Министерство образования и науки Республики Татарстан

ГАПОУ «Зеленодольский механический колледж»

Специальность 09.02.07 «Информационные системы и программирование»

ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем»

**КУРСОВОЙ ПРОЕКТ**

Тема:

Разработка программного модуля «Выдача банком кредитов»

Оценка\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Руководитель (Парфенов А.В.)

Студент гр.235 (Иванов И.И.)

г.Зеленодольск

2025

**Содержание**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

*ЗМК.09.02.07.000.235.ПЗ*

Разраб.

*Иванов И.И.*

Провер.

*Парфенов А.В.*

Н. Контр.

*Демидова В.М.*

.

Пояснительная

записка

Лит.

Листов

48

Гр. 235

СОДЕРЖАНИЕ 2

ВВЕДЕНИЕ 3

ЦЕЛЬ И ЗАДАЧИ ВЫПОЛНЕНИЯ РАБОТЫ 4

1 ВНЕШНЕЕ ОПИСАНИЕ 5

1.1 Анализ предметной области 5

1.2 Функциональная спецификация 6

2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 7

3 ПРОГРАММИРОВАНИЕ 8

3.1 Описание классов 8

3.2 Схема данных 10

4 КЛИЕНТСКОЕ ПРИЛОЖЕНИЕ 13

4.1 Структура 13

4.2 Проектирование графического интерфейса 14

5 ТЕСТИРОВАНИЕ 20

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 22

ПРИЛОЖЕНИЕ А. Листинг программного модуля 23

ПРИЛОЖЕНИЕ Б. Диск с исполняемым кодом программного модуля 33

**ВВЕДЕНИЕ**

В данной пояснительной записке рассматривается описание программного модуля «Выдача банком кредитов» на основе объектно-ориентированного подхода.

При объектно-ориентированном подходе программа представляет собой описание объектов, их свойств (или атрибутов), совокупностей (или классов), отношений между ними, способов их взаимодействия и операций над объектами (или методов).

Несомненным преимуществом данного подхода является концептуальная близость к предметной области произвольной структуры и назначения. Механизм наследования атрибутов и методов позволяет строить производные понятия на основе базовых и таким образом создавать модель сколь угодно сложной предметной области с заданными свойствами.

Еще одним теоретически интересным и практически важным свойством объектно-ориентированного подхода является поддержка механизма обработки событий, которые изменяют атрибуты объектов и моделируют их взаимодействие в предметной области.

В отличие от предыдущих подходов к программированию, объектно-ориентированный подход требует глубокого понимания основных принципов, или, иначе, концепций, на которых он базируется. К числу основополагающих понятий ООП обычно относят абстракцию данных, наследование, инкапсуляцию и полиморфизм.

Объектно-ориентированное программирование в настоящее время является абсолютным лидером в области прикладного программирования.

В качестве основного инструмента разработки применяется Intellij IDEA. Язык программирования Java.

**ЦЕЛЬ И ЗАДАЧИ ВЫПОЛНЕНИЯ РАБОТЫ**

Целью данного курсового проекта является автоматизация выдачи банком кредитов на основе объектно-ориентированного подхода, а также получение навыков в реализации этого подхода, проектировании и реализации схемы данных.

Задачи выполнения работы:

* исследование предметной области с целью выявления основных сущностей и их атрибутов;
* проектирование иерархии классов и интерфейсов на основе выделенных сущностей;
* использование наследования и агрегации для структуризации классов;
* применение принципа инкапсуляции к классам;
* использование интерфейсов, описывающих операции в данной предметной области, для обеспечения множественного наследования;
* тестирование созданных классов в клиентском приложении, автоматизирующем деятельность предметной области;
* документация разработанных классов при помощи XML комментариев.

**1 ВНЕШНЕЕ ОПИСАНИЕ**

**1.1 Анализ предметной области**

Анализ предметной области – важнейший этап разработки программного обеспечения.

Анализ выдачи банком кредитов позволяет определить ключевые понятия, выяснить задачи, требующие решения в рамках проекта, проанализировать функциональные требования будущего продукта (набор функций, предоставляемых программой) и нефункциональные (надёжность, эффективность, изучаемость, модифицируемость и т.п.).

Компонентами данной предметной области являются виды кредитов, клиенты и кредиты. Данное приложение должно позволять пользователю создавать записи, удалять, редактировать, сохранять и открывать файлы, создаваемые программой PostgreSQL 17.

Нефункциональные требования к программному средству:

* надежность: программа должна быть автономной;
* эффективность: программа должна иметь минимальные требования к аппаратному обеспечению. Для использования данного программного средства не должна требоваться установка на компьютер. Не должно требоваться дополнительных периферийных средств (принтер, сканер, дополнительные дисководы и т.д.);
* изучаемость: программа должна быть интуитивно понятна, иметь удобный пользовательский интерфейс;
* модифицируемость: программа должна быть легко модифицируемой вследствие небольшого исходного размера и объектно-ориентированного подхода.

**1.2. Функциональная спецификация**

Минимальные технические характеристики компьютера, на котором гарантируется стабильная работа программы:

* компьютер/процессор: компьютер с процессором класса Pentium II 450 МГц;
* память: 64 МБ ОЗУ;
* монитор: монитор VGA с разрешением 800x600 точек или более высоким, поддерживающий 256 цветов;
* операционная система: операционная система Windows XP с установленным Microsoft .NET Framework 4.5.2;
* наличие свободного дискового пространства на жёстком диске.

Внешние функции:

* внесение данных и редактирование данных о монетах, коллекционерах и коллекции;
* сохранение данных в mdf -файл;
* считывание данных из mdf -файла;
* проверка вводимых данных и вывод сообщений об ошибках;
* программа должна выполнять все функции за предельно малое количество времени;

Критические случаи:

* отсутствие свободного места на жёстком диске. Любая операция с файловой системой должна быть отменена;
* некорректный mdf-файл. Программа не должна аварийно завершаться;
* завершение работы Windows. В этом случае программа должна завершать свою работу, не применяя выбранные пользователем настройки.

**2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Техническое задание определяет требования к будущему программному модулю со стороны основных процессов.

Система предназначена для решения перечисленных ниже задач:

* изменение списка коллекций (добавление, удаление, редактирование);
* изменение списка монет (добавление, удаление, редактирование);
* изменение списка коллекционеров (добавление, удаление, редактирование);
* отображение информации, хранящейся в mdf –файле;
* сохранение информации в mdf –файл.

**3 ПРОГРАММИРОВАНИЕ**

**3.1 Описание классов**

В ООП главным элементом является класс, включающий множество объектов с одинаковыми свойствами, операциями и отношениями. Класс имеет внутреннее (реализацию) и внешнее представление – интерфейс. В данной программе реализованы классы Client, Credit, Kind Credit, а также интерфейс MainWindow.

Класс Client описывает клиентов. Содержит такие свойства, как код клиента, имя клиента, вид имущества, адрес, номер телефона, контакты.

@Entity  
@Table(name = "clients")  
public class Client {  
  
 @Id  
 @Column(name = "client\_id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer clientId;  
  
 @Column(name = "name")  
 private String name;  
  
 @Column(name = "kind\_property")  
 private String kindProperty;  
  
 @Column(name = "address")  
 private String address;  
  
 @Column(name = "phone")  
 private String phone;  
  
 @Column(name = "contact")  
 private String contact;  
  
 *// Геттеры и сеттеры* public Integer getClientId() {  
 return clientId;  
 }  
  
 public void setClientId(Integer clientId) {  
 this.clientId = clientId;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 if (!name.isEmpty()) this.name = name;  
 else throw new IllegalArgumentException("Имя не должно быть пустым!");  
 }  
  
 public String getKindProperty() {  
 return kindProperty;  
 }  
  
 public void setKindProperty(String kindProperty) {  
 if (!kindProperty.isEmpty()) this.kindProperty = kindProperty;  
 else throw new IllegalArgumentException("Тип имущества не должен быть пуcтым!");  
 }  
  
 public String getAddress() {  
 return address;  
 }  
  
 public void setAddress(String address) {  
 if (!address.isEmpty()) this.address = address;  
 else throw new IllegalArgumentException("Адрес не может быть пустым!");  
 }  
  
 public String getPhone() {  
 return phone;  
 }  
  
 public void setPhone(String phone) {  
 if (!phone.isEmpty() && phone.length() <= 20) this.phone = phone;  
 else throw new IllegalArgumentException("Телефон не может быть пустым или содержать более 20 символов!");  
 }  
  
 public String getContact() {  
 return contact;  
 }  
  
 public void setContact(String contact) {  
 if (!contact.isEmpty()) this.contact = contact;  
 else throw new IllegalArgumentException("Почта не может быть пустой!");  
 }

@Override  
 public String toString(){  
 return name;  
 }  
}

Класс KindCredit описывает виды кредитов и содержит такие атрибуты и параметры, как код вида кредита, название, условия получения, ставка, срок.

@Entity  
@Table(name = "kind\_credit")  
public class KindCredit {  
  
 @Id  
 @Column(name = "kind\_credit\_id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer kindCreditId;  
  
 @Column(name = "name")  
 private String name;  
  
 @Column(name = "conditions")  
 private String conditions;  
  
 @Column(name = "rate")  
 private BigDecimal rate;  
  
 @Column(name = "term")  
 private Integer term;  
  
 *// Геттеры и сеттеры* public Integer getKindCreditId() {  
 return kindCreditId;  
 }  
  
 public void setKindCreditId(Integer kindCreditId) {  
 this.kindCreditId = kindCreditId;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 if (!name.isEmpty()) {  
 this.name = name;  
 } else {  
 throw new IllegalArgumentException("Имя не должно быть пустым");  
 }  
 }  
  
 public String getConditions() {  
 return conditions;  
 }  
  
 public void setConditions(String conditions) {  
 if (!conditions.isEmpty()) {  
 this.conditions = conditions;  
 } else {  
 throw new IllegalArgumentException("Условие не должно быть пустым");  
 }  
 }  
  
 public BigDecimal getRate() {  
 return rate;  
 }  
  
 public void setRate(String rateText) {  
 if (!rateText.isEmpty() && rateText.matches("^[0-9]+([.,][0-9]+)?$")) {  
 BigDecimal rate = BigDecimal.*valueOf*(Double.*parseDouble*(rateText.replace(",", ".")));  
 if (rate.compareTo(BigDecimal.*valueOf*(0)) == 1 && rate.compareTo(BigDecimal.*valueOf*(99)) == -1) {  
 this.rate = rate;  
 } else {  
 throw new IllegalArgumentException("Ставка должна быть в диапозоне от 0 до 99!");  
 }  
 } else {  
 throw new IllegalArgumentException("Ставка должна быть числом!");  
 }  
 }  
  
 public Integer getTerm() {  
 return term;  
 }  
  
 public void setTerm(String termText) {  
 if (!termText.isEmpty() && termText.matches("-?\\d+")) {  
 int term = Integer.*parseInt*(termText);  
 this.term = term;  
 } else {  
 throw new IllegalArgumentException("Срок должен быть целым числом!");  
 }  
 }  
@Override  
 public String toString() {  
 return name;  
 }  
}

Класс Credit описывает кредит и содержит такие атрибуты, как код кредита, код вида кредита, код клиента, сумма, дата выдачи.

@Entity  
@Table(name = "credits")  
public class Credit {  
  
 @Id  
 @Column(name = "credit\_id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer creditId;  
  
 @ManyToOne  
 @JoinColumn(name = "kind\_credit\_id")  
 private KindCredit kindCredit;  
  
 @ManyToOne  
 @JoinColumn(name = "client\_id")  
 private Client client;  
  
 @Column(name = "summa")  
 private BigDecimal summa;  
  
 @Column(name = "date")  
 @Temporal(TemporalType.*DATE*)  
 private Date date;  
  
 *// Геттеры и сеттеры* public Integer getCreditId() {  
 return creditId;  
 }  
  
 public void setCreditId(Integer creditId) {  
 this.creditId = creditId;  
 }  
  
 public KindCredit getKindCredit() {  
 return kindCredit;  
 }  
  
 public void setKindCredit(KindCredit kindCredit) {  
 this.kindCredit = kindCredit;  
 }  
  
 public Client getClient() {  
 return client;  
 }  
  
 public void setClient(Client client) {  
 this.client = client;  
 }  
  
 public BigDecimal getSumma() {  
 return summa;  
 }  
  
 public void setSumma(String summa) {  
 if (!summa.isEmpty() && summa.matches("-?\\d+")) {  
 BigDecimal sum = BigDecimal.*valueOf*(Long.*parseLong*(summa));  
 if (sum.compareTo(BigDecimal.*valueOf*(0)) == 1) {  
 this.summa = sum;  
 } else {  
 throw new IllegalArgumentException("Сумма должна быть больше 0!");  
 }  
 } else {  
 throw new IllegalArgumentException("Сумма должна быть числом!");  
 }  
 }  
  
 public Date getDate() {  
 return date;  
 }  
  
 public void setDate(Date date) {  
 if (new Date().after(date)) {  
 this.date = date;  
 }else {  
 throw new IllegalArgumentException("Дата не должна превышать текущую!");  
 }  
 }  
  
 @Override  
 public String toString() {  
 return "Credit{" +  
 "creditId=" + creditId +  
 ", kindCredit=" + kindCredit +  
 ", client=" + client +  
 ", summa=" + summa +  
 ", date=" + date +  
 '}';  
 }  
}

Класс HibernateUtil для создания соединения с базой данных.

public class HibernateUtil {  
  
 private static final SessionFactory *sessionFactory* = *buildSessionFactory*();  
  
 private static SessionFactory buildSessionFactory() {  
 try {  
 return new Configuration().configure().buildSessionFactory();  
 } catch (Throwable ex) {  
 System.*err*.println("Initial SessionFactory creation failed." + ex);  
 throw new ExceptionInInitializerError(ex);  
 }  
 }  
  
 public static SessionFactory getSessionFactory() {  
 return *sessionFactory*;  
 }  
  
 public static void shutdown() {  
 *getSessionFactory*().close();  
 }  
  
}

**3.2 Схема данных**

На рисунках 1-3 представлены таблицы файла mdf в режиме конструктора.

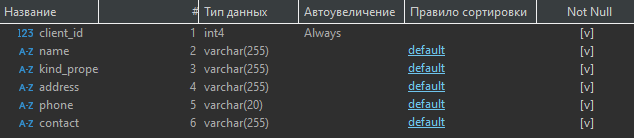


Рисунок 1 – Таблица clients

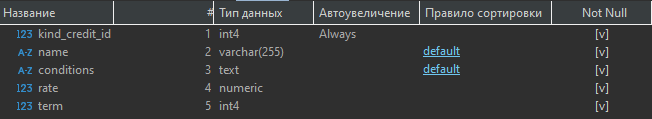


Рисунок 2 – Таблица kind credits

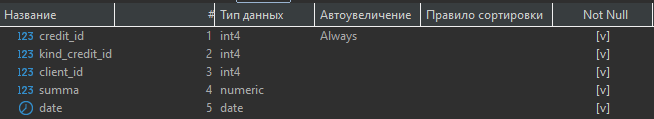


Рисунок 3 – Таблица credits

На рисунке 4 представлена схема данных

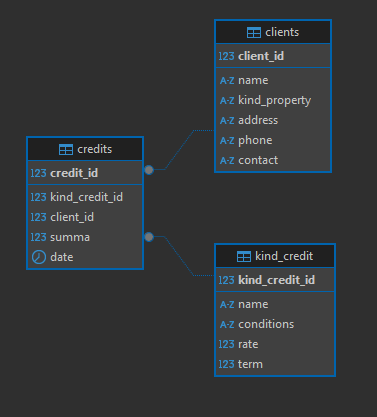


Рисунок 4 – Схема данных

Словарь данных представлен в таблицах 1-3.

Таблица 1 – Словарь данных таблицы Clients

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clients (Клиенты) | | | | |
| KEY | FIELD NAME | DATA TYPE | REQURIED? | NOTES |
| PK | client\_id | int4 | Y | Autoincrement |
|  | name | varchar(255) | Y |  |
|  | kind\_property | varchar(255) | Y |  |
|  | address | varchar(255) | Y |  |
|  | phone | varchar(20) | Y |  |
|  | contact | varchar(255) | Y |  |

Таблица 2 – Словарь данных таблицы Collectors

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kind credits (виды кредитов) | | | | |
| KEY | FIELD NAME | DATA TYPE | REQURIED? | NOTES |
| PK | kind\_credit\_id | int4 | Y | Autoincrement |
|  | name | varchar(255) | Y |  |
|  | conditions | text | Y |  |
|  | rate | numeric | Y |  |
|  | term | Int4 | Y |  |

Таблица 3 – Словарь данных таблицы Collection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Credits (Кредиты) | | | | |
| KEY | FIELD NAME | DATA TYPE | REQURIED? | NOTES |
| PK | credit\_id | int4 | Y | Autoincrement |
| FK | kind\_credit\_id | int4 | Y | Внешний ключ на kind\_credits |
| FK | client\_id | Int4 | Y | Внешний ключ на clients |
|  | summa | numeric | Y |  |
|  | date | date | Y |  |

**4 КЛИЕНТСКОЕ ПРИЛОЖЕНИЕ**

**4.1 Структура**

В данном ПП для продуктивной работы реализован удобный пользовательский интерфейс, похожий на большинство Windows – приложений. Данное программное средство можно разбить на модули, взаимодействующие друг с другом. На рисунке 5 представлена структура приложения:

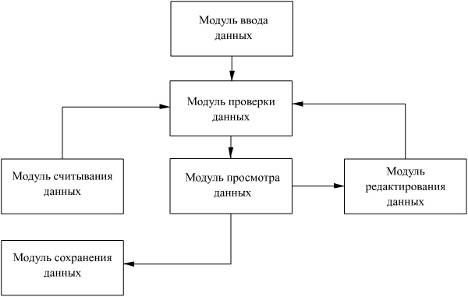


Рисунок 5 - Структура приложения

Модуль просмотра данных предоставляет возможность наглядного отображения данных.

Модуль проверки данных реализует методы:

* корректность вводимых пользователем данных;
* корректность считываемых из файла данных.

Модуль ввода данных, реализует простой графический интерфейс для ввода необходимых значений.

Модуль сохранения данных, формирует все записи в один mdf - файл.

Модуль считывания предоставляет возможность открытия mdf - файла и заполнения списков в программе.

Модуль редактирования записей предоставляет простой интерфейс для изменения ранее внесенных данных, а также добавления и удаления.

**4.2 Проектирование графического интерфейса**

На рисунке 6-8 представлена форма приложения и панель, содержащая таблицы данных:

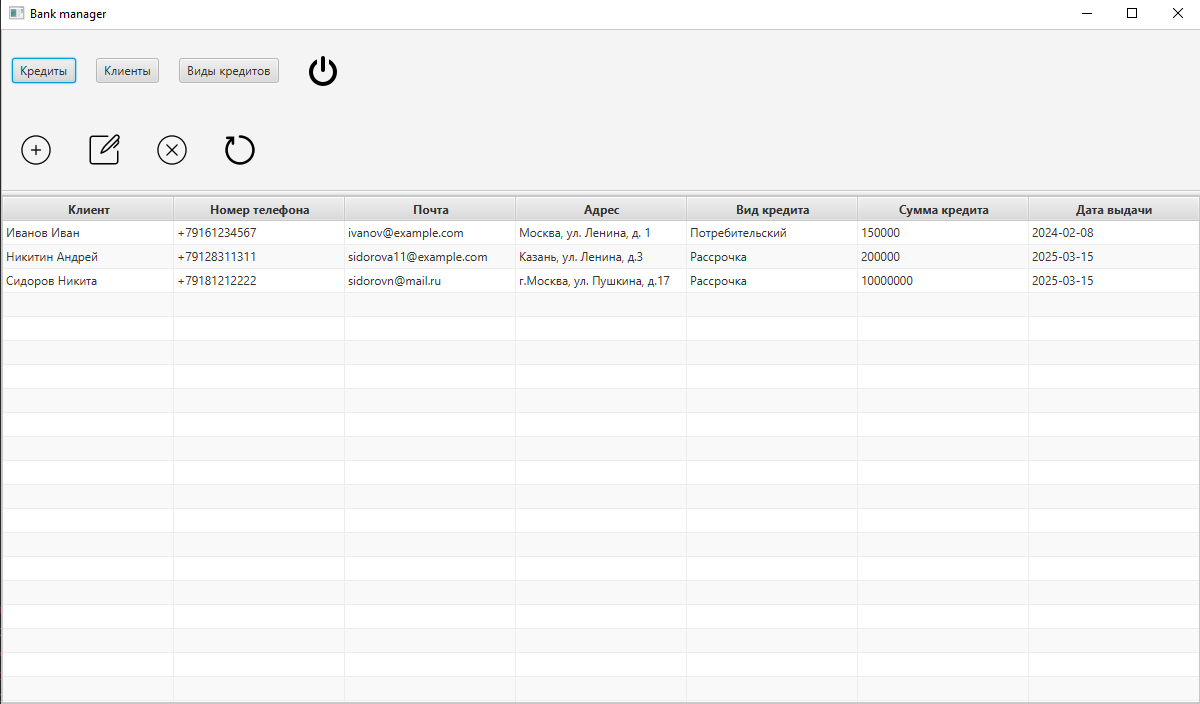


Рисунок 6 - Форма credit-view. Таблица: Кредиты

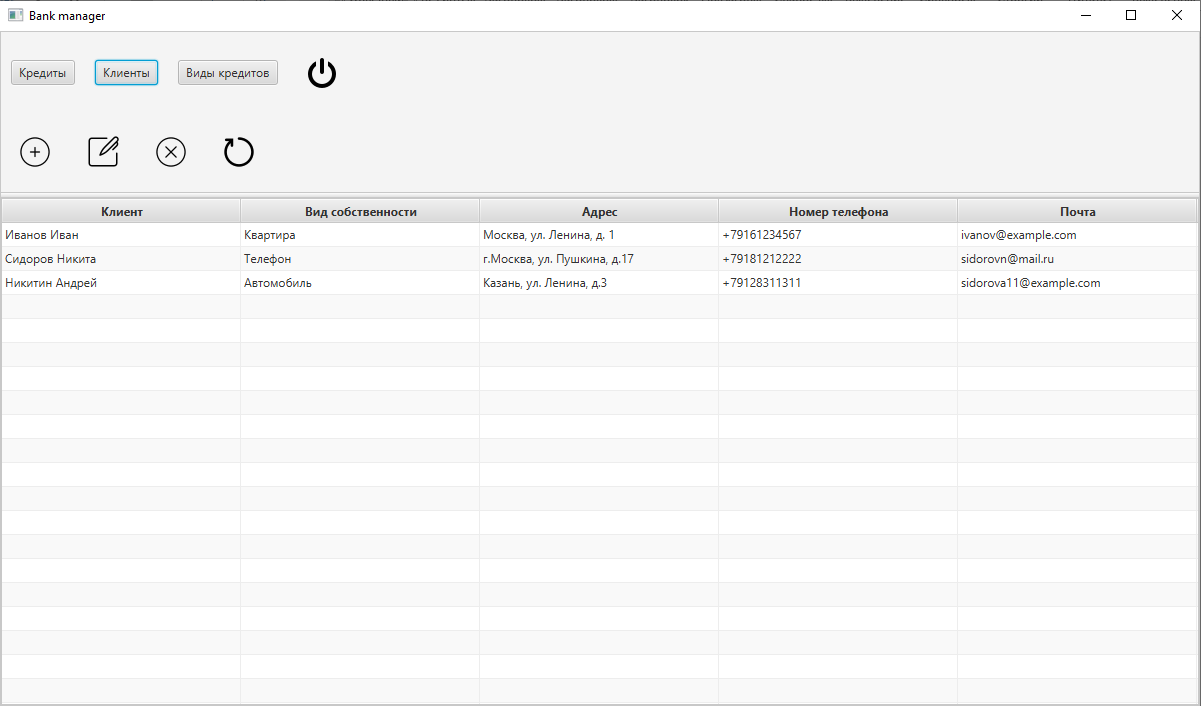


Рисунок 7 – Форма client-view. Таблица: клиенты

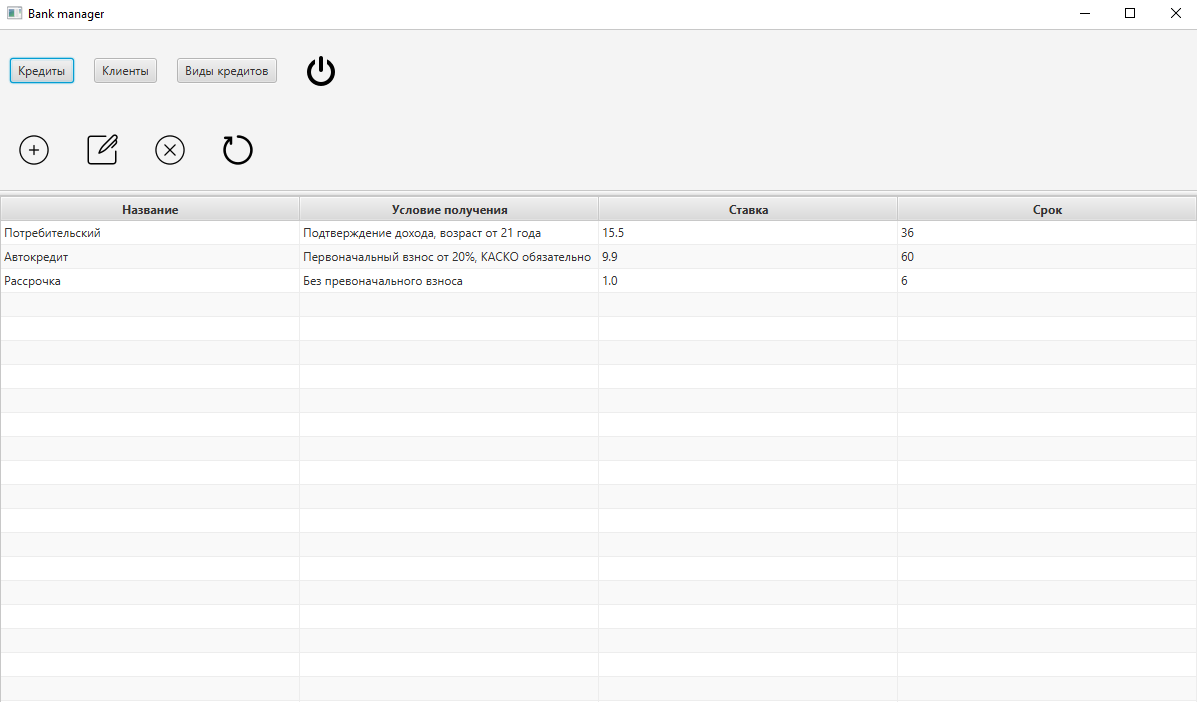


Рисунок 8 – Форма kind-credit-view. Таблица: Виды кредитов

Для редактирования записей о монетах, коллекционерах и коллекциях используются отдельные формы приложения и панели для редактирования таблиц в MainWindow. Для добавления/редактирования таблицы «Клиенты» используется форма add-client-dialog/edit-client-dialog под названием «Добавление\Редактирование клиента», для таблицы «Виды кредитов», форма add-kind-credit-dialog/edit-kind-credit-dialog с названием «Добавление\Редактирование вида кредита», а для последней таблицы используется форма add-credit-dialog/edit-credit-dialog, название которой «Добавление\Редактирование кредита». Эти формы изображены на рисунках 9-11.

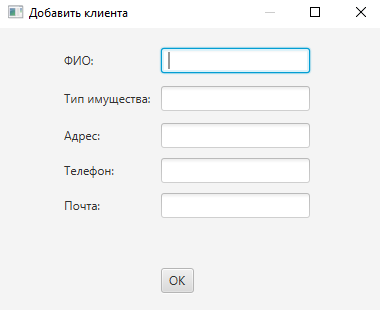


Рисунок 9 - Форма add-client-dialog

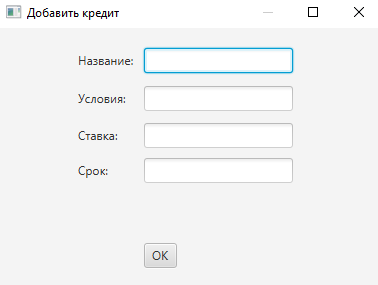


Рисунок 10 - Форма add-kind-credit-dialog

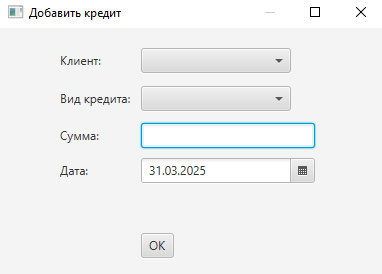


Рисунок 11 - Форма add-credit-dialog

Удаление информации о монетах, коллекционерах и коллекциях, осуществляется с помощью специальной кнопки, расположенной на главной форме приложения у каждой таблицы. Нажатием кнопки «Удалить» - удаляем из списка информацию о монете. Аналогично для других списков. Кнопки отображены на рисунках 12-14:

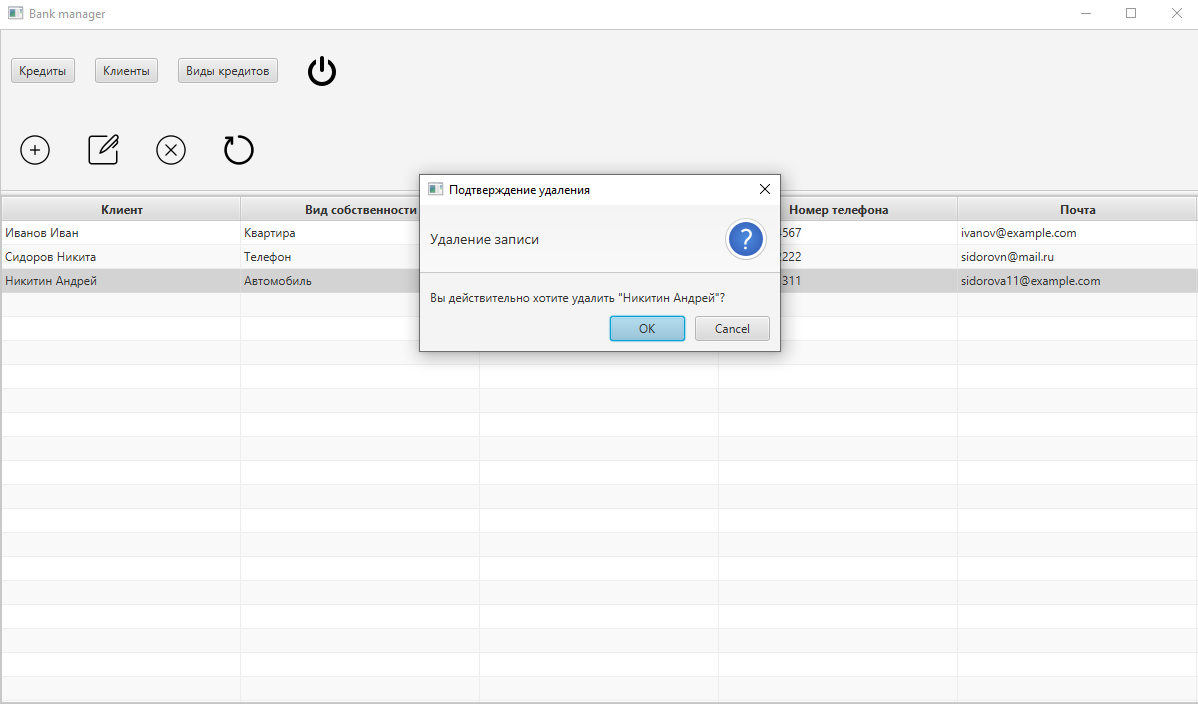


Рисунок 12 - Таблица «Клиенты»

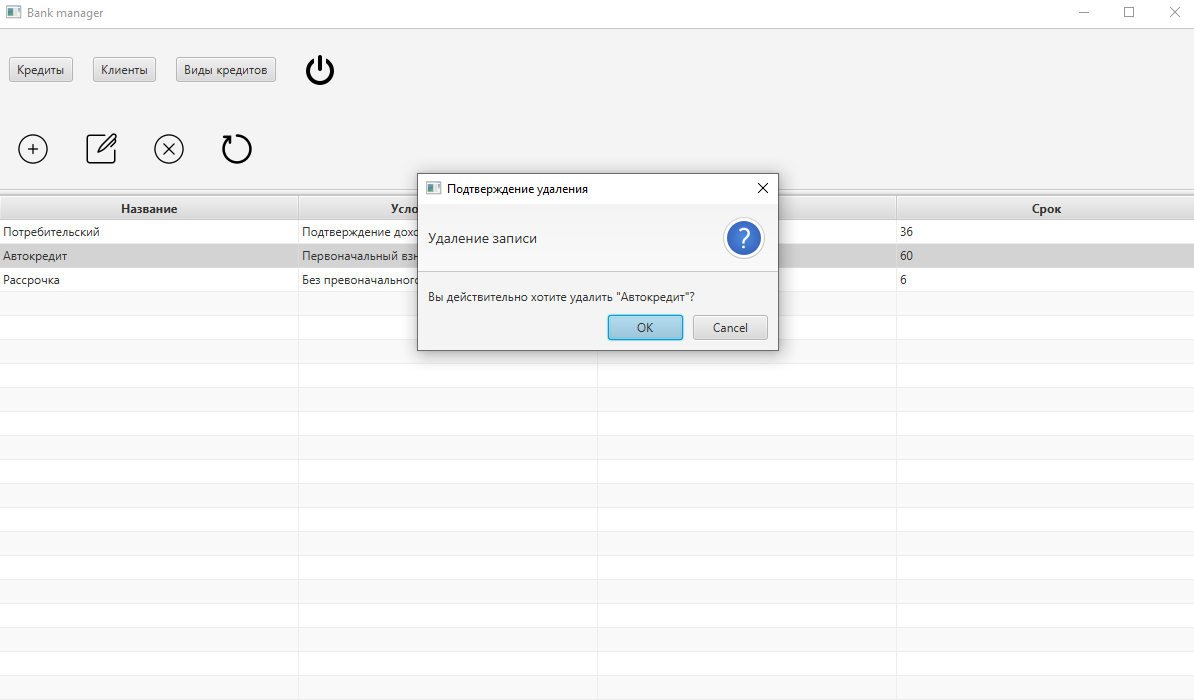


Рисунок 13 - Таблица «Виды кредита»

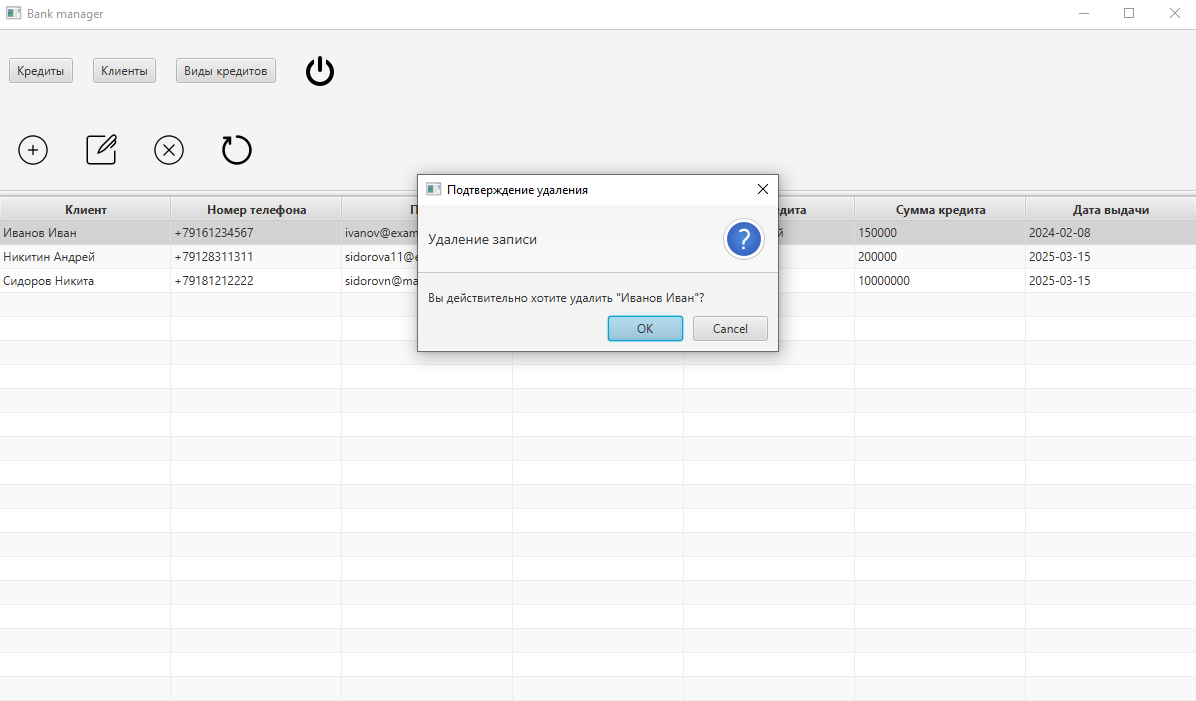


Рисунок 14 - Таблица «Кредиты»

**5 ТЕСТИРОВАНИЕ**

Для тестирования программы производились различные манипуляции с данными.

Проводились следующие тесты:

Тест 1 – Пустое значение в обязательном поле ввода данных, изображен на рисунке 15:

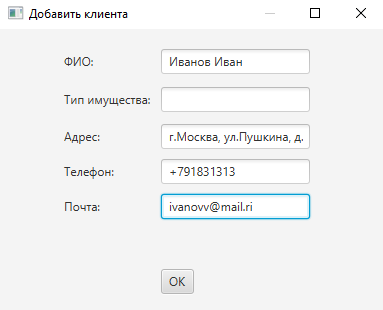


Рисунок 15 - Ввод пустого значения в поле «Тип имущества»

Сообщение, которое будет показано пользователю после ввода пустого значения, представлено на рисунке 16:

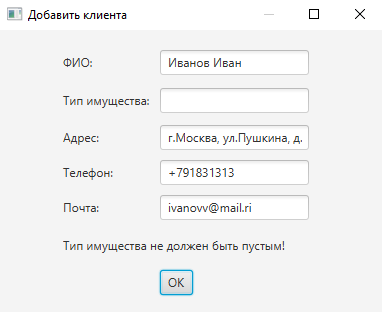


Рисунок 16 - Ошибка при вводе пустого значения

Тест 2 Пустое значение в обязательном поле ввода данных, изображен на рисунке 17:

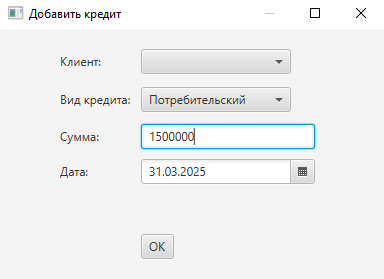


Рисунок 17 – Ввод пустого значения в поле «Клиент»

Сообщение, которое будет показано пользователю после ввода пустого значения, представлено на рисунке 18:

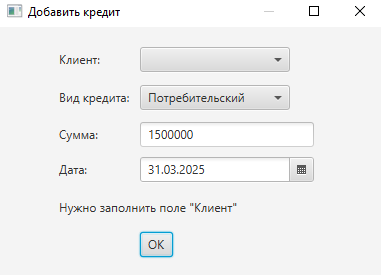


Рисунок 18 - Ввод некорректных данных

А так же были проведены другие операции, призванные выявить ошибки в программе.

Тестирование выявило мелкие недочеты, которые впоследствии были устранены.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Основы алгоритмизации и программирования : учеб. пособие / В.Д.  Колдаев; под ред. проф. Л.Г. Гагариной. — М. : ИД «ФОРУМ» : ИНФРА-М, 2020. — 414 с. — (Среднее профессиональное образование). - Режим доступа: http://znanium.com/catalog/product/980416

2. Программирование графики на С#. Теория и примеры : учеб. пособие / В.И. Корнеев, Л.Г. Гагарина, М.В. Корнеева. — М. : ИД «ФОРУМ» : ИНФРА-М, 2020. — 517 с. + Доп. материалы [Электронный ресурс; Режим доступа http://www.znanium.com].

3. Программирование на С# с погружением: практические задания и примеры кода - М.:НИЦ ИНФРА-М, 2020. - 80 с.: 60x90 1/16 - Режим доступа: http://znanium.com/catalog/product/563294

4. Программирование на языке высокого уровня. Программирование на языке С#: учеб. пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев ; под ред. Л.Г. Гагариной. — М. : ИД «ФОРУМ» : ИНФРА-М, 2020. — 512 с. — (Среднее профессиональное образование). - Режим доступа: http://znanium.com/catalog/product/918098

5. Программирование. Процедурное программирование: Учебное пособие / Кучунова Е.В., Олейников Б.В., Чередниченко О.М. - Краснояр.:СФУ, 2020. - 92 с.: ISBN 978-5-7638-3555-7 - Режим доступа: http://znanium.com/catalog/product/978627

6. Введение в программирование на языке Visual C# : учеб. пособие / С.Р. Гуриков. — М. : ФОРУМ : ИНФРА-М, 2020. — 447 с. — (Высшее образование: Бакалавриат). - Режим доступа: http://znanium.com/catalog/product/967691

7. Объектно-ориентированное программирование с примерами на С# : учеб. пособие / П.Б. Хорев. — М. : ФОРУМ : ИНФРА-М, 2020. — 200 с. — (Высшее образование: Бакалавриат). - Режим доступа: http://znanium.com/catalog/product/1018034

**ПРИЛОЖЕНИЕ А**

**Листинг программного модуля**

**Форма приложения MainWindow**

package com.example.kp;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
  
public class BankApp extends Application {  
 public static Stage *primaryStage*;  
 public static Scene *clients*;  
 public static Scene *credits*;  
 public static Scene *kindCredits*;  
  
 @Override  
 public void start(Stage stage) throws IOException {  
 *primaryStage* = stage;  
 *clients* = createScene("client-view.fxml");  
 *credits* = createScene("credit-view.fxml");  
 *kindCredits* = createScene("kind-credit-view.fxml");  
 *primaryStage*.setMinWidth(1200);  
 *primaryStage*.setMinHeight(675);  
 *primaryStage*.setTitle("Bank manager");  
 *primaryStage*.setScene(*credits*);  
 *primaryStage*.show();  
 }  
  
 private Scene createScene(String name) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(BankApp.class.getResource(name));  
 return new Scene(fxmlLoader.load());  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

**Форма приложения client-view**

package com.example.kp.controller.client;  
  
import com.example.kp.BankApp;  
import com.example.kp.model.Client;  
import com.example.kp.service.ClientService;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.util.List;  
import java.util.Optional;  
  
public class ClientController {  
  
 private List<Client> clients;  
 private ObservableList<ClientTableItem> creditsObservable;  
 @FXML  
 private TableColumn<?, ?> addressColumn;  
  
 @FXML  
 private Button clientsButton;  
  
 @FXML  
 private TableView<ClientTableItem> clientsTable;  
  
 @FXML  
 private Button creditsButton;  
  
 @FXML  
 private TableColumn<?, ?> emailColumn;  
  
 @FXML  
 private Button kindKreditButton;  
  
 @FXML  
 private TableColumn<?, ?> kindPropertyColumn;  
  
 @FXML  
 private TableColumn<?, ?> nameColumn;  
  
 @FXML  
 private TableColumn<?, ?> phoneColumn;  
  
 @FXML  
 private Button offButton;  
  
 @FXML  
 void addClient(ActionEvent event) {  
 try {  
 FXMLLoader loader = new FXMLLoader(BankApp.class.getResource("add-client-dialog.fxml"));  
 Stage dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(BankApp.*primaryStage*);  
 dialogStage.setMinWidth(400);  
 dialogStage.setScene(new Scene(loader.load()));  
 dialogStage.setTitle("Добавить клиента");  
 AddClientDialog controller = loader.getController();  
 controller.setDialogStage(dialogStage);  
 dialogStage.showAndWait();  
 updateList();  
 } catch (IOException e) {  
 System.*out*.println("Ошибка открытия окна: " + e.getMessage());  
 }  
 }  
  
 @FXML  
 void deleteClient(ActionEvent event) {  
 ClientTableItem currentItem = clientsTable.getSelectionModel().getSelectedItem();  
 int currentItemId = clientsTable.getSelectionModel().getSelectedIndex();  
 if (currentItemId != -1) {  
 Alert alert = new Alert(Alert.AlertType.*CONFIRMATION*);  
 alert.setTitle("Подтверждение удаления");  
 alert.setHeaderText("Удаление записи");  
 alert.setContentText("Вы действительно хотите удалить \"" + currentItem.getName() + "\"?");  
  
 Optional<ButtonType> result = alert.showAndWait();  
  
 if (result.isPresent() && result.get() == ButtonType.*OK*) {  
 new ClientService().delete(currentItem.getClient());  
 clientsTable.getItems().remove(currentItemId);  
 }  
 }else {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Предупреждение");  
 alert.setContentText("Выберите запись в таблице для удаления");  
 alert.showAndWait();  
 }  
 }  
  
 @FXML  
 void editClient(ActionEvent event) {  
 ClientTableItem currentItem = clientsTable.getSelectionModel().getSelectedItem();  
 int currentItemId = clientsTable.getSelectionModel().getSelectedIndex();  
 if (currentItemId != -1) {  
 try {  
 FXMLLoader loader = new FXMLLoader(BankApp.class.getResource("edit-client-dialog.fxml"));  
 Stage dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(BankApp.*primaryStage*);  
 dialogStage.setMinWidth(400);  
 dialogStage.setScene(new Scene(loader.load()));  
 dialogStage.setTitle("Редактировать клиента");  
 EditClientDialog controller = loader.getController();  
 controller.setDialogStage(dialogStage, currentItem.getClient());  
 dialogStage.showAndWait();  
 updateList();  
 } catch (IOException e) {  
 System.*out*.println("Ошибка открытия окна: " + e.getMessage());  
 }  
 } else {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Предупреждение");  
 alert.setContentText("Выберите запись в таблице для редактирования");  
 alert.showAndWait();  
 }  
 }  
  
 @FXML  
 void onClickCredit(ActionEvent event) throws IOException {  
 BankApp.*primaryStage*.setScene(BankApp.*credits*);  
 }  
  
 @FXML  
 void onKindKreditClick(ActionEvent event) {  
 BankApp.*primaryStage*.setScene(BankApp.*kindCredits*);  
 }  
  
 @FXML  
 void powerOff(ActionEvent event) {  
 BankApp.*primaryStage*.close();  
 }  
  
 @FXML  
 void updateClients(ActionEvent event) {  
 updateList();  
 }  
  
 private void updateList() {  
 clients = new ClientService().findAll();  
 creditsObservable = FXCollections.*observableArrayList*();  
  
 for (Client client : clients) {  
 creditsObservable.add(new ClientTableItem(client));  
 }  
 clientsTable.setItems(creditsObservable);  
 }  
  
 public void initialize() {  
 phoneColumn.setCellValueFactory(new PropertyValueFactory<>("phone"));  
 addressColumn.setCellValueFactory(new PropertyValueFactory<>("address"));  
 emailColumn.setCellValueFactory(new PropertyValueFactory<>("email"));  
 kindPropertyColumn.setCellValueFactory(new PropertyValueFactory<>("kindProperty"));  
 nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));  
  
 updateList();  
 }  
  
}

**Форма приложения kind-credit-view**

package com.example.kp.controller.kindcredit;  
  
import com.example.kp.BankApp;  
import com.example.kp.controller.client.AddClientDialog;  
import com.example.kp.controller.client.ClientTableItem;  
import com.example.kp.controller.client.EditClientDialog;  
import com.example.kp.model.KindCredit;  
import com.example.kp.service.KindCreditService;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.util.List;  
import java.util.Optional;  
  
public class KindCreditController {  
  
 private List<KindCredit> kindCredits;  
  
 @FXML  
 private Button clientsButton;  
  
 @FXML  
 private TableColumn<?, ?> conditionColumn;  
  
 @FXML  
 private Button creditsButton;  
  
 @FXML  
 private TableView<KindCreditTableItem> kindCreditsTable;  
  
 @FXML  
 private Button kindKreditButton;  
  
 @FXML  
 private TableColumn<?, ?> nameColumn;  
  
 @FXML  
 private TableColumn<?, ?> rateColumn;  
  
 @FXML  
 private TableColumn<?, ?> termColumn;  
 private ObservableList<KindCreditTableItem> creditsObservable;  
  
 @FXML  
 void addKindCredit(ActionEvent event) {  
 try {  
 FXMLLoader loader = new FXMLLoader(BankApp.class.getResource("add-kind-credit-dialog.fxml"));  
 Stage dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(BankApp.*primaryStage*);  
 dialogStage.setMinWidth(400);  
 dialogStage.setScene(new Scene(loader.load()));  
 dialogStage.setTitle("Добавить кредит");  
 AddKindCreditDialog controller = loader.getController();  
 controller.setDialogStage(dialogStage);  
 dialogStage.showAndWait();  
 updateList();  
 } catch (IOException e) {  
 System.*out*.println("Ошибка открытия окна: " + e.getMessage());  
 }  
 }  
  
 @FXML  
 void deleteKindCredit(ActionEvent event) {  
 KindCreditTableItem currentItem = kindCreditsTable.getSelectionModel().getSelectedItem();  
 int currentItemId = kindCreditsTable.getSelectionModel().getSelectedIndex();  
 if (currentItemId != -1) {  
 Alert alert = new Alert(Alert.AlertType.*CONFIRMATION*);  
 alert.setTitle("Подтверждение удаления");  
 alert.setHeaderText("Удаление записи");  
 alert.setContentText("Вы действительно хотите удалить \"" + currentItem.getName() + "\"?");  
  
 Optional<ButtonType> result = alert.showAndWait();  
  
 if (result.isPresent() && result.get() == ButtonType.*OK*) {  
 new KindCreditService().delete(currentItem.getKindCredit());  
 kindCreditsTable.getItems().remove(currentItemId);  
 }  
 }  
 else {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Предупреждение");  
 alert.setContentText("Выберите запись в таблице для удаления");  
 alert.showAndWait();  
 }  
 }  
  
 @FXML  
 void editKindCredit(ActionEvent event) {  
 KindCreditTableItem currentItem = kindCreditsTable.getSelectionModel().getSelectedItem();  
 int currentItemId = kindCreditsTable.getSelectionModel().getSelectedIndex();  
 if (currentItemId != -1) {  
 try {  
 FXMLLoader loader = new FXMLLoader(BankApp.class.getResource("edit-kind-credit-dialog.fxml"));  
 Stage dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(BankApp.*primaryStage*);  
 dialogStage.setMinWidth(400);  
 dialogStage.setScene(new Scene(loader.load()));  
 dialogStage.setTitle("Редактировать кредит");  
 EditKindCreditDialog controller = loader.getController();  
 controller.setDialogStage(dialogStage, currentItem.getKindCredit());  
 dialogStage.showAndWait();  
 updateList();  
 } catch (IOException e) {  
 System.*out*.println("Ошибка открытия окна: " + e.getMessage());  
 }  
 } else {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Предупреждение");  
 alert.setContentText("Выберите запись в таблице для редактирования");  
 alert.showAndWait();  
 }  
 }  
  
 @FXML  
 void onClientsButtonClick(ActionEvent event) {  
 BankApp.*primaryStage*.setScene(BankApp.*clients*);  
 }  
  
 @FXML  
 void onCreditButtonClick(ActionEvent event) {  
 BankApp.*primaryStage*.setScene(BankApp.*credits*);  
 }  
  
 @FXML  
 void powerOff(ActionEvent event) {  
 BankApp.*primaryStage*.close();  
 }  
  
 @FXML  
 void updateKindCredits(ActionEvent event) {  
 updateList();  
 }  
  
 public void updateList() {  
 kindCredits = new KindCreditService().findAll();  
 creditsObservable = FXCollections.*observableArrayList*();  
  
 for (KindCredit kindCredit : kindCredits) {  
 creditsObservable.add(new KindCreditTableItem(kindCredit));  
 }  
 kindCreditsTable.setItems(creditsObservable);  
 }  
  
 public void initialize() {  
 nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));  
 conditionColumn.setCellValueFactory(new PropertyValueFactory<>("condition"));  
 rateColumn.setCellValueFactory(new PropertyValueFactory<>("rate"));  
 termColumn.setCellValueFactory(new PropertyValueFactory<>("term"));  
  
 updateList();  
 }  
  
}

package com.example.kp.controller.kindcredit;  
  
import com.example.kp.BankApp;  
import com.example.kp.controller.client.AddClientDialog;  
import com.example.kp.controller.client.ClientTableItem;  
import com.example.kp.controller.client.EditClientDialog;  
import com.example.kp.model.KindCredit;  
import com.example.kp.service.KindCreditService;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.util.List;  
import java.util.Optional;  
  
public class KindCreditController {  
  
 private List<KindCredit> kindCredits;  
  
 @FXML  
 private Button clientsButton;  
  
 @FXML  
 private TableColumn<?, ?> conditionColumn;  
  
 @FXML  
 private Button creditsButton;  
  
 @FXML  
 private TableView<KindCreditTableItem> kindCreditsTable;  
  
 @FXML  
 private Button kindKreditButton;  
  
 @FXML  
 private TableColumn<?, ?> nameColumn;  
  
 @FXML  
 private TableColumn<?, ?> rateColumn;  
  
 @FXML  
 private TableColumn<?, ?> termColumn;  
 private ObservableList<KindCreditTableItem> creditsObservable;  
  
 @FXML  
 void addKindCredit(ActionEvent event) {  
 try {  
 FXMLLoader loader = new FXMLLoader(BankApp.class.getResource("add-kind-credit-dialog.fxml"));  
 Stage dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(BankApp.*primaryStage*);  
 dialogStage.setMinWidth(400);  
 dialogStage.setScene(new Scene(loader.load()));  
 dialogStage.setTitle("Добавить кредит");  
 AddKindCreditDialog controller = loader.getController();  
 controller.setDialogStage(dialogStage);  
 dialogStage.showAndWait();  
 updateList();  
 } catch (IOException e) {  
 System.*out*.println("Ошибка открытия окна: " + e.getMessage());  
 }  
 }  
  
 @FXML  
 void deleteKindCredit(ActionEvent event) {  
 KindCreditTableItem currentItem = kindCreditsTable.getSelectionModel().getSelectedItem();  
 int currentItemId = kindCreditsTable.getSelectionModel().getSelectedIndex();  
 if (currentItemId != -1) {  
 Alert alert = new Alert(Alert.AlertType.*CONFIRMATION*);  
 alert.setTitle("Подтверждение удаления");  
 alert.setHeaderText("Удаление записи");  
 alert.setContentText("Вы действительно хотите удалить \"" + currentItem.getName() + "\"?");  
  
 Optional<ButtonType> result = alert.showAndWait();  
  
 if (result.isPresent() && result.get() == ButtonType.*OK*) {  
 new KindCreditService().delete(currentItem.getKindCredit());  
 kindCreditsTable.getItems().remove(currentItemId);  
 }  
 }  
 else {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Предупреждение");  
 alert.setContentText("Выберите запись в таблице для удаления");  
 alert.showAndWait();  
 }  
 }  
  
 @FXML  
 void editKindCredit(ActionEvent event) {  
 KindCreditTableItem currentItem = kindCreditsTable.getSelectionModel().getSelectedItem();  
 int currentItemId = kindCreditsTable.getSelectionModel().getSelectedIndex();  
 if (currentItemId != -1) {  
 try {  
 FXMLLoader loader = new FXMLLoader(BankApp.class.getResource("edit-kind-credit-dialog.fxml"));  
 Stage dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(BankApp.*primaryStage*);  
 dialogStage.setMinWidth(400);  
 dialogStage.setScene(new Scene(loader.load()));  
 dialogStage.setTitle("Редактировать кредит");  
 EditKindCreditDialog controller = loader.getController();  
 controller.setDialogStage(dialogStage, currentItem.getKindCredit());  
 dialogStage.showAndWait();  
 updateList();  
 } catch (IOException e) {  
 System.*out*.println("Ошибка открытия окна: " + e.getMessage());  
 }  
 } else {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Предупреждение");  
 alert.setContentText("Выберите запись в таблице для редактирования");  
 alert.showAndWait();  
 }  
 }  
  
 @FXML  
 void onClientsButtonClick(ActionEvent event) {  
 BankApp.*primaryStage*.setScene(BankApp.*clients*);  
 }  
  
 @FXML  
 void onCreditButtonClick(ActionEvent event) {  
 BankApp.*primaryStage*.setScene(BankApp.*credits*);  
 }  
  
 @FXML  
 void powerOff(ActionEvent event) {  
 BankApp.*primaryStage*.close();  
 }  
  
 @FXML  
 void updateKindCredits(ActionEvent event) {  
 updateList();  
 }  
  
 public void updateList() {  
 kindCredits = new KindCreditService().findAll();  
 creditsObservable = FXCollections.*observableArrayList*();  
  
 for (KindCredit kindCredit : kindCredits) {  
 creditsObservable.add(new KindCreditTableItem(kindCredit));  
 }  
 kindCreditsTable.setItems(creditsObservable);  
 }  
  
 public void initialize() {  
 nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));  
 conditionColumn.setCellValueFactory(new PropertyValueFactory<>("condition"));  
 rateColumn.setCellValueFactory(new PropertyValueFactory<>("rate"));  
 termColumn.setCellValueFactory(new PropertyValueFactory<>("term"));  
  
 updateList();  
 }  
  
}

**Форма приложения credit-view**

package com.example.kp.controller.credit;  
  
import com.example.kp.BankApp;  
import com.example.kp.controller.client.AddClientDialog;  
import com.example.kp.controller.client.ClientTableItem;  
import com.example.kp.controller.client.EditClientDialog;  
import com.example.kp.controller.credit.EditCreditDialog;  
import com.example.kp.model.Client;  
import com.example.kp.model.Credit;  
import com.example.kp.service.CreditService;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.util.List;  
import java.util.Optional;  
  
public class CreditController {  
  
 private List<Credit> credits;  
 private ObservableList<CreditTableItem> creditsObservable;  
  
 @FXML  
 private TableColumn<?, ?> addressColumn;  
  
 @FXML  
 private Button clientsButton;  
  
 @FXML  
 private Button creditsButton;  
  
 @FXML  
 private TableView<CreditTableItem> creditsTable;  
  
 @FXML  
 private TableColumn<?, ?> dateColumn;  
  
 @FXML  
 private TableColumn<?, ?> emailColumn;  
  
 @FXML  
 private Button kindKreditButton;  
  
 @FXML  
 private TableColumn<?, ?> kindKreditColumn;  
  
 @FXML  
 private TableColumn<Client, String> nameColumn;  
  
 @FXML  
 private TableColumn<?, ?> phoneColumn;  
  
 @FXML  
 private TableColumn<?, ?> summaColumn;  
  
 @FXML  
 void addCredit(ActionEvent event) {  
 try {  
 FXMLLoader loader = new FXMLLoader(BankApp.class.getResource("add-credit-dialog.fxml"));  
 Stage dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(BankApp.*primaryStage*);  
 dialogStage.setMinWidth(400);  
 dialogStage.setScene(new Scene(loader.load()));  
 dialogStage.setTitle("Добавить кредит");  
 AddCreditDialog controller = loader.getController();  
 controller.setDialogStage(dialogStage);  
 dialogStage.showAndWait();  
 updateList();  
 } catch (IOException e) {  
 System.*out*.println("Ошибка открытия окна: " + e.getMessage());  
 }  
 }  
  
 @FXML  
 void deleteCredit(ActionEvent event) {  
 CreditTableItem currentItem = creditsTable.getSelectionModel().getSelectedItem();  
 int currentItemId = creditsTable.getSelectionModel().getSelectedIndex();  
 if (currentItemId != -1) {  
 Alert alert = new Alert(Alert.AlertType.*CONFIRMATION*);  
 alert.setTitle("Подтверждение удаления");  
 alert.setHeaderText("Удаление записи");  
 alert.setContentText("Вы действительно хотите удалить \"" + currentItem.getName() + "\"?");  
  
 Optional<ButtonType> result = alert.showAndWait();  
  
 if (result.isPresent() && result.get() == ButtonType.*OK*) {  
 new CreditService().delete(currentItem.getCredit());  
 creditsTable.getItems().remove(currentItemId);  
 }  
 }  
 else {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Предупреждение");  
 alert.setContentText("Выберите запись в таблице для удаления");  
 alert.showAndWait();  
 }  
 }  
  
 @FXML  
 void editCredit(ActionEvent event) {  
 CreditTableItem currentItem = creditsTable.getSelectionModel().getSelectedItem();  
 int currentItemId = creditsTable.getSelectionModel().getSelectedIndex();  
 if (currentItemId != -1) {  
 try {  
 FXMLLoader loader = new FXMLLoader(BankApp.class.getResource("edit-credit-dialog.fxml"));  
 Stage dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(BankApp.*primaryStage*);  
 dialogStage.setMinWidth(400);  
 dialogStage.setScene(new Scene(loader.load()));  
 dialogStage.setTitle("Редактировать кредит");  
 EditCreditDialog controller = loader.getController();  
 controller.setDialogStage(dialogStage, currentItem.getCredit());  
 dialogStage.showAndWait();  
 updateList();  
 } catch (IOException e) {  
 System.*out*.println("Ошибка открытия окна: " + e.getMessage());  
 }  
 } else {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Предупреждение");  
 alert.setContentText("Выберите запись в таблице для редактирования");  
 alert.showAndWait();  
 }  
 }  
  
 @FXML  
 void onClientsButtonClick(ActionEvent event) throws IOException {  
 BankApp.*primaryStage*.setScene(BankApp.*clients*);  
 }  
  
 @FXML  
 void onKindKreditClick(ActionEvent event) throws IOException {  
 BankApp.*primaryStage*.setScene(BankApp.*kindCredits*);  
 }  
  
 @FXML  
 void powerOff(ActionEvent event) {  
 BankApp.*primaryStage*.close();  
 }  
  
 @FXML  
 void updateCredits(ActionEvent event) {  
 updateList();  
 }  
  
  
 public void updateList() {  
 credits = new CreditService().findAll();  
 creditsObservable = FXCollections.*observableArrayList*();  
  
 for (Credit credit : credits) {  
 creditsObservable.add(new CreditTableItem(credit));  
 }  
 creditsTable.setItems(creditsObservable);  
 }  
  
 public void initialize() {  
 phoneColumn.setCellValueFactory(new PropertyValueFactory<>("phone"));  
 addressColumn.setCellValueFactory(new PropertyValueFactory<>("address"));  
 emailColumn.setCellValueFactory(new PropertyValueFactory<>("email"));  
 kindKreditColumn.setCellValueFactory(new PropertyValueFactory<>("kindCredit"));  
 nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));  
 summaColumn.setCellValueFactory(new PropertyValueFactory<>("summa"));  
 dateColumn.setCellValueFactory(new PropertyValueFactory<>("date"));  
  
 updateList();  
 }  
}

**Форма приложения add-client-dialog**

package com.example.kp.controller.client;  
  
import com.example.kp.model.Client;  
import com.example.kp.service.ClientService;  
import javafx.fxml.FXML;  
import javafx.fxml.Initializable;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
  
import java.net.URL;  
import java.util.ResourceBundle;  
  
public class AddClientDialog implements Initializable {  
  
 @FXML  
 private TextField addressField;  
  
 @FXML  
 private TextField emailField;  
  
 @FXML  
 private Label errorLabel;  
  
 @FXML  
 private TextField kindPropertyField;  
  
 @FXML  
 private TextField nameField;  
  
 @FXML  
 private Button okButton;  
  
 @FXML  
 private TextField phoneField;  
  
 private Stage dialogStage;  
  
 @Override  
 public void initialize(URL url, ResourceBundle resourceBundle) {  
 }  
  
 @FXML  
 private void handleOk() {  
 try {  
 Client client = new Client();  
 client.setName(nameField.getText());  
 client.setPhone(phoneField.getText());  
 client.setAddress(addressField.getText());  
 client.setContact(emailField.getText());  
 client.setKindProperty(kindPropertyField.getText());  
 ClientTableItem clientTableItem = new ClientTableItem(client);  
  
 new ClientService().save(client);  
  
 dialogStage.close();  
 }catch (IllegalArgumentException e){  
 errorLabel.setText(e.getMessage());  
 }  
 }  
  
 public void setDialogStage(Stage dialogStage) {  
 this.dialogStage = dialogStage;  
 }  
}

**Форма приложения add-kind-credit-dialog**

package com.example.kp.controller.kindcredit;  
  
import com.example.kp.controller.client.ClientTableItem;  
import com.example.kp.model.Client;  
import com.example.kp.model.KindCredit;  
import com.example.kp.service.ClientService;  
import com.example.kp.service.KindCreditService;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
  
import java.math.BigDecimal;  
  
public class AddKindCreditDialog {  
  
 @FXML  
 private TextField conditionField;  
  
 @FXML  
 private Label errorLabel;  
  
 @FXML  
 private TextField nameField;  
  
 @FXML  
 private Button okButton;  
  
 @FXML  
 private TextField rateField;  
  
 @FXML  
 private TextField termField;  
 private Stage dialogStage;  
  
 @FXML  
 void handleOk(ActionEvent event) {  
 try {  
 KindCredit kindCredit = new KindCredit();  
 kindCredit.setName(nameField.getText());  
 kindCredit.setConditions(conditionField.getText());  
 kindCredit.setRate(rateField.getText());  
 kindCredit.setTerm(termField.getText());  
 KindCreditTableItem kindCreditTableItem = new KindCreditTableItem(kindCredit);  
  
 new KindCreditService().save(kindCredit);  
  
 dialogStage.close();  
 }catch (IllegalArgumentException e){  
 errorLabel.setText(e.getMessage());  
 }  
 }  
  
 public void setDialogStage(Stage dialogStage) {  
 this.dialogStage = dialogStage;  
 }  
}

**Форма приложения add-credit-dialog**

package com.example.kp.controller.credit;  
  
import com.example.kp.controller.client.ClientTableItem;  
import com.example.kp.model.Client;  
import com.example.kp.model.Credit;  
import com.example.kp.model.KindCredit;  
import com.example.kp.service.ClientService;  
import com.example.kp.service.CreditService;  
import com.example.kp.service.KindCreditService;  
import javafx.collections.FXCollections;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.DatePicker;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
  
import java.time.LocalDate;  
import java.time.ZoneId;  
import java.util.Date;  
import java.util.List;  
  
public class AddCreditDialog {  
  
 @FXML  
 private DatePicker dateField;  
  
 @FXML  
 private Label errorLabel;  
  
 @FXML  
 private ComboBox<KindCredit> kindCreditField;  
  
 @FXML  
 private Button okButton;  
  
 @FXML  
 private TextField sumField;  
  
 @FXML  
 private ComboBox<Client> userField;  
 private Stage dialogStage;  
  
 @FXML  
 void handleOk(ActionEvent event) {  
 try {  
 if (userField.getSelectionModel().getSelectedIndex() == -1){  
 throw new IllegalArgumentException("Нужно заполнить поле \"Клиент\"");  
 }  
 if (kindCreditField.getSelectionModel().getSelectedIndex() == -1){  
 throw new IllegalArgumentException("Нужно заполнить поле \"Вид кредита\"");  
 }  
 Credit credit = new Credit();  
 credit.setClient(userField.getSelectionModel().getSelectedItem());  
 credit.setKindCredit(kindCreditField.getSelectionModel().getSelectedItem());  
 credit.setSumma(sumField.getText());  
 credit.setDate((Date.*from*(dateField.getValue().atStartOfDay(ZoneId.*systemDefault*()).toInstant())));  
 CreditTableItem clientTableItem = new CreditTableItem(credit);  
  
 new CreditService().save(credit);  
  
 dialogStage.close();  
 }catch (IllegalArgumentException e){  
 errorLabel.setText(e.getMessage());  
 }  
 }  
  
 public void setDialogStage(Stage dialogStage) {  
 this.dialogStage = dialogStage;  
  
 List<Client> clients = new ClientService().findAll();  
 userField.getItems().addAll(FXCollections.*observableList*(clients));  
 List<KindCredit> kindCredits = new KindCreditService().findAll();  
 kindCreditField.getItems().addAll(FXCollections.*observableList*(kindCredits));  
 dateField.setValue(LocalDate.*now*());  
 }  
}

**Форма приложения edit-client-dialog**

package com.example.kp.controller.client;  
  
import com.example.kp.model.Client;  
import com.example.kp.service.ClientService;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
  
public class EditClientDialog {  
 Client client;  
  
 @FXML  
 private TextField addressField;  
  
 @FXML  
 private TextField emailField;  
  
 @FXML  
 private Label errorLabel;  
  
 @FXML  
 private TextField kindPropertyField;  
  
 @FXML  
 private TextField nameField;  
  
 @FXML  
 private Button okButton;  
  
 @FXML  
 private TextField phoneField;  
  
 private Stage dialogStage;  
  
 @FXML  
 void handleOk(ActionEvent event) {  
 try {  
 client.setName(nameField.getText());  
 client.setPhone(phoneField.getText());  
 client.setAddress(addressField.getText());  
 client.setContact(emailField.getText());  
 client.setKindProperty(kindPropertyField.getText());  
  
 new ClientService().update(client);  
  
 dialogStage.close();  
 }catch (IllegalArgumentException e){  
 errorLabel.setText(e.getMessage());  
 }  
 }  
  
 public void setDialogStage(Stage dialogStage, Client client) {  
 this.client = client;  
 this.dialogStage = dialogStage;  
  
 nameField.setText(client.getName());  
 kindPropertyField.setText(client.getKindProperty());  
 addressField.setText(client.getAddress());  
 phoneField.setText(client.getPhone());  
 emailField.setText(client.getContact());  
 }  
  
}

**Форма приложения edit-kind-credit-dialog**

package com.example.kp.controller.kindcredit;  
  
import com.example.kp.model.Client;  
import com.example.kp.model.KindCredit;  
import com.example.kp.service.KindCreditService;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
  
public class EditKindCreditDialog {  
  
 @FXML  
 private TextField conditionField;  
  
 @FXML  
 private Label errorLabel;  
  
 @FXML  
 private TextField nameField;  
  
 @FXML  
 private Button okButton;  
  
 @FXML  
 private TextField rateField;  
  
 @FXML  
 private TextField termField;  
  
 private KindCredit kindCredit;  
 private Stage dialogStage;  
  
 @FXML  
 void handleOk(ActionEvent event) {  
 try {  
 kindCredit.setName(nameField.getText());  
 kindCredit.setConditions(conditionField.getText());  
 kindCredit.setRate(rateField.getText());  
 kindCredit.setTerm(termField.getText());  
  
 new KindCreditService().update(kindCredit);  
  
 dialogStage.close();  
 }catch (IllegalArgumentException e){  
 errorLabel.setText(e.getMessage());  
 }  
 }  
 public void setDialogStage(Stage dialogStage, KindCredit kindCredit) {  
 this.kindCredit = kindCredit;  
 this.dialogStage = dialogStage;  
  
 nameField.setText(kindCredit.getName());  
 conditionField.setText(kindCredit.getConditions());  
 rateField.setText(String.*valueOf*(kindCredit.getRate()));  
 termField.setText(String.*valueOf*(kindCredit.getTerm()));  
 }  
}

**Форма приложения edit-credit-dialog**

package com.example.kp.controller.credit;  
  
import com.example.kp.model.Client;  
import com.example.kp.model.Credit;  
import com.example.kp.model.KindCredit;  
import com.example.kp.service.ClientService;  
import com.example.kp.service.CreditService;  
import com.example.kp.service.KindCreditService;  
import javafx.collections.FXCollections;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.stage.Stage;  
  
import java.time.Instant;  
import java.time.LocalDate;  
import java.time.ZoneId;  
import java.util.Date;  
import java.util.List;  
  
public class EditCreditDialog {  
  
 @FXML  
 private DatePicker dateField;  
  
 @FXML  
 private Label errorLabel;  
  
 @FXML  
 private ComboBox<KindCredit> kindCreditField;  
  
 @FXML  
 private Button okButton;  
  
 @FXML  
 private TextField sumField;  
  
 @FXML  
 private ComboBox<Client> userField;  
 private Credit credit;  
 private Stage dialogStage;  
  
 @FXML  
 void handleOk(ActionEvent event) {  
 try {  
 credit.setClient(userField.getSelectionModel().getSelectedItem());  
 credit.setKindCredit(kindCreditField.getSelectionModel().getSelectedItem());  
 credit.setSumma(sumField.getText());  
 credit.setDate((Date.*from*(dateField.getValue().atStartOfDay(ZoneId.*systemDefault*()).toInstant())));  
  
 new CreditService().update(credit);  
  
 dialogStage.close();  
 } catch (IllegalArgumentException e) {  
 errorLabel.setText(e.getMessage());  
 }  
 }  
  
 public void setDialogStage(Stage dialogStage, Credit credit) {  
 this.dialogStage = dialogStage;  
 this.credit = credit;  
  
 List<Client> clients = new ClientService().findAll();  
 userField.getItems().addAll(FXCollections.*observableList*(clients));  
 List<KindCredit> kindCredits = new KindCreditService().findAll();  
 kindCreditField.getItems().addAll(FXCollections.*observableList*(kindCredits));  
 dateField.setValue(LocalDate.*now*());  
  
 userField.setValue(credit.getClient());  
 kindCreditField.setValue(credit.getKindCredit());  
 sumField.setText(credit.getSumma().toString());  
 dateField.setValue(Instant.*ofEpochMilli*(credit.getDate().getTime())  
 .atZone(ZoneId.*systemDefault*())  
 .toLocalDate());  
 }  
}

**ПРИЛОЖЕНИЕ Б**

**Диск с исполняемым кодом программного модуля**