setEditTriggers(QAbstractItemView::NoEditTriggers);

table->setSelectionMode(QAbstractItemView::SingleSelection);

table->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

table->setHorizontalHeaderLabels(

QStringList({QString("Автомобиль"),

QString("Дата покупки")}));

table->horizontalHeader()->setSectionResizeMode(

QHeaderView::Fixed);

table->horizontalHeader()->setSectionsClickable(false);

table->verticalHeader()->hide();

}

button = new QPushButton("Закрыть");

{

connect(button, SIGNAL(clicked(bool)), this, SLOT(close()));

}

central\_layout->addWidget(table);

central\_layout->addWidget(button);

}

setLayout(central\_layout);

}

OrdersListWindow::~OrdersListWindow()

{

}

void OrdersListWindow::uploadOrders(const std::vector<Order> &orders,

const std::vector<Car<cpt::ElectroEngine>> &ecars,

const std::vector<Car<cpt::FuelEngine>> &fcars,

const std::vector<AutoMark> &marks) {

unsigned int size = orders.size();

table->setRowCount(size);

for(unsigned int i = 0; i < size; i++) {

int model\_ID = orders[i].getCarID();

QString Car\_name;

QString Model\_name;

std::vector<Car<cpt::ElectroEngine>>::const\_iterator it1 = cars.cbegin();

std::vector<Car<cpt::FuelEngine>>::const\_iterator it2 = fcars.cbegin();

for(; it1 != ecars.cend() ; ++it1)

if(it1->getID() == model\_ID) {

for(std::vector<AutoMark>::const\_iterator it3 = marks.cbegin();

it3 != marks.cend(); ++it3)

if(it3->getID() == it1->getMarkID())

Car\_name = it3->getName().getQString();

Model\_name = it1->getModel().getQString();

break;

}

for(; it1 == ecars.cend() && it2 != fcars.cend(); ++it2)

if(it2->getID() == model\_ID) {

for(std::vector<AutoMark>::const\_iterator it3 = marks.cbegin();

it3 != marks.cend(); ++it3)

if(it3->getID() == it2->getMarkID()) {

Car\_name = it3->getName().getQString();

break;

}

Model\_name = it2->getModel().getQString();

break;

}

if(Model\_name.isEmpty()) Car\_name = QString("[УДАЛЕНО]");

else Car\_name += QString(" ") + Model\_name;

QString day(QString::number(orders[i].getPurchaseDate().day));

if(orders[i].getPurchaseDate().day - 10 < 0)

day = QString::number(0) + day;

QString month(QString::number(orders[i].getPurchaseDate().month));

if(orders[i].getPurchaseDate().month - 10 < 0)

month = QString::number(0) + month;

table->setItem(i, 0, new QTableWidgetItem(Car\_name));

table->setItem(i, 1, new QTableWidgetItem(day + QString('.') +

month + QString('.') +

QString::number(

orders[i].getPurchaseDate().year)));

}

}

void OrdersListWindow::clear() {

table->clearContents();

table->setRowCount(0);

}

bool OrdersListWindow::close() {

clear();

return QWidget::close();

}

void OrdersListWindow::reject() {

clear();

QDialog::reject();

}

//addmarkwindow.hpp

#pragma once

#include <QDialog>

#include <QLabel>

#include <QLineEdit>

#include <QPushButton>

class AddMarkWindow : public QDialog

{

Q\_OBJECT

protected:

QLabel \* label;

QLineEdit \* line\_edit;

QPushButton \* button[3];

public:

AddMarkWindow(QWidget \*parent);

~AddMarkWindow();

void setMarkName(QString mark\_name);

public slots:

void updateSaveButton();

void saveMark();

virtual void clear();

bool close();

void reject();

signals:

void saveClicked(QString mark\_name);

};

//addmarkwindow.cpp

#include "addmarkwindow.hpp"

#include <QIcon>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QRegExpValidator>

AddMarkWindow::AddMarkWindow(QWidget \*parent)

: QDialog(parent)

{

setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);

setWindowTitle(QString("Добавить марку"));

setWindowIcon(QIcon(":/resources/addcar\_icon.png"));

setFixedSize(300, 90);

QVBoxLayout \* central\_layout = new QVBoxLayout;

{

QHBoxLayout \* name\_layout = new QHBoxLayout;

{

label = new QLabel(QString("Название"));

line\_edit = new QLineEdit;

{

line\_edit->setFixedSize(200, 23);

line\_edit->setValidator(

new QRegExpValidator(QRegExp(QString(

"^\\S{1}(\\s{0,1}\\S{1,}){0,}$"))));

connect(line\_edit, SIGNAL(textChanged(QString)),

SLOT(updateSaveButton()));

}

name\_layout->addWidget(label);

name\_layout->addWidget(line\_edit);

}

QHBoxLayout \* buttons\_layout = new QHBoxLayout;

{

button[0] = new QPushButton(QString("Очистить"));

{

connect(button[0], SIGNAL(clicked(bool)), SLOT(clear()));

}

button[1] = new QPushButton(QString("Сохранить"));

{

button[1]->setEnabled(false);

connect(button[1], SIGNAL(clicked(bool)),

SLOT(saveMark()));

}

button[2] = new QPushButton(QString("Отмена"));

{

button[2]->setDefault(true);

connect(button[2], SIGNAL(clicked(bool)), SLOT(close()));

}

buttons\_layout->addWidget(button[0]);

buttons\_layout->addWidget(button[1]);

buttons\_layout->addWidget(button[2]);

}

central\_layout->addLayout(name\_layout);

central\_layout->addLayout(buttons\_layout);

}

setLayout(central\_layout);

}

AddMarkWindow::~AddMarkWindow()

{

}

void AddMarkWindow::setMarkName(QString mark\_name) {

line\_edit->setText(mark\_name);

}

void AddMarkWindow::updateSaveButton() {

bool check = !(line\_edit->text().isEmpty());

button[1]->setEnabled(check);

button[1]->setDefault(check);

button[2]->setDefault(!check);

}

void AddMarkWindow::saveMark() {

QString temp = line\_edit->text();

if(temp[temp.size()-1] == ' ') temp.remove(temp.size()-1, 1);

emit saveClicked(temp);

close();

}

void AddMarkWindow::clear() {

line\_edit->clear();

}

bool AddMarkWindow::close() {

clear();

return QWidget::close();

}

void AddMarkWindow::reject() {

clear();

QDialog::reject();

}

//addservicewindow.hpp

#pragma once

#include <QDialog>

#include <QLabel>

#include <QLineEdit>

#include <QPushButton>

#include "autoservice.hpp"

class AddServiceWindow : public QDialog

{

Q\_OBJECT

QLabel \* label[2];

QLineEdit \* line\_edit[2];

QPushButton \* button[3];

int current\_mark;

public:

AddServiceWindow(QWidget \*parent);

~AddServiceWindow();

void uploadMark(const int& mark\_ID);

public slots:

void updateSaveButton();

void saveService();

void clear();

bool close();

void reject();

signals:

void saveClicked(AutoService new\_service);

};

//addservicxewindow.cpp

#include "addservicewindow.hpp"

#include <QIcon>

#include <QHBoxLayout>

#include <QVBoxLayout>

AddServiceWindow::AddServiceWindow(QWidget \*parent)

: QDialog(parent), current\_mark(-1)

{

setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);

setWindowTitle(QString("Добавить сервис"));

setWindowIcon(QIcon(":/resources/addservice\_icon.png"));

setFixedSize(300, 120);

QVBoxLayout \* central\_layout = new QVBoxLayout;

{

QHBoxLayout \* name\_layout = new QHBoxLayout;

{

label[0] = new QLabel(QString("Название"));

line\_edit[0] = new QLineEdit;

{

line\_edit[0]->setFixedSize(200, 23);

connect(line\_edit[0], SIGNAL(textChanged(QString)), this,

SLOT(updateSaveButton()));

}

name\_layout->addWidget(label[0]);

name\_layout->addWidget(line\_edit[0]);

}

QHBoxLayout \* address\_layout = new QHBoxLayout;

{

label[1] = new QLabel(QString("Адрес"));

line\_edit[1] = new QLineEdit;

{

line\_edit[1]->setFixedSize(200, 23);

connect(line\_edit[1], SIGNAL(textChanged(QString)), this,

SLOT(updateSaveButton()));

}

address\_layout->addWidget(label[1]);

address\_layout->addWidget(line\_edit[1]);

}

QHBoxLayout \* buttons\_layout = new QHBoxLayout;

{

button[0] = new QPushButton(QString("Очистить"));

{

connect(button[0], SIGNAL(clicked(bool)), this,

SLOT(clear()));

}

button[1] = new QPushButton(QString("Сохранить"));

{

button[1]->setEnabled(false);

connect(button[1], SIGNAL(clicked(bool)),

SLOT(saveService()));

}

button[2] = new QPushButton(QString("Отмена"));

{

button[2]->setDefault(true);

connect(button[2], SIGNAL(clicked(bool)), this,

SLOT(close()));

}

buttons\_layout->addWidget(button[0]);

buttons\_layout->addWidget(button[1]);

buttons\_layout->addWidget(button[2]);

}

central\_layout->addLayout(name\_layout);

central\_layout->addLayout(address\_layout);

central\_layout->addLayout(buttons\_layout);

}

setLayout(central\_layout);

}

AddServiceWindow::~AddServiceWindow()

{

}

void AddServiceWindow::uploadMark(const int &mark\_ID) {

current\_mark = mark\_ID;

}

void AddServiceWindow::updateSaveButton() {

bool check = !(line\_edit[0]->text().isEmpty() ||

line\_edit[1]->text().isEmpty());

button[1]->setEnabled(check);

button[1]->setDefault(check);

button[2]->setDefault(!check);

}

void AddServiceWindow::saveService() {

emit saveClicked(AutoService(current\_mark,

UString(line\_edit[0]->text().toStdString().c\_str()),

UString(line\_edit[1]->text().toStdString().c\_str())));

close();

}

void AddServiceWindow::clear() {

for(int i = 0; i < 2; i++) line\_edit[i]->clear();

}

bool AddServiceWindow::close() {

clear();

return QWidget::close();

}

void AddServiceWindow::reject() {

clear();

QDialog::reject();

}

//servicelistwindow.hpp

#pragma once

#include <QDialog>

#include <QTableWidget>

#include <QPushButton>

#include <stack>

#include "autoservice.hpp"

class ServicesListWindow : public QDialog

{

Q\_OBJECT

QTableWidget \* table;

QPushButton \* button[4];

std::vector<AutoService> \* services;

int current\_mark;

std::stack<AutoService> deleted\_services;

public:

ServicesListWindow(QWidget \*parent);

~ServicesListWindow();

void uploadServices(std::vector<AutoService>& new\_services,

const int& mark\_ID);

public slots:

void callAddServiceWindow();

void addServiceElement(AutoService new\_service);

void deleteServiceElement();

void undoDeleting();

void clear();

bool close();

void reject();

};

//servicelistwindow.cpp

#include "serviceslistwindow.hpp"

#include "addservicewindow.hpp"

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QHeaderView>

#include <QMessageBox>

using namespace std;

ServicesListWindow::ServicesListWindow(QWidget \*parent)

: QDialog(parent), services(nullptr), current\_mark(-1)

{

setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);

setWindowTitle(QString("Сервисы"));

setWindowIcon(QIcon(":/resources/service\_icon.png"));

setFixedSize(500, 250);

QVBoxLayout \* central\_layout = new QVBoxLayout;

{

table = new QTableWidget;

{

table->setRowCount(0);

table->setColumnCount(2);

table->setColumnWidth(0, 150);

table->setColumnWidth(1, 309);

table->setEditTriggers(QAbstractItemView::NoEditTriggers);

table->setSelectionMode(QAbstractItemView::SingleSelection);

table->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

table->setHorizontalHeaderLabels(QStringList(

{QString("Название"),

QString("Адрес")}));

table->horizontalHeader()->setSectionResizeMode(

QHeaderView::Fixed);

table->horizontalHeader()->setSectionsClickable(false);

table->verticalHeader()->hide();

}

QHBoxLayout \* buttons\_layout = new QHBoxLayout;

{

button[0] = new QPushButton("Добавить сервис");

{

connect(button[0], SIGNAL(clicked(bool)),

SLOT(callAddServiceWindow()));

}

button[1] = new QPushButton("Удалить сервис");

{

connect(button[1], SIGNAL(clicked(bool)),

SLOT(deleteServiceElement()));

}

button[2] = new QPushButton("Отменить удаление");

{

button[2]->setEnabled(false);

connect(button[2], SIGNAL(clicked(bool)),

SLOT(undoDeleting()));

}

button[3] = new QPushButton("Закрыть");

{

connect(button[3], SIGNAL(clicked(bool)), SLOT(close()));

}

//buttons\_layout->addSpacing(100);

buttons\_layout->addWidget(button[0]);

buttons\_layout->addWidget(button[1]);

buttons\_layout->addWidget(button[2]);

buttons\_layout->addWidget(button[3]);

}

central\_layout->addWidget(table);

central\_layout->addLayout(buttons\_layout);

}

setLayout(central\_layout);

}

ServicesListWindow::~ServicesListWindow()

{

}

void ServicesListWindow::uploadServices(std::vector<AutoService> &new\_services, const int &mark\_ID) {

current\_mark = mark\_ID;

services = &new\_services;

unsigned int size = services->size();

table->setRowCount(size);

for(unsigned int i = 0; i < size; i++) {

table->setItem(i, 0, new QTableWidgetItem(

services->at(i).getName().getQString()));

table->setItem(i, 1, new QTableWidgetItem(

services->at(i).getAddress().getQString()));

}

}

void ServicesListWindow::callAddServiceWindow() {

static AddServiceWindow \* a = new AddServiceWindow(this);

static QMetaObject::Connection a\_c1 =

connect(this, SIGNAL(rejected()), a, SLOT(reject()));

static QMetaObject::Connection a\_c2 =

connect(a, SIGNAL(saveClicked(AutoService)),

SLOT(addServiceElement(AutoService)));

a->setWindowTitle(QString("Добавить сервис") +

QString(windowTitle().toStdString().c\_str() + 14));

a->uploadMark(current\_mark);

a->show();

this->setEnabled(false);

a->setEnabled(true);

a->exec();

this->setEnabled(true);

}

void ServicesListWindow::deleteServiceElement() {

if(table->rowCount() > 0 && table->currentRow() >= 0) {

deleted\_services.push(\*(services->begin() + table->currentRow()));

button[2]->setEnabled(true);

services->erase(services->begin() + table->currentRow());

uploadServices(\*services, current\_mark);

}

}

void ServicesListWindow::addServiceElement(AutoService new\_service) {

if(services->end() != std::find(services->cbegin(), services->cend(),

new\_service))

return;

services->insert(services->end(), new\_service);

sort(services->begin(), services->end());

uploadServices(\*services, current\_mark);

}

void ServicesListWindow::undoDeleting() {

addServiceElement(deleted\_services.top());

deleted\_services.pop();

if(deleted\_services.empty()) button[2]->setEnabled(false);

}

void ServicesListWindow::clear() {

table->clearContents();

table->setRowCount(0);

button[2]->setEnabled(false);

while(!deleted\_services.empty())

deleted\_services.pop();

}

bool ServicesListWindow::close() {

clear();

return QWidget::close();

}

void ServicesListWindow::reject() {

clear();

QDialog::reject();

}

//aboutwindow.hpp

#pragma once

#include <QDialog>

#include <QPushButton>

#include <QLabel>

class AboutWindow : public QDialog

{

Q\_OBJECT

QLabel \* label[4];

QPushButton \* button;

public:

AboutWindow(QWidget \* parent);

~AboutWindow();

};

//aboutwindow.cpp

#include "aboutwindow.hpp"

AboutWindow::AboutWindow(QWidget \*parent)

: QDialog(parent)

{

setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);

setWindowTitle(QString("О программе"));

setFixedSize(210, 120);

button = new QPushButton(QString("Закрыть"), this);

button->setGeometry(QRect(130, 90, 75, 23));

connect(button, SIGNAL(clicked(bool)), SLOT(close()));

label[0] = new QLabel(QString("Программный модуль \"Автосалон\","),this);

label[0]->setGeometry(QRect(10, 10, 180, 16));

label[1] = new QLabel(QString("Версия 1.0"), this);

label[1]->setGeometry(QRect(10, 25, 180, 15));

label[2] = new QLabel(QString("Автор: Холупко Игорь Сергеевич,"), this);

label[2]->setGeometry(QRect(10, 50, 180, 15));

label[3] = new QLabel(QString("БГУИР, ФКСиС, гр. 650501"), this);

label[3]->setGeometry(QRect(10, 65, 180, 15));

}

AboutWindow::~AboutWindow()

{

}