**Содержание**

[Введение 4](#_Toc10465698)

[Часть 1 – Теоретическое введение 5](#_Toc10465699)

[1.1 Объектно-ориентированное программирование 5](#_Toc10465700)

[1.2 UML 5](#_Toc10465701)

[1.3 Создание интерфейса 6](#_Toc10465702)

[Часть 2 – Проектная часть 10](#_Toc10465703)

[2.1 Постановка задачи 10](#_Toc10465704)

[2.2 Проектное решение 12](#_Toc10465705)

[2.3 Проектирование интерфейса 14](#_Toc10465706)

[Часть 3 – Экспериментальная часть 19](#_Toc10465707)

[3.1 Тестирование 19](#_Toc10465708)

[3.2 Руководство пользователя 26](#_Toc10465709)

[Заключение 29](#_Toc10465710)

[Список литературы 30](#_Toc10465711)

[Приложение 31](#_Toc10465712)

# **Введение**

В данной курсовой работе представлена реализация системы подбора кадров для организации с использованием объектно-ориентированного подхода. Программа реализована на языке программирования C++ с использованием кроссплатформенной среды разработки Qt Creator и фреймворка Qt. Любому предприятию чтобы начать свою работу необходимо нанять сотрудников. И для каждой должности необходимо находить лучших и наиболее подходящих для этого людей из множества претендентов. Поэтому для этой задачи необходима система, которая будет хранить данные о вакансиях организации, и соответственно, данные о кандидатах, претендующих на соответствующие вакансии. Интерфейс системы позволяет добавлять в систему вакансии организации, добавлять кандидатов на вакансии, организовывать переподготовку кандидатов по востребованным специальностям, принимать или отклонять кандидатов, а также сохранять данные о всех кандидатах в архив.

# **Часть 1 – Теоретическое введение**

## **1.1 Объектно-ориентированное программирование**

**Объектно-ориентированное программирование** – это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса, а классы образуют иерархию на принципах наследуемости.

Основные [принципы](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF) структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

* [*абстрагирование*](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте — контекстное понимание предмета, формализуемое в виде класса;
* [*инкапсуляция*](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления;
* [*наследование*](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтённое на предыдущих шагах;
* [*полиморфизм*](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.

## **1.2 UML**

**UML** – унифицированный язык моделирования (Unified Modeling Language) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования.  
Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

Словарь UML включает три вида строительных блоков:

* Диаграммы.
* Сущности.
* Связи.

***Сущности*** – это абстракции, которые являются основными элементами модели, ***связи*** соединяют их между собой, а ***диаграммы*** группируют представляющие интерес наборы сущностей.  
  
***Диаграмма*** – это графическое представление набора элементов, чаще всего изображенного в виде связного графа вершин (сущностей) и путей (связей). Язык UML включает **13** видов диаграмм, среди которых на первом месте в списке — диаграмма классов, о которой и пойдет речь.  
Диаграммы классов показывают набор классов, интерфейсов, а также их связи. Диаграммы этого вида чаще всего используются для моделирования объектно-ориентированных систем. Они предназначены для статического представления системы.  
Большинство элементов UML имеют уникальную и прямую графическую нотацию, которая дает визуальное представление наиболее важных аспектов элемента.

## **1.3 Создание интерфейса**

Интерфейс имеет важное значение для любой программной системы и является неотъемлемой ее составляющей, ориентированной, прежде всего, на конечного пользователя. Именно через интерфейс пользователь судит о прикладной программе в целом; более того, часто решение об использовании прикладной программы пользователь принимает по тому, насколько ему удобен и понятен пользовательский интерфейс. Вместе с тем, трудоемкость проектирования и разработки интерфейса достаточно велика. По оценкам специалистов в среднем она составляет более половины времени реализации проекта. Актуальным является снижение затрат на разработку и сопровождение программных систем или разработка эффективного программного инструментария.   
Одним из путей снижения затрат на разработку и сопровождение программных систем является наличие в инструментарии средств четвертого поколения, позволяющих на высоком уровне описать (специфицировать) создаваемое программное средство и далее по спецификации автоматически сгенерировать исполнимый код.   
В литературе не существует единой общепринятой классификации средств для разработки пользовательского интерфейса. Так, программное обеспечение для разработки пользовательского интерфейса можно разделить на две основные группы - инструментарий для разработки пользовательского интерфейса (toolkits) и высокоуровневые средства разработки интерфейса (higher level development tools). Инструментарий для разработки пользовательского интерфейса, как правило, включает в себя библиотеку примитивов компонентов интерфейса (меню, кнопки, полосы прокрутки и др.) и предназначен для использования программистами. Высокоуровневые средства разработки интерфейса могут быть использованы непрограммистами и снабжены языком, который позволяет специфицировать функции ввода-вывода, а также определять, используя технику непосредственного манипулирования, интерфейсные элементы. К таким средствам относятся построители диалога (interface builders) и СУПИ - системы управления пользовательским интерфейсом (User Interface Management Systems - UIMS). Помимо СУПИ, некоторые авторы используют такие термины, как User Interface Development Systems (UIDS) - системы разработки пользовательского интерфейса, User Interface Design Environment (UIDE) - среда разработки пользовательского интерфейса и др.   
Специализированные средства для разработки интерфейса позволяют упростить разработку пользовательского интерфейса, предлагая разработчику специфицировать компоненты пользовательского интерфейса с использованием языков спецификаций. Можно выделить несколько основных способов спецификации интерфейса:

* 1. Языковой, когда применяются специальные языки для задания синтаксиса интерфейса (декларативные, объектно-ориентированные, языки событий и др.).
  2. Графическая спецификация связана с определением интерфейса, как правило, средствами визуального программирования, программированием демонстраций и по примерам. Подобный способ поддерживает ограниченный класс интерфейсов.
  3. Спецификация интерфейса, основанная на объектно-ориентированном подходе, связана с принципом, называемым непосредственное манипулирование. Основное его свойство - взаимодействие пользователя с индивидуальными объектами, а не со всей системой как единым целым. Типичными компонентами, используемыми для манипуляций с объектами и управляющими функциями, являются обработчики, меню, зоны диалога, кнопки различного вида.
  4. Спецификация интерфейса по спецификации прикладной задачи. Здесь интерфейс создается автоматически по спецификации семантики прикладной задачи. Однако сложность описания интерфейса затрудняет возможности скорого появления систем, реализующих данный подход.

Основной концепцией СУПИ является отделение разработки пользовательского интерфейса от остального приложения. В настоящее время идея раздельного проектирования интерфейса и приложения либо закреплена в определении СУПИ либо является основным его свойством.

В состав СУПИ определен как набор инструментов этапа разработки и периода исполнения. Инструменты этапа разработки оперируют с моделями интерфейса для построения их проектов. Они могут разделяться на две группы: интерактивные инструменты, например редакторы моделей, и автоматические инструменты, например генератор форм. Инструменты периода исполнения используют модель интерфейса для поддержки деятельности пользователя, например, для сбора и анализа используемых данных.

Функциями СУПИ является содействие и облегчение разработки и сопровождения пользовательского интерфейса, а также управление взаимодействием между пользователем и прикладной программой.

Таким образом, в настоящее время существует большое количество инструментальных средств для разработки интерфейса, поддерживающих различные методы его реализации.

# **Часть 2 – Проектная часть**

## **2.1 Постановка задачи**

Цель данной курсовой работы – создание системы подбора кадров. Система разрабатывается для начальника организации, или же для работника отдела кадров для удобного сравнения и подбора лучших работников на должности. Пользователь добавляет в систему вакансии, в которых нуждается организация, и кандидатов на эти вакансии. Сведения о вакансии и о претенденте на вакансию заполняются пользователем вручную через диалоговое окно. Претендент также может отправить своё резюме в строгом формате в файле .txt на служебную почту организации, чтобы работник организации смог быстро внести сведения о претенденте на должность в систему. Пользователь может удалить вакансию или кандидата, принять или отклонить кандидата на должность, организовать переподготовку по востребованной специальности, уведомить конкретного кандидата о переподготовке посредством отправки автоматической рассылки на почту и по СМС, которые кандидат указал в резюме. Принятые и отклонённые кандидаты переносятся в архив. Данные из архива могут быть удалены по мере необходимости и быть использованы для внесения принятых кандидатов в систему менеджмента сотрудниками.

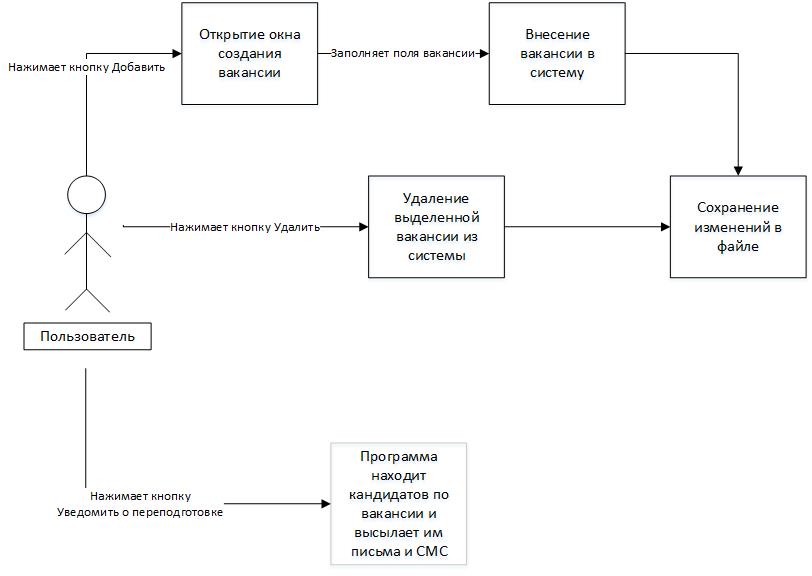


Рисунок 1. Схема взаимодействия с программой. Работа с вакансиями

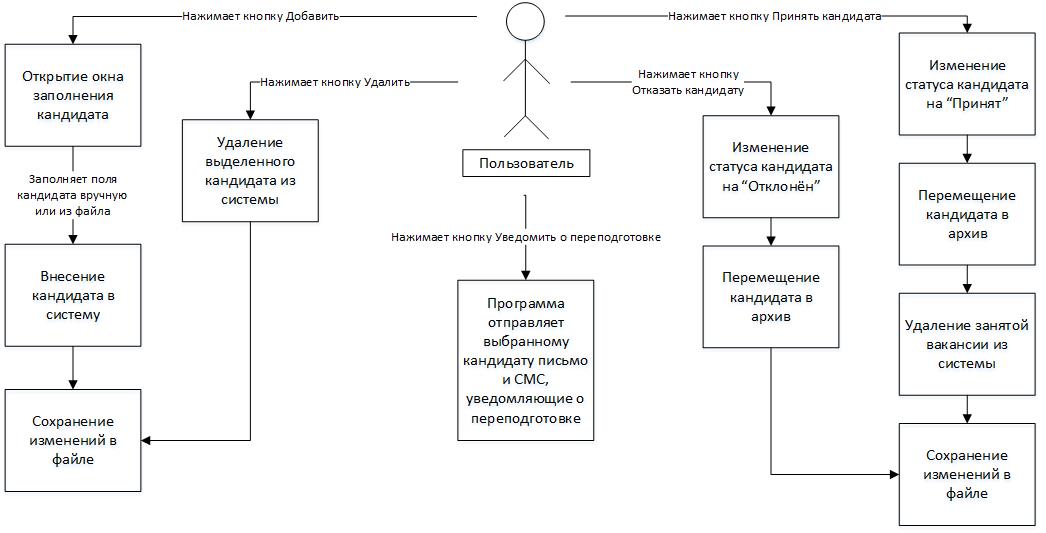


Рисунок 2. Схема взаимодействия с программой. Работа с кандидатами

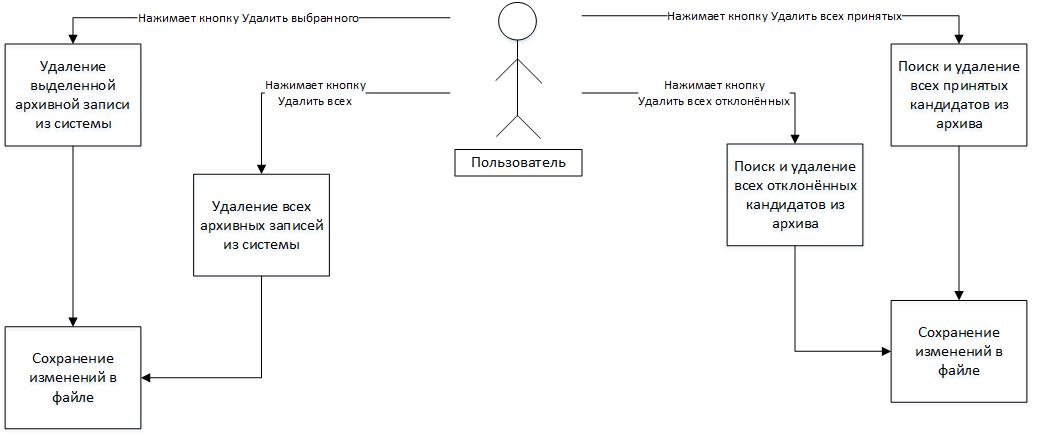


Рисунок 3. Схема взаимодействия с программой. Работа с архивом

## **2.2 Проектное решение**

В данной системе действующих лиц несколько: пользователь программы, непосредственно работающий с ней, и люди, присылающие резюме для приёма на работу. Согласно заданию, система должна хранить вакансии организации, поэтому очевидно, что вакансия должна быть реализована как отдельный класс Vacancy с параметрами соответствующей специальности: названием специальности и шифром. Необходимые методы для этого класса: конструктор, функции для получения значения поля.

Еще одним классом необходимо реализовать класс кандидата на должность. Этот класс должен наследовать два класса – Person, и описанный выше Vacancy. Класс Person является классом содержащим информацию о человеке: ФИО, дата рождения, пол, образование и его прошлая деятельность; соответственно, этот класс имеет такие методы как конструктор и методы для получения значения полей. Я решил сделать класс кандидата наследником классов Person и Vacancy чтобы продемонстрировать возможности множественного наследования классов в C++. Таким образом, наследуя два этих класса, образуется класс Applicant, с информацией о человеке от класса Person, и информацией о должности на которую он претендует от класса Vacancy. У этого класса есть собственные поля, характерные именно как претенденту на должность. Это номер телефона и адрес электронной почты для обратной связи, стаж работы, статус в системе и дата внесения кандидатуры в систему.

Для сохранения данных используются два файла “data.bin” и “archive.bin”. Данные сохраняются с помощью встроенных в Qt средств сериализации – QDataStream. В файле “data.bin” хранятся данные о текущих вакансиях и кандидатах. В файл “archive.bin” сохраняются данные о о принятых и отклонённых кандидатах. Эта информация может быть использована программой менеджмента сотрудников, для того чтобы внести принятых кандидатов в систему. Для корректной работы механизмов сериализации в программе реализованы перегрузки операторов потоков для классов Vacancy и Applicant.

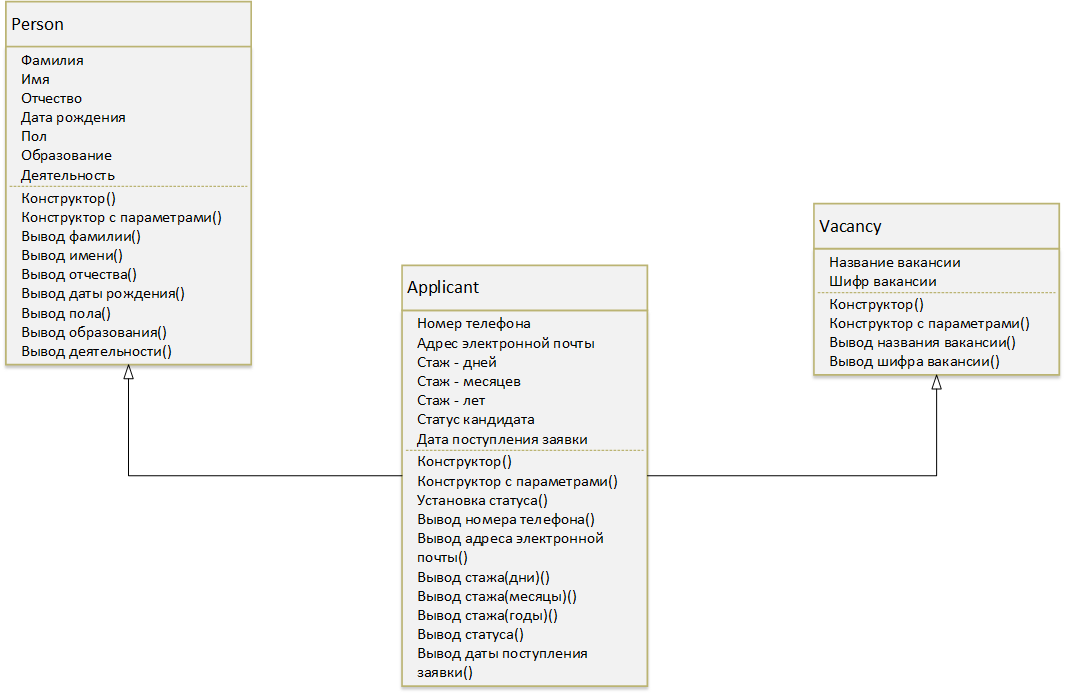
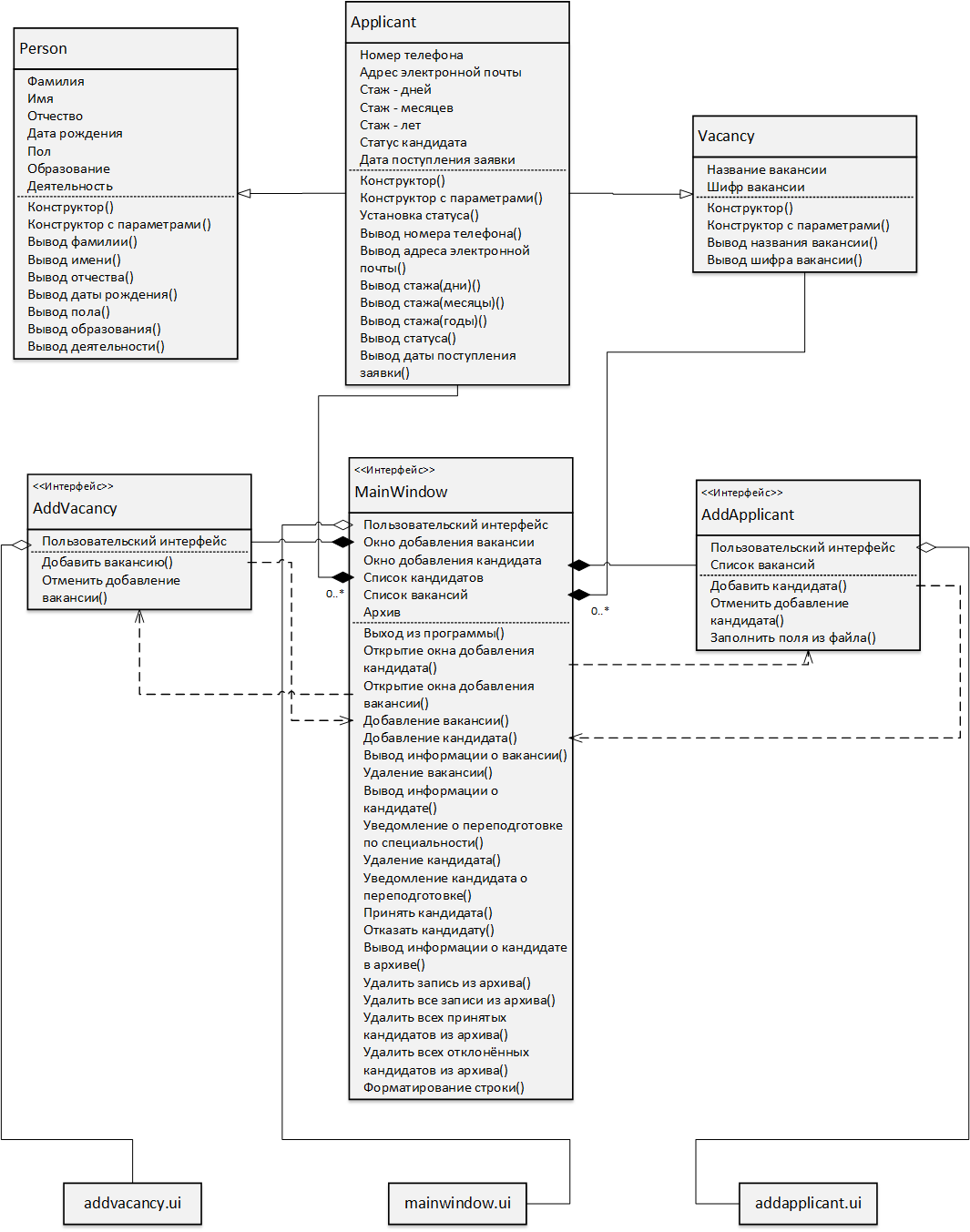


Рисунок 4. Диаграмма основных классов

## **2.3 Проектирование интерфейса**

Для создания графического интерфейса средствами фреймворка Qt нам так же понадобятся классы. Эти классы – главное окно приложения MainWindow и диалоговые окна AddVacancy и AddApplicant. MainWindow является наследником класса Qt QMainWindow, а диалоговые окна наследуют класс QDialog. Все эти классы имеют связанные с ними одноименные файлы интерфейса с расширением .ui, в которых описываются имеющиеся элементы интерфейса и их расположение. В классе MainWindow имеются экземпляры классов AddVacancy и AddApplicant и 3 структуры QMap для хранения информации; ключом в этих структурах является строка вида индификационный\_номер:[ название вакансии/ФИО кандидата], а значением является объект Vacancy или Applicant. Также он содержит методы-слоты, которые обрабатывают приходящие к экземпляру класса сигналы от элементов интерфейса, метод для перезаписи данных в файл, и один сигнал, посылающий данные в объект AddApplicant. Классы AddVacancy и AddApplicant имеют по два слота обработки сигналов диалогового окна и один сигнал для отправки данных в MainWindow. Но у AddApplicant имеются еще 2 слота для получения данных из главного окна и из файла.

В Qt элементы интерфейса называются виджетами. Всю площадь главного окна занимает виджет панели вкладок, содержащий 3 вкладки: “Кандидаты”, “Вакансии” и “Архив”. В каждой вкладке имеется виджет Список элементов(List Widget) для отображения записей и виджет Метка(Label) для вывода информации о выделенной записи. Внизу расположены кнопки(PushButton) действий, количество и значение которых во вкладках различается. В окнах добавления вакансий и кандидатов в два ряда расположены элементы: слева – метки(Label) названия полей; справа – различные элементы для внесения информации(LineEdit, DateEdit, ComboBox, SpinBox). Внизу расположен виджет диалоговых кнопок(DialogButtonBox), а в окне создания кандидата – кнопка для открытия файла.

Рисунок 5. Диаграмма основных классов и классов интерфейса

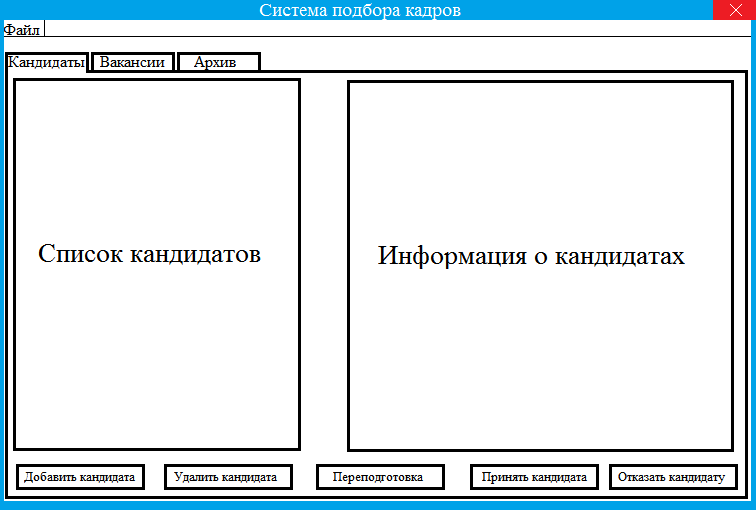


Рисунок 6. Макет интерфейса. Вкладка “Кандидаты”

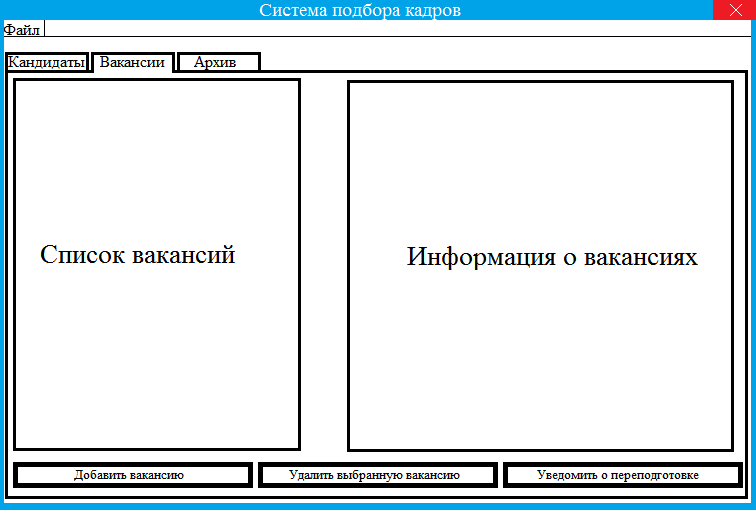


Рисунок 7. Макет интерфейса. Вкладка “Вакансии”

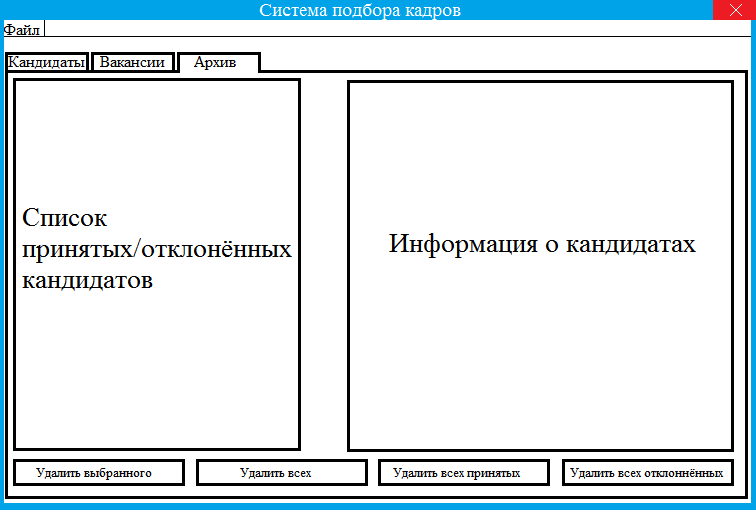


Рисунок 8. Макет интерфейса. Вкладка “Архив”

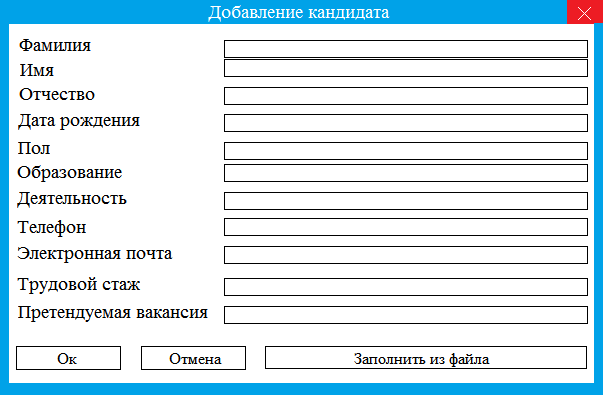


Рисунок 9. Макет интерфейса. Окно добавления кандидата

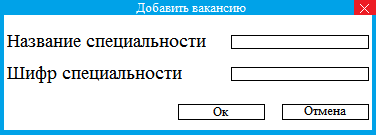


Рисунок 10. Макет интерфейса. Окно добавления вакансии

# **Часть 3 – Экспериментальная часть**

## **3.1 Тестирование**

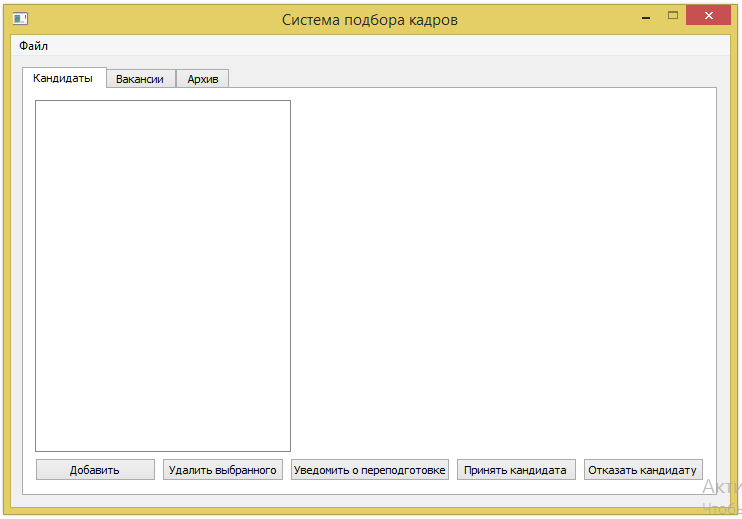


Рисунок 11. Окно запущенного приложения

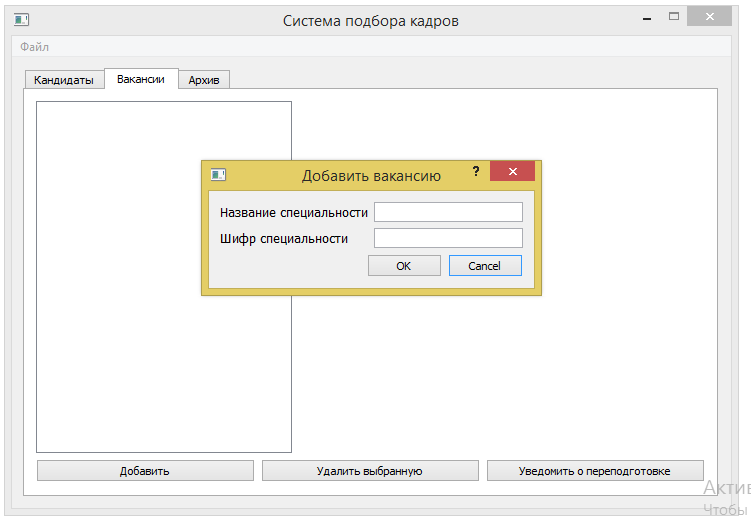


Рисунок 12. Окно добавления вакансии

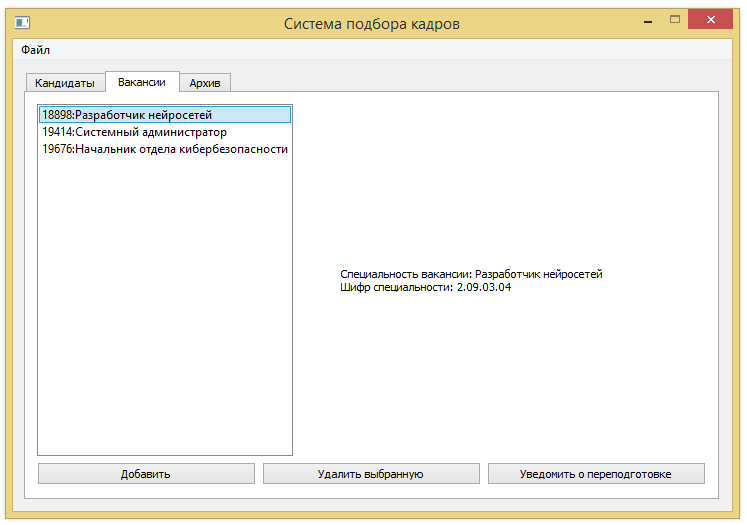


Рисунок 13. Вывод информации о вакансии

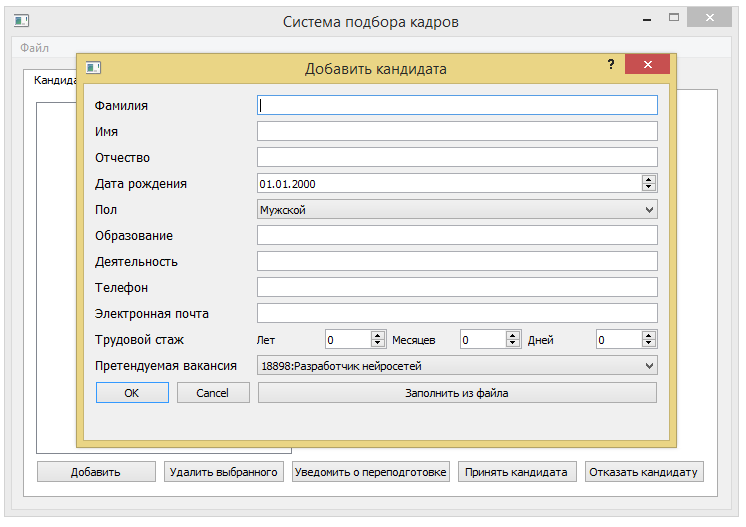


Рисунок 14. Окно добавления кандидата

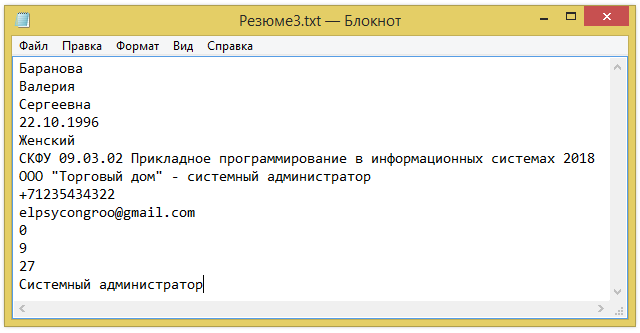


Рисунок 15. Образец файла с резюме

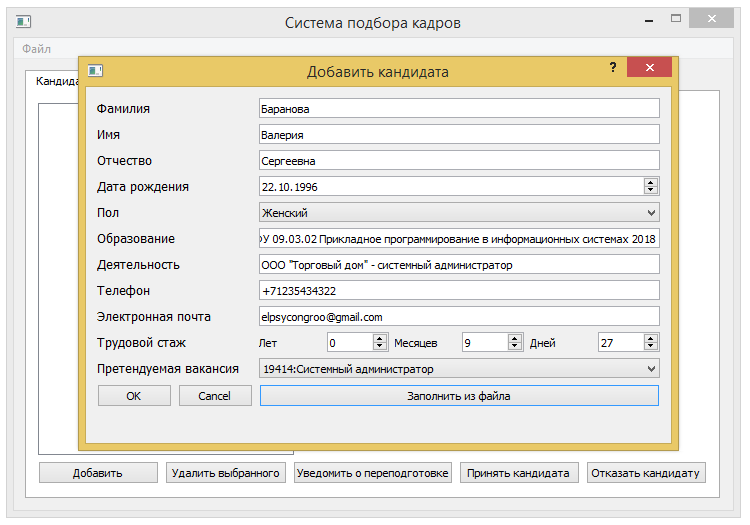
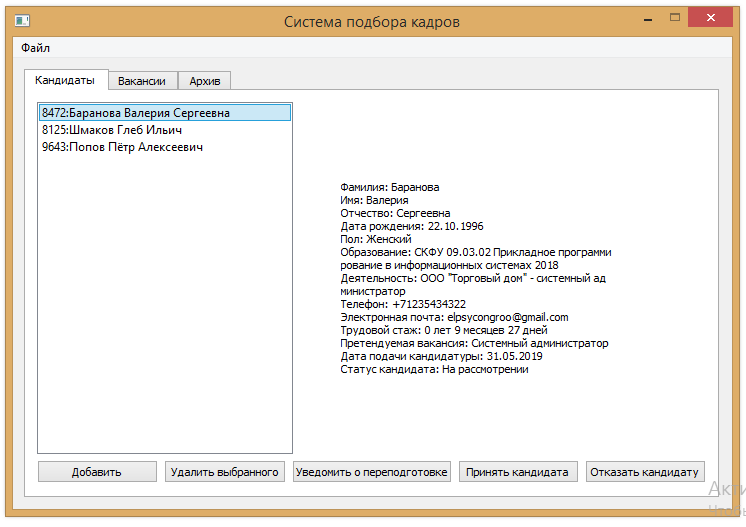
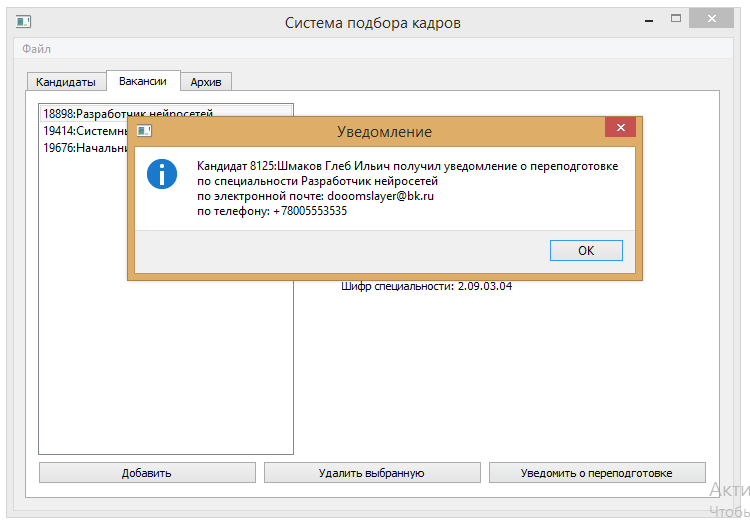
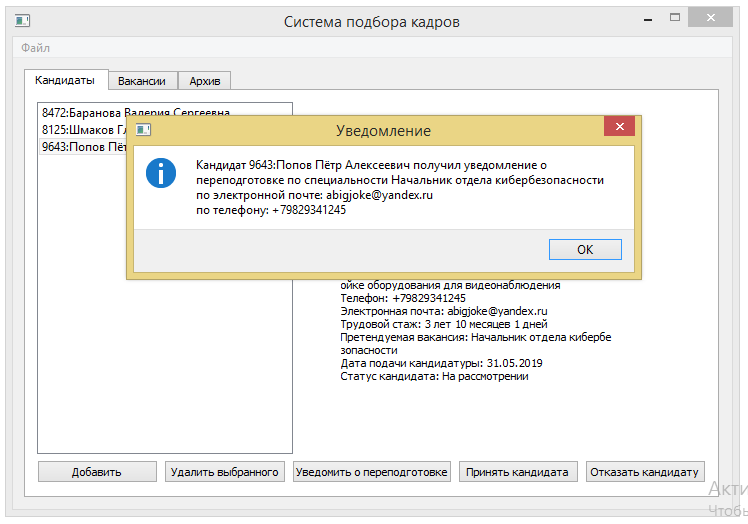


Рисунок 16. Окно, заполненное из файла

Рисунок 17. Вывод информации о кандидатах

Рисунок 18. Уведомление всех кандидатов на вакансию о переподготовке

Рисунок 19. Уведомление отдельного кандидата о переподготовке

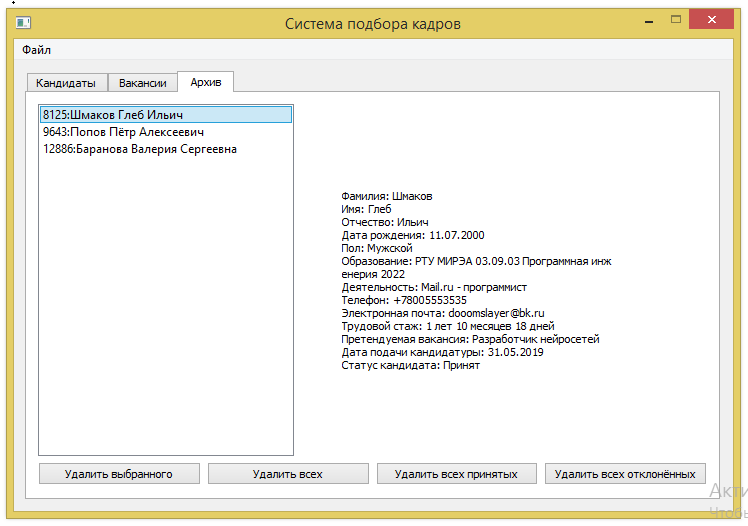
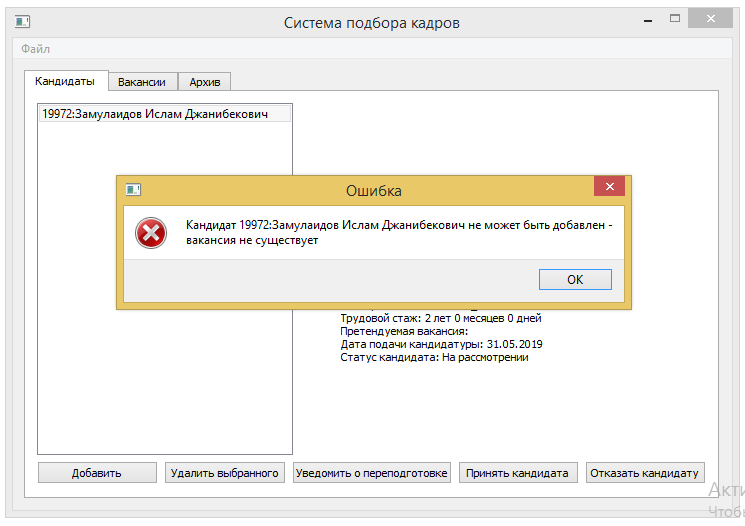


Рисунок 20. Вывод информации в архиве

 Рисунок 21. Ошибка при попытке принять кандидата на не существующую вакансию

## **3.2 Руководство пользователя**

Окно программы состоит из 3-х вкладок: Кандидаты, Вакансии и Архив.

Кнопки вкладки Кандидаты:

1. Добавить – открывает окна добавления кандидата. Данные можно заполнить вручную, или же из текстового файла, который должен быть оформлен по следующему образцу:  
   Фамилия  
   Имя  
   Отчество  
   Дата рождения(ДД.ММ.ГГГГ)  
   Пол  
   Образование(Институт, шифр специальности, название специальности,год выпуска)  
   Предыдущие места работы  
   Номер мобильного телефона  
   Адрес электронной почты  
   Стаж в годах  
   Стаж в месяцах  
   Стаж в днях  
   Претендуемая вакансия
2. Удалить выбранного – удаляет выделенного кандидата из системы.
3. Уведомить о переподготовке – программа посылает автоматически сгенерированные письмо и СМС по указанным кандидатом реквизитам чтобы уведомить кандидата о переподготовке по его специальности.
4. Принять кандидата – отправляет запись кандидата с пометкой “Принят” в архив. Занятая вакансия удаляется из системы.
5. Отказать кандидату – отправляет запись кандидата с пометкой “Отклонён” в архив.

Кнопки вкладки Вакансии:

1. Добавить – открывает окно создания вакансии.
2. Удалить выбранную – удаляет выделенную вакансию из системы.
3. Уведомить о переподготовке – посылает программа посылает автоматически сгенерированные письмо и СМС по указанным реквизитам всем претендентам на данную вакансию.

Кнопки вкладки Архив:

1. Удалить выбранного – удаляет выделленную архивную запись из системы.
2. Удалить всех – очищает архив.
3. Удалить всех принятых – удаляет всех архивных кандидатов с пометкой “Принят”.
4. Удалить всех принятых – удаляет всех архивных кандидатов с пометкой “Отклонён”.

*Важно: Проверяйте заполненные поля перед внесением в систему! Внесенные данные нельзя изменить!*

Вся архивная информация хранится в файле “archive.bin”. Этот файл может быть использован системой менеджмента сотрудниками для внесения новых сотрудников в систему менеджмента работников.

# **Заключение**

В настоящее время ни один работодатель не может обойтись без электронного контроля набора сотрудников на работу. Поэтому на любом предприятии отдел кадров нуждается в подобном приложении.

Данное приложение было разработано на языке C++ с использованием фреймворка Qt в среде разработки Qt Creator. В ходе разработки были созданы и описаны модель классов, модель всей системы и макет интерфейса. В процессе мною были освоены объектно-ориентированный анализ, язык моделирования UML, методы создания интерфейса и получены навыки разработки программного продукта и документации к нему.

# **Список литературы**

1. Объектно-ориентированный анализ и проектирование с примерами приложений [Текст] / Буч Грэди, Максимчук Роберт А. , Энгл Майкл У.; пер. с англ. Клюшин Д. А. – Киев: Вильямс, 2017. – 720 с.
2. Герберт, Ш. C++: базовый курс [Текст] / Ш. Герберт. – Москва: Вильямс, 2015. – 620 с.
3. Разработка ПО. Документация по фреймворку Qt: [Электронный ресурс]  
   https://doc.qt.io/qt-5/  
   (Дата обращения: 12.05.2019).
4. Разработка ПО. Обзор объектно-ориентированного программирования: [Электронный ресурс]  
   http://www.firststeps.ru/theory/oop/  
   (Дата обращения: 26.05.2019).
5. Разработка ПО. UML-диаграммы классов: [Электронный ресурс]  
   https://prog-cpp.ru/uml-classes/  
   (Дата обращения: 26.05.2019).
6. Разработка ПО. Методы и средства разработки пользовательского интерфейса: современное состояние: [Электронный ресурс]  
   <http://guimachine.ru/?p=937>  
   (Дата обращения: 26.05.2019).

# **Приложение**

* **Заголовочный файл addapplicant.h**

#ifndef ADDAPPLICANT\_H

#define ADDAPPLICANT\_H

#include <QDialog>

#include <QMap>

#include "applicant.h"

namespace **Ui** {

class **AddApplicant**;

}

class **AddApplicant** : public QDialog

{

Q\_OBJECT

public:

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

~***AddApplicant***();

signals:

void **sendApplicant**(Applicant a);

public slots:

void **getVacList**(QMap<QString, Vacancy> m);

private slots:

void **on\_buttonBox\_accepted**();

void **on\_buttonBox\_rejected**();

void **on\_openFile\_clicked**();

private:

Ui::AddApplicant \*ui;

QMap<QString, Vacancy> VacList;

QByteArray **del\_n**(QByteArray s);

};

#endif // ADDAPPLICANT\_H

* **Файл с реализацией методов addapplicant.cpp**

#include "addapplicant.h"

#include "ui\_addapplicant.h"

#include "applicant.h"

#include <QFile>

#include <QFileDialog>

#include <QTextCodec>

#include <QDebug>

AddApplicant::**AddApplicant**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::AddApplicant)

{

ui->setupUi(this);

ui->comboBox->addItem("Мужской");

ui->comboBox->addItem("Женский");

}

AddApplicant::~***AddApplicant***()

{

delete ui;

}

//Получение списка вакансий

void AddApplicant::**getVacList**(QMap<QString, Vacancy> m)

{

ui->comboBox\_2->clear();

VacList = m;

for(QMap<QString, Vacancy>::const\_iterator i = m.constBegin(); i != m.constEnd(); i++)

{

ui->comboBox\_2->addItem(i.key());

}

}

//Кнопка Ok

void AddApplicant::**on\_buttonBox\_accepted**()

{

Applicant a(ui->lineEdit->text(), ui->lineEdit\_2->text(), ui->lineEdit\_3->text(), ui->dateEdit->date(),

ui->comboBox->currentText(), ui->lineEdit\_4->text(),ui->lineEdit\_5->text(),

VacList[ui->comboBox\_2->currentText()].getVacancyName(), VacList[ui->comboBox\_2->currentText()].getId(),

ui->lineEdit\_6->text(), ui->lineEdit\_7->text(), ui->spinBox\_3->value(), ui->spinBox\_2->value(),

ui->spinBox->value(), *CONSIDERATION*, QDate::currentDate());

emit sendApplicant(a);

ui->lineEdit->setText("");

ui->lineEdit\_2->setText("");

ui->lineEdit\_3->setText("");

ui->dateEdit->setDate(QDate(2000,1,1));

ui->comboBox->setCurrentIndex(0);

ui->lineEdit\_4->setText("");

ui->lineEdit\_5->setText("");

ui->lineEdit\_6->setText("");

ui->lineEdit\_7->setText("");

ui->spinBox->setValue(0);

ui->spinBox\_2->setValue(0);

ui->spinBox\_3->setValue(0);

}

//Кнопка Отмена

void AddApplicant::**on\_buttonBox\_rejected**()

{

ui->lineEdit->setText("");

ui->lineEdit\_2->setText("");

ui->lineEdit\_3->setText("");

ui->dateEdit->setDate(QDate(2000,1,1));

ui->comboBox->setCurrentIndex(0);

ui->lineEdit\_4->setText("");

ui->lineEdit\_5->setText("");

ui->lineEdit\_6->setText("");

ui->lineEdit\_7->setText("");

ui->spinBox->setValue(0);

ui->spinBox\_2->setValue(0);

ui->spinBox\_3->setValue(0);

}

//Открытие файла

void AddApplicant::**on\_openFile\_clicked**()

{

QByteArray s;

QTextCodec \*codec = QTextCodec::codecForName("windows-1251");

QFile file(QFileDialog::getOpenFileName(this, "Открыть резюме", "", "\*.txt"));

if(file.*fileName*() == "")

return;

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

ui->lineEdit->setText(codec->toUnicode(del\_n(file.readLine())));

ui->lineEdit\_2->setText(codec->toUnicode(del\_n(file.readLine())));

ui->lineEdit\_3->setText(codec->toUnicode(del\_n(file.readLine())));

ui->dateEdit->setDate(QDate::fromString(del\_n(file.readLine()), "dd.MM.yyyy"));

if(codec->toUnicode(del\_n(file.readLine())) == "Женский")

ui->comboBox->setCurrentIndex(1);

else

ui->comboBox->setCurrentIndex(0);

ui->lineEdit\_4->setText(codec->toUnicode(del\_n(file.readLine())));

ui->lineEdit\_5->setText(codec->toUnicode(del\_n(file.readLine())));

ui->lineEdit\_6->setText(codec->toUnicode(del\_n(file.readLine())));

ui->lineEdit\_7->setText(codec->toUnicode(del\_n(file.readLine())));

ui->spinBox->setValue(file.readLine().toInt());

ui->spinBox\_2->setValue(file.readLine().toInt());

ui->spinBox\_3->setValue(file.readLine().toInt());

QString vac = codec->toUnicode(file.readLine());

for(int i = 0;i < ui->comboBox\_2->count();i++)

{

ui->comboBox\_2->setCurrentIndex(i);

if(VacList[ui->comboBox\_2->currentText()].getVacancyName() == vac)

break;

}

file.*close*();

}

QByteArray AddApplicant::**del\_n**(QByteArray s)

{

s.replace('\n',"");

s.replace('\r',"");

return s;

}

* **Заголовочный файл addvacancy.h**

#ifndef ADDVACANCY\_H

#define ADDVACANCY\_H

#include <QDialog>

#include "vacancy.h"

namespace **Ui** {

class **AddVacancy**;

}

class **AddVacancy** : public QDialog

{

Q\_OBJECT

public:

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

~***AddVacancy***();

signals:

void **sendVacancy**(Vacancy v);

private slots:

void **on\_buttonBox\_rejected**();

void **on\_buttonBox\_accepted**();

private:

Ui::AddVacancy \*ui;

};

#endif // ADDVACANCY\_H

* **Файл с реализацией методов addvacancy.cpp**

#include "addvacancy.h"

#include "ui\_addvacancy.h"

#include <QDebug>

AddVacancy::AddVacancy(QWidget \*parent) :

QDialog(parent),

ui(new Ui::AddVacancy)

{

ui->setupUi(this);

}

AddVacancy::~AddVacancy()

{

delete ui;

}

//Кнопка ОК

void AddVacancy::on\_buttonBox\_accepted()

{

Vacancy v(ui->lineEdit->text(), ui->lineEdit\_2->text());

emit sendVacancy(v);

ui->lineEdit->setText("");

ui->lineEdit\_2->setText("");

}

//Кнопка Отмена

void AddVacancy::on\_buttonBox\_rejected()

{

ui->lineEdit->setText("");

ui->lineEdit\_2->setText("");

}

* **Заголовочный файл applicant.h**

#ifndef APPLICANT\_H

#define APPLICANT\_H

#include "person.h"

#include "vacancy.h"

#include <QString>

#include <QDebug>

enum **Status** {*CONSIDERATION*, *APPLIED*, *DENIED*};

class **Applicant** : public Person, public Vacancy

{

QString phone\_number;

QString email;

unsigned int e\_days;

unsigned int e\_months;

unsigned int e\_years;

Status status;

QDate date;

public:

**Applicant**();

**Applicant**(QString snm, QString nm, QString ptr, QDate bd, QString sx, QString edu, QString act,

QString vnm, QString vid,

QString p, QString e, unsigned int d, unsigned int m, unsigned int y, Status s,

QDate dt);

void **setStatus**(Status s);

QString **getPhoneNumber**() const {return phone\_number;}

QString **getEmail**() const {return email;}

unsigned int **getExpDays**() const {return e\_days;}

unsigned int **getExpMonth**() const {return e\_months;}

unsigned int **getExpYears**() const {return e\_years;}

QString **getStatus**() const;

QDate **getDateOfConsideration**() const {return date;}

friend QDataStream& operator<<(QDataStream &d, const Applicant &a);

friend QDataStream& operator>>( QDataStream& d, Applicant &a);

};

#endif // APPLICANT\_H

* **Файл с реализацией методов applicant.cpp**

#include "applicant.h"

#include <QDebug>

Applicant::**Applicant**(): Person (), Vacancy ()

{

phone\_number = "";

email = "";

e\_days = 0;

e\_months = 0;

e\_years = 0;

status = *CONSIDERATION*;

date = QDate::currentDate();

}

Applicant::**Applicant**(QString snm, QString nm, QString ptr, QDate bd, QString sx, QString edu, QString act,

QString vnm, QString vid,

QString p, QString e, unsigned int d, unsigned int m, unsigned int y, Status s,

QDate dt):

Person (snm, nm, ptr, bd, sx, edu, act), Vacancy (vnm, vid)

{

phone\_number = p;

email = e;

e\_days = d;

e\_months = m;

e\_years = y;

status = s;

date = dt;

}

//Изменить статус

void Applicant::**setStatus**(Status s)

{

status = s;

}

//Получить статус

QString Applicant::**getStatus**() const

{

switch (status)

{

case 0:

return "На рассмотрении";

case 1:

return "Принят";

case 2:

return "Отклонён";

}

}

QDataStream& operator<<(QDataStream &d, const Applicant &a)

{

d << a.getSurname() << a.getName() << a.getPatronimic() << a.getbday() << a.getSex() << a.getEducation()

<< a.getActivity() << a.getVacancyName() << a.getId() << a.getPhoneNumber() << a.getEmail()

<< a.getExpDays() << a.getExpMonth() << a.getExpYears() << a.status << a.getDateOfConsideration();

return d;

}

QDataStream& operator>>( QDataStream& d, Applicant &a)

{

int s;

d >> a.surname >> a.Person::name >> a.patronimic >> a.bday >> a.sex >> a.education >> a.activity

>> a.Vacancy::name >> a.id >> a.phone\_number >> a.email >> a.e\_days >> a.e\_months >> a.e\_years >> s

>> a.date;

if(s == 0)

a.setStatus(Status::*CONSIDERATION*);

else if(s == 1)

a.setStatus(Status::*APPLIED*);

else if(s == 2)

a.setStatus(Status::*DENIED*);

return d;

}

* **Файл main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

* **Заголовочный файл mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "person.h"

#include "vacancy.h"

#include "applicant.h"

#include "addvacancy.h"

#include <QString>

#include <QMap>

#include <QListWidgetItem>

#include "addapplicant.h"

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

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

~***MainWindow***();

inline void **rewrite\_data**();

signals:

void **sendVacList**(QMap<QString, Vacancy> m);

private slots:

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

void **on\_pushButton\_2\_clicked**();

void **on\_addApp\_clicked**();

void **on\_AddVac\_clicked**();

void **addVac**(Vacancy v);

void **addApp**(Applicant a);

void **on\_listWidget\_2\_currentItemChanged**(QListWidgetItem \*current, QListWidgetItem \*previous);

void **on\_DelVac\_clicked**();

void **on\_listWidget\_currentItemChanged**(QListWidgetItem \*current, QListWidgetItem \*previous);

void **on\_NotByVac\_clicked**();

void **on\_delApp\_clicked**();

void **on\_notifyApp\_clicked**();

void **on\_AcApp\_clicked**();

void **on\_DecApp\_clicked**();

void **on\_listWidget\_3\_currentItemChanged**(QListWidgetItem \*current, QListWidgetItem \*previous);

void **on\_DelArch\_clicked**();

void **on\_DelAllArch\_clicked**();

void **on\_DelAc\_clicked**();

void **on\_DelDec\_clicked**();

private:

Ui::MainWindow \*ui;

AddVacancy addvacancy;

AddApplicant addapplicant;

QMap<QString , Applicant> applicants;

QMap<QString , Vacancy> vacancies;

QMap<QString , Applicant> archive;

QString **format\_string**(QString s);

};//19

#endif // MAINWINDOW\_H

* **Файл с реализацией методов mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "addvacancy.h"

#include <QDebug>

#include <cstdlib>

#include <ctime>

#include <QMessageBox>

const QString FILE\_NAME("data.bin");

const QString FILE\_NAME\_2("archive.bin");

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

setWindowFlags(Qt::*Window* | Qt::*MSWindowsFixedSizeDialogHint*);

//Загрузка данных из файла

QFile file(FILE\_NAME), file\_2(FILE\_NAME\_2);

QDataStream stream(&file), stream\_2(&file\_2);

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

stream >> applicants >> vacancies;

file.*close*();

file\_2.*open*(QIODevice::*ReadOnly*);

stream\_2 >> archive;

file\_2.*close*();

//Заполнение списков в окне

for(QMap<QString, Applicant>::const\_iterator i = applicants.constBegin(); i != applicants.constEnd(); i++)

{

ui->listWidget->addItem(i.key());

}

for(QMap<QString, Vacancy>::const\_iterator i = vacancies.constBegin(); i != vacancies.constEnd(); i++)

{

ui->listWidget\_2->addItem(i.key());

}

for(QMap<QString, Applicant>::const\_iterator i = archive.constBegin(); i != archive.constEnd(); i++)

{

ui->listWidget\_3->addItem(i.key());

}

connect(&addvacancy, SIGNAL(sendVacancy(Vacancy)), this, SLOT(addVac(Vacancy)));

connect(&addapplicant, SIGNAL(sendApplicant(Applicant)), this, SLOT(addApp(Applicant)));

connect(this, SIGNAL(sendVacList(QMap<QString, Vacancy>)),

&addapplicant, SLOT(getVacList(QMap<QString, Vacancy>)));

}

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

{

rewrite\_data();

delete ui;

}

//Перезапись данных

inline void MainWindow::**rewrite\_data**()

{

QFile file(FILE\_NAME), file\_2(FILE\_NAME\_2);

QDataStream stream(&file), stream\_2(&file\_2);

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

stream << applicants << vacancies;

file.*close*();

file\_2.*open*(QIODevice::*WriteOnly*);

stream\_2 << archive;

file\_2.*close*();

}

//Форматирование строки для нормального отображения

QString MainWindow::**format\_string**(QString s)

{

for(int i = 0, check = 0;i < s.length(); i++, check++)

{

if(check == 48)

{

s.insert(i,'\n');

check = 0;

}

else if(s[i] == '\n')

check = 0;

}

return s;

}

//Выход через меню

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

{

QApplication::quit(); //Выход

}

//Открытие окна создания кандидата

void MainWindow::**on\_addApp\_clicked**()

{

emit sendVacList(vacancies);

addapplicant.setModal(true);

addapplicant.*exec*();

}

//Открытие окна создания вакансии

void MainWindow::**on\_AddVac\_clicked**()

{

addvacancy.setModal(true);

addvacancy.*exec*();

}

//Добавление вакансии в систему

void MainWindow::**addVac**(Vacancy v)

{

srand(time(nullptr));

for(;;)

{

bool f = true;

int listNumber = rand() % 10000000;

QString ln = QString::number(listNumber);

for(int i = 0;i < ui->listWidget\_2->count(); i++)

{

if(ui->listWidget\_2->item(i)->text().startsWith(ln))

{

f = false;

break;

}

}

if(f)

{

ui->listWidget\_2->addItem(ln + ":" + v.getVacancyName());

vacancies[ln + ":" + v.getVacancyName()] = v;

break;

}

}

rewrite\_data();

}

//Добавление кандидата в систему

void MainWindow::**addApp**(Applicant a)

{

srand(time(nullptr));

for(;;)

{

bool f = true;

int listNumber = rand() % 10000000;

QString ln = QString::number(listNumber);

for(int i = 0;i < ui->listWidget->count(); i++)

{

if(ui->listWidget->item(i)->text().startsWith(ln))

{

f = false;

break;

}

}

if(f)

{

ui->listWidget->addItem(ln + ":" + a.getSurname() + " " + a.getName() + " " + a.getPatronimic());

applicants[ln + ":" + a.getSurname() + " " + a.getName() + " " + a.getPatronimic()] = a;

break;

}

}

rewrite\_data();

}

//НЕ УДАЛЯТЬ

void MainWindow::**on\_pushButton\_2\_clicked**()

{

}

//НЕ УДАЛЯТЬ

//Вывод данных о вакансии

void MainWindow::**on\_listWidget\_2\_currentItemChanged**(QListWidgetItem \*current, QListWidgetItem \*previous)

{

if(current == nullptr)

{

ui->label\_2->setText("");

return;

}

Vacancy v = vacancies[current->text()];

ui->label\_2->setText(format\_string("Специальность вакансии: " + v.getVacancyName() + "\nШифр специальности: " +

v.getId()));

}

//Удаление вакансии

void MainWindow::**on\_DelVac\_clicked**()

{

if(ui->listWidget\_2->count() == 0)

return;

if(ui->listWidget\_2->count() == 1)

{

vacancies.clear();

ui->listWidget\_2->clear();

return;

}

if(ui->listWidget\_2->currentItem() != nullptr)

{

vacancies.remove(ui->listWidget\_2->currentItem()->text());

ui->listWidget\_2->takeItem(ui->listWidget\_2->currentRow());

}

rewrite\_data();

}

//Вывод данных о кандидате

void MainWindow::**on\_listWidget\_currentItemChanged**(QListWidgetItem \*current, QListWidgetItem \*previous)

{

if(current == nullptr)

{

ui->label->setText("");

return;

}

Applicant a = applicants[current->text()];

ui->label->setText(format\_string("Фамилия: " + a.getSurname() + "\nИмя: " + a.getName() + "\nОтчество: " +

a.getPatronimic() + "\nДата рождения: " + a.getbday().toString("dd.MM.yyyy") +

"\nПол: " + a.getSex() + "\nОбразование: " + a.getEducation() +

"\nДеятельность: " + a.getActivity() + "\nТелефон: " + a.getPhoneNumber() +

"\nЭлектронная почта: " + a.getEmail() + "\nТрудовой стаж: " +

QString::number(a.getExpYears()) + " лет " + QString::number(a.getExpMonth()) +

" месяцев " + QString::number(a.getExpDays()) + " дней\nПретендуемая вакансия: " +

a.getVacancyName() + "\nДата подачи кандидатуры: " +

a.getDateOfConsideration().toString("dd.MM.yyyy") + "\nСтатус кандидата: " +

a.getStatus()));

}

//Уведомление о переподготовке по специальности

void MainWindow::**on\_NotByVac\_clicked**()

{

if(ui->listWidget\_2->currentItem() != nullptr)

{

Vacancy v = vacancies[ui->listWidget\_2->currentItem()->text()];

for(QMap<QString, Applicant>::const\_iterator i = applicants.constBegin(); i != applicants.constEnd(); i++)

{

if(i.value().getVacancyName() == v.getVacancyName())

{

QMessageBox(QMessageBox::*Information*, "Уведомление", QString("Кандидат %1 получил уведомление о "

"переподготовке по специальности %2\n"

"по электронной почте: %3\n"

"по телефону: %4")

.arg(i.key()).arg(v.getVacancyName()).arg(i.value().getEmail()).arg(i.value().getPhoneNumber()))

.*exec*();

}

}

}

}

//Удаление кандидата

void MainWindow::**on\_delApp\_clicked**()

{

if(ui->listWidget->count() == 0)

return;

if(ui->listWidget->count() == 1)

{

applicants.clear();

ui->listWidget->clear();

return;

}

if(ui->listWidget->currentItem() != nullptr)

{

applicants.remove(ui->listWidget->currentItem()->text());

ui->listWidget->takeItem(ui->listWidget->currentRow());

}

rewrite\_data();

}

//Уведомление о переподготовке конкретного кандидата

void MainWindow::**on\_notifyApp\_clicked**()

{

if(ui->listWidget->currentItem() != nullptr)

{

Applicant a = applicants[ui->listWidget->currentItem()->text()];

QMessageBox(QMessageBox::*Information*, "Уведомление", QString("Кандидат %1 получил уведомление о "

"переподготовке по специальности %2\n"

"по электронной почте: %3\n"

"по телефону: %4")

.arg(ui->listWidget->currentItem()->text()).arg(a.getVacancyName()).arg(a.getEmail())

.arg(a.getPhoneNumber())).*exec*();

}

}

//Принять кандидата

void MainWindow::**on\_AcApp\_clicked**()

{

if(ui->listWidget->currentItem() != nullptr)

{

Applicant a = applicants[ui->listWidget->currentItem()->text()];

bool b = false;

for(QMap<QString, Vacancy>::const\_iterator i = vacancies.constBegin(); i != vacancies.constEnd(); i++)

{

if(a.getVacancyName() == i.value().getVacancyName() && a.getId() == i.value().getId())

{

for(int j = 0;j < ui->listWidget\_2->count(); j++)

{

if(ui->listWidget\_2->item(j)->text() == i.key())

{

ui->listWidget\_2->takeItem(j);

break;

}

}

vacancies.remove(i.key());

b = true;

break;

}

}

if(!b)

{

QMessageBox(QMessageBox::*Critical*, "Ошибка", QString("Кандидат %1 не может быть добавлен - вакансия не "

"существует")

.arg(ui->listWidget->currentItem()->text())).*exec*();

return;

}

a.setStatus(Status::*APPLIED*);

archive[ui->listWidget->currentItem()->text()] = a;

//Вывод в архиве

ui->listWidget\_3->addItem(ui->listWidget->currentItem()->text());

applicants.remove(ui->listWidget->currentItem()->text());

ui->listWidget->takeItem(ui->listWidget->currentRow());

}

rewrite\_data();

}

//Отказать кандидату

void MainWindow::**on\_DecApp\_clicked**()

{

if(ui->listWidget->currentItem() != nullptr)

{

Applicant a = applicants[ui->listWidget->currentItem()->text()];

a.setStatus(Status::*DENIED*);

archive[ui->listWidget->currentItem()->text()] = a;

//Вывод в архиве

ui->listWidget\_3->addItem(ui->listWidget->currentItem()->text());

applicants.remove(ui->listWidget->currentItem()->text());

ui->listWidget->takeItem(ui->listWidget->currentRow());

}

rewrite\_data();

}

//Вывод кандидатов в архиве

void MainWindow::**on\_listWidget\_3\_currentItemChanged**(QListWidgetItem \*current, QListWidgetItem \*previous)

{

if(current == nullptr)

{

ui->label\_3->setText("");

return;

}

Applicant a = archive[current->text()];

ui->label\_3->setText(format\_string("Фамилия: " + a.getSurname() + "\nИмя: " + a.getName() + "\nОтчество: " +

a.getPatronimic() + "\nДата рождения: " + a.getbday().toString("dd.MM.yyyy") +

"\nПол: " + a.getSex() + "\nОбразование: " + a.getEducation() +

"\nДеятельность: " + a.getActivity() + "\nТелефон: " + a.getPhoneNumber() +

"\nЭлектронная почта: " + a.getEmail() + "\nТрудовой стаж: " +

QString::number(a.getExpYears()) + " лет " + QString::number(a.getExpMonth()) +

" месяцев " + QString::number(a.getExpDays()) +

" дней\nПретендуемая вакансия: " + a.getVacancyName() +

"\nДата подачи кандидатуры: " +

a.getDateOfConsideration().toString("dd.MM.yyyy") + "\nСтатус кандидата: " +

a.getStatus()));

}

//Удаление записи из архива

void MainWindow::**on\_DelArch\_clicked**()

{

if(ui->listWidget\_3->count() == 0)

return;

if(ui->listWidget\_3->count() == 1)

{

archive.clear();

ui->listWidget\_3->clear();

return;

}

if(ui->listWidget\_3->currentItem() != nullptr)

{

archive.remove(ui->listWidget\_3->currentItem()->text());

ui->listWidget\_3->takeItem(ui->listWidget\_3->currentRow());

}

rewrite\_data();

}

//Удаление всех записей из архива

void MainWindow::**on\_DelAllArch\_clicked**()

{

archive.clear();

ui->listWidget\_3->clear();

rewrite\_data();

}

//Удаление принятых кандидатов

void MainWindow::**on\_DelAc\_clicked**()

{

for(int i = 0;i < ui->listWidget\_3->count(); i++)

{

if(archive[ui->listWidget\_3->item(i)->text()].getStatus() == "Принят")

{

archive.remove(ui->listWidget\_3->item(i)->text());

ui->listWidget\_3->takeItem(i);

i--;

}

}

rewrite\_data();

}

//Удаление отклонённых кандидатов

void MainWindow::**on\_DelDec\_clicked**()

{

for(int i = 0;i < ui->listWidget\_3->count(); i++)

{

if(archive[ui->listWidget\_3->item(i)->text()].getStatus() == "Отклонён")

{

archive.remove(ui->listWidget\_3->item(i)->text());

ui->listWidget\_3->takeItem(i);

i--;

}

}

rewrite\_data();

}

* **Заголовочный файл person.h**

#ifndef PERSON\_H

#define PERSON\_H

#include <QString>

#include <QDate>

class **Person**

{

protected:

QString surname;

QString name;

QString patronimic;

QDate bday;

QString sex;

QString education;

QString activity;

public:

**Person**();

**Person**(QString s, QString n, QString p, QDate b,QString se, QString e, QString a);

QString **getSurname**() const {return surname; }

QString **getName**() const {return name; }

QString **getPatronimic**() const {return patronimic; }

QDate **getbday**() const {return bday; }

QString **getSex**() const {return sex; }

QString **getEducation**() const {return education; }

QString **getActivity**() const {return activity; }

};

#endif // PERSON\_H

* **Файл с реализацией методов person.cpp**

#include "person.h"

Person::**Person**()

{

surname = "";

name = "";

patronimic = "";

bday.setDate(1,1,1);

sex = "";

education = "";

activity = "";

}

Person::**Person**(QString s, QString n, QString p, QDate b,QString se, QString e, QString a)

{

surname = s;

name = n;

patronimic = p;

bday = b;

sex = se;

education = e;

activity = a;

}

* **Заголовочный файл vacancy.h**

#ifndef VACANCY\_H

#define VACANCY\_H

#include <QString>

#include <QDate>

#include <QDataStream>

#include <QFile>

#include <QDebug>

using namespace std;

class **Vacancy**

{

protected:

QString name;

QString id;

public:

**Vacancy**() {name = ""; id = "";}

**Vacancy**(QString n, QString i) {name = n; id = i;}

QString **getVacancyName**() const {return name;}

QString **getId**() const {return id;}

friend QDataStream& operator<<(QDataStream &d, const Vacancy &v);

friend QDataStream& operator>>( QDataStream& d, Vacancy &v);

};

#endif // VACANCY\_H

* **Файл с реализацией методов vacancy.cpp**

#include "vacancy.h"

QDataStream& operator<<(QDataStream &d, const Vacancy &v)

{

d << v.name << v.id;

return d;

}

QDataStream& operator>>( QDataStream& d, Vacancy &v)

{

d >> v.name >> v.id;

return d;

}