

Содержание

[Введение 2](#_Toc9986045)

[1. Теоретическая часть 6](#_Toc9986046)

[1.1. Аудиотека 6](#_Toc9986047)

[1.2. Постановка задачи 6](#_Toc9986048)

[1.3. Обоснование классов 7](#_Toc9986049)

[1.4. Описание выбора средств разработки 11](#_Toc9986050)

[1.4.1. Описание выбора языка программирование С++ 11](#_Toc9986051)

[1.4.2. Описание выбора Qt 11](#_Toc9986052)

[2. Практическая часть 12](#_Toc9986053)

[2.1. Разработка и описание алгоритмов 12](#_Toc9986054)

[2.2. Разработка интерфейса программы 13](#_Toc9986055)

[3. Программная документация 17](#_Toc9986056)

[3.1. Требование к программе 17](#_Toc9986057)

[3.2. Входные и выходные данные 17](#_Toc9986058)

[4. Руководство оператора 18](#_Toc9986059)

[5. Тестирование программного обеспечения 21](#_Toc9986060)

[5.1. Определение тестирования программного обеспечения 21](#_Toc9986061)

[5.2. Результаты тестирования системы 22](#_Toc9986062)

[6. Список используемой литературы 31](#_Toc9986063)

[7. Приложение 32](#_Toc9986064)

# Введение

Современный уровень развития различных программных и аппаратно-программных систем, и в первую очередь систем управления, связан со значительным повышением их сложности из-за присущих им требований к гибкости, открытости, эргономичности и перенастраиваемости.

Проектирование таких сложных систем с трудом укладывается в рамки традиционного подхода разработки, когда процесс создания системы рассматривается как последовательность отдельных завершенных этапов и считается, что работа имеет однонаправленный характер.

Все это требует новых подходов к проектированию. На данный момент наиболее перспективным можно считать объектно-ориентированный подход (ООП) проектирования. К существенным отличиям объектно-ориентированного подхода от традиционного проектирования относят возможность использования итеративно-поступательного цикла создания программного обеспечения и перенос акцента проектирования с разработки алгоритмов функционирования системы на построения системы абстракций и их взаимодействия.

Объектно-ориентированный подход включает в себя такие понятия:

* Абстракция — это придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня. Такой подход является основой объектно-ориентированного программирования. Это позволяет работать с объектами, не вдаваясь в особенности их реализации. В каждом конкретном случае применяется тот или иной подход: инкапсуляция, полиморфизм или наследование.
* Инкапсуляция - это принцип, согласно которому любой класс должен рассматриваться как чёрный ящик - пользователь класса должен видеть и использовать только интерфейсную часть класса (т. е. список декларируемых свойств и методов класса) и не вникать в его внутреннюю реализацию. Поэтому данные принято инкапсулировать в классе таким образом, чтобы доступ к ним по чтению или записи осуществлялся не напрямую, а с помощью методов. Принцип инкапсуляции (теоретически) позволяет минимизировать число связей между классами и, соответственно, упростить независимую реализацию и модификацию классов. Сокрытие данных — неотделимая часть ООП, управляющая областями видимости. Является логическим продолжением инкапсуляции. Целью сокрытия является невозможность для пользователя узнать или испортить внутреннее состояние объекта.
* Наследованием называется возможность порождать один класс от другого с сохранением всех свойств и методов класса-предка (прародителя, иногда его называют суперклассом) и добавляя, при необходимости, новые свойства и методы. Набор классов, связанных отношением наследования, называют иерархией. Наследование призвано отобразить такое свойство реального мира, как иерархичность.
* Полиморфизмом называют явление, при котором функции (методу) с одним и тем же именем соответствует разный программный код (полиморфный код) в зависимости от того, объект какого класса используется при вызове данного метода. Полиморфизм обеспечивается тем, что в классе-потомке изменяют реализацию метода класса-предка с обязательным сохранением сигнатуры метода. Это обеспечивает сохранение неизменным интерфейса класса-предка и позволяет осуществить связывание имени метода в коде с разными классами - из объекта какого класса осуществляется вызов, из того класса и берётся метод с данным именем. Такой механизм называется динамическим (или поздним) связыванием - в отличие от статического (раннего) связывания, осуществляемого на этапе компиляции.

Актуальность выбранного метода разработки иерархии классов – упрощение структурирования данных, гибкость применения созданного алгоритма, открытость, а также эргономичность и перенастраиваемость, позволяющие подстраивать созданные алгоритмы под требования пользователя.

В мире есть множество программных продуктов, позволяющих выполнять задачи по структурированию данных подобного типа, а также выполняющих различные действия над ними. Главной отличительной особенностью данного приложения является хорошая переносимость на разные ОС, маленький объем и базовый набор действий с аудио. Также данное приложение имеет простой пользовательский интерфейс, интуитивно понятные действия, возможность корректировки, поиска и удаления данных.

Основными задачами курсового проекта являются – разработка приложения с использованием структуры иерархии классов по заданной тематике, провести тестирование приложения, разработать справочную систему для созданного приложения.

# Теоретическая часть

## Аудиотека

Аудиотека – это библиотека, только вместо книг – аудиофайлы. В ней хранятся множество аудиофайлов разных форматов, жанров, направлений и т.д.

## Постановка задачи

В данном курсовом проекте необходимо разработать структуру иерархии классов «Аудиотека». На основе этой структуры необходимо создать приложение для тестирования иерархии**.**

Для реализации поставленных задач необходимо:

* Разработать иерархию классов, определить базовый и наследуемые классы;
* Разработать и описать структуру каждого класса в отдельности, объявить поля и методы класса;
* Реализовать визуальные компоненты для работы с иерархией классов.

Необходимо реализовать проект, который будет выполнять следующие действия:

* хранение данных о аудио в файле;
* просмотр существующей данных;
* добавление новой аудио;
* удаление существующего аудио;
* изменение выбранного аудио;
* фиксация изменения;
* сортировка по некоторым полям;
* поиск и просмотр полученных совпадений.

В соответствии с поставленными задача, необходимо разработать приложение «Аудиоплеер» со списком прослушиваемых аудио, который можно редактировать: добавлять, удалять и изменять характеристики аудио.

Данный курсовой проект нацелен на создание приложения, позволяющего хранить аудиофайлы, воспроизводить их и просматривать информацию о них в одном месте.

Целью проекта является разработка структуры класса «Аудиотека», которая будет основой для хранения информации о аудиофайлах и их характеристиках. Также целью проекта является развитие навыков самостоятельной и творческой работы и закрепление навыков работы на языке С++.

## Обоснование классов

Самым базовым и универсальным классом, подходящим почти под любую предметную область, является товар. У любого товара есть наименование, цена и модель, но для решения данной задачи нам необходимо лишь наименование. Следовательно, будет базовый абстрактный класс «Product» с полем «Name». Дальнейшее расширение данного класса может представлено в виде списка товаров, различных предметных областей.

Аудио является разновидностью компьютерного файла, имея схожие черты со всеми остальными видами файлов. Например, общими характеристиками аудиофайла и текстового файла являются:

* путь к файлу,
* его тип,
* размер,
* язык,
* дата создания/изменения/добавления и т.д.

Значит необходимо выделить класс уровнем ниже класса «Product», который будет его наследником и названием его будет «File». Данный класс должен содержать следующие характеристики: путь к файлу, дата добавления и язык. Дальнейшее применение данного может быть представлено в виде проигрывателя видео- и аудиофайлов. То есть в списке будут как видео- и аудиофайлы, над которыми будут выполнятся все те же действия, что и в аудиоплеере.

Аудиотека – это хранилище аудио. Следовательно, следующий класс будет «Audio». У всех аудио есть общие характеристики:

* исполнитель,
* жанр,
* продолжительность.

В свою очередь аудио можно разделить на две большие категории: музыкальные записи и аудиокниги. Соответственно, у класса «Audio» будет два наследника «Music» и «AudioBook», каждый из которых имеет свои отличительные характеристики. Такие как автор книги или музыкальный альбом. Работа с данным классом представлена в моей курсовой работе. Используется список указателей на Audio, при заполнении списка данными этот родительский класс хранить ссылки на объекты классов Music, AudioBook.

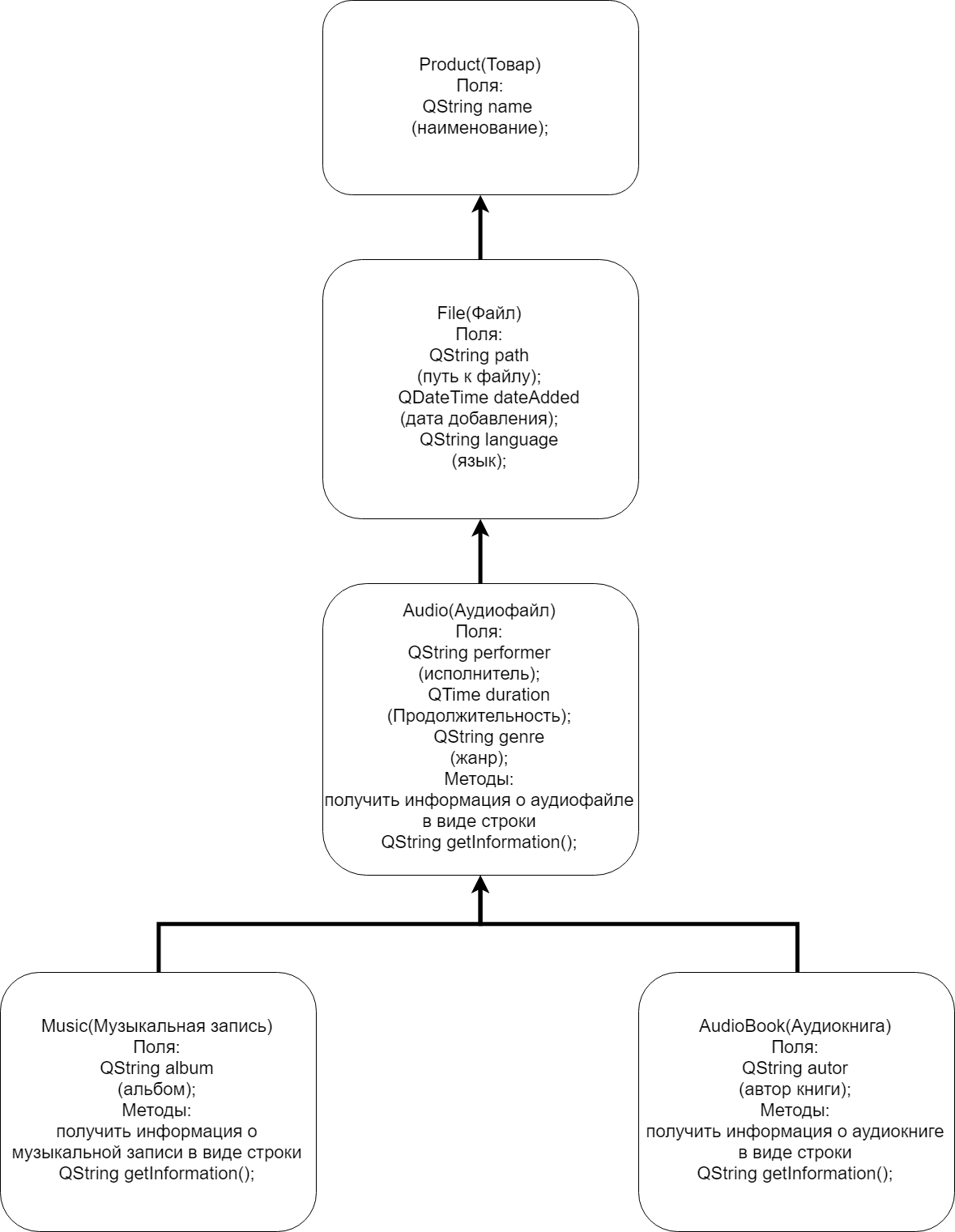


Рис.1 «Диаграмма иерархии классов»

На Рис.1 есть базовый абстрактный класс «Product» с полями:

* name – наименование

Поле name будет использоваться для сортировки и поиска. Например, пользователь захочет увидеть список отсортированный по алфавиту, сортировка будет выполняться по этому полю, или пользователь захочет найти аудио по его названию.

Класс «File» с полями:

* path– путь к файлу
* dateAdded– дата добавления
* language– язык

Поле path будет использоваться для воспроизведения аудио.

Поле dateAdded будет использоваться для сортировки по умолчанию.

Поле language будет использоваться для поиска всех аудио по определенному языку.

Класс «Audio» с полями:

* performer– исполнитель
* duration– продолжительность
* genre– жанр

Поле performer будет использоваться для поиска аудио по её исполнителю.

Поле duration использоваться для воспроизведения аудио.

Поле genre будет использоваться для поиска аудио по определенному жанру.

Класс «Music» с полями:

* album – альбом

Поле album будет использоваться для поиска музыки, принадлежащей определенному альбому.

Класс «AudioBook» с полями:

* autor – автор

Поле autor будет использоваться для поиска аудиокниг, принадлежащих определенному автору.

## Описание выбора средств разработки

### Описание выбора языка программирование С++

Важной частью в разработке программного обеспечения выбор языка программирования, среды для разработки и компилятора.

Чтобы облегчить работу программиста практически все современные компиляторы с языка C++ содержат специальные библиотеки классов. Такие библиотеки включают в себя практически весь программный интерфейс Windows и позволяют пользоваться при программировании средствами более высокого уровня, чем обычные вызовы функций. За счет этого значительно упрощается разработка приложений, имеющих сложный интерфейс пользователя, облегчается поддержка технологии OLE и взаимодействие с базами данных.

Выбора данного языка программирования обусловлен спецификой изучаемого курса.

### Описание выбора Qt

Для разработки данного программного обеспечения использовался язык С++ и компилятор MCVS2017 и набирающая свою популярность среда разработки QT Creator.

Qt является кроссплатформенным фреймворком для разработки приложений от компании Trolltech, а до недавнего времени владельцем кода является компания Nokia (на данный момент компания Nokia передала разработку Qt компании Digia). Фреймворк предоставляет API на языке C++. Qt часто используется разработчиками для разработки кроссплатформенных приложений.

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных, XML и 3D-графикой. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Основным аспектом выбора данной среды разработки было наличие классов для работы с проигрывание аудио QMediaPlayer и QMediaPlaylist.

# Практическая часть

## Разработка и описание алгоритмов

Общий алгоритм программы заключается в том, что система ожидает какого-либо события (нажатие на клавишу клавиатуры, клик по визуальной кнопки формы приложения и т.д.) и после свершения такого события, система запускает тот или иной обработчик.

Например, пользователь нажал на кнопку «Добавить аудио», то обработчик будет выглядеть так:

void MainWindow::on\_addAudio()

{

AudioForm \* form = new AudioForm(this);

form->setModal(true);

if(form->exec() == QDialog::Accepted)

{

if(dataBase->addAudio(form->audio) == 0)

{

this->setTableData(dataBase->getListAudio());

warning.setWindowTitle(this->windowTitle());

warning.setText("Аудио добавлено.");

warning.setIcon(QMessageBox::Information);

warning.exec();

}

else

{

this->setTableData(dataBase->getListAudio());

warning.setWindowTitle(this->windowTitle());

warning.setText("Такая аудиозапись уже существует.");

warning.setIcon(QMessageBox::Warning);

warning.exec();

}

}

}

## Разработка интерфейса программы

На главной форме необходимо разместить следующие компоненты:

* Меню с пунктами
* Таблица данных
* Кнопки добавления и удаления
* Кнопки воспроизведения, паузы, стоп, следующее, предыдущее
* Кнопку поиска с полем вводы для строки поиска
* Также кнопку для регулировки звука и времени воспроизведения.

Интерфейс программы изображен на Рис.2.

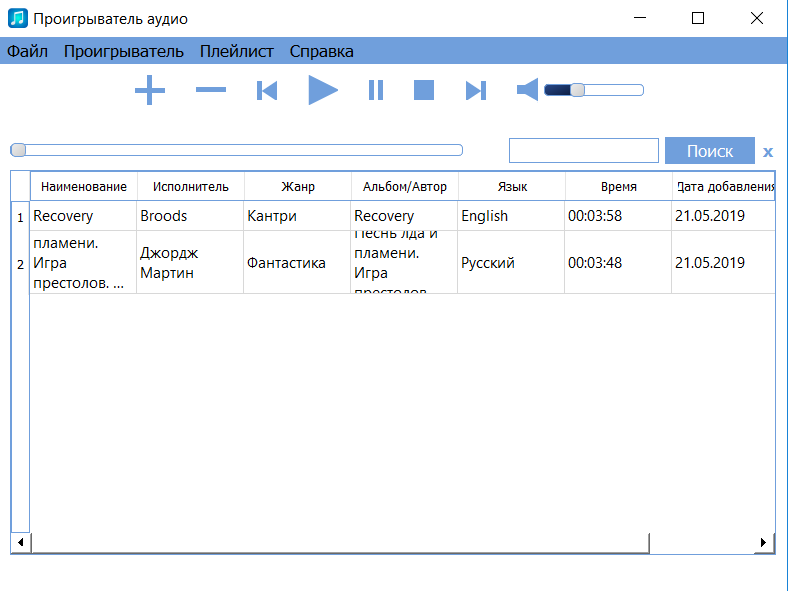


Рис.2 «Интерфейс главной формы программы»

Для ввода данных при добавлении и изменения аудио необходимо разработать модальную форму со всеми необходимыми полями ввода и выпадающими списками.

Модальная форма изображена на Рис.3.

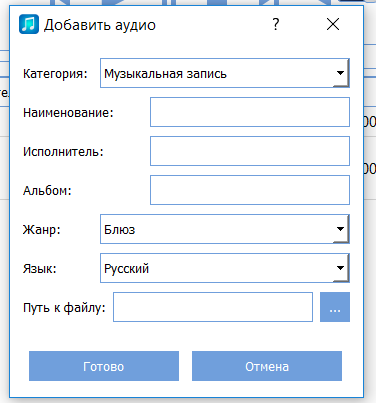


Рис.3 «Интерфейс формы для редактирования аудио»

Справка приложения разработана с помощью приложения Dr.Explain и изображена на Рис.4.

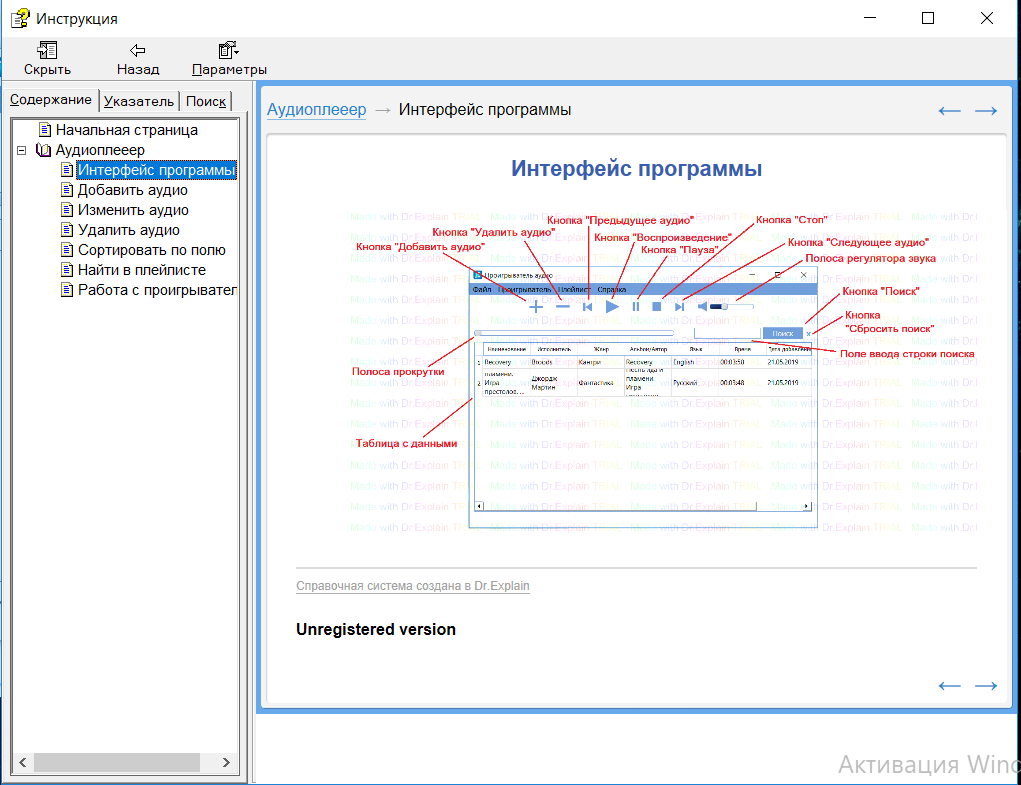


Рис.4 «Справка программы»

Все формы разработаны стандартными компонентами QT и спроектированы в QT Designer.

Таблица данных – QTableView, меню – QMenu, кнопки – QPushButton, поля вводы – QLineEdit, статический текст – QLable, полосы прокрутки – QSilder

Все сообщения, которые выводятся на экран, сделаны стандартным способом через QMessageBox, с измененными заголовком и содержащимся текстом.

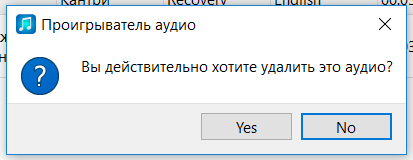


Рис.5 «Сообщение подтверждение удаления аудио»

# Программная документация

## Требование к программе

* Процессор: Intel 2200 МГц и выше.
* Операционная система: Microsoft Windows 7, Windows 10.
* Оперативная память: 256 Мбайт.
* Пространство на жестком диске: 25 Мбайт.

## Входные и выходные данные

Входные данные: категория аудио, его наименование, жанр, путь к файлу .mp3 и т.д.

Выходные данные: визуальное их представление в виде таблицы.

# Руководство оператора

Управление данными происходит посредством выбора функций в форме программы.

Для работы с данными реализован блок функций, который называется **«**Блок управления данными**»**. Данный блок состоит из таких функций:

* Добавить аудио.
* Изменить аудио.
* Удалить аудио.

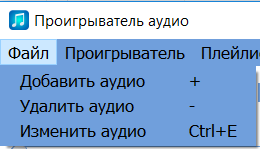


Рис.6 **«**Команды управления данными через главное меню**»**



Рис.7 **«**Кнопки управления данными**»**

Для управления проигрывателем существует «Блок управления проигрывателем».

* Воспроизведение аудио;
* Пауза;
* Стоп;
* Следующее аудио;
* Предыдущее аудио;
* Регуляция звука;
* Регуляция времени воспроизведения аудио.

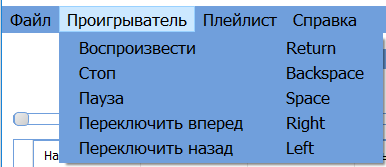


Рис.8 **«**Команды управления проигрывателем через главное меню**»**



Рис.9 **«**Кнопки для управления**»**



Рис.10 **«**Регулятор времени воспроизведения аудио**»**

Для визуального отображения данных используется таблица.

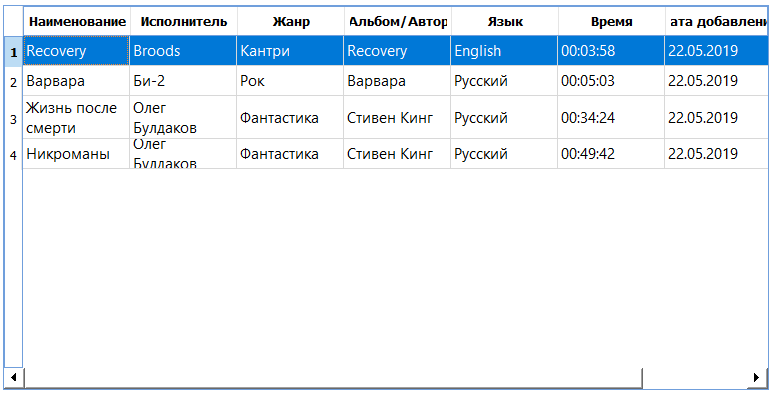


Рис.11 «Таблица отображения данных о аудио»

После нажатия на столбцы «Наименование», «Исполнитель», «Время» и «Дата добавления» происходит сортировка по перечисленным полям.

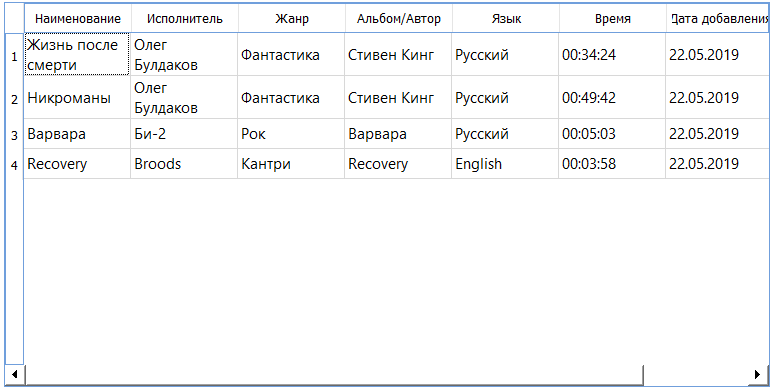


Рис.12 «Таблица данных, отсортированных по полю «Исполнитель»

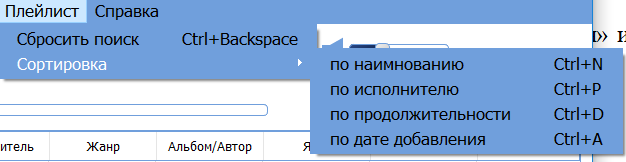


Рис.13 «Сортировка с помощью главного меню программы»

Для поиска по аудио используется «Блок для поиска по данным».



Рис.14 «Блок для поиска по данным»

Для вызова справки программы существует команда «Справка».

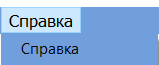


Рис.15 «Команда вызова справки».

# Тестирование программного обеспечения

## Определение тестирования программного обеспечения

Тестирование программного обеспечения (ПО) выявляет недоработки, изъяны и ошибки в коде, которые необходимо устранить. Его также можно определить как процесс оценки функциональных возможностей и корректности ПО с помощью анализа. Основные методы интеграции и тестирования программных продуктов обеспечивают качество приложений и заключаются в проверке спецификации, дизайна и кода, оценке надежности, валидации и верификации.

Главная цель тестирования ПО – подтверждение качества программного комплекса путем систематической отладки приложений в тщательно контролируемых условиях, определение их полноты и корректности, а также обнаружение скрытых ошибок. Методы проверки (тестирования) программ можно разделить на статические и динамические. К первым относятся неформальное, контрольное и техническое рецензирование, инспекция, пошаговый разбор, аудит, а также статический анализ потока данных и управления.

Динамические техники следующие:

* Тестирование методом белого ящика. Это подробное исследование внутренней логики и структуры программы. При этом необходимо знание исходного кода.
* Тестирование методом черного ящика. Данная техника не требует каких-либо знаний о внутренней работе приложения. Рассматриваются только основные аспекты системы, не связанные или мало связанные с ее внутренней логической структурой.

При тестировании данного программного обеспечения использовались оба варианта.

В ходе тестирования программного обеспечения по стратегии «Белого ящика» использовались два вида испытаний:

* Статический и динамический анализ

Статический анализ состоит в просмотре кода, чтобы выяснить любые возможные дефекты в коде, динамический анализ предполагает выполнение кода и анализ выходных данных.

* Покрытие решений

Ни одно приложение не может быть написано в непрерывном режиме кодирования. В какой-то момент мы должны разветвить код для того, чтобы выполнить ту или иную функциональность. Тестирование покрытия решений помогает в проверке всех ветвей в коде, и помогает убедиться, что ветвление не приводит к непредсказуемому поведению приложения.

## Результаты тестирования системы

Блок тестирования по стратегии «Черного ящика»:

1. Необходимо добавить аудиокнигу жанра «Фантастика» с названием «Жизнь после смерти» Стивена Кинга в исполнении Олега Булдакова.

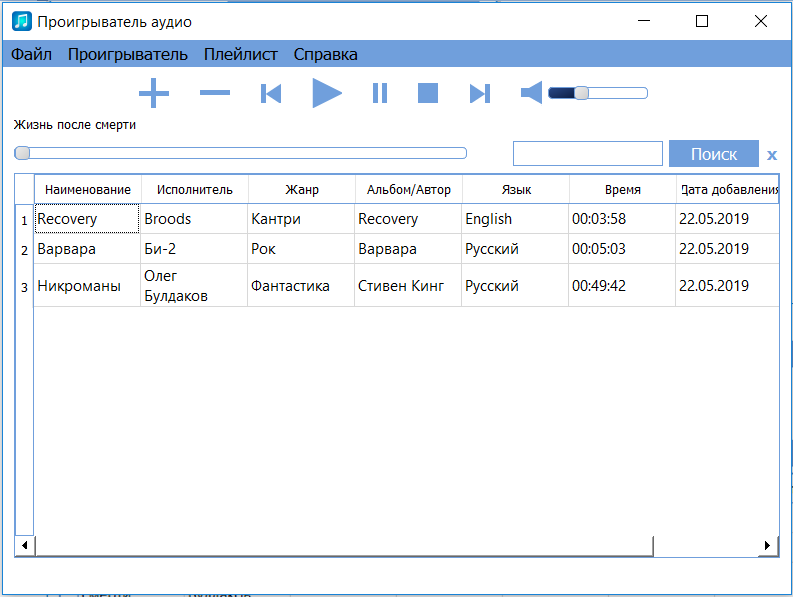


Рис.16 «Состояние программа до добавления аудио»

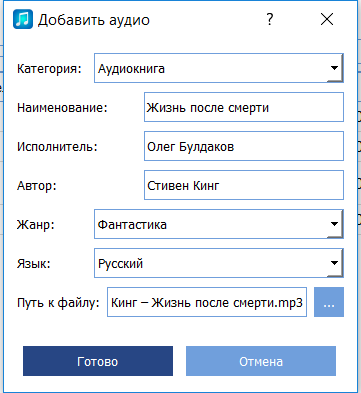


Рис.17 «Процесс добавления аудио»

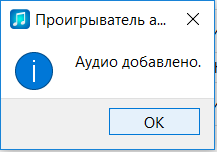


Рис.18 «Сообщение успешного добавления аудио»

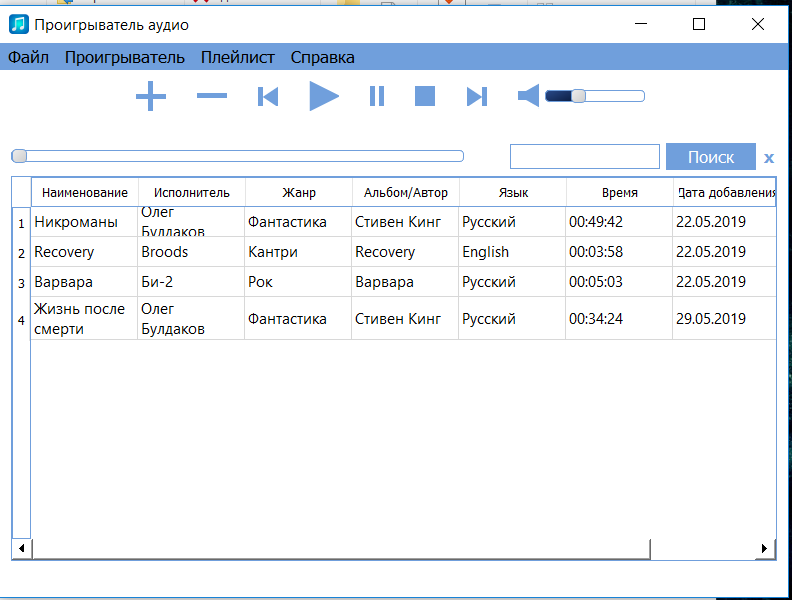


Рис.19 «Состояние программы после добавления программы»

В результате в таблице данных появилось добавленное нами аудио.

1. Необходимо изменить жанр с «Кантри» на «Рок» музыкальной записи под номером 2.

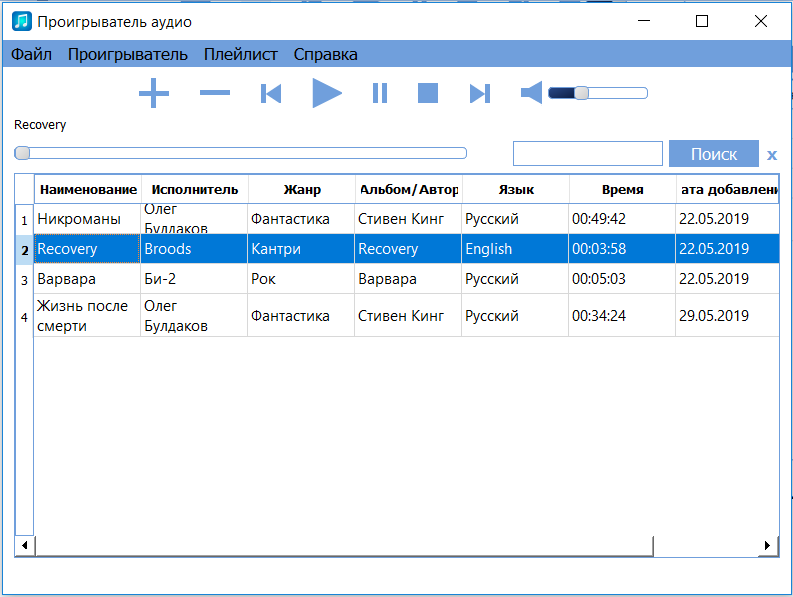


Рис.20 «Состояние программа до изменения аудио»

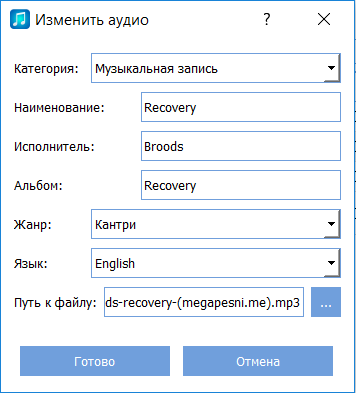


Рис.21 «Процесс изменения аудио»

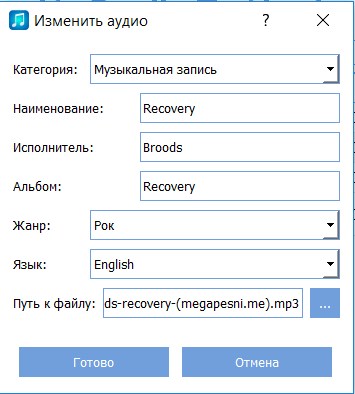


Рис.22 «Процесс изменения аудио»

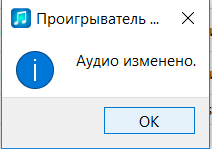


Рис.23 «Сообщение успешного изменения аудио»

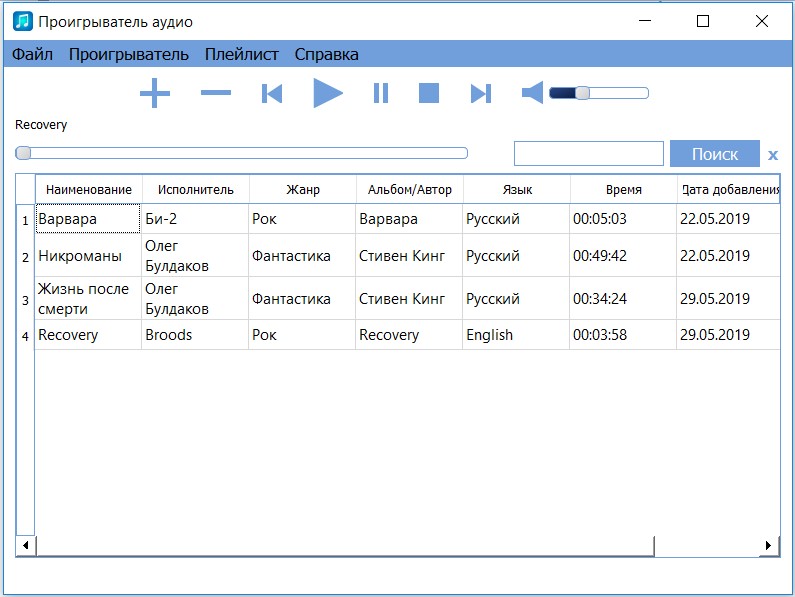


Рис.24 «Состояние программы после изменения аудио»

1. Необходимо удалить аудио под номером 2.

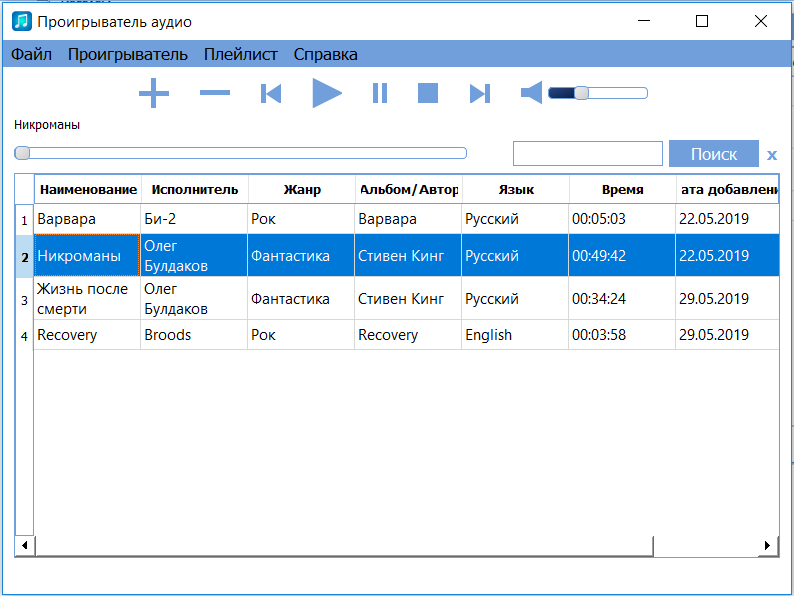


Рис.25 «Состояние программы до удаления аудио»

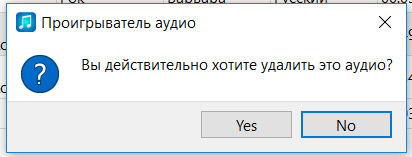


Рис.26 «Сообщение подтверждение удаления аудио»

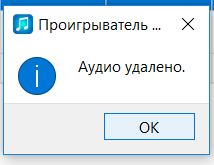


Рис.27 «Сообщение успешного удаления аудио»

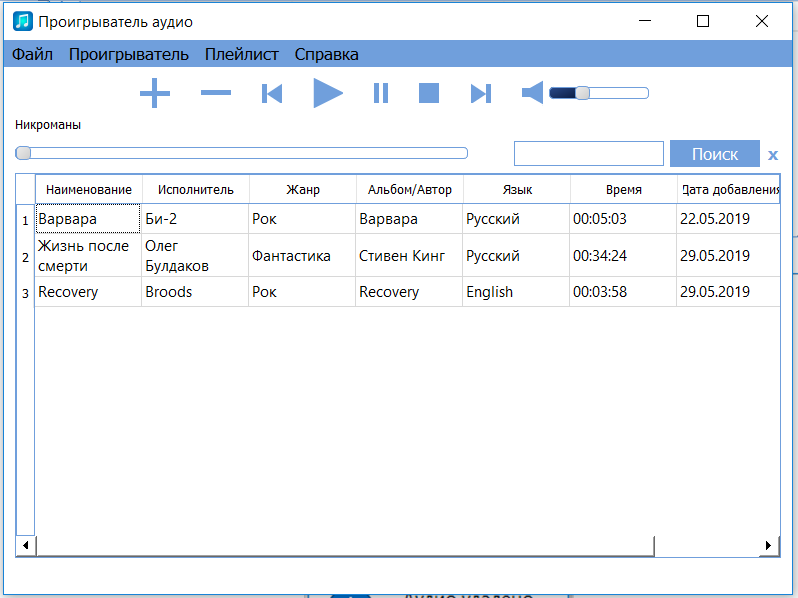


Рис.28 «Состояние программы после удаления аудио»

В результате этого блока был выявлен и устранен в следствии недочет. При изменения аудио дата добавления изменялась на текущую.

В ходе испытаний прибегая к статическому и динамическому анализу стратегии «Белого ящика» был найден и справлен следующий недочет:

QList<Audio\*>\* DataBase::**searchString**(QString inputData)

{

QRegExp date("(0[1-9]|(([1]|[2])[0-9])|3[0-1])[- /.](0[1-9]|1[0-2])[- /.](20(19|20))");

QRegExp time("^([0-1][0-9]|2[0-3])(:[0-5][0-9]){2}$");

QList<Audio\*>\* searched = new QList<Audio\*>();

for(QList<Audio\*>::iterator it = this->listAudio->begin(); it != this->listAudio->end(); it++)

{

Audio \* audio = \*it;

if(date.exactMatch(inputData) == true)

{

QDate d = QDate::fromString(inputData, "dd.MM.yyyy");

int y = audio->getDateAdded().year();

int m = audio->getDateAdded().month();

int day = audio->getDateAdded().day();

int y1 = d.year();

int m1 = d.month();

int day1 = d.day();

if((m == m1) && (y == y1) && ((day1 - 5) <= day && day <= (day1 + 5)))

{

searched->append(audio);

}

}

if(time.exactMatch(inputData) == true)

{

QTime t = QTime::fromString(inputData);

int h = audio->getDuration().hour();

int m = audio->getDuration().minute();

int h1 = t.hour();

int m1 = t.minute();

if((h == h1) && ((m1 - 2) <= m && m <= (m1 + 5)))

{

searched->append(audio);

}

}

if(time.exactMatch(inputData) == false && date.exactMatch(inputData) == false && audio->*getInformation*().toLower().contains(inputData.toLower()))

{

searched->append(audio);

}

}

return searched;

}

При проверки диапазона дня даты и минут времени были заданы такие строчки: ((day1 - 5) <= day <= (day1 + 5)) и ((m1 - 2) <= m (m1 + 5)). Данная запись кода работала некорректно и в следствии была заменена на код, который виден выше.

Тестируя программу методом «Покрытия решений» ошибок или недочетов выявлено не было. Все ветки программы работают корректно и не выдают никаких исключений и неверных результатов.

# Список используемой литературы

1. <https://doc.qt.io>
2. <http://fb.ru/article/247668/metodyi-testirovaniya-programmnogo-obespecheniya-i-ih-sravnenie-testirovanie-metodom-chernogo-yaschika-i-testirovanie-metodom-belogo-yaschika>
3. <http://juice-health.ru/program/software-testing/493-white-box>
4. <https://studref.com/311869/informatika/laboratornaya_rabota_testirovanie_programm_metodami_belogo_yaschika>
5. <https://code.qt.io>

# Приложение

#pragma

#ifndef PRODUCT\_H

#define PRODUCT\_H

#include <QString>

#include <string>

class **Product**

{

private:

QString name;

public:

**Product**(QString const& name);

void **setName**(QString const& name);

QString **getName**();

};

#endif // PRODUCT\_H

#include "product.h"

Product::**Product**(QString const& name)

{

this->setName(name);

}

void Product::**setName**(QString const& name)

{

this->name = name;

}

QString Product::**getName**()

{

return this->name;

}

#pragma

#ifndef FILE\_H

#define FILE\_H

#include "product.h"

#include <QDateTime>

class **File** : public Product

{

private:

QString path;

QDate dateAdded;

QString language;

public:

**File**(QString const& name, QString const& path, QDate const& date, QString const& language);

void **setPath**(QString const& path);

QString **getPath**();

void **setDateAdded**(QDate const& date);

QDate **getDateAdded**();

void **setLanguage**(QString const& language);

QString **getLanguage**();

};

#endif // FILE\_H

#include "file.h"

File::**File**(QString const& name, QString const& path, QDate const& date, QString const& language) : Product(name)

{

this->setPath(path);

this->setDateAdded(date);

this->setLanguage(language);

}

void File::**setPath**(QString const& path)

{

this->path = path;

}

QString File::**getPath**()

{

return this->path;

}

void File::**setDateAdded**(QDate const& date)

{

this->dateAdded = date;

}

QDate File::**getDateAdded**()

{

return this->dateAdded;

}

void File::**setLanguage**(QString const& language)

{

this->language = language;

}

QString File::**getLanguage**()

{

return this->language;

}

#pragma

#ifndef AUDIO\_H

#define AUDIO\_H

#include "file.h"

class **Audio** : public File

{

private:

QString performer;

QString genre;

QTime duration;

public:

**Audio**(QString const& name, QString const& path, QDate const& date, QString const& language,

QString const& performer, QString const& genre, QTime const& time);

void **setPerformer**(QString const& performer);

QString **getPerformer**();

void **setGenre**(QString const& genre);

QString **getGenre**();

void **setDuration**(QTime const& time);

QTime **getDuration**();

virtual QString ***getCategory***() = 0;

virtual QString ***getInformation***() = 0;

virtual QString ***getFeature***() = 0;

bool operator==(Audio & audio);

};

#endif // AUDIO\_H

#include "audio.h"

Audio::**Audio**(QString const& name, QString const& path, QDate const& date, QString const& language,

QString const& performer, QString const& genre, QTime const& time)

: File(name, path, date, language)

{

this->setPerformer(performer);

this->setGenre(genre);

this->setDuration(time);

}

void Audio::**setPerformer**(QString const& performer)

{

this->performer = performer;

}

QString Audio::**getPerformer**()

{

return this->performer;

}

void Audio::**setGenre**(QString const& genre)

{

this->genre = genre;

}

QString Audio::**getGenre**()

{

return this->genre;

}

void Audio::**setDuration**(QTime const& time)

{

this->duration = time;

}

QTime Audio::**getDuration**()

{

return this->duration;

}

bool Audio::operator==(Audio & audio)

{

if(this->getName() == audio.getName() || this->getPath() == audio.getPath())

{

return true;

}

return false;

}

#pragma

#ifndef MUSIC\_H

#define MUSIC\_H

#include "audio.h"

class **Music** : public Audio

{

private:

QString album;

public:

**Music**(QString const& name, QString const& path, QDate const& date, QString const& language,

QString const& performer, QString const& genre, QTime const& time, QString const& album);

void **setAlbum**(QString const& album);

QString **getAlbum**();

QString ***getInformation***() override;

QString ***getFeature***() override;

QString ***getCategory***() override;

};

#endif // MUSIC\_H

#include "music.h"

Music::**Music**(QString const& name, QString const& path, QDate const& date, QString const& language,

QString const& performer, QString const& genre, QTime const& time, QString const& album)

: Audio(name, path, date, language, performer, genre, time)

{

this->setAlbum(album);

}

void Music::**setAlbum**(QString const& album)

{

this->album = album;

}

QString Music::**getAlbum**()

{

return this->album;

}

QString Music::***getInformation***()

{

return "Музыка|" + this->getName() + "|" + this->getPath() + "|" + this->getDateAdded().toString("dd.MM.yyyy")

+ "|" + this->getLanguage() + "|" + this->getPerformer() + "|" + this->getGenre() + "|"

+ this->getDuration().toString("hh:mm:ss") + "|" + this->getAlbum();

}

QString Music::***getFeature***()

{

return this->getAlbum();

}

QString Music::***getCategory***()

{

return "Музыкальная запись";

}

#pragma

#ifndef AUDIOBOOK\_H

#define AUDIOBOOK\_H

#include "audio.h"

class **AudioBook** : public Audio

{

private:

QString autor;

public:

**AudioBook**(QString const& name, QString const& path, QDate const& date, QString const& language,

QString const& performer, QString const& genre, QTime const& time, QString const& autor);

void **setAutor**(QString const& autor);

QString **getAutor**();

QString ***getInformation***() override;

QString ***getFeature***() override;

QString ***getCategory***() override;

};

#endif // AUDIOBOOK\_H

#include "audiobook.h"

AudioBook::**AudioBook**(QString const& name, QString const& path, QDate const& date, QString const& language,

QString const& performer, QString const& genre, QTime const& time, QString const& autor)

: Audio(name, path, date, language, performer, genre, time)

{

this->setAutor(autor);

}

void AudioBook::**setAutor**(QString const& autor)

{

this->autor = autor;

}

QString AudioBook::**getAutor**()

{

return this->autor;

}

QString AudioBook::***getInformation***()

{

return "Аудиокнига|" + this->getName() + "|" + this->getPath() + "|" + this->getDateAdded().toString("dd.MM.yyyy")

+ "|" + this->getLanguage() + "|" + this->getPerformer() + "|" + this->getGenre() + "|"

+ this->getDuration().toString("hh:mm:ss") + "|" + this->getAutor();

}

QString AudioBook::***getFeature***()

{

return this->getAutor();

}

QString AudioBook::***getCategory***()

{

return "Аудиокнига";

}

#ifndef DATABASE\_H

#define DATABASE\_H

#include <QList>

#include <QTextStream>

#include "music.h"

#include "audiobook.h"

#include <QFileInfo>

#include <QRegExp>

class **DataBase**

{

private:

QList<Audio\*>\* listAudio;

QString pathFromFile;

public:

**DataBase**(QString const& path);

~**DataBase**();

bool **findAudio**(Audio \* audio);

void **updateDataBaseFromFile**();

int **addAudio**(Audio \* audio);

void **deleteAudio**(Audio \* audio);

QList<Audio\*>\* **getListAudio**();

void **setDataBaseFromFile**();

void **editAudio**(Audio \* audio, Audio \* editAudio);

QList<Audio\*>\* **searchString**(QString inputData);

void **sort**(int index, int status);

QString **getInformation**();

void **setPathFromFile**(QString const& path);

QString **getPathFromFile**();

void **setStatusSort**(int k);

};

#endif // DATABASE\_H

#include "database.h"

DataBase::**DataBase**(QString const& path)

{

this->setPathFromFile(path);

this->listAudio = new QList<Audio\*>();

this->setDataBaseFromFile();

this->sort(6,1);

}

DataBase::~**DataBase**()

{

delete listAudio;

}

bool DataBase::**findAudio**(Audio \* audio)

{

QList<Audio\*>\* list = this->listAudio;

for(QList<Audio\*>::iterator it = list->begin(); it != list->end(); it++)

{

Audio \* thisAudio = \*it;

if(thisAudio == audio)

{

return true;

}

}

return false;

}

void DataBase::**setPathFromFile**(QString const& path)

{

if(!path.isEmpty())

{

this->pathFromFile = path;

}

}

QString DataBase::**getPathFromFile**()

{

return this->pathFromFile;

}

QString DataBase::**getInformation**()

{

QString result;

for(QList<Audio\*>::iterator it = this->listAudio->begin(); it != this->listAudio->end(); it++)

{

Audio \* audio = \*it;

result += audio->*getInformation*() + "\n";

}

return result;

}

void DataBase::**updateDataBaseFromFile**()

{

QFile file(this->pathFromFile);

if(file.*open*(QIODevice::WriteOnly))

{

this->sort(6,1);

QString allText = this->getInformation();

QByteArray text = allText.toUtf8();

file.write(text);

}

file.*close*();

}

int DataBase::**addAudio**(Audio \* audio)

{

if(!this->listAudio->empty())

{

if(this->findAudio(audio) == false)

{

this->listAudio->append(audio);

this->updateDataBaseFromFile();

}

else {

return -1;

}

}

else {

this->listAudio->append(audio);

this->updateDataBaseFromFile();

}

return 0;

}

void DataBase::**deleteAudio**(Audio \* audio)

{

if(!this->listAudio->empty())

{

if(this->listAudio->indexOf(audio) >= 0)

{

this->listAudio->removeOne(audio);

this->updateDataBaseFromFile();

}

}

}

QList<Audio\*>\* DataBase::**getListAudio**()

{

return this->listAudio;

}

void DataBase::**setDataBaseFromFile**()

{

QString text;

QFile file(this->pathFromFile);

if(file.exists())

{

if(file.*open*(QIODevice::ReadOnly))

{

QByteArray line = file.readAll();

text = line;

}

file.*close*();

}

QStringList line = text.split("\n");

for(int i = 0; i < line.length(); i++)

{

Audio \* audio;

QStringList textAudio = line[i].split("|");

if(!textAudio[0].isEmpty())

{

if(QFileInfo::exists(textAudio[2]))

{

if(textAudio[0] == "Музыка")

{

audio = new Music(textAudio[1], textAudio[2], QDate::fromString(textAudio[3],"dd.MM.yyyy"), textAudio[4], textAudio[5],

textAudio[6], QTime::fromString(textAudio[7],"hh:mm:ss"), textAudio[8]);

this->listAudio->append(audio);

}

if(textAudio[0] == "Аудиокнига")

{

audio = new AudioBook(textAudio[1], textAudio[2], QDate::fromString(textAudio[3],"dd.MM.yyyy"), textAudio[4], textAudio[5],

textAudio[6], QTime::fromString(textAudio[7],"hh:mm:ss"), textAudio[8]);

this->listAudio->append(audio);

}

}

}

}

}

void DataBase::**editAudio**(Audio \* audio, Audio \* editAudio)

{

if(!this->listAudio->empty())

{

int index = this->listAudio->indexOf(audio);

if(index >= 0)

{

this->listAudio->replace(index, editAudio);

this->updateDataBaseFromFile();

}

}

}

QList<Audio\*>\* DataBase::**searchString**(QString inputData)

{

QRegExp date("(0[1-9]|(([1]|[2])[0-9])|3[0-1])[- /.](0[1-9]|1[0-2])[- /.](20(19|20))");

QRegExp time("^([0-1][0-9]|2[0-3])(:[0-5][0-9]){2}$");

QList<Audio\*>\* searched = new QList<Audio\*>();

for(QList<Audio\*>::iterator it = this->listAudio->begin(); it != this->listAudio->end(); it++)

{

Audio \* audio = \*it;

if(date.exactMatch(inputData) == true)

{

QDate d = QDate::fromString(inputData, "dd.MM.yyyy");

int y = audio->getDateAdded().year();

int m = audio->getDateAdded().month();

int day = audio->getDateAdded().day();

int y1 = d.year();

int m1 = d.month();

int day1 = d.day();

if((m == m1) && (y == y1) && ((day1 - 5) <= day && day <= (day1 + 5)))

{

searched->append(audio);

}

}

if(time.exactMatch(inputData) == true)

{

QTime t = QTime::fromString(inputData);

int h = audio->getDuration().hour();

int m = audio->getDuration().minute();

int h1 = t.hour();

int m1 = t.minute();

if((h == h1) && ((m1 - 2) <= m && m <= (m1 + 5)))

{

searched->append(audio);

}

}

if(time.exactMatch(inputData) == false && date.exactMatch(inputData) == false && audio->*getInformation*().toLower().contains(inputData.toLower()))

{

searched->append(audio);

}

}

return searched;

}

void DataBase::**sort**(int index, int status)

{

if(status == 1)

{

if(index == 0)

{

qSort(this->listAudio->begin(), this->listAudio->end(),[](Audio\* a, Audio\* b) { return a->getName() < b->getName(); });

}

if(index == 1)

{

qSort(this->listAudio->begin(), this->listAudio->end(),[](Audio\* a, Audio\* b) { return a->getPerformer() < b->getPerformer(); });

}

if(index == 5)

{

qSort(this->listAudio->begin(), this->listAudio->end(),[](Audio\* a, Audio\* b) { return a->getDuration() < b->getDuration(); });

}

if(index == 6)

{

qSort(this->listAudio->begin(), this->listAudio->end(),[](Audio\* a, Audio\* b) { return a->getDateAdded() < b->getDateAdded(); });

}

}

else

{

if(index == 0)

{

qSort(this->listAudio->begin(), this->listAudio->end(),[](Audio\* a, Audio\* b) { return a->getName() > b->getName(); });

}

if(index == 1)

{

qSort(this->listAudio->begin(), this->listAudio->end(),[](Audio\* a, Audio\* b) { return a->getPerformer() > b->getPerformer(); });

}

if(index == 5)

{

qSort(this->listAudio->begin(), this->listAudio->end(),[](Audio\* a, Audio\* b) { return a->getDuration() > b->getDuration(); });

}

if(index == 6)

{

qSort(this->listAudio->begin(), this->listAudio->end(),[](Audio\* a, Audio\* b) { return a->getDateAdded() > b->getDateAdded(); });

}

}

}

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "audioform.h"

#include "database.h"

#include <QStandardItemModel>

#include <QStandardItem>

#include <QMediaPlayer>

#include <QMediaPlaylist>

#include <QDesktopServices>

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

private:

DataBase \*dataBase;

QMediaPlayer \*player;

QMediaPlaylist \*playlist;

int statusVolume = 1;

int statusSort = -1;

QMessageBox warning;

QModelIndex currentIndex;

QString currentName;

public:

explicit **MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

void **setTableData**(QList<Audio\*> \* currentList);

void **setCurrentName**();

void **setCurrentDuration**();

void **sort**(int index);

virtual void ***resizeEvent***(QResizeEvent \*event) override;

private slots:

void **on\_addAudio**();

void **on\_editAudio\_triggered**();

void **on\_deleteAudio\_triggered**();

void **switching**(int index);

void **on\_previous**();

void **on\_next**();

void **on\_play**();

void **on\_pause**();

void **on\_stop**();

void **on\_listAudio\_doubleClicked**(const QModelIndex &index);

void **on\_slider\_volume\_valueChanged**(int value);

void **on\_btn\_volume\_clicked**();

void **positionAudio**(qint64);

void **on\_slider\_durationAudio\_sliderMoved**(int position);

void **on\_listAudio\_clicked**(const QModelIndex &index);

void **sortByColumn**(int indexColumn);

void **on\_searchAudio\_triggered**();

void **on\_btn\_search\_clicked**();

void **on\_sortName\_triggered**();

void **on\_sortPerformer\_triggered**();

void **on\_sortDuration\_triggered**();

void **on\_sortDateAdded\_triggered**();

void **openHelp**();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

this->setWindowIcon(QIcon(":/resource/image/windowIcon.png"));

warning.setWindowIcon(this->windowIcon());

dataBase = new DataBase(QString(QDir::currentPath() + "/DataBase.txt"));

ui->listAudio->setEditTriggers(QAbstractItemView::NoEditTriggers);

ui->listAudio->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->listAudio->setSelectionMode(QAbstractItemView::SingleSelection);

player = new QMediaPlayer(this);

playlist = new QMediaPlaylist(player);

player->setPlaylist(playlist);

player->setVolume(30);

playlist->setPlaybackMode(QMediaPlaylist::Loop);

this->setTableData(dataBase->getListAudio());

connect(ui->addAudio, SIGNAL(triggered()), this, SLOT(on\_addAudio()));

connect(ui->btn\_add, SIGNAL(clicked()), this, SLOT(on\_addAudio()));

connect(ui->btn\_delete, SIGNAL(clicked()), this, SLOT(on\_deleteAudio\_triggered()));

connect(ui->btn\_previous, SIGNAL(clicked()), this, SLOT(on\_previous()));

connect(ui->previousAudio, SIGNAL(triggered()), this, SLOT(on\_previous()));

connect(ui->btn\_next, SIGNAL(clicked()), this, SLOT(on\_next()));

connect(ui->nextAudio, SIGNAL(triggered()), this, SLOT(on\_next()));

connect(ui->btn\_play, SIGNAL(clicked()), this, SLOT(on\_play()));

connect(ui->playAudio, SIGNAL(triggered()), this, SLOT(on\_play()));

connect(ui->btn\_pause, SIGNAL(clicked()), this, SLOT(on\_pause()));

connect(ui->pauseAudio, SIGNAL(triggered()), this, SLOT(on\_pause()));

connect(ui->btn\_stop, SIGNAL(clicked()), this, SLOT(on\_stop()));

connect(ui->stopAudio, SIGNAL(triggered()), this, SLOT(on\_stop()));

connect(ui->listAudio->horizontalHeader(), SIGNAL(sectionClicked(int)), this, SLOT(sortByColumn(int)));

connect(ui->btn\_reset, SIGNAL(clicked()), this, SLOT(on\_searchAudio\_triggered()));

connect(this->player, SIGNAL(positionChanged(qint64)), this, SLOT(positionAudio(qint64)));

connect(this->playlist, SIGNAL(currentIndexChanged(int)), this, SLOT(switching(int)));

connect(ui->helper, SIGNAL(triggered()), this, SLOT(openHelp()));

}

void MainWindow::**setTableData**(QList<Audio\*> \*currentList)

{

playlist->clear();

QStandardItemModel \*model = new QStandardItemModel;

QStandardItem \*item;

QStringList horizontalHeader;

horizontalHeader.append("Наименование");

horizontalHeader.append("Исполнитель");

horizontalHeader.append("Жанр");

horizontalHeader.append("Альбом/Автор");

horizontalHeader.append("Язык");

horizontalHeader.append("Время");

horizontalHeader.append("Дата добавления");

model->setHorizontalHeaderLabels(horizontalHeader);

int i = 0;

for(QList<Audio\*>::iterator it = currentList->begin(); it != currentList->end(); it++)

{

Audio \* audio = \*it;

playlist->addMedia(QUrl(audio->getPath()));

item = new QStandardItem(audio->getName());

model->setItem(i,0,item);

item = new QStandardItem(audio->getPerformer());

model->setItem(i,1,item);

item = new QStandardItem(audio->getGenre());

model->setItem(i,2,item);

item = new QStandardItem(audio->*getFeature*());

model->setItem(i,3,item);

item = new QStandardItem(audio->getLanguage());

model->setItem(i,4,item);

item = new QStandardItem(audio->getDuration().toString("hh:mm:ss"));

model->setItem(i,5,item);

item = new QStandardItem(audio->getDateAdded().toString("dd.MM.yyyy"));

model->setItem(i,6,item);

i++;

}

ui->listAudio->*setModel*(model);

ui->listAudio->resizeRowsToContents();

ui->listAudio->resizeColumnsToContents();

int width = this->width() - 35;

ui->listAudio->setColumnWidth(0, width/7);

ui->listAudio->setColumnWidth(1, width/7);

ui->listAudio->setColumnWidth(2, width/7);

ui->listAudio->setColumnWidth(3, width/7);

ui->listAudio->setColumnWidth(4, width/7);

ui->listAudio->setColumnWidth(5, width/7);

ui->listAudio->setColumnWidth(6, width/7);

}

void MainWindow::**setCurrentName**()

{

this->currentName = ui->listAudio->model()->*data*(ui->listAudio->model()->*index*(ui->listAudio->currentIndex().row(), 0)).toString();

ui->currentAudio->setText(this->currentName);

}

void MainWindow::**setCurrentDuration**()

{

QString time = ui->listAudio->model()->*data*(ui->listAudio->model()->*index*(ui->listAudio->currentIndex().row(), 5)).toString();

QTime timeBegin = QTime(0,0);

QTime timeDuration = QTime::fromString(time);

int duration = timeBegin.msecsTo(timeDuration);

ui->slider\_durationAudio->setMaximum(duration);

}

void MainWindow::**sort**(int index)

{

this->statusSort \*= -1;

this->dataBase->sort(index, this->statusSort);

this->setTableData(this->dataBase->getListAudio());

}

void MainWindow::***resizeEvent***(QResizeEvent \*event)

{

int width = event->size().width() - 35;

ui->listAudio->setColumnWidth(0, width/7);

ui->listAudio->setColumnWidth(1, width/7);

ui->listAudio->setColumnWidth(2, width/7);

ui->listAudio->setColumnWidth(3, width/7);

ui->listAudio->setColumnWidth(4, width/7);

ui->listAudio->setColumnWidth(5, width/7);

ui->listAudio->setColumnWidth(6, width/7);

}

MainWindow::~***MainWindow***()

{

delete ui;

delete playlist;

delete player;

delete dataBase;

}

void MainWindow::**on\_addAudio**()

{

AudioForm \* form = new AudioForm(this);

form->setModal(true);

if(form->*exec*() == QDialog::Accepted)

{

if(dataBase->addAudio(form->audio) == 0)

{

this->setTableData(dataBase->getListAudio());

warning.setWindowTitle(this->windowTitle());

warning.setText("Аудио добавлено.");

warning.setIcon(QMessageBox::Information);

warning.*exec*();

}

else

{

this->setTableData(dataBase->getListAudio());

warning.setWindowTitle(this->windowTitle());

warning.setText("Такая аудиозапись уже существует.");

warning.setIcon(QMessageBox::Warning);

warning.*exec*();

}

}

}

void MainWindow::**on\_editAudio\_triggered**()

{

QModelIndex indexTable = ui->listAudio->currentIndex();

if(indexTable.row() < 0)

{

warning.setWindowTitle(this->windowTitle());

warning.setText("Аудио не выбрано.");

warning.setIcon(QMessageBox::Warning);

warning.*exec*();

}

else

{

QList<Audio\*>\* list = dataBase->getListAudio();

for(QList<Audio\*>::iterator it = list->begin(); it != list->end(); it++)

{

Audio \* audio = \*it;

if(audio->getName() == this->currentName)

{

AudioForm \* form = new AudioForm(audio,this);

form->setModal(true);

if(form->*exec*() == QDialog::Accepted)

{

this->dataBase->editAudio(audio, form->audio);

this->setTableData(dataBase->getListAudio());

warning.setWindowTitle(this->windowTitle());

warning.setText("Аудио изменено.");

warning.setIcon(QMessageBox::Information);

warning.*exec*();

return;

}

}

}

}

}

void MainWindow::**on\_deleteAudio\_triggered**()

{

QModelIndex indexTable = ui->listAudio->currentIndex();

if(indexTable.row() < 0)

{

warning.setWindowTitle(this->windowTitle());

warning.setText("Аудио не выбрано.");

warning.setIcon(QMessageBox::Warning);

warning.*exec*();

}

else

{

QMessageBox deleteMessage;

deleteMessage.setWindowIcon(this->windowIcon());

deleteMessage.setWindowTitle(this->windowTitle());

deleteMessage.setText("Вы действительно хотите удалить это аудио?");

deleteMessage.setIcon(QMessageBox:: Question);

deleteMessage.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

deleteMessage.setDefaultButton(QMessageBox::No);

if(deleteMessage.*exec*() == QMessageBox::Yes)

{

QList<Audio\*>\* list = dataBase->getListAudio();

for(QList<Audio\*>::iterator it = list->begin(); it != list->end(); it++)

{

Audio \* audio = \*it;

if(audio->getName() == this->currentName)

{

this->dataBase->deleteAudio(audio);

this->setTableData(dataBase->getListAudio());

warning.setWindowTitle(this->windowTitle());

warning.setText("Аудио удалено.");

warning.setIcon(QMessageBox::Information);

warning.*exec*();

return;

}

}

}

}

}

void MainWindow::**on\_previous**()

{

this->playlist->previous();

this->setCurrentDuration();

this->setCurrentName();

}

void MainWindow::**on\_next**()

{

this->playlist->next();

this->setCurrentDuration();

this->setCurrentName();

}

void MainWindow::**on\_play**()

{

if(ui->listAudio->currentIndex().row() == -1)

{

ui->listAudio->selectRow(0);

}

this->playlist->setCurrentIndex(ui->listAudio->currentIndex().row());

this->setCurrentDuration();

this->player->play();

this->setCurrentName();

}

void MainWindow::**on\_pause**()

{

this->player->pause();

}

void MainWindow::**on\_stop**()

{

this->player->stop();

ui->currentAudio->clear();

}

void MainWindow::**on\_listAudio\_doubleClicked**(const QModelIndex &index)

{

this->switching(index.row());

this->on\_play();

}

void MainWindow::**switching**(int index)

{

ui->listAudio->selectRow(index);

}

void MainWindow::**on\_slider\_volume\_valueChanged**(int value)

{

player->setVolume(value);

}

void MainWindow::**on\_btn\_volume\_clicked**()

{

this->statusVolume \*= -1;

if(this->statusVolume == -1)

{

player->setVolume(0);

ui->slider\_volume->setValue(0);

}

else

{

player->setVolume(30);

ui->slider\_volume->setValue(30);

}

}

void MainWindow::**positionAudio**(qint64 value)

{

ui->slider\_durationAudio->setValue(int(value));

}

void MainWindow::**on\_slider\_durationAudio\_sliderMoved**(int position)

{

this->player->setPosition(position);

}

void MainWindow::**on\_listAudio\_clicked**(const QModelIndex &index)

{

this->switching(index.row());

this->setCurrentName();

}

void MainWindow::**sortByColumn**(int indexColumn)

{

this->sort(indexColumn);

}

void MainWindow::**on\_searchAudio\_triggered**()

{

ui->editSearch->clear();

this->setTableData(this->dataBase->getListAudio());

}

void MainWindow::**on\_btn\_search\_clicked**()

{

this->setTableData(this->dataBase->searchString(ui->editSearch->text()));

}

void MainWindow::**on\_sortName\_triggered**()

{

this->sort(0);

}

void MainWindow::**on\_sortPerformer\_triggered**()

{

this->sort(1);

}

void MainWindow::**on\_sortDuration\_triggered**()

{

this->sort(5);

}

void MainWindow::**on\_sortDateAdded\_triggered**()

{

this->sort(6);

}

void MainWindow::**openHelp**()

{

QDesktopServices::openUrl(QUrl(QDir::currentPath() + "/help/Audioplayer.chm"));

}

#ifndef AUDIOFORM\_H

#define AUDIOFORM\_H

#include <QDialog>

#include <QAbstractButton>

#include "audiobook.h"

#include "music.h"

#include <QStandardItemModel>

#include <QStandardItem>

#include <QFileDialog>

#include <QMessageBox>

#include <QMediaPlayer>

namespace **Ui** {

class **AudioForm**;

}

class **AudioForm** : public QDialog

{

Q\_OBJECT

private:

int duration = 0;

QMediaPlayer \* player;

QTime durationEditAudio;

public:

Audio \* audio;

explicit **AudioForm**(QWidget \*parent = nullptr);

explicit **AudioForm**(Audio \* audio, QWidget \*parent = nullptr);

bool **validated**();

bool **validatedFormatFile**();

bool **audiofileExist**();

~***AudioForm***();

private slots:

void **on\_comboBoxCategory\_currentIndexChanged**(int index);

void **on\_lineEditPath\_editingFinished**();

void **changedStatus**(QMediaPlayer::MediaStatus status);

void **on\_btn\_ok\_clicked**();

void **on\_btn\_cancel\_clicked**();

void **on\_btn\_path\_clicked**();

private:

Ui::AudioForm \*ui;

};

#endif // AUDIOFORM\_H

#include "audioform.h"

#include "ui\_audioform.h"

QStringList **setGenre**(QString category)

{

QStringList genre = QStringList();

if(category == "Музыкальная запись")

{

genre << "Блюз" << "Рок" << "Джаз" << "Реп" << "Кантри" << "Классическая музыка" << "Мюзикл" << "Народная песня" << "Опера"

<< "Панк-рок" << "Поп-музыка" << "Рок-н-ролл" << "Хип-хоп" << "Шансон";

}

else

{

genre << "Фантастика" << "Роман, проза" << "Детективы, триллеры" << "Классика" << "Психология, философия" << "Ужасы, мистика"

<< "Аудиоспектакли" << "Приключения" << "История" << "Эзотерика" << "Бизнес" << "Научно-популярное" << "Детские аудиокниги"

<< "Юмор, сатира" << "Обучение" << "Ранобэ" << "Биографии, мемуары" << "На иностранных языках" << "Поэзия" << "Религия"

<< "Путешествия" << "Разное";

}

return genre;

}

QTime **setTime**(int ms)

{

QTime \*time = new QTime(0,0);

return time->addMSecs(ms);

}

AudioForm::**AudioForm**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::AudioForm)

{

ui->setupUi(this);

this->duration = 0;

this->player = new QMediaPlayer(this);

this->setWindowTitle("Добавить аудио");

ui->comboBoxGenre->addItems(setGenre(ui->comboBoxCategory->currentText()));

connect(this->player, SIGNAL(mediaStatusChanged(QMediaPlayer::MediaStatus)), this, SLOT(changedStatus(QMediaPlayer::MediaStatus)));

}

AudioForm::**AudioForm**(Audio \* audio, QWidget \*parent) :

QDialog(parent),

ui(new Ui::AudioForm)

{

ui->setupUi(this);

this->audio = audio;

this->setWindowTitle("Изменить аудио");

durationEditAudio = audio->getDuration();

ui->comboBoxGenre->addItems(setGenre(audio->*getCategory*()));

ui->comboBoxCategory->setCurrentText(audio->*getCategory*());

ui->comboBoxGenre->setCurrentText(audio->getGenre());

ui->lineEditName->setText(audio->getName());

ui->lineEditPath->setText(audio->getPath());

ui->lineEditPerformer->setText(audio->getPerformer());

ui->comboBoxLenguage->setCurrentText(audio->getLanguage());

ui->lineEditAlbum->setText(audio->*getFeature*());

if(audio->*getCategory*() == "Музыкальная запись")

{

ui->labelAlbum->setText("Альбом:");

}

else

{

ui->labelAlbum->setText("Автор:");

}

}

bool AudioForm::**validated**()

{

if(ui->lineEditName->text().isEmpty() || ui->lineEditAlbum->text().isEmpty() || ui->lineEditPerformer->text().isEmpty()

|| ui->comboBoxLenguage->currentText().isEmpty() || ui->lineEditPath->text().isEmpty())

{

return false;

}

return true;

}

bool AudioForm::**validatedFormatFile**()

{

QFileInfo audioFile(ui->lineEditPath->text());

if(audioFile.suffix() == "mp3")

{

return true;

}

return false;

}

bool AudioForm::**audiofileExist**()

{

if(QFileInfo::exists(ui->lineEditPath->text()))

{

return true;

}

return false;

}

AudioForm::~***AudioForm***()

{

delete ui;

delete player;

}

void AudioForm::**on\_comboBoxCategory\_currentIndexChanged**(int index)

{

ui->comboBoxGenre->clear();

if(index == 0)

{

ui->labelAlbum->setText("Альбом:");

}

if(index == 1)

{

ui->labelAlbum->setText("Автор:");

}

ui->comboBoxGenre->addItems(setGenre(ui->comboBoxCategory->currentText()));

}

void AudioForm::**on\_lineEditPath\_editingFinished**()

{

if(ui->lineEditPath->text().isEmpty() == false)

{

QMessageBox warning;

if(this->audiofileExist() == false)

{

warning.setWindowTitle(this->windowTitle());

warning.setText("Такого файла не существует!");

warning.setIcon(QMessageBox::Critical);

warning.*exec*();

ui->lineEditPath->setFocus();

}

else

{

if(this->validatedFormatFile() == false)

{

warning.setWindowTitle(this->windowTitle());

warning.setText("Формат файла не язвляется форматом аудиофайла!");

warning.setIcon(QMessageBox::Critical);

warning.*exec*();

ui->lineEditPath->setFocus();

}

else

{

player->setMedia(QUrl(ui->lineEditPath->text()));

}

}

}

}

void AudioForm::**changedStatus**(QMediaPlayer::MediaStatus status)

{

if (status == QMediaPlayer::LoadedMedia)

{

this->duration = int(player->duration());

}

}

void AudioForm::**on\_btn\_ok\_clicked**()

{

if(this->validated() == true)

{

this->setResult(QDialog::Accepted);

QTime duration;

QDate dateAdded;

if(this->windowTitle() == "Добавить аудио")

{

duration = setTime(this->duration);

dateAdded = QDate::currentDate();

}

else

{

duration = this->durationEditAudio;

dateAdded = this->audio->getDateAdded();

}

if(ui->comboBoxCategory->currentIndex() == 0)

{

audio = new Music(ui->lineEditName->text(), ui->lineEditPath->text(), dateAdded,

ui->comboBoxLenguage->currentText(), ui->lineEditPerformer->text(), ui->comboBoxGenre->currentText(),

duration, ui->lineEditAlbum->text());

}

if(ui->comboBoxCategory->currentIndex() == 1)

{

audio = new AudioBook(ui->lineEditName->text(), ui->lineEditPath->text(), dateAdded,

ui->comboBoxLenguage->currentText(), ui->lineEditPerformer->text(), ui->comboBoxGenre->currentText(),

duration, ui->lineEditAlbum->text());

}

this->*done*(1);

}

else

{

QMessageBox warning;

warning.setWindowTitle(this->windowTitle());

warning.setText("Заполнены не все поля, либо поля введены некорректно!");

warning.setIcon(QMessageBox::Warning);

warning.*exec*();

}

}

void AudioForm::**on\_btn\_cancel\_clicked**()

{

this->close();

}

void AudioForm::**on\_btn\_path\_clicked**()

{

QString pathFolder = QDir::currentPath() + "/playlist";

if(QFileInfo::exists(pathFolder) == false)

{

QDir().mkdir(pathFolder);

}

QString path = QFileDialog::getOpenFileName(this, "Выберите аудиофайл", pathFolder, "\*.mp3");

if(path.isEmpty() == false)

{

ui->lineEditPath->setText(path);

player->setMedia(QUrl(ui->lineEditPath->text()));

}

}

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}